CS-401_Group Project Design

Design Documentation

Revision History

Date	Revision	Description	Author
10/30	0.1	Initiate document, skeletal frame format	Phuong
10/30	0.2	Added use cases, all the available diagrams	Rohan

1. Purpose

This document outlines the classes and their interactions.

1.1. Scope

This document provides design for a Large Banking System. This software will support countless people and interactions. The primary objective being, providing a GUI for the bank employees and users to interact with their accounts. The bank has a server that stores all the data and is retrieved through the messages between the server and the clients.

1.2. Definitions, Acronyms, Abbreviations

List any acronyms, terms etc. that need to be defined.
SU - SuperUser, refers to the class of user with more permissions and functionalities.
GUI - Graphical user interface.
User - A regular account user. Has the ability to draw or add funds from their account.
Joint Account - Multiple users can share the same account.
Client - Module for client code.
Host - Module for server code.
Message - Module for data transfer between client and server.
JVM - Java Virtual Machine.

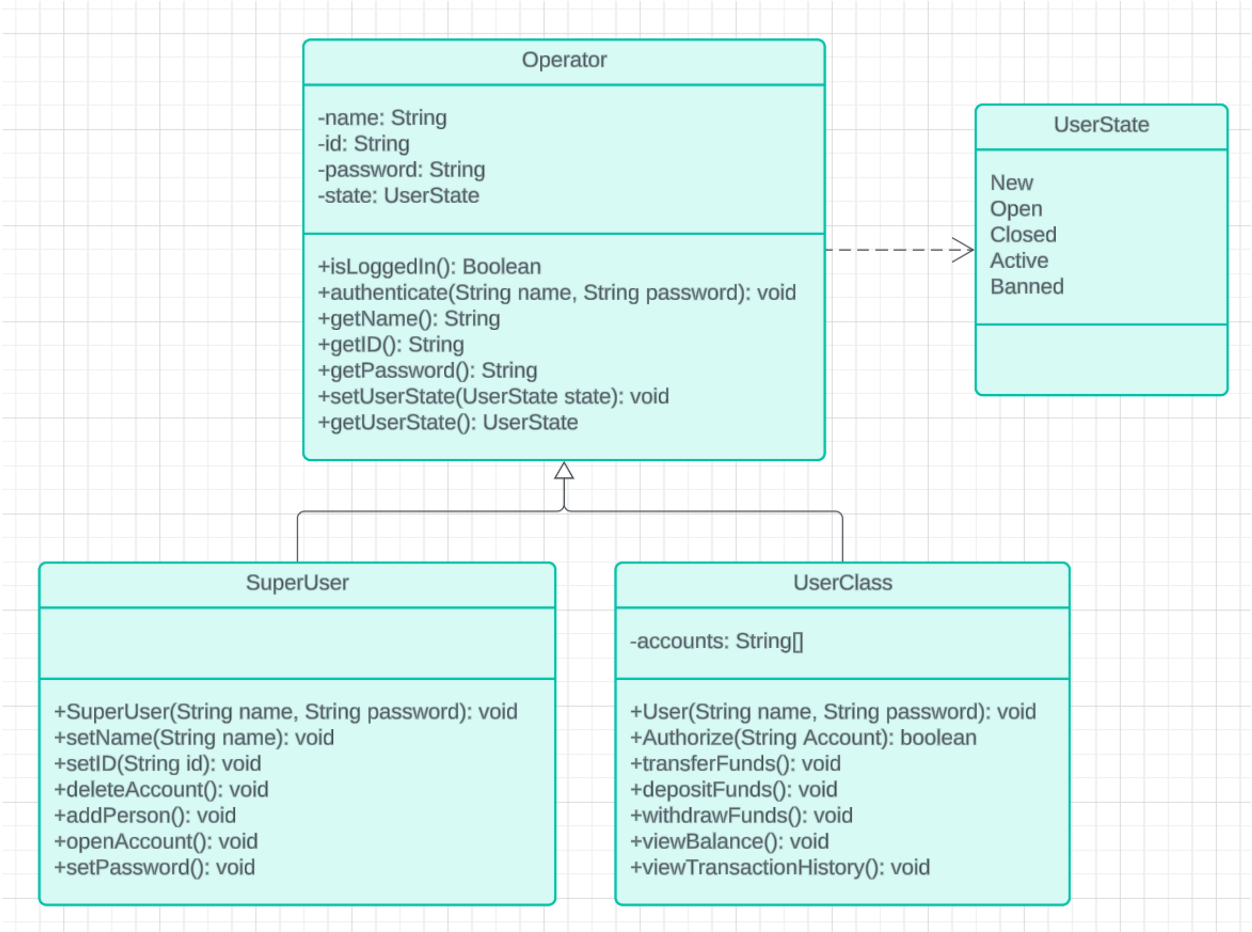
1.3. References

SRS documentation - Requirements

2. Class candidates

2.1. Operator module

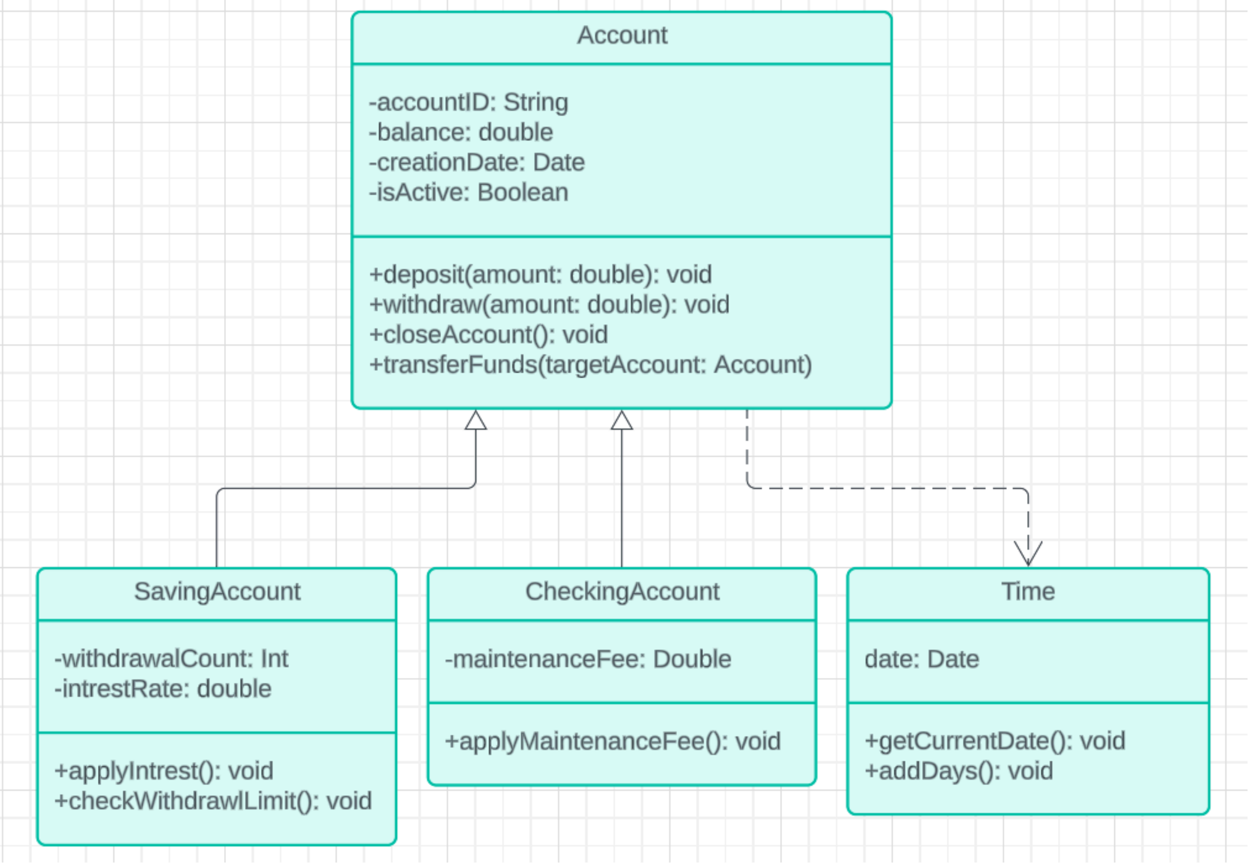
Operator module consists of the User class and its child class SuperUser.
The main functionality of User is to authenticate a user and provide a reference to what account authorized by said user.
SuperUser can also create a User object and manipulate its data, including the reference to authorized accounts.



2.2. Account module

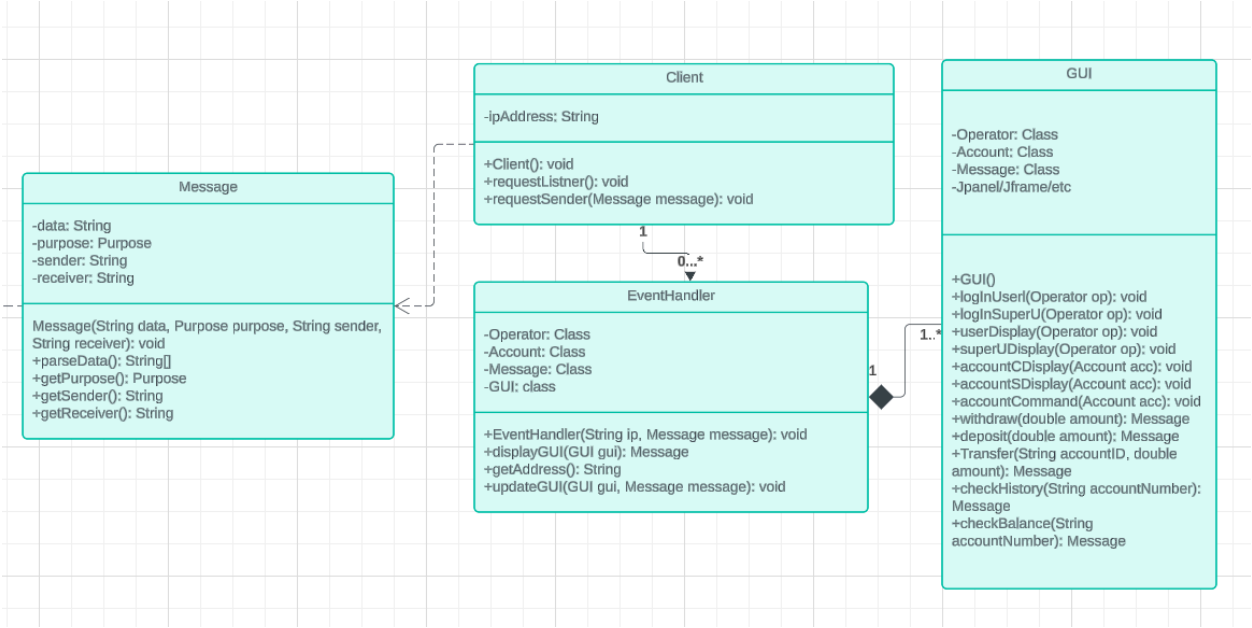
Account module consists of the CheckingAccount class and its child class SavingAccount. CheckingAccount keeps track of the state and balance of an account, providing methods to manipulate these data.

