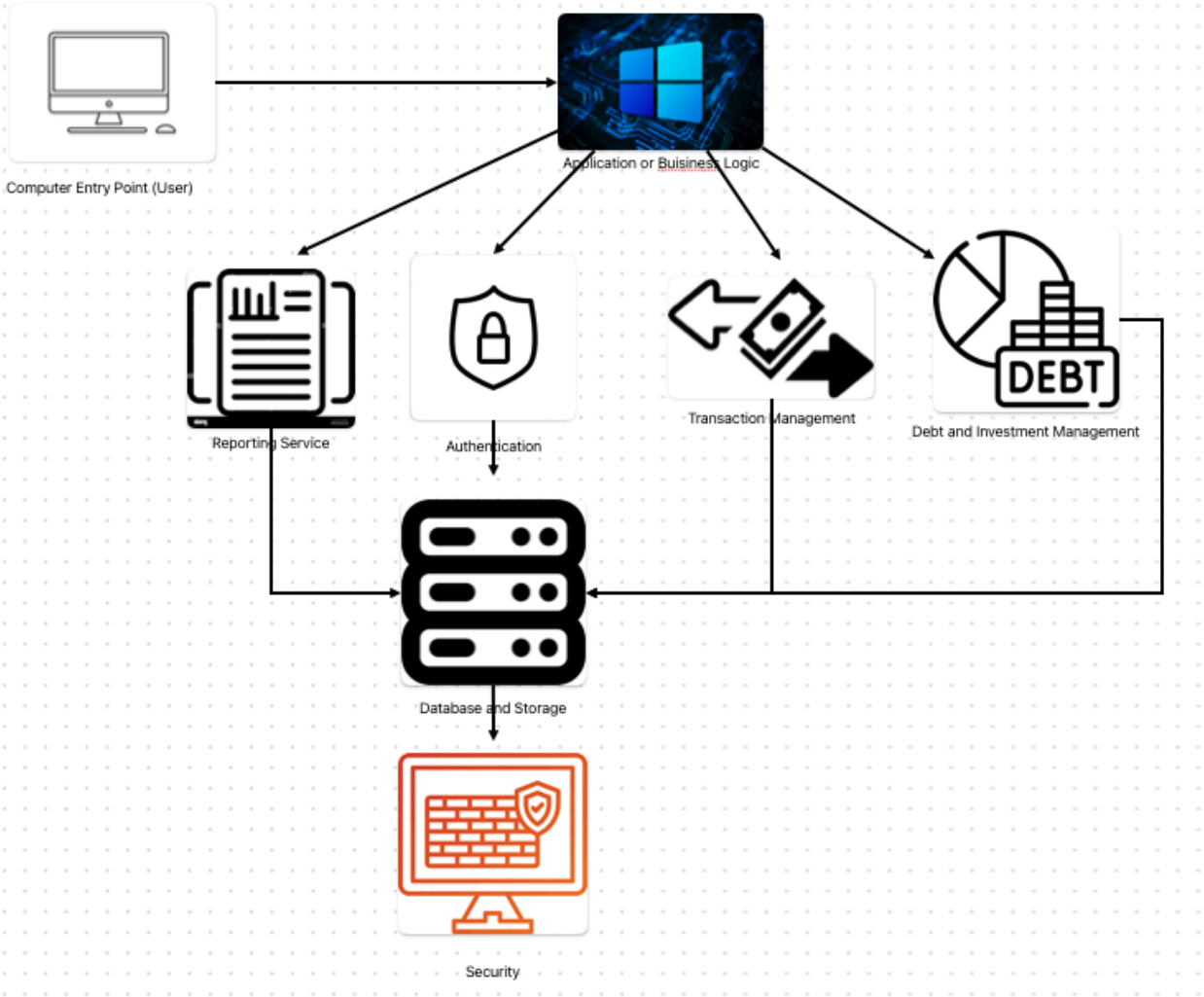
Software title: Personal Finance Assistant

Team members: Duc Nguyen, Adrian Balingit

Brief overview of the system: The Personal Finance Assistant is a comprehensive software designed to help individuals, especially the young, achieve their financial goals by helping them manage their finances effectively. It provides users with tools for budgeting including tracking income and spending to aid in financial management. Moreover, the software also offers tools for managing debt.

I. Architectural diagram of all major components



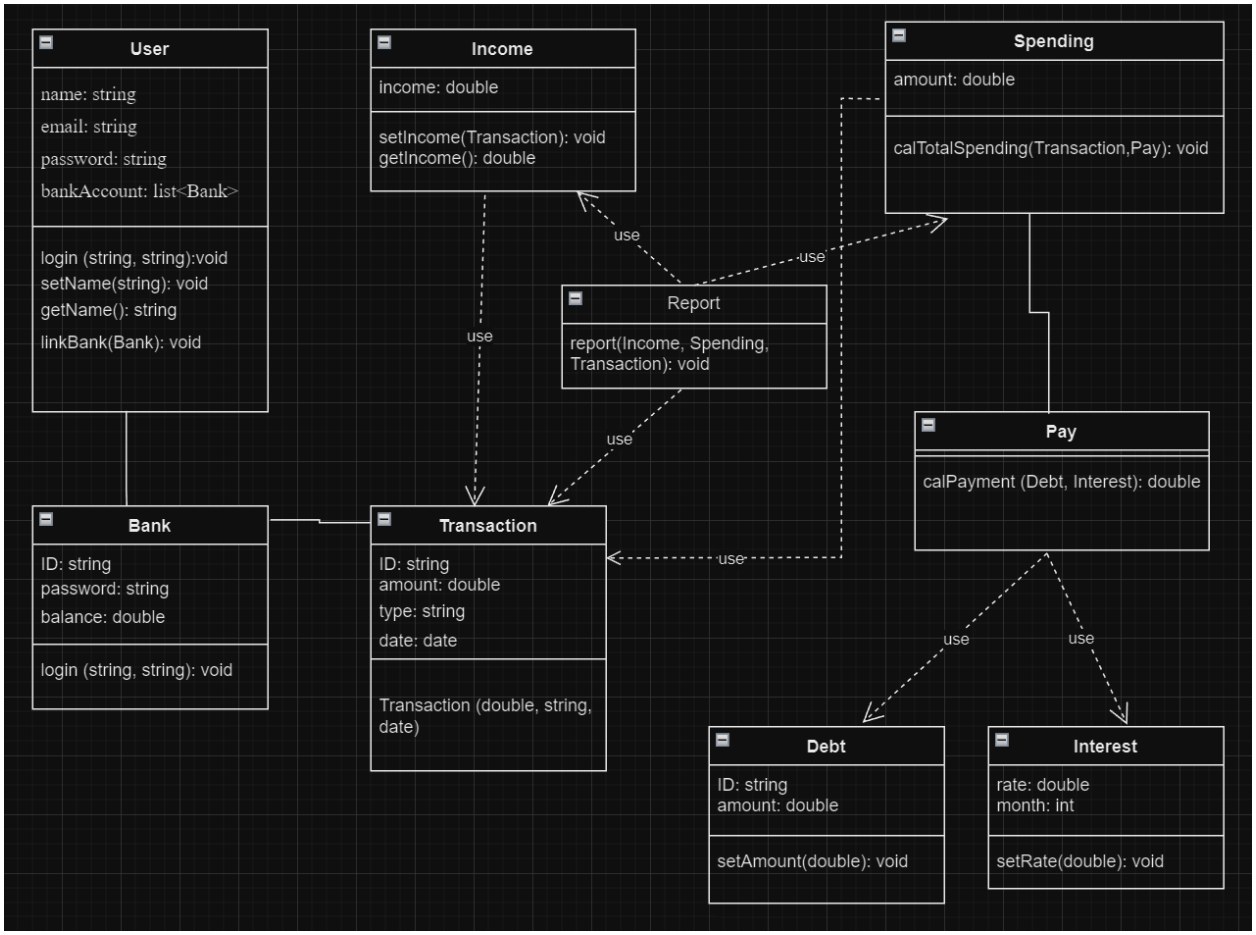
Description of SWA Diagram

1. User Interface (UI)
    - Will include all the graphical interface and user interaction mechanisms. This will include mobile applications, web pages, and the interactive tools for budgeting, financial planning, and spendings tracking.
    - Connects and communicates directly with the Application/Business Logic Layer to take in user input and display data.
2. Application/Business Logic
    - Manages user login, user session, and security

- Connects with Data Access Layer for data operations such as reporting services, authentication, transaction management, and Debt and investment management

3. Reporting Service
    - Generates detailed financial reports for the user that also includes insights.
    - Connects to the Database and Storage
4. Authentication
    - Manages the user login and security checks
    - Connects to the database and storage
5. Transaction management
    - Tracks, categorizes, and stores user transactions
    - Connects to the database and storage
6. Debt and Investment Management
    - Tracks debt and gives investment advice
7. Database and Storage
    - Consists of databases and storage systems responsible for persisting user data, transaction record, and financial plans.
    - Connects to the last layer of security
8. Security
    - All security measures for sensitive financial data that will include encryption, authentication protocols, and authorization checks.

II. UML Class Diagram: User Class, Bank Class, Transaction Class, Income Class, Debt Class, Interest Class, Pay Class, Spending Class, Report Class.



Description of classes:

User Class: Represents users of the software

Bank Class: Represents the bank linked by the user

Transaction Class: Represents all the transactions that are happening in the bank

Income Class: Represents the income based on transactions

Debt Class: Represents the amount of the user's debt

Interest Class: Represents the debt's interest rate and the number of months needed to pay

Pay Class: Represents the amount the user needs to pay based on the amount of debt and interest

Spending Class: Represents the total spending including transactions and debt pay

Report Class: Showing report of user spending and income.

Description of attributes:

1. User Class:

- name: a name chosen by the user
- email: email address of the user for communication and authentication
- password: password chosen by the user for authentication
- bankAccount: a list of linked bank accounts of the user

2. Bank Class:

- ID: the user's bank account ID for authentication to link the bank account
- password: user's bank password for authentication to link the bank account
- balance: the balance user has in their bank account

3. Transaction Class:

- amount: the amount of money in each transaction
- type: type of transaction
- date: the date the transaction occurred

4. Income Class:

- income: the amount of user income determined by the positive value transaction

5.. Spending Class:

- amount: the amount of user spending determined by the negative value transaction plus debt pay

6. Debt Class:

- amount: the amount of debt of the user

7. Interest Class:

- rate: the interest rate of the debt
- month: the amount of month needed to pay off the debt

8. Pay Class: no attributes

9. Report Class: no attributes

Description of operations

1. User Class:

a. login(string email, string password): void

- Description: Authenticates the user to establish a secure connection to access their account.
- Parameter: email and password (type: string for both) provided by the user to connect to their account.
- Return: void.

b. setName(string name): void

- Description: Set the name provided by the user for their account.
- Parameter: name (type: string) provided by the user.
- Return: void.

c. getName(): string

- Description: Get the user name to display it.
- Parameter: none.
- Return: the user name (type string)

d. linkBank(Bank account): void

- Description: Links the user's bank account to their personal finance assistant account to track their transactions within their bank account.
- Parameter: account (type: bank) the user's bank account linked to their personal finance assistant account.
- Return: void

2. Bank Class:

login(string email, string password): void

- Description: Authenticates the user to establish a secure connection to link their bank account to access their balance and transaction.
- Parameter: email and password (type: string for both) of the bank provided by the user to link to their account.
- Return: void

3. Transaction Class:

(Updated) Transaction(double amount, string type, date date): void

- Description: get transactions from the linked bank account and add them to the user's personal finance assistant account to track their income and spending.
- Parameter: basic information of the transaction including amount of money (double), type of transaction (string), and the date of transaction (date).
- Return: void.

4. Income Class:

a. setIncome(Transaction t): void

- Description: Get the positive value transactions of the linked bank account and set it as income.

- Parameter: t (type Transaction) the transactions from the linked bank account and only accept the positive amount.
- Return: void.

b. getIncome(): double

- Description: Get the income for further use.
- Parameter: none.
- Return: user income (type: double).

5. Spending Class:

calTotalSpending(Transaction t, Pay p): void

- Description: calculate the total spending including the debt payment and assign it to the amount
- Parameter: the transactions (type Transaction) from the linked bank account, only accept the negative amount and the debt payment (type Pay)
- Return: void

6. Debt Class:

setAmount(double amount): void

- Description: set the debt amount
- Parameter: amount (type double) amount of debt provided by the user
- Return: void

7. Interest Class:

setRate(double rate): void

- Description: set the debt interest rate
- Parameter: rate (type double) interest rate provided by the user

- Return: void

8. Pay Class:

calPayment(Debt d, Interest i): double

- Description: calculate the debt payment per month based on the amount, rate, and how many months to pay.
- Parameter: d (type Debt) amount of debt, i (type Interest) interest rate with the month
- Return: the payment per month (type double)

9. Report Class: no attributes

report(Income i, Spending s, Transaction t): void

- Description: Report all the information about user finance including how much they earn and spend each month, and also their detailed information in the transactions.
- Parameter: i (type Income) amount of income, s (type Spending) , and t (type Transaction)
- Return: void

III. Verification Test Plan

UPDATE: The addTransaction function has been changed to the constructor, now every transaction will become a unique transaction block for easier management.

Unit test set 1: Addition of Transaction Function

The purpose of this test is to verify that the transaction function operates properly by manually getting new input transaction data with different kinds of scenarios into the function. It should accurately handle both positive and negative values. Moreover, it should display the appropriate

amount, type, and date of the transaction. Importantly, transactions should not be allowed to have future dates.

Input 1: the new transaction with an income of 50.00, type test, date 03/19/2024

Transaction t1(50.00, “test”, 03/19/2024)

Print t1.amount + t1.type + t1.date

Expected output: 50.00 test 03/19/2024

Input 2: the new transaction with an income of -10.00 type test, date 03/19/2024

Transaction t2(-10.00, “test”, 03/19/2024)

Print t2.amount + t2.type + t2.date;

Expected output: -10.00 test 03/19/2024

Input 3: The new transaction with an income of 50.00, type test, date 03/19/2029

Transaction t3(50.00, “test”, 03/19/2029)

Print t3.amount + t3.type +t3.date;

Expected output: Error message indicating the invalid date.

Unit test set 2: User Authentication Function

The purpose of this authentication mechanism is to correctly handle user logins, including the validation of email and password, and appropriately restrict access with incorrect credentials

Test 1: Valid Login

Input: Correct email and password

Expected Output: Successful login message and redirection to the user dashboard.

Test 2: Invalid Password

Input: Correct email address but incorrect password

Expected Output: Error message indicating incorrect password.

Test 3: Invalid Email

Input: Incorrect email and any password

Expected Output: Error message indicating user does not exist

Test 4: Empty Fields

Input: Empty email and password fields

Expected Output: Error message indicating fields cannot be empty

Integration test set 1: Transaction Management

The purpose of this test is to verify that the income function and debt function are working properly together with the transaction and the report function by adding the transaction with different kinds of scenarios. Transactions with positive amounts should increase income, while those with negative amounts should increase debt. The report function should then show the correct income and spending amounts along with a list of transactions.

Constraint: This test will assume the user has zero debt since we only want to test how the Transactions, Income, Spending, and Report functions work together.

Input 1: The new transaction with an income of 50.00, type work, date 03/19/2024

Transaction t1(50.00, “test”, 03/19/2024)

Income i

Spending s

i.setIncome(t1)

s.setIncome(t1)

Print i.income

Print s.amount

Report()

Expected output:

50.00

0.00

Income: 50.00, Spending: 0.00

List of transactions: test type in 03/19/2024 amount 50.00

Input 2: The new transaction with an income of -50.00, type test, date 03/19/2024

Transaction t2(-50.00, “test”, 03/19/2024)

i.setIncome(t2)

s.setIncome(t2)

Print i.income

Print s.amount

Report()

Expected output:

50.00

50.00

Income: 50.00, Spending: 50.00

List of transactions: test type in 03/19/2024 amount 50.00

test type in 03/19/2024 amount -50.00

Input 3: The new transaction with an income of -50.00, type test, date 03/19/2024

Transaction t3(-50.00, “test”, 03/19/2024)

i.setIncome(t3)

s.setIncome(t3)

Print i.income

Print s.amount

Report()

Expected output:

50.00

100.00

Income: 50.00, Spending: 100.00

List of transactions: test type in 03/19/2024 amount 50.00

test type in 03/19/2024 amount -50.00

test type in 03/19/2024 amount -50.00

Input 4: The new transaction with an income of -50.00, type test, date 03/19/2029

Transaction t4(-50.00, “test”, 03/19/2029)

i.setIncome(t4)

s.setIncome(t4)

Print i.income

Print s.amount

Report()

Expected output: Error message indicating that t4 does not exist because it’s already been validated in the transaction function.

Integration test set 2:

Objective: To verify that the system properly integrates the functionality of linking bank accounts with the fetching of transactions from linked accounts, ensuring transactions are accurately reflected in the user's financial overview.

Test 1: Link Bank Account and Fetch Transactions

Input: User successfully links a bank account

Procedure: Upon linking, the system automatically fetches the latest transactions

Expected Output: A list of recent transactions from the linked bank account is displayed.

Test 2: Link Invalid Bank Account

Input: User attempts to link an invalid or unsupported bank account.

Expected Output: Error message indicating failure to link account.

Test 3: Unlink Bank Account

Input: User decides to unlink a previously linked bank account

Procedure: After unlinking, the system should no longer fetch transactions from account

Expected Output: Confirmation of account unlinking and cessation of new transactions from the account being displayed.

System test set 1: Overall Functionality

Customers should be able to log in with their email and password credentials, enabling them to link their bank accounts. Upon successful linkage, there is a list of their linked bank account for customers to review. The income and debt function should correctly retrieve the info from the transaction function and sort it into the corresponding function. Customer can manually input their debt and the software will automatically calculate the monthly payment, directing it to the

spending function. Finally, customer can view a comprehensive report detailing how much their income and how much they are spending with a list of transactions to prevent potential fraudulent activity.

System test set 2: Financial Planning and Analysis

Objective: To assess the system's overall capability to assist users in financial planning and analysis, including setting financial goals, tracking spending against goals, and generating insightful reports.

Test 1: Setting and Tracking Financial Goals

Input: User sets a monthly saving goal.

Procedure: User logs transactions over the month.

Expected Output: A report showing progress towards the saving goal, indicating whether the user is on track, under, or over the goal.

Test 2: Generating Financial Health Report

Input: User requests a financial health report.

Procedure: The system analyzes transactions, spending patterns, and saving goals.

Expected Output: A comprehensive financial health report that includes insights on spending habits, areas for improvement, and progress toward financial goals.

Test 3: Detecting and Alerting on Unusual Spending

Input: An unusually large transaction is recorded.

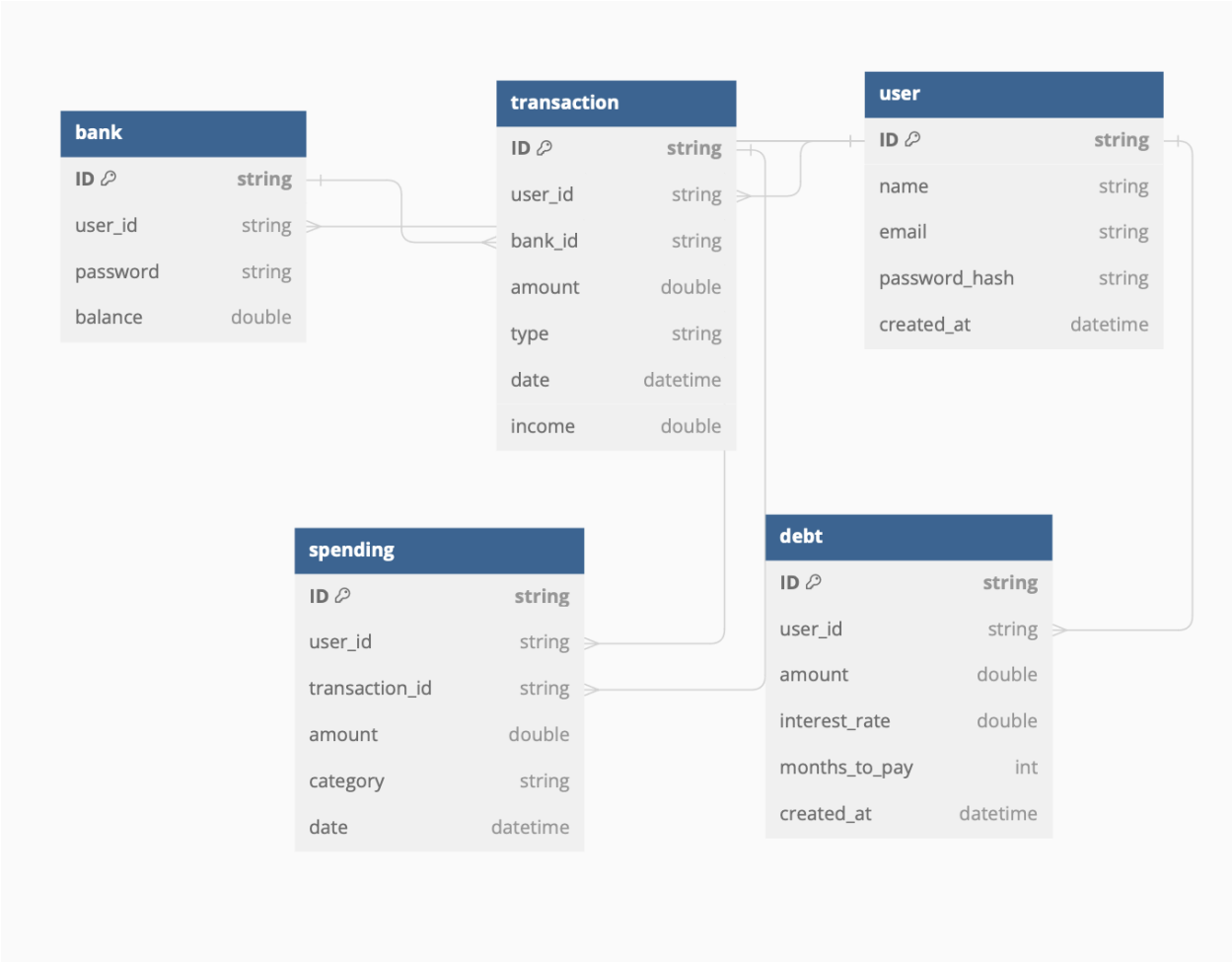
Procedure: The system compares this transaction against typical spending patterns.

Expected Output: An alert is generated for the user to review the transaction, helping detect potential fraudulent activity or encourage reflection on spending habits.

IV. Data Management Strategy

Update: Adding a new variable called ID for both transaction and debt class to serve as the primary key and distinct identification of each record.

Data management strategy: SQL diagram



Description:

Transaction Database: Stores information related to financial transactions, amount of money in each transaction, type of transaction, date, and ID which will become the primary key because it is the unique identifier for each transaction. Moreover, spending will become one column in the transaction table which will store only a positive value of the amount column.

Bank Database: Stores information about bank accounts, including ID (Primary key), password,

and total balance in that account. Again, the account ID will serve as the unique identifier for each account.

User Database:

Maintains user profile information, including a unique ID, name, email, and encrypted password. This database is fundamental to user authentication and personalization o the finance management experience.

Debt Database:

Keeps track of all user debts, detailed by a unique ID, the amount owed, interest rate, repayment period (months_to_pay), and the date the debt was recorded. Each debt entry is tied back to a user, facilitating personalized debt

Spending Database: Records user spending activities, categorized by type and linked to specific transactions where applicable. Each spending record is associated with a user, providing insights into spending habits and areas for budget optimization.

Relationship

Transaction Database vs Bank Database: In the transaction table, there is a column named bank_id, which serves as a foreign key referencing the ID column of the bank table.

This foreign key will ensure that each record of the transaction table is associated with a specific bank from the bank table in case customers want to link multiple bank accounts.

We believe the best relationship will be one-to-many between the bank table (one) and the transaction table (many) since one bank account can have multiple records of transactions.

Discussion

We decided to use SQL for our data management strategy and split it into 5 databases which are the 5 main components of our system: transaction, bank, user, debt, and spending. Splitting data into multiple databases can help increase modularity. Having separate databases allows for better scalability, as we can scale each database independently. However, this could also add complexity to the system, and require more effort to ensure data consistency across multiple databases. That's why we believe 5 databases are enough to achieve all the pros from above, which is particularly advantageous for large-scale systems without making any unnecessary tables causing the cons. For example, the income class does not need a separate table as it can be one column in the transaction table.

One alternative to our current design is to combine all the data into a single table. While this approach could simplify the data system in general. In the long run, it might lead to confusion due to the potential of a large volume of transactions occurring within a short period. Thus, this could result in performance issues.

Partitioning of tasks:

Feb 28: Software title and Brief overview by Duc Nguyen

March 7: UML diagram and description by Duc Nguyen

March 7, SWA diagram and description by Adrian Balingit

March 19: First test set of Unit, Integration, and System by Duc Nguyen

March 22: Second test set of Unit, Integration, and System by Adrian Balingit

April 11: Choosing SQL data management strategy, update UML diagram, doing bank database, transaction database included the income and discussion

Team member responsibilities

Duc Nguyen - Software title and brief description

Duc Nguyen - UML Diagram description, first test set of Unit, Integration, and System

Duc Nguyen - update UML diagram, doing bank database, transaction database included the income and discussion

Adrian Balingit - SWA diagram and description, second test set of Unit, Integration, and System