CSE305 Software Engineering – Sprint 2 Report

Student: Başar Orhanbulucu
Student ID: 221805031
Project: My Budget Flow (Budget & Cash Flow Manager)
Sprint: #2 (Budgeting Engine)

1. Plan for Sprint 2

This section summarizes the Sprint 2 goals and their current status, as defined in the "Agile Process Model" document.

- Sprint Goal: To implement the core "Budgeting Engine" where users can log income/expenses, view them on a calendar, automate recurring payments, and track their financial status via the Dashboard.
- Date Range: Dec 14 - Dec 28
- Status: ✅ Completed

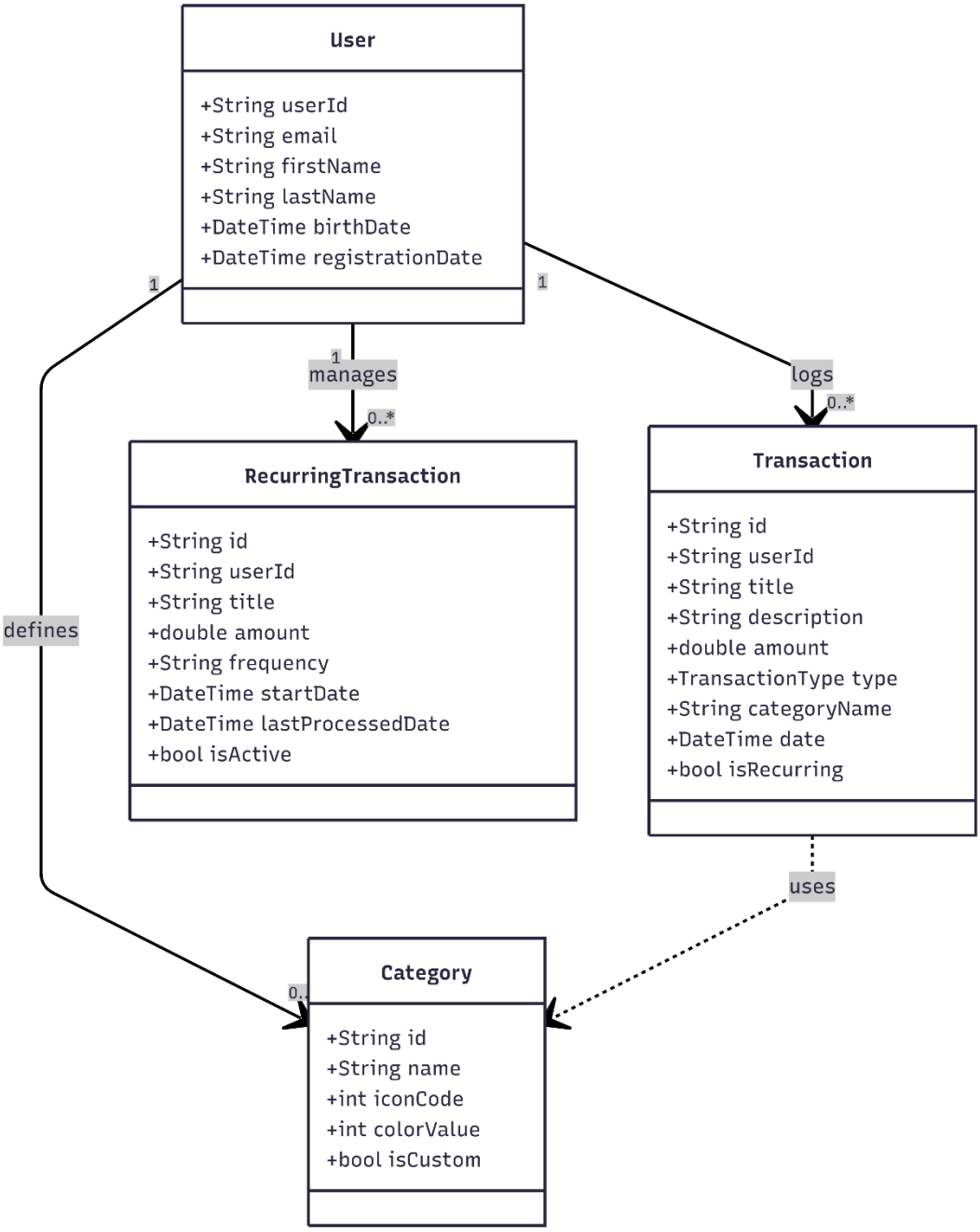
Sprint Backlog Scope:

User Story	Task	Description	Status
US 3.1	Transaction CRUD	Implementation of AddTransactionScreen, TransactionRepository and Firestore integration for adding Income/Expense.	Done
US 3.2	Transaction Listing & Calendar	Development of TransactionsScreen for history list and CalendarScreen for monthly view.	Done
US 4.1	Recurring Transactions	Logic for RecurringTransactionModel and auto-generation of fixed monthly payments.	Done
US 4.2	Dashboard & Analytics	DashboardProvider implementation to calculate Net Balance, Total Income/Expense and Category Distribution.	Done
Tech	Repository Pattern	Refactoring data access layer using TransactionRepository and CategoryRepository.	Done

2. Domain Class Diagram (Sprint 2 Coverage)

In Sprint 2, the domain model has been significantly expanded. The Transaction, RecurringTransaction, and Category entities have been implemented and linked to the User.

