Banking Software System

Design Specification

Revision History

Date	Revision	Description	Author
10/09/2024	1.0	Initial Version created from SRS template	-
10/27/2024	1.1	Edits to SRS Template to better communicate info important for Design Phase -1.Purpose, 2.System Overview	Edgar Romero
10/28/2024	1.2	Use Case 6-10 Sequence Diagrams added	Edgar Romero
10/28/2024	1.3	Added sequence diagrams for use cases 1 - 5	Kyle Lam
10/29/2024	1.4	Added Class Diagrams 1-4	Nicholas Manha
10/29/2024	1.5	Revision of Use Case 6-10 Sequence Diagrams, removed unnecessary extra terms from 1.2, edited Table of Contents, fixed formatting inconsistencies	Edgar Romero
10/29/2024	1.6	Added class diagram with every class and defined classes in section 3.	Nicholas Manha
10/30/2024	1.7	Revised and Edited Architectural Design	Nicholas Ferreira
10/30/2024	1.8	Added Request class in the diagrams	Nicholas Manha
10/31/2024	1.9	Added server client class diagram	Nicholas Manha
10/31/2024	1.10	Added Finished Gantt	Edgar Romero
12/3/2024	1.11	Added updated class diagrams	Nicholas Manha

Table of Contents

1. Purpose

This document outlines the design specifications of the Banking Software System.

1.1 Scope

This document will catalog the series of diagrams and visualizations which describe the overall design of the Banking Software System application, as well as the relationship between the subcomponents within the application.

1.2 Definitions, Acronyms, Abbreviations

1.2.1 Banking Software System: The Banking Software System, or BSS, refers to the system of programs which make up the application that allows bank employees and customers to engage with the bank and all the services it provides including deposits, withdrawals, money transfers, account freezing, account recovery, and more.
1.2.2 Customer: A customer refers to the person who owns or will sign up for a bank account(s), and utilizes the bank's services through a Teller or through the ATM system.
1.2.3 Bank Teller: A Bank Teller, or Teller, is an employee of the bank that facilitates account transactions for a Customer that the Customer cannot do themselves on an ATM.
1.2.4 Automated Teller Machine: An Automated Teller Machine, or ATM, is a computer that provides a GUI for a Customer to complete quick transactions with their bank accounts which don't require a Teller.
1.2.5 Bank's Services: This term refers to every service the Bank provides for managing the Customers' bank accounts and the money coming to and from the bank accounts, as performed by a Teller or via ATM.
1.2.6 Account Number: A unique identifier associated with a Customer's Account.
1.2.14 Checking Account: An Account made for Customers to conveniently manage and access their money.
1.2.7 Balance: The current amount of money stored within a given Account.
1.2.8 Savings Account: An Account made for Customers to primarily store money.
1.2.9 PIN: Refers to the sequence of numbers a Customer uses to verify they have access to an account.
1.2.10 Fraud: Fraud refers to any illegitimate addition or subtraction of money from an Account through any of the Bank's Services using an ATM, a Bank Teller, or through any unidentified exploits within the BSS' Server-Client infrastructure.
1.2.11 Account Recovery: Account Recovery is a process a Bank Teller and Customer go through to regain access to a Bank Account that a Customer may have forgotten their PIN or Account Number for.
1.2.12 Transaction: The completion of any action related to managing the money in an Account or managing aspects of an Account itself are referred to as Transactions.
1.2.13 Sequential Transactions:
1.2.14 Deposit: The action of adding funds to a customer's bank account, typically through an ATM or by a bank employee. A deposit increases the balance of the account.
1.2.15 Withdraw: The action of removing funds from a customer's bank account which decreases the amount of the account's balance. It can be done by the ATM or bank teller.

1.2.16 Transfer: The action of moving funds from one bank account to another and can occur between a customer's own accounts or between accounts owned by different customers
1.2.17 Account Freeze: When a bank account is placed under a temporary restriction to prevent any transactions (like withdrawals, deposits, or transfers). Only bank tellers can initiate or stop an account freeze.
1.2.18 GUI: The interface that allows users to interact with the banking system visually and uses graphical elements like buttons, windows, and forms.
1.2.19 TCP/IP: Transmission Control Protocol/Internet Protocol
1.2.20 Port: A number to uniquely distinguish multiple connections to a single IP.
1.2.21 Server Application (Server): A computer that provides resources to other computers
1.2.22 Client Application (Client): A computer that receives the resources given by a server
1.2.23 Receipt: A Receipt is the message printed to the screen/given to a customer by a Bank Teller after the Customer performs or requests a transaction be completed from their account.

1.3 References

Banking Software System Github Link

Use Case Specification Document

UML Use Case Diagrams Document

Class Diagram

Sequence Diagrams

Gantt Chart

1.4 Overview

The Banking Software System is designed to facilitate secure and efficient interactions between customers and bank tellers. It provides ATM services with services to minimize fraud. From these services and roles, there will be two main interfaces of an ATM interface for the customer and a Teller interface for bank employees. The BSS will be using a Client-Server architecture over TCP/IP.

2. System Overview

2.1 System Context

The Banking Software System application is one that will be used at Bank locations nationwide. Bank Tellers will use the BSS application from behind the counter on company computers to help customers manage their bank accounts and provide other services to customers on behalf of the Bank.

2.2 Stakeholders

The following are identified as stakeholders and are users of, are affected by, or are involved in the development and design of the Banking Software System:

- Client: Christopher Smith
- End Users: Bank Customers, Bank Tellers
- Developers: Edgar Romero, Kyle Lam, Nicholas Ferreira, Nicholas Manha

2.3 General Design Considerations

The following assumptions and constraints were made and considered for the design of the Banking System Software:

Assumptions and Dependencies

2.3.1.1 It is assumed that customers are communicating with the teller to perform actions they request.

2.3.1.2 Money that is deposited by a customer is assumed to be real tender.

2.3.1.3 There must be at least one trained teller capable of operating the employee duties.

2.3.1.4 The system relies on TCP/IP communication for all transactions so it is assumed that both the client and server applications will be deployed in spots with good, stable network connections.

2.3.1.5 It is assumed that only real humans are using the application and that there are no bots.

2.3.1.6 The user running the software must have an operating system that can run Java programs.

2.3.1.7 It is assumed that all users accessing the system are given correct credentials and proper logins and that security policies are followed by all personnel.

Constraints

2.3.2.1 The system cannot use a database, frameworks, or other external technologies and must store data in a text file.

2.3.2.1 The system must operate using TCP/IP for communication between the Client and Server applications. No other communication protocols may be used.

2.3.2.3 Server-Client communication must occur using ports above 1024, because lower ports are reserved for other services.

2.3.2.4 Encryption can’t be used for transmitting sensitive information like account numbers, PINs, or transaction details between the Client and Server. Security must rely on other ways.

2.3.2.5 The user interfaces for the Teller and ATM must be built using Java's GUI libraries

3. Architectural Design

Overview of Components

3.1.1 Account:

The Account module is a key component of the Banking System Software (BSS) to be used in teller and ATM environments and supports customer and employee interactions.

3.1.1.1 Users can check their account ID, authorized users, and account balance.

3.1.1.2 Users can update their PIN, freeze their account if necessary, and add additional users to the account with appropriate permissions.

3.1.1.3 Users can perform essential transactions like withdrawals, deposits, and transfers between accounts.

3.1.2 User:

The User module defines user roles in the system, including customers, tellers, accounts, and manages their interactions with accounts and system features.

3.1.2.1 The User class is the interface for the Customer, Teller and Account class.

3.1.2.2 The Customer class represents a person who has an account under the bank. It holds an arraylist of accounts that are associated with that customer. It also has a method to get the account with a given index.

3.1.2.3 The Teller class has all of the methods associated with a teller, including creating an account, freezing an account, adding a customer to an account, reading logs, recovering and changing pins and all of the basic operations an ATM provides.

3.1.3 GUI:

The GUI module provides user interfaces for ATM and teller operations, allowing customers and tellers to interact with the system through tailored screens with corresponding controls.

The GUI class provides a front facing interface for customers and tellers. Customers and Tellers use the same GUI with different front facing features. Tellers use their own interface that allows them to perform the actions described in 3.1.2.3.

3.1.4 Log:

The Log module records critical events within the system, creating logs for actions like account creation, transfers, withdrawals, deposits, and ATM sessions.

3.1.4.1 AccountCreationLog is for when an account is created. It logs the ID of the teller who made the account, the timestamp and the ID of the created account.

3.1.4.2 TransferLog is for when a transfer happens, it stores the timestamp of the event, account ID of the account the funds are being transferred from, and the account ID of the account that is being transferred to.

3.1.4.3 WithdrawLog is for when a withdrawal occurs. It stores the timestamp of the event and the ID of the account that was withdrawn from.

3.1.4.4 DepositLog is for when a deposit occurs. It stores the timestamp of the event and the ID of the account that was deposited to.

3.1.4.5 SessionLog is for keeping information about a Session. It stores the timestamp of when the Session was started as well as when it ended, and the ID of the account that is using the session.

3.1.5 Bank Class

The Bank class manages all accounts, customers, and tellers, providing access and functions inside the BSS.

3.1.5.1 The Bank class keeps attributes keeping track of all the accounts, customers, and tellers associated with the bank.

3.1.5.2 The Bank class has methods to find accounts, customers, and tellers from the bank.

3.1.6 ATM Class

The ATM class enables customers access to basic banking functions such as balance inquiry, withdrawals, deposits, and transfers through the ATM interface.

3.1.6.1 The ATM class stores a session that keeps track of the actions taken within the timeframe of connecting to the ATM.

3.1.6.2 The ATM class contains the functionality for a user to login/logout, perform interactions (withdraw, deposit, transfer, view balance), and see receipts.

3.1.7 Session Class

The Session class manages login sessions on the ATM, tracking usage time and actions performed, as well as generating logs for auditing purposes.

3.1.7.1 The Session class has a unique ID to track the session, a message, the date/time of the session, and the user's ID who initiated the session.

3.1.7.2 The Session class has the functionality to write the session information to a log.

Banking System Github Link

https://github.com/NFerreira98/Banking-System

Banking System Use Case Specification Document

Use Case ID: 1
Use Case Name: Teller Creates an Account
Relevant Requirements: 3.1.4.3

Primary Actor: Bank Teller

Pre-conditions:
1. A Customer must request from a Teller that a Bank Account be made for them.

Post-conditions:
1. The newly made Bank Account will be accessible by the Customer and a Teller via PIN and Account Number.
2. This event has the date and time recorded in a log.

Basic Flow or Main Scenario:
1. A Bank Teller asks the Customer what kind of Account they would like to establish with the Bank.
2. The Bank Teller creates a new Checking Account.
3. A PIN number is created for the Account.
4. The Bank Teller gives the Customer their Account number and PIN number.
5. The Customer gets a receipt detailing the date and time of their Account's creation.

Extensions or Alternate Flows:
Alternate Flow 1:
1. A Bank Teller asks the Customer what kind of Account they would like to establish with the Bank.
2. The Bank Teller creates a new Savings Account.
3. A PIN number is created for the Account.
4. The Bank Teller gives the Customer their Account number and PIN number.
5. The Customer gets a receipt detailing the date and time of their Account's creation.

Exceptions:
N/A

Related Use Cases:
1. Use Case 7: Logging Transactions
2. Use Case 8: Print Receipt

Use Case ID: 2
Use Case Name: Customer manages money at ATM
Relevant Requirements: 3.1.4.3, 3.3.3

Primary Actor: Customer

Pre-conditions:
1. There must exist an Account for a Customer to access and manage.
2. The Customer must have money in their Account to manage, or on hand to deposit into their Account.

Post-conditions:
1. A receipt of the completed Transactions and changes in the total Balance of the Account accessed.
2. This event has the date and time recorded in a log.

Basic Flow or Main Scenario:

1. A Customer requests to make a Deposit at an ATM.
2. The ATM prompts the user for an Account Number and PIN for access to an Account.
3. The Customer enters their credentials.
4. The Customer deposits money into an account.
5. A Receipt with the date, time and amount deposited is printed for the Customer.

Extensions or Alternate Flows:

Alternate Flow 1:

1. A Customer requests to make a Withdrawal at an ATM.
2. The ATM prompts the user for an Account Number and PIN for access to an Account.
3. The Custo
4. mer enters their credentials.
5. The Customer withdraws money from their account in an amount that does not exceed their Balance.
6. A Receipt with the date, time and amount withdrawn is printed for the Customer.

Alternate Flow 2:

1. A Customer requests to make a Transfer at an ATM.
2. The ATM prompts the user for an Account Number and PIN for access to an Account to Transfer money from.
3. The Customer enters their credentials.
4. The Customer then enters the amount of money they would like to transfer into another Account.
5. The ATM prompts the user for an Account Number.
6. The transaction occurs and a receipt with the date, time, Account Numbers and the amount transferred is printed for the Customer.

Exceptions:

1. A Customer cannot attempt to Transfer or Withdraw more money than their total Account Balance

Related Use Cases:

1. Use Case 4: Customer Transfers money between Accounts
2. Use Case 6: Check Account Balance
3. Use Case 7: Logging Transactions
4. Use Case 8: Print Receipt

Use Case ID: 3
Use Case Name: Bank Teller adds Customer to Account
Relevant Requirements: 3.1.1.6, 3.1.3.3

Primary Actor: Teller

Pre-conditions:

1. There must be an Account for the teller to add the Customer to.

Post-conditions:

1. The Customer is added to said Account.
2. The Customer is given his own login for the Account and can.
   interact with it.
3. A log detailing the addition of a Customer to an Account is created.

Basic Flow or Main Scenario:

1. Customer requests that the teller add them to an Account.
2. A prompt is given to the Account owner asking if they want this Customer allowed on their Account.
3. The Account owner agrees.
4. That Customer is allowed access to said Account.

5. Login credentials are given to said Customer.
6. The event that the Customer was added to another Account is logged.

Extensions or Alternate Flows:
7. Customer requests that the Teller add them to an Account.
8. A prompt is given to the account owner asking if they want this Customer allowed on their Account
9. The Account owner says no
10. That Customer is rejected from being added to that Account.

Exceptions:
1. The requested Account does not exist.
2. The owner of the Account does not consent to adding the new Customer

Related Use Cases:
1. Use Case 8: Print Receipt

Use Case ID: 4
Use Case Name: Customer Transfers money between Accounts
Relevant Requirements: 3.1.2.2, 3.1.2.3, 3.1.3.2, 3.1.3.3

Primary Actor: Customer

Pre-conditions:
1. A Customer must have an Account with money, and a target Account to send money to.

Post-conditions:
1. Money is transferred from the Customer's Account to the target Account.
2. A log detailing this transfer of money is created.

Basic Flow or Main Scenario:
1. Customer logs into their account
2. Customer begins a transfer operation using ATM GUI
3. ATM prompts customer with target account
4. Customer enters target account ID
5. The ATM prompts a Customer with what Account they want to send money from
6. A Customer chooses a Checking Account
7. ATM prompts Customers with how much money they wish to send
8. Money is sent from the customers account to the target Account
9. A receipt with the date, time and amount Transferred is printed for the Customer

Extensions or Alternate Flows:
1. Customer requests a Teller Transfer money from a source Account to a target Account.
2. Teller accesses funds from source Account.
3. Teller Transfers requested an amount of money to target Account.

Exceptions:
1. Insufficient funds in sender Account.
2. Target account does not exist.

Related Use Cases:
1. Use Case 2: Customer manages money at ATM
2. Use Case 7: Logging Transactions
3. Use Case 8: Print Receipt

Use Case ID: 5
Use Case Name: An Account is accessed using PIN and account number
Relevant Requirements: 3.1.1.2, 3.1.4.2

Primary Actor: Customer

Pre-conditions:
1. The account that is being accessed must exist.

Post-conditions:
1. The customer is logged into their account and is allowed to manage their funds.

Basic Flow or Main Scenario:
1. The Customer enters their PIN and Account Number.
2. ATM logs them into their account
3. A receipt of any transactions performed on the Customer's Account is created.

Extensions or Alternate Flows:
1. Teller is given a Customers PIN and Account Number.
2. Teller logs into the Customer's Account.
3. A receipt detailing every one of the Bank Services performed by the Bank Teller is created.

Exceptions:
1. Credentials entered are not associated with any Account
2. Credentials entered do not permit the customer to access the Account associated with said credentials.

Related Use Cases:
1. Use Case 2: Customer manages money at ATM
2. Use Case 6: Check Account Balance

Use Case ID: 6
Use Case Name: Check Account Balance
Relevant Requirements: 3.1.2.2, 3.1.3.2

Primary Actor: Customer

Pre-conditions: The customer must be logged in with a valid account and PIN.

Post-conditions: The current account balance is displayed, and no further changes are made.

Basic Flow or Main Scenario:
1. Customer logs into account
2. Customer selects the option to check balance on ATM interface
3. ATM displays the customer's current balance
4. Customer views the account balance
5. ATM offers the Customer the option to perform another action or to log out.
6. Customer logs out and ATM ends the session

Extensions or Alternate Flows:
Alternate Flow 1:
1. Customer logs in with invalid credentials
2. ATM sends prompt informing Customer that credentials are invalid
3. Customer is prompted to enter credentials again or stop

4. This flow continues until Customer enters valid credentials or choses to stop

Exceptions:

N/A

Related Use Cases:

1. Use case 2: Customer manages money at ATM
2. Use case 4: Customer Transfers money between Accounts
3. Use case 5: An Account is accessed using PIN and account number

---

Use Case ID: 7
Use Case Name: Logging Transactions
Relevant Requirements: 3.1.1.5, 3.1.5.1, 3.1.5.2, 3.1.5.3

Primary Actor: Log file

Pre-conditions:

1. A valid action was taken in the banking system, whether it's withdrawing, depositing, or transferring.

Post-conditions:

1. The log file will have the recent transaction added.

Basic Flow or Main Scenario:

1. The customer initiates a banking action such as withdrawing, depositing, or transferring with a teller.
2. The teller performs the corresponding response.
3. Once the action is performed, the logging file will add the date, time, the amount of money in the transaction, specify the transaction itself, and the accounts involved in the transaction, and a unique id.

Extensions or Alternate Flows:
Alternate Flow 1 (ATM transaction)

1. The customer initiates a banking action such as withdrawing, depositing, or transferring with the teller with an ATM.
2. The ATM performs the corresponding response.
3. Once the action is performed, the logging file will add the date, time, the amount of money in the transaction, specify the transaction itself, and the accounts involved in the transaction, and a unique id.

Exceptions:

1. Text file not found

Related Use Cases:

1. Use case 8: Print Receipt
2. Use case 9: View Transaction Logs

Use Case ID: 8
Use Case Name: Print Receipt
Relevant Requirements: 3.1.5.3, 3.3.4

Primary Actor: User

Pre-conditions:

1. There must exist an Account for a Customer to access and perform a service on.
2. A Bank Service is performed for a Customer

Post-conditions:
1. A change in the Customer's Account occurs.
2. A log is created detailing the transaction.

Basic Flow or Main Scenario:
1. A Customer is prompted for their credentials to access an Account at the ATM.
2. A Customer logs into their Account at an ATM.
3. The Customer performs a Transaction at the ATM.
4. A Receipt is printed for the user detailing the transaction.

Extensions or Alternate Flows:
Alternate Flow 1:
1. A Customer requests one of the Bank's services through a Teller.
2. The Teller performs the Bank Service.
3. A Receipt is printed for the Customer detailing the transaction.

Exceptions:
1. A receipt will not be printed if the user is only checking their Account Balance.

Related Use Cases:
1. Use case 7: Logging Transactions

Use Case ID: 9
Use Case Name: View Transaction Logs
Relevant Requirements: 3.1.5.1, 3.1.3.5

Primary Actor: Teller

Pre-conditions:
1. The Customer or Teller is authenticated and the system has stored a log of transactions in a text file associated with the account

Post-conditions:
1. The Transaction is retrieved from the text file and displayed to the user and no changes were made to the Account or Log file.

Basic Flow or Main Scenario:
1. Customer or Teller picks the view transaction log option on their respective GUI
2. Text File Log System retrieves the relevant log file, reads the data, and returns it to the GUI
3. GUI displays the transaction logs for the user.
4. ATM/GUI asks user to perform another option
5. User chooses to log out and session ends

Extensions or Alternate Flows:
Alternate Flow 1: No transaction history
1. Customer or Teller picks the view transaction log option on their respective GUI
2. Text file retrieves the log file but can not find any records.
3. Text file will return a message that says the file is empty
4. GUI will display that message
5. User chooses to log out and the session ends.

Exceptions:
N/A
Related Use Cases:
1. Use case 7: Logging Transactions

Use Case ID: 10
Use Case Name: Teller Freezes Account
Relevant Requirements: 3.1.4.4

Primary Actor: Teller

Pre-conditions:
1. The Account that is being frozen must exist.

Post-conditions:
1. The target Account is flagged as Frozen, meaning Bank Services cannot be performed on that account other than viewing it.

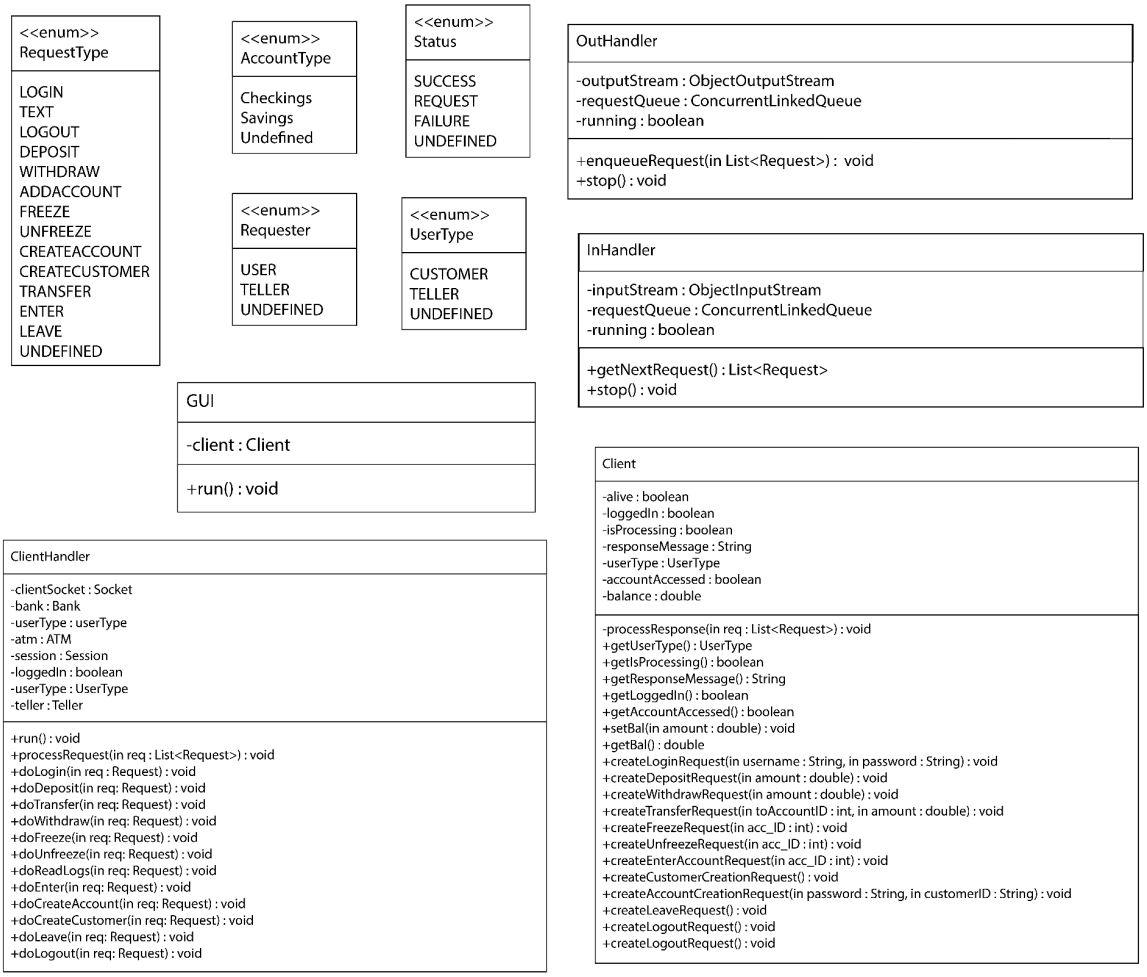
Basic Flow or Main Scenario:
1. Teller receives Account ID
2. Teller flags account as Frozen

