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

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

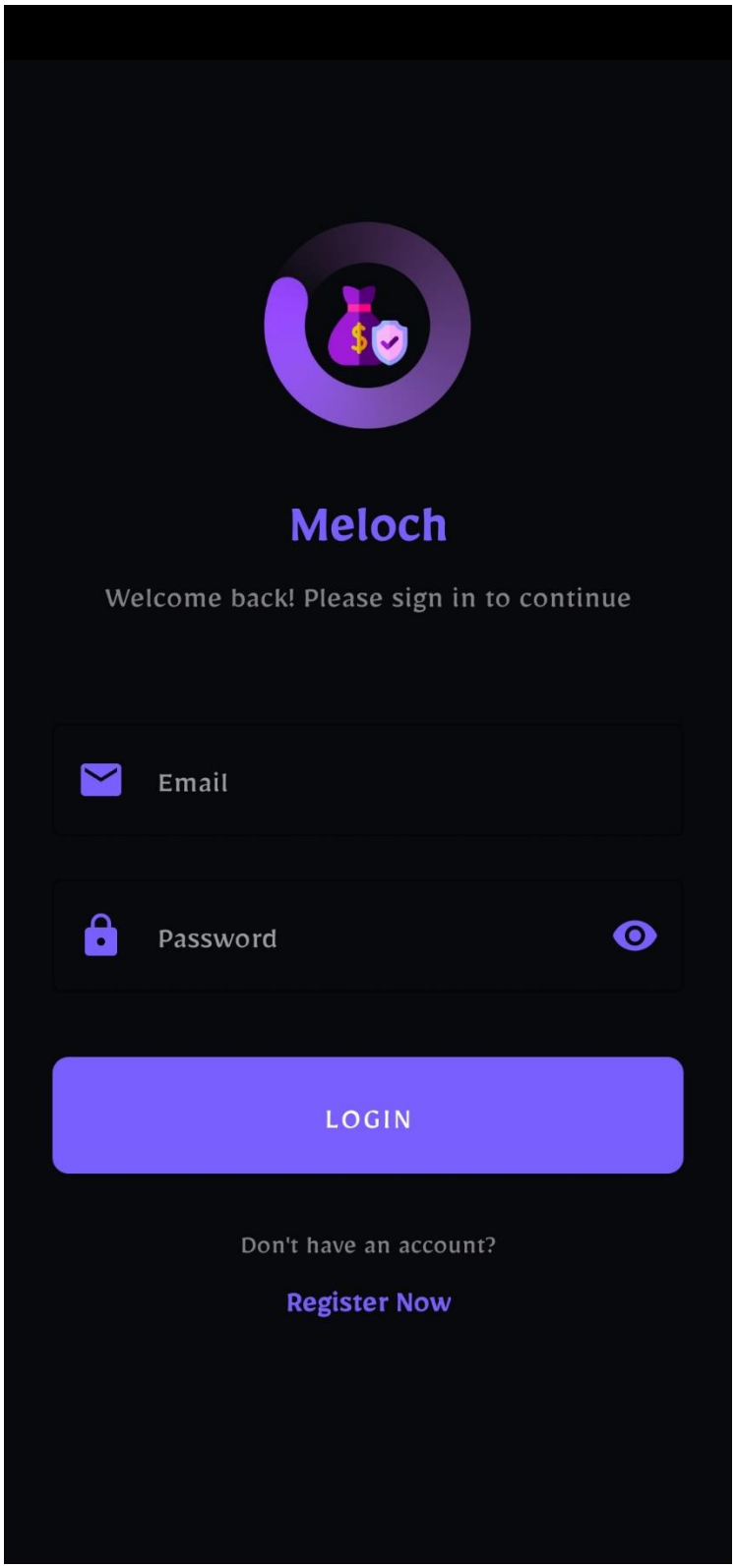
Smart Notifications

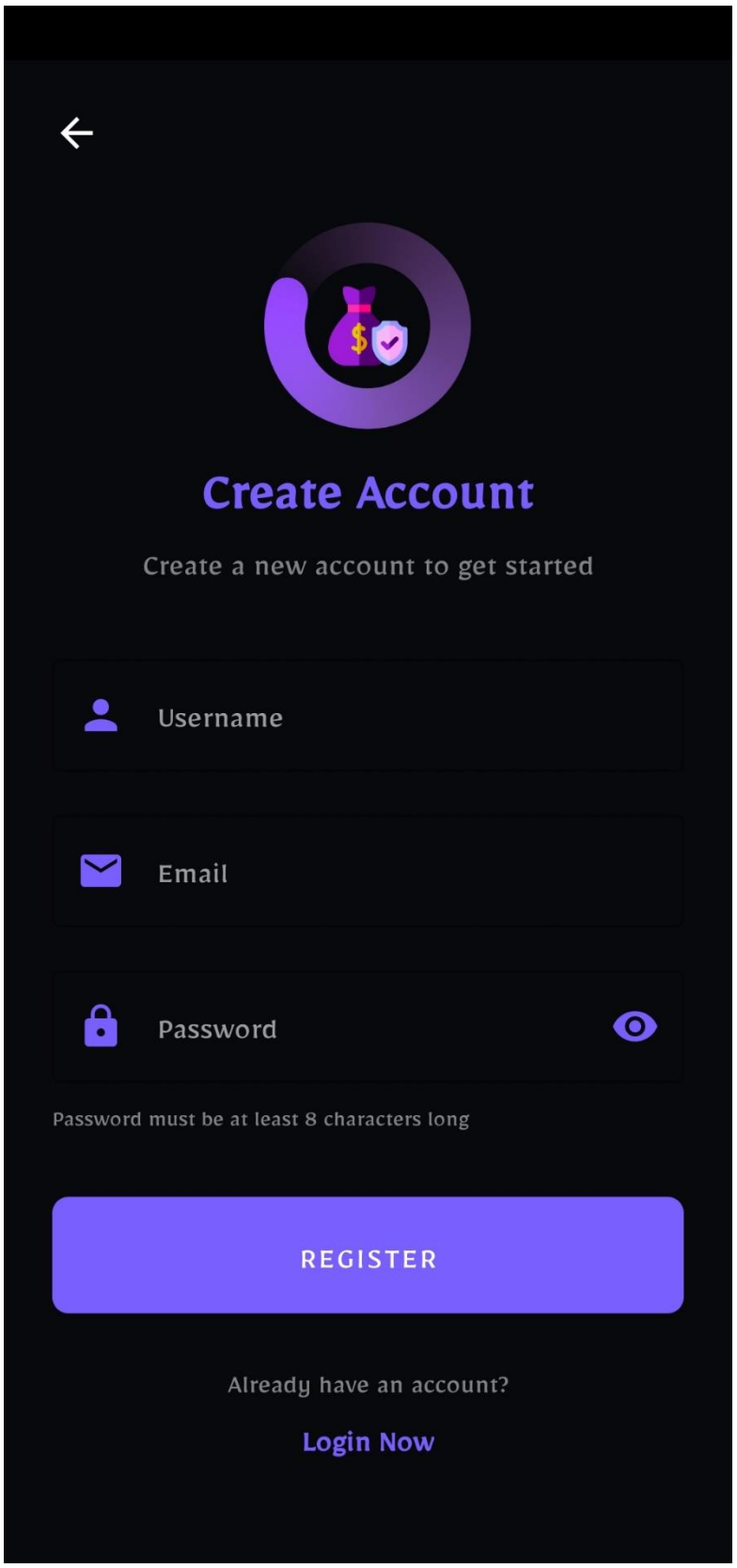
- The app alerts users when budgets are running low or fully used
- Notifications also appear for budget resets or important financial activity
- Designed to keep users financially aware and informed without being intrusive

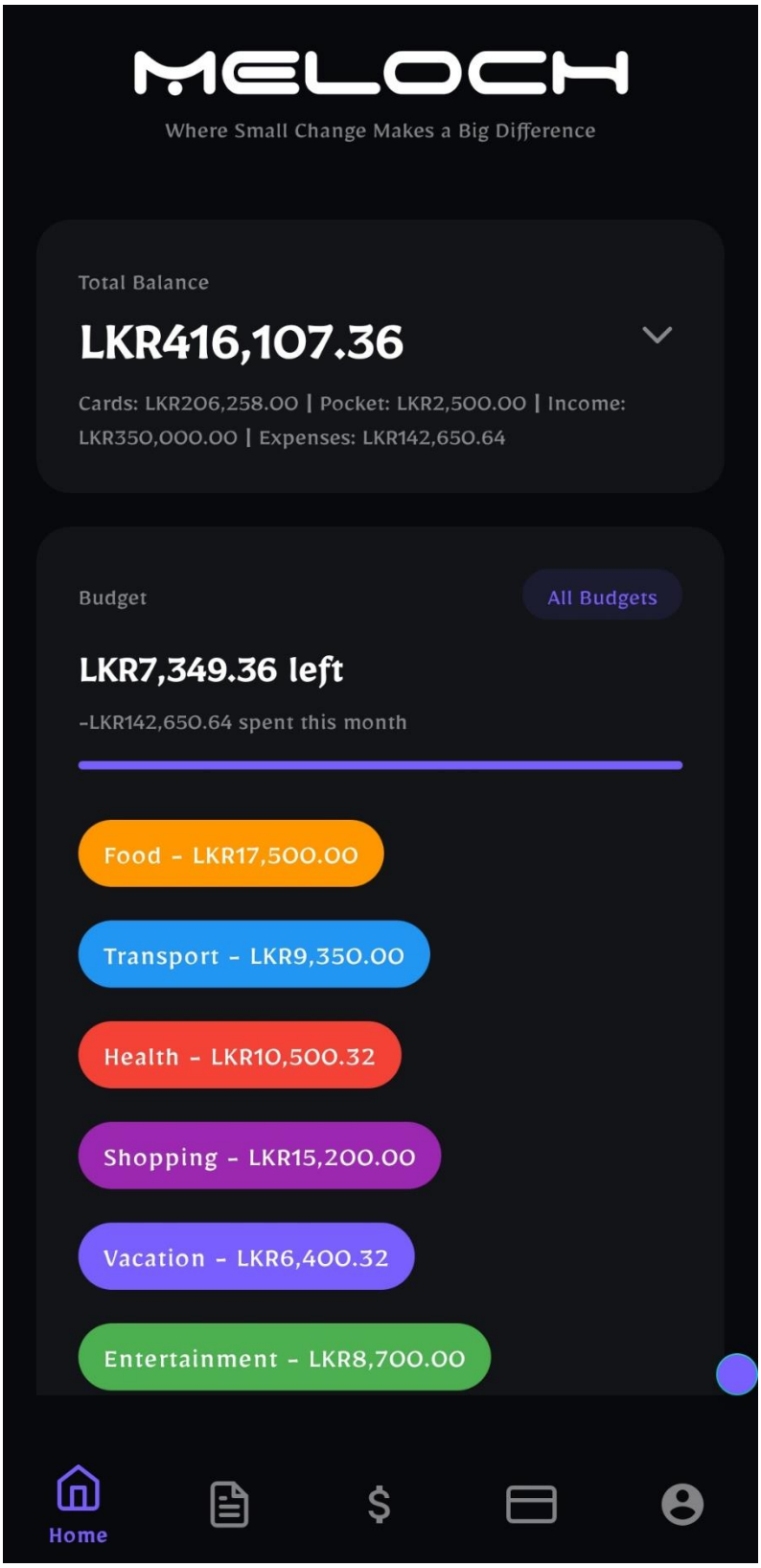
Conclusion

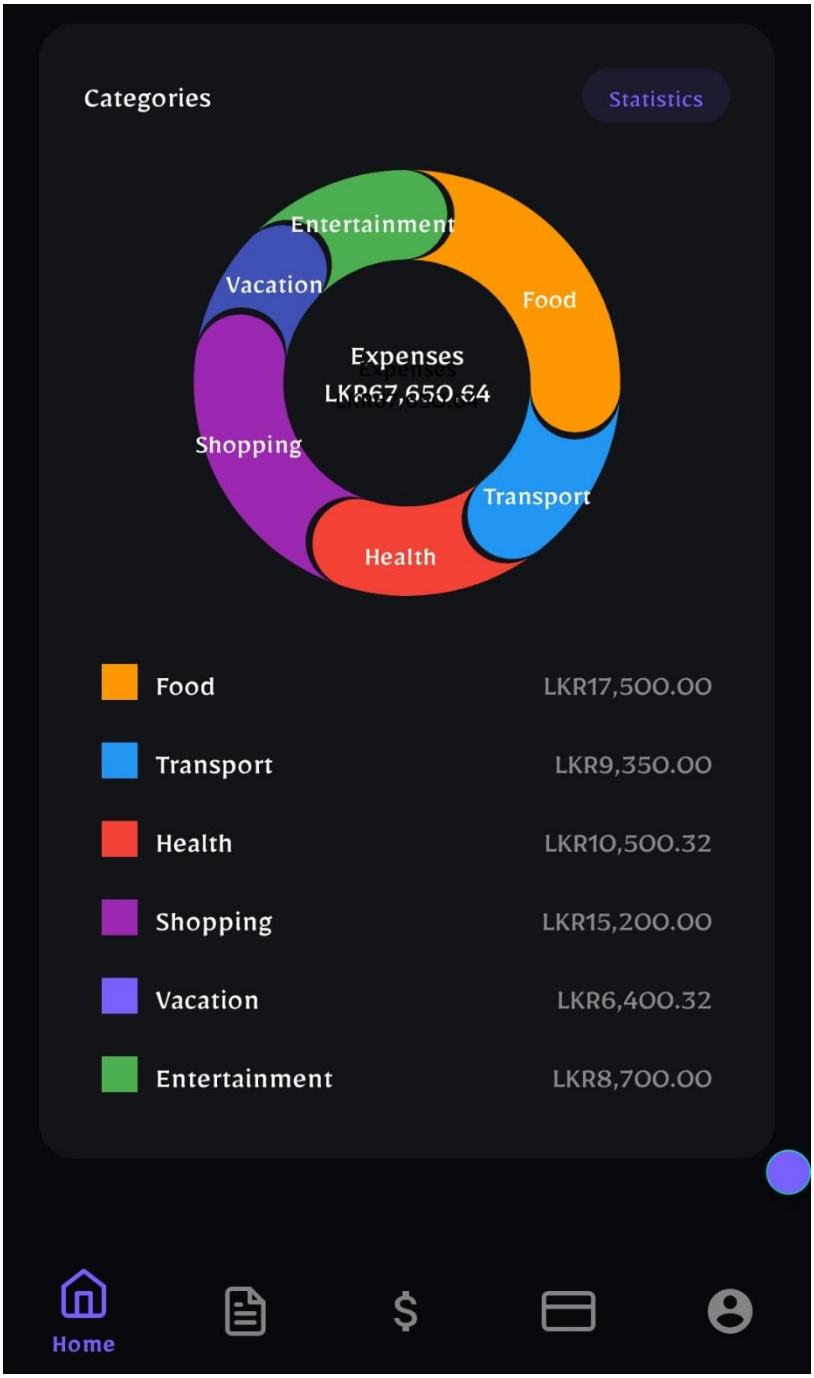
Meloch delivers an all-in-one financial management solution that combines everyday convenience with powerful financial insights. Through its thoughtful features—such as budget monitoring, transaction tracking, wallet card storage, and data export—the app supports users in building responsible financial 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.

