Low Level Design
MotoMoto

The New Panelists
12/12/2021

Blake Del Rey
Isabel Guzman
Jacob Sunia
James Austin Jr.
Naeun Yu

Team Leader: James Austin Jr.

Table of Contents

1. Overview

1.1 Scope

This document covers User management, Logging, Archiving, Authorization, and Authentication.

1.2 Purpose

The document is intended to understand the low level design of our planned implemented features of the MotoMoto application. THis document will include the diagrams necessary flow and handling of errors. The diagrams will be designed to allow developers to understand the features that our team will be using.

2. User Management

2.1 Overview

This core requirement is split into four operations: Create, Update, Delete, Disable/Enable. The Disable/Enable operation is a subset of the Update operation but we diagram both separately. All of these operations can be triggered by system admin and in our diagrams we assume the operation is triggered by an authorized system admin. Clearly, the authorization and authentication features would need to be implemented to verify a system admin before triggering these operations.

2.2 Sequence Diagrams

We organized these sequence diagrams according to the layers modeled in our High Level Design. For our Data Access layer, we use a CRUD repository pattern for accessing and storing data so we can have more definitive control of interaction with the data store. We include a sequence of successful actions for the operation to be successful as well and the failure cases are designed in the case one or some of these actions are not successful.

2.2.1 Logging

Logging appears towards the end of successful operations while logging is triggered at the point an error is recognized (typically at the Business or Service layer) for failure cases. We use this strategy to meet the clients for information logging of system activity. Logging will record the time of each operation as well as the security level, the user id, category, and an automated description of the incident.

3. Logging

3.1 Overview

This core requirement is incorporated into every user management sequence diagram and will be included in every feature designed in the future. Note that we use a separate data store from where the accounts are stored and the logging process in features using it will interact with its own data store separate from the user account data store.

3.2 Sequence Diagrams

We created a base design for future logging implementations to extrapolate from. The process can be triggered by any operation requesting the logging service and therefore is not limited in use.

4. Archiving

4.1 Overview

This core requirement is triggered at midnight of every first of the month, this would indicate a Business layer be involved, but we utilize the logging service to instead invoke the Archiving process.

4.2 Sequence Diagrams

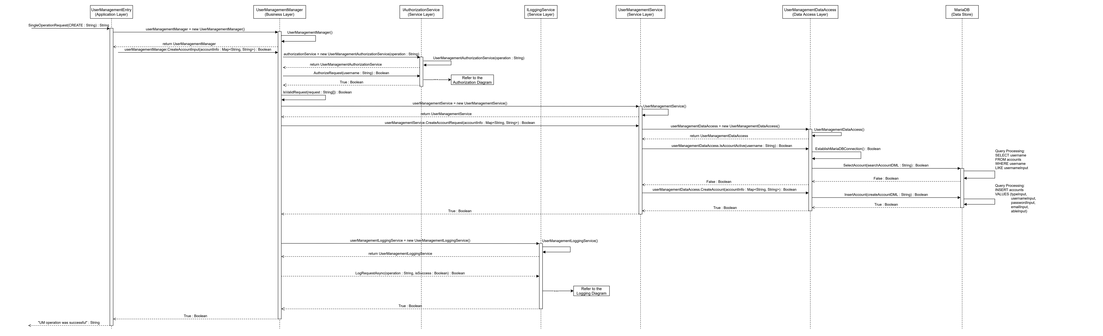
We also created a base design for future archiving implementations to extrapolate from. The process can be triggered by the system admin requesting the archiving service, but we plan to automate this feature to offload this responsibility from the system admin.

5. Appendix

See next pages for operation sequence diagrams and database models.

CREATE Operation

SUCCESS CASE DIAGRAM

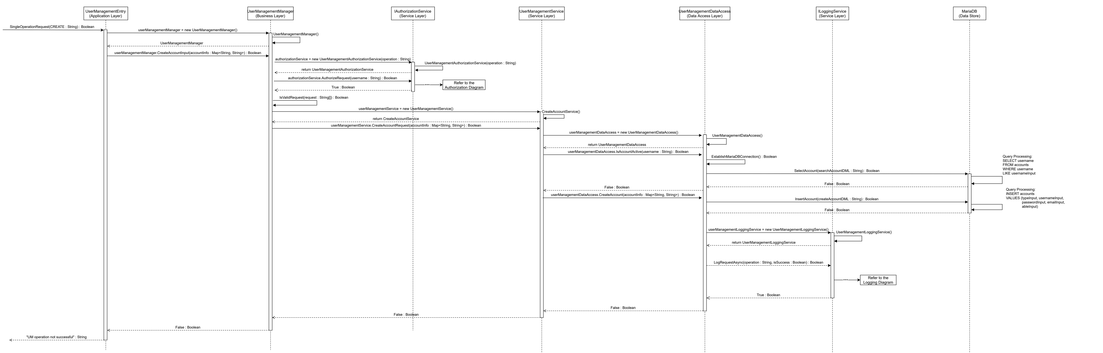


Success Scenario

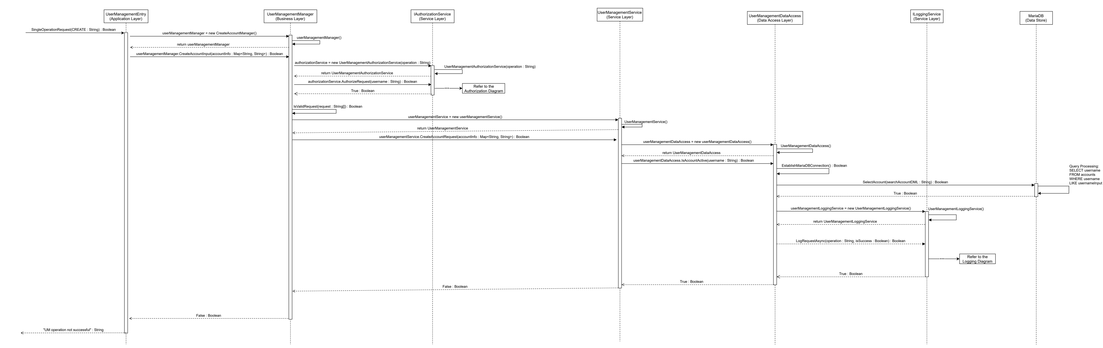
1. Single Request is made for the CREATE operation
2. The UserManagementManager or CreateAccountManager is created
3. Input account info
4. Check if request is valid, the following is checked:
   - is the input in the correct format
   - does the input contain all necessary info for the operation request
5. New instance of CreateAccountService is created
6. Call the CreateAccount Service to process operation with the validated input
7. New instance of CreateAccountDataAccess is created
8. Call the CreateAccount DataAccess to process operation
9. Establish the MariaDB connection in DataAccess if not already opened
10. Perform a SELECT query on the Accounts table in the database:
    SELECT username
    FROM accounts
    WHERE username
    LIKE usernameInput
    Returns FALSE if not found
11. Return FALSE to the Service if no account already exists, and continue
12. Create the account with the account info
13. Perform a INSERT query on the Accounts table in the database:
    INSERT accounts
    VALUES (hashInput, usernameInput, passwordInput, emailInput, attempts)
    Returns TRUE if account is successfully created.
14. Return TRUE to the Service if the account was successfully created
15. Return TRUE to the Manager if the request was successfully performed
16. Return TRUE to System and follow with the appropriate Logging for this operation

FAILURE CASE DIAGRAMS

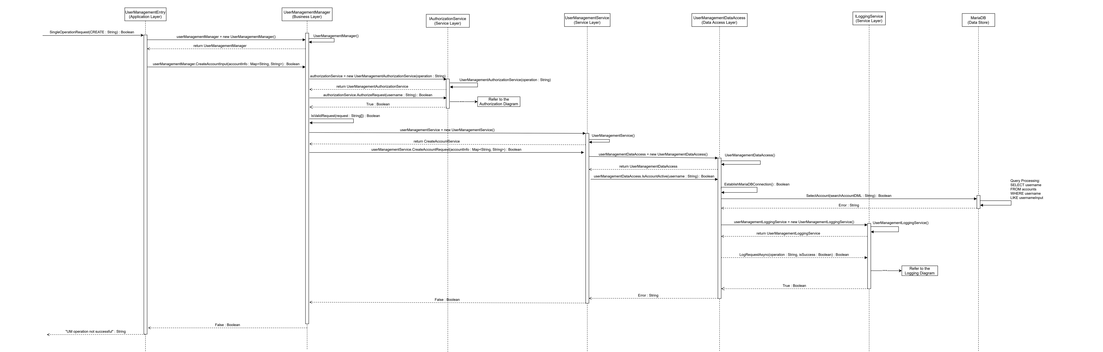
Fail Case 1$
New Account is not Inserted into MariaDB
Reasons
- incorrect DML syntax
- MariaDB fails either by connection timeout or external factors