- Transaction: Represents a single financial record.
- RecurringTransaction: Represents a template for automated payments.
- Category: Represents transaction types (System defaults + User defined).

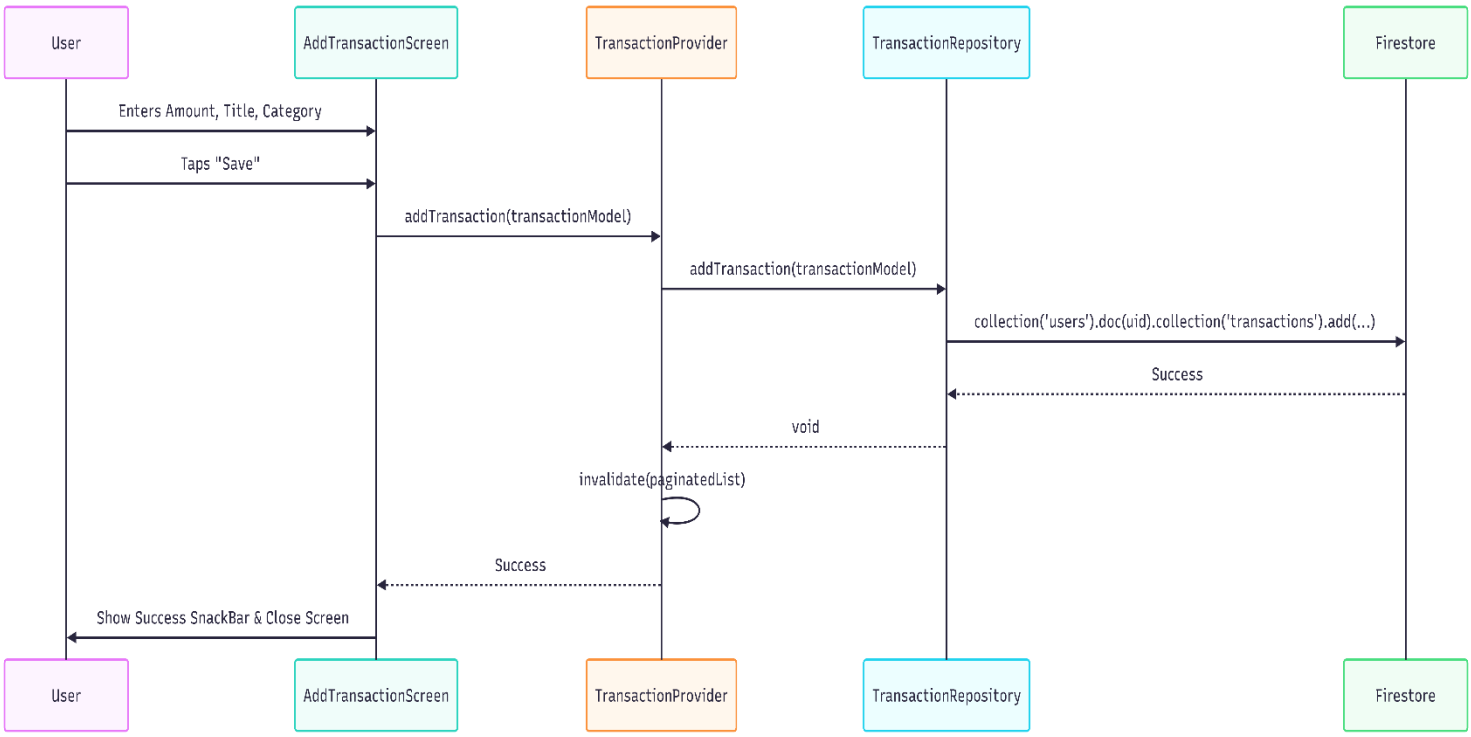


3. Sequence Diagrams

Below are the technical sequence diagrams for the critical operations developed in Sprint 2:

Scenario 1: Adding a New Transaction

The process of a user adding a new expense via AddTransactionScreen and writing data to Firestore via Repository pattern.



Actors and Components:

- User: The person inputting financial data.
- AddTransactionScreen: The UI component with form fields and category selector.
- TransactionProvider: The Riverpod state manager that handles UI-Repository communication.
- TransactionRepository: The data layer responsible for Firestore operations.
- Firestore: The cloud database storing transaction records.