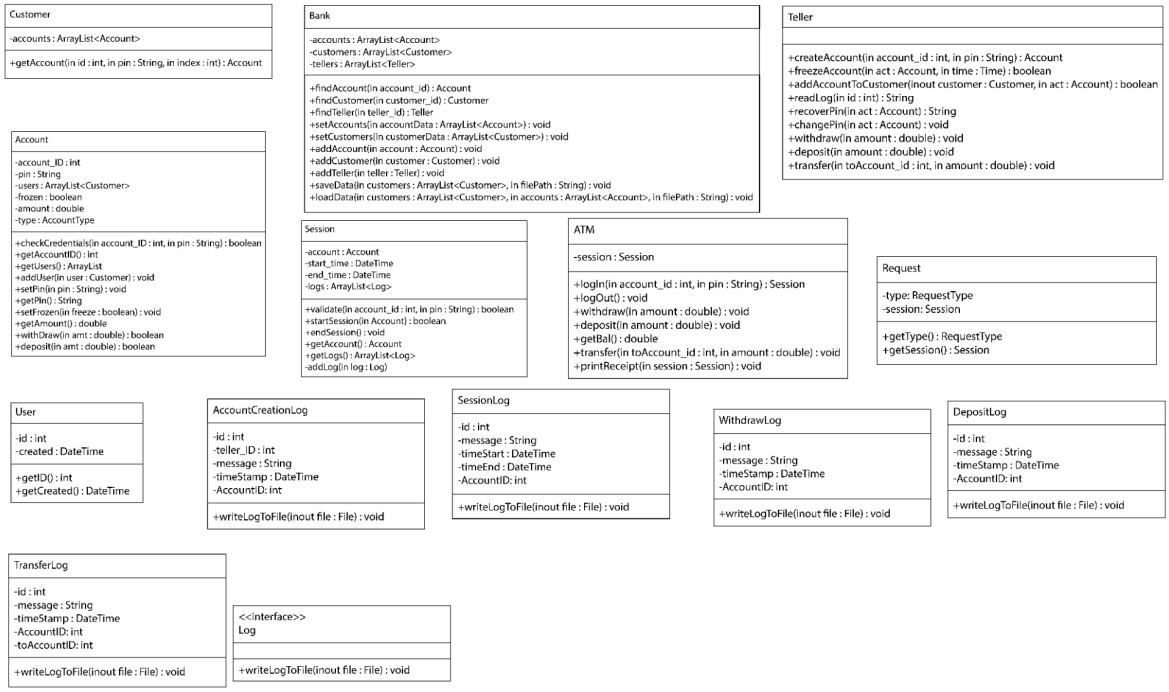
Extensions or Alternate Flows:
N/A
Exceptions:
1. Account doesn't exist

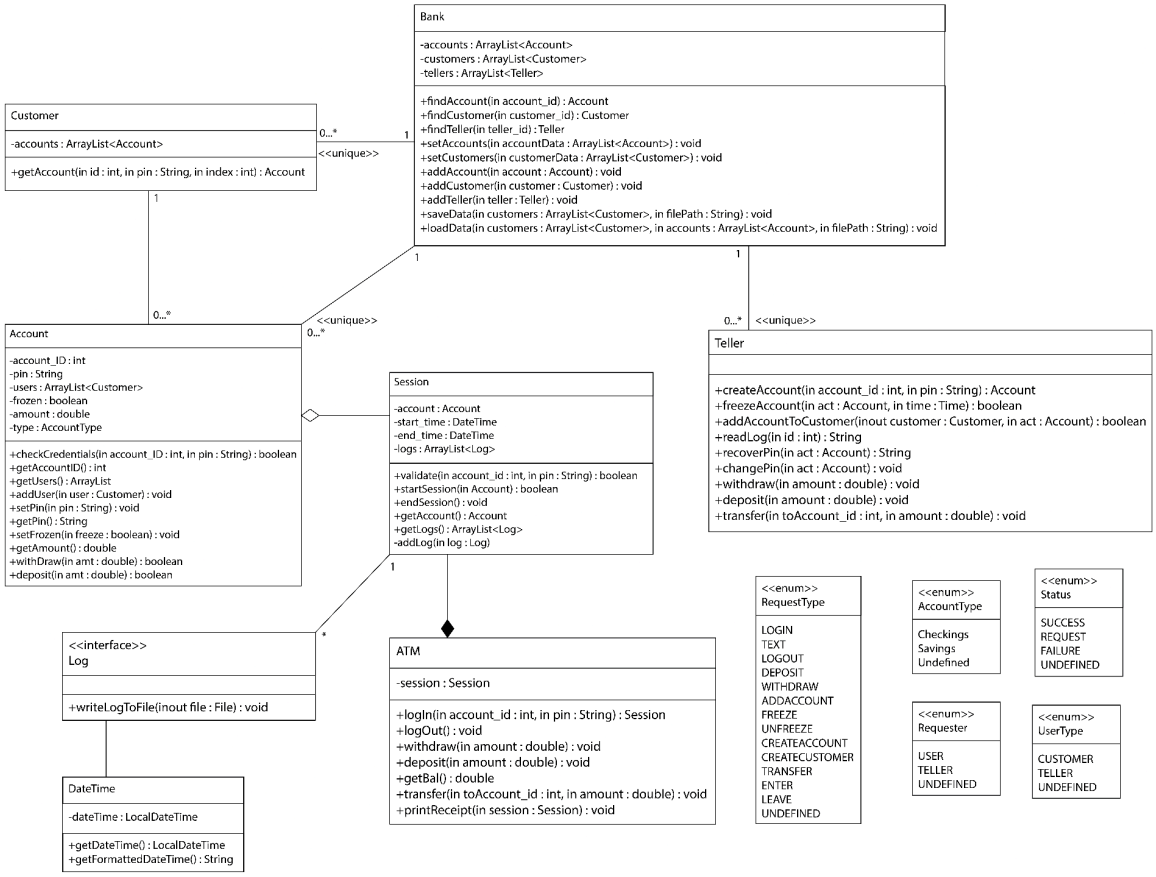
Related Use Cases:
1. Use case 1: Teller Creates an Account
2. Use Case 7: Logging Transactions