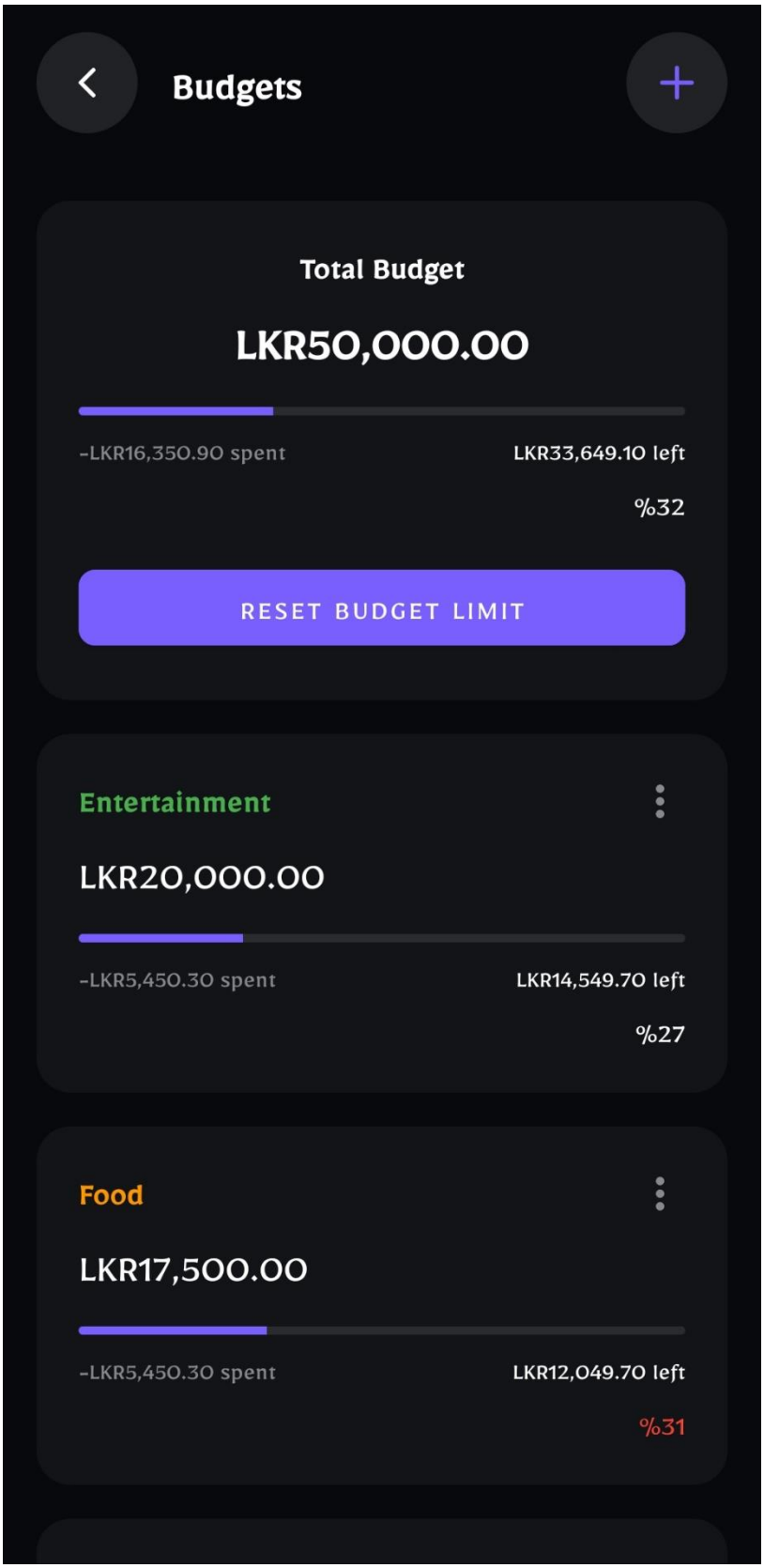
Graphical User Interfaces

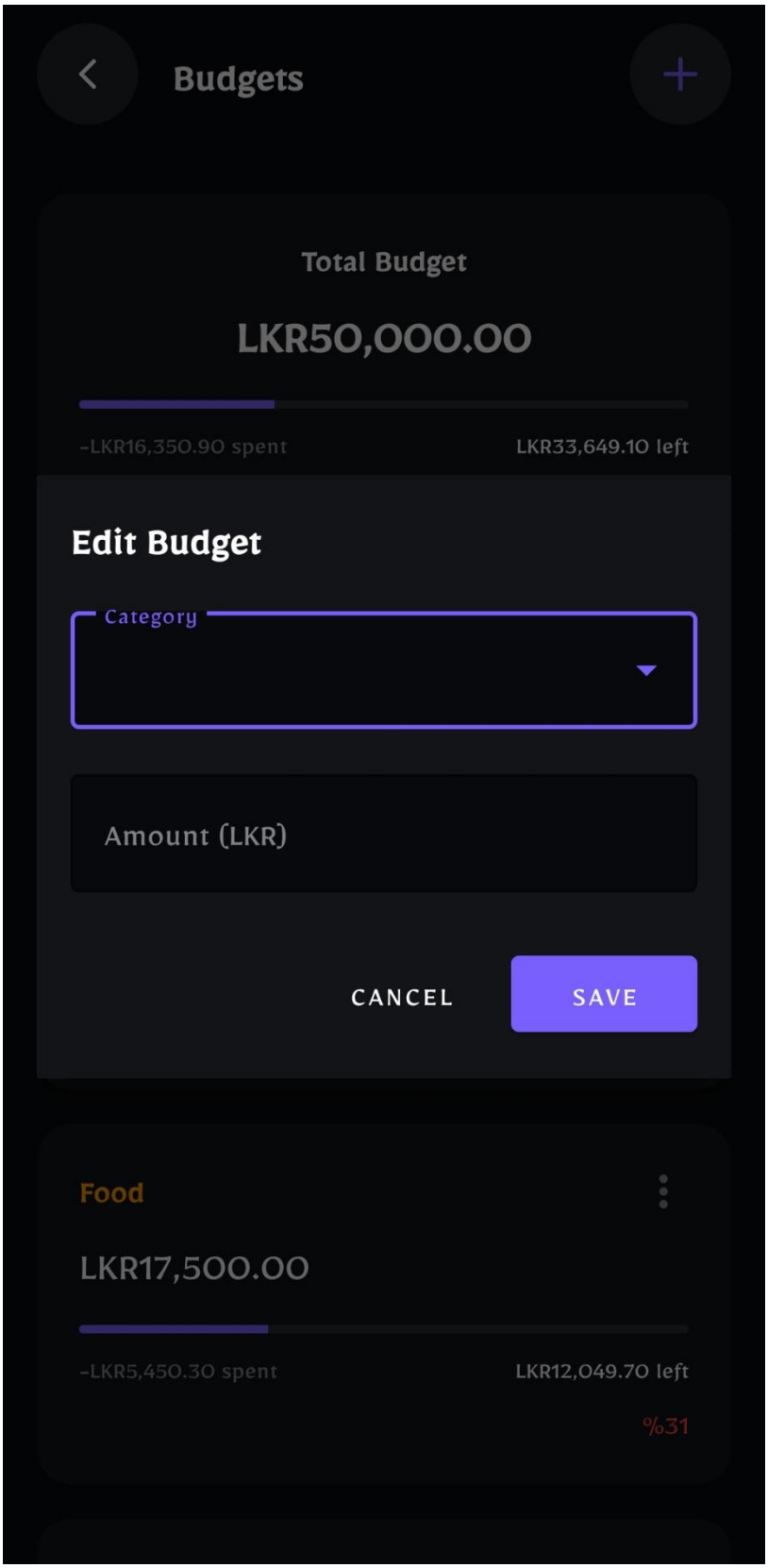

Meloch
Welcome back! Please sign in to continue
Email
Password
LOGIN
Don't have an account?
Register Now

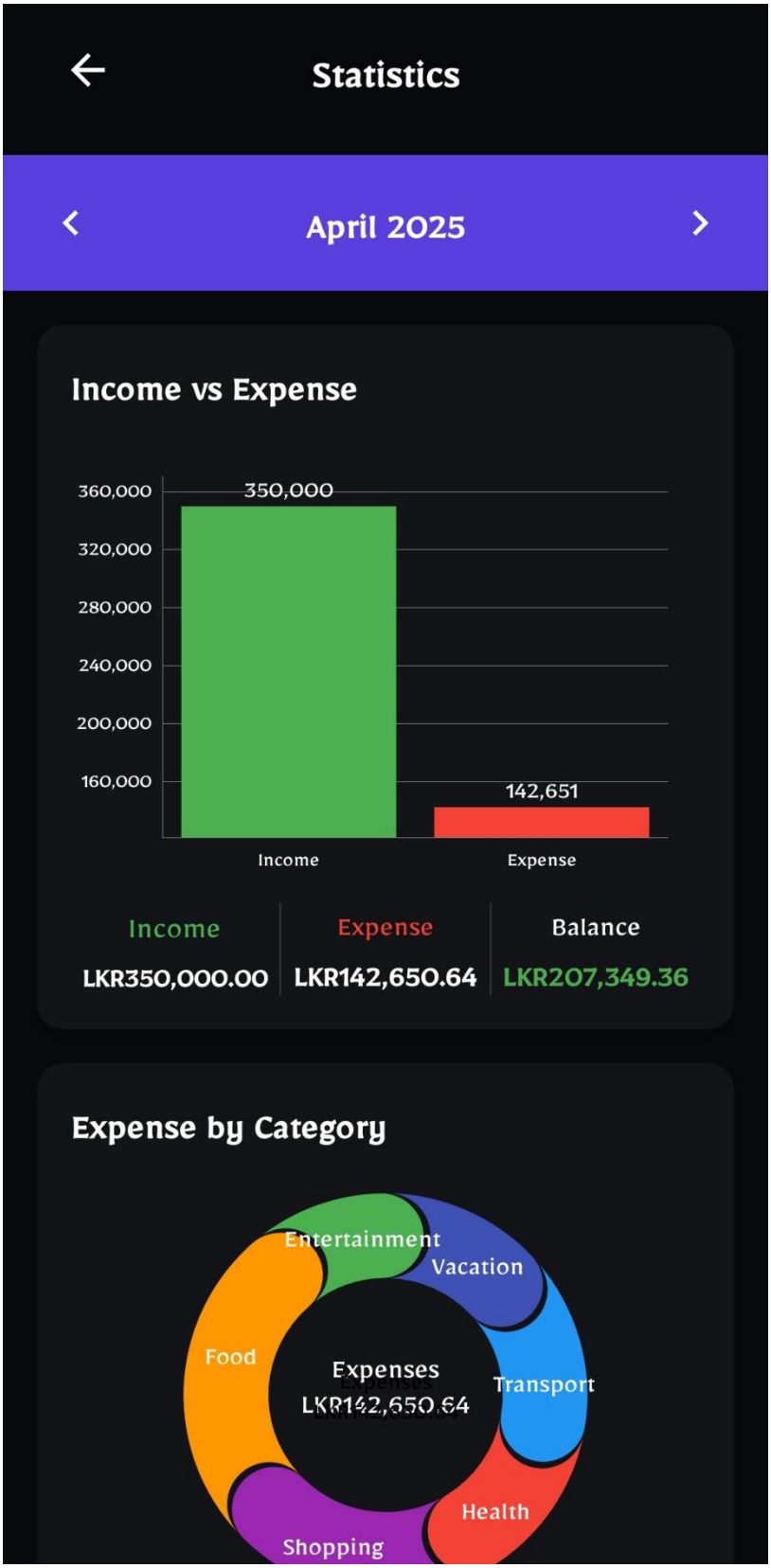

Create Account
Create a new account to get started
Username
Email
Password
Password must be at least 8 characters long
REGISTER
Already have an account?
Login Now

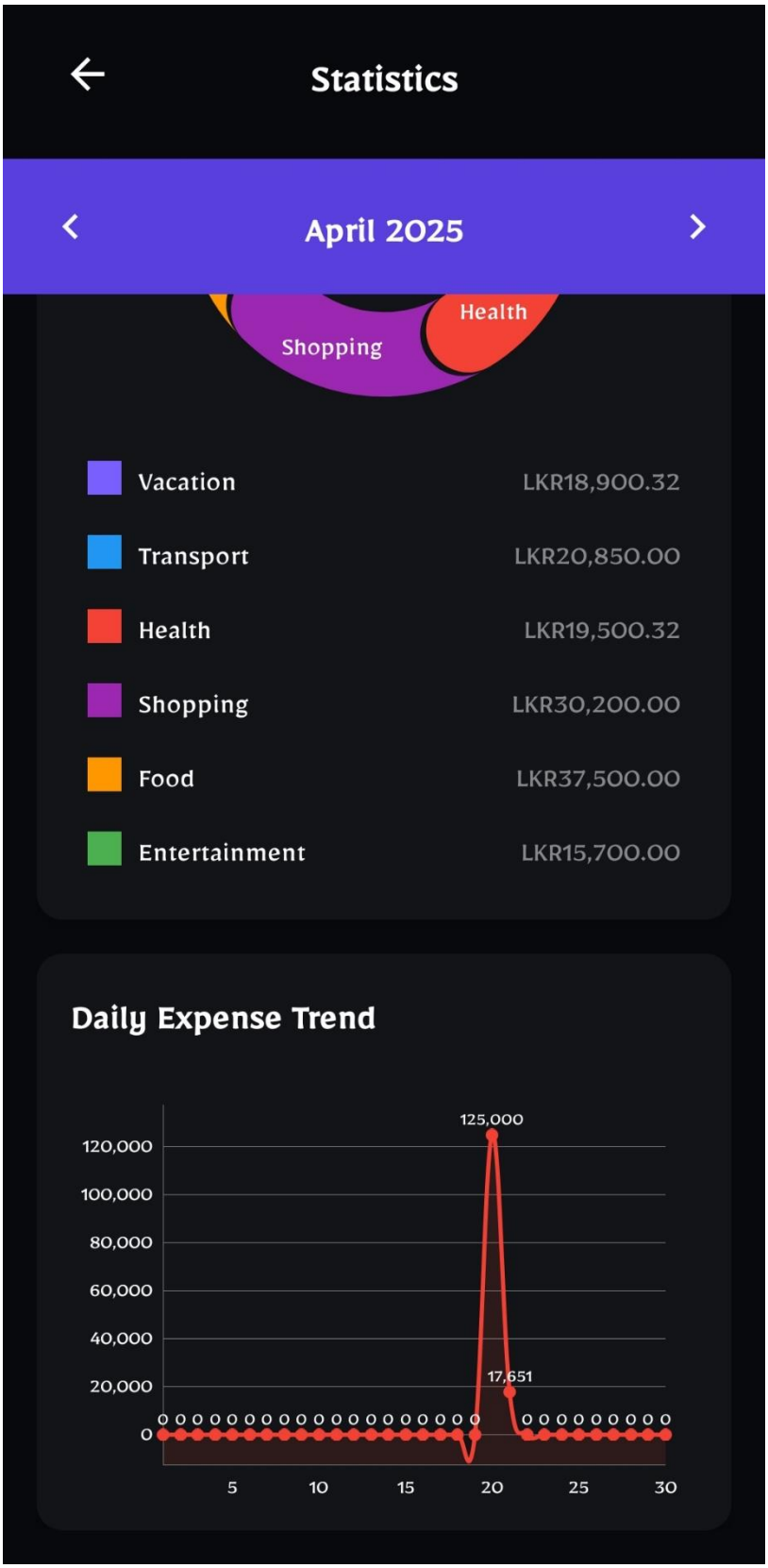

MELOCH
Where Small Change Makes a Big Difference
Total Balance
LKR416,107.36
Cards: LKR206,258.00 | Pocket: LKR2,500.00 | Income: LKR350,000.00 | Expenses: LKR142,650.64
Budget
All Budgets
LKR7,349.36 left
-LKR142,650.64 spent this month
Food - LKR17,500.00
Transport - LKR9,350.00
Health - LKR10,500.32
Shopping - LKR15,200.00
Vacation - LKR6,400.32
Entertainment - LKR8,700.00
Home

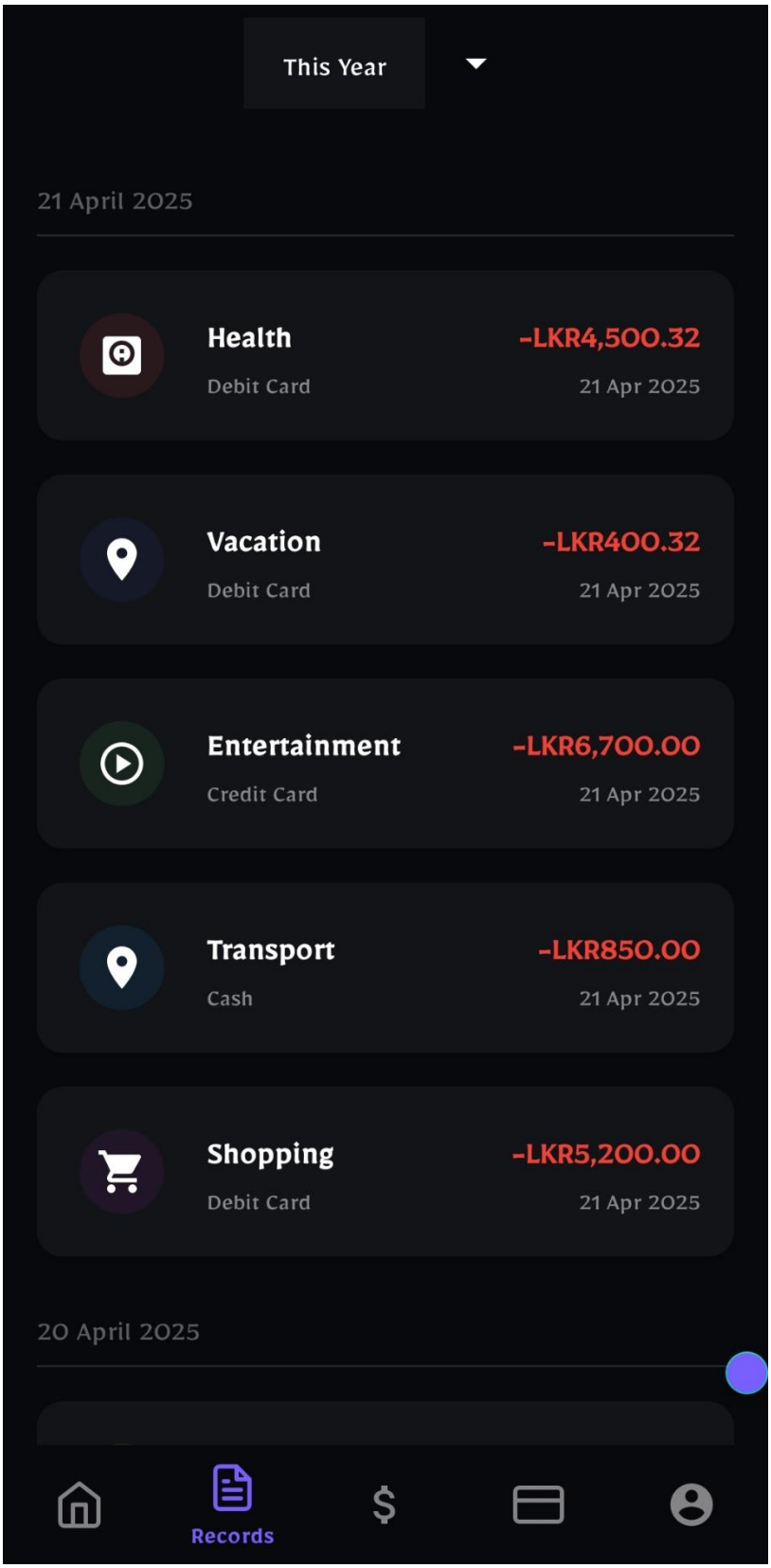

Categories
Statistics
Entertainment
Vacation
Food
Expenses
LKR67,650.64
Shopping
Transport
Health
Food
LKR17,500.00
Transport
LKR9,350.00
Health
LKR10,500.32
Shopping
LKR15,200.00
Vacation
LKR6,400.32
Entertainment
LKR8,700.00
Home

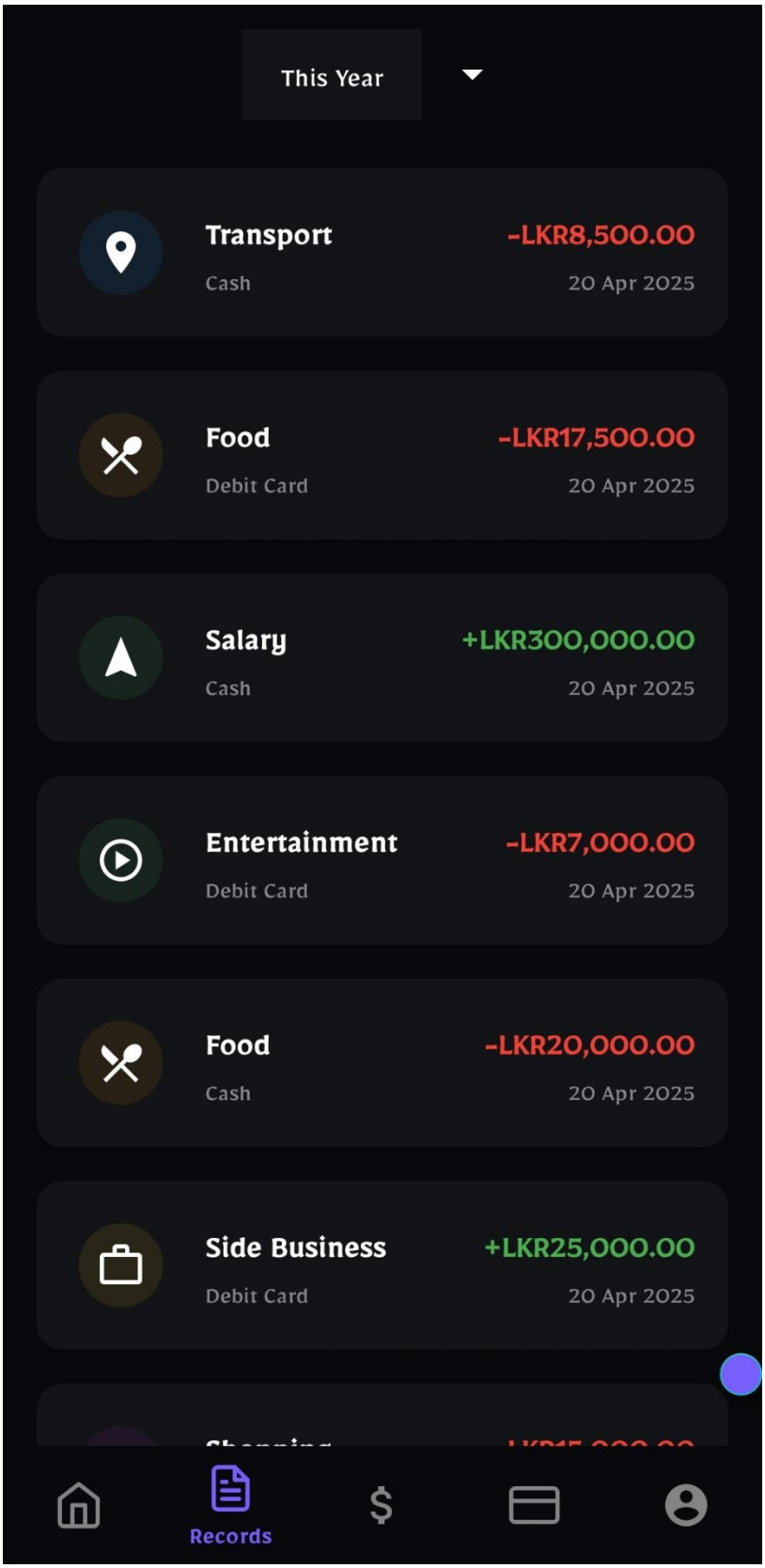

Budgets
Total Budget
LKR50,000.00
-LKR16,350.90 spent
LKR33,649.10 left
%32
RESET BUDGET LIMIT
Entertainment
LKR20,000.00
-LKR5,450.30 spent
LKR14,549.70 left
%27
Food
LKR17,500.00
-LKR5,450.30 spent
LKR12,049.70 left
%31