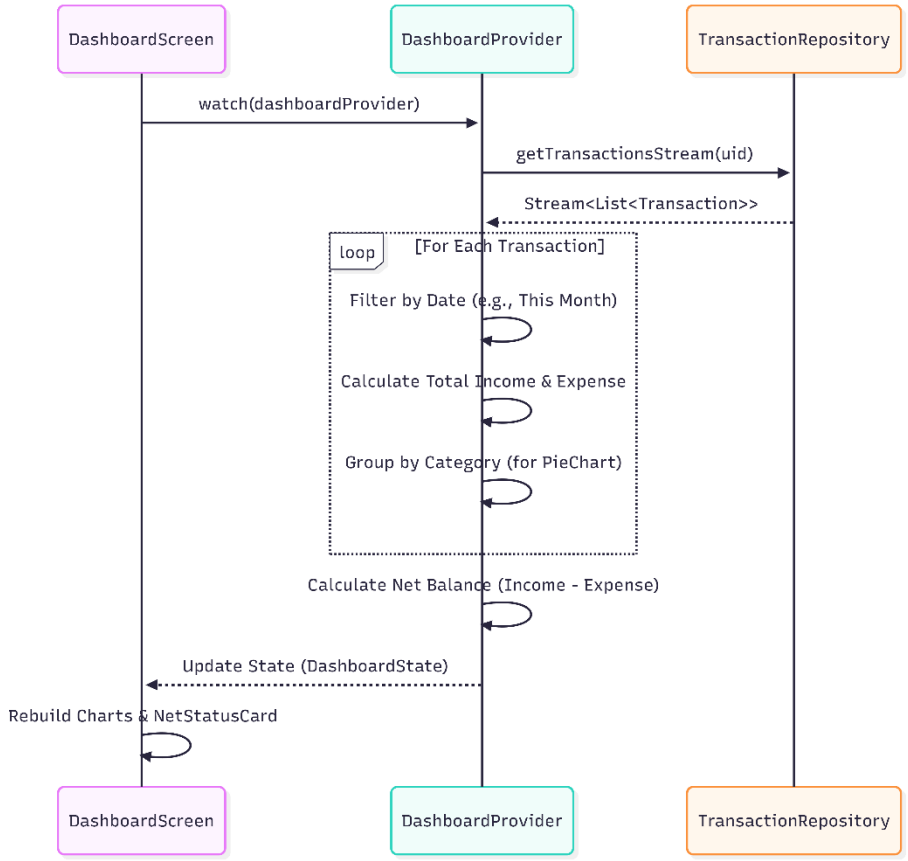
Step-by-Step Process:

1. Input: The User enters the amount, selects a category, picks a date, and clicks "Save".
2. Validation: AddTransactionScreen checks if the amount is valid and required fields are filled.

3. State Call: The UI calls the addTransaction method in TransactionProvider.
4. Data Persistence: The Provider delegates the task to TransactionRepository.
5. Database Write: The Repository creates a JSON-like map from the model and writes it to the transactions sub-collection in Firestore.
6. State Invalidation: Upon success, the Provider invalidates the cached transaction list to force a refresh.
7. Feedback: The UI displays a success SnackBar and navigates back to the previous screen.

Scenario 2: Dashboard Calculation Logic

How the DashboardProvider calculates net balance, totals, and chart data in real-time when the user opens the app.



Actors and Components:

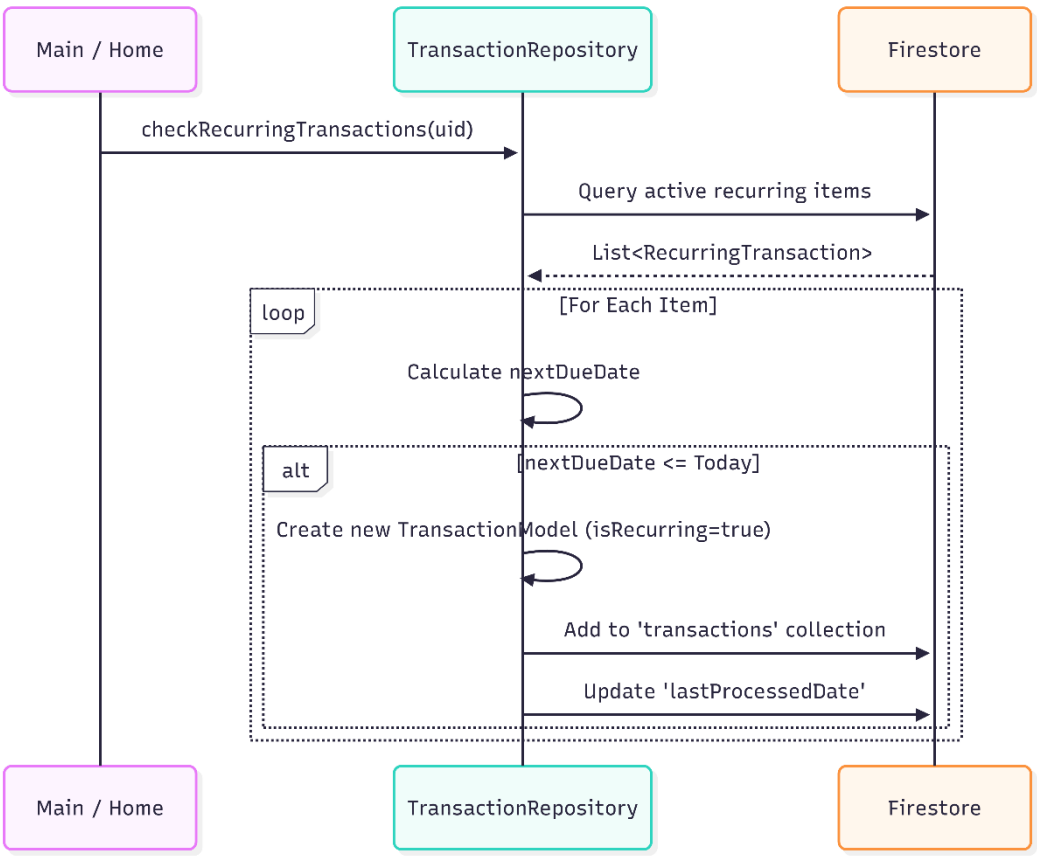
- DashboardScreen: The main home screen displaying charts and summaries.
- DashboardProvider: The logic layer that processes raw data into meaningful statistics.
- TransactionRepository: The source of the transaction data stream.

Step-by-Step Process:

1. Subscription: DashboardScreen watches the dashboardProvider on load.
2. Data Fetching: The Provider requests a real-time stream of transactions from the Repository.
3. Filtering: The Provider filters the raw list based on the selected date range (e.g., "This Month").
4. Calculation Loop: The Provider iterates through the filtered list:
    - Adds amounts to totalIncome or totalExpense based on TransactionType.
    - Groups expenses by categoryName to prepare data for the "Spending Distribution" chart.
5. Net Balance: Net Balance is calculated as Total Income - Total Expense.
6. UI Update: The DashboardState is updated with new values, triggering a rebuild of the UI widgets (Charts, Cards).

Scenario 3: Recurring Transaction Automation

The logic runs at app startup to check if any recurring payment is due.



Actors and Components:

- BudgetScreen (Home): The main UI screen that triggers the check upon initialization (initState).
- TransactionRepository: The service containing the business logic for processing recurring items.
- Firestore: The database storing both the templates (recurring_transactions) and the generated logs (transactions).

Step-by-Step Process:

1. Trigger: The User opens the application. The BudgetScreen initializes and calls the checkRecurringTransactions(uid) function in TransactionRepository.
2. Query: The Repository queries Firestore for all documents in the recurring_transactions collection where the isActive field is true.
3. Iteration: The system iterates through each active recurring template found.
4. Calculation: For each item, the RecurringTransactionModel logic calculates the nextDueDate using the lastProcessedDate and the defined frequency (e.g., Monthly).
5. Decision Block (Catch-up Logic):
    - Condition: The system checks if (nextDueDate <= DateTime.now()).
    - If Due:
        - Create Transaction: A new TransactionModel is instantiated with isRecurring = true, using the details (amount, category) from the template.
        - Batch Write: The new transaction is added to a Firestore WriteBatch to ensure atomicity.
        - Update Template: The lastProcessedDate of the recurring template is updated to the nextDueDate.
        - Loop: If multiple periods were missed (e.g., user hasn't opened the app for 2 months), the logic loops until nextDueDate is in the future.
6. If Not Due: The item is skipped.
7. Commit: The Repository commits the batch operation to Firestore, saving all new transactions and updates simultaneously.
8. Result: The UI streams update automatically, and the User sees the new automated transactions in their history list without manual input.

4. Trello Board Management

https://trello.com/invite/b/6921eb3ffb6fe09b90661536/ATTIa90b3944b9023750cc0ed893be35c9f093256CBD/my-budget-flow

5. PSI (Potentially Shippable Increment) - Sprint 2 Review

The target for Sprint 2 was to deliver a fully working "Budgeting Engine". This target has been met successfully. The increment includes:

- Transaction Management: Users can perform full CRUD operations for Income and Expenses.
- Real-time Dashboard: A dashboard that calculates "Net Balance" instantly based on the transaction history.
- Recurring Engine: An automated system that generates transactions for fixed monthly payments (Rent, Bills) without user intervention.
- Calendar Module: A visual representation of financial activity on a monthly calendar view.

Screenshots of the Increment: