# 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**

. Overview	3
1.1 Scope	3
1.2 Purpose	3
2. User Management	3
2.1 Overview	3
2.2 Sequence Diagrams	3
2.2.1 Logging	3
3. Logging	4
3.1 Overview	4
3.2 Sequence Diagrams	4
4. Archiving	4
4.1 Overview	4
4.2 Sequence Diagrams	4
5. Appendix	4

## 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 UserManagementEntry UserManagementManager **IAuthorizationService** ILoggingService UserManagementService UserManagementDataAccess MariaDB (Application Layer) (Business Layer) (Service Layer) (Service Layer) (Service Layer) (Data Access Layer) (Data Store) SingleOperationRequest(CREATE : String) : String userManagementManager = new UserManagementManager() UserManagementManager() return UserManagementManager userManagementManager.CreateAccountInput(accountInfo: Map<String, String>): Boolean authorizationService = new UserManagementAuthorizationService(operation : String) UserManagementAuthorizationService(operation : String) return UserManagementAuthorizationService AuthorizeRequest(username : String) : Boolean Refer to the True : Boolean Authorization Diagram IsValidRequest(request : String[]) : Boolean userManagementService = new UserManagementService() teturn UserManagementService user Management Service. Create Account Request (account Info: Map < String, String >): BooleanuserManagementDataAccess = new UserManagementDataAccess() UserManagementDataAccess() return UserManagementDataAccess userManagementDataAccess.IsAccountActive(username : String) : Boolean EstablishMariaDBConnection(): Boolean Query Processing: SELECT username FROM accounts WHERE username SelectAccount(searchAccountDML : String) : Boolean LIKE usernameInput False : Boolean False : Boolean Query Processing: INSERT accounts userManagementDataAccess.CreateAccount(accountInfo: Map<String, String>): Boolean VALUES (typeInput, InsertAccount(createAccountDML : String) : Boolean usernameInput, passwordInput, True : Boolean emailInput, True : Boolean ableInput) userManagementLoggingService = new UserManagementLoggingService() UserManagementLoggingService() return UserManagementLogging\$ervice LogRequestAsync(operation : String, isSuccess : Boolean) : Boolean Refer to the Logging Diagram True : Boolean True : Boolean "UM operation was successful" : String Success Scenario 1. Single Request is made for the CREATE operation 2. New instance of CreateAccountManager is created Input account info 4. Check if request is valid, the following is checked: does the input follow the appropriate regex
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. Check if the account already exists using the username 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 (typeInput, usernameInput, passwordInput, emailInput, 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 13 New Account is not Inserted into MariaDB Reasons: incorrect DML syntax - MariaDB fails either by connection timeout or external factors JserManagementManager UserManagementService UserManagementEntry IAuthorizationService UserManagementDataAccess ILoggingService MariaDB (Application Layer) (Business Layer) (Data Access Layer) (Service Layer) (Data Store) (Service Layer) (Service Layer) SingleOperationRequest(CREATE : String) : Boolean userManagementManager = new UserManagementManager() UserManagementManager() UserManagementManager user Management Manager. Create Account Input (account Info: Map < String, String >): BooleanauthorizationService = new UserManagementAuthorizationService(operation : String) UserManagementAuthorizationService(operation : String) return UserManagementAuthorizationService authorizationService.AuthorizeRequest(username : String) : Boolean Refer to the True: Boolean Authorization Diagram IsValidRequest(request : String[]) : Boolean userManagementService = new UserManagementService() return CreateAccountService userManagementService.CreateAccountRequest(accountInfo : Map<String, String>) : Boolean userManagementDataAccess = new UserManagementDataAccess() UserManagementDataAccess() return UserManagementDataAccess userManagementDataAccess.lsAccountActive(username : String) : Boolean EstablishMariaDBConnection(): Boolean Query Processing: SELECT username FROM accounts WHERE username SelectAccount(searchAccountDML : String) : Boolean LIKE usernameInput False : Boolean Query Processing: INSERT accounts userManagementDataAccess.CreateAccount(accountInfo : Map<String, String>) : Boolean VALUES (typeInput, usernameInput, InsertAccount(createAccountDML : String) : Boolean passwordInput, emailInput, ableInput) False: Boolean userManagementLoggingService = new UserManagementLoggingService()

UserManagementLoggingService() return UserManagementLoggingService LogRequestAsync(operation : String, isSuccess : Boolean) : Boolean Refer to the Logging Diagram True : Boolean False : Boolean False : Boolean False : Boolean "UM operation not successful" : String Fail Case 10a New Account already exists in MariaDB Reasons: account already exists UserManagementService UserManagementManager UserManagementEntry IAuthorizationService ILoggingService UserManagementDataAccess MariaDB (Service Layer) (Application Layer) (Business Layer) (Service Layer) (Service Layer) (Data Access Layer) (Data Store) SingleOperationRequest(CREATE : String) : Boolean userManagementManager = new CreateAccountManager() return userManagementManager user Management Manager. Create Account Input (account Info: Map < String, String >): BooleanauthorizationService = new UserManagementAuthorizationService(operation : String) UserManagementAuthorizationService(operation : String) return UserManagementAuthorizationService authorizationService.AuthorizeRequest(username : String) : Boolean Refer to the True : Boolean Authorization Diagram IsValidRequest(request : String[]) : Boolean userManagementService = new userManagementService() return UserManagementService  $user Management Service. Create \cite{Account} Request (account Info: Map < String, String >): Boolean$ userManagementDataAccess = new userManagementDataAccess() UserManagementDataAccess() return UserManagementDataAccess user Management Data Access. Is Account Active (username: String): BooleanEstablishMariaDBConnection(): Boolean SelectAccount(searchAccountDML : String) : Boolean userManagementLoggingService = new UserManagementLoggingService() UserManagementLoggingService() return UserManagementLoggingService LogRequest A sync (operation: String, is Success: Boolean): BooleanRefer to the Logging Diagram True : Boolean True : Boolean False : Boolean False : Boolean "UM operation not successful" : String Fail Case 10b Could Not Process Query Reasons: incorrect DML syntax - MariaDB fails either by connection timeout or external factors UserManagementEntry (Application Layer) UserManagementManager ILoggingService (Service Layer) (Business Layer) UserManagementService UserManagementDataAccess MariaDB IAuthorizationService (Data Access Layer) (Data Store) (Service Layer) (Service Layer) SingleOperationRequest(CREATE : String) : Boolean userManagementManager = new UserManagementManager() return UserManagementManager userManagementManager.CreateAccountInput(accountInfo: Map<String, String>): Boolean authorizationService = new UserManagementAuthorizationService(operation : String) UserManagementAuthorizationService(operation : String) return UserManagementAuthorizationService authorizationService.AuthorizeRequest(username : String) : Boolean Refer to the True : Boolean Authorization Diagram IsValidRequest(request : String[]) : Boolean userManagementService = new UserManagementService() UserManagementService() return CreateAccountService userManagementService.CreateAccountRequest(accountInfo: Map<String, String>): Boolean userManagementDataAccess = new UserManagementDataAccess() UserManagementDataAccess() return UserManagementDataAccess userManagementDataAccess.lsAccountActive(username : String) : Boolean Query Processing: SELECT username EstablishMariaDBConnection(): Boolean FROM accounts WHERE username SelectAccount(searchAccountDML : String) : Boolean LIKE usernameInput Error : String userManagementLoggingService = new UserManagementLoggingService() UserManagementLoggingService() return UserManagementLoggingService LogRequestAsync(operation : String, isSuccess : Boolean) : Boolean Refer to the Logging Diagram True: Boolean Error : String False : Boolean False : Boolean "UM operation not successful" : String Fail Case 9 MariaDB Connection could not be Established - 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 UserManagementManager UserManagementEntry IAuthorizationService UserManagementService ILoggingService UserManagementDataAccess MariaDB (Application Layer) (Business Layer) (Service Layer) (Data Access Layer) (Data Store) (Service Layer) (Service Layer) SingleOperationRequest(CREATE : String) : Boolean userManagementManager = new UserManagementManager() UserManagementManager() return UserManagementManager userManagementManager.CreateAccountInput(accountInfo: Map<String, String>): Boolean authorizationService = new UserManagementAuthorizationService(operation : String) UserManagementAuthorizationService(operation : String) return UserManagementAuthorizationService authorizationService.AuthorizeRequest(username : String) : Boolean True: Boolean Authorization Diagram IsValidRequest(request : String[]) : Boolean userManagementService = new UserManagementService() UserManagementService() return UserManagementService userManagementService.CreateAccountRequest(accountInfo: Map<String, String>): Boolean userManagementDataAccess = new UserManagementDataAccess() UserManagementDataAccess() return CreateAccountDataAccess EstablishMariaDBConnection(): Boolean False : Boolean userManagementLoggingService = new UserManagementLoggingService() UserManagementLoggingService() return UserManagementLoggingService LogRequest A sync (operation: String, is Success: Boolean): BooleanRefer to the Logging Diagram True : Boolean False : Boolean False : Boolean "UM operation not successful" : String 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

> ILoggingService (Service Layer)

> > UserManagementLoggingService()

Refer to the Logging Diagram

UserManagementDataAccess

(Data Access Layer)

UserManagementService

(Service Layer)

MariaDB

(Data Store)

**IAuthorizationService** 

(Service Layer)

userManagementLoggingService = new UserManagementLoggingService()

LogRequestAsync(operation : String, isSuccess : Boolean) : Boolean

True : Boolean

return UserManagementLoggingService

UserManagementAuthorizationService(operation : String)

Refer to the

Authorization Diagram

UserManagementManager

(Business Layer)

**▶** UserManagementManager()

IsValidRequest(request : String[]) : Boolean

authorizationService = new UserManagementAuthorizationService(operation : String)

authorizationService.AuthorizeRequest(username : String) : Boolean

True : Boolean

return UserManagementAuthorizationService

userManagementManager = new UserManagementManager()

user Management Manager. Create Account Input (account Info: Map < String, String >): Boolean

False : Boolean

return UserManagementManager

UserManagementEntry

(Application Layer)

SingleOperationRequest(CREATE : String) : Boolean

"UM operation not successful" : String

Query Processing: SELECT username

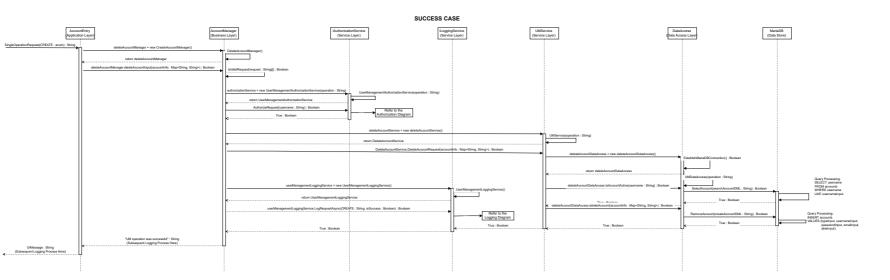
FROM accounts
WHERE username

LIKE usernameInput

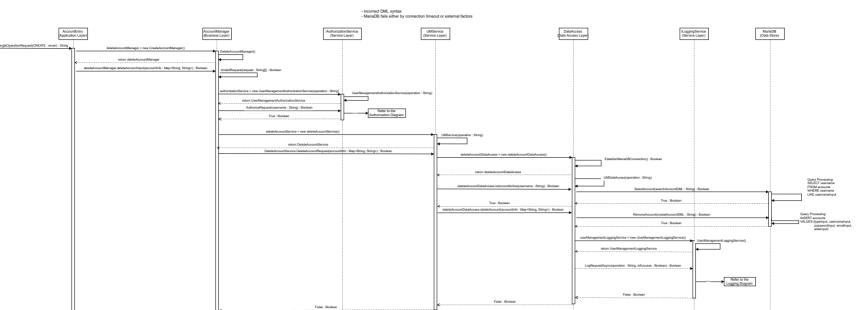
SUCCESS CASE DIAGRAM ILoggingService (Service Layer) UserManagementDataAccess MariaDB UserManagementService UserManagementEntry (Application Layer) UserManagementManager (Service Layer) (Data Access Layer) (Data Store) (Business Layer) SingleOperationRequest(UPDATE : enum) : Boolean userManagementManager = new UserManagementManager() return UserManagementManager userManagementInput.UserManagementInput(accountInfo : Map<String, String>) : Boolean **Success Scenario** IsValidRequest(request : String[]) : Boolean 1. Single Request is made for the UPDATE operation 2. New instance of UpdateAccountManager is successfully created 3. Input car info return UserManagementService 4. Check if request is valid, the following is checked: does the input follow the appropriate regex
does the input contain all necessary info for the operation request userManagementDataAccess = new UserManagementDataAccess() UserManagementDataAccess() 5. New instance of UpdateAccountService is successfully created return UserManagementDataAccess 6. Call the UpdateAccount Service to process operation with the validated input userManagementService = new UserManagementService() 7. New instance of UpdateAccountDataAccess is successfully created userManagementDataAccess.IsAccountActive(username : String) : Boolean 8. Check if the car already exists using the username
9. Establish the MariaDB connection in DataAccess if not already opened EstablishMariaDBConnection(): Boolean Query Processing: SELECT user 10. Perform a SELECT query on the Accounts table in the database: FROM accounts SELECT car WHERE userName LIKE userNameInput SelectAccount(searchAccountDML : String) : Boolean FROM accounts WHERE car userManagementService.UpdateAccountRequest(accountInfo : Map<String, String>) : Boolean LIKE carInput False : Boolean Returns FALSE if not found. Query Processing: INSERT updatedInfo Make Updates 11. Return FALSE to the Service if no car already exists, and continue userManagementDataAcess.UpdateAccount(accountInfo : Map<String, String>) : Boolean VALUES (userName, email, 12. Create the car with the info given by the user Save Updates userPassword, userCar 13. Perform a INSERT query on the Accounts table in the database: True : Boolean True : Boolean INSERT accounts (Subsequent Logging Process Here) VALUES (typeInput, carYear, carMake, carModel, True : Boolean userManagementLoggingService = new UserManagementLoggingService True : Boolean UserManagementLoggingService() (Subsequent Logging Process Here) Returns TRUE if car is successfully created and saved to account 14. Return TRUE to the Service