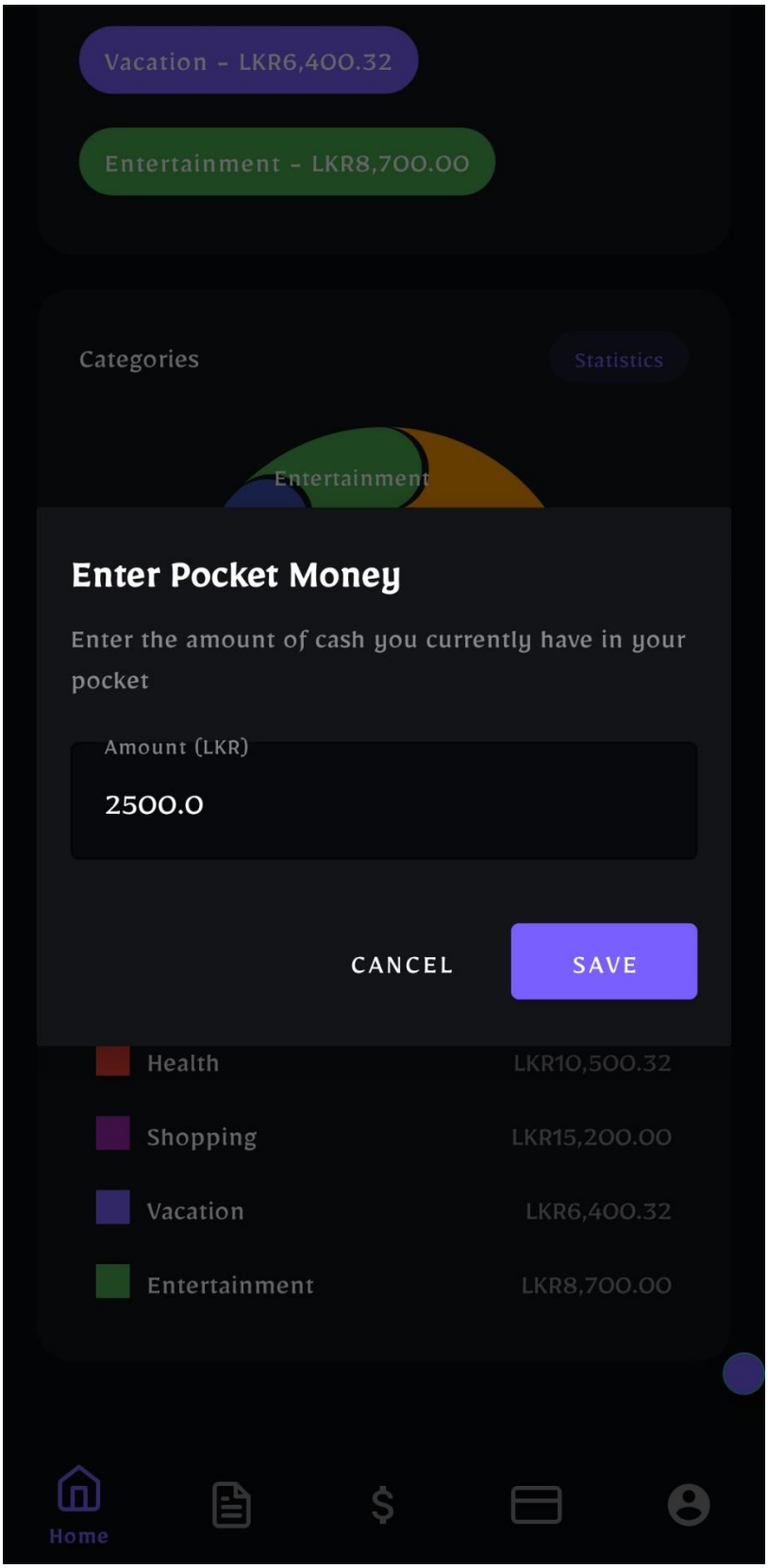

Budgets
Total Budget
LKR50,000.00
-LKR16,350.90 spent
LKR33,649.10 left
Edit Budget
Category
Amount (LKR)
CANCEL
SAVE
Food
LKR17,500.00
-LKR5,450.30 spent
LKR12,049.70 left
%31

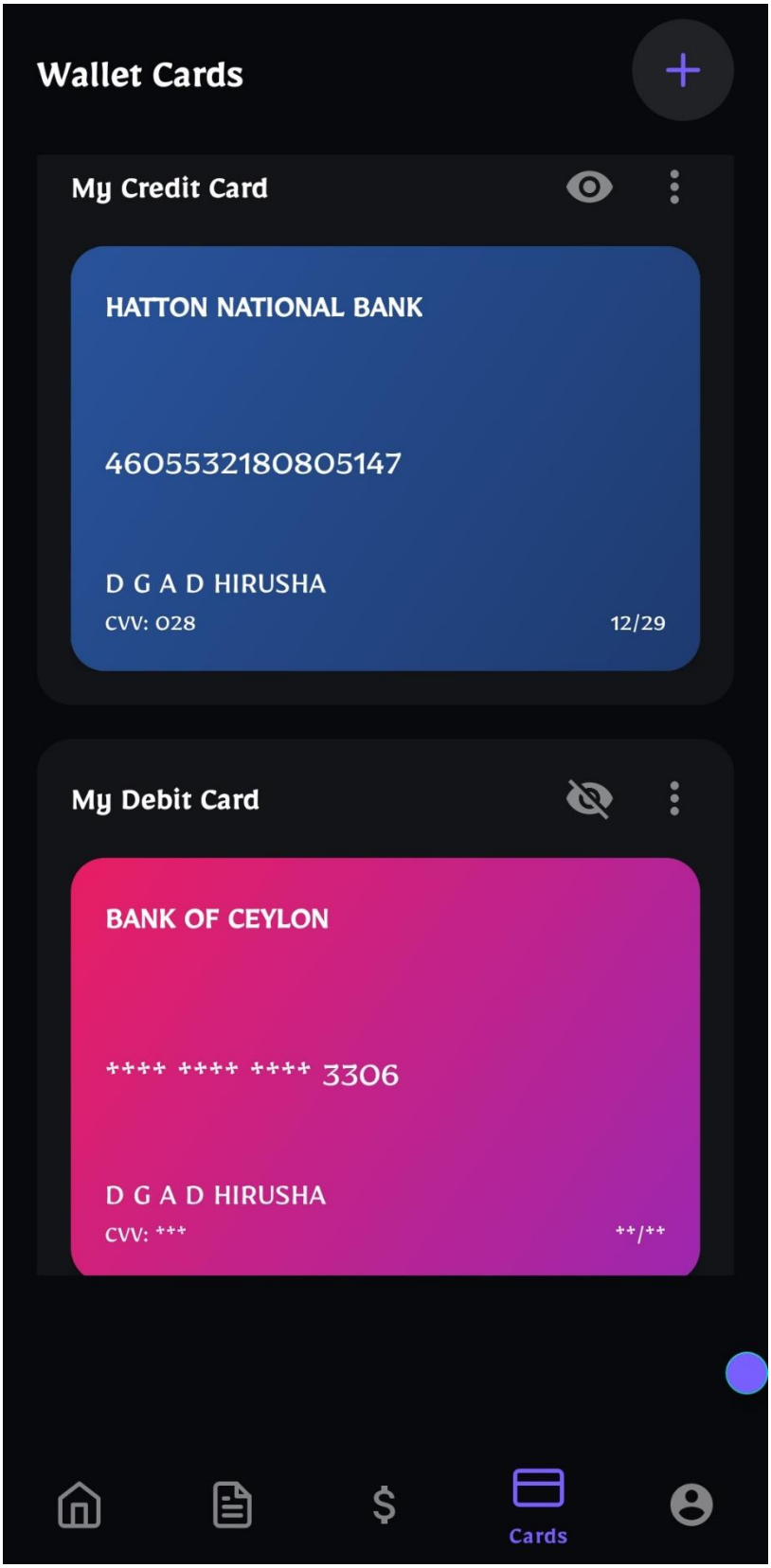

Statistics
April 2025
Income vs Expense
360,000
320,000
280,000
240,000
200,000
160,000
350,000
142,651
Income
Expense
Income
Expense
Balance
LKR350,000.00
LKR142,650.64
LKR207,349.36
Expense by Category
Entertainment
Vacation
Food
Expenses
LKR142,650.64
Transport
Health
Shopping

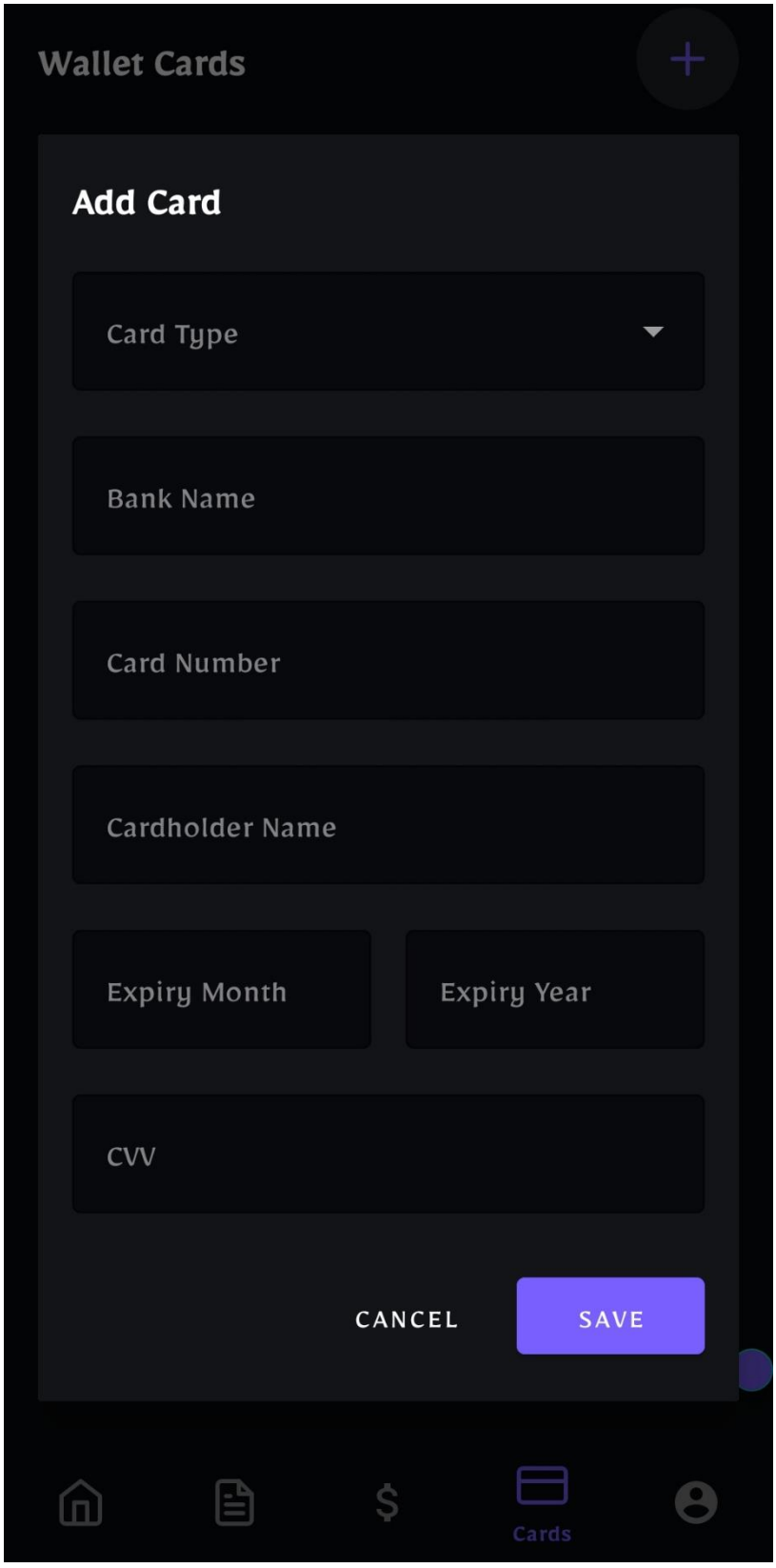

Statistics
April 2025
Health
Shopping
Vacation LKR18,900.32
Transport LKR20,850.00
Health LKR19,500.32
Shopping LKR30,200.00
Food LKR37,500.00
Entertainment LKR15,700.00
Daily Expense Trend
125,000
17,651
120,000
100,000
80,000
60,000
40,000
20,000
0
5
10
15
20
25
30

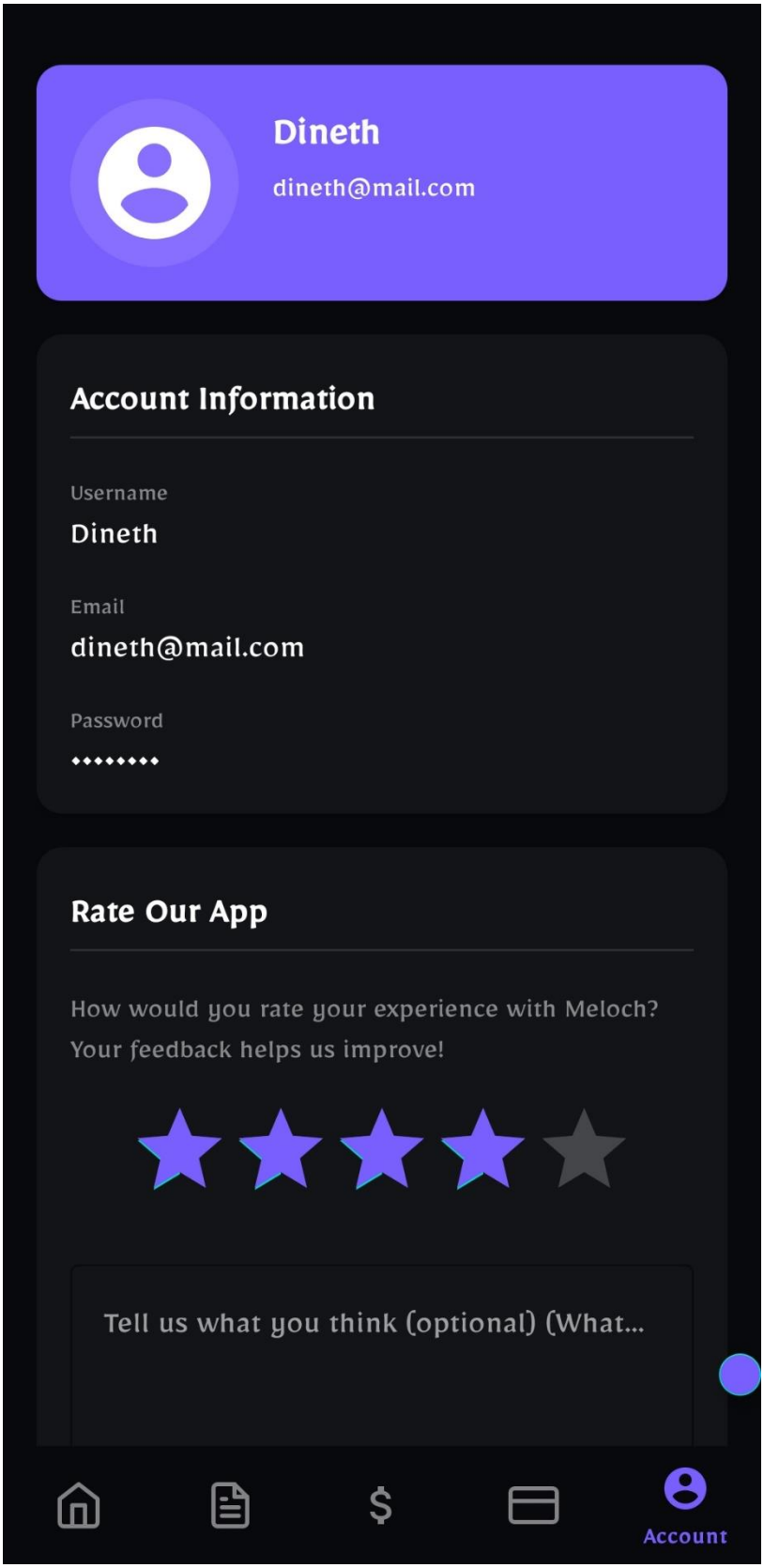

This Year
21 April 2025
Health
-LKR4,500.32
Debit Card
21 Apr 2025
Vacation
-LKR400.32
Debit Card
21 Apr 2025
Entertainment
-LKR6,700.00
Credit Card
21 Apr 2025
Transport
-LKR850.00
Cash
21 Apr 2025
Shopping
-LKR5,200.00
Debit Card
21 Apr 2025
20 April 2025
Records

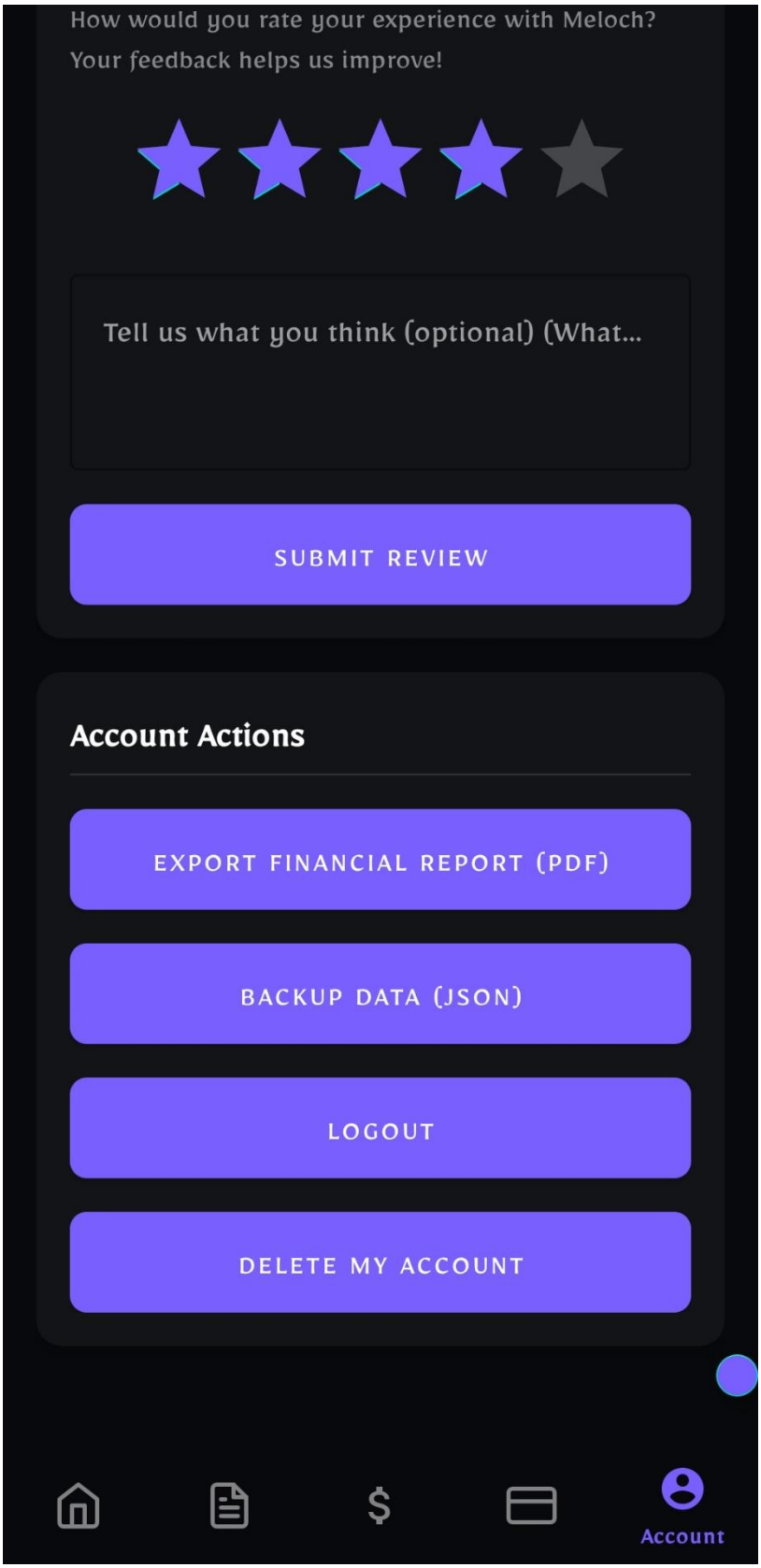

This Year
Transport
-LKR8,500.00
Cash
20 Apr 2025
Food
-LKR17,500.00
Debit Card
20 Apr 2025
Salary
+LKR300,000.00
Cash
20 Apr 2025
Entertainment
-LKR7,000.00
Debit Card
20 Apr 2025
Food
-LKR20,000.00
Cash
20 Apr 2025
Side Business
+LKR25,000.00
Debit Card
20 Apr 2025
Records

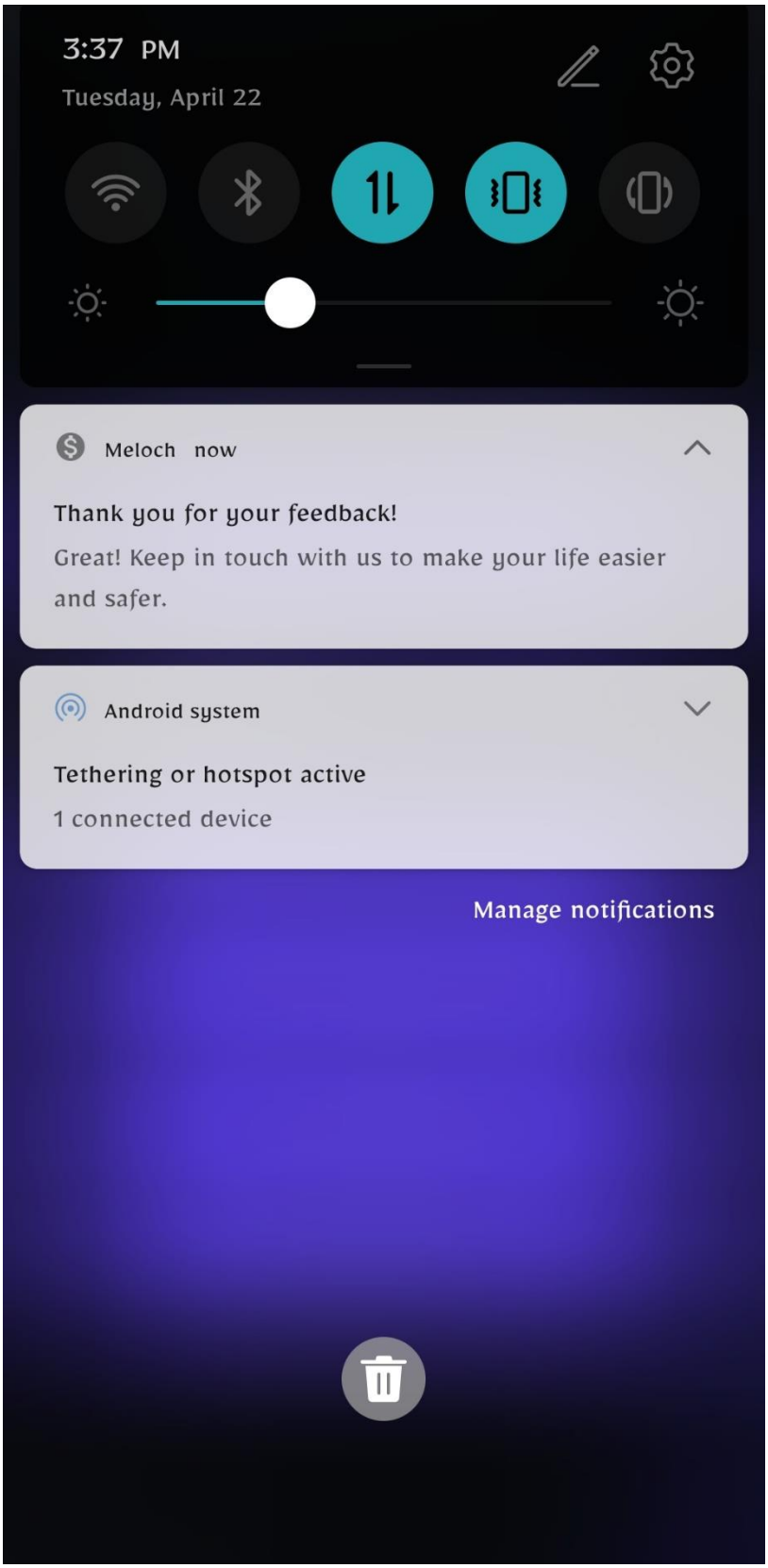

Vacation - LKR6,400.32
Entertainment - LKR8,700.00
Categories
Statistics
Entertainment
Enter Pocket Money
Enter the amount of cash you currently have in your pocket
Amount (LKR)
2500.0
CANCEL
SAVE
Health
LKR10,500.32
Shopping
LKR15,200.00
Vacation
LKR6,400.32
Entertainment
LKR8,700.00
Home

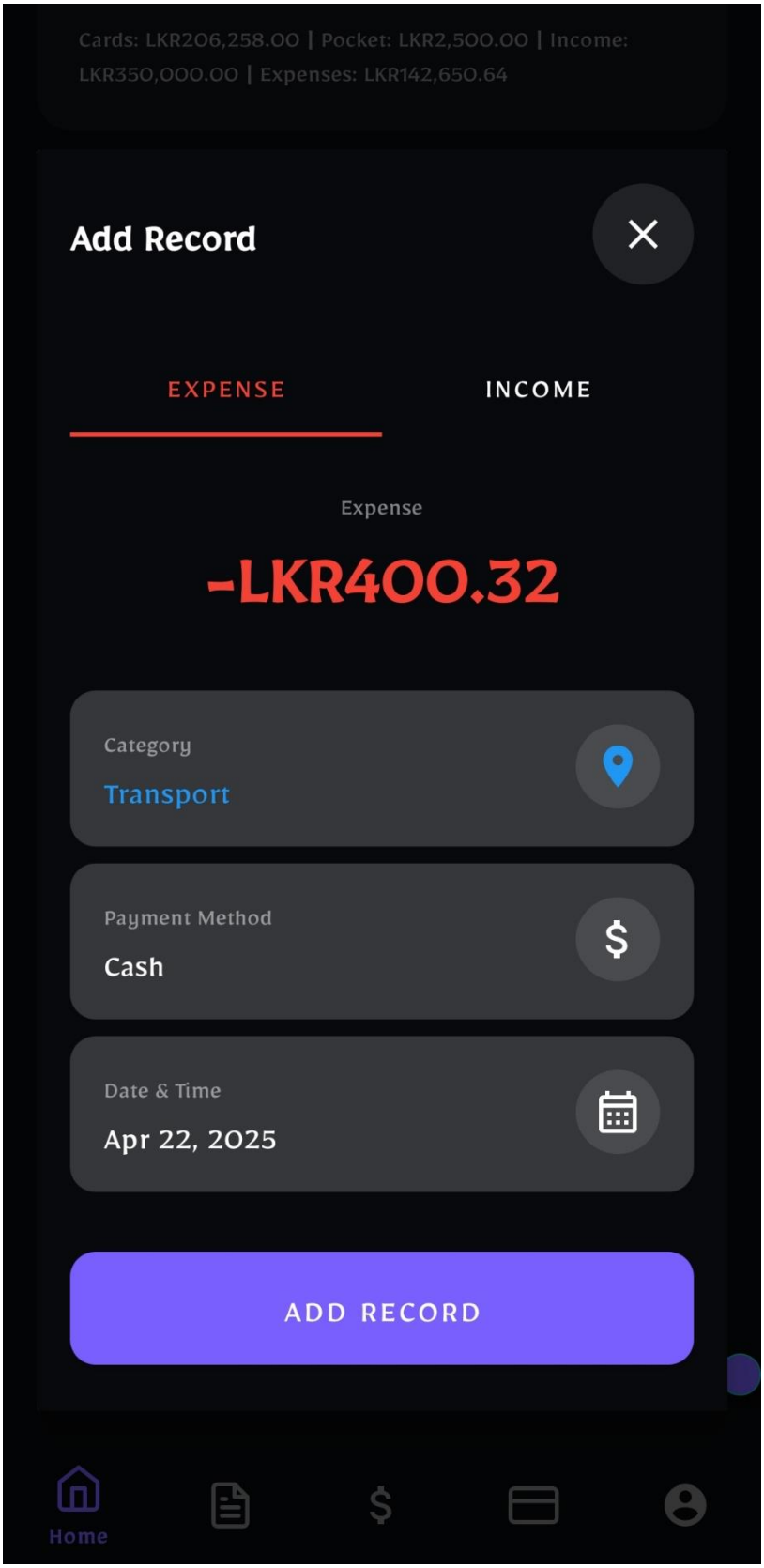

Wallet Cards
My Credit Card
HATTON NATIONAL BANK
4605532180805147
D G A D HIRUSHA
CVV: 028
12/29
My Debit Card
BANK OF CEYLON
**** **** **** 3306
D G A D HIRUSHA
CVV: ***
/
Cards

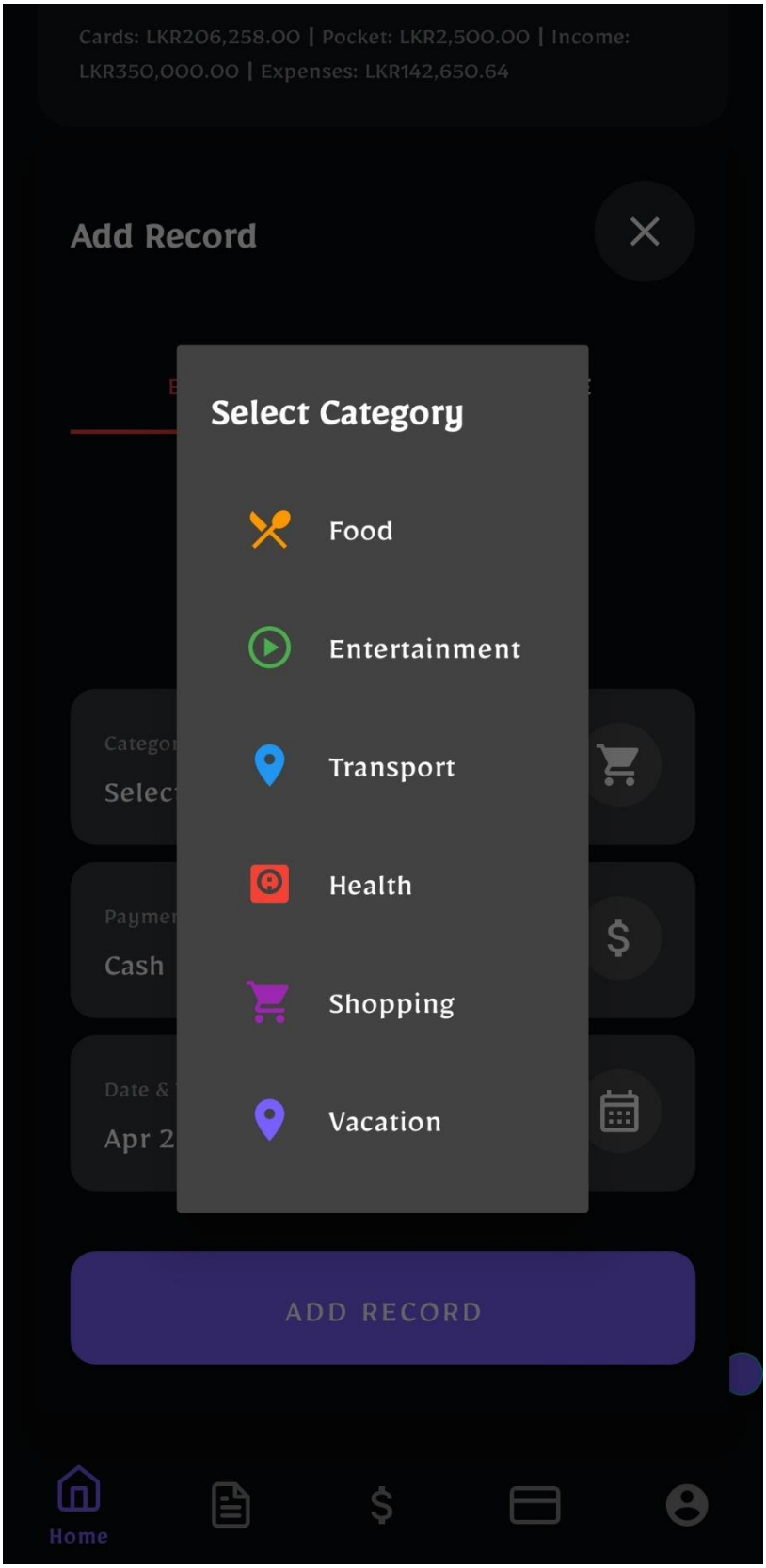

Wallet Cards
Add Card
Card Type
Bank Name
Card Number
Cardholder Name
Expiry Month
Expiry Year
CVV
CANCEL
SAVE
Cards

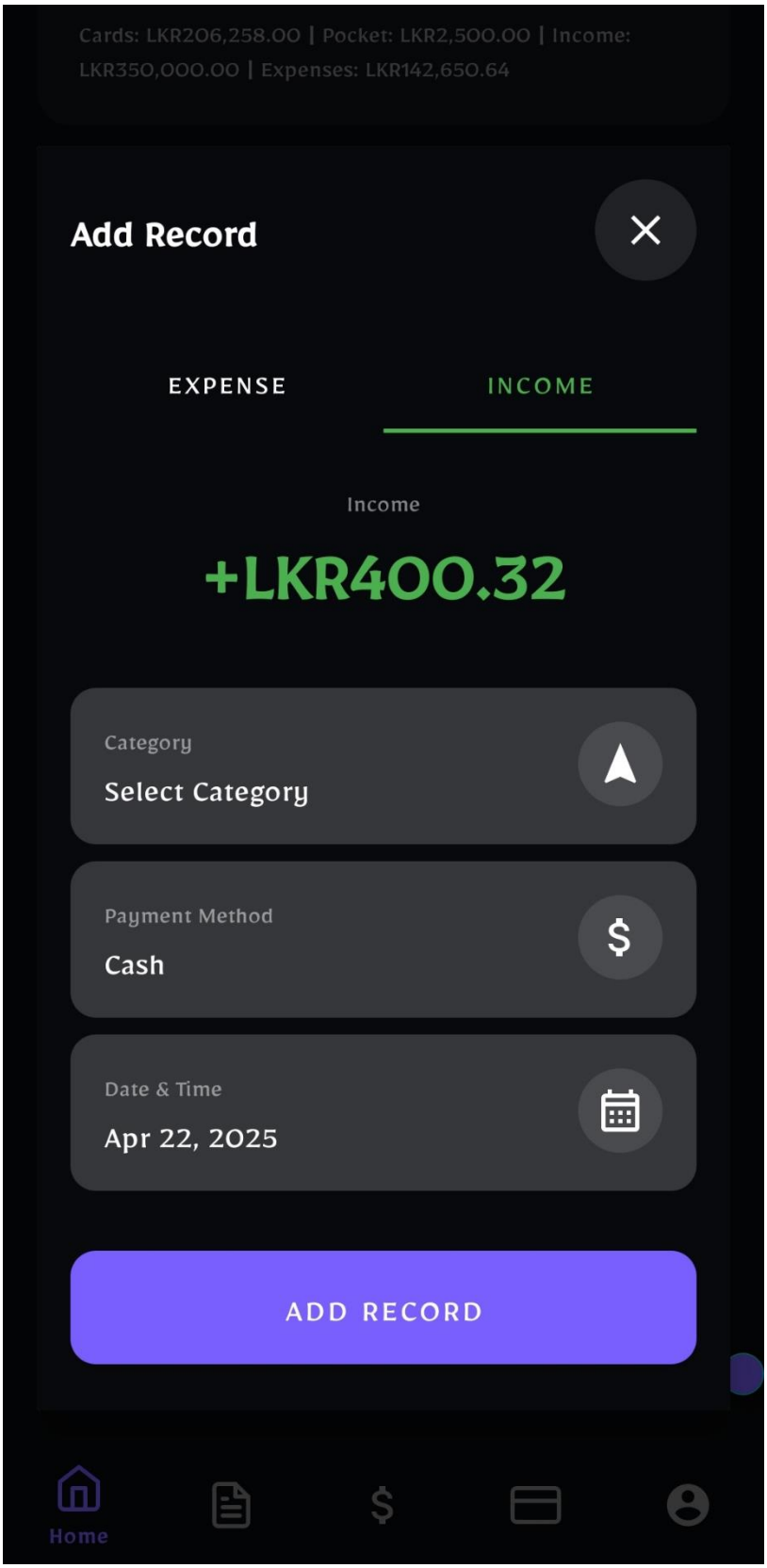

Dineth
dineth@mail.com
Account Information
Username
Dineth
Email
dineth@mail.com
Password
••••••••
Rate Our App
How would you rate your experience with Meloch?
Your feedback helps us improve!
Tell us what you think (optional) (What...
Account