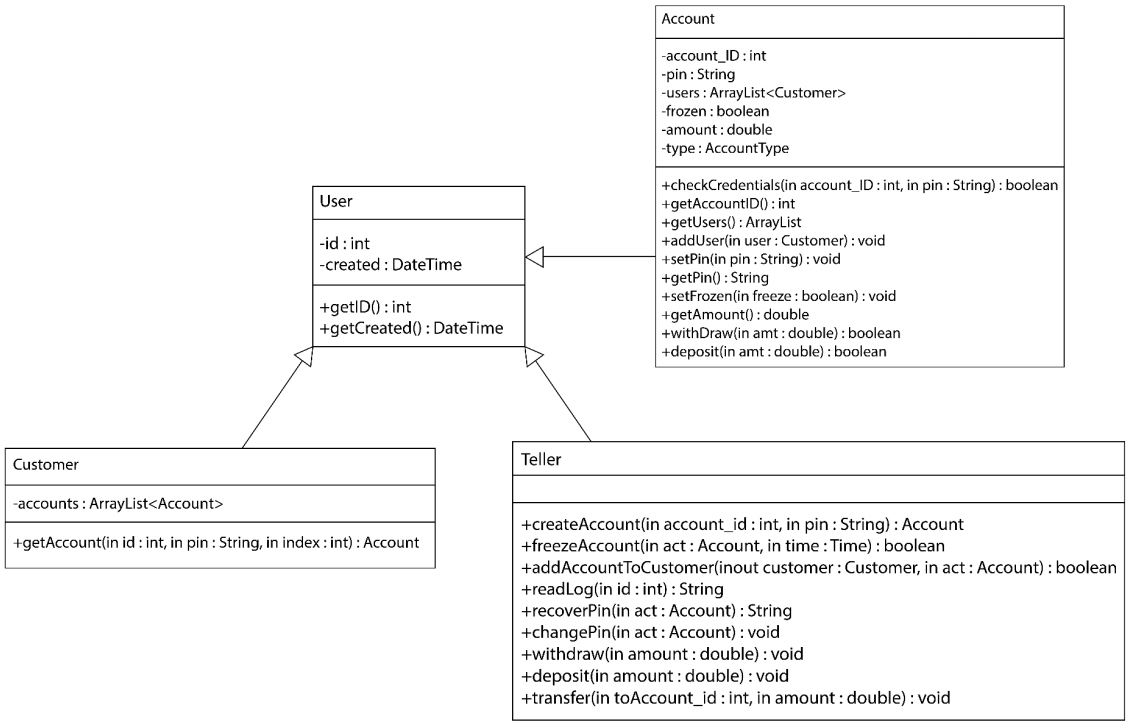
Bank Software System Use Case Diagram

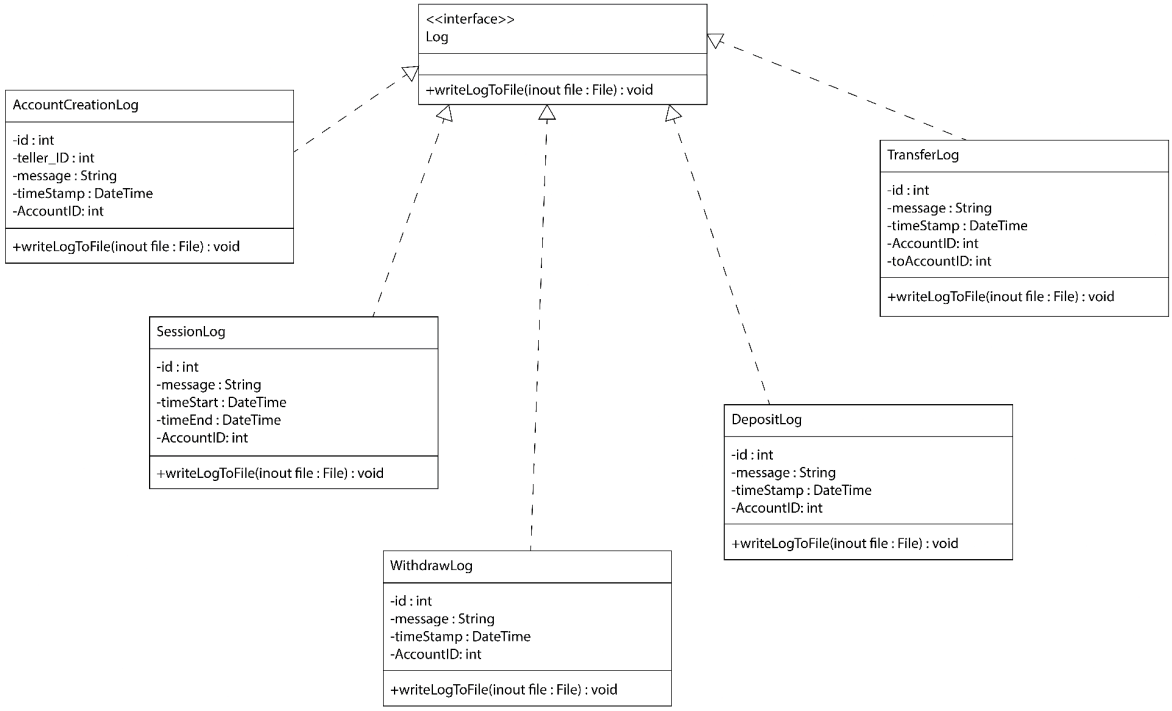
Bank Software System Class Diagrams

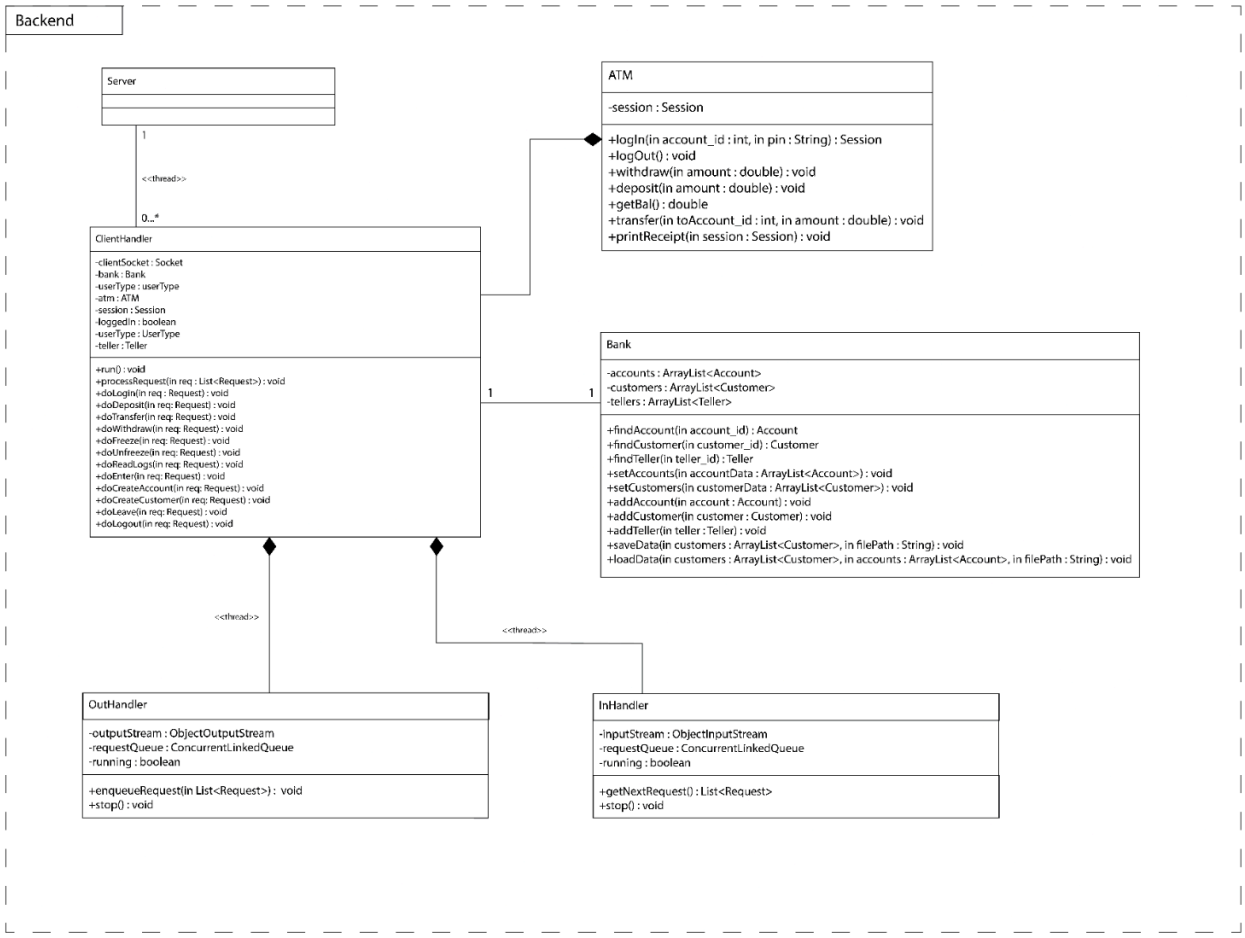


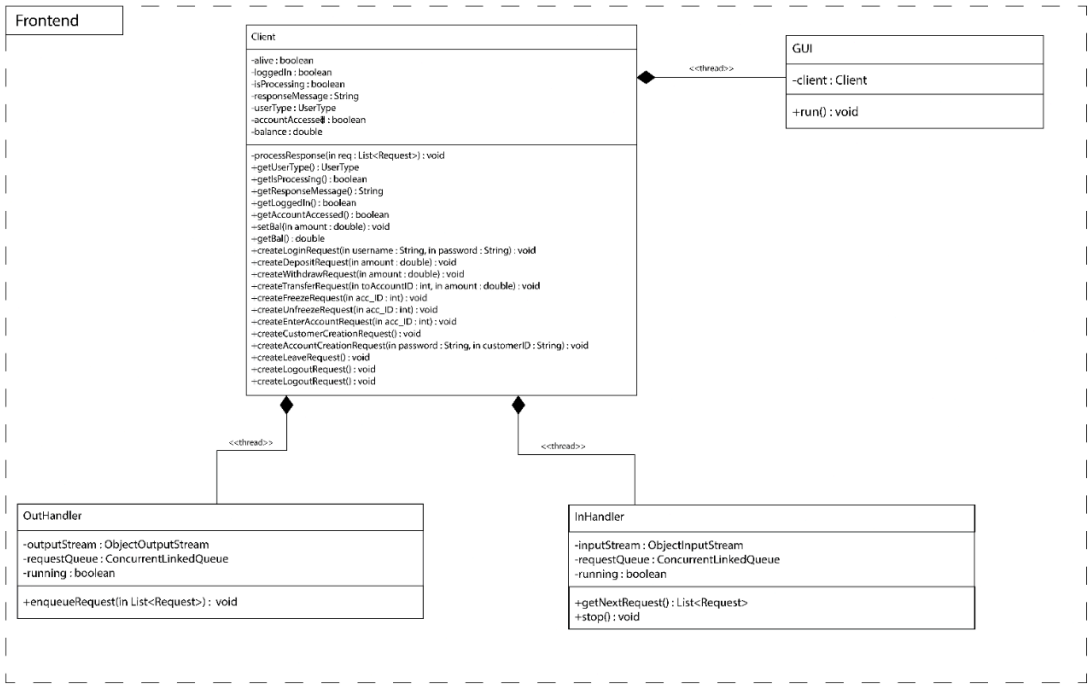

Customer
-accounts : ArrayList<Account>
+getAccount(in id : int, in pin : String, in index : int) : Account
Bank
-accounts : ArrayList<Account>
-customers : ArrayList<Customer>
-tellers : ArrayList<Teller>
+findAccount(in account_id) : Account
+findCustomer(in customer_id) : Customer
+findTeller(in teller_id) : Teller
+setAccounts(in accountData : ArrayList<Account>) : void
+setCustomers(in customerData : ArrayList<Customer>) : void
+addAccount(in account : Account) : void
+addCustomer(in customer : Customer) : void
+addTeller(in teller : Teller) : void
+saveData(in customers : ArrayList<Customer>, in filePath : String) : void
+loadData(in customers : ArrayList<Customer>, in accounts : ArrayList<Account>, in filePath : String) : void
Teller
+createAccount(in account_id : int, in pin : String) : Account
+freezeAccount(in act : Account, in time : Time) : boolean
+addAccountToCustomer(inout customer : Customer, in act : Account) : boolean
+readLog(in id : int) : String
+recoverPin(in act : Account) : String
+changePin(in act : Account) : void
+withdraw(in amount : double) : void
+deposit(in amount : double) : void
+transfer(in toAccount_id : int, in amount : double) : void
Account
-account_ID : int
-pin : String
-users : ArrayList<Customer>
-frozen : boolean
-amount : double
-type : AccountType
+checkCredentials(in account_ID : int, in pin : String) : boolean
+getAccountID() : int
+getUsers() : ArrayList
+addUser(in user : Customer) : void
+setPin(in pin : String) : void
+getPin() : String
+setFrozen(in freeze : boolean) : void
+getAmount() : double
+withDraw(in amt : double) : boolean
+deposit(in amt : double) : boolean
Session
-account : Account
-start_time : DateTime
-end_time : DateTime
-logs : ArrayList<Log>
+validate(in account_id : int, in pin : String) : boolean
+startSession(in Account) : boolean
+endSession() : void
+getAccount() : Account
+getLogs() : ArrayList<Log>
-addLog(in log : Log)
ATM
-session : Session
+logIn(in account_id : int, in pin : String) : Session
+logOut() : void
+withdraw(in amount : double) : void
+deposit(in amount : double) : void
+getBal() : double
+transfer(in toAccount_id : int, in amount : double) : void
+printReceipt(in session : Session) : void
Request
-type: RequestType
-session: Session
+getType() : RequestType
+getSession() : Session
User
-id : int
-created : DateTime
+getID() : int
+getCreated() : DateTime
AccountCreationLog
-id : int
-teller_ID : int
-message : String
-timeStamp : DateTime
-AccountID: int
+writeLogToFile(inout file : File) : void
SessionLog
-id : int
-message : String
-timeStart : DateTime
-timeEnd : DateTime
-AccountID: int
+writeLogToFile(inout file : File) : void
WithdrawLog
-id : int
-message : String
-timeStamp : DateTime
-AccountID: int
+writeLogToFile(inout file : File) : void
DepositLog
-id : int
-message : String
-timeStamp : DateTime
-AccountID: int
+writeLogToFile(inout file : File) : void
TransferLog
-id : int
-message : String
-timeStamp : DateTime
-AccountID: int
-toAccountID: int
+writeLogToFile(inout file : File) : void
<<interface>>
Log
+writeLogToFile(inout file : File) : void


Bank
-accounts : ArrayList<Account>
-customers : ArrayList<Customer>
-tellers : ArrayList<Teller>
+findAccount(in account_id) : Account
+findCustomer(in customer_id) : Customer
+findTeller(in teller_id) : Teller
+setAccounts(in accountData : ArrayList<Account>) : void
+setCustomers(in customerData : ArrayList<Customer>) : void
+addAccount(in account : Account) : void
+addCustomer(in customer : Customer) : void
+addTeller(in teller : Teller) : void
+saveData(in customers : ArrayList<Customer>, in filePath : String) : void
+loadData(in customers : ArrayList<Customer>, in accounts : ArrayList<Account>, in filePath : String) : void
Customer
-accounts : ArrayList<Account>
+getAccount(in id : int, in pin : String, in index : int) : Account
0...*
1
<<unique>>
1
0...*
Account
-account_ID : int
-pin : String
-users : ArrayList<Customer>
-frozen : boolean
-amount : double
-type : AccountType
+checkCredentials(in account_ID : int, in pin : String) : boolean
+getAccountID() : int
+getUsers() : ArrayList
+addUser(in user : Customer) : void
+setPin(in pin : String) : void
+getPin() : String
+setFrozen(in freeze : boolean) : void
+getAmount() : double
+withDraw(in amt : double) : boolean
+deposit(in amt : double) : boolean
<<unique>>
0...*
1
1
0...*
<<unique>>
Teller
+createAccount(in account_id : int, in pin : String) : Account
+freezeAccount(in act : Account, in time : Time) : boolean
+addAccountToCustomer(inout customer : Customer, in act : Account) : boolean
+readLog(in id : int) : String
+recoverPin(in act : Account) : String
+changePin(in act : Account) : void
+withdraw(in amount : double) : void
+deposit(in amount : double) : void
+transfer(in toAccount_id : int, in amount : double) : void
Session
-account : Account
-start_time : DateTime
-end_time : DateTime
-logs : ArrayList<Log>
+validate(in account_id : int, in pin : String) : boolean
+startSession(in Account) : boolean
+endSession() : void
+getAccount() : Account
+getLogs() : ArrayList<Log>
-addLog(in log : Log)
1
*
<<interface>>
Log
+writeLogToFile(inout file : File) : void
DateTime
-dateTime : LocalDateTime
+getDateTime() : LocalDateTime
+getFormattedDateTime() : String
ATM
-session : Session
+logIn(in account_id : int, in pin : String) : Session
+logOut() : void
+withdraw(in amount : double) : void
+deposit(in amount : double) : void
+getBal() : double
+transfer(in toAccount_id : int, in amount : double) : void
+printReceipt(in session : Session) : void
<<enum>>
RequestType
LOGIN
TEXT
LOGOUT
DEPOSIT
WITHDRAW
ADDACCOUNT
FREEZE
UNFREEZE
CREATEACCOUNT
CREATECUSTOMER
TRANSFER
ENTER
LEAVE
UNDEFINED
<<enum>>
AccountType
Checkings
Savings
Undefined
<<enum>>
Status
SUCCESS
REQUEST
FAILURE
UNDEFINED
<<enum>>
Requester
USER
TELLER
UNDEFINED
<<enum>>
UserType
CUSTOMER
TELLER
UNDEFINED