SavingAccount is a CheckingAccount with an additional method to update its balance based on a rate, and the time since last update (server side clock).



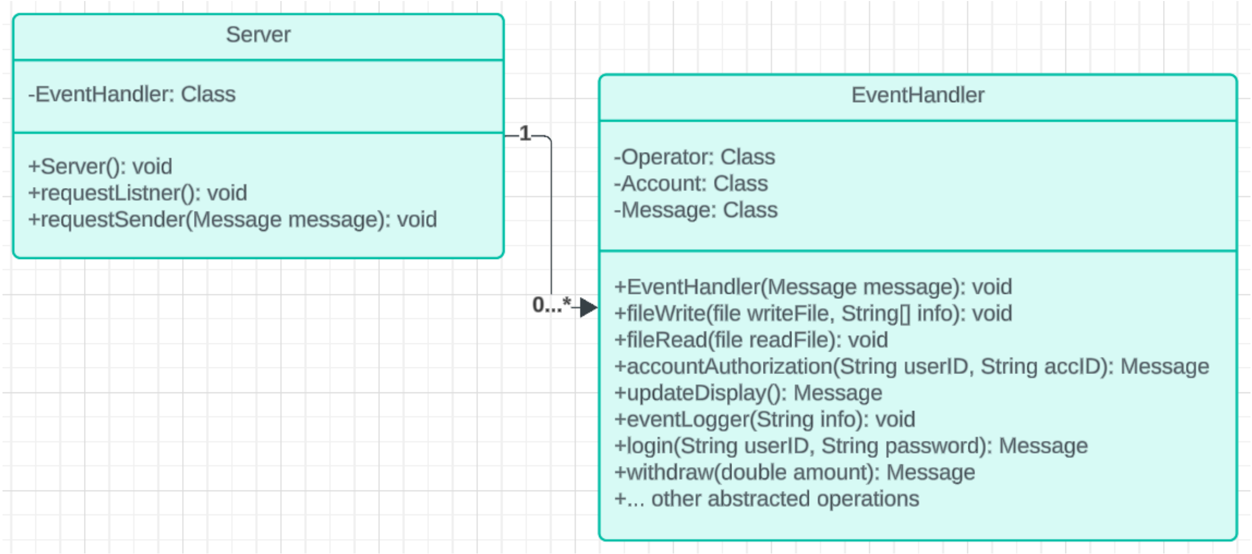
2.3. Client module

Client module consist of the Client class, its facade the ClientHandler, and the GUI class.
Client Class will handle the multithread network communication with the server using Message class, while holding data from these messages.
GUI will display data held by client, and also provide client with data of the user's interactions, which will be processed into a message to be used by client class' network communicator by the ClientHandler.


Client
-ipAddress: String
+Client(): void
+requestListner(): void
+requestSender(Message message): void
GUI
-Operator: Class
-Account: Class
-Message: Class
-Jpanel/Jframe/etc
+GUI()
+logInUserl(Operator op): void
+logInSuperU(Operator op): void
+userDisplay(Operator op): void
+superUDisplay(Operator op): void
+accountCDisplay(Account acc): void
+accountSDisplay(Account acc): void
+accountCommand(Account acc): void
+withdraw(double amount): Message
+deposit(double amount): Message
+Transfer(String accountID, double amount): Message
+checkHistory(String accountNumber): Message
+checkBalance(String accountNumber): Message
Message
-data: String
-purpose: Purpose
-sender: String
-receiver: String
Message(String data, Purpose purpose, String sender, String receiver): void
+parseData(): String[]
+getPurpose(): Purpose
+getSender(): String
+getReceiver(): String
1
0...*
EventHandler
-Operator: Class
-Account: Class
-Message: Class
-GUI: class
+EventHandler(String ip, Message message): void
+displayGUI(GUI gui): Message
+getAddress(): String
+updateGUI(GUI gui, Message message): void
1..*
1

2.4. Server module

Server module consists of the Server class and its facade the RequestHandler.
Server class main functionality is to handle the network communication.
Server class will fork threads of RequestHandler class to handle incoming requests.
RequestHandler will be able to decapsulate and call appropriate methods based on the content of the message received, and log down appropriately.

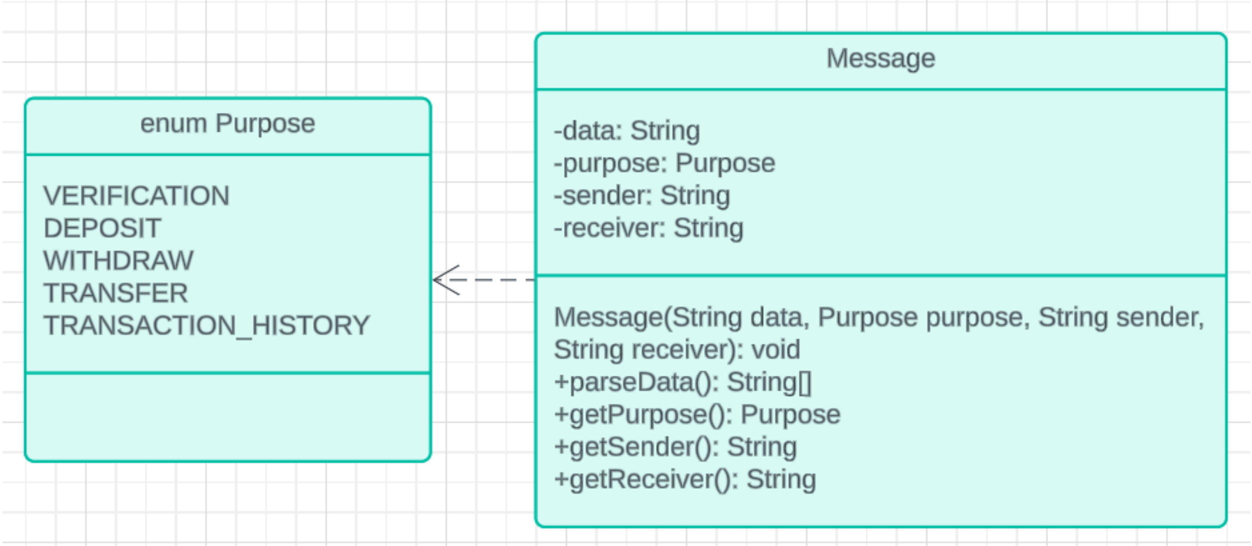

Server
-EventHandler: Class
+Server(): void
+requestListner(): void
+requestSender(Message message): void
1
0...*
EventHandler
-Operator: Class
-Account: Class
-Message: Class
+EventHandler(Message message): void
+fileWrite(file writeFile, String[] info): void
+fileRead(file readFile): void
+accountAuthorization(String userID, String accID): Message
+updateDisplay(): Message
+eventLogger(String info): void
+login(String userID, String password): Message
+withdraw(double amount): Message
+... other abstracted operations

2.5. Message module

Message module consists of a message class.

Message class will contain the data that will be transmitted, together with an enum of the type of transmission, the address of from and to, and time of creation.

Message class will be able to parse the data string into an array of string that can be used as arguments easily.



3. Use Cases

3.1. GUI Use Cases

3.1.1. G001: Operator run Client

Use Case ID: G001

Use Case Name: Operator run Client

Relevant Requirements:

Primary Actor:

- GUI

Pre-condition:

- The executable for the client is successfully launch

Post-condition:

- Operator is led through interfaces to finish logging in and able to access features.

Basic Flow:

1. GUI shows operator log in interface.
2. GUI runs the client's facade method to handle authentication.
3. GUI receives response from method, which will contain data.
4. GUI runs client facade's method to parse and process data.
5. Client's facade runs GUI's method to display data to the user.

Alternate Flows:

1 - 4. same as basic Flow

2. Client's facade detects superuser through regex the operator id.

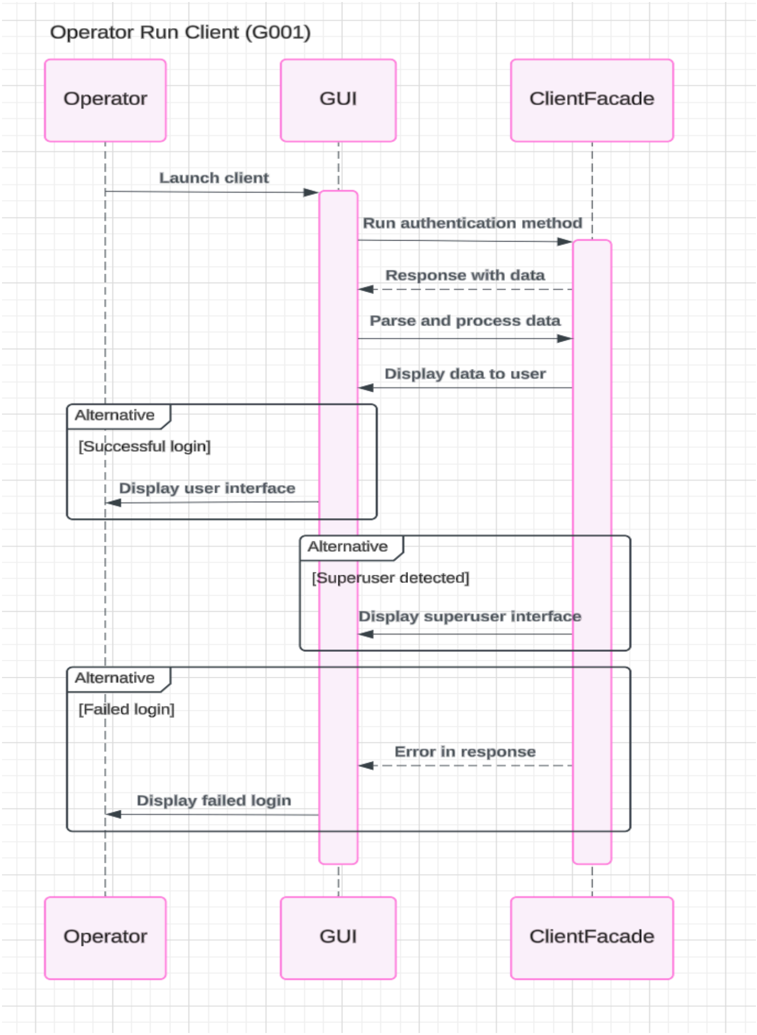
3. Client's facade runs a GUI method to display superuser's interface.