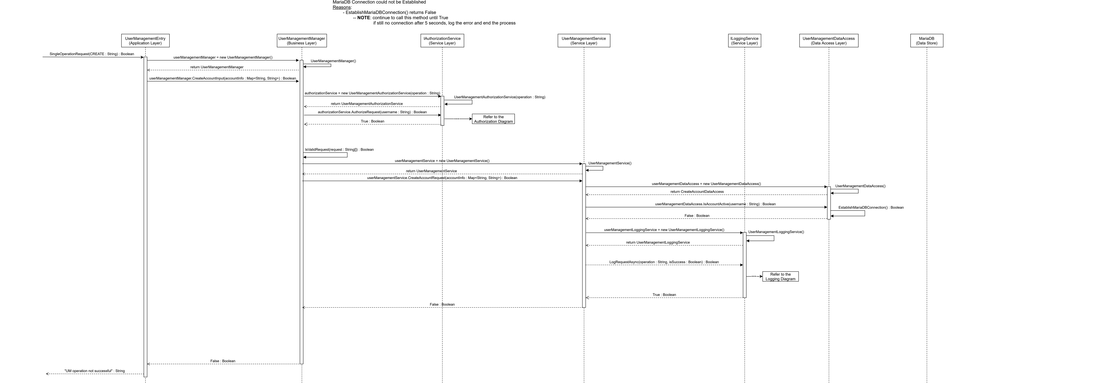
Fail Case 10a
New Account already exists in MariaDB
Reasons
- account already exists



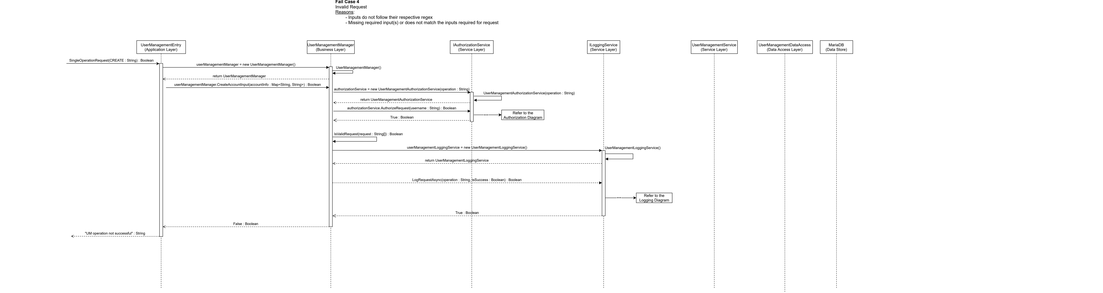
Fail Case 10b
Could Not Process Query
Reasons
- incorrect DML syntax
- MariaDB fails either by connection timeout or external factors



Fail Case 9
MariaDB Connection could not be Established
Reasons
- EstablishMariaDBConnection() returns False
- **NOTE** : continue to call this method until True
  if still no connection after 5 seconds, log the error and end the process

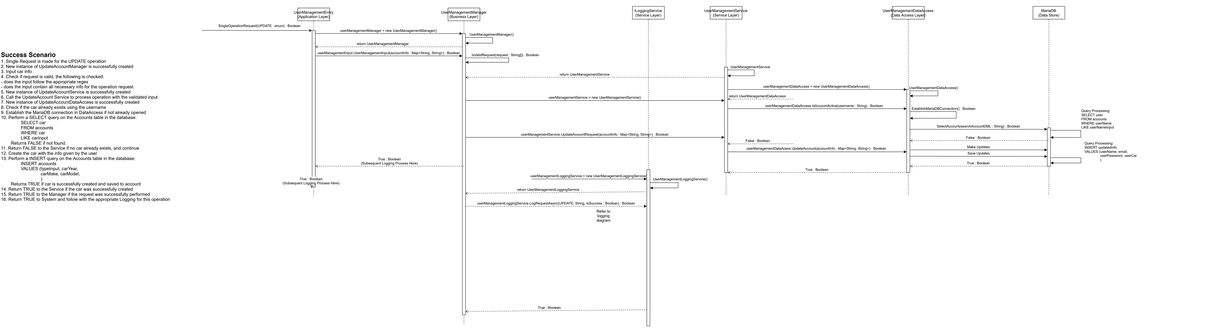


Fail Case 4
Invalid Request
Reasons
- inputs do not follow their respective regex
- Missing required input(s) or does not match the inputs required for request



UPDATE Operation

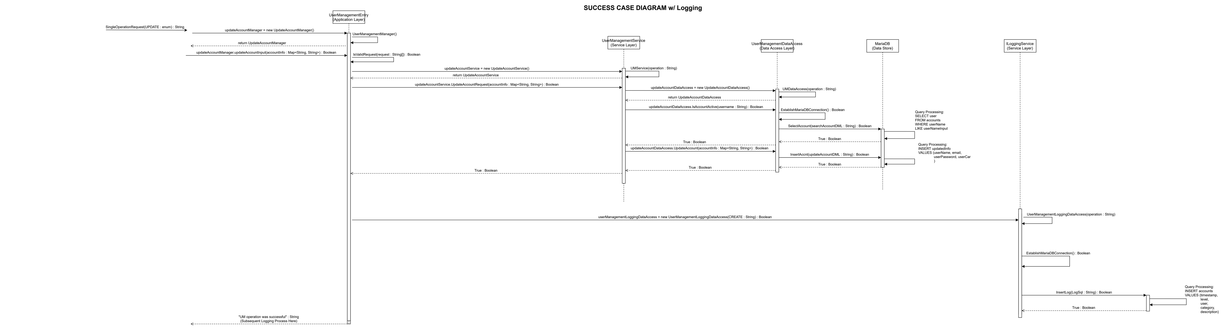
SUCCESS CASE DIAGRAM



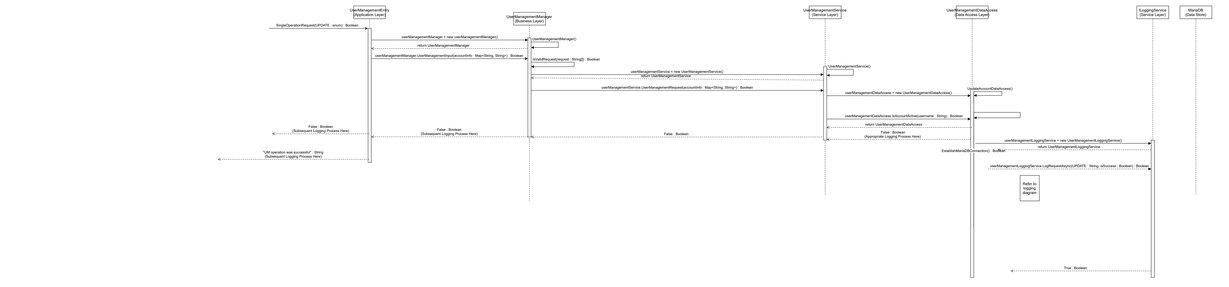
Success Scenario

1. Single Request is made for the UPDATE operation
2. The UserManagementManager is successfully created
3. Input car info
4. Check if request is valid, the following is checked:
   - does the input contain all info
   - does the input contain all necessary info for the operation request
5. New instance of UpdateAccountService is successfully created
6. Call the UpdateAccount Service to process operation with the validated input
7. New instance of UpdateAccountDataAccess is successfully created
8. Call the UpdateAccountDataAccess to process operation
9. Establish the MariaDB connection in DataAccess if not already opened
10. Perform a SELECT query on the Accounts table in the database:
    SELECT user
    FROM accounts
    WHERE car
    LIKE :carInput
    Returns FALSE if not found.
11. Return FALSE to the Service if no car already exists, and continue
12. Create the car with the info given by the user
13. Perform a INSERT query on the Accounts table in the database:
    INSERT updateInfo
    VALUES (topSpeed, carYear,
    carMake, carModel,
    Returns TRUE if car is successfully created and saved to account
14. Return TRUE to the Service if the car was successfully created
15. Return TRUE to the Manager if the request was successfully performed
16. Return TRUE to System and follow with the appropriate Logging for this operation

SUCCESS CASE DIAGRAM w/ Logging

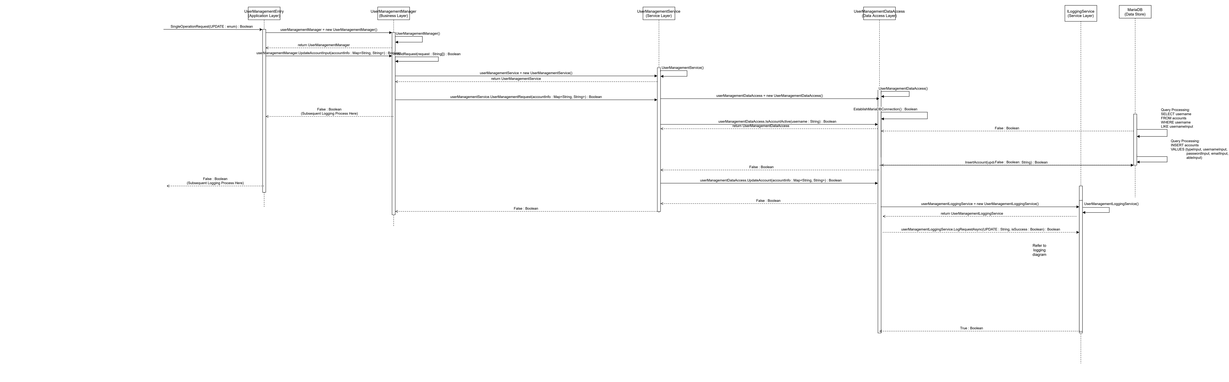


Fail Case 3
MariaDB Connection could not be Established
Reasons:
- EstablishMariaDBConnection(): returns False
  - **NOTE**: continue to call this method until True
    If still no connection after 5 seconds, log the error and end the process