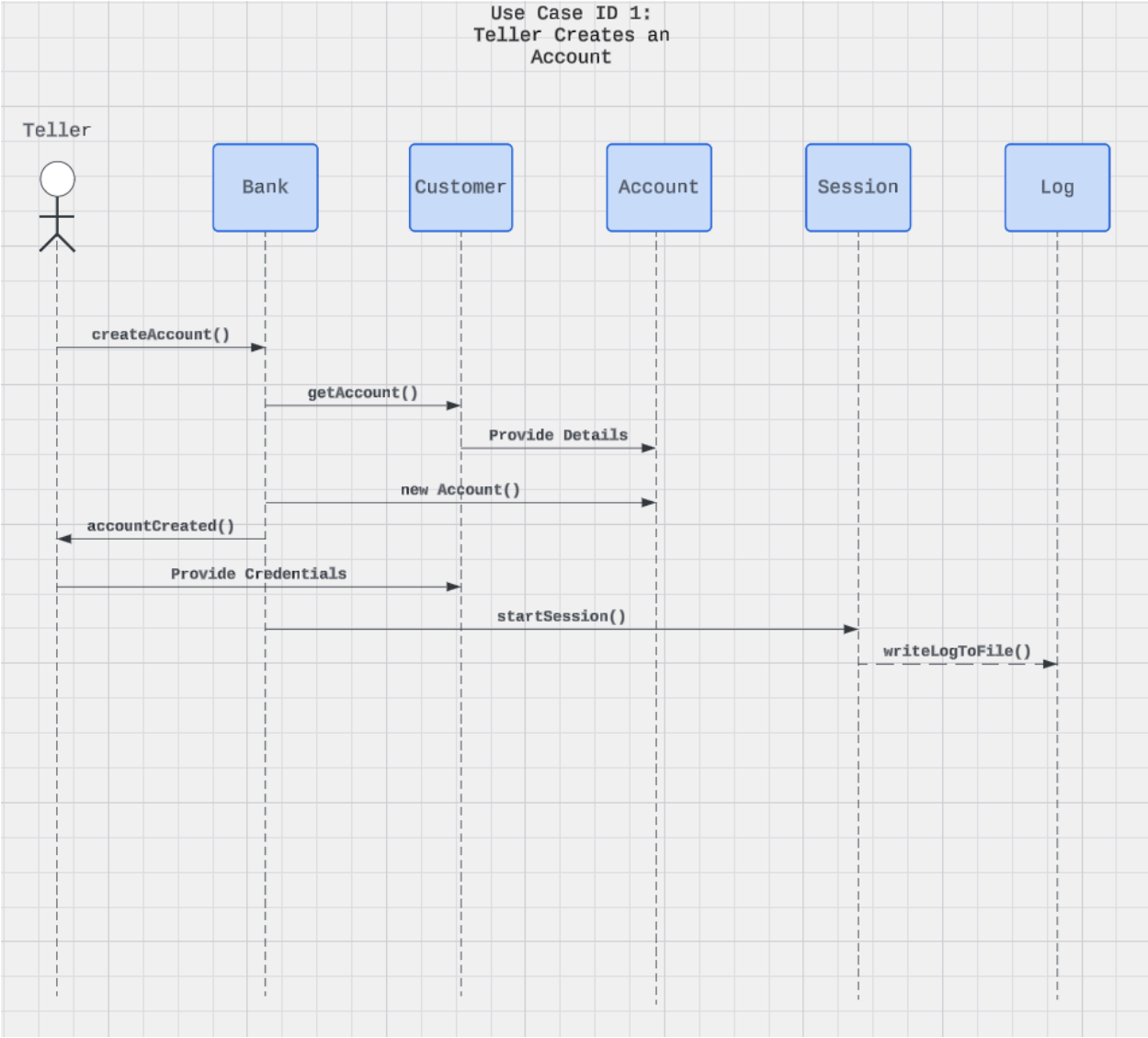

Account
-account_ID : int
-pin : String
-users : ArrayList<Customer>
-frozen : boolean
-amount : double
-type : AccountType
+checkCredentials(in account_ID : int, in pin : String) : boolean
+getAccountID() : int
+getUsers() : ArrayList
+addUser(in user : Customer) : void
+setPin(in pin : String) : void
+getPin() : String
+setFrozen(in freeze : boolean) : void
+getAmount() : double
+withDraw(in amt : double) : boolean
+deposit(in amt : double) : boolean
User
-id : int
-created : DateTime
+getID() : int
+getCreated() : DateTime
Customer
-accounts : ArrayList<Account>
+getAccount(in id : int, in pin : String, in index : int) : Account
Teller
+createAccount(in account_id : int, in pin : String) : Account
+freezeAccount(in act : Account, in time : Time) : boolean
+addAccountToCustomer(inout customer : Customer, in act : Account) : boolean
+readLog(in id : int) : String
+recoverPin(in act : Account) : String
+changePin(in act : Account) : void
+withdraw(in amount : double) : void
+deposit(in amount : double) : void
+transfer(in toAccount_id : int, in amount : double) : void

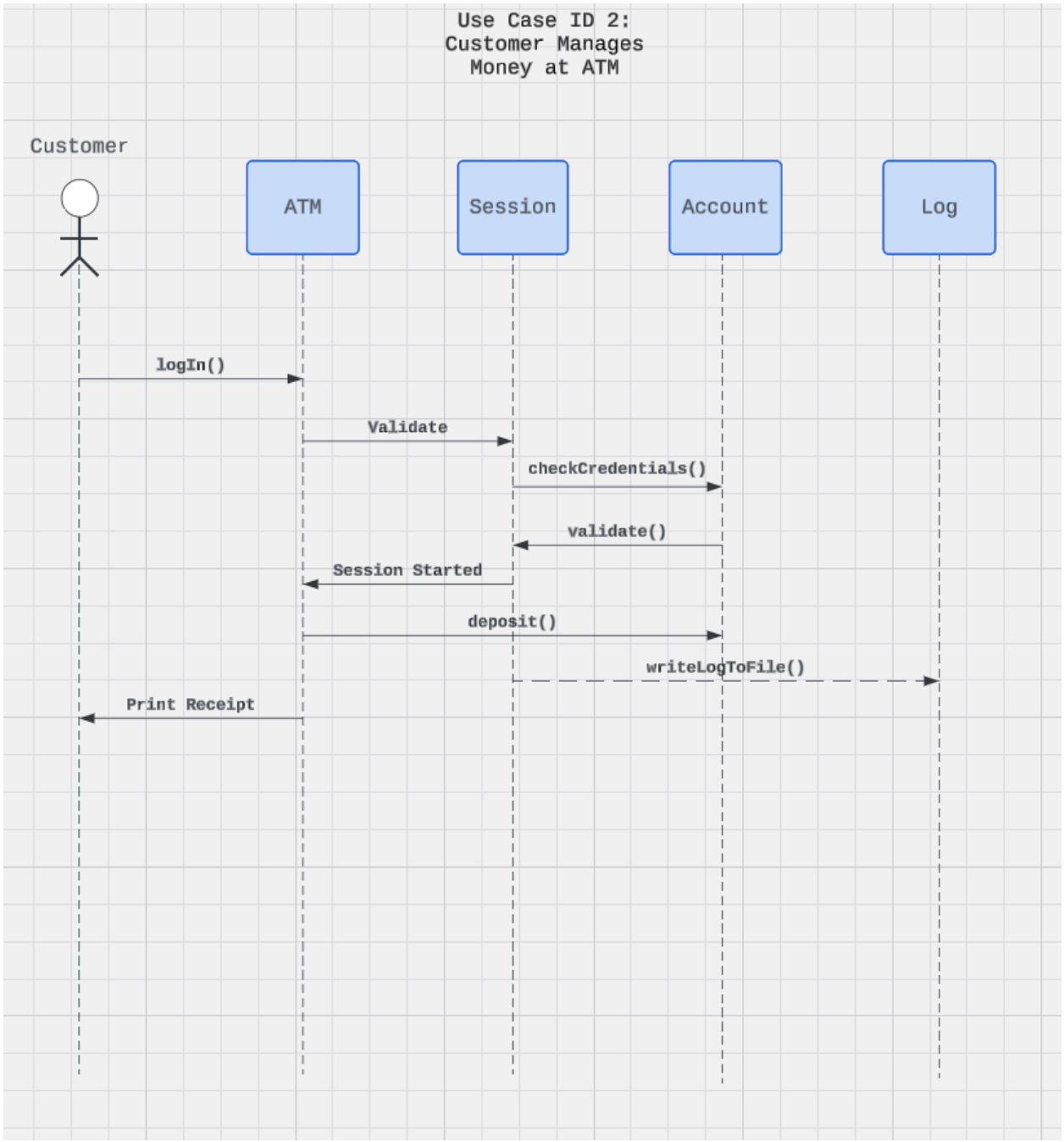

<<interface>>
Log
+writeLogToFile(inout file : File) : void
AccountCreationLog
-id : int
-teller_ID : int
-message : String
-timeStamp : DateTime
-AccountID: int
+writeLogToFile(inout file : File) : void
TransferLog
-id : int
-message : String
-timeStamp : DateTime
-AccountID: int
-toAccountID: int
+writeLogToFile(inout file : File) : void
SessionLog
-id : int
-message : String
-timeStart : DateTime
-timeEnd : DateTime
-AccountID: int
+writeLogToFile(inout file : File) : void
DepositLog
-id : int
-message : String
-timeStamp : DateTime
-AccountID: int
+writeLogToFile(inout file : File) : void
WithdrawLog
-id : int
-message : String
-timeStamp : DateTime
-AccountID: int
+writeLogToFile(inout file : File) : void