Exceptions:

- If response data contain failed login exceptions, display failed login GUI.

Related Use Cases:

- CS003, CS001, CS002


Operator Run Client (G001)
Operator
GUI
ClientFacade
Launch client
Run authentication method
Response with data
Parse and process data
Display data to user
Alternative
[Successful login]
Display user interface
Alternative
[Superuser detected]
Display superuser interface
Alternative
[Failed login]
Error in response
Display failed login
Operator
GUI
ClientFacade

3.1.2 G002: Superuser doing some work on User behalf

Use Case ID: G002

Use Case Name: Operator at the Banker table

Relevant Requirements:

Primary Actor:

- Operator Module

Pre-condition:

- Client launch successful.

Post-condition:

- Super users have access to user accounts.

Basic Flow:

1. Superuser initiates login on client.
2. Client follows login process, detects a superuser ID.
3. Client shows superuser GUI.
4. GUI includes an additional log in panel, with the option to create a user.
5. Superuser enters user authenticator, client will use the same method as if an user is attempting to log in.
6. Upon 2nd log in success, GUI display user's GUI alongside superuser GUI.
7. Each log ins are logged.

Alternate Flows:

1-3. as basic

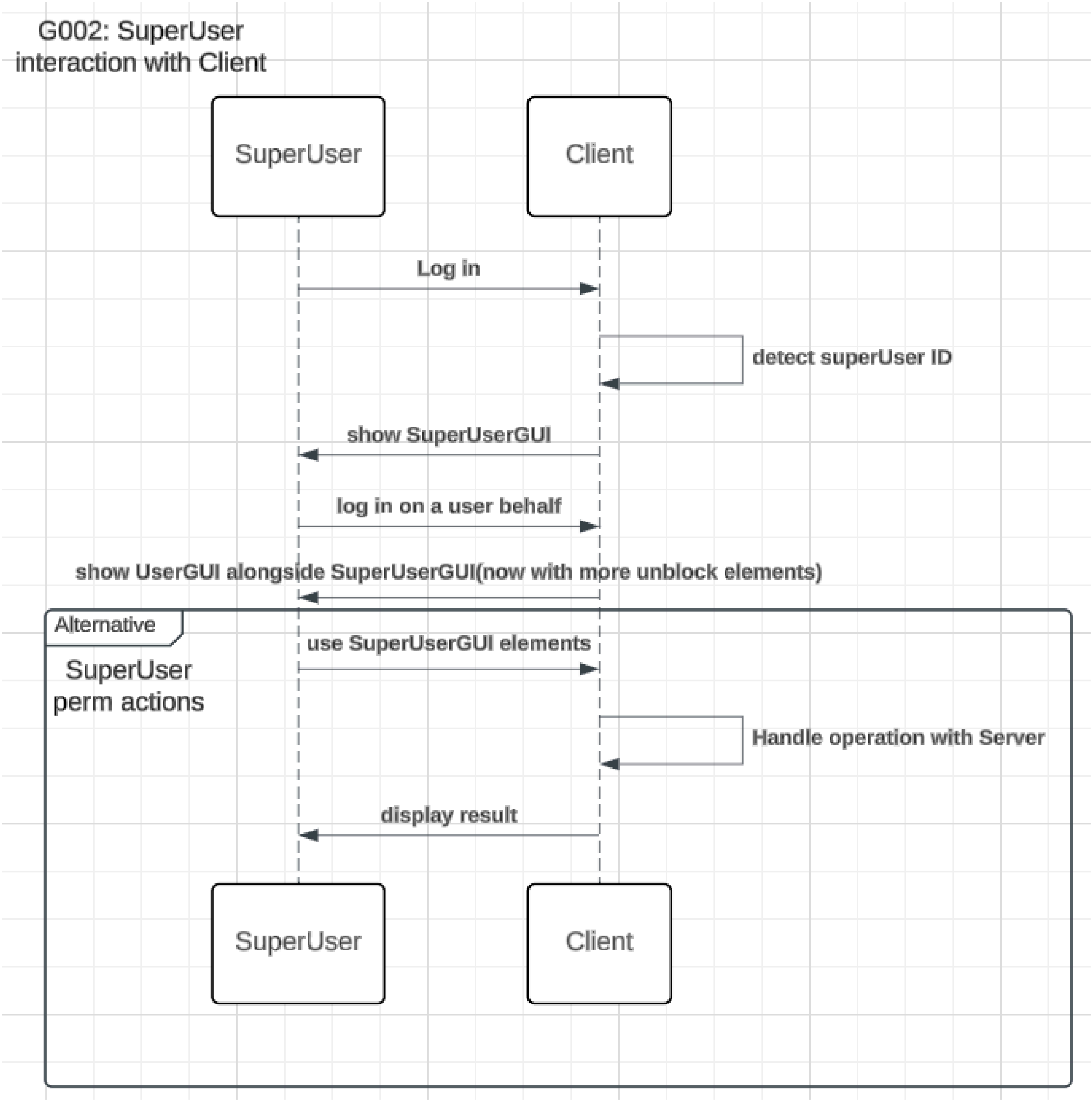
4. Superuser select element to create user.
5. GUI show textfield similar to log in, take 2 string user id and password.
6. Facade's create user method is called.
7. Once user is created, GUI shows a new user with an empty list of accounts, ready to be populated.

Exceptions:

- Superuser id authenticate failed.

Related Use Cases:

- CS001, CS002, CS003, G001


G002: SuperUser interaction with Client
SuperUser
Client
Log in
detect superUser ID
show SuperUserGUI
log in on a user behalf
show UserGUI alongside SuperUserGUI(now with more unblock elements)
Alternative
SuperUser perm actions
use SuperUserGUI elements
Handle operation with Server
display result
SuperUser
Client

3.2. Client/Server Use Cases

3.2.1 Use Case ID: CS001

Use Case Name: Send message from one host

Relevant Requirements: 3.1.4.1, 3.1.4.2, 3.1.

Primary Actor:

- Server Module
- Client Module

Pre-condition:

- Server or Client have some data for a specific request.

Post-condition:

- The data is sent to the recipient with the correct enum flag.

Basic Flow:

1. The server or client makes a request by passing the data, a corresponding enum flag, and the destination address to the message module.
2. The server or client sends the created message object to the address on the message object.

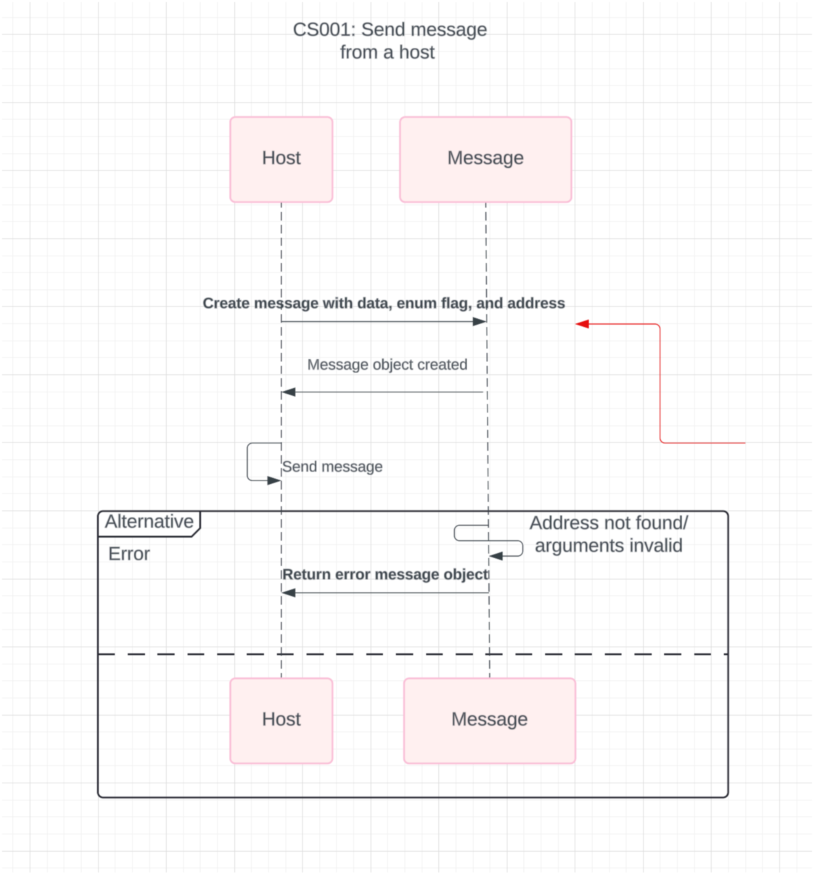
Alternate Flows:

Exceptions:

- The recipient address is not known to the server or client.

Related Use Cases:

- M001


CS001: Send message
from a host
Host
Message
Create message with data, enum flag, and address
Message object created
Send message
Alternative
Error
Address not found/
arguments invalid
Return error message object
Host
Message

3.1.2 Use Case ID: CS002

Use Case Name: Receive message from one host

Relevant Requirements: 3.1.4.1, 3.1.4.2, 3.1.4.3, 3.1.4.5.

Primary Actor:

- Client Module

Pre-condition:

- Port listeners detect a connection.

Post-condition:

- A request is processed.

Basic Flow:

1. Port listeners detect a connection.
2. Port listeners fork a thread to handle this connection.
3. Thread call facade method to handle this message object contained by the inbound connection.
4. Facade check message enum, pass parsed data as arguments on correct facade method based on the enum.