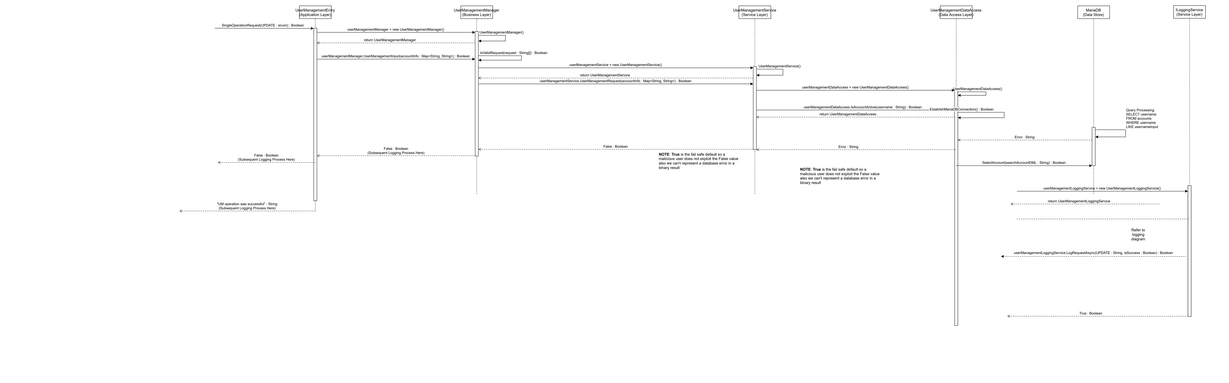


FAILURE CASE DIAGRAMS

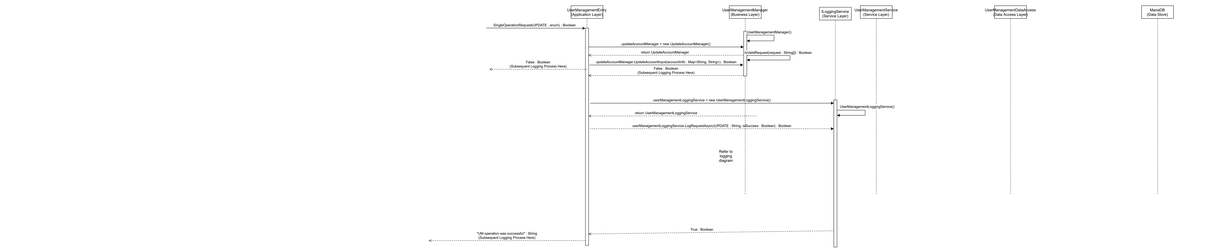
Fail Case 1
New information is not Inserted into MariaDB
Reasons:
- incorrect DML syntax
- MariaDB fails either by connection timeout or external factors

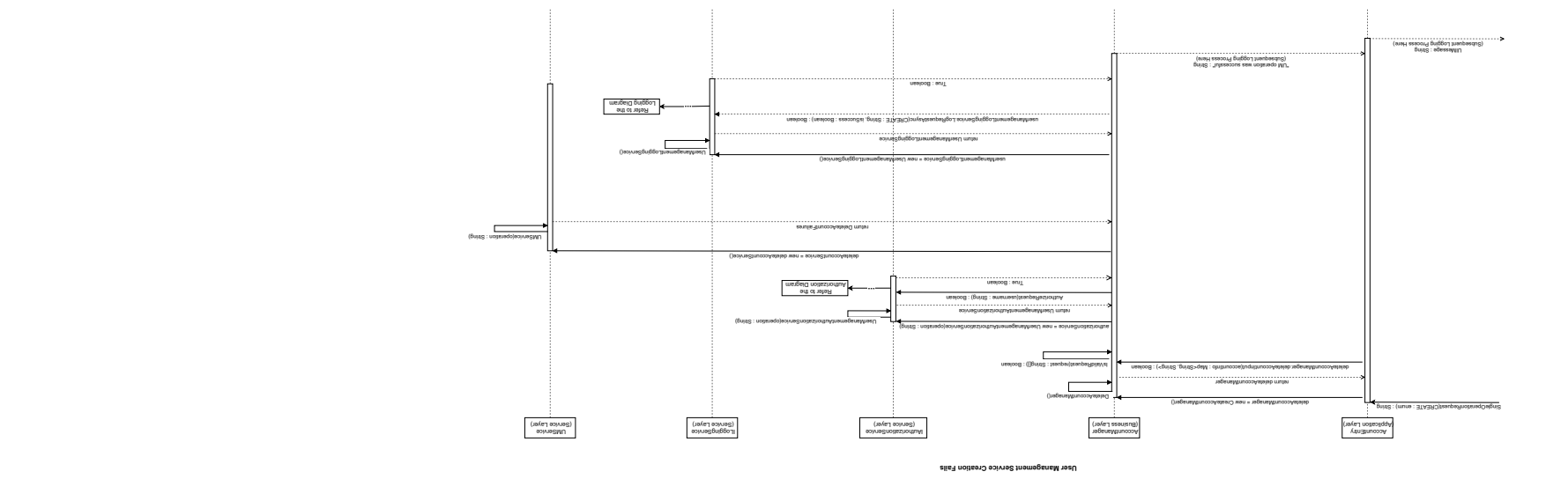


Fail Case 2
Could Not Process Query
Reasons:
- incorrect DML syntax
- MariaDB fails either by connection timeout or external factors

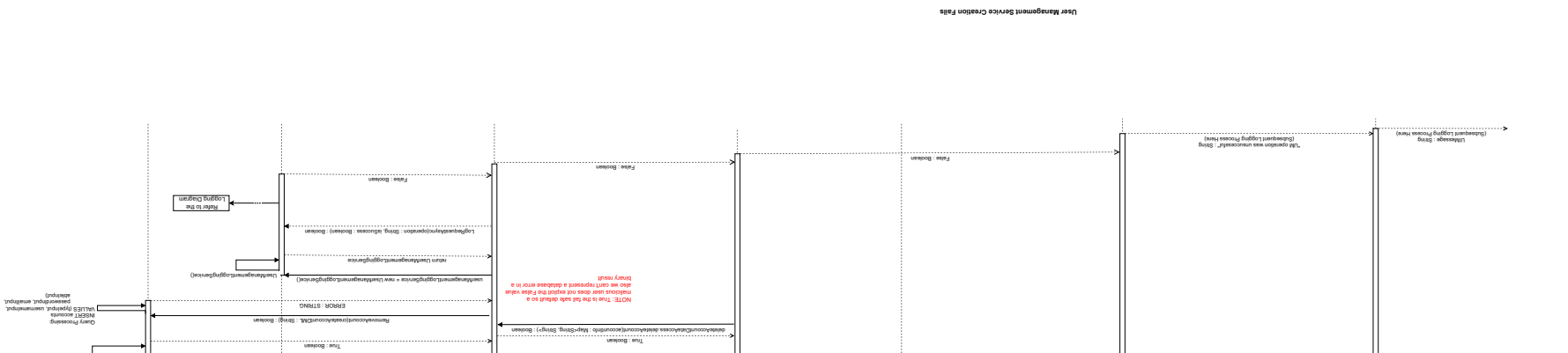


Fail Case 4
Invalid Request
Reasons:
- Inputs do not follow their respective regex
- Missing required input(s) or does not match the inputs required for request

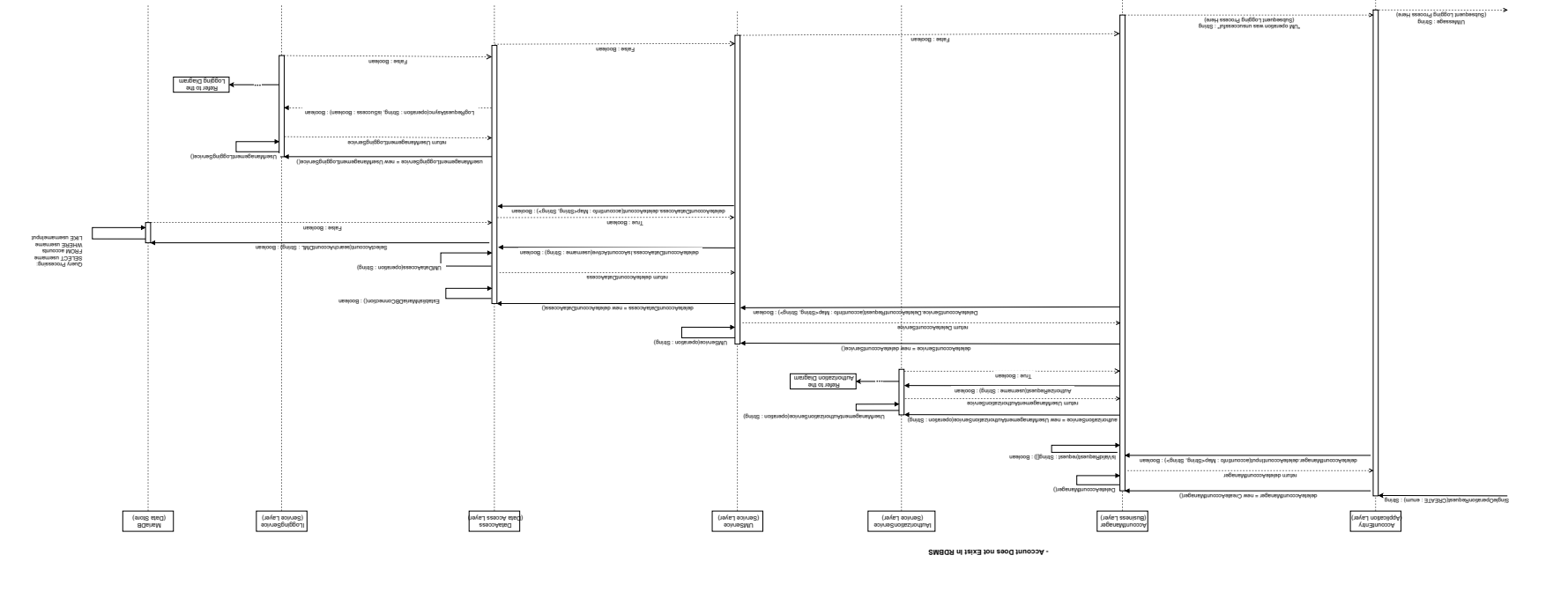




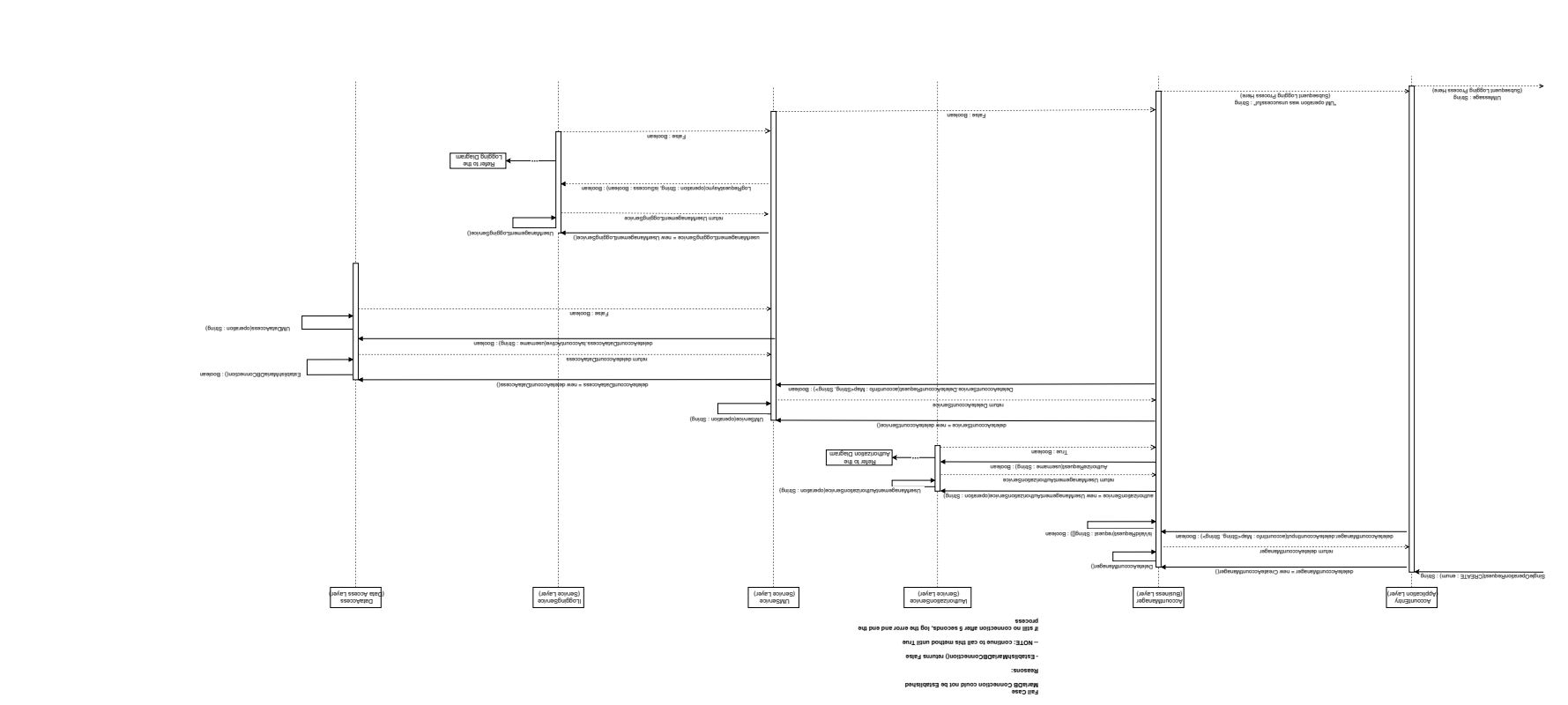
- Account Is Not Deleted From the Data Store Due To System Error/Connection Failures



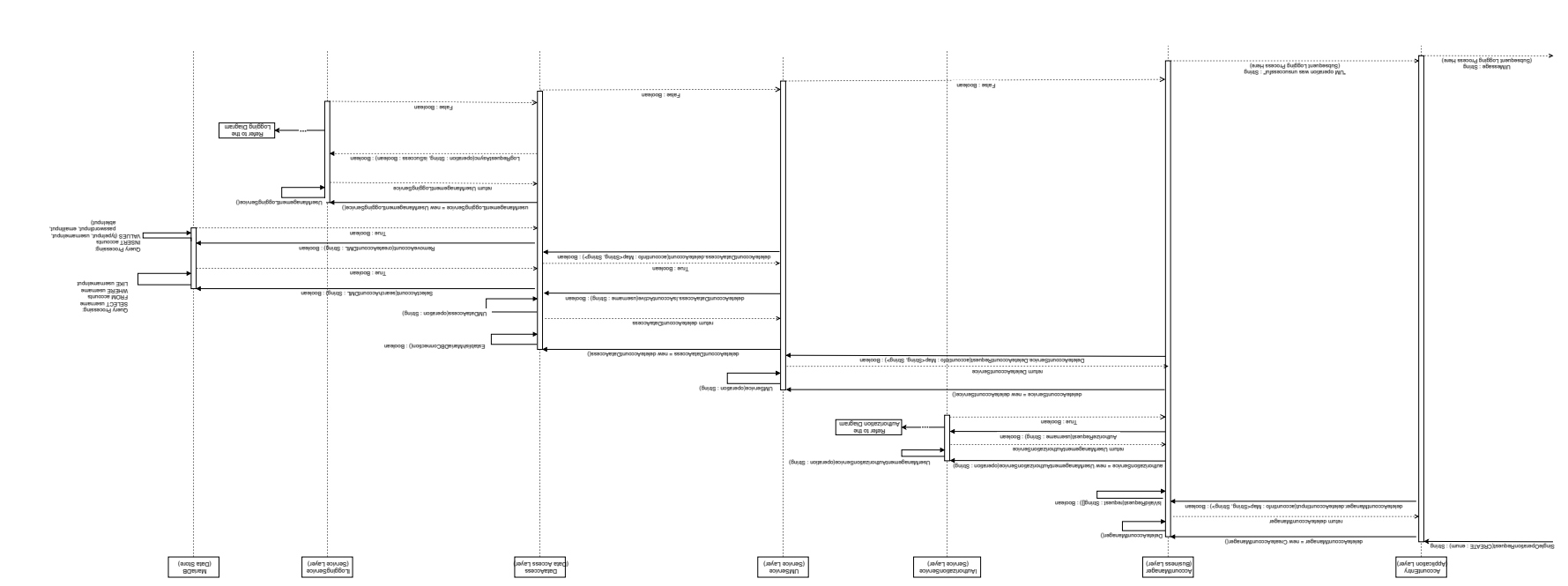
- Account Does not Exist in RDBMS



- Case
MariaDB Connection could not be Established
Reasons:
- EstablishMariaDBConnection() returns False
-- NOTE: continue to call this method until True
If still no connection after 5 seconds, log the error and end the process

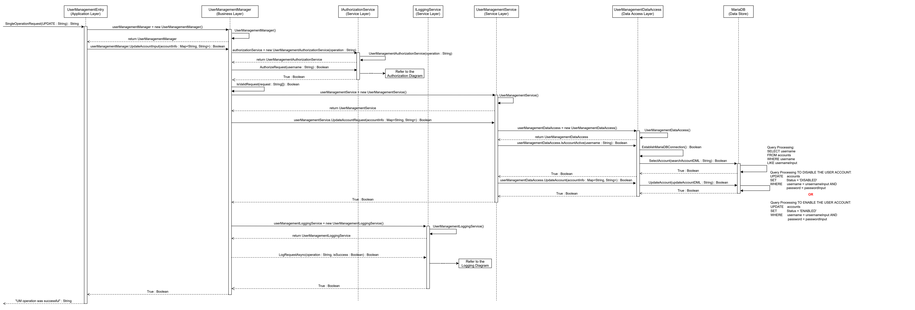


SUCCESS CASE



DELETE OPERATION

Disable/Enable SUCCESS CASE DIAGRAM

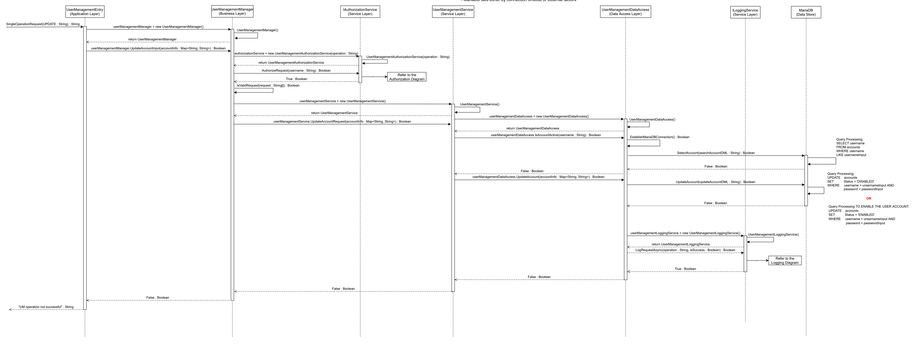


Disable Success Scenario
1. Single Request is made for the DISABLE Operation
2. New instance of DisableAccountManager is successfully created
3. Input Accountinfo
4. Validate request, the following is checked
- Does the input provide all necessary information
5. Call the Email of DisableAccountService is created
6. New instance of DisableAccountService is created
7. New instance of DisableAccountDataAccess is created with the validated input
8. Check if the account already exists using the username
9. Establish connection to the MariaDB connection in DataAccess if not already opened
10. Perform a SELECT query on the Accounts table in the database:
```
SELECT
    FROM accounts
    WHERE username
    LIKE username=input
        Return TRUE if found
```
11. Return TRUE if the account if an account is in existence, and continue
12. Update the account status with the account information provided
13. Write to the LOG identifying that the account has been removed
14. Perform UPDATE query:
```
UPDATE      accounts
SET         Status = DISABLED
WHERE       username = usernameInput AND
            password = passwordInput
```
15. Returns TRUE if the account is successfully disabled from the datastore
16. Returns TRUE to the service if the account was successfully updated
17. Returns TRUE to the manager if the request was successfully completed
18. Returns TRUE to the system
19. Writes to the LOG the account number that was disabled and description of the action

Enable Success Scenario
1. Single Request is made for the ENABLE Operation
2. New instance of EnableAccountManager is successfully created
3. Input Accountinfo
4. Validate request, the following is checked
- Does the input provide all necessary information for the operations request
5. Call the Email of EnableAccountService is created
6. New instance of EnableAccountService is created with the validated input
7. New instance of the EnableAccountDataAccess is created
8. Check if the account already exists using the username
9. Establish connection to the MariaDB connection in DataAccess if not already opened
10. Perform a SELECT query
```
SELECT
    FROM accounts
    WHERE username
    LIKE username=input
        Return TRUE if found
```
11. Return TRUE to the service if account is in existence, and continue
12. Update the account status with the account information provided
13. Write to the LOG identifying that the account has been removed
14. Perform UPDATE query:
```
UPDATE      accounts
SET         Status = ENABLED
WHERE       username = usernameInput AND
            password = passwordInput
```
15. Returns TRUE if the account is successfully enabled from the datastore
16. Returns TRUE to the service if the account was successfully updated
17. Returns TRUE to the manager if the request was successfully completed
18. Returns TRUE to the system
19. Writes to the LOG the account number that was enabled and description of the action

Disable/Enable FAILURE CASE DIAGRAM

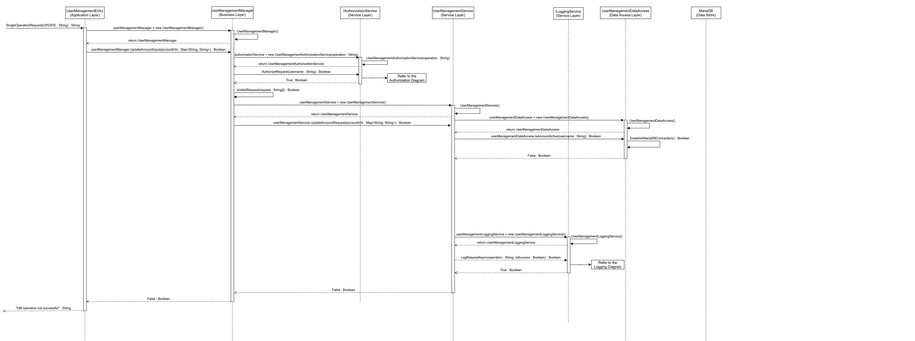
Fail Case 1:
Request Not Updated into MariaDB
Reason's
- Incorrect CML syntax
- MariaDB fails either by connection timeout or external factors



Disable/Enable FAILURE CASE DIAGRAM

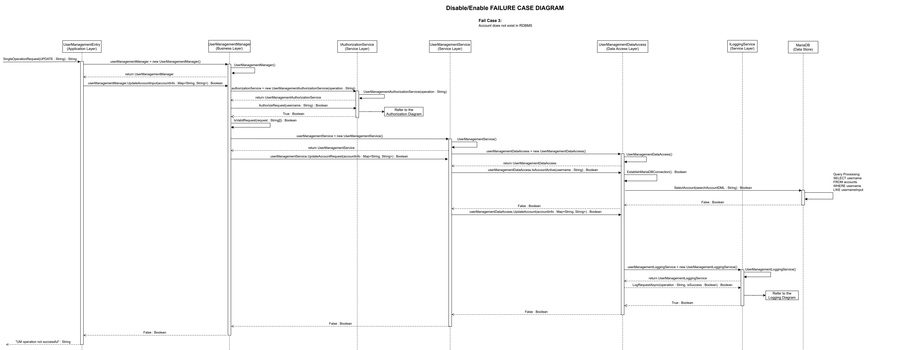
Fail Case 2
MariaDB Connection could not be Established
Reason's
- EstablishMariaDBConnection() returns False
- **NOTE**: continue to call this method until True

if still no connection after 5 seconds, log the error and exit the process



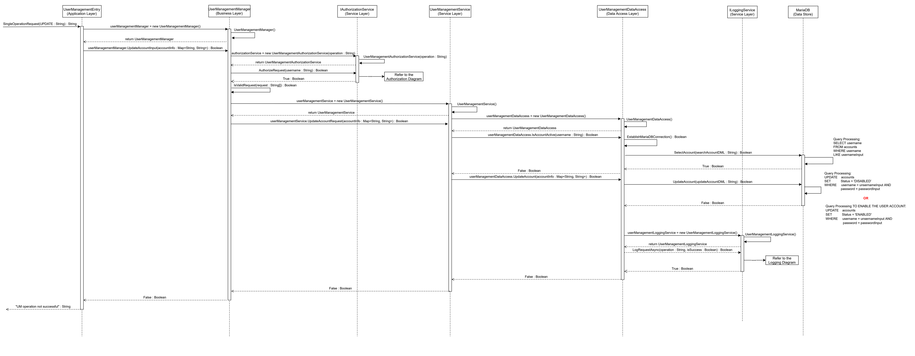
Disable/Enable FAILURE CASE DIAGRAM

Fail Case 3
Account does not exist in RDBMS



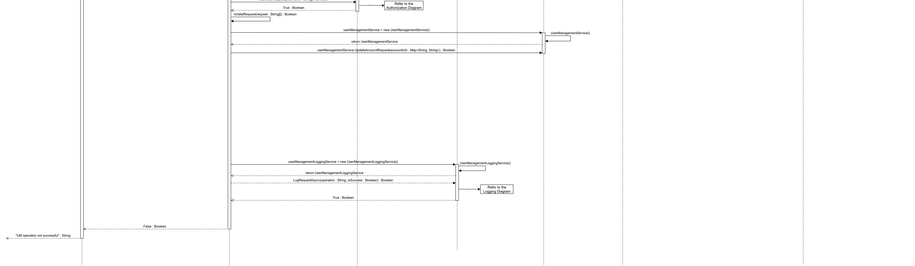
Disable/Enable FAILURE CASE DIAGRAM

Fail Case 4:
Account status did not change from the Data Store
due to system errors/connection failures

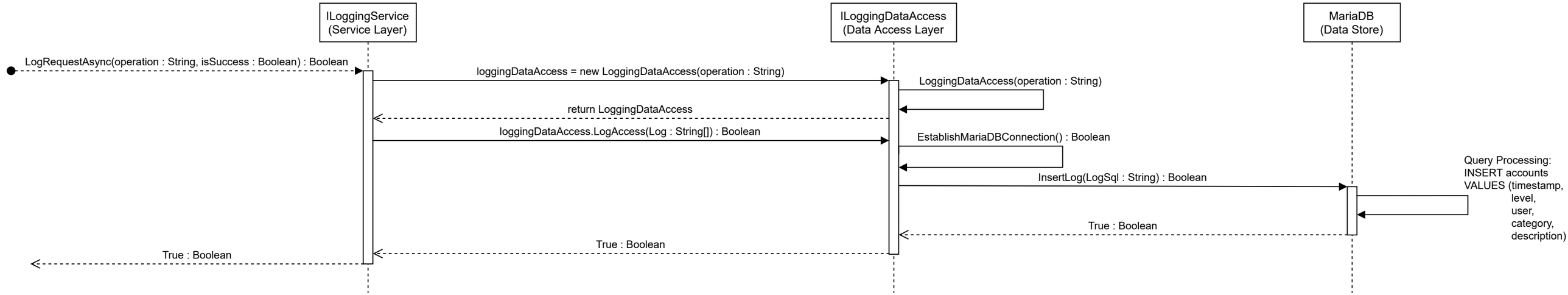


Disable/Enable FAILURE CASE DIAGRAM

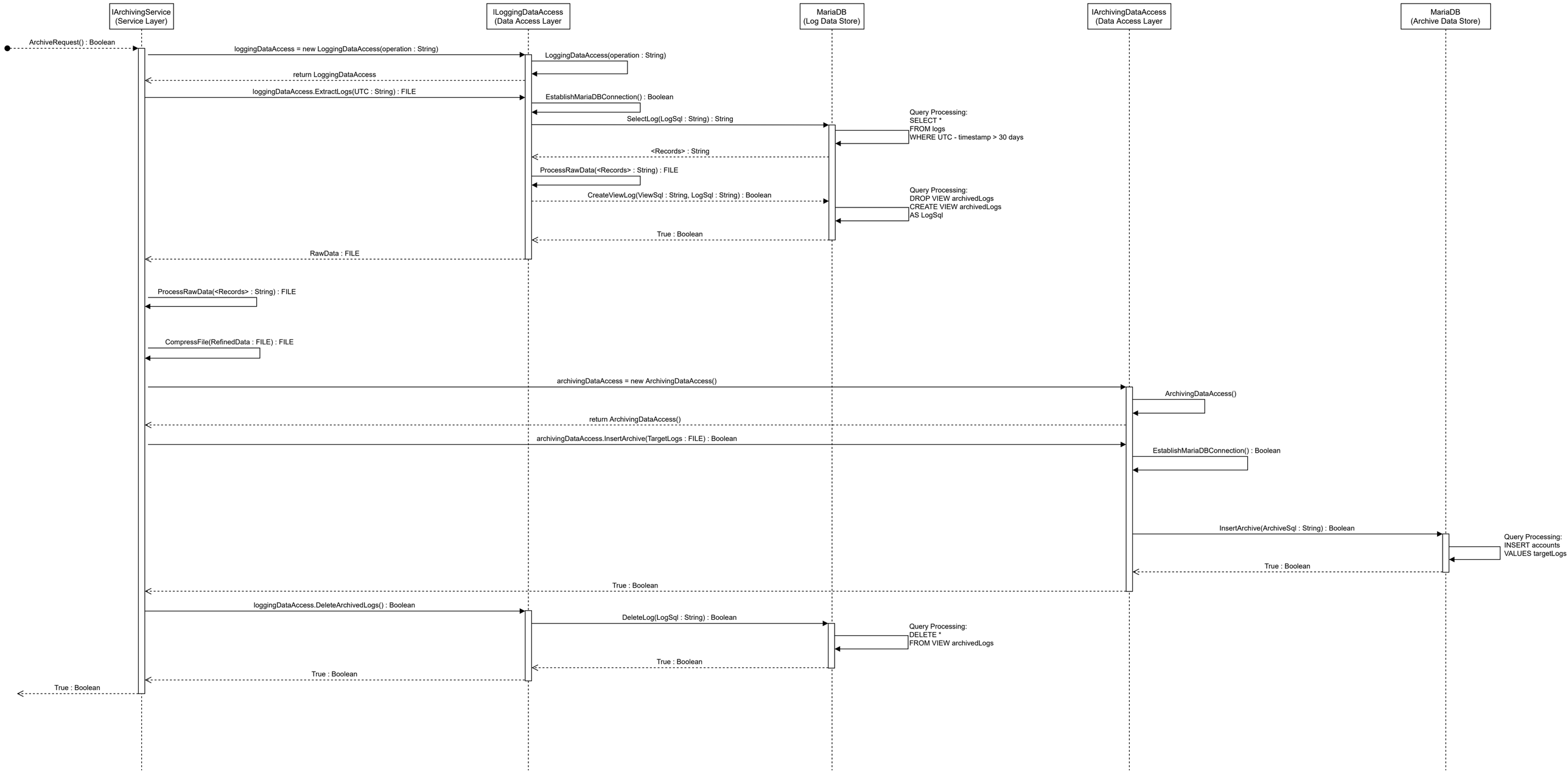
Fail Case 5:
User Management Service Creation Fails



Logging Success Diagram



Archiving Success Diagram

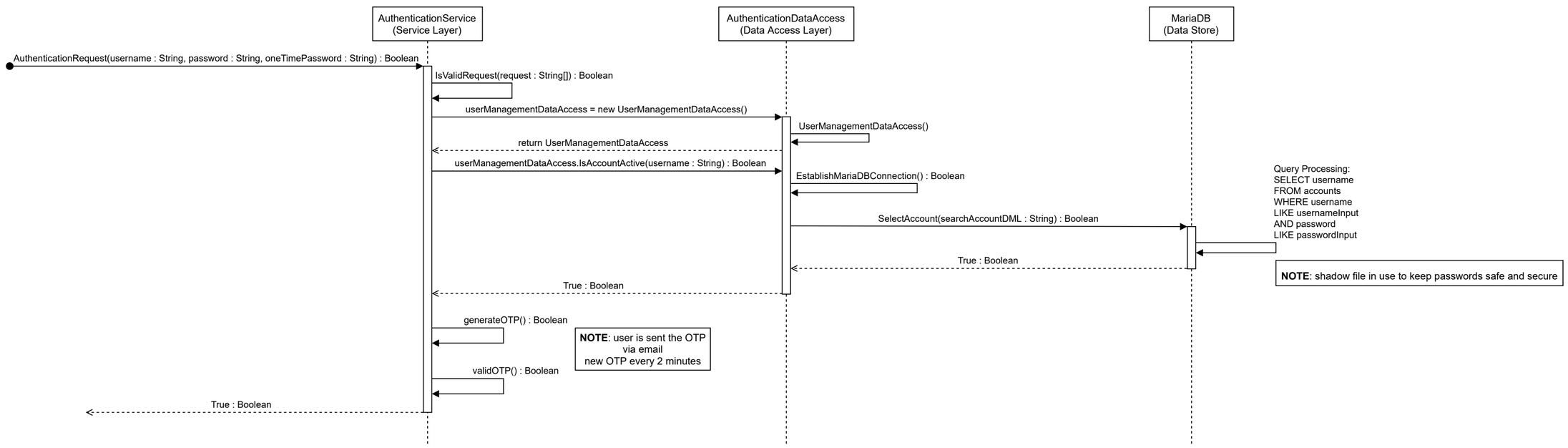


Access Control List

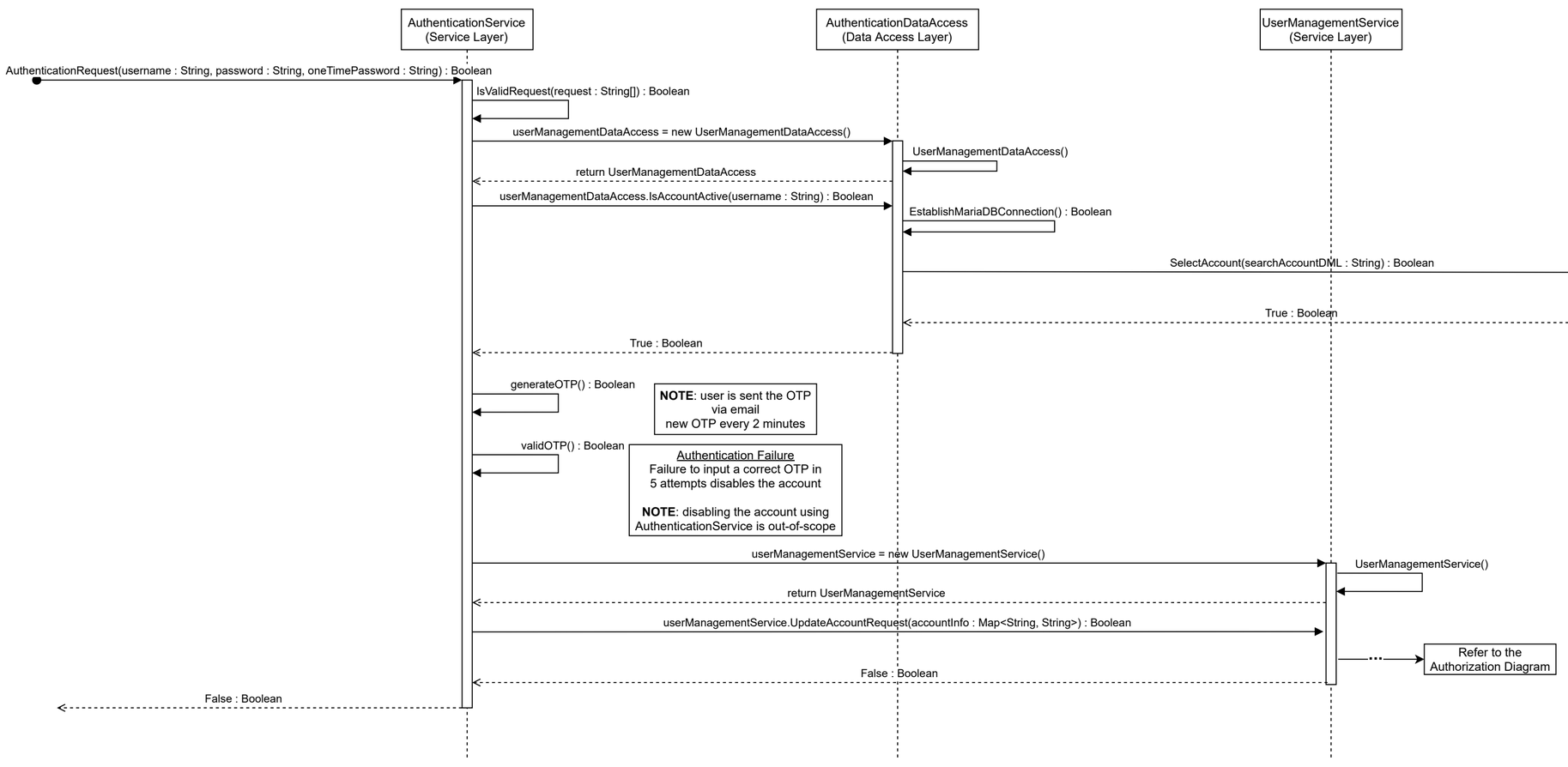
		User Management				Data Store	
User Type	Privileges	CREATE	UPDATE	DISABLE/ENABLE	DELETE	READ	WRITE
System Admin	ALL	Yes	Yes	Yes	Yes	Yes	Yes
Logged Out User Account	LOW	Yes	No	No	No	No	No
Logged In User Account	SOME	Yes	Yes	No	Yes	No	Yes
Logged In Event Account	SOME	Yes	Yes	No	Yes	No	No

Authentication Diagrams

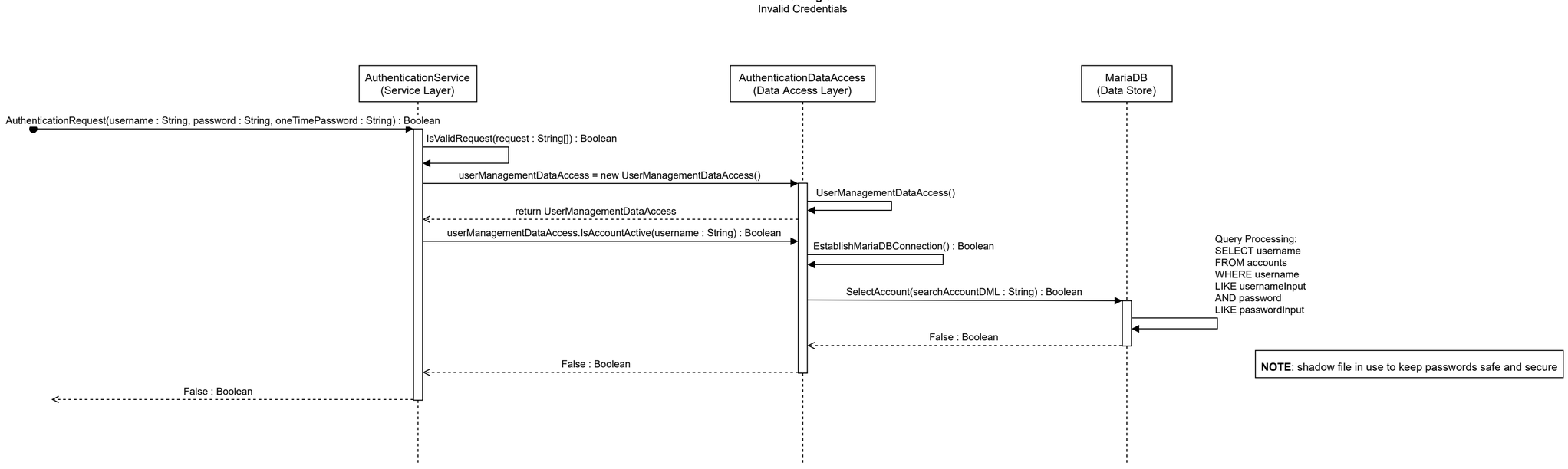
Success Diagram



Failure Diagram:
OTP Failure



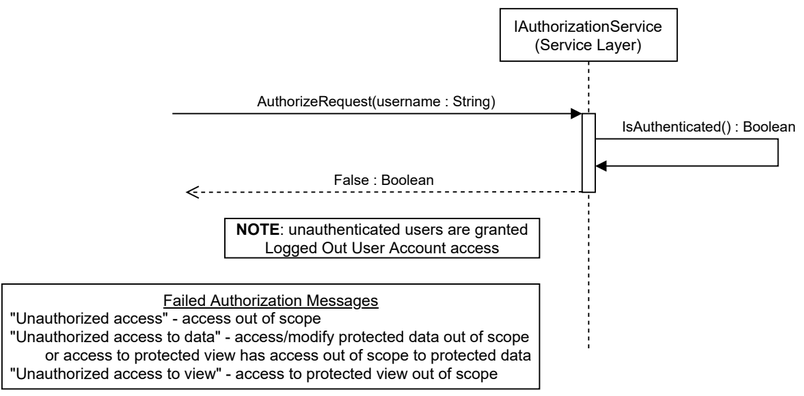
Failure Diagram:
Invalid Credentials



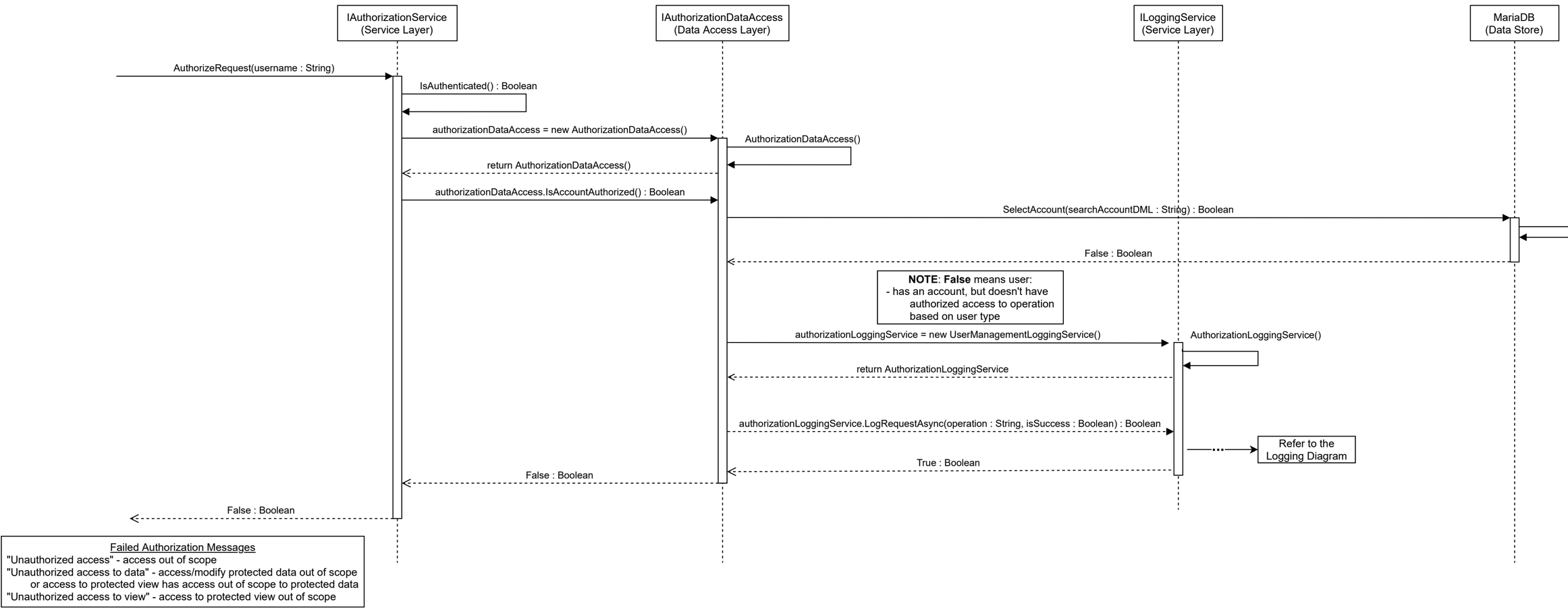
Failure Diagram:
Invalid Request

Authorization Diagrams

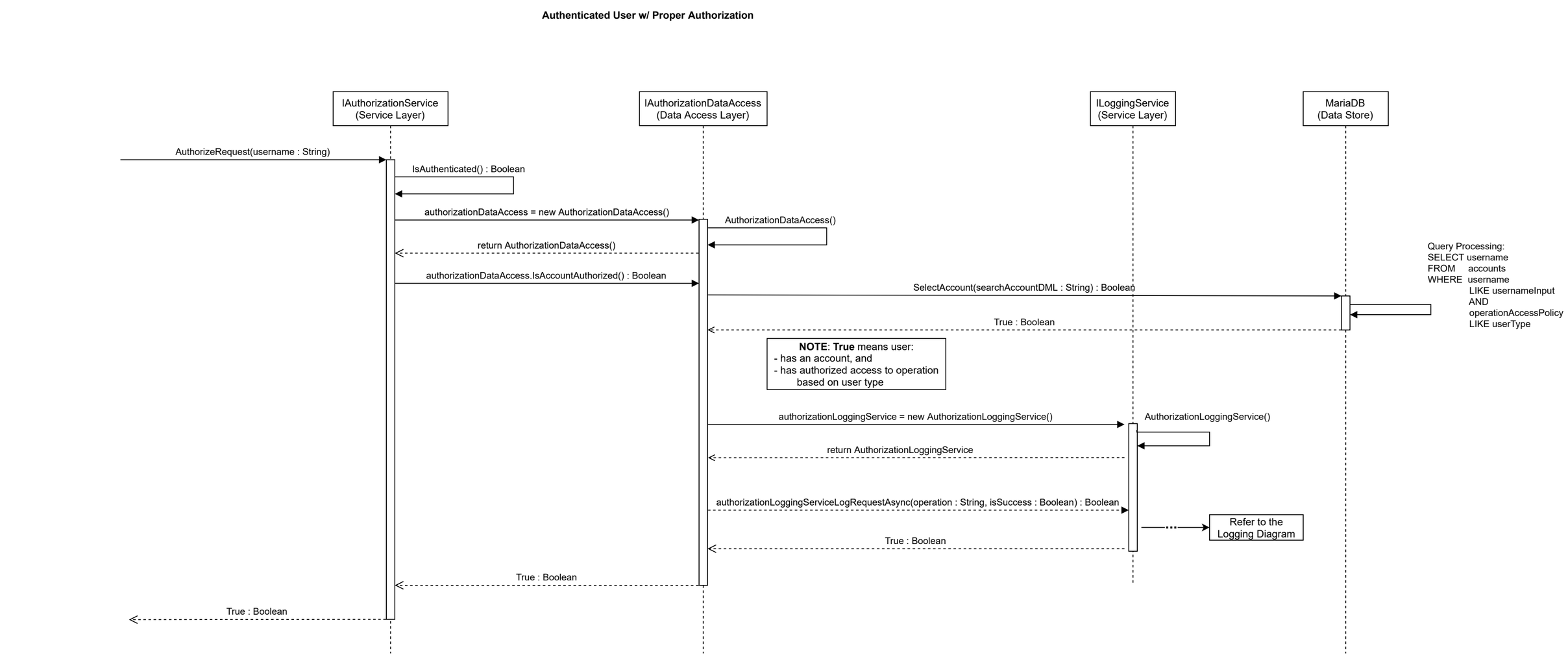
Unauthenticated User



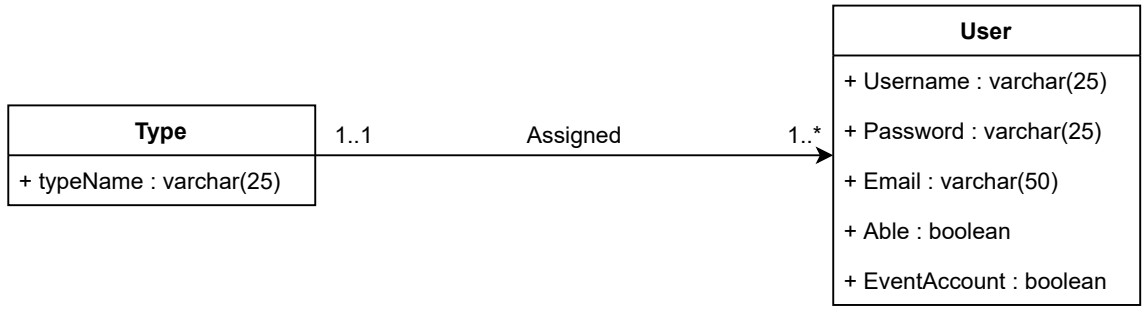
Authenticated User w/o Proper Authorization

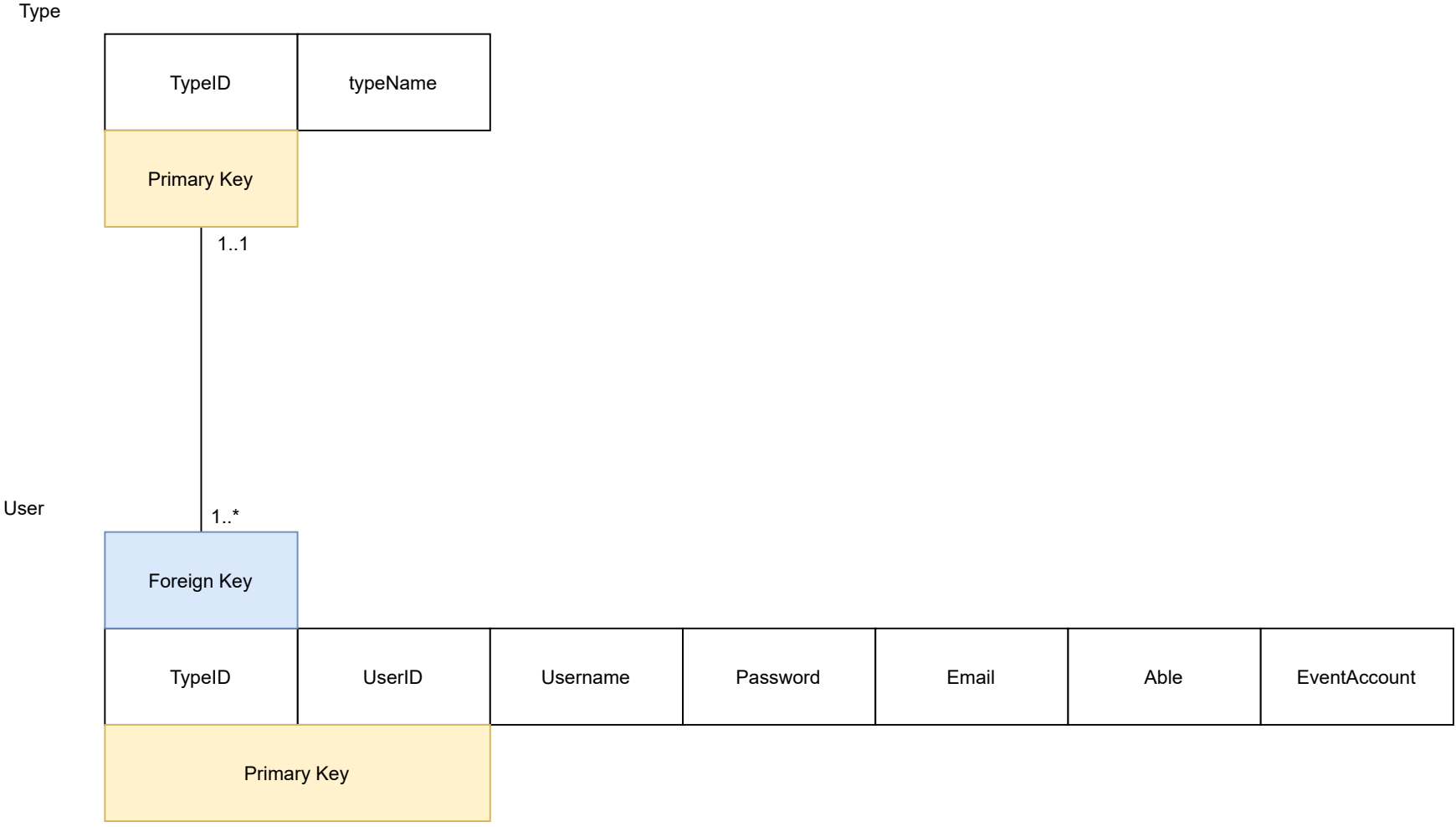


Authenticated User w/ Proper Authorization



USER MANAGEMENT DATA STORE UML and RELATION SCHEME





DATA STORE LOG UML and RELATION SCHEME