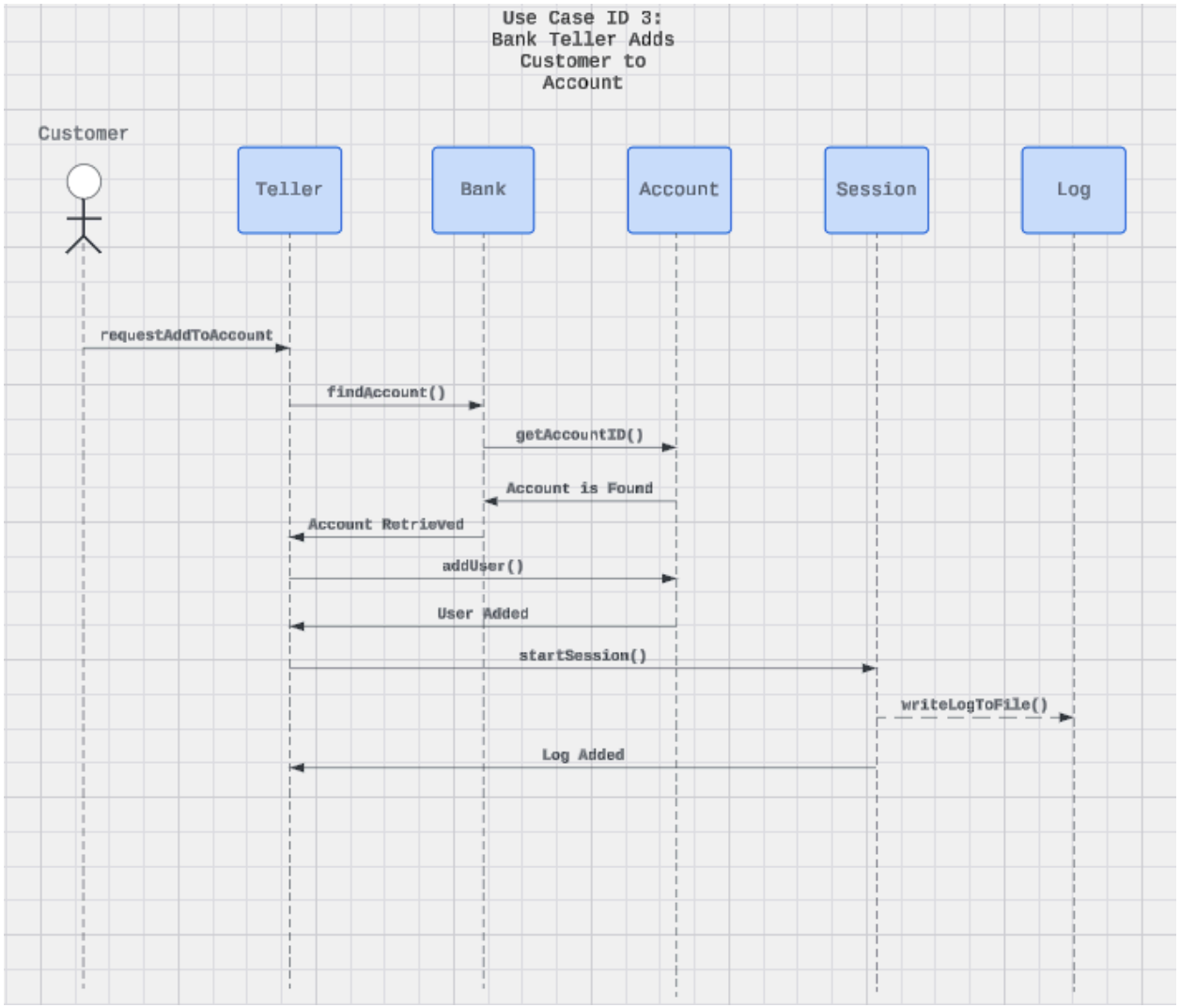

Backend
Server
1
<<thread>>
0...*
ClientHandler
-clientSocket : Socket
-bank : Bank
-userType : userType
-atm : ATM
-session : Session
-loggedIn : boolean
-userType : UserType
-teller : Teller
+run() : void
+processRequest(in req : List<Request>) : void
+doLogin(in req : Request) : void
+doDeposit(in req: Request) : void
+doTransfer(in req: Request) : void
+doWithdraw(in req: Request) : void
+doFreeze(in req: Request) : void
+doUnfreeze(in req: Request) : void
+doReadLogs(in req: Request) : void
+doEnter(in req: Request) : void
+doCreateAccount(in req: Request) : void
+doCreateCustomer(in req: Request) : void
+doLeave(in req: Request) : void
+doLogout(in req: Request) : void
ATM
-session : Session
+logIn(in account_id : int, in pin : String) : Session
+logOut() : void
+withdraw(in amount : double) : void
+deposit(in amount : double) : void
+getBal() : double
+transfer(in toAccount_id : int, in amount : double) : void
+printReceipt(in session : Session) : void
Bank
-accounts : ArrayList<Account>
-customers : ArrayList<Customer>
-tellers : ArrayList<Teller>
+findAccount(in account_id) : Account
+findCustomer(in customer_id) : Customer
+findTeller(in teller_id) : Teller
+setAccounts(in accountData : ArrayList<Account>) : void
+setCustomers(in customerData : ArrayList<Customer>) : void
+addAccount(in account : Account) : void
+addCustomer(in customer : Customer) : void
+addTeller(in teller : Teller) : void
+saveData(in customers : ArrayList<Customer>, in filePath : String) : void
+loadData(in customers : ArrayList<Customer>, in accounts : ArrayList<Account>, in filePath : String) : void
1
1
<<thread>>
<<thread>>
OutHandler
-outputStream : ObjectOutputStream
-requestQueue : ConcurrentLinkedQueue
-running : boolean
+enqueueRequest(in List<Request>) : void
+stop() : void
InHandler
-inputStream : ObjectInputStream
-requestQueue : ConcurrentLinkedQueue
-running : boolean
+getNextRequest() : List<Request>
+stop() : void

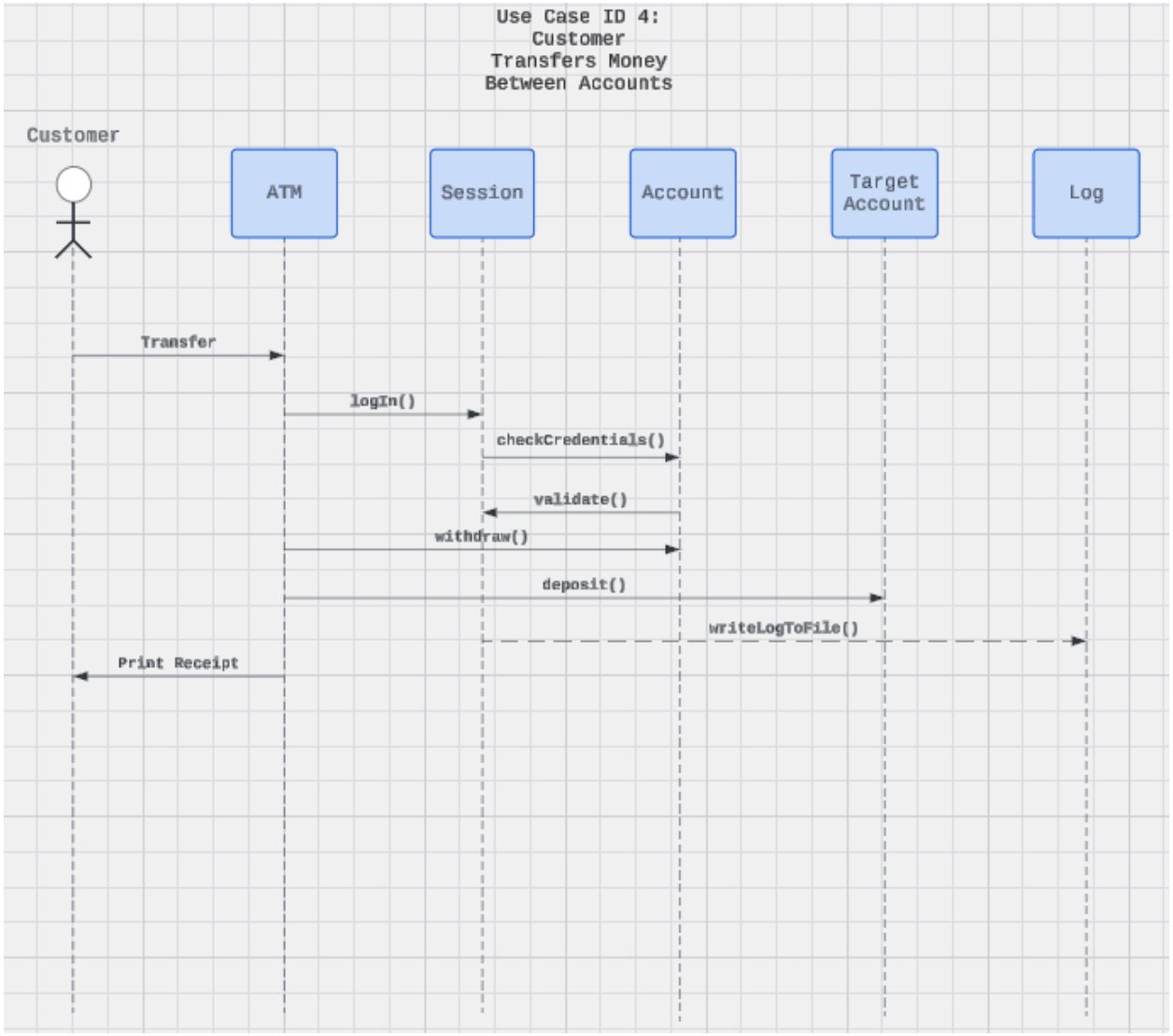

Frontend
Client
-alive : boolean
-loggedIn : boolean
-isProcessing : boolean
-responseMessage : String
-userType : UserType
-accountAccessed : boolean
-balance : double
-processResponse(in req : List<Request>) : void
-getUserType() : UserType
-getIsProcessing() : boolean
-getResponseMessage() : String
-getLoggedIn() : boolean
-getAccountAccessed() : boolean
-setBal(in amount : double) : void
-getBal() : double
-createLoginRequest(in username : String, in password : String) : void
-createDepositRequest(in amount : double) : void
-createWithdrawRequest(in amount : double) : void
-createTransferRequest(in toAccountID : int, in amount : double) : void
-createFreezeRequest(in acc_ID : int) : void
-createUnfreezeRequest(in acc_ID : int) : void
-createEnterAccountRequest(in acc_ID : int) : void
-createCustomerCreationRequest() : void
-createAccountCreationRequest(in password : String, in customerID : String) : void
-createLeaveRequest() : void
-createLogoutRequest() : void
-createLogoutRequest() : void
<<thread>>
GUI
-client : Client
+run() : void
<<thread>>
<<thread>>
OutHandler
-outputStream : ObjectOutputStream
-requestQueue : ConcurrentLinkedQueue
-running : boolean
+enqueueRequest(in List<Request>) : void
InHandler
-inputStream : ObjectInputStream
-requestQueue : ConcurrentLinkedQueue
-running : boolean
+getNextRequest() : List<Request>
+stop() : void

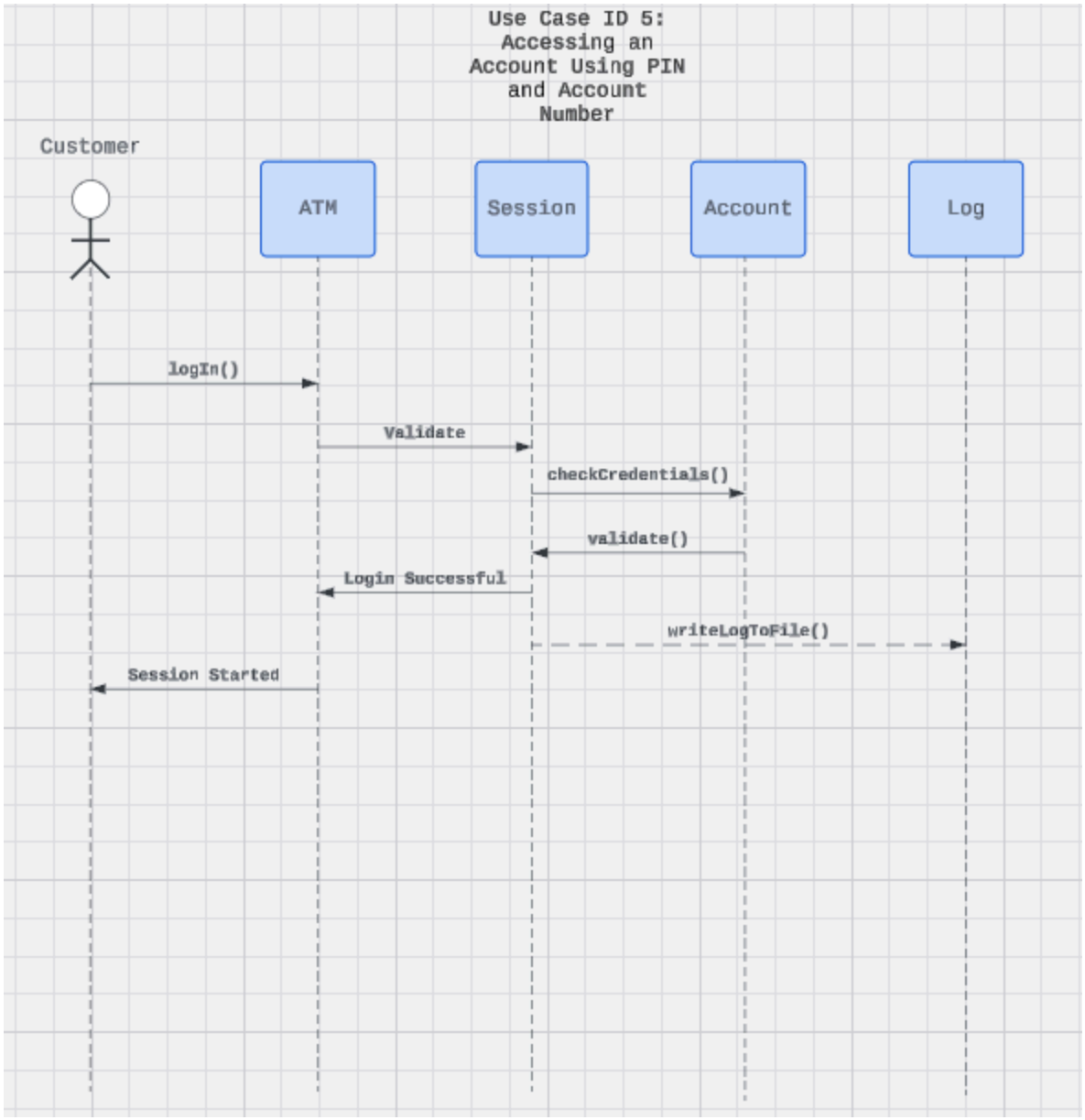
Banking System Software Sequence Diagrams

