Assignment 2

Software Design for MyBankUML App

COMP 354
Introduction to Software Engineering

Mykyta Onipchenko (40281057)
Bogdan Ivan (40283137)
Mate Barabas (40285661)
Veronika Pontolillo (40286664)
Kimberley Kam Sing (40319425)
Gorden Quach (40263250)

Due: October 26th, 2025

Table of Contents

(11) Design Strategies

Since this project is about banking, in addition to core object-oriented principles such as abstraction and encapsulation, we decided that we should focus the design on ensuring operation consistency and data correctness.

To follow object-oriented principles, we designed the components in a layered manner from bottom to top. We designed each component to be as standalone as possible and defined a public interface, making sure the components are well abstracted (the complexity is hidden through methods) and encapsulated (the internal state is not exposed, setters and mutators are used sparingly).

To ensure data consistency, we designed public interfaces to only contain atomic operations, disallowing partial updates. Furthermore, we chose appropriate data types for every property. Notably, we use **BigDecimal** instead of **float** to represent monetary amounts since they do not lose precision. Finally, we will use an operation queue that, along with database-level transactions, will ensure that all banking operations are valid and atomic.

(12) Architectural Level Design

The following table gives an overview of different classes defined in our software. They are designed with high cohesion, low coupling, abstraction, encapsulation, operation consistency, and data correctness as a priority.

Class	Description
BankDB	Serves as a high-level API to interact with the banking information (query accounts, transfer money, lock accounts, etc).
Bank	Data object storing bank information. Provides access to branches.
Branch	Data object storing bank branch information. Provides access to customers.
Customer	Data object storing customer information. Provides access to accounts.
Operation	Interface providing a database modification.
LockOperation	Class storing references to an account to be locked/unlocked and providing a method to perform this operation.
TransactionOperation	Class storing references to accounts and amount to be transferred, and providing a method to perform this operation.
Account	Abstract data object storing account information. Provides access to transactions.
CheckingAccount	Class representing a checking account inheriting properties from **Account**.
SavingsAccount	Class representing a savings account inheriting properties from **Account**.
CreditAccount	Class representing a credit account inheriting properties from **Account**.
Transaction	Data object storing a performed transaction operation (i.e. correlates to a database entry).
TransactionInfo	Data object storing transaction information.

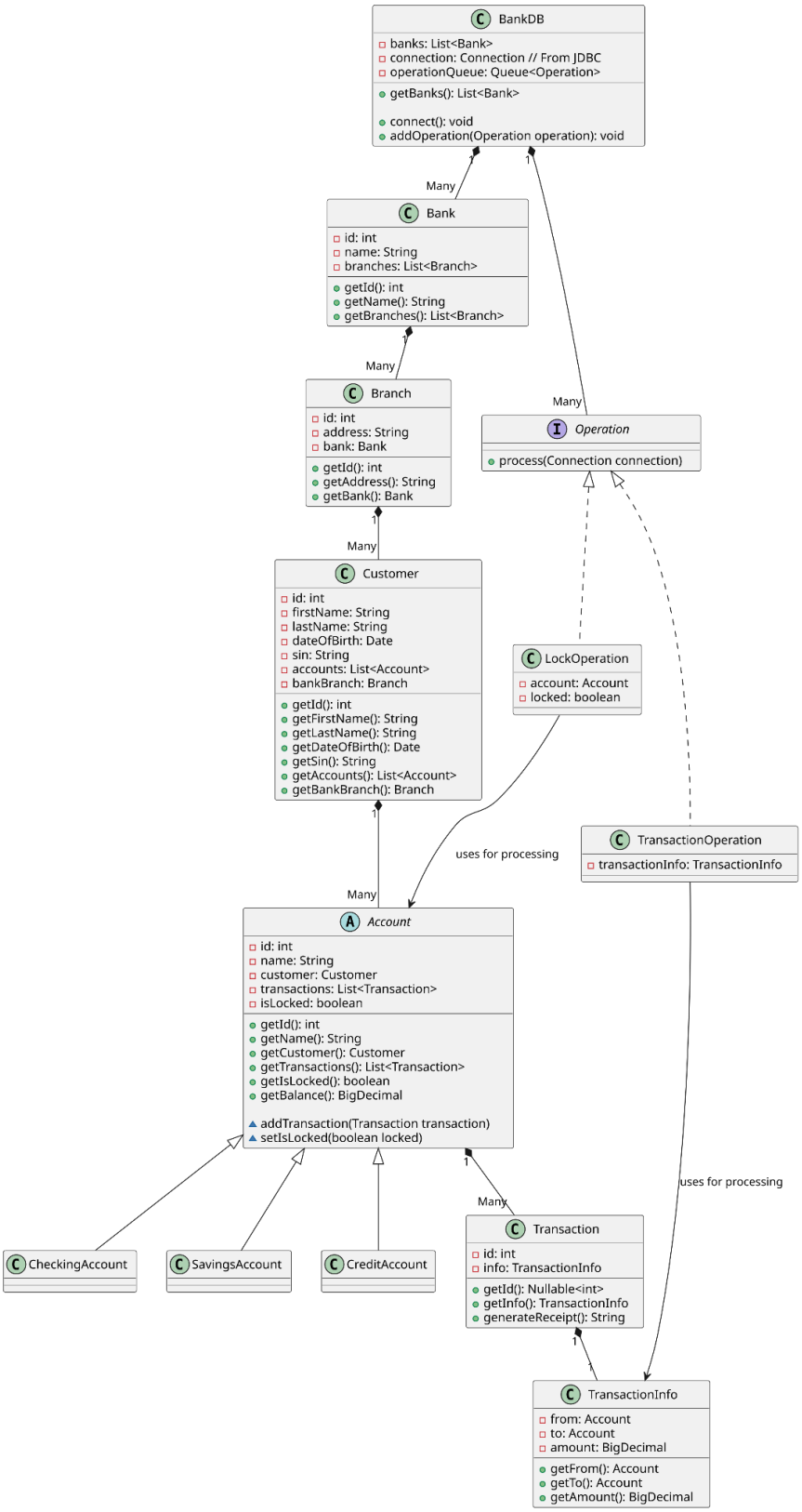

Figure 1: UML Class Diagram

These classes can also be represented as these components:

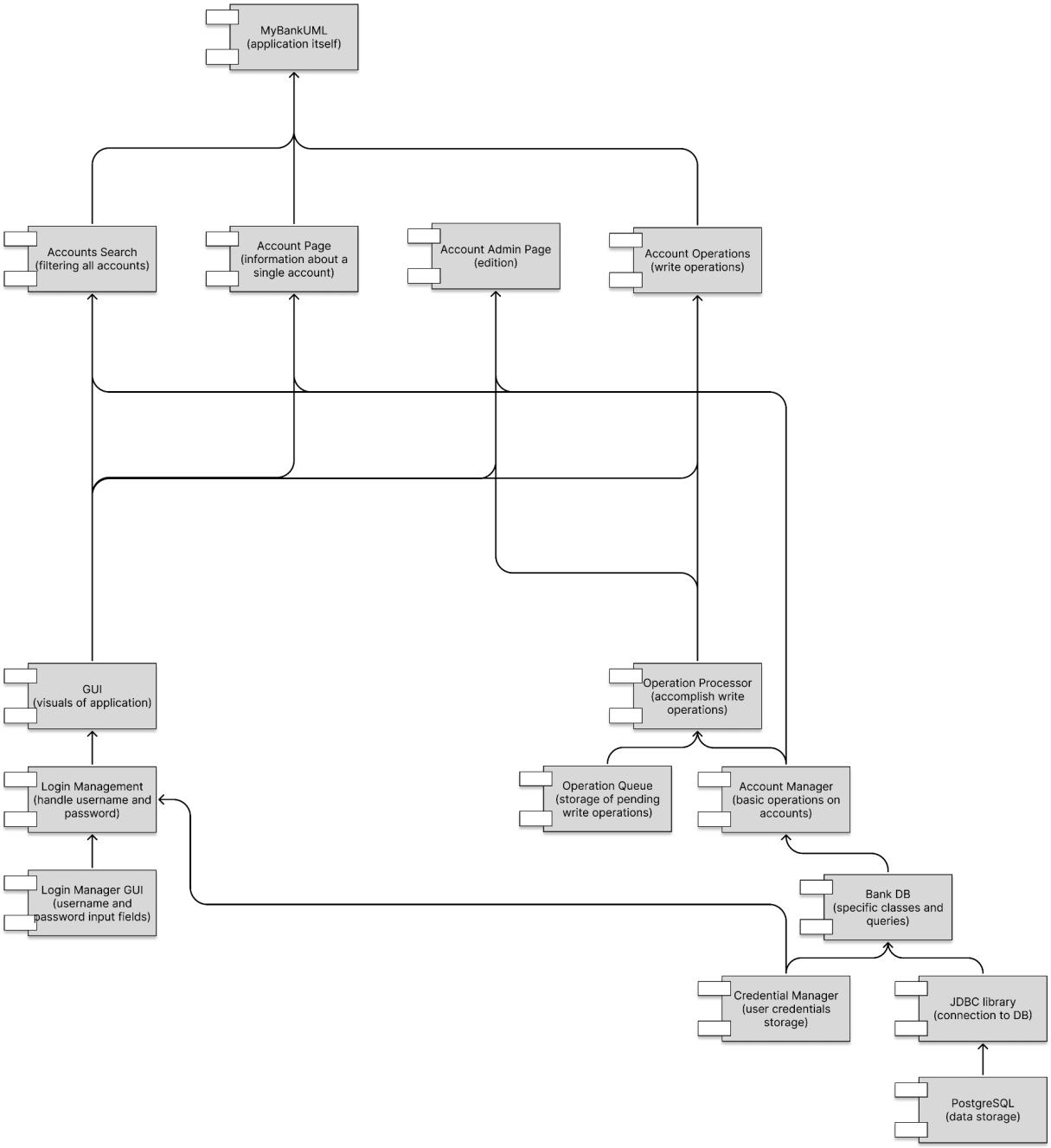

Figure 2: Application Components

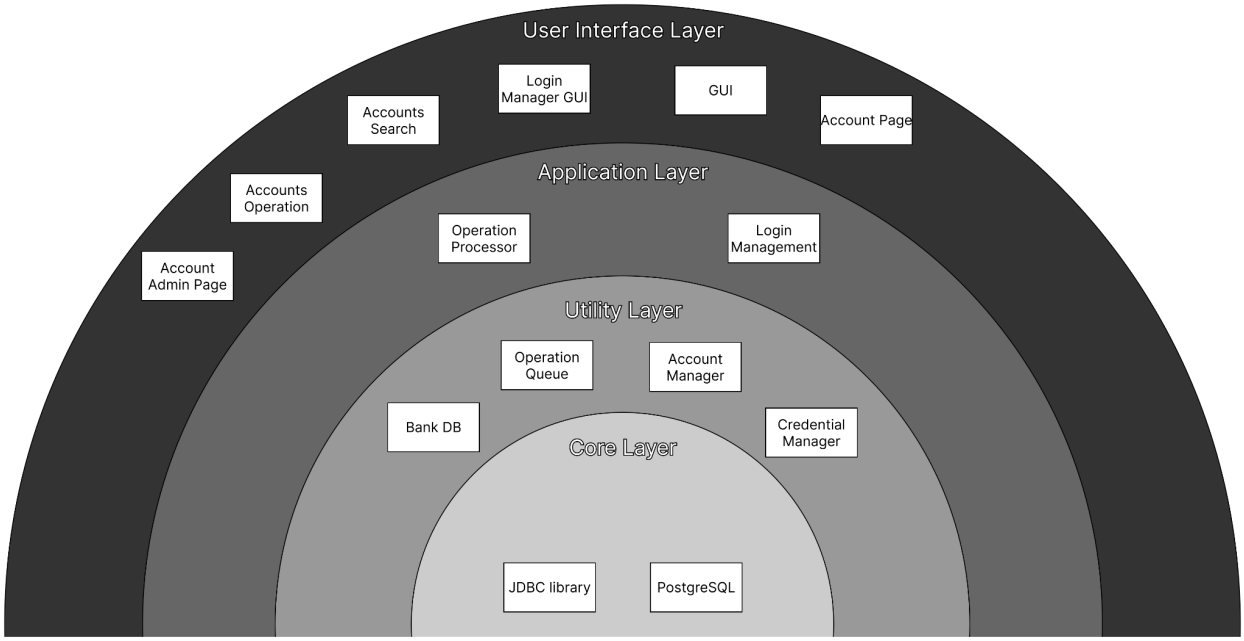

Figure 3: Application Components Represented as Layers

(13) Component Level Design

BankDB

1. **High Cohesion:** Encompasses all operations related to connecting to the API, managing stored bank information, and adding new operations.
2. **Low Coupling:** Does not depend directly on internal implementation of other classes like Bank, Branch, Customer.
3. **Definition:** Represents the database management component that maintains connections and operations related to the banks stored.
4. **Responsibilities:** Manage database connections, store and retrieve bank information, add database operations.
5. **Composition:**
    - **Attributes:**
        - `banks: List of banks managed`
        - `connection: Database connection object`
        - `operationQueue: Queued operations to execute`
    - **Functions:**
        - `getBanks: Returns list of banks`
        - `connect: Initializes connection to database, populates banks array`
        - `addOperation: queues an operation`
6. **Interaction:** Interacts with Bank for storing and retrieving bank information, and with Operation interface to manage the operation queues.
7. **Interface:** provides methods such as getBanks(), connect(), addOperation().

Bank

1. **High Cohesion:** Encompasses all data and behaviours related to the bank entity like its identification, name, and branches.
2. **Low Coupling:** Interacts with BankDB for data storage and retrieval.
3. **Definition:** Represents a single financial institution within the system.
4. **Responsibilities:** Store and provide access to bank's identification and name, maintain list of branches belonging to the bank.
5. **Composition:**
    - **Attributes:**
        - `id:` Unique DB identifier for the bank
        - `name:` Name of the bank
        - `branches:` List of branches managed by the bank
    - **Functions:**
        - `getId:` Returns bank's ID
        - `getName:` Returns bank's name
        - `getBranches:` Returns list of all branches associated with this bank
6. **Interaction:** interacts with the BankDB to store and retrieve Bank instances, interacts with Branch objects to retrieve them.
7. **Interface:** provides methods like getId(). getName(), getBranches().

Branch

1. **High Cohesion:** Encompasses all data and behaviours related to a branch of a bank. Includes identification, address, and the parent bank.
2. **Low Coupling:** Interacts with Bank for parent bank retrieval. Independent from external database management (BankDB).
3. **Definition:** Represents a single branch of a bank.
4. **Responsibilities:** store and provide access to the branch's ID and address, maintain reference to the bank it belongs to, and provide methods to retrieve branch information.
5. **Composition:**
    - **Attributes:**
        - `id:` Unique DB identifier for the branch
        - `address:` Physical location of the branch
        - `bank:` Reference to the parent bank object
    - **Functions:**
        - `getId:` Returns branch's ID
        - `getAddress:` Returns branch's address
        - `getBank:` Returns parent bank object
6. **Interaction:** interacts with Bank to reference parent bank, Customer interacts with it to get their bank's branch.
7. **Interface:** provides methods such as getID(), getAddress(), and getBank().

Customer

1. **High Cohesion:** All operations related to managing and retrieving customer information such as identification, first and last name, birth date, sin, accounts and bank branch.
2. **Low Coupling:** Interacts minimally with other components. Branch for its associated bank branch, holds references to Account objects.
3. **Definition:** Represents a bank customer with personal data, banking accounts, and branch affiliations.
4. **Responsibilities:** store and provide access to customer's personal information, manage and provide access to the list of accounts held by them, maintain reference to the bank branch the customer belongs to.
5. **Composition:**
    - **Attributes:**
        - `id:` Unique DB identifier for the customer
        - `firstName:` Customer's first name
        - `lastName:` Customer's last name
        - `dateOfBirth:` Customer's date of birth
        - `Sin:` Customer's social insurance number
        - `accounts:` List of accounts owned by the customer
        - `bankBranch:` Bank branch that the customer is associated with
    - **Functions:**
        - `getId:` Returns customer's ID
        - `getFirstName:` Returns customer's first name
        - `getLastName:` Returns customer's last name
        - `getDateOfBirth:` Returns customer's DOB
        - `getSin:` Returns customer's SIN
        - `getAccounts:` Returns list of accounts owned by the customer
        - `getBankBranch:` Returns Branch object that customer is associated with
6. **Interaction:** Customer holds a reference to its Branch, Customer can get list of accounts, Account can get customer.
7. **Interface:** provides methods such as getId(), getFirstName(), getLastName(), etc.

Operation

1. **High Cohesion:** processes all relevant changes to the data inside the database as needed.
2. **Low Coupling:** only interacts with the components LockOperation and TransactionOperation which implements this interface.
3. **Definition:** represents all database changes for the components that implement it
4. **Responsibilities:** processes all changes requested concerning data inside the database.
5. **Composition:**
    - **Attributes:**
        - `none`
    - **Functions:**
        - `Process: Applies changes to the database as needed`
6. **Interaction:** Interacts with the LockOperation, TransactionOperation and the database which updates the data depending on the request.
7. **Interface:** provides process() method.

LockOperation

1. **High Cohesion:** Encompasses all attributes for the locking of each bank account.
2. **Low Coupling:** Interacts only with Account to get the status of isLocked but doesn't need to know all the details of Account.
3. **Definition:** Manages the Lock function of Account which allows the restriction to all access to the account.
4. **Responsibilities:** Stores the lock state of all accounts.
5. **Composition:**
    - **Attributes:**
        - `account: Reference to the account that is used during operation`
        - `locked: New locked/unlocked state of the account`
    - **Functions:**
        - `process: Applies changes to the database as needed`
6. **Interaction:** Interacts with the Account class in order to manage the lock state of the accounts. It will change the database accordingly with the implementation of the Operation interface.
7. **Interface:** Provides process() method.

TransactionOperation

1. **High Cohesion:** Contains all relevant information used in the transaction to update the Database.
2. **Low Coupling:** Interacts with the operation interface and with the TransactionInfo Class while limiting the information access to the Accounts used in the transaction.
3. **Definition:** processes the transaction using the information from the transaction information.
4. **Responsibilities:** Passes the information from the TransactionInfo class to the Operation interface
5. **Composition:**
    - **Attributes:**
        - `transactionInfo: Reference to the transaction information found in TransactionInfo class`
    - **Functions:**
        - `process: Applies changes to the database as needed`
6. **Interaction:** Interacts with the TransactionInfo class to get the account information which will update the database based on the changes since this component implements Operation.
7. **Interface:** provides process() method.

Account

1. **High Cohesion:** Contains all relevant information related to all accounts.
2. **Low Coupling:** It interacts with the Customer and with Transaction but the certain details of the Customer class as well as the information of Transaction
3. **Definition:** Represents the accounts of all users using MyBankUML and includes their level of access as well as allowing access to more specific accounts such as credit, savings and checking accounts.
4. **Responsibilities:** Maintain all details about accounts.
5. **Composition:**
    - **Attributes:**
        - `id:` Unique DB identifier for the account
        - `name:` Name of the Account
        - `customer:` Reference to the Customer that owns the account
        - `transactions:` List of Transactions operating on this account
        - `isLocked:` Lock status of the account
    - **Functions:**
        - `getId:` returns the Id of the account
        - `getName:` returns the Name of the account
        - `getCustomer:` returns the Customer's information
        - `getTransactions:` returns all Transaction objects associated with this account
        - `getIsLocked:` returns the lock status of the account
        - `getBalance:` returns the information of the balance of the account.
        - `addTransaction:` Adds a new transaction object to the transaction list of the account
        - `setIsLocked:` Changes the lock status of the account
6. **Interaction:** Admin can modify it, Users can get to use and access it.
7. **Interface:** Provides the methods of getId(), getName(), getCustomer(), getTransactions(), getIsLocked(), getBalance(), addTransaction(), addTransaction(), setIsLocked(), etc.

CheckingAccount

1. **High Cohesion:** Contains all relevant information related to all checking accounts.
2. **Low Coupling:** It extends the Account class and only interacts with the Account component.
3. **Definition:** Represents the checking accounts of all users using MyBankUML.
4. **Responsibilities:** Maintain all details about checking accounts.
5. **Composition:**
    - **Attributes:**
        - `Inherits from Account Class`
    - **Functions:**
        - `Inherits from Account Class`
6. **Interaction:** Admin can modify it, Users can get to use and access it.
7. **Interface:** Provides the methods inherited from Account.

SavingsAccount

1. **High Cohesion:** Contains all relevant information related to all savings accounts.
2. **Low Coupling:** It extends the Account class and only interacts with the Account component.
3. **Definition:** Represents the saving accounts of all users using MyBankUML.
4. **Responsibilities:** Maintain all details about savings accounts.
5. **Composition:**
    - **Attributes:**
        - `Inherits from Account Class`
    - **Functions:**
        - `Inherits from Account Class`
6. **Interaction:** Admin can modify it, Users can get to use and access it.
7. **Interface:** Provides the methods inherited from Account.

CreditAccount

1. **High Cohesion:** Contains all relevant information related to all credit accounts.
2. **Low Coupling:** It extends the Account class and only interacts with the Account component.
3. **Definition:** Represents the credit accounts of all users using MyBankUML.
4. **Responsibilities:** Maintain all details about credit accounts.
5. **Composition:**
    - **Attributes:**
        - `Inherits from Account Class`
    - **Functions:**
        - `Inherits from Account Class`
6. **Interaction:** Admin can modify it, Users can get to use and access it.
7. **Interface:** Provides the methods inherited from Account.

Transaction

1. **High Cohesion:** This component strictly deals with all financial exchanges.
2. **Low Coupling:** This component interacts only with the Account and TransactionInfo class but does so using the Operation interface.
3. **Definition:** transports the information needed of the account for the transaction process.
4. **Responsibilities:** securely pass relevant information to the transactionInfo class in order to handle the transactions correctly between accounts.
5. **Composition:**
    - **Attributes:**
        - `Id: Unique DB identifier for the transaction`
        - `Info: Reference to the transaction details from TransactionInfo.`
    - **Functions:**
        - `getId: Returns the ID of the transaction.`
        - `getInfo: Returns the information of the referenced TransactionInfo.`
        - `generateReceipt: Returns a string that encompasses all details of the transaction as well as the accounts that were used.`
6. **Interaction:** interacts closely with the Account component and passes information to the TransactionInfo in order to ensure secure transaction.
7. **Interface:** Provides methods that allow to view information of the transaction using getID(),getInfo(), generateReceipt(), etc.

TransactionInfo

1. **High Cohesion:** This TransactionInfo component strictly manages all monetary transfers between accounts.
2. **Low Coupling:** It interacts solely with the Transaction component as well as the TransactionOperation component.
3. **Definition:** manages all monetary transfers from all accounts.
4. **Responsibilities:** To securely process monetary transfers between accounts.
5. **Composition:**
    - **Attributes:**
        - `From:` Account reference that will transfer the money from
        - `To:` Account reference that will transfer the money into
        - `Amount:` Amount that will be transferred between accounts.
    - **Functions:**
        - `getFrom:` Returns the account information that the money will transfer from
        - `getTo:` Returns the account information that the money will transfer into
        - `getAmount:` Returns the amount that will be transferred between accounts
6. **Interaction:** Collaborates the Transaction component in order to associate the transfer to the correct Transaction ID. Also, collaborates with the TransactionComponent in order to process the transfer.
7. **Interface:** it provides methods to confirm or retrieve the information of accounts that are used in the transaction with getFrom() and getTo(). It also provides a method which retrieves the amount that has been transferred using getAmount().

(14) Interface Level Design

14.1 Information Design

The following diagram shows a visual representation of the features related to the user.

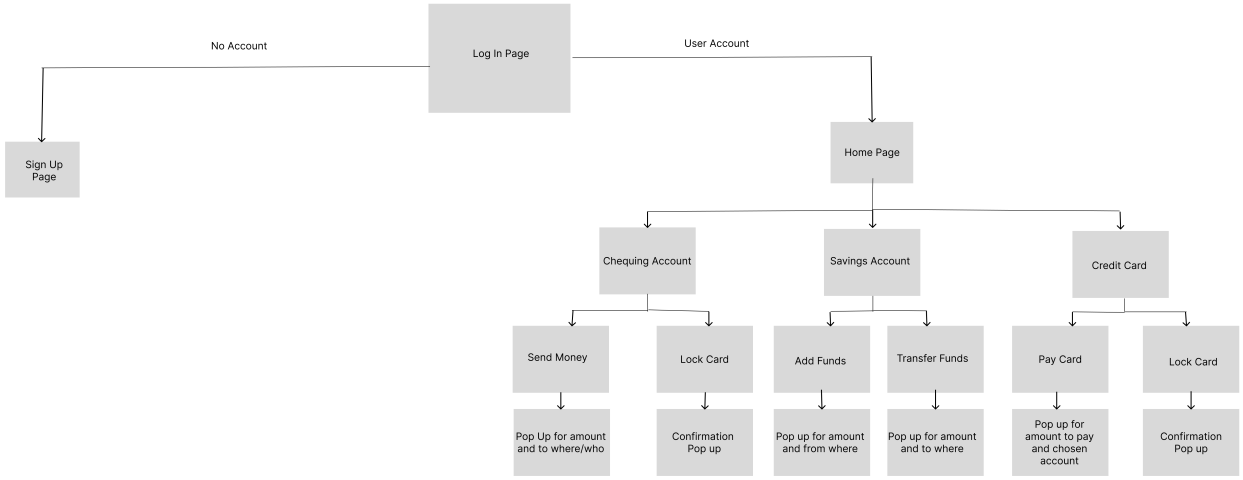

Figure 4: Information Design of the Interface

14.2 Interaction Design

The following diagram represents the flow of the UI from a user perspective.

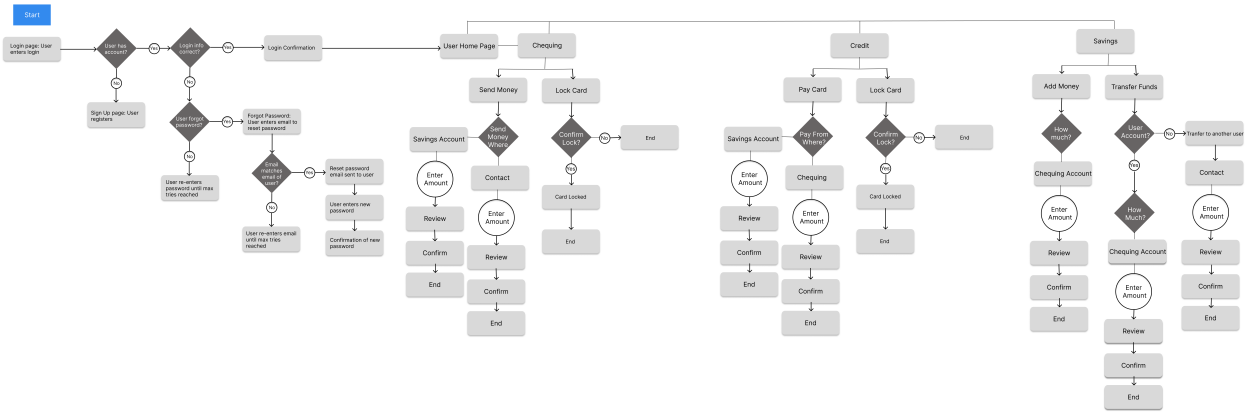

Figure 5: Interaction Design of the Interface

14.3 Visual Design

The following visual is the UI design that we're considering for the login page and the home page.

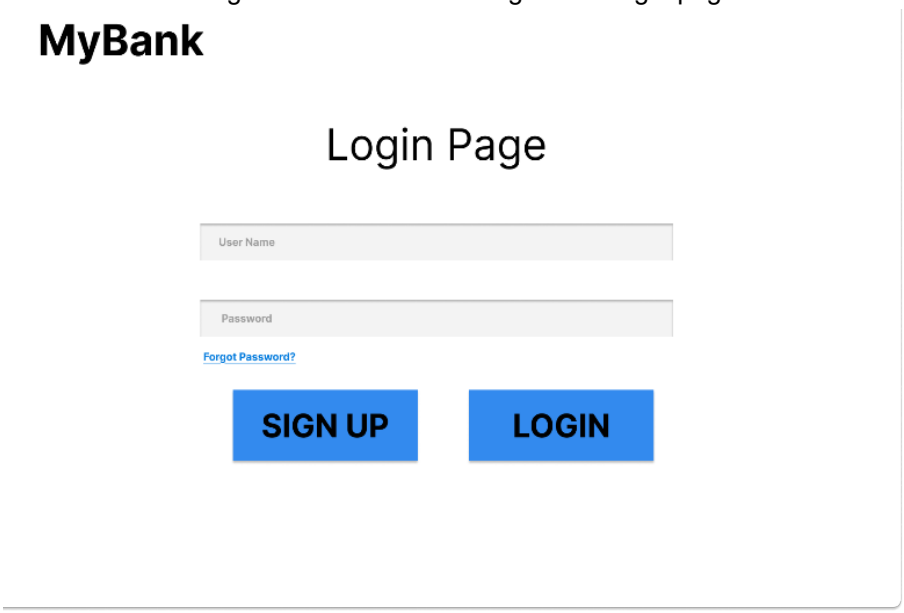

Figure 6: Visual Design of the Login Page

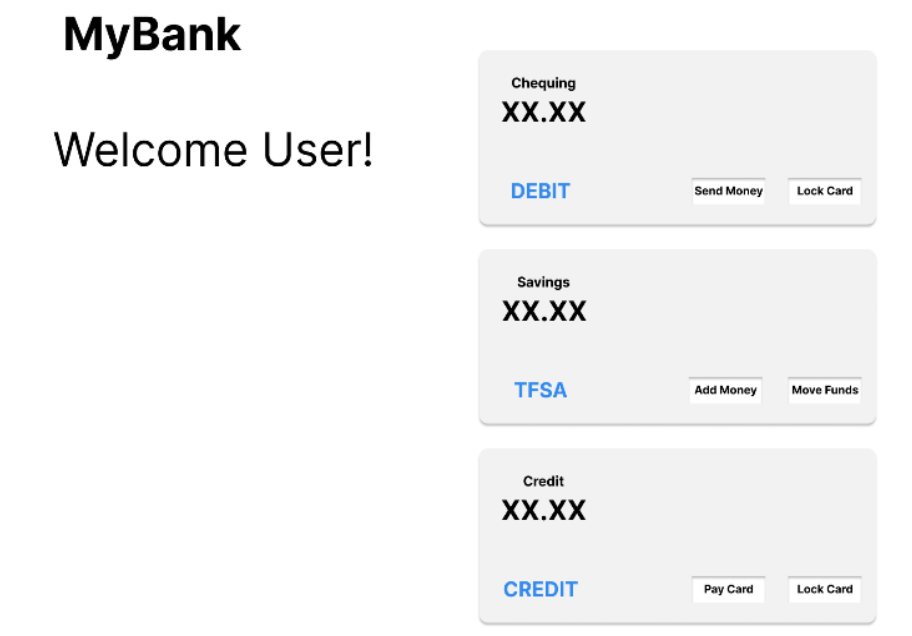

Figure 7: Visual Design of the Account List Page

13.4 User Experience Design

The following visual is the same UI design that we're considering for the login page and the home page, but explained in more detail.

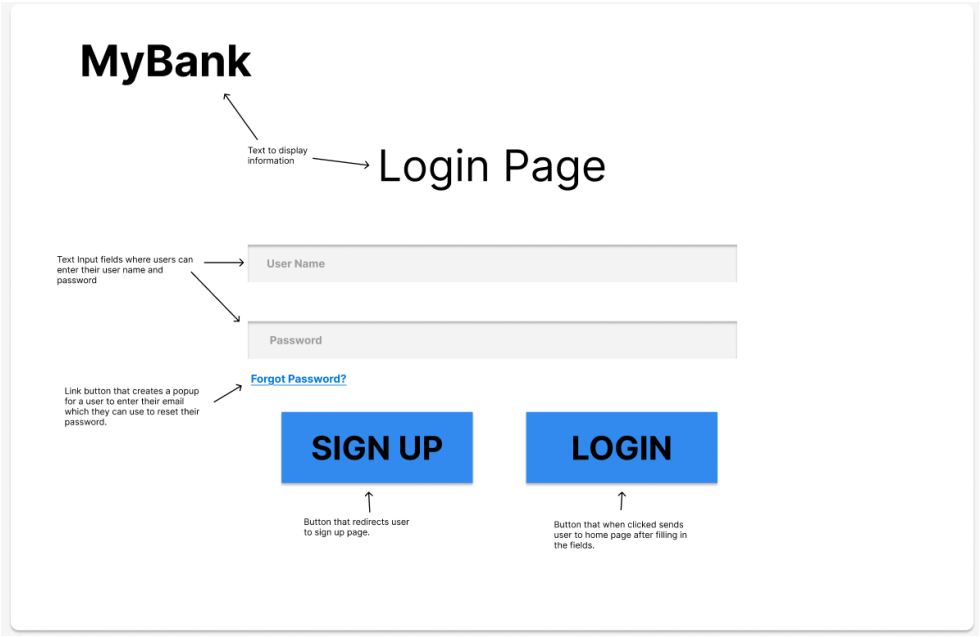

Figure 8: User Experience of the Account List Page

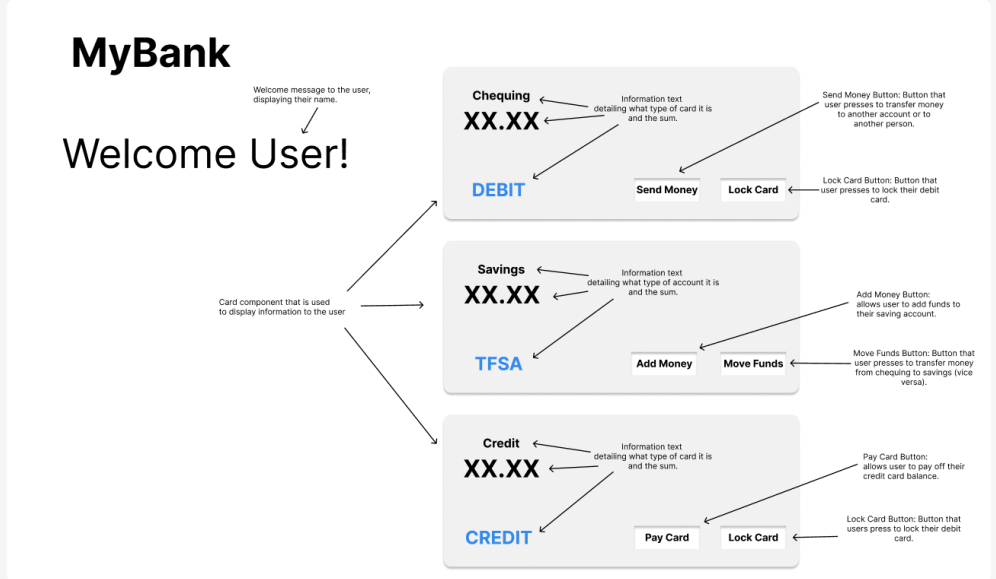

Figure 9: User Experience of the Account List Page

(15) Appendix

(15.1) Activity Log

Date	Name(s)	Task(s)	Time
2025-10-21	**Entire team**	● Sync between team members ● Clarified work division	15mins
2025-10-22	Mykyta Onipchenko Bogdan Ivan	● Component Architecture Design ● Layer Architecture Design	2h
2025-10-23	Mykyta Onipchenko Bogdan Ivan	● UML Architecture Design	1h30mins
2025-10-23	Gorden Quach	● Component Design level	1h30mins
2025-10-23	Kimberley Kam Sing	● Component Design level	1h30mins
2025-10-23	**Entire team**	● Review of assignment requirements	30mins
2025-10-24	Mate Barabas Veronika Pontolillo	● Interface Level Design	2h30mins
2025-10-24	Mykyta Onipchenko	● Design strategy	30mins
2025-10-25	Gorden Quach	● Component Design level	45min
2025-10-26	Mykyta Onipchenko	● Final formatting	45mins

(15.2) Contributions

Name	Diagrams
Mykyta Onipchenko	● Architectural Level Design ● Design strategies
Bogdan Ivan	● Architectural Level Design
Mate Barabas	● Interface Level Design
Veronika Pontolillo	● Interface Level Design
Kimberley Kam Sing	● Component Level Design
Gorden Quach	● Component Level Design