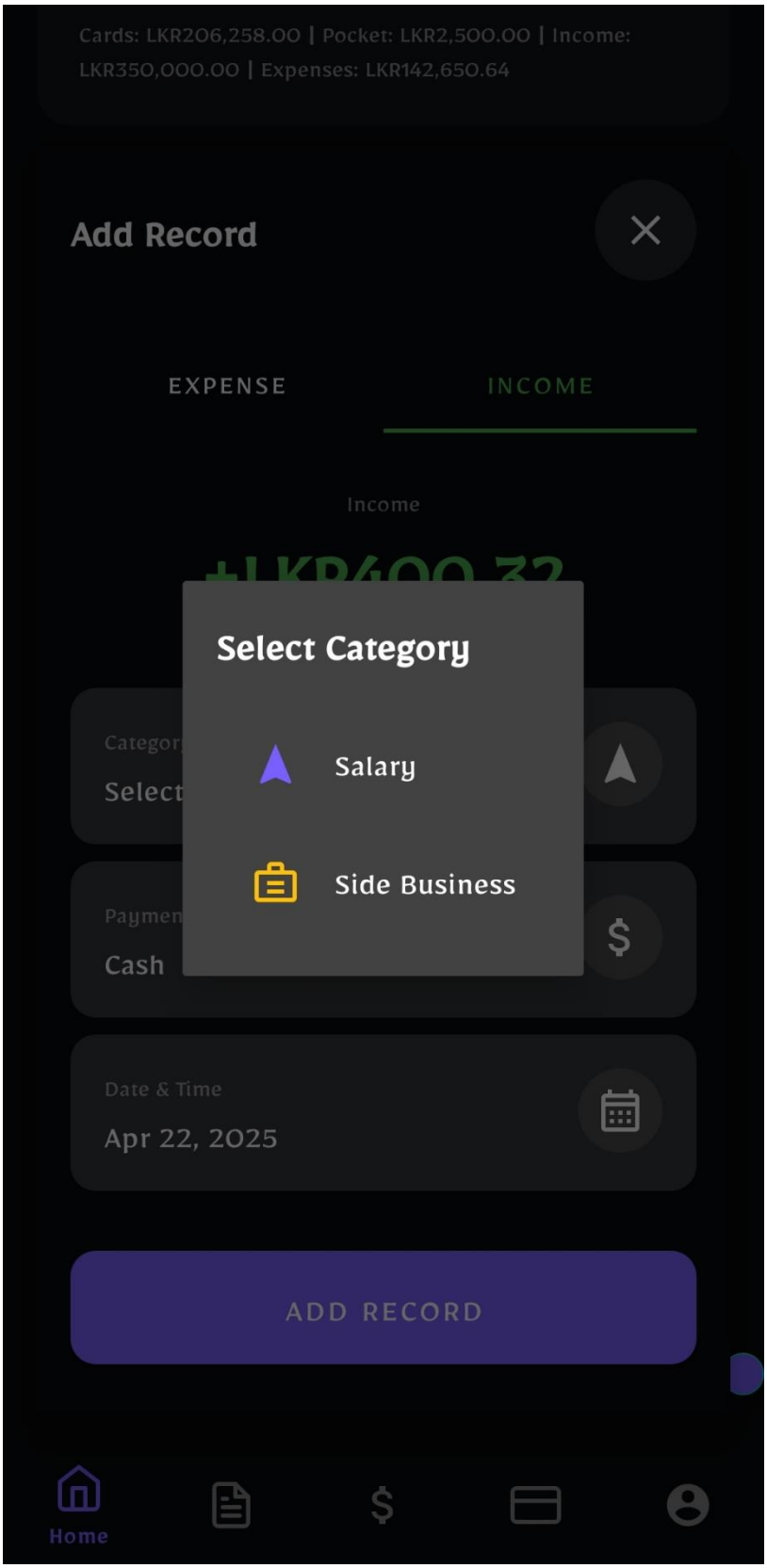

How would you rate your experience with Meloch?
Your feedback helps us improve!
Tell us what you think (optional) (What...
SUBMIT REVIEW
Account Actions
EXPORT FINANCIAL REPORT (PDF)
BACKUP DATA (JSON)
LOGOUT
DELETE MY ACCOUNT
Account

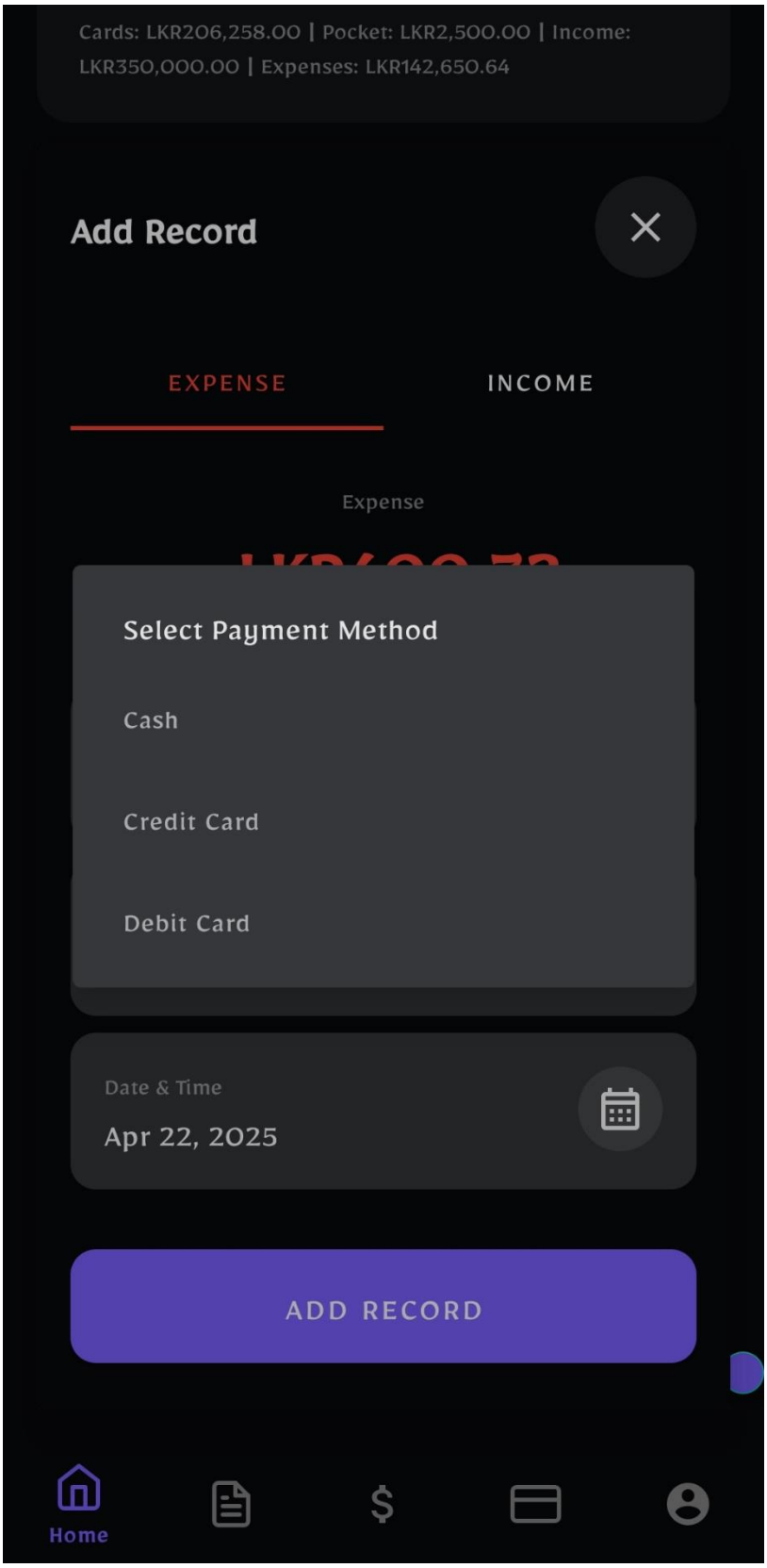

3:37 PM
Tuesday, April 22
Meloch now
Thank you for your feedback!
Great! Keep in touch with us to make your life easier and safer.
Android system
Tethering or hotspot active
1 connected device
Manage notifications

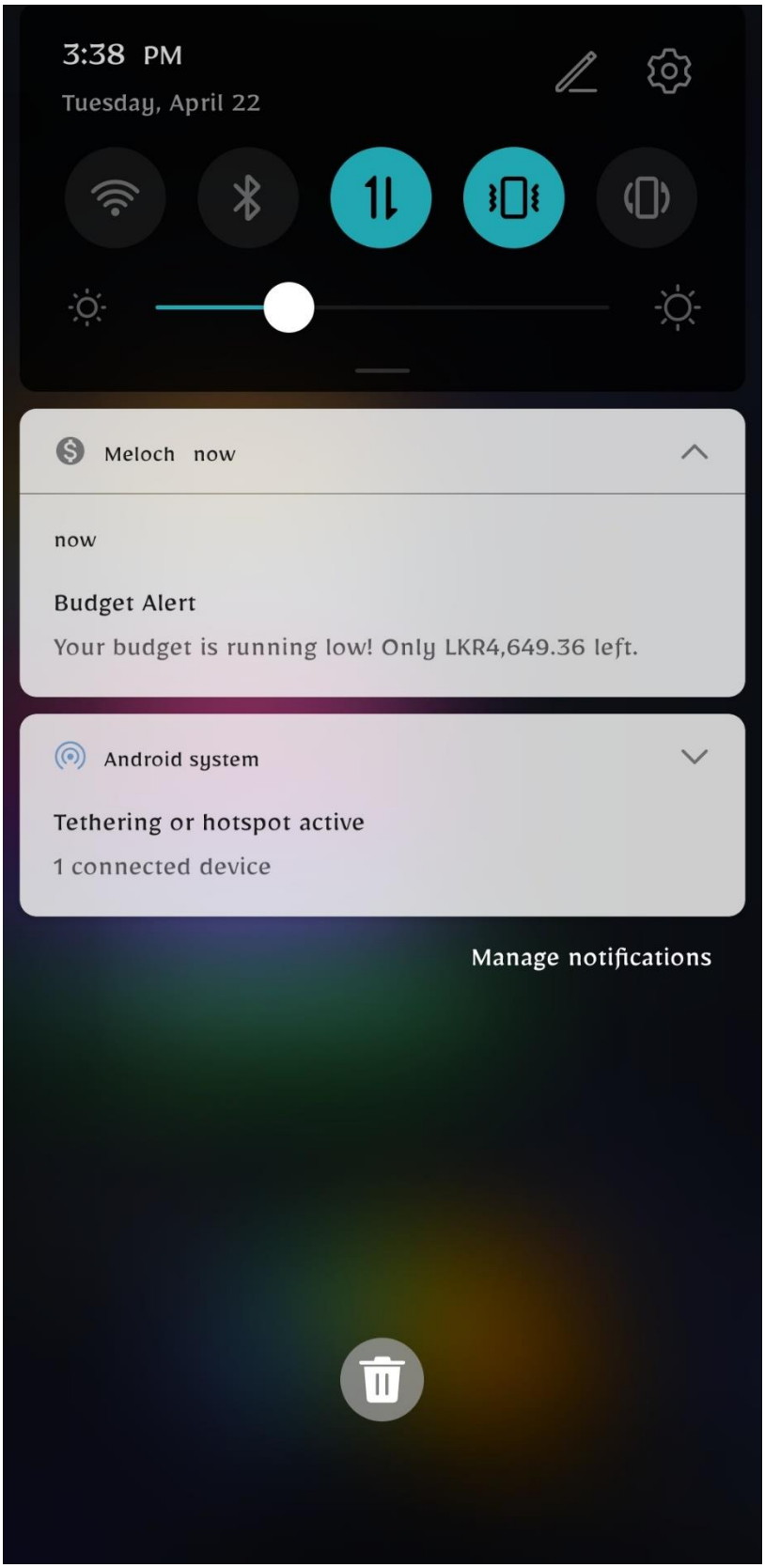

Cards: LKR206,258.00 | Pocket: LKR2,500.00 | Income: LKR350,000.00 | Expenses: LKR142,650.64
Add Record
EXPENSE
INCOME
Expense
-LKR400.32
Category
Transport
Payment Method
Cash
Date & Time
Apr 22, 2025
ADD RECORD
Home

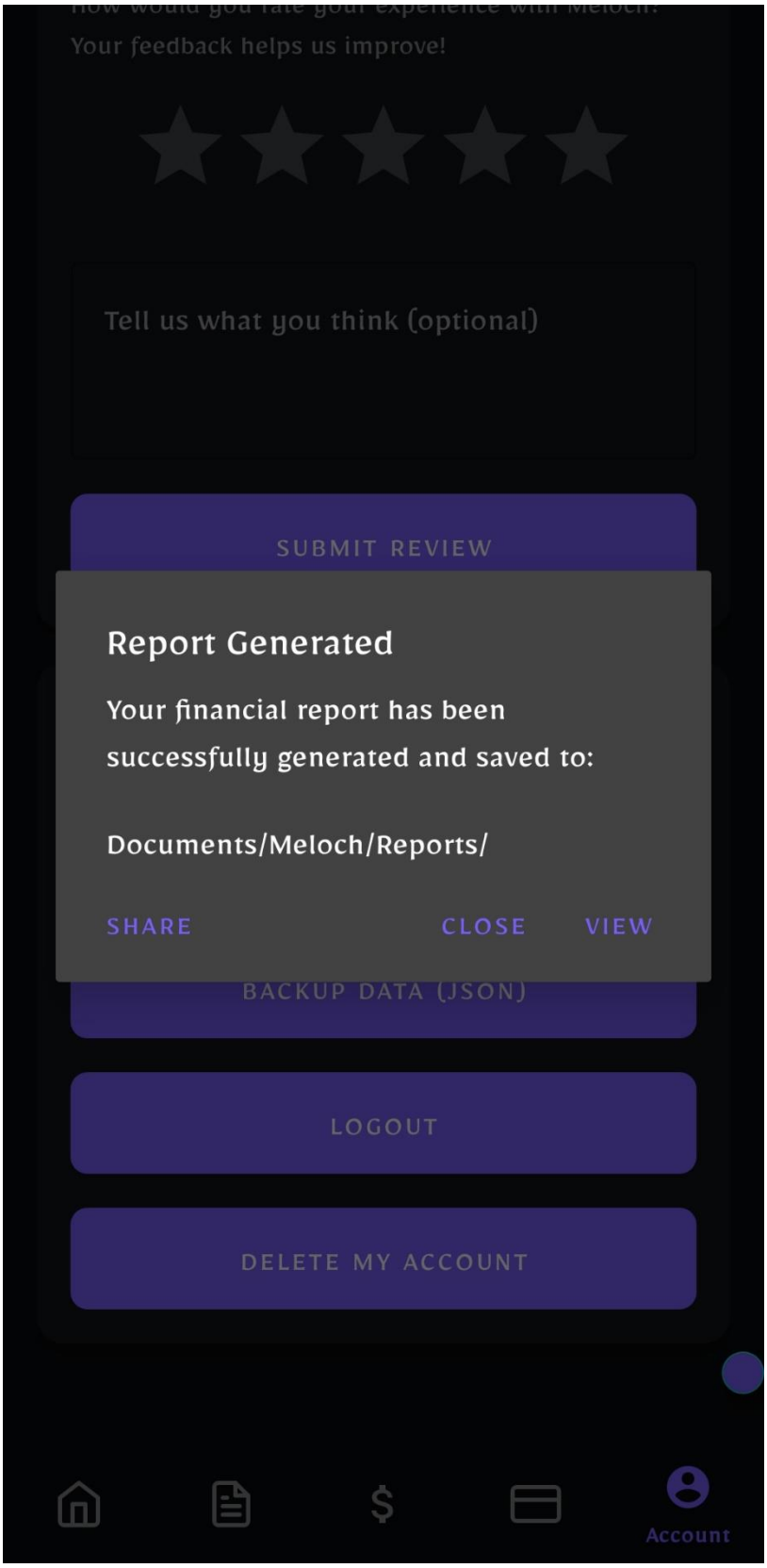

Cards: LKR206,258.00 | Pocket: LKR2,500.00 | Income: LKR350,000.00 | Expenses: LKR142,650.64
Add Record
Select Category
Food
Entertainment
Transport
Health
Shopping
Vacation
Cash
Apr 2
ADD RECORD
Home

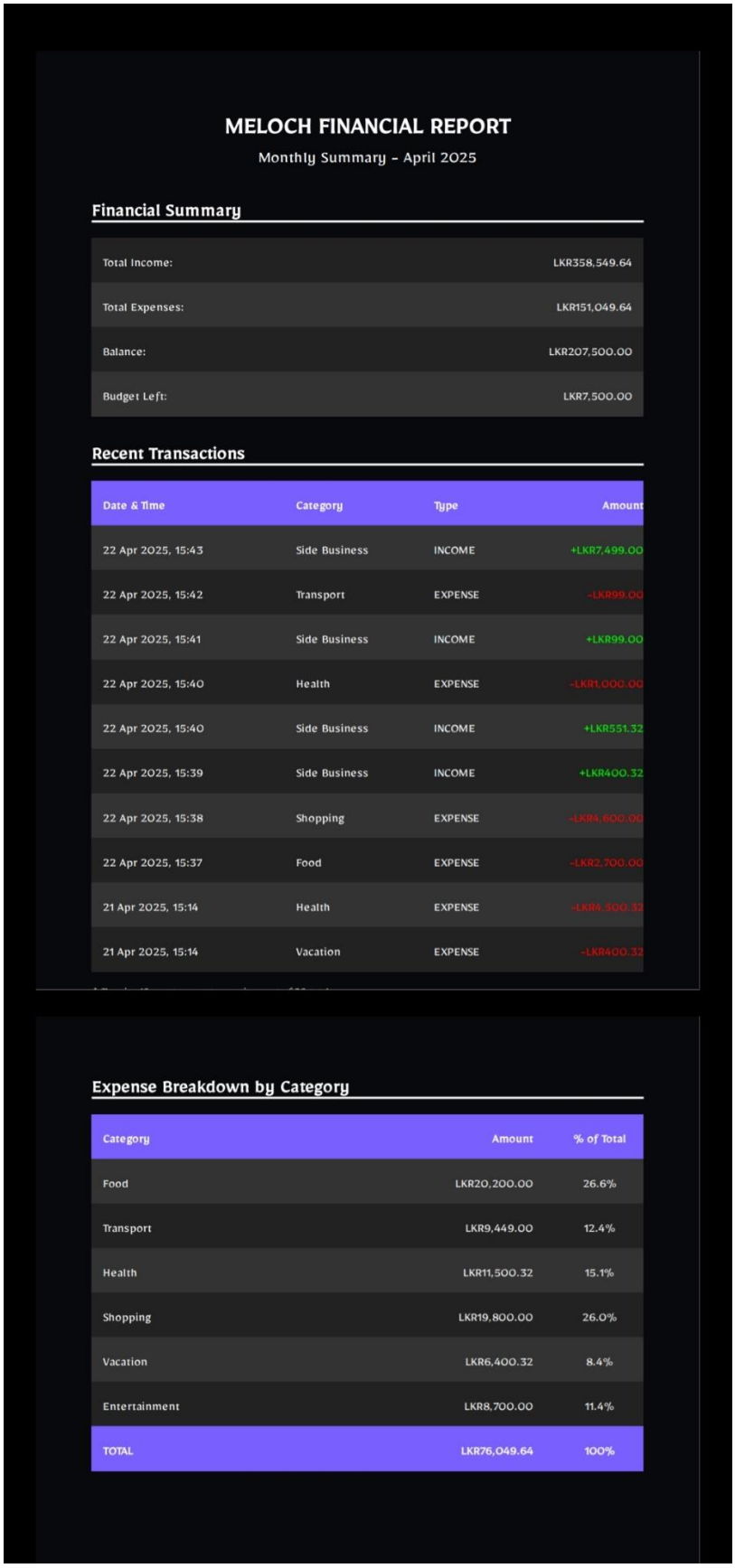

Cards: LKR206,258.00 | Pocket: LKR2,500.00 | Income: LKR350,000.00 | Expenses: LKR142,650.64
Add Record
EXPENSE
INCOME
Income
+LKR400.32
Category
Select Category
Payment Method
Cash
Date & Time
Apr 22, 2025
ADD RECORD
Home