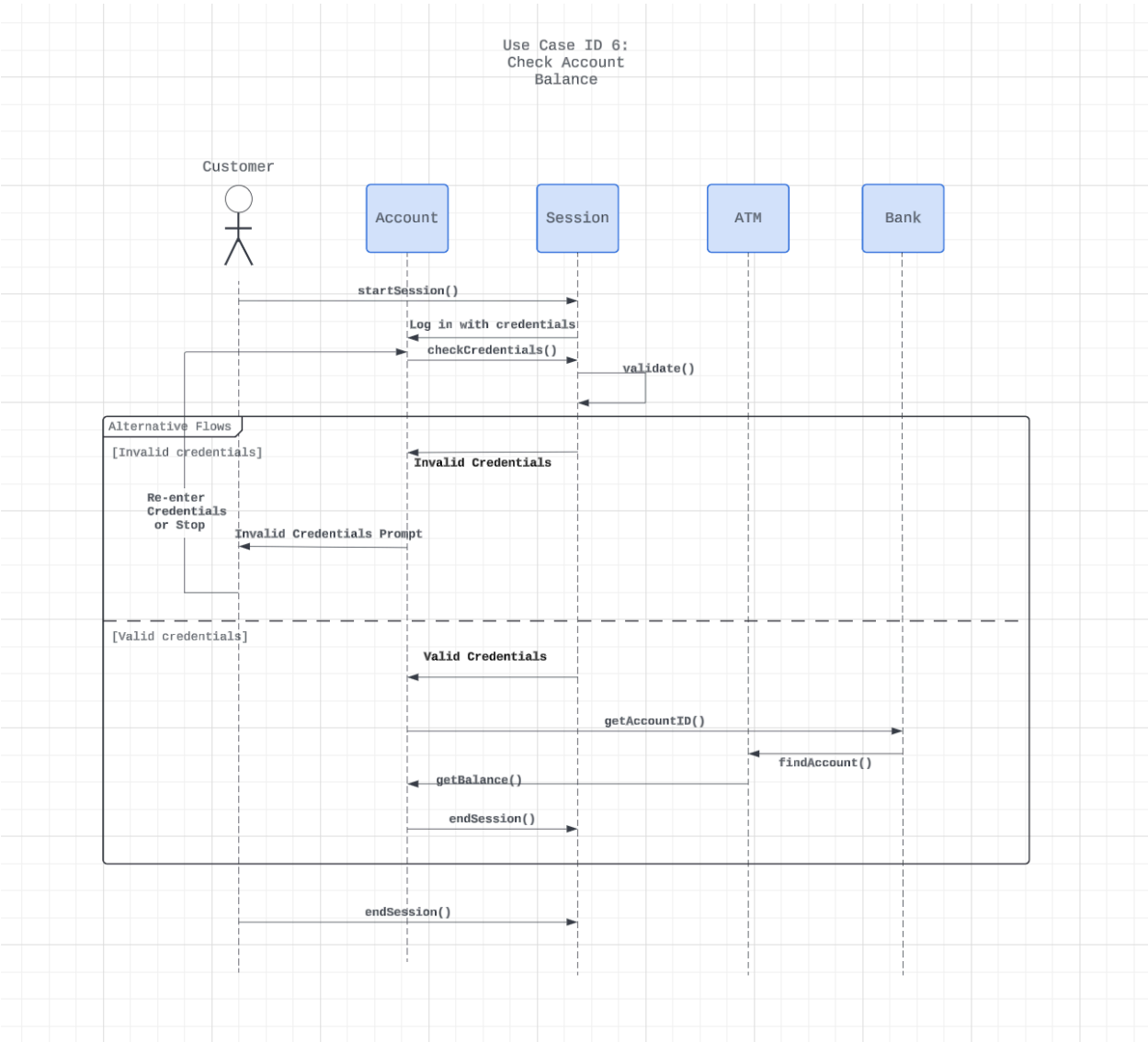


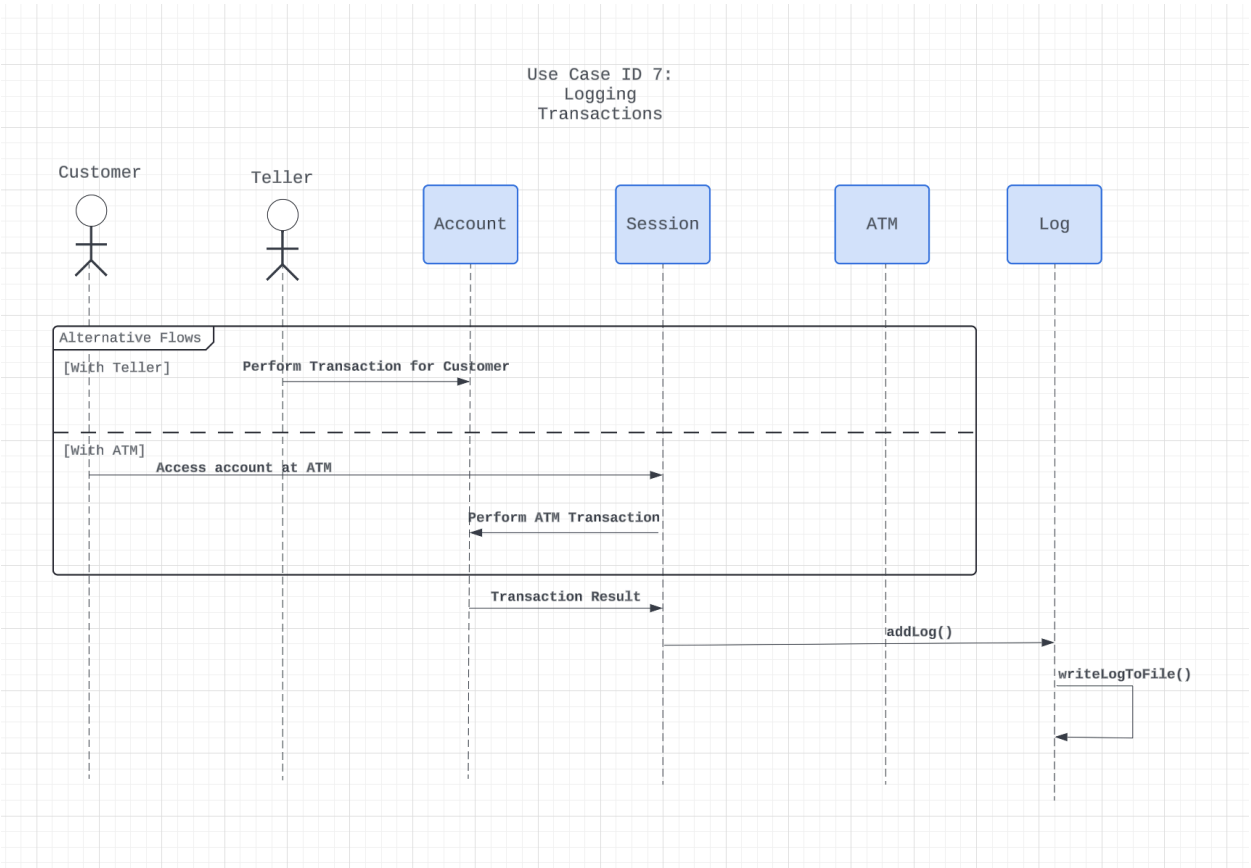

Use Case ID 2:
Customer Manages
Money at ATM
Customer
ATM
Session
Account
Log
logIn()
Validate
checkCredentials()
validate()
Session Started
deposit()
writeLogToFile()
Print Receipt

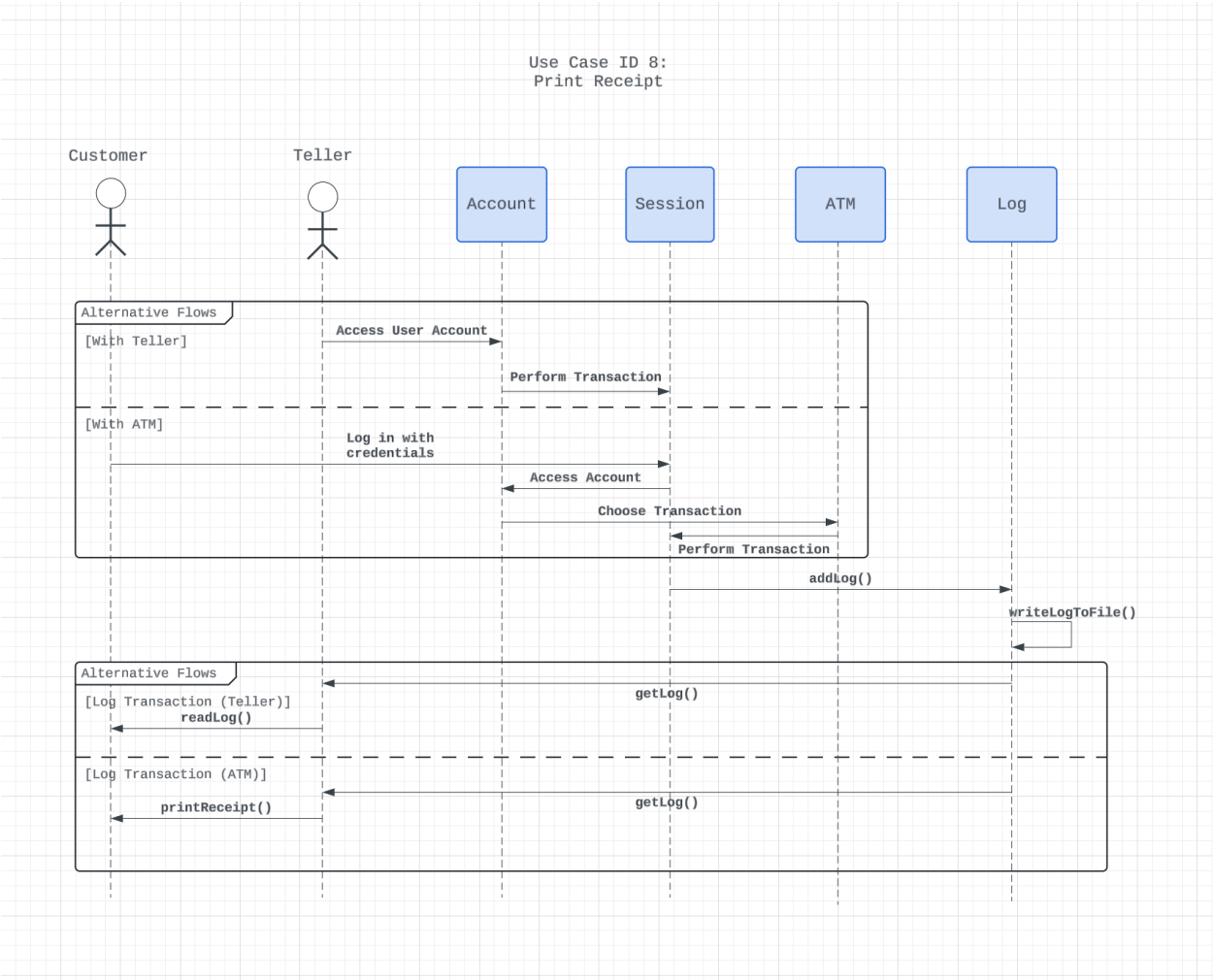

Use Case ID 3:
Bank Teller Adds
Customer to
Account
Customer
Teller
Bank
Account
Session
Log
requestAddToAccount
findAccount()
getAccountID()
Account is Found
Account Retrieved
addUser()
User Added
startSession()
writeLogToFile()
Log Added

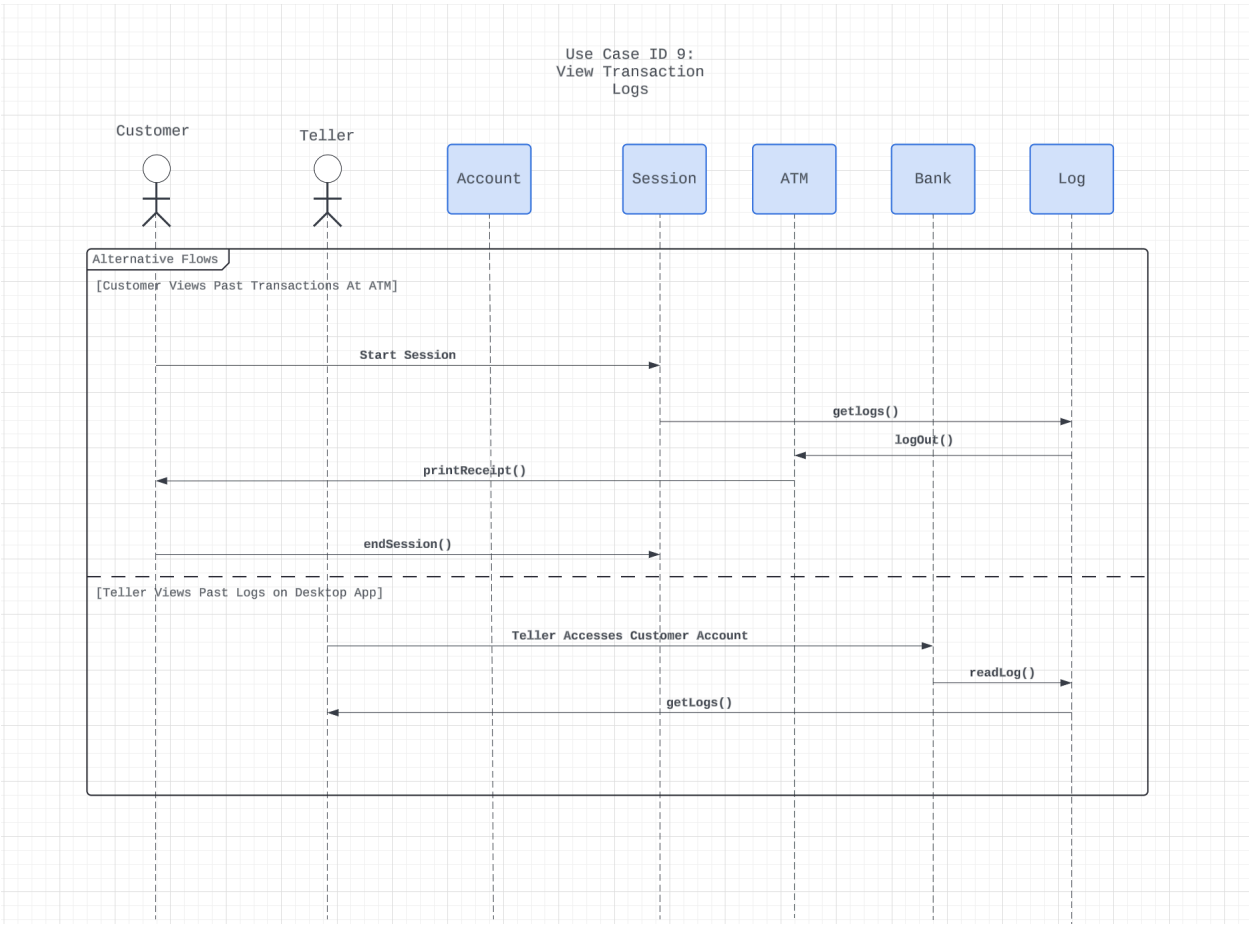

Use Case ID 4:
Customer
Transfers Money
Between Accounts
Customer
ATM
Session
Account
Target Account
Log
Transfer
logIn()
checkCredentials()
validate()
withdraw()
deposit()
writeLogToFile()
Print Receipt

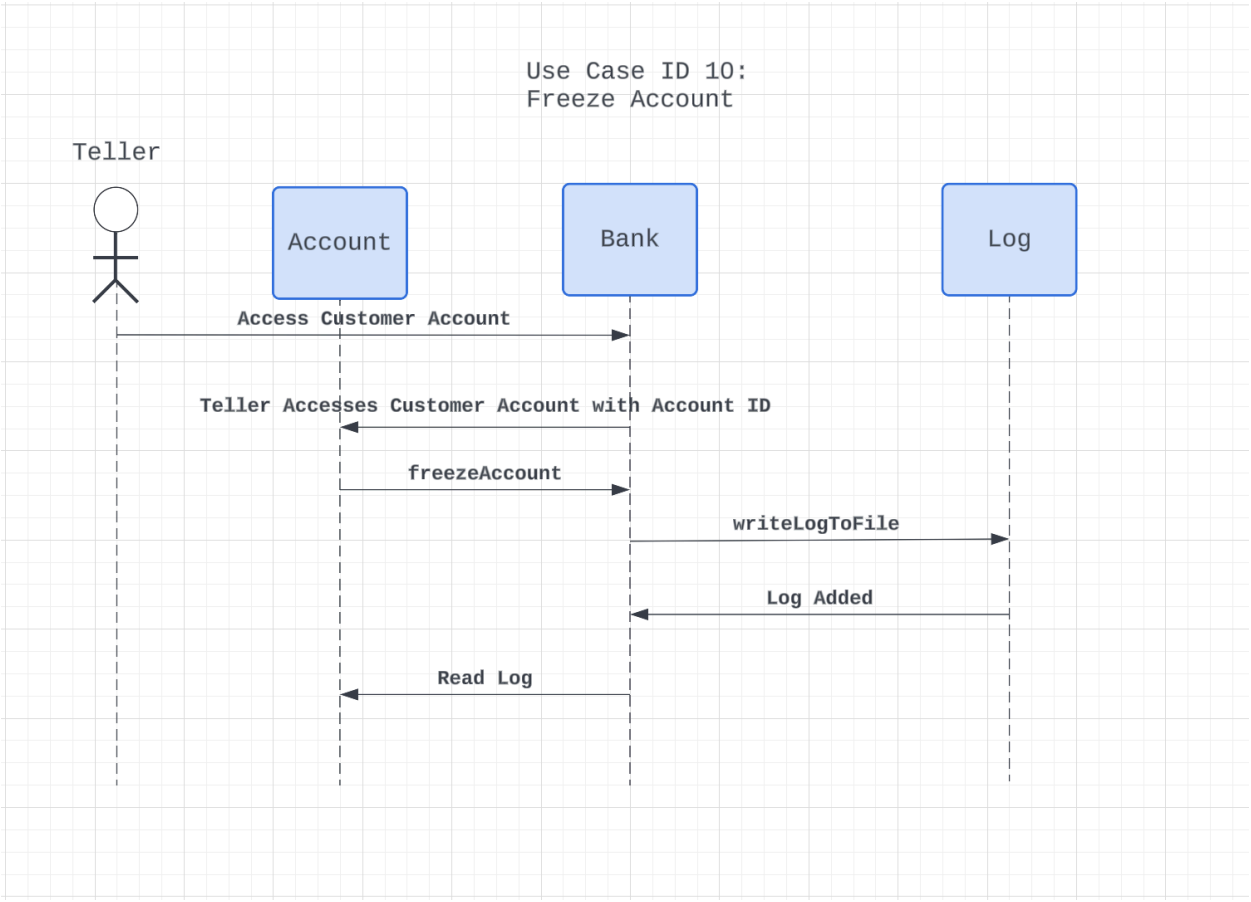

Use Case ID 5: Accessing an Account Using PIN and Account Number
Customer
ATM
Session
Account
Log
logIn()
Validate
checkCredentials()
validate()
Login Successful
WriteLogToFile()
Session Started


Use Case ID 6:
Check Account
Balance
Customer
Account
Session
ATM
Bank
startSession()
Log in with credentials
checkCredentials()
validate()
Alternative Flows
[Invalid credentials]
Invalid Credentials
Re-enter
Credentials
or Stop
Invalid Credentials Prompt
[Valid credentials]
Valid Credentials
getAccountID()
findAccount()
getBalance()
endSession()
endSession()

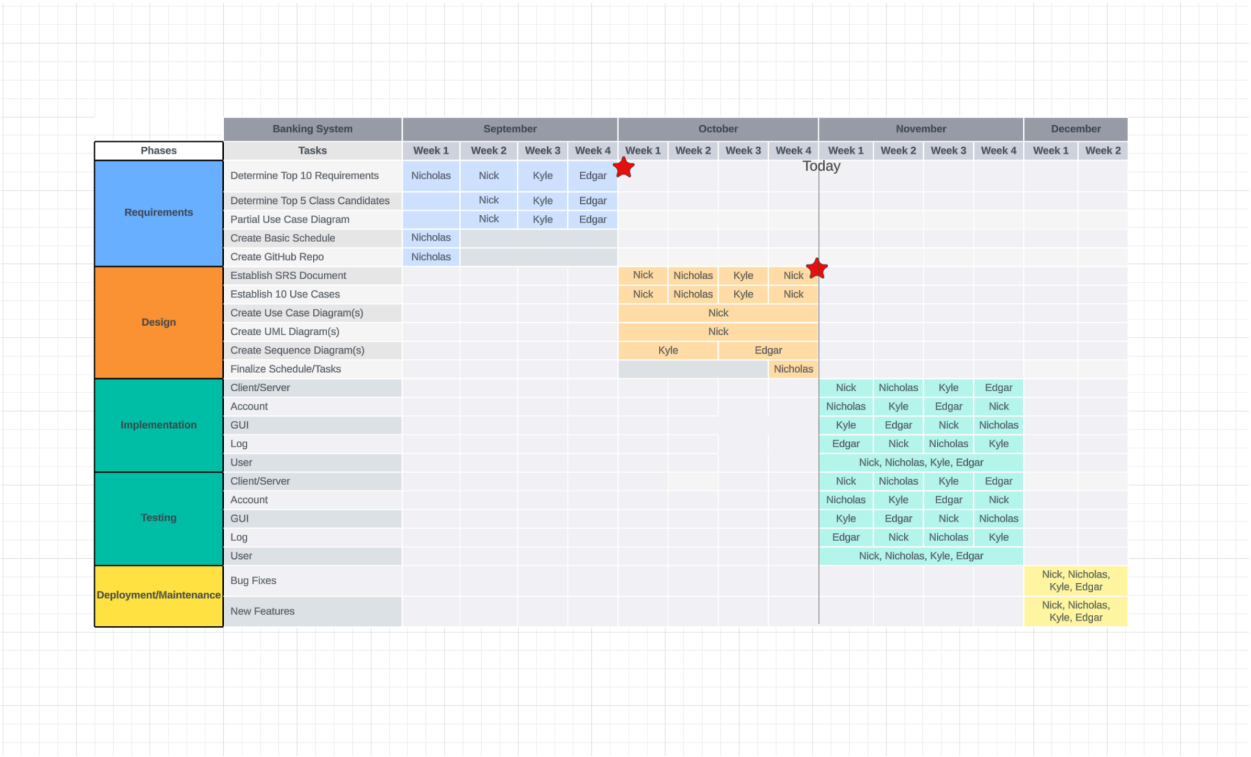

Use Case ID 7:
Logging
Transactions
Customer
Teller
Account
Session
ATM
Log
Alternative Flows
[With Teller]
Perform Transaction for Customer
[With ATM]
Access account at ATM
Perform ATM Transaction
Transaction Result
addLog()
writeLogToFile()


Use Case ID 8:
Print Receipt
Customer
Teller
Account
Session
ATM
Log
Alternative Flows
[With Teller]
Access User Account
Perform Transaction
[With ATM]
Log in with credentials
Access Account
Choose Transaction
Perform Transaction
addLog()
writeLogToFile()
Alternative Flows
[Log Transaction (Teller)]
getLog()
readLog()
[Log Transaction (ATM)]
getLog()
printReceipt()


Use Case ID 9:
View Transaction
Logs
Customer
Teller
Account
Session
ATM
Bank
Log
Alternative Flows
[Customer Views Past Transactions At ATM]
Start Session
getlogs()
logOut()
printReceipt()
endSession()
[Teller Views Past Logs on Desktop App]
Teller Accesses Customer Account
readLog()
getLogs()


Use Case ID 10:
Freeze Account
Teller
Account
Bank
Log
Access Customer Account
Teller Accesses Customer Account with Account ID
freezeAccount
writeLogToFile
Log Added
Read Log

Gantt Chart



Phases	Banking System	September				October				November				December	
	Tasks	Week 1	Week 2	Week 3	Week 4	Week 1	Week 2	Week 3	Week 4	Week 1	Week 2	Week 3	Week 4	Week 1	Week 2
Requirements	Determine Top 10 Requirements	Nicholas	Nick	Kyle	Edgar				Today						
	Determine Top 5 Class Candidates		Nick	Kyle	Edgar										
	Partial Use Case Diagram		Nick	Kyle	Edgar										
	Create Basic Schedule	Nicholas													
	Create GitHub Repo	Nicholas													
Design	Establish SRS Document					Nick	Nicholas	Kyle	Nick						
	Establish 10 Use Cases					Nick	Nicholas	Kyle	Nick						
	Create Use Case Diagram(s)					Nick									
	Create UML Diagram(s)					Nick									
	Create Sequence Diagram(s)					Kyle		Edgar							
	Finalize Schedule/Tasks								Nicholas						
Implementation	Client/Server									Nick	Nicholas	Kyle	Edgar		
	Account									Nicholas	Kyle	Edgar	Nick		
	GUI									Kyle	Edgar	Nick	Nicholas		
	Log									Edgar	Nick	Nicholas	Kyle		
	User									Nick, Nicholas, Kyle, Edgar					
Testing	Client/Server									Nick	Nicholas	Kyle	Edgar		
	Account									Nicholas	Kyle	Edgar	Nick		
	GUI									Kyle	Edgar	Nick	Nicholas		
	Log									Edgar	Nick	Nicholas	Kyle		
	User									Nick, Nicholas, Kyle, Edgar					
Deployment/Maintenance	Bug Fixes													Nick, Nicholas, Kyle, Edgar	
	New Features													Nick, Nicholas, Kyle, Edgar	