Alternate Flows:

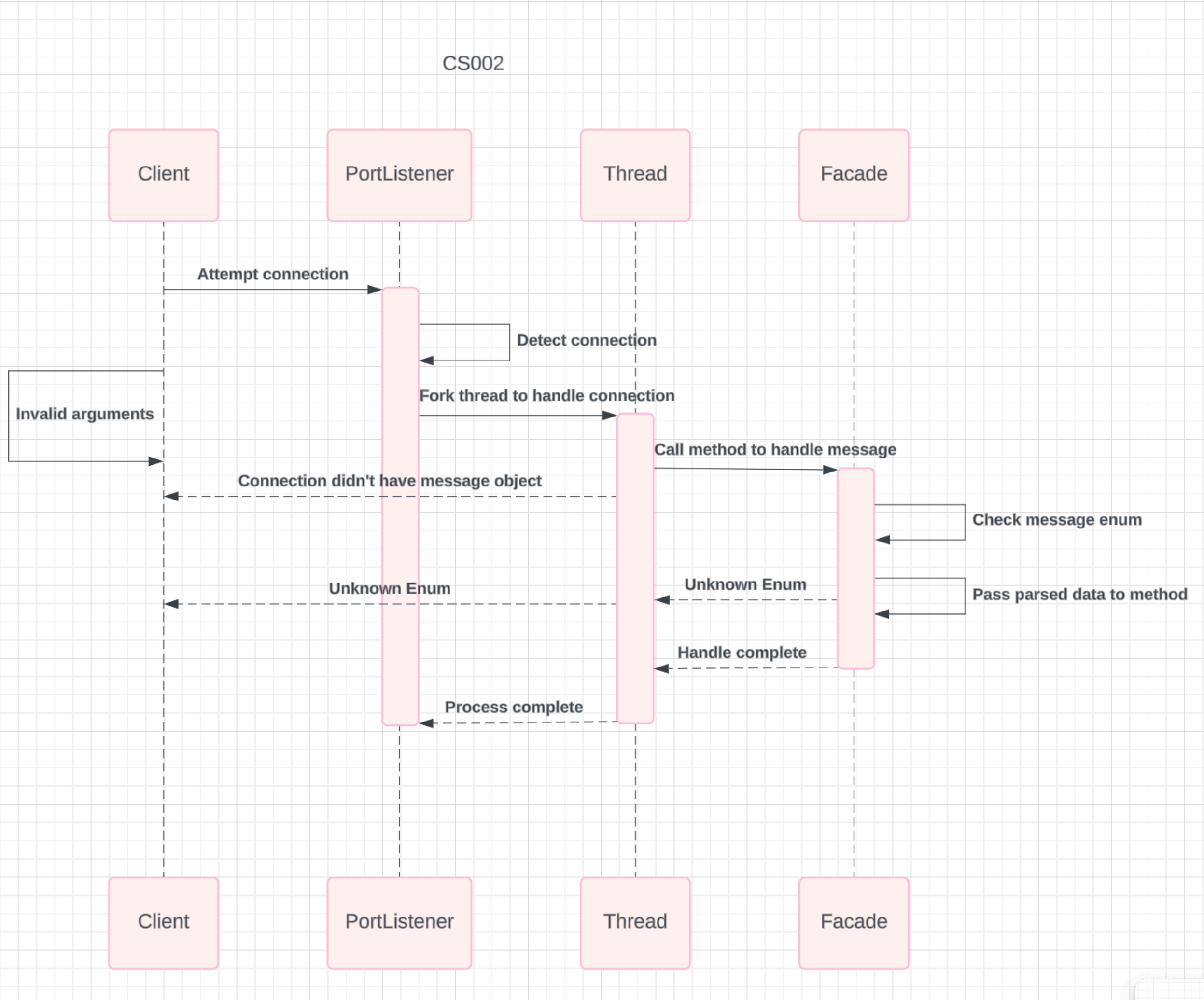
- None

Exceptions:

- Enum is unknown to the facade.
- Connection does not have a message object.
- Message does not contain correct arguments for the determined facade method.

Related Use Cases:

- M002


CS002
Client
PortListener
Thread
Facade
Attempt connection
Detect connection
Invalid arguments
Fork thread to handle connection
Call method to handle message
Connection didn't have message object
Check message enum
Unknown Enum
Unknown Enum
Pass parsed data to method
Handle complete
Process complete
Client
PortListener
Thread
Facade

3.1.3 Use Case ID: CS003

Use Case Name: Log in
Relevant Requirements: 3.1.2.1, 3.1.2.2, 3.1.2.3
Primary Actor:

- Client Module

Pre-condition:

- The operator is not authenticated.

Post-condition:

- The operator is authenticated.

Basic Flow:

1. Client shows the operator a login interface.
2. The operator enters their authenticator.
3. Client invokes a message to send a request to Server using authenticator.
4. Account is authenticated by the server.

Alternate Flows:

1. The authentication failed.
2. Server denies request.
3. Client invoke correct GUI response.
4. Events are logged

1. entered authenticator belongs to superuser

2-4. as basic Flow

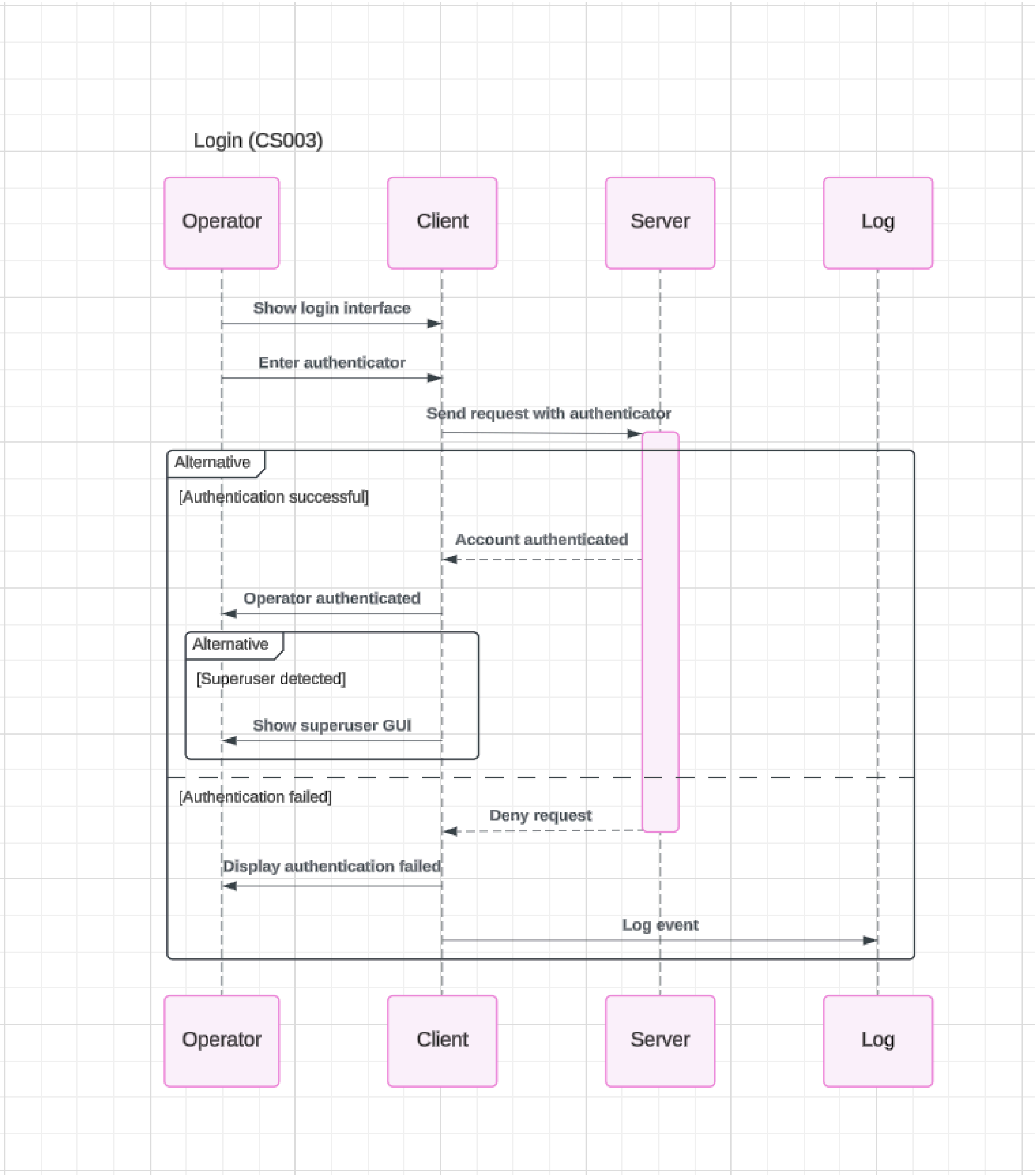
7. show superuser GUI

Exceptions:

- If the operator doesn't exist, the response is the same as if the authentication failed.

Related Use Cases:

- CS001, CS002


Login (CS003)
Operator
Client
Server
Log
Show login interface
Enter authenticator
Send request with authenticator
Alternative
[Authentication successful]
Account authenticated
Operator authenticated
Alternative
[Superuser detected]
Show superuser GUI
[Authentication failed]
Deny request
Display authentication failed
Log event
Operator
Client
Server
Log

3.1.4 Use Case ID: CS004

Use Case Name: Account authorization

Relevant Requirements: 3.1.2.4.6

Primary Actor:

- Operator module

Pre-condition:

- Server receives a message object.
- message object enum is not logIn
- Message object contains an operator id and an account id among its parsed arguments

Post-condition:

- The operator is authorized.

Basic Flow:

1. Server message handler thread detects non-logIn enum.
2. Facade's method must have the first method call be an authorized method with operator id and an account id as arguments.
3. Authorize method returns true, facade method continues operation.
4. Authorization success logged.

Alternate Flows:

1. Authorize method return false.
2. Facade method halt, return with authorization fails exception.
3. Facade calls a method to send authorization errors together with update display data for clients.
4. Clients handle response and update displays.
5. Events are logged.