Cards: LKR206,258.00 | Pocket: LKR2,500.00 | Income: LKR350,000.00 | Expenses: LKR142,650.64
Add Record
EXPENSE
INCOME
Income
Select Category
Salary
Side Business
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
EXPENSE
INCOME
Expense
Select Payment Method
Cash
Credit Card
Debit Card
Date & Time
Apr 22, 2025
ADD RECORD
Home


3:38 PM
Tuesday, April 22
Meloch now
now
Budget Alert
Your budget is running low! Only LKR4,649.36 left.
Android system
Tethering or hotspot active
1 connected device
Manage notifications


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

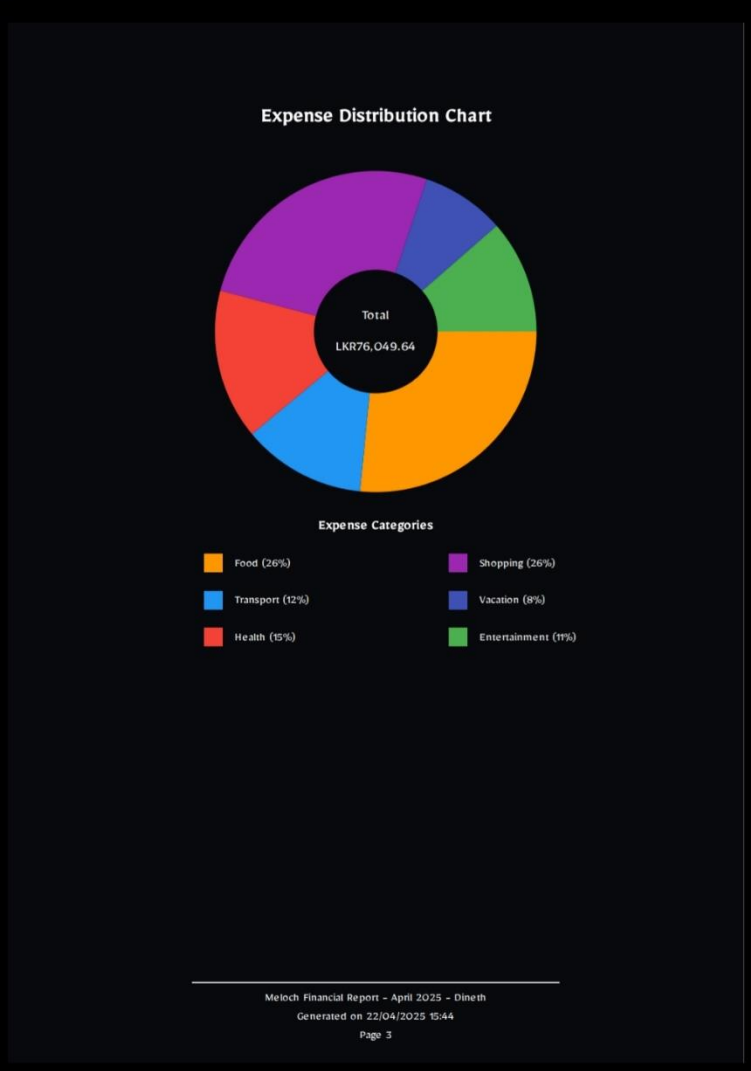
Recent Transactions

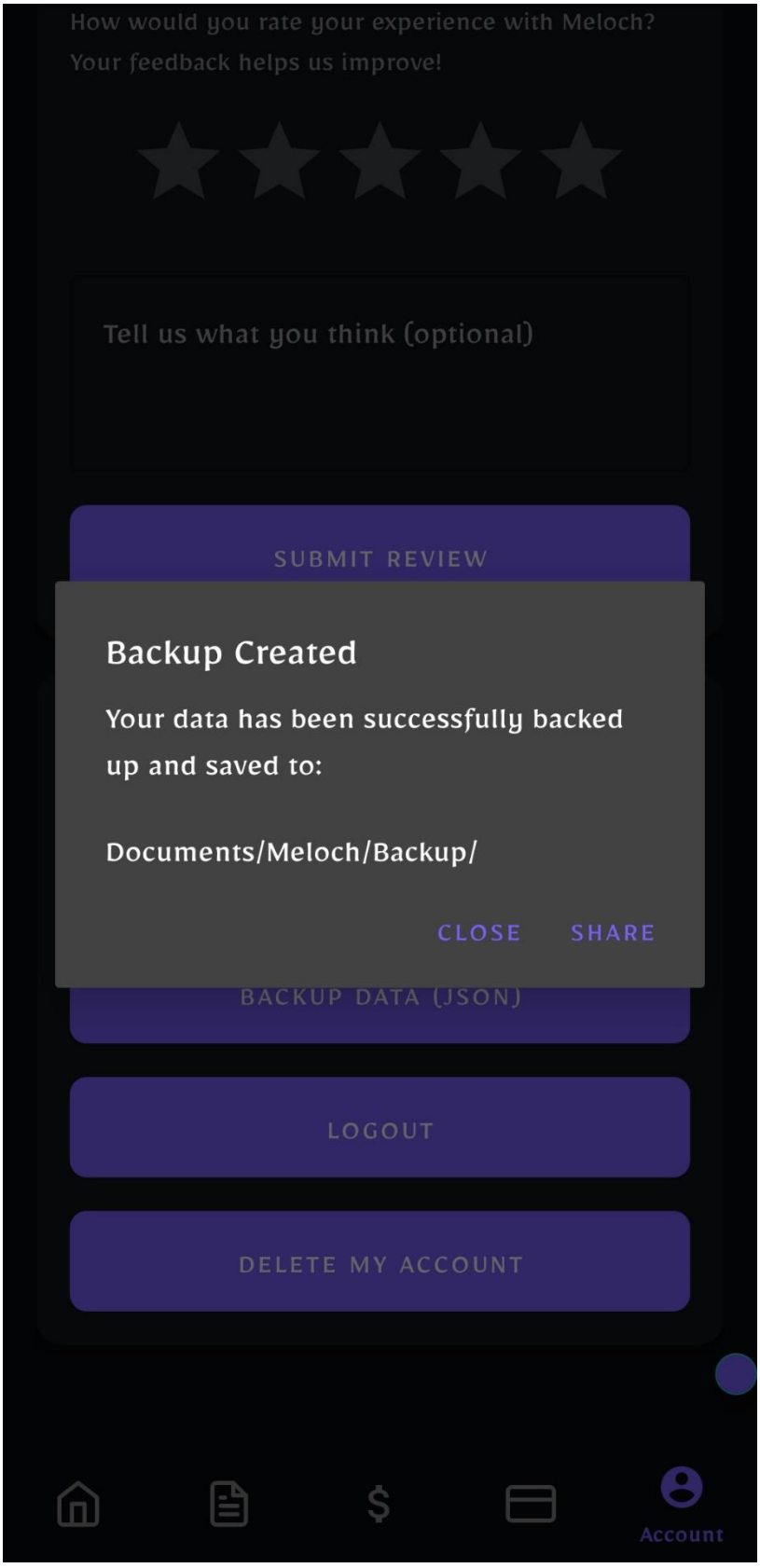
Date & Time	Category	Type	Amount
22 Apr 2025, 15:43	Side Business	INCOME	+LKR7,499.00
22 Apr 2025, 15:42	Transport	EXPENSE	-LKR99.00
22 Apr 2025, 15:41	Side Business	INCOME	+LKR99.00
22 Apr 2025, 15:40	Health	EXPENSE	-LKR1,000.00
22 Apr 2025, 15:40	Side Business	INCOME	+LKR551.32
22 Apr 2025, 15:39	Side Business	INCOME	+LKR400.32
22 Apr 2025, 15:38	Shopping	EXPENSE	-LKR4,600.00
22 Apr 2025, 15:37	Food	EXPENSE	-LKR2,700.00
21 Apr 2025, 15:14	Health	EXPENSE	-LKR4,500.32
21 Apr 2025, 15:14	Vacation	EXPENSE	-LKR400.32

Expense Breakdown by Category

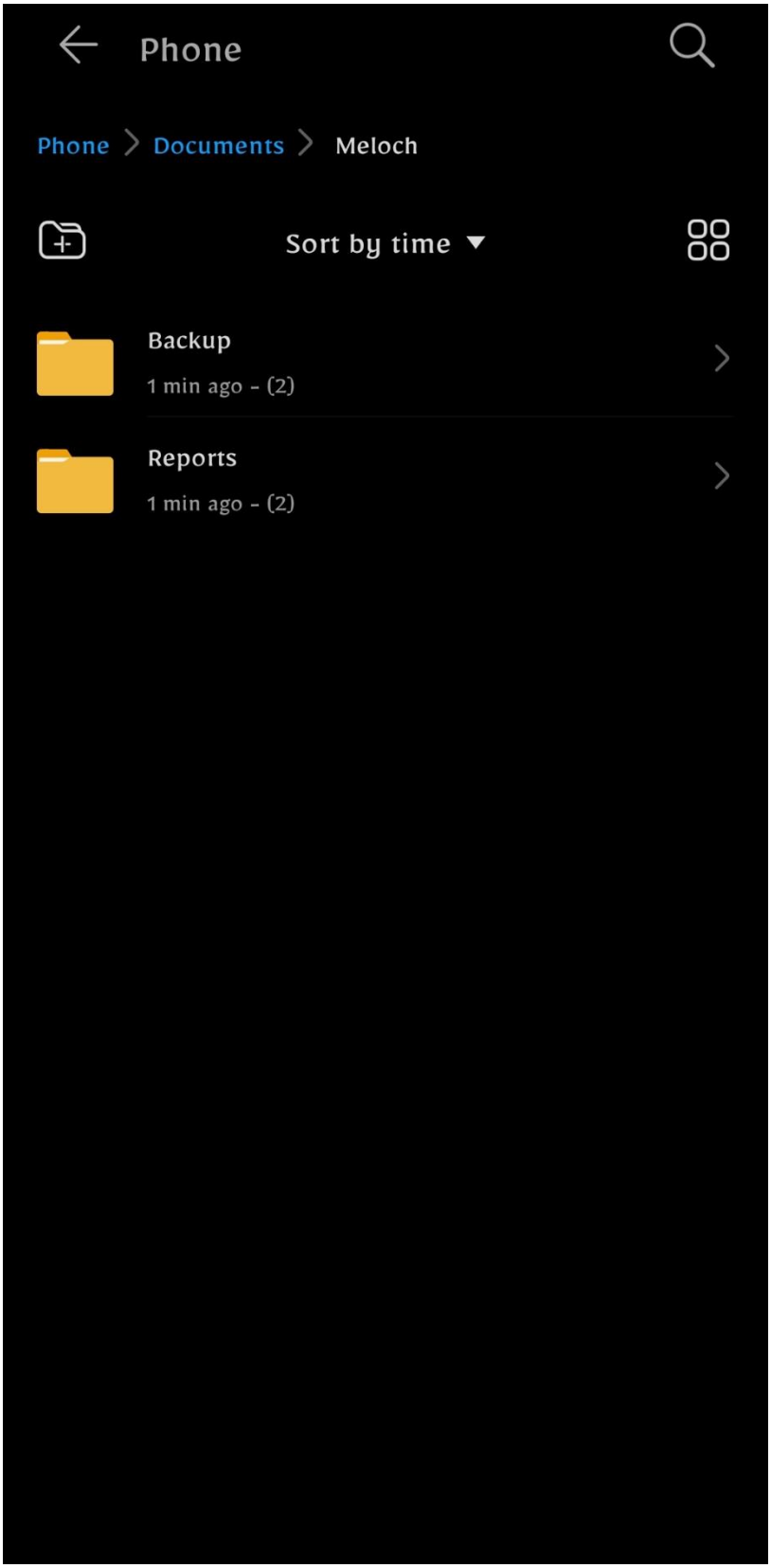
Category	Amount	% of Total
Food	LKR20,200.00	26.6%
Transport	LKR9,449.00	12.4%
Health	LKR11,500.32	15.1%
Shopping	LKR19,800.00	26.0%
Vacation	LKR6,400.32	8.4%
Entertainment	LKR8,700.00	11.4%
TOTAL	LKR76,049.64	100%

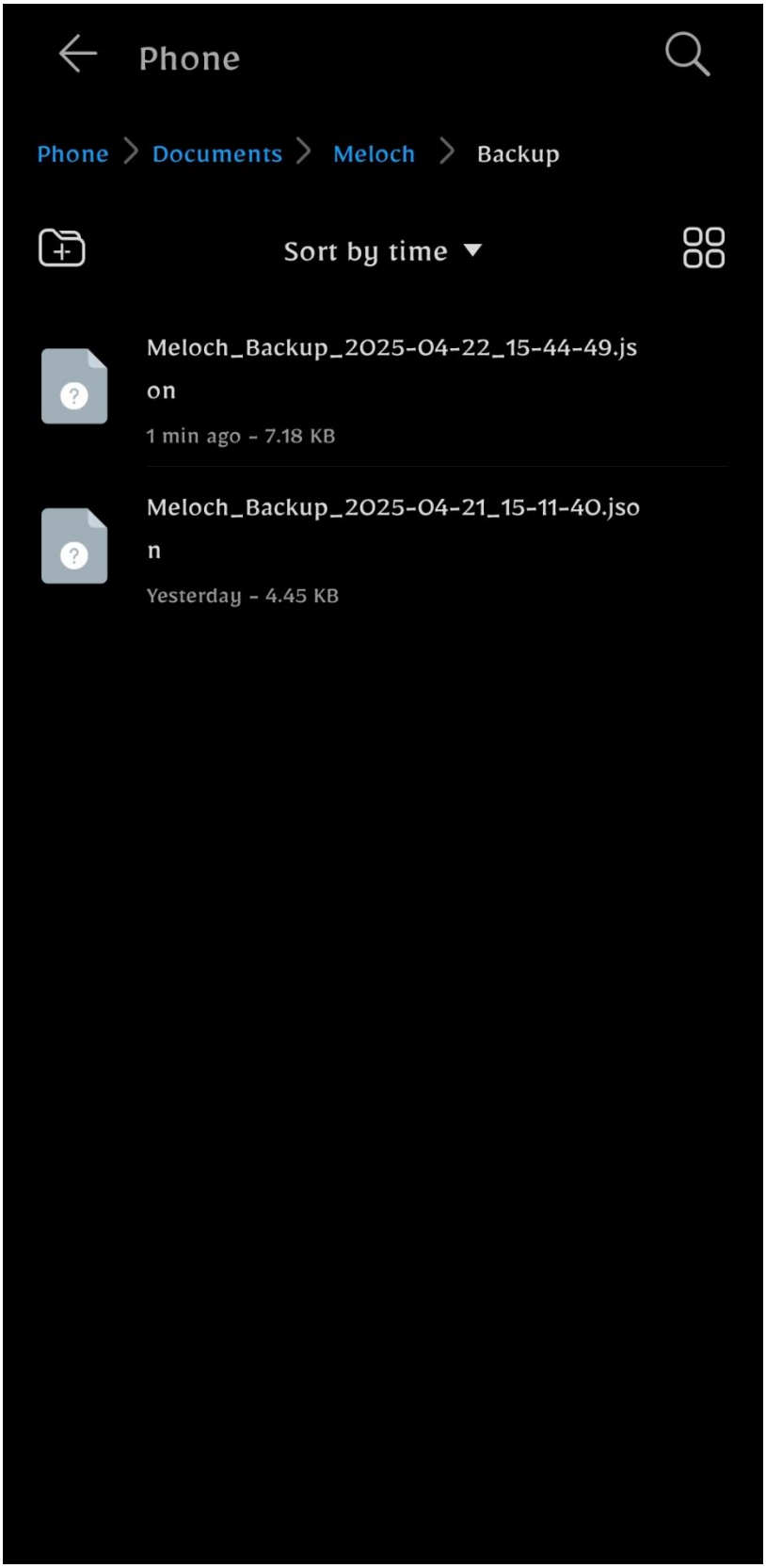
Entertainment	LKR8,700.00	11.4%
TOTAL	LKR76,049.64	100%