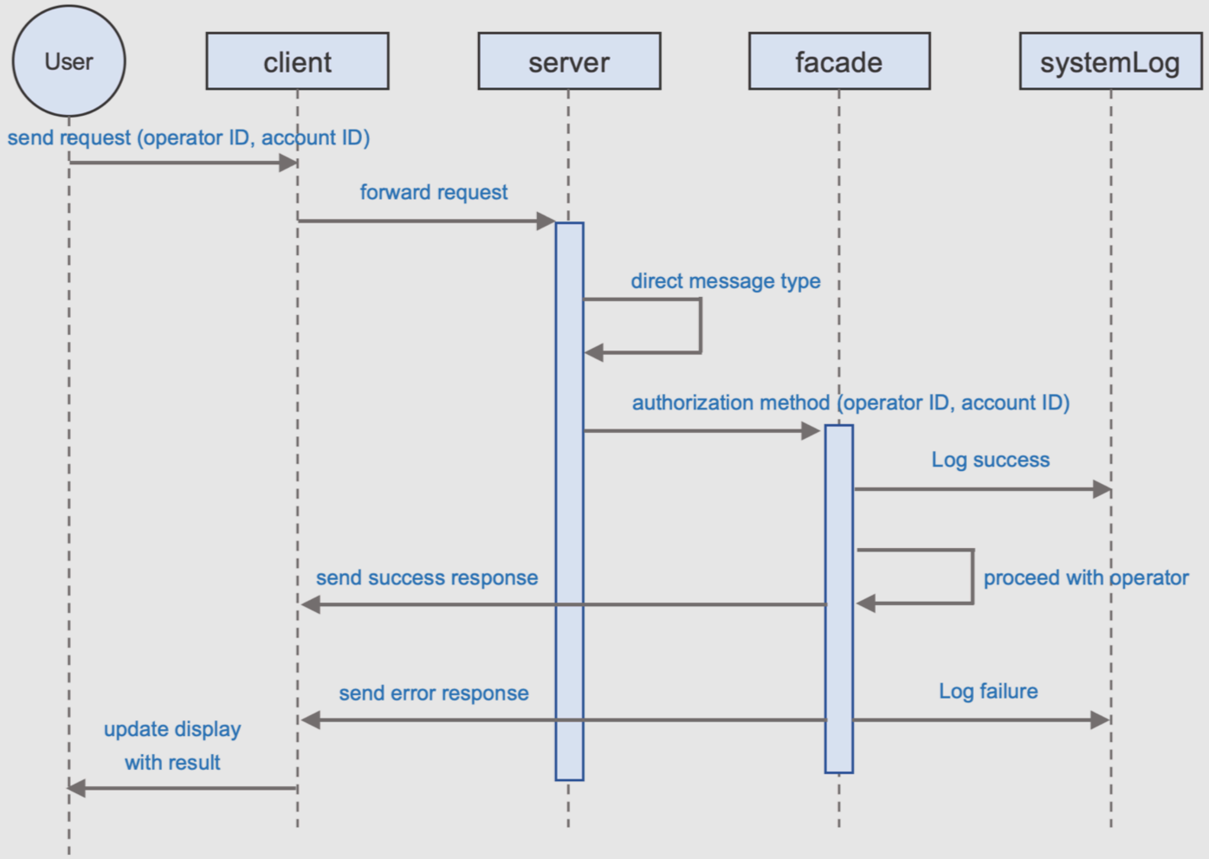
Exceptions:

Related Use Cases:

- CS001, CS002

Use Case ID: CS004 Client and Server Section 004

Use Case Name: Account authorization



3.3. Message Use Cases

3.3.1 Use Case ID: M001

Use Case Name: Creating Message

Relevant Requirements: 3.1.4.2, 3.1.4.3, 3.1.4.4, 3.1.4.5, 3.1.4.5.1, 3.1.4.5.4

Primary Actor: Message Module

Pre-condition:

- Message constructor is called with data string and enum

Post-condition:

- Message object created

Basic Flow:

//remove multithreading as now that is handled by other modules

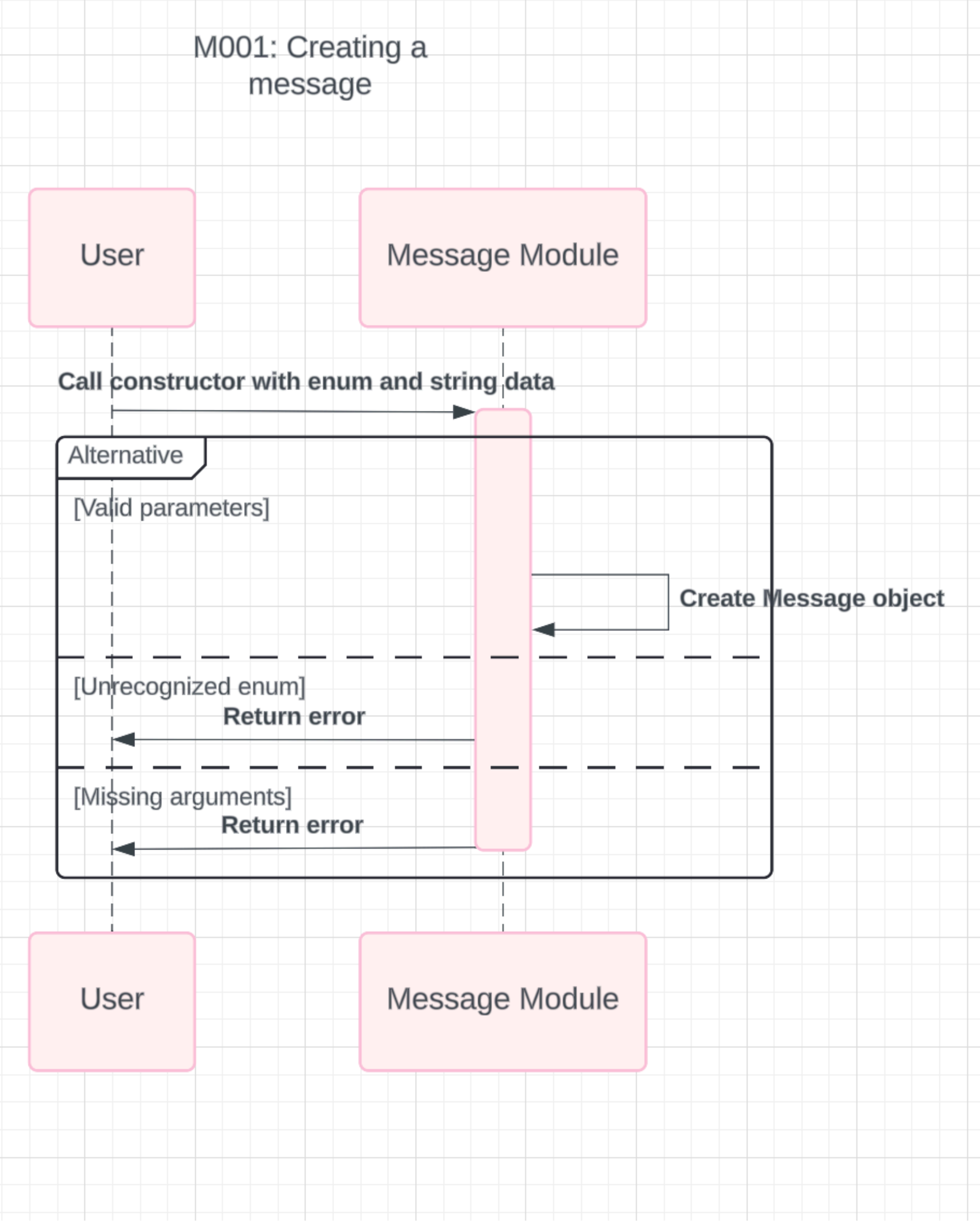
1. Constructor is called with enum and string data, and other parameters
2. Message object created.

Alternate Flows:

Exceptions:

1. Unrecognized enum.
2. MIssing arguements

Related Use Cases: CS001



M001: Creating a message

3.3.2 Use Case ID: M002

Use Case Name: Processing Message
Relevant Requirements: 3.1.4.2, 3.1.4.3, 3.1.4.4, 3.1.4.6, 3.1.4.6.1, 3.1.4.6.3
Primary Actor: Facade
Pre-condition:

- Facade is handling a message object

Post-condition:

- Facade decides which appropriate methods to call

Basic Flow:

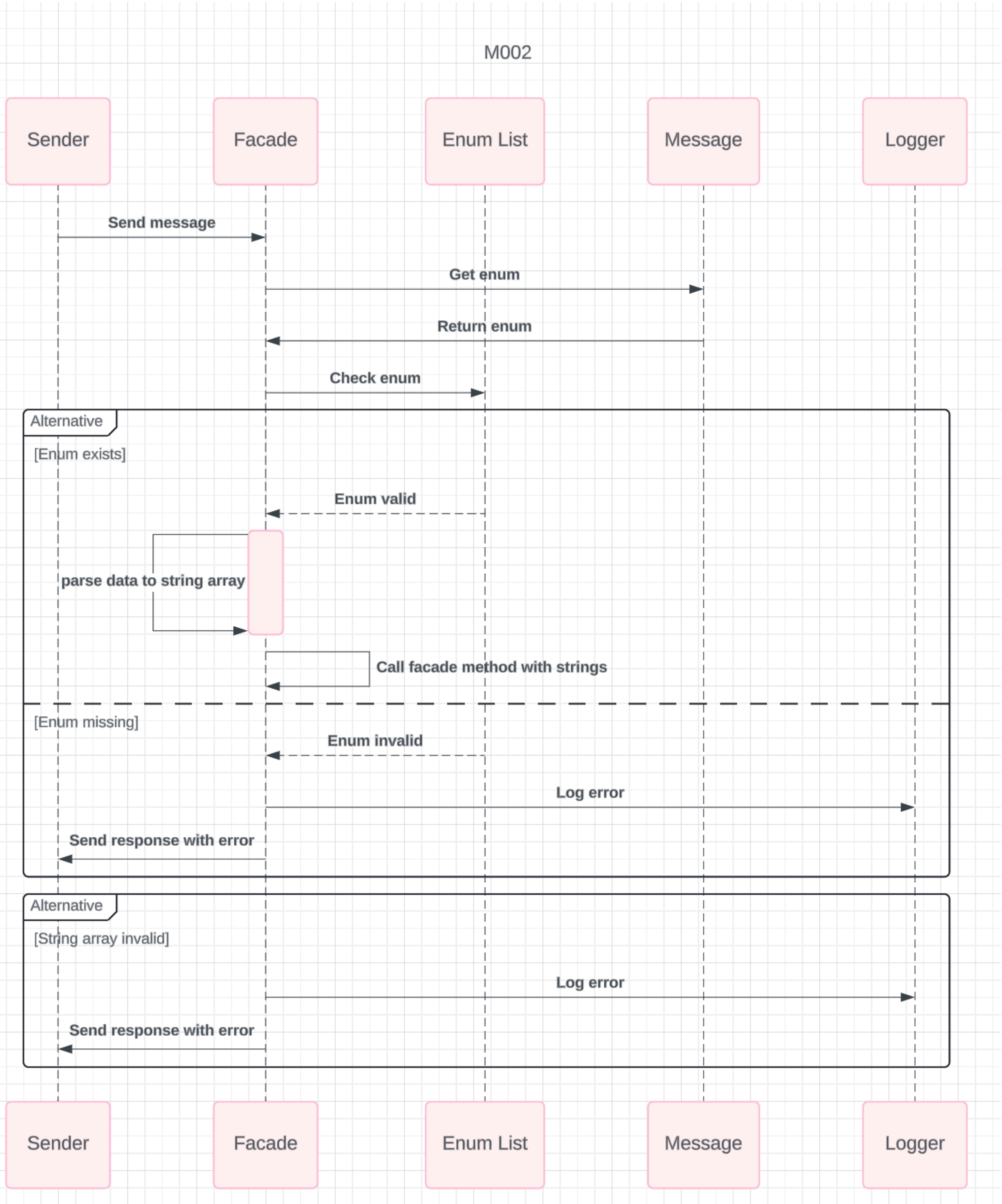
1. Get an enum and check it with the message's enum.
2. If enum exist in enum list, call message method to parse data string into array of strings
3. Call correct facade methods based on enum, pass strings from parsed string arrays as arguments in receiving order.

Alternate Flows:
Exceptions:

1. Message object missing enum.
2. Enum does not exist in the enum list.
3. Parsed string array does not contain correct arguments for method call.

-> Log, thow correct exception and facade handle with a response message to the sender.

Related Use Cases: CS001


M002
Sender
Facade
Enum List
Message
Logger
Send message
Get enum
Return enum
Check enum
Alternative
[Enum exists]
Enum valid
parse data to string array
Call facade method with strings
[Enum missing]
Enum invalid
Log error
Send response with error
Alternative
[String array invalid]
Log error
Send response with error
Sender
Facade
Enum List
Message
Logger

3.4. Account Use Cases

3.4.1 Use Case ID: A001

Use Case Name: Account transfer

Relevant Requirements:

- received a request of transferring

Primary Actor:

- Account module

Pre-condition:

- Account is accessed by authorized operator.
- Server invokes account transfer method.

Post-condition:

- Account transferred, balance update.

Basic Flow:

1. Checking current balance against transfer amount.
2. Checking transfer constraint.
3. Account initialized correct destination accounts.
4. Update balance from both accounts to reflect change.
5. Update new balance to files.

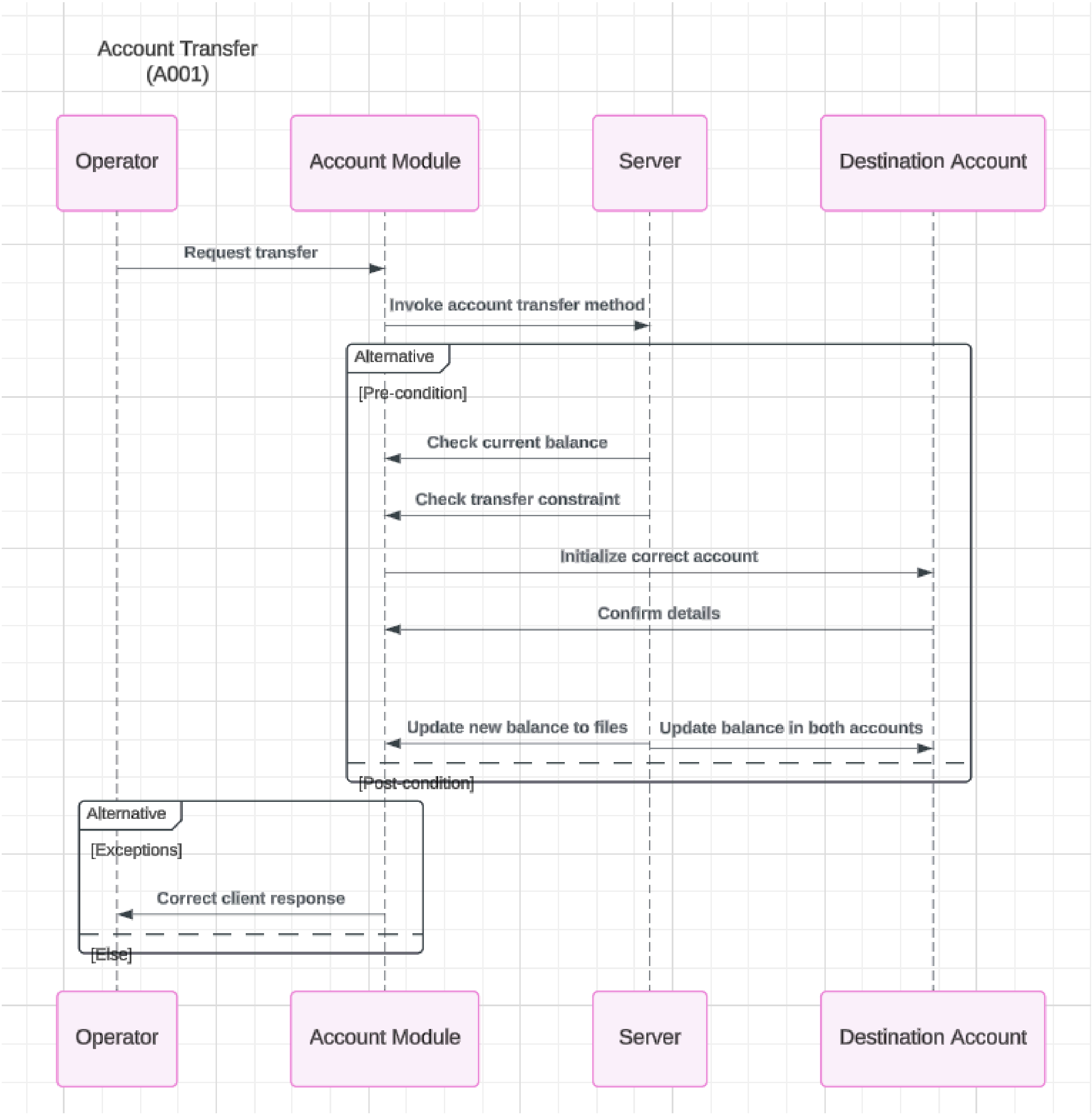
Alternate Flows:

Exceptions:

- Account dont exist, invoke correct client response.
- Transfer constraint invoked, abort

Related Use Cases:

- Transferring request


Account Transfer
(A001)
Operator
Account Module
Server
Destination Account
Request transfer
Invoke account transfer method
Alternative
[Pre-condition]
Check current balance
Check transfer constraint
Initialize correct account
Confirm details
Update new balance to files
Update balance in both accounts
[Post-condition]
Alternative
[Exceptions]
Correct client response
[Else]
Operator
Account Module
Server
Destination Account

3.4.2 Use Case ID: A002

Use Case Name: Creating an account

Relevant Requirements:

- A user wants to create an account to be able to store their money in either checkings or savings.

Primary Actor:

- Client Module
- Server Module

Pre-condition:

- The user is in position where it is beneficial to create an account.

Post-condition:

- The user has now made an account and has gained access to further pursue a bank account.

Basic Flow:

1. The user initiates an account creation request.
2. The user provides necessary personal information.
3. The user's identification begins to be verified.
4. The superuser verifies the user's information.
5. The user now has an account.

Alternate Flows:

1. The user enters any type of wrong information during the process of wanting to create an account. So the user is prompted to make changes to the incorrect information.
2. The user's request to create an account was rejected.

Exceptions:

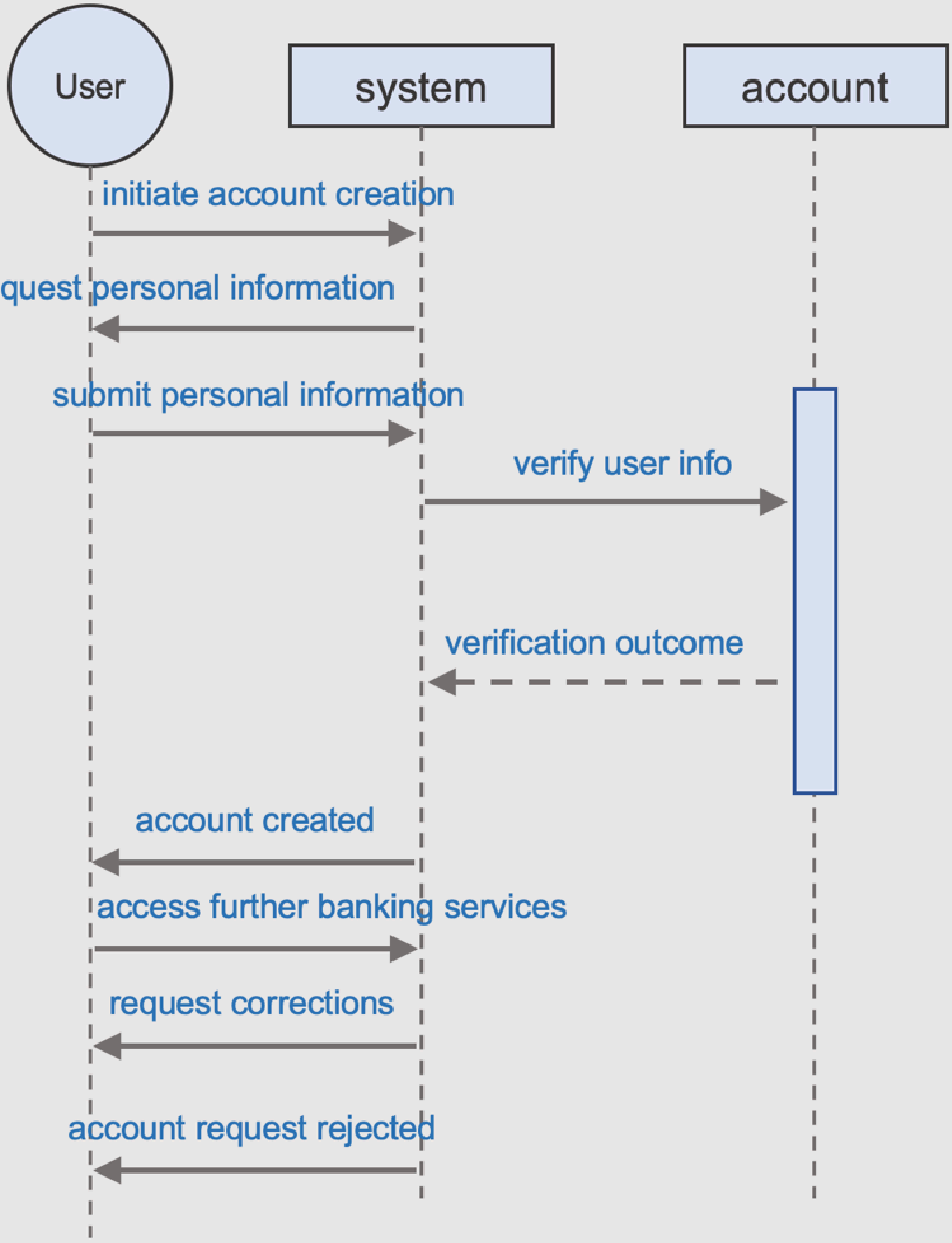
1. The user provides invalid personal information.
2. The user's ID and password are already taken so the user must think of a different ID and password.
3. The ID and or password don't meet bank's security requirements.
4. The user doesn't meet the bank's requirements in order to create an account. (Ex: Age).

Related Use Cases:

Logging in, get authorized, send message to server, retrieve message from server

Use Case ID: A002 Account use cases 002

Use Case Name: Creating an account


User
system
account
initiate account creation
request personal information
submit personal information
verify user info
verification outcome
account created
access further banking services
request corrections
account request rejected

3.4.3 Use Case ID: A003

Use Case Name: Deleting an account

Relevant Requirements:

- A user's account has reached a point in which it is not needed anymore.

Primary Actor:

- Client Module
- Server Module

Pre-condition:

- The user's funds have been removed from the account and has been verified for deletion of account.

Post-condition:

- The user's account has been deleted.

Basic Flow:

1. Initialize deleting an account.
2. User is authenticated to proceed with the deleting process.
3. The account's information is under review and the user is notified of deleting an account.
4. Funds are removed from the account, if any.
5. The account has been deleted.

Alternate Flows:

1. The user makes the decision of not wanting to delete their account.
2. The account can't be deleted if the user has any type of outstanding charges.

Exceptions:

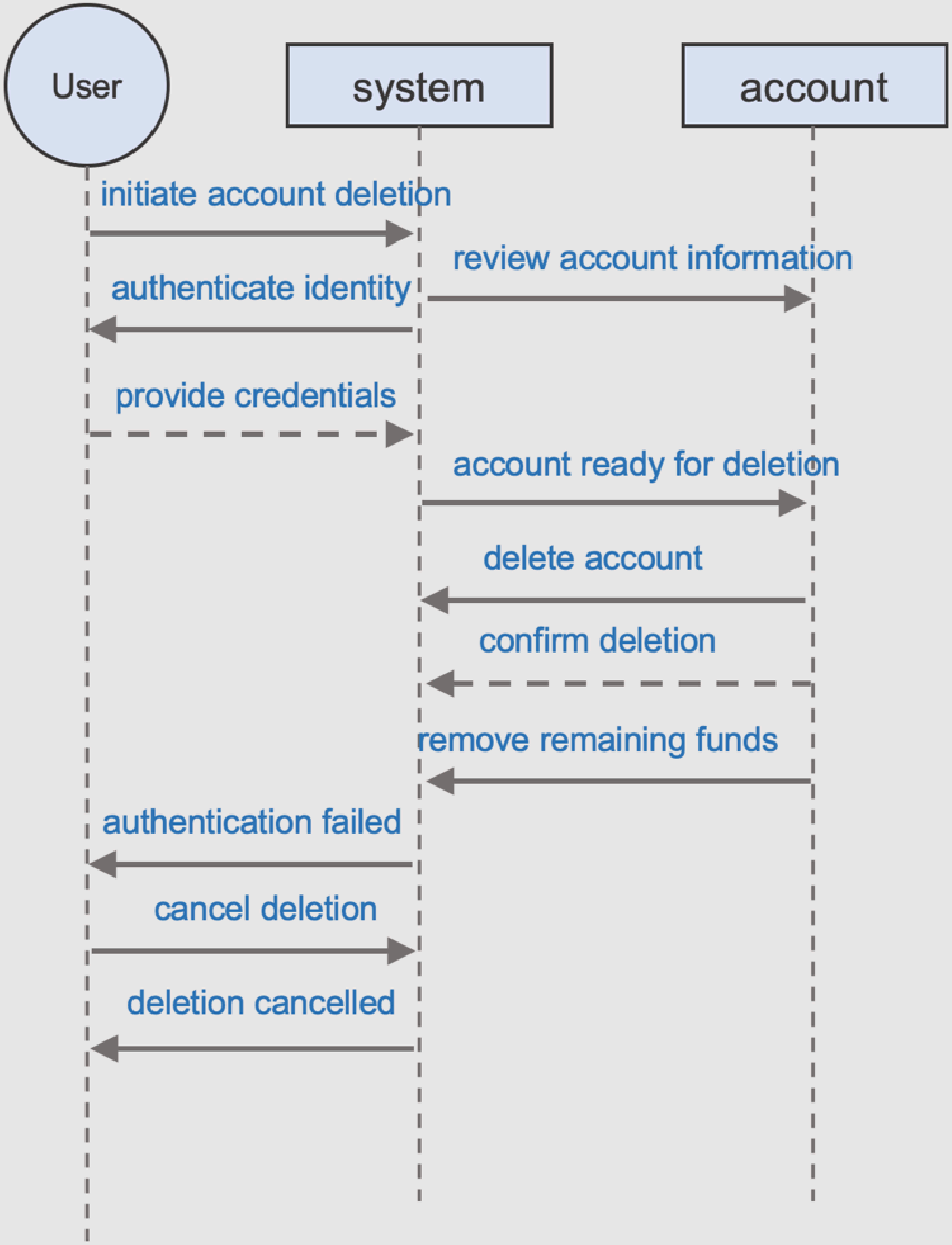
1. The user fails to authenticate credentials.
2. There are outstanding charges to the user's account.
3. The server or client run into technical issues throughout the deletion process.
4. The user's account has already been deleted.
5. The user's account can't be deleted if the account's current status is locked, or restricted.

Related Use Cases:

- Log in ,Unauthorized user

Use Case ID: A003 Account use cases 003

Use Case Name: Deleting an account



3.4.4 Use Case ID: A004

Use Case Name: Add a user to an account

Relevant Requirements: 3.1.3.15, 3.1.3.16.

Primary Actor: Account Module

Pre-condition: An account exists and a user exists.

Post-condition: The account and user are connected and the user has access to the account.

Basic Flow:

1. The primary holder of the account logs-in from the Teller's interface.
2. The user provides the user ID of the user that is to be added.
3. The to be added user logs in through the teller interface.

Alternate Flows:

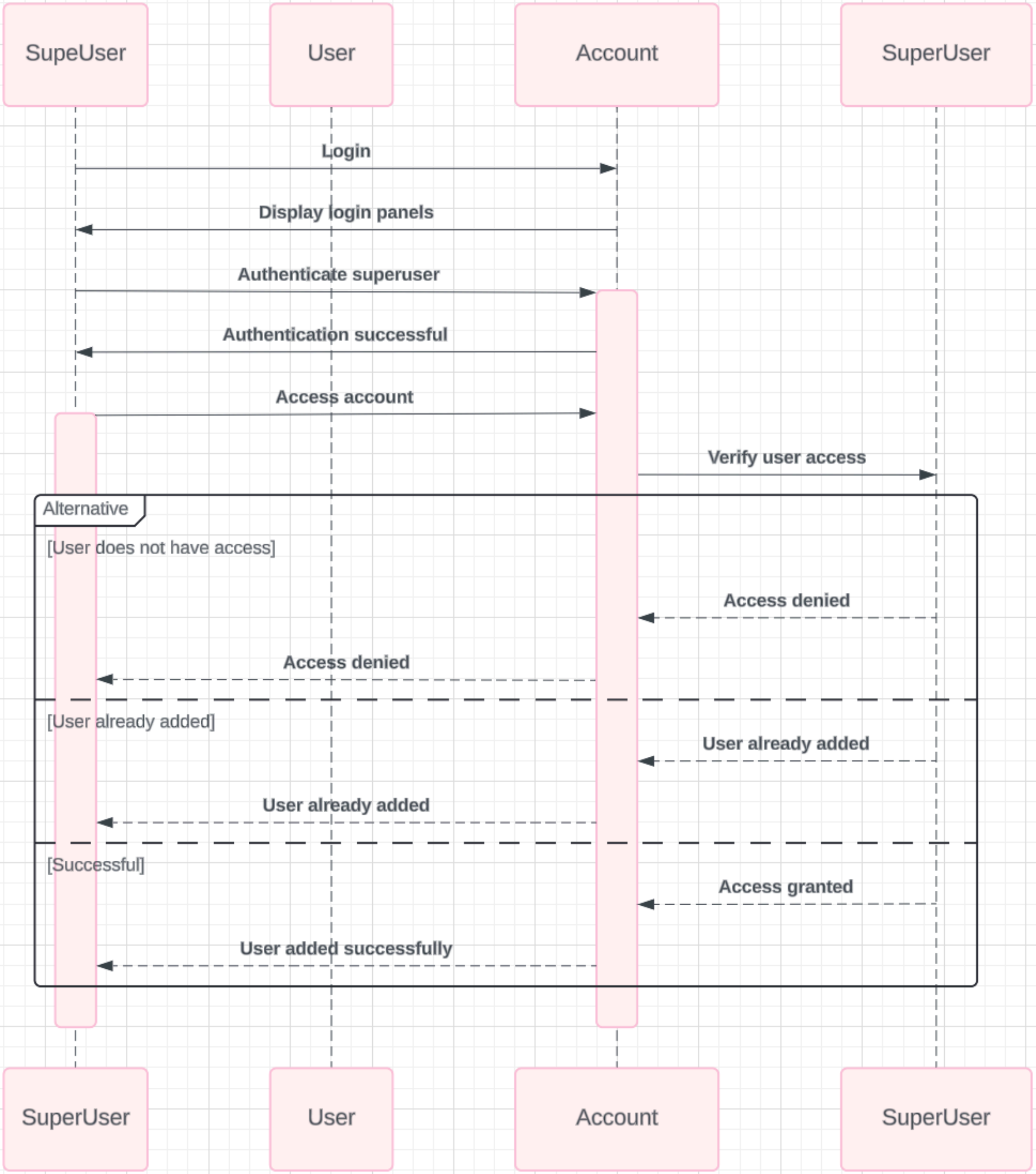
none

Exceptions:

1. The user does not have the required access to the account.
2. The user is already added to the account.

Related Use Cases:

A004: Add a user to an account



3.4.5 Use Case ID: A005

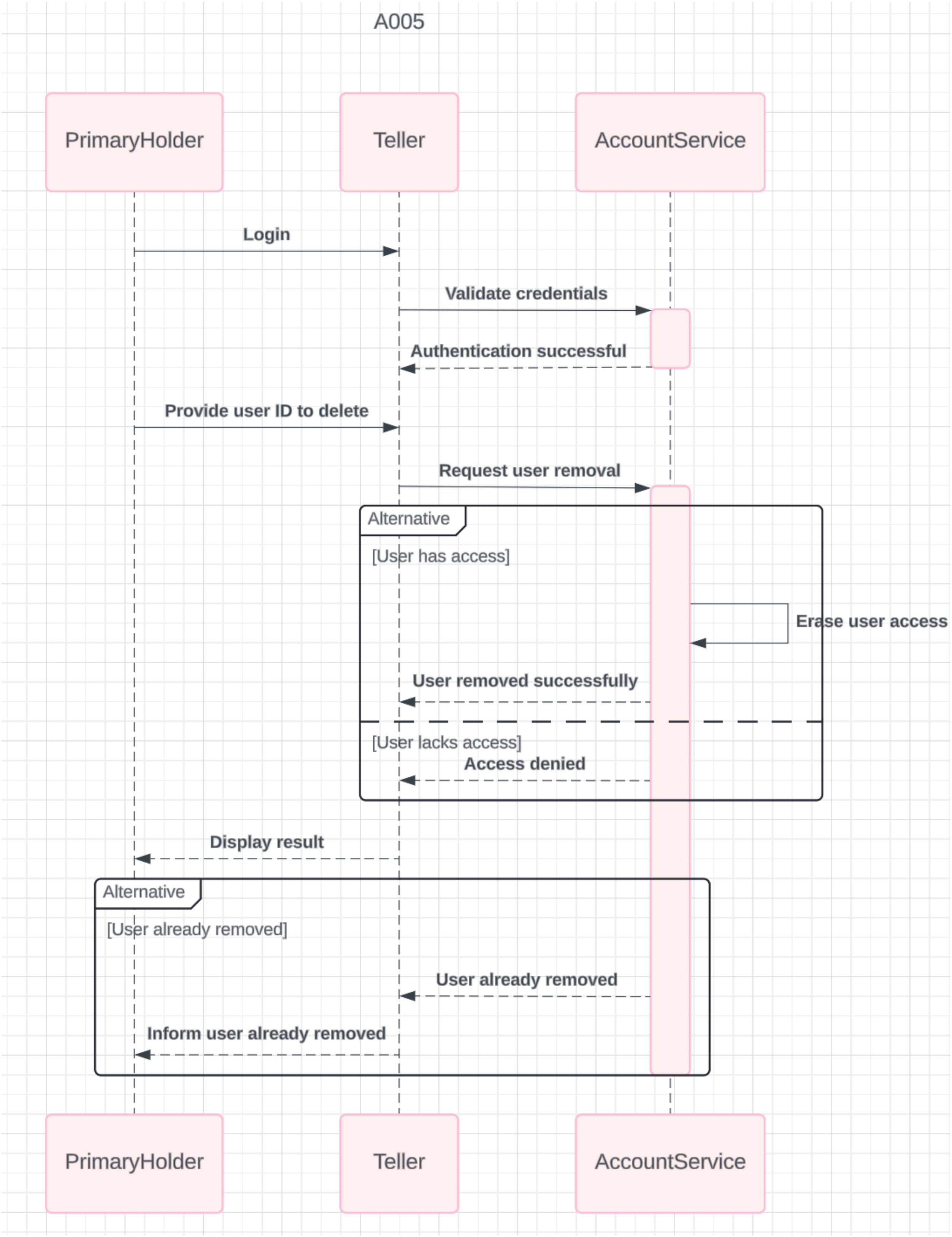
Use Case Name: Remove a user from an account.
Relevant Requirements: 3.1.3.15, 3.1.3.16.
Primary Actor: Client Module.
Pre-condition: An additional user is a part of an account.
Post-condition: The user is removed from the account.
Basic Flow:

1. The primary holder of the account logs-in from the Teller's interface.
2. The user provides the user ID of the user that is to be deleted.
3. The user's access to the account is erased.

Alternate Flows:
Exceptions:

1. The user does not have the required access to the account.
2. The user is already removed from the account.

Related Use Cases:


A005
PrimaryHolder
Teller
AccountService
Login
Validate credentials
Authentication successful
Provide user ID to delete
Request user removal
Alternative
[User has access]
Erase user access
User removed successfully
[User lacks access]
Access denied
Display result
Alternative
[User already removed]
User already removed
Inform user already removed
PrimaryHolder
Teller
AccountService

3.4.6 Use Case ID: A006

Use Case Name: View Transaction History

Relevant Requirements: 3.1.2.4.3, 3.1.2.3.5 .

Primary Actor: Accounts Module, Client Modules

Pre-condition:

1. The customer must be logged into the system with valid credentials.
2. The customer must have at least one bank account registered.
3. The customer initiates a request to view their transaction history.

Post-condition:

The transaction history is presented to the customer.

Basic Flow:

1. The customer navigates to the "Account Summary" or "Transactions" section.
2. The system displays a default view of the most recent transactions for the selected account.

Alternate Flows:

No Transactions: If the account has no transactions for the selected date range, the system will display a “No transactions found” message.

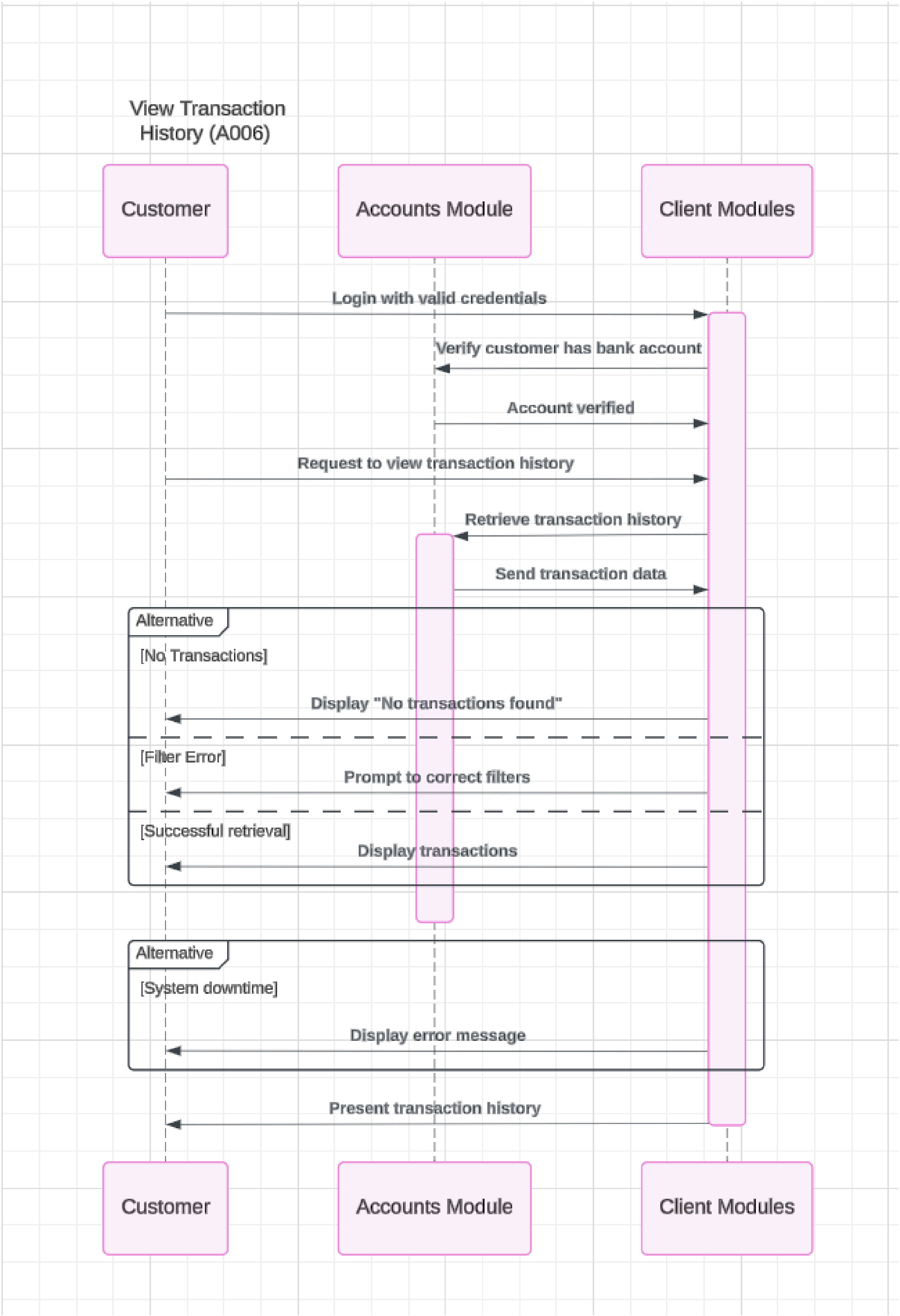
Filter Error: If the filters entered by the customer are invalid (e.g., future date), the system prompts the operator to correct the input.

Exceptions:

System downtime or unavailability might prevent transaction data from being retrieved.

If there’s a lag in the backend systems, some recent transactions may not appear immediately.

Related Use Cases:


View Transaction History (A006)
Customer
Accounts Module
Client Modules
Login with valid credentials
Verify customer has bank account
Account verified
Request to view transaction history
Retrieve transaction history
Send transaction data
Alternative
[No Transactions]
Display "No transactions found"
[Filter Error]
Prompt to correct filters
[Successful retrieval]
Display transactions
Alternative
[System downtime]
Display error message
Present transaction history
Customer
Accounts Module
Client Modules