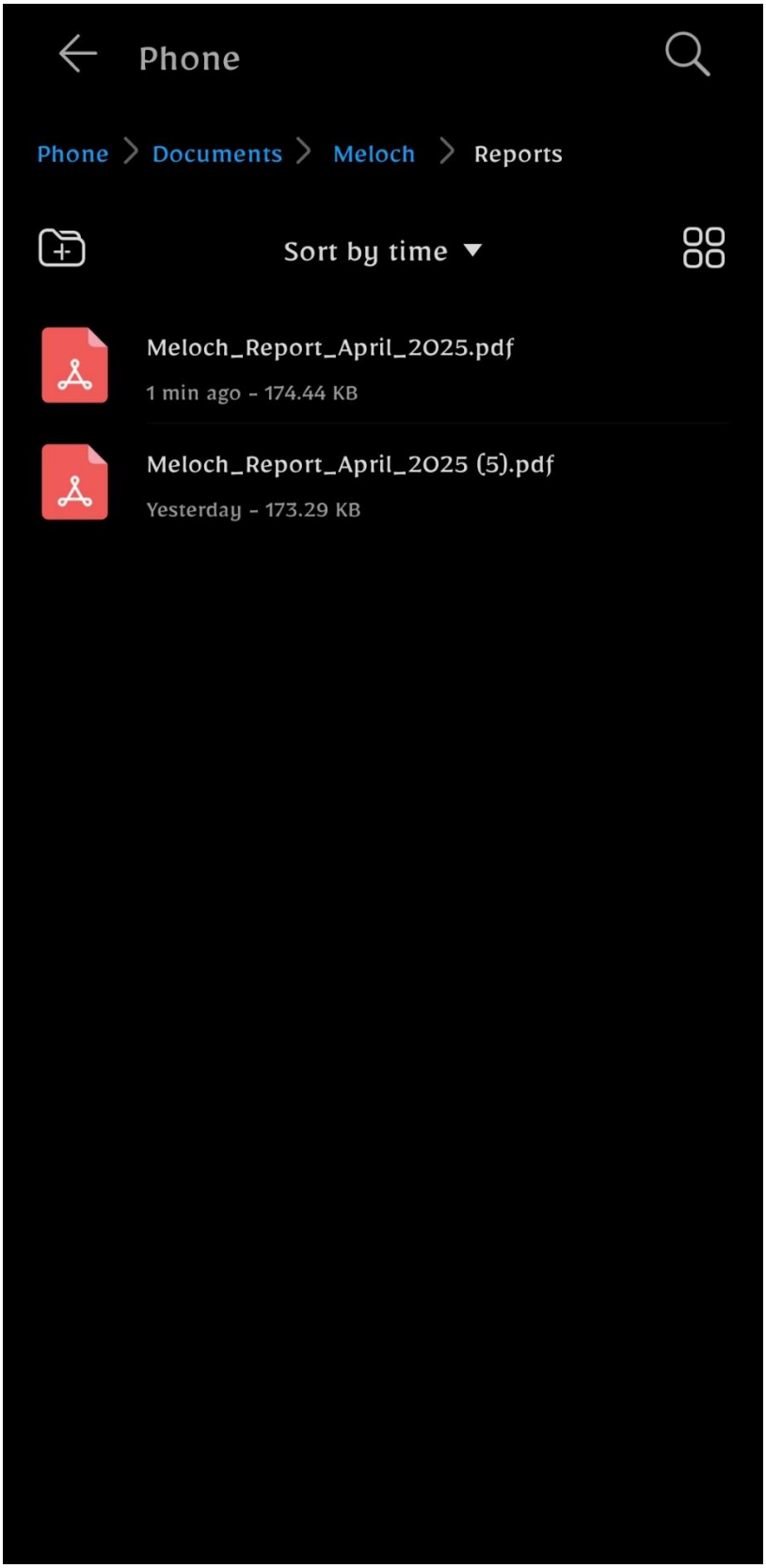



How would you rate your experience with Meloch?
Your feedback helps us improve!
Tell us what you think (optional)
SUBMIT REVIEW
Backup Created
Your data has been successfully backed up and saved to:
Documents/Meloch/Backup/
CLOSE
SHARE
BACKUP DATA (JSON)
LOGOUT
DELETE MY ACCOUNT
Account


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",


Phone
Phone
Documents
Meloch
Sort by time
Backup
1 min ago - (2)
Reports
1 min ago - (2)

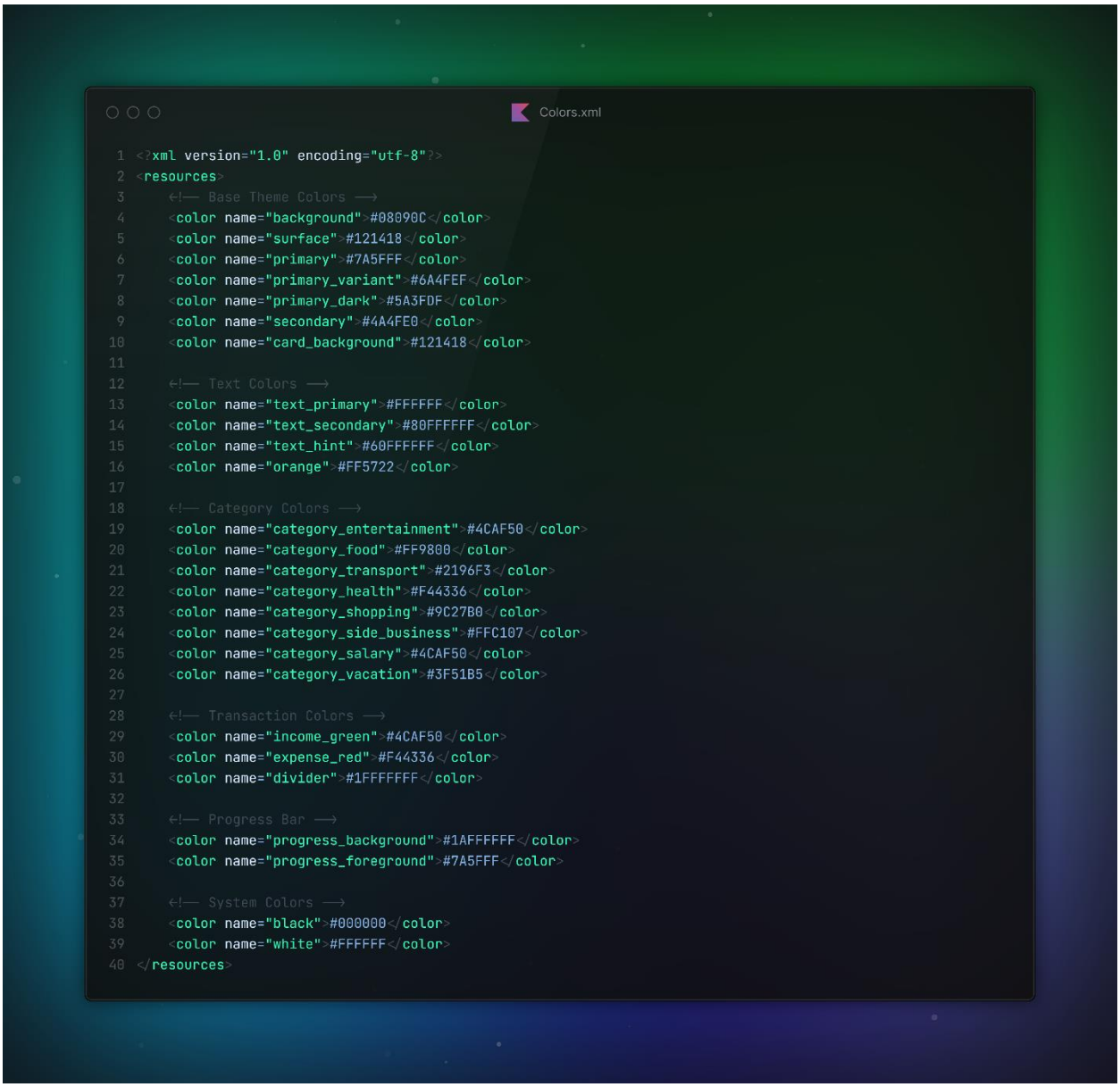

Phone
Phone > Documents > Meloch > Backup
Sort by time
Meloch_Backup_2025-04-22_15-44-49.json
1 min ago - 7.18 KB
Meloch_Backup_2025-04-21_15-11-40.json
Yesterday - 4.45 KB


Phone
Phone > Documents > Meloch > Reports
Sort by time
Meloch_Report_April_2025.pdf
1 min ago - 174.44 KB
Meloch_Report_April_2025 (5).pdf
Yesterday - 173.29 KB

Strings XML


strings.xml
<resources>
<string name="app_name">Meloch</string>
<string name="no_transactions">No transactions yet</string>
<!-- Statistics Activity -->
<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 Texts -->
<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_vs_expense">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_enter_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_this_week">This Week</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_account">Account</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_backup_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>

Colors XML


Colors.xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
<!-- Base Theme Colors -->
<color name="background">#08090C</color>
<color name="surface">#121418</color>
<color name="primary">#7A5FFF</color>
<color name="primary_variant">#6A4FEF</color>
<color name="primary_dark">#5A3FDF</color>
<color name="secondary">#4A4FE0</color>
<color name="card_background">#121418</color>
<!-- Text Colors -->
<color name="text_primary">#FFFFFF</color>
<color name="text_secondary">#80FFFFFF</color>
<color name="text_hint">#60FFFFFF</color>
<color name="orange">#FF5722</color>
<!-- Category Colors -->
<color name="category_entertainment">#4CAF50</color>
<color name="category_food">#FF9800</color>
<color name="category_transport">#2196F3</color>
<color name="category_health">#F44336</color>
<color name="category_shopping">#9C27B0</color>
<color name="category_side_business">#FFC107</color>
<color name="category_salary">#4CAF50</color>
<color name="category_vacation">#3F51B5</color>
<!-- Transaction Colors -->
<color name="income_green">#4CAF50</color>
<color name="expense_red">#F44336</color>
<color name="divider">#1FFFFFFF</color>
<!-- Progress Bar -->
<color name="progress_background">#1AFFFFFF</color>
<color name="progress_foreground">#7A5FFF</color>
<!-- System Colors -->
<color name="black">#000000</color>
<color name="white">#FFFFFF</color>
</resources>

SharedPreferences Kt

PreferencesManager.kt

```kotlin
fun getTransactions(): List<Transaction> {
        // Get user-specific key
        val userKey = getUserSpecificKey(KEY_TRANSACTIONS)
        val json = sharedPreferences.getString(userKey, "[]")
        val type = object : TypeToken<List<Transaction>>() {}.type
        return gson.fromJson(json, type) ?: emptyList()
    }

    private fun getUserSpecificKey(baseKey: String): String {
        val email = currentUserEmail
        val result = if (email != null) {
            "${email}_$baseKey"
        } else {
            baseKey
        }

        // Add debug logging to track key generation
        Log.d("UserSpecificKey", "Generated key: $result from base key: $baseKey for user: $email")

        return result
    }

    fun saveTransactions(transactions: List<Transaction>) {
        val userKey = getUserSpecificKey(KEY_TRANSACTIONS)
        val json = gson.toJson(transactions)
        sharedPreferences.edit().putString(userKey, json).apply()
    }

    fun setMonthlyBudget(amount: Double) {
        val userKey = getUserSpecificKey(KEY_MONTHLY_BUDGET)
        sharedPreferences.edit().putFloat(userKey, amount.toFloat()).apply()
    }

    fun getMonthlyBudget(): Double {
        val userKey = getUserSpecificKey(KEY_MONTHLY_BUDGET)
        // Default to 21000 (sum of default category budgets) if not set
        return sharedPreferences.getFloat(userKey, 21000f).toDouble()
    }

    fun setCurrency(currency: String) {
        val userKey = getUserSpecificKey(KEY_CURRENCY)
        sharedPreferences.edit().putString(userKey, currency).apply()
    }

    fun getCurrency(): String {
        val userKey = getUserSpecificKey(KEY_CURRENCY)
        return sharedPreferences.getString(userKey, "LKR") ?: "LKR"
    }

    fun setTotalBalance(balance: Double) {
        val userKey = getUserSpecificKey(KEY_TOTAL_BALANCE)
        sharedPreferences.edit().putFloat(userKey, balance.toFloat()).apply()
    }

    fun getTotalBalance(): Double {
        val userKey = getUserSpecificKey(KEY_TOTAL_BALANCE)
        return sharedPreferences.getFloat(userKey, 0f).toDouble()
    }

    fun getMonthlyExpenses(): Double {
        val currentMonth = Calendar.getInstance().get(Calendar.MONTH)
        val currentYear = Calendar.getInstance().get(Calendar.YEAR)

        return getTransactions()
            .filter { transaction ->
                val cal = Calendar.getInstance().apply { time = transaction.date }
                cal.get(Calendar.MONTH) == currentMonth &&
                cal.get(Calendar.YEAR) == currentYear &&
                transaction.type == TransactionType.EXPENSE
            }
            .sumOf { it.amount }
    }
```

PreferencesManager.kt

```kotlin
fun resetBudget() {
        // Get the current monthly budget and expenses
        val currentBudget = getMonthlyBudget()
        val currentExpenses = getMonthlyExpenses()

        // Reset category expenses
        for (category in Category.values()) {
            setCategoryExpense(category, 0.0)
        }

        // Set the budget left value to equal the total budget
        setBudgetLeft(currentBudget)

        // Store the current timestamp as the last reset time
        setLastResetTime(System.currentTimeMillis())

        // Set the budget reset flag to true
        setBudgetResetFlag(true)

        // Verify the flag was set
        val isReset = isBudgetReset()
        Log.d("Budget", "Verified budget reset flag after setting: $isReset")

        val currentBalance = getTotalBalance()
        val newBalance = currentBalance + currentExpenses
        setTotalBalance(newBalance)
        Log.d("Budget", "Total balance updated: $currentBalance + $currentExpenses = $newBalance")

        // Log the reset for debugging
        Log.d("Budget", "Budget reset. Current budget: $currentBudget, Budget left: $currentBudget, Expenses reset: $currentExpenses")
    }

    fun resetBudgetWithoutBalanceUpdate() {
        // Get the current monthly budget and expenses
        val currentBudget = getMonthlyBudget()
        val currentExpenses = getMonthlyExpenses()

        Log.d("Budget", "Resetting budget without balance update")
        Log.d("Budget", "Current budget: $currentBudget, Current expenses: $currentExpenses")

        // Reset category expenses
        for (category in Category.values()) {
            setCategoryExpense(category, 0.0)
        }

        // Set the budget left value to equal the total budget
        setBudgetLeft(currentBudget)
        Log.d("Budget", "Budget left set to: $currentBudget")

        // Store the current timestamp as the last reset time
        setLastResetTime(System.currentTimeMillis())

        // Set the budget reset flag to true
        setBudgetResetFlag(true)

        // Verify the flag was set
        val isReset = isBudgetReset()
        Log.d("Budget", "Verified budget reset flag after setting: $isReset")

        // Log the reset for debugging
        Log.d("Budget", "Budget reset without balance update. Current budget: $currentBudget, Budget left: $currentBudget, Expenses reset: $currentExpenses")
    }

    fun setBudgetResetFlag(reset: Boolean) {
        val userKey = getUserSpecificKey(KEY_BUDGET_RESET_FLAG)
        Log.d("Budget", "Setting budget reset flag to: $reset for key: $userKey")
        sharedPreferences.edit().putBoolean(userKey, reset).commit()
    }

    fun isBudgetReset(): Boolean {
        val userKey = getUserSpecificKey(KEY_BUDGET_RESET_FLAG)
        val isReset = sharedPreferences.getBoolean(userKey, false)
        Log.d("Budget", "Budget reset flag: $isReset for key: $userKey")
        return isReset
    }
```

PreferencesManager.kt

```kotlin
fun setBudgetLeft(amount: Double) {
        val userKey = getUserSpecificKey(KEY_BUDGET_LEFT)
        Log.d("Budget", "Setting budget left to: $amount for key: $userKey")
        sharedPreferences.edit().putFloat(userKey, amount.toFloat()).commit()
    }

    fun getBudgetLeft(): Double {
        // If budget has never been reset, return the monthly budget
        if (!isBudgetReset()) {
            val monthlyBudget = getMonthlyBudget()
            Log.d("Budget", "Budget not reset, returning monthly budget: $monthlyBudget")
            return monthlyBudget
        }
        val userKey = getUserSpecificKey(KEY_BUDGET_LEFT)
        val budgetLeft = sharedPreferences.getFloat(userKey, getMonthlyBudget().toFloat()).toDouble()
        Log.d("Budget", "Budget reset, returning budget left: $budgetLeft for key: $userKey")
        return budgetLeft
    }

    fun setLastResetTime(timestamp: Long) {
        val userKey = getUserSpecificKey(KEY_LAST_RESET_TIME)
        Log.d("Budget", "Setting last reset time to: $timestamp for key: $userKey")
        sharedPreferences.edit().putLong(userKey, timestamp).commit()
    }

    fun getLastResetTime(): Long {
        val userKey = getUserSpecificKey(KEY_LAST_RESET_TIME)
        val timestamp = sharedPreferences.getLong(userKey, 0)
        Log.d("Budget", "Last reset time: $timestamp for key: $userKey")
        return timestamp
    }

    fun getExpensesSinceLastReset(): Map<Category, Double> {
        val lastResetTime = getLastResetTime()
        if (lastResetTime == 0L) {
            // If never reset, return all expenses
            return getCategoryExpenses()
        }

        // Get transactions since the last reset
        return getTransactions()
            .filter { transaction ->
                transaction.date.time > lastResetTime &&
                transaction.type == TransactionType.EXPENSE
            }
            .groupBy { it.category }
            .mapValues { (_, transactions) -> transactions.sumOf { it.amount } }
    }

    fun getCategoryExpenses(): Map<Category, Double> {
        val currentMonth = Calendar.getInstance().get(Calendar.MONTH)
        val currentYear = Calendar.getInstance().get(Calendar.YEAR)

        return getTransactions()
            .filter { transaction ->
                val cal = Calendar.getInstance().apply { time = transaction.date }
                cal.get(Calendar.MONTH) == currentMonth &&
                cal.get(Calendar.YEAR) == currentYear &&
                transaction.type == TransactionType.EXPENSE
            }
            .groupBy { it.category }
            .mapValues { (_, transactions) -> transactions.sumOf { it.amount } }
    }

    fun getMonthlyIncome(): Double {
        val currentMonth = Calendar.getInstance().get(Calendar.MONTH)
        val currentYear = Calendar.getInstance().get(Calendar.YEAR)

        return getTransactions()
            .filter { transaction ->
                val cal = Calendar.getInstance().apply { time = transaction.date }
                cal.get(Calendar.MONTH) == currentMonth &&
                cal.get(Calendar.YEAR) == currentYear &&
                transaction.type == TransactionType.INCOME
            }
            .sumOf { it.amount }
    }

    fun updateMonthlyIncome(amount: Double) {
        val currentIncome = getMonthlyIncome()
        val newIncome = currentIncome + amount
        val userKey = getUserSpecificKey(KEY_MONTHLY_INCOME)
        sharedPreferences.edit().putFloat(userKey, newIncome.toFloat()).apply()
    }

    fun updateMonthlyExpenses(amount: Double) {
        val currentExpenses = getMonthlyExpenses()
        val newExpenses = currentExpenses + amount
        val userKey = getUserSpecificKey(KEY_MONTHLY_EXPENSES)
        sharedPreferences.edit().putFloat(userKey, newExpenses.toFloat()).apply()
    }

    fun updateCategoryExpense(category: Category, amount: Double) {
        val categoryExpenses = getCategoryExpensesFromPrefs()
        val currentAmount = categoryExpenses[category] ?: 0.0
        categoryExpenses[category] = currentAmount + amount
        saveCategoryExpenses(categoryExpenses)
    }

    fun setCategoryExpense(category: Category, amount: Double) {
        val categoryExpenses = getCategoryExpensesFromPrefs()
        categoryExpenses[category] = amount
        saveCategoryExpenses(categoryExpenses)
    }
```

PreferencesManager.kt

```kotlin
private fun getCategoryExpensesFromPrefs(): MutableMap<Category, Double> {
        val userKey = getUserSpecificKey(KEY_CATEGORY_EXPENSES)
        val json = sharedPreferences.getString(userKey, "{}")
        val type = object : TypeToken<Map<String, Double>>() {}.type
        val stringMap: Map<String, Double> = gson.fromJson(json, type) ?: emptyMap()

        // Convert string keys to Category enum
        return stringMap.entries.associate {
            Category.valueOf(it.key) to it.value
        }.toMutableMap()
    }

    private fun saveCategoryExpenses(expenses: Map<Category, Double>) {
        // Convert Category enum keys to strings
        val stringMap = expenses.entries.associate {
            it.key.name to it.value
        }
        val json = gson.toJson(stringMap)
        val userKey = getUserSpecificKey(KEY_CATEGORY_EXPENSES)
        sharedPreferences.edit().putString(userKey, json).apply()
    }

    fun deleteTransaction(transactionId: String) {
        val transactions = getTransactions().toMutableList()
        val transaction = transactions.find { it.id == transactionId }

        if (transaction != null) {
            transactions.remove(transaction)
            val json = gson.toJson(transactions)
            val userKey = getUserSpecificKey(KEY_TRANSACTIONS)
            sharedPreferences.edit().putString(userKey, json).apply()

            // Update total balance
            val currentBalance = getTotalBalance()
            val newBalance = when (transaction.type) {
                TransactionType.INCOME -> currentBalance - transaction.amount
                TransactionType.EXPENSE -> currentBalance + transaction.amount
            }
            setTotalBalance(newBalance)

            // Update monthly budget if it was an income transaction
            if (transaction.type == TransactionType.INCOME) {
                val currentBudget = getMonthlyBudget()
                setMonthlyBudget(currentBudget - transaction.amount)
            }

            // Update category expenses for UI purposes if it was an expense
            if (transaction.type == TransactionType.EXPENSE) {
                updateCategoryExpense(transaction.category, -transaction.amount)

                // Update budget left if budget has been reset
                if (isBudgetReset()) {
                    val currentBudgetLeft = getBudgetLeft()
                    setBudgetLeft(currentBudgetLeft + transaction.amount)
                }
            } else if (transaction.type == TransactionType.INCOME) {
                // Update budget left if budget has been reset and it's not a salary income
                if (isBudgetReset() && transaction.category != Category.SALARY) {
                    val currentBudgetLeft = getBudgetLeft()
                    setBudgetLeft(currentBudgetLeft - transaction.amount)
                    Log.d("Budget", "Income (${transaction.category}) removed from budget left:
-${transaction.amount}")
                } else if (transaction.category == Category.SALARY) {
                    Log.d("Budget", "Salary income not removed from budget left: ${transaction.amount}")
                }
            }
        }
    }
```

PreferencesManager.kt

```kotlin
// Category Budget Methods
    fun saveEntertainmentBudget(amount: Double) {
        val userKey = getUserSpecificKey(KEY_ENTERTAINMENT_BUDGET)
        sharedPreferences.edit().putFloat(userKey, amount.toFloat()).apply()
    }

    fun saveFoodBudget(amount: Double) {
        val userKey = getUserSpecificKey(KEY_FOOD_BUDGET)
        sharedPreferences.edit().putFloat(userKey, amount.toFloat()).apply()
    }

    fun saveTransportBudget(amount: Double) {
        val userKey = getUserSpecificKey(KEY_TRANSPORT_BUDGET)
        sharedPreferences.edit().putFloat(userKey, amount.toFloat()).apply()
    }

    fun saveLifestyleBudget(amount: Double) {
        val userKey = getUserSpecificKey(KEY_LIFESTYLE_BUDGET)
        sharedPreferences.edit().putFloat(userKey, amount.toFloat()).apply()
    }

    fun getEntertainmentBudget(): Double {
        val userKey = getUserSpecificKey(KEY_ENTERTAINMENT_BUDGET)
        return sharedPreferences.getFloat(userKey, 10000f).toDouble()
    }

    fun getFoodBudget(): Double {
        val userKey = getUserSpecificKey(KEY_FOOD_BUDGET)
        return sharedPreferences.getFloat(userKey, 5000f).toDouble()
    }

    fun getTransportBudget(): Double {
        val userKey = getUserSpecificKey(KEY_TRANSPORT_BUDGET)
        return sharedPreferences.getFloat(userKey, 1000f).toDouble()
    }

    fun getLifestyleBudget(): Double {
        val userKey = getUserSpecificKey(KEY_LIFESTYLE_BUDGET)
        return sharedPreferences.getFloat(userKey, 5000f).toDouble()
    }

    fun saveCategoryBudget(category: String, amount: Double) {
        when (category) {
            "Entertainment" -> saveEntertainmentBudget(amount)
            "Food" -> saveFoodBudget(amount)
            "Transport" -> saveTransportBudget(amount)
            "Lifestyle" -> saveLifestyleBudget(amount)
        }
    }

    fun getCategoryBudget(category: String): Double {
        return when (category) {
            "Entertainment" -> getEntertainmentBudget()
            "Food" -> getFoodBudget()
            "Transport" -> getTransportBudget()
            "Lifestyle" -> getLifestyleBudget()
            else -> 0.0
        }
    }

    fun clearAllData() {
        // Only clear data for the current user
        val email = currentUserEmail ?: return

        // Get all keys in SharedPreferences
        val allPrefs = sharedPreferences.all
        val editor = sharedPreferences.edit()

        // Remove only keys that start with the current user's email
        for (key in allPrefs.keys) {
            if (key.startsWith("${email}_")) {
                editor.remove(key)
            }
        }

        // Apply changes
        editor.apply()
        Log.d("PreferencesManager", "Cleared all data for user: $email")
    }
}
```

NotificationManager Kt

ProfileNotificationHelper.kt

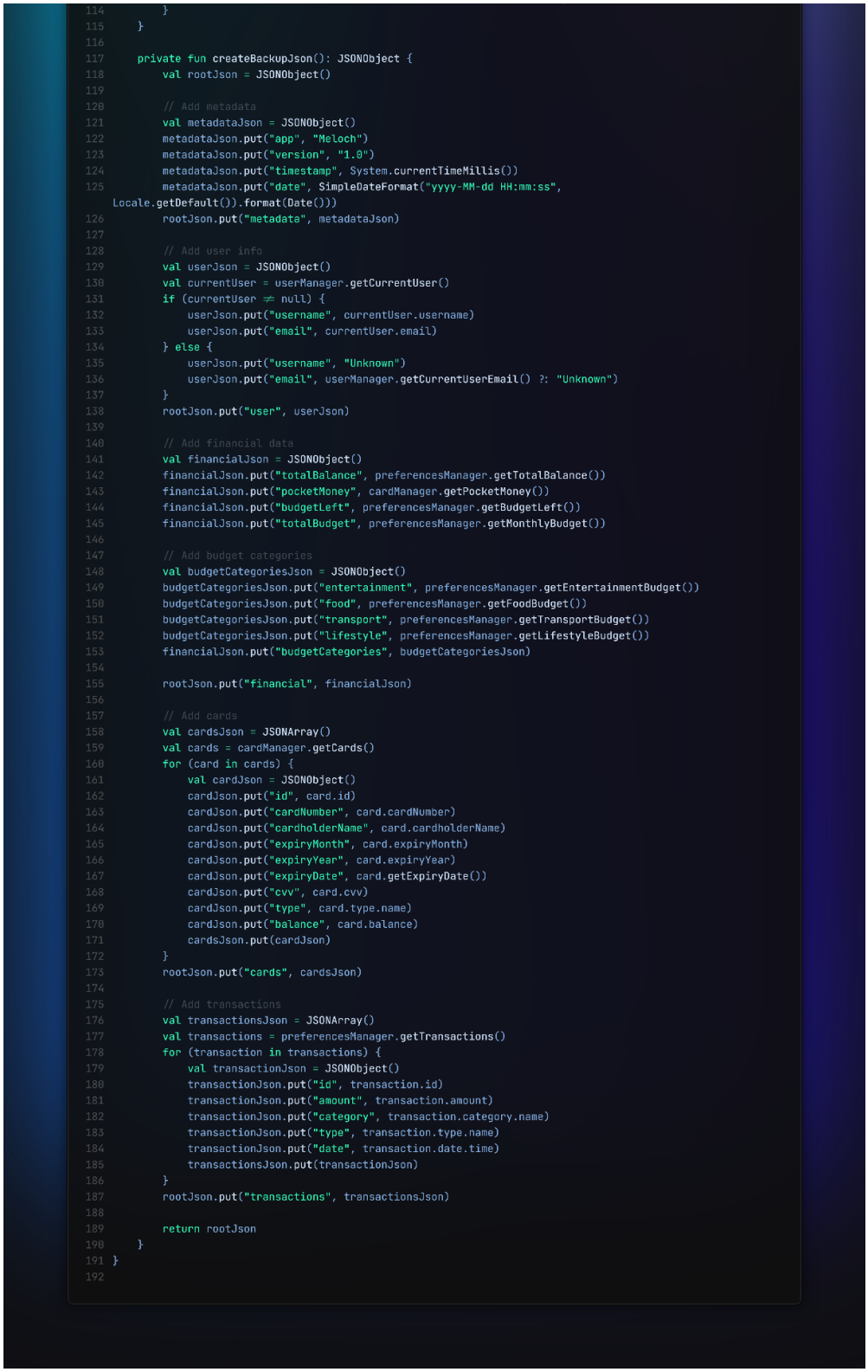
```kotlin
package com.example.meloch.data

import android.app.NotificationChannel
import android.app.NotificationManager
import android.app.PendingIntent
import android.content.Context
import android.content.Intent
import android.os.Build
import androidx.core.app.NotificationCompat
import androidx.core.app.NotificationManagerCompat
import com.example.meloch.MainActivity
import com.example.meloch.R

class ProfileNotificationHelper(private val context: Context) {

    companion object {
        private const val CHANNEL_ID = "profile_channel"
        private const val NOTIFICATION_ID_REVIEW = 2001
    }

    init {
        createNotificationChannel()
    }

    private fun createNotificationChannel() {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            val name = "Profile Notifications"
            val descriptionText = "Notifications related to profile actions"
            val importance = NotificationManager.IMPORTANCE_DEFAULT
            val channel = NotificationChannel(CHANNEL_ID, name, importance).apply {
                description = descriptionText
            }

            // Register the channel with the system
            val notificationManager = context.getSystemService(Context.NOTIFICATION_SERVICE) as
NotificationManager
            notificationManager.createNotificationChannel(channel)
        }
    }

    fun showReviewFeedbackNotification(title: String, message: String) {
        // Create an intent that will open the app when the notification is tapped
        val intent = Intent(context, MainActivity::class.java).apply {
            flags = Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
        }

        val pendingIntent = PendingIntent.getActivity(
            context, 0, intent,
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) PendingIntent.FLAG_IMMUTABLE else 0
        )

        // Build the notification
        val builder = NotificationCompat.Builder(context, CHANNEL_ID)
            .setSmallIcon(R.drawable.ic_notification)
            .setContentTitle(title)
            .setContentText(message)
            .setStyle(NotificationCompat.BigTextStyle().bigText(message))
            .setPriority(NotificationCompat.PRIORITY_DEFAULT)
            .setContentIntent(pendingIntent)
            .setAutoCancel(true)

        // Show the notification
        with(NotificationManagerCompat.from(context)) {
            try {
                notify(NOTIFICATION_ID_REVIEW, builder.build())
            } catch (e: SecurityException) {
                // Handle the case where notification permission is not granted
                e.printStackTrace()
            }
        }
    }
}
```

NotificationHelper.kt

```kotlin
package com.example.meloch.util

import android.app.NotificationChannel
import android.app.NotificationManager
import android.app.PendingIntent
import android.content.Context
import android.content.Intent
import android.os.Build
import androidx.core.app.NotificationCompat
import androidx.core.app.NotificationManagerCompat
import com.example.meloch.MainActivity
import com.example.meloch.R

class NotificationHelper(private val context: Context) {

    companion object {
        private const val CHANNEL_ID = "meloch_budget_alerts"
        private const val BUDGET_LOW_NOTIFICATION_ID = 1001
        private const val BUDGET_LIMIT_NOTIFICATION_ID = 1002
        private const val BUDGET_RESET_NOTIFICATION_ID = 1003
    }

    init {
        createNotificationChannel()
    }


    private fun createNotificationChannel() {
        // Create the NotificationChannel, but only on API 26+ because
        // the NotificationChannel class is new and not in the support library
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            val name = "Budget Alerts"
            val descriptionText = "Notifications about your budget status"
            val importance = NotificationManager.IMPORTANCE_DEFAULT
            val channel = NotificationChannel(CHANNEL_ID, name, importance).apply {
                description = descriptionText
            }
            // Register the channel with the system
            val notificationManager: NotificationManager =
                context.getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
            notificationManager.createNotificationChannel(channel)
        }
    }

    fun showBudgetAlertNotification(remainingBudget: Double, currencySymbol: String) {
        // Create an intent to open the app when notification is tapped
        val intent = Intent(context, MainActivity::class.java).apply {
            flags = Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
        }
        val pendingIntent: PendingIntent = PendingIntent.getActivity(
            context, 0, intent,
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) PendingIntent.FLAG_IMMUTABLE else 0
        )

        // Build the notification
        val builder = NotificationCompat.Builder(context, CHANNEL_ID)
            .setSmallIcon(R.drawable.ic_notification)
            .setContentTitle("Budget Alert")
            .setContentText("Your budget is running low! Only $currencySymbol${String.format("%,.2f",
remainingBudget)} left.")
            .setPriority(NotificationCompat.PRIORITY_DEFAULT)
            .setContentIntent(pendingIntent)
            .setAutoCancel(true) // Make the notification dismissible

        // Show the notification
        with(NotificationManagerCompat.from(context)) {
            notify(BUDGET_LOW_NOTIFICATION_ID, builder.build())
        }
    }

```

```
    fun showBudgetZeroNotification(currencySymbol: String) {
        // Create an intent to open the app when notification is tapped
        val intent = Intent(context, MainActivity::class.java).apply {
            flags = Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
        }
        val pendingIntent: PendingIntent = PendingIntent.getActivity(
            context, 0, intent,
            if (Build.VERSION.SDK_INT ≥ Build.VERSION_CODES.M) PendingIntent.FLAG_IMMUTABLE else 0
        )

        // Build the notification
        val builder = NotificationCompat.Builder(context, CHANNEL_ID)
            .setSmallIcon(R.drawable.ic_notification)
            .setContentTitle("Budget Limit Reached")
            .setContentText("You have reached your budget limit for this month!")
            .setPriority(NotificationCompat.PRIORITY_HIGH)
            .setContentIntent(pendingIntent)
            .setAutoCancel(true) // Make the notification dismissible

        // Show the notification
        with(NotificationManagerCompat.from(context)) {
            notify(BUDGET_LIMIT_NOTIFICATION_ID, builder.build())
        }
    }


    fun cancelBudgetAlertNotification() {
        with(NotificationManagerCompat.from(context)) {
            cancel(BUDGET_LOW_NOTIFICATION_ID)
        }
    }

    fun cancelBudgetZeroNotification() {
        with(NotificationManagerCompat.from(context)) {
            cancel(BUDGET_LIMIT_NOTIFICATION_ID)
        }
    }

    fun showBudgetResetNotification(budgetAmount: Double, currencySymbol: String) {
        // Create an intent to open the app when notification is tapped
        val intent = Intent(context, MainActivity::class.java).apply {
            flags = Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
        }
        val pendingIntent: PendingIntent = PendingIntent.getActivity(
            context, 0, intent,
            if (Build.VERSION.SDK_INT ≥ Build.VERSION_CODES.M) PendingIntent.FLAG_IMMUTABLE else 0
        )

        // Build the notification
        val builder = NotificationCompat.Builder(context, CHANNEL_ID)
            .setSmallIcon(R.drawable.ic_notification)
            .setContentTitle("Budget Reset")
            .setContentText("Your budget has been reset to ${currencySymbol}${String.format("%,.2f",
budgetAmount)}")
            .setPriority(NotificationCompat.PRIORITY_DEFAULT)
            .setContentIntent(pendingIntent)
            .setAutoCancel(true) // Make the notification dismissible

        // Show the notification
        with(NotificationManagerCompat.from(context)) {
            notify(BUDGET_RESET_NOTIFICATION_ID, builder.build())
        }
    }

    fun cancelAllBudgetNotifications() {
        with(NotificationManagerCompat.from(context)) {
            cancel(BUDGET_LOW_NOTIFICATION_ID)
            cancel(BUDGET_LIMIT_NOTIFICATION_ID)
            cancel(BUDGET_RESET_NOTIFICATION_ID)
        }
    }
}
```

JSONBackupGenerator Kt



```
        }
    }

    private fun createBackupJson(): JSONObject {
        val rootJson = JSONObject()

        // Add metadata
        val metadataJson = JSONObject()
        metadataJson.put("app", "Meloch")
        metadataJson.put("version", "1.0")
        metadataJson.put("timestamp", System.currentTimeMillis())
        metadataJson.put("date", SimpleDateFormat("yyyy-MM-dd HH:mm:ss",
    Locale.getDefault()).format(Date()))
        rootJson.put("metadata", metadataJson)

        // Add user info
        val userJson = JSONObject()
        val currentUser = userManager.getCurrentUser()
        if (currentUser != null) {
            userJson.put("username", currentUser.username)
            userJson.put("email", currentUser.email)
        } else {
            userJson.put("username", "Unknown")
            userJson.put("email", userManager.getCurrentUserEmail() ?: "Unknown")
        }
        rootJson.put("user", userJson)

        // Add financial data
        val financialJson = JSONObject()
        financialJson.put("totalBalance", preferencesManager.getTotalBalance())
        financialJson.put("pocketMoney", cardManager.getPocketMoney())
        financialJson.put("budgetLeft", preferencesManager.getBudgetLeft())
        financialJson.put("totalBudget", preferencesManager.getMonthlyBudget())

        // Add budget categories
        val budgetCategoriesJson = JSONObject()
        budgetCategoriesJson.put("entertainment", preferencesManager.getEntertainmentBudget())
        budgetCategoriesJson.put("food", preferencesManager.getFoodBudget())
        budgetCategoriesJson.put("transport", preferencesManager.getTransportBudget())
        budgetCategoriesJson.put("lifestyle", preferencesManager.getLifestyleBudget())
        financialJson.put("budgetCategories", budgetCategoriesJson)

        rootJson.put("financial", financialJson)

        // Add cards
        val cardsJson = JSONArray()
        val cards = cardManager.getCards()
        for (card in cards) {
            val cardJson = JSONObject()
            cardJson.put("id", card.id)
            cardJson.put("cardNumber", card.cardNumber)
            cardJson.put("cardholderName", card.cardholderName)
            cardJson.put("expiryMonth", card.expiryMonth)
            cardJson.put("expiryYear", card.expiryYear)
            cardJson.put("expiryDate", card.getExpiryDate())
            cardJson.put("cvv", card.cvv)
            cardJson.put("type", card.type.name)
            cardJson.put("balance", card.balance)
            cardsJson.put(cardJson)
        }
        rootJson.put("cards", cardsJson)

        // Add transactions
        val transactionsJson = JSONArray()
        val transactions = preferencesManager.getTransactions()
        for (transaction in transactions) {
            val transactionJson = JSONObject()
            transactionJson.put("id", transaction.id)
            transactionJson.put("amount", transaction.amount)
            transactionJson.put("category", transaction.category.name)
            transactionJson.put("type", transaction.type.name)
            transactionJson.put("date", transaction.date.time)
            transactionsJson.put(transactionJson)
        }
        rootJson.put("transactions", transactionsJson)

        return rootJson
    }
}
```