if the car was successfully created return UserManagementLoggingService 15. Return TRUE to the Manager if the request was successfully performed16. Return TRUE to System and follow with the appropriate Logging for this operation user Management Logging Service. LogRequest Async (UPDATE: String, is Success: Boolean): Booleanlogging diagram True : Boolean SUCCESS CASE DIAGRAM w/ Logging UserManagementEntry (Application Layer) SingleOperationRequest(UPDATE : enum) : String updateAccountManager = new UpdateAccountManager() UserManagementService UserManagementDataAccess (Data Access Layer) MariaDB (Data Store) ILoggingService (Service Layer) return UpdateAccountManager (Service Layer) updateAccountManager.updateAccountInput(accountInfo : Map<String, String>) : Boolean IsValidRequest(request : String[]) : Boolean UMService(operation : String) updateAccountService = new UpdateAccountService() return UpdateAccountService update Account Service. Update Account Request (account Info: Map < String, String >): BooleanupdateAccountDataAccess = new UpdateAccountDataAccess() UMDataAccess(operation : String) return UpdateAccountDataAccess update Account Data Access. Is Account Active (username: String): BooleanEstablishMariaDBConnection(): Boolean Query Processing: SELECT user FROM accounts WHERE userName SelectAccount(searchAccountDML : String) : Boolean LIKE userNameInput True : Boolean True : Boolean Query Processing: INSERT updatedInfo updateAccountDataAccess.UpdateAccount(accountInfo : Map<String, String>) : Boolean VALUES (userName, email, InsertAccnt(updateAccountDML : String) : Boolean userPassword, userCar True : Boolean True : Boolean True : Boolean UserManagementLoggingDataAccess(operation : String) userManagementLoggingDataAccess = new UserManagementLoggingDataAccess(CREATE : String) : Boolean EstablishMariaDBConnection(): Boolean Query Processing: INSERT accounts InsertLog(LogSql : String) : Boolean VALUES (timestamp, user, category, description) True : Boolean "UM operation was successful" : String (Subsequent Logging Process Here) Fail Case 3 MariaDB Connection could not be Established - 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 MariaDB (Data Store) UserManagementEntry (Application Layer) UserManagementDataAccess ILoggingService (Service Layer) UserManagementService (Service Layer) (Data Access Layer) UserManagementManager (Business Layer) SingleOperationRequest(UPDATE : enum) : Boolean userManagementManager = new userManagementManager() return UserManagementManager user Management Input (account Info: Map < String, String >): BooleanIsValidRequest(request : String[]) : Boolean userManagementService = new UserManagementService()
return UserManagementService userManagementService.UserManagementRequest(accountInfo : Map<String, String>) : Boolean userManagementDataAccess = new UserManagementDataAccess() userManagementDataAccess.lsAccountActive(username : String) : Boolean return UserManagementDataAccess False : Boolean False : Boolean (Subsequent Logging Process Here) (Subsequent Logging Process Here) False : Boolean (Appropriate Logging Process Here) False : Boolean userManagementLoggingService = new UserManagementLoggingService() return UserManagementLoggingService EstablishMariaDBConnection(): Boofean "UM operation was successful" : String (Subsequent Logging Process Here) userManagementLoggingService.LogRequestAsync(UPDATE : String, isSuccess : Boolean) : Boolean Refer to logging diagram True : Boolean FAILURE CASE DIAGRAMS New Information is not Inserted into MariaDB Reasons: - incorrect DML syntax - MariaDB fails either by connection timeout or external factors MariaDB UserManagementDataAccess (Data Access Layer) ILoggingService (Service Layer) UserManagementManager UserManagementService UserManagementEntry (Data Store) (Application Layer) (Business Layer) (Service Layer) SingleOperationRequest(UPDATE : enum) : Boolean userManagementManager = new UserManagementManager() return UserManagementManager nagementManager.UpdateAccountInput(accountInfo: Map<String, String>): Boolean UserManagementService() userManagementService = new UserManagementService() return UserManagementService userManagementDataAccess = new UserManagementDataAccess() user Management Service. User Management Request (account Info: Map < String, String >): BooleanQuery Processing: SELECT username False : Boolean EstablishMariaDBConnection(): Boolean (Subsequent Logging Process Here) FROM accounts userManagementDataAccess.IsAccountActive(username : String) : Boolean return UserManagementDataAccess WHERE username LIKE usernameInput False : Boolean INSERT accounts VALUES (typeInput, usernameInput, passwordInput, emailInput ableInput) InsertAccount(updaFalse : Boolean: String) : Boolean False : Boolean False : Boolean user Management Data Access. Update Account (account Info: Map < String, String >): Boolean(Subsequent Logging Process Here) False : Boolean userManagementLoggingService = new UserManagementLoggingService() UserManagementLoggingService() False : Boolean return UserManagementLoggingService user Management Logging Service. Log Request Async (UPDATE: String, is Success: Boolean): BooleanRefer to logging diagram True : Boolean Fail Case 2
Could Not Process Query Reasons: - incorrect DML syntax - MariaDB fails either by connection timeout or external factors MariaDB (Data Store) ILoggingService (Service Layer) UserManagementService UserManagementDataAccess UserManagementManager UserManagementEntry (Application Layer) (Data Access Layer) (Business Layer) (Service Layer) SingleOperationRequest(UPDATE : enum) : Boolean userManagementManager = new UserManagementManager() return UserManagementManager IsValidRequest(request : String[]) : Boolean userManagementManager.UserManagementInput(accountInfo : Map<String, String>) : Boolean userManagementService = new UserManagementService() return UserManagementService userManagementService.UserManagementRequest(accountInfo : Map<String, String>) : Boolean userManagementDataAccess = new UserManagementDataAccess() Query Processing: SELECT username return UserManagementDataAccess FROM accounts WHERE username LIKE usernameInput Error : String Error : String False : Boolean False : Boolean (Subsequent Logging Process Here) NOTE: True is the fail safe default so a False : Boolean malicious user does not exploit the False value (Subsequent Logging Process Here) SelectAccount(searchAccountDML : String) : Boolean also we can't represent a database error in a binary result NOTE: True is the fail safe default so a malicious user does not exploit the False value also we can't represent a database error in a binary result userManagementLoggingService = new UserManagementLoggingService() return UserManagementLoggingService "UM operation was successful" : String (Subsequent Logging Process Here) Refer to logging userManagementLoggingService.LogRequestAsync(UPDATE : String, isSuccess : Boolean) : Boolean True : Boolean 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 ILoggingService UserManagementService UserManagementDataAccess MariaDB UserManagementManager UserManagementEntry (Business Layer) (Data Access Layer) (Data Store) (Service Layer) (Service Layer) SingleOperationRequest(UPDATE : enum) : Boolean updateAccountManager = new UpdateAccountManager() sValidRequest(request : String[]) : Boolean return UpdateAccountManager False : Boolean update Account Manager. Update Account Input (account Info: Map < String, String >): Boolean(Subsequent Logging Process Here) False : Boolean (Subsequent Logging Process Here) userManagementLoggingService = new UserManagementLoggingService() UserManagementLoggingService() return UserManagementLoggingService userManagementLoggingService.LogRequestAsync(UPDATE : String, is Success : Boolean) : Boolean Refer to logging diagram True : Boolean "UM operation was successful" : String (Subsequent Logging Process Here)

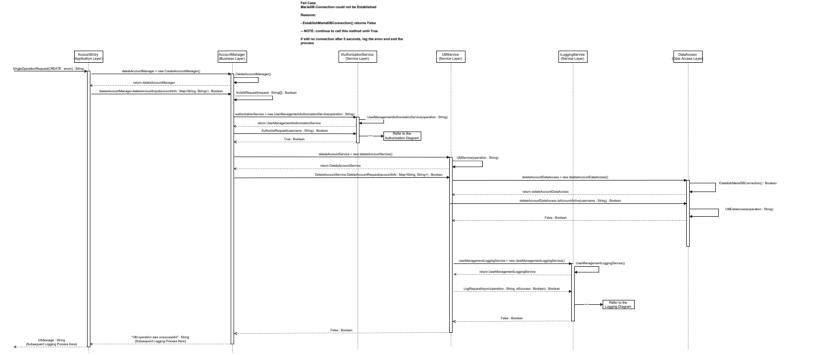
**UPDATE Operation** 

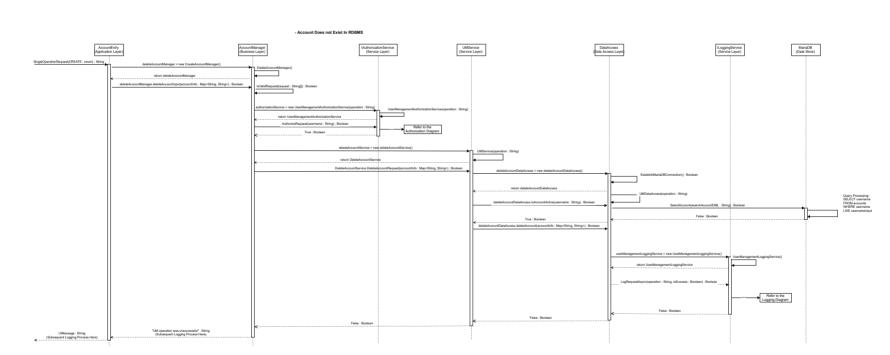
#### DELETE OPERATION

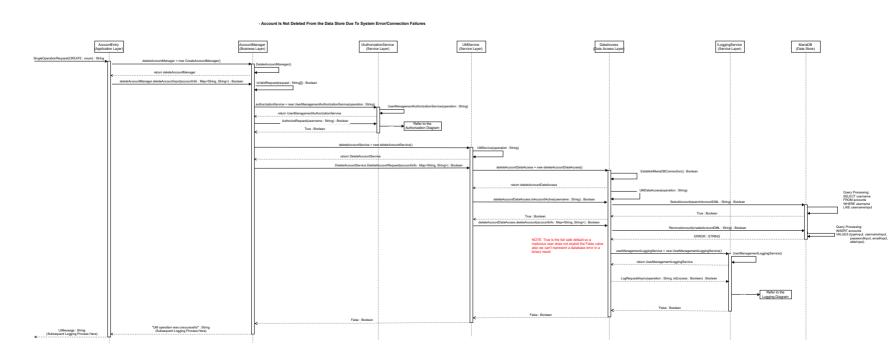


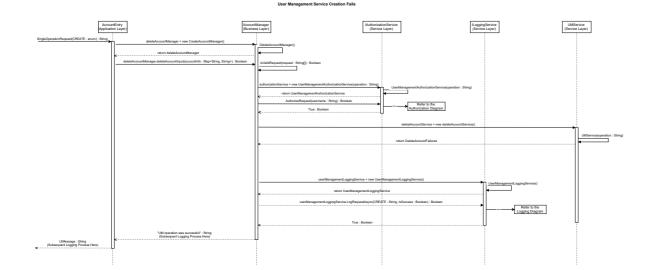
#### FAILURE CASE DIAGRAMS

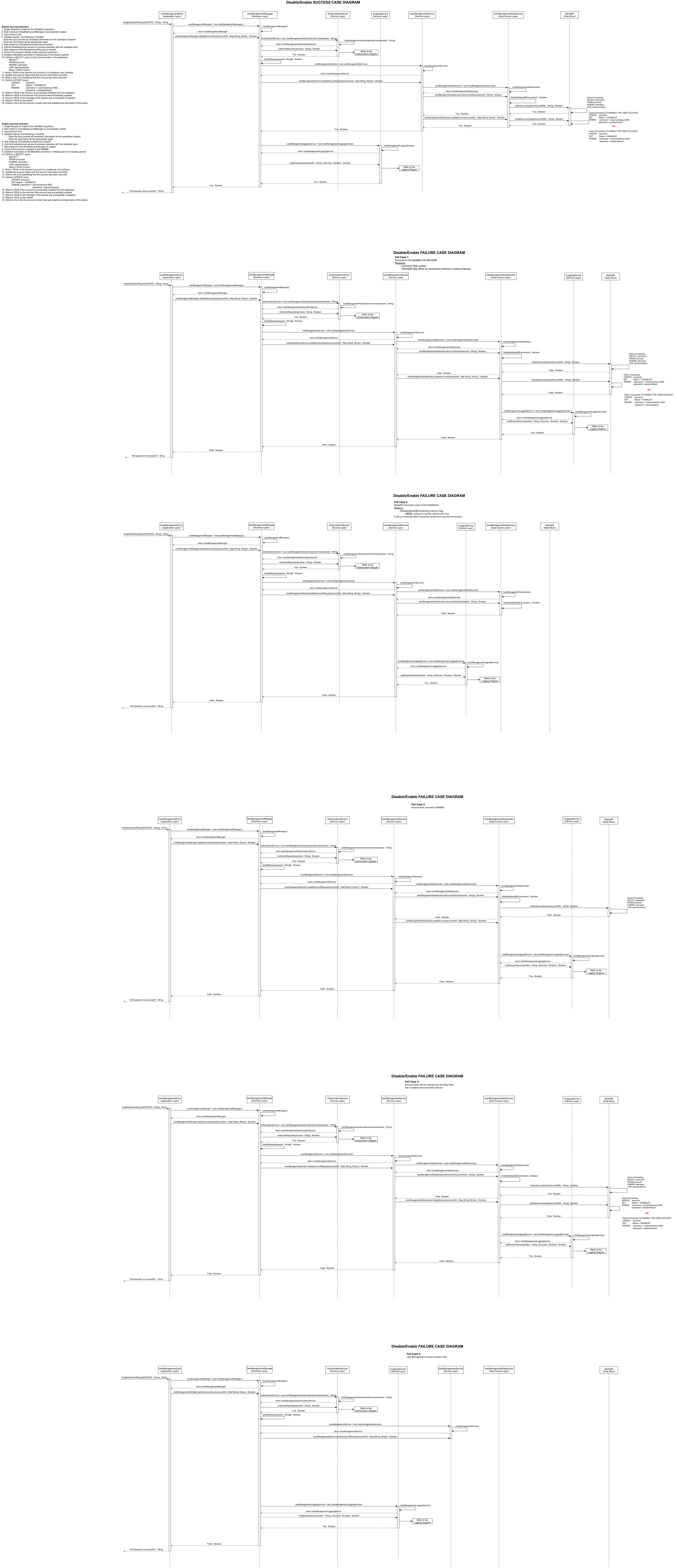












Disable Success Scenario

1. Single Request is made for the DISABLE Operation

5. New instance of DisableAccountService is created

7. New instance of the DisableAccountAccess is created

Input Account info

SELECT \* FROM accounts

Enable Success Scenario

10. Perform a SELECT query SELECT \* FROM accounts WHERE username

14. Perform UPDATE query

18. Returns TRUE to the system

LIKE usernameInput Return TRUE if found

UPDATE accounts
SET Status = 'ENABLED'

WHERE username = usernameInput AND

Input Account info

1. Single Request is made for the ENABLE Operation

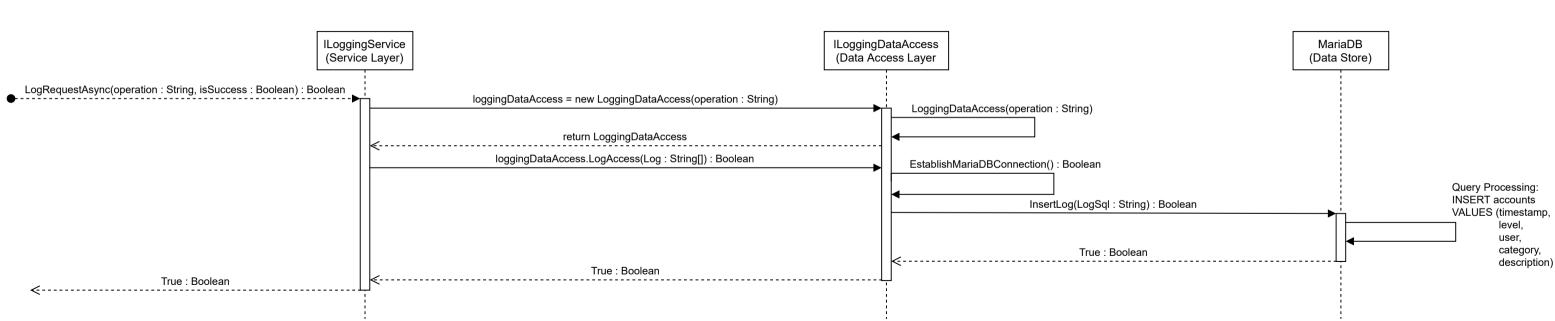
- Does the input follow all the appropriate regex 5. New instance of EnableAccountService Created

7. New instance of the EnableAccountAccess is created 8. Check if the account is existent in the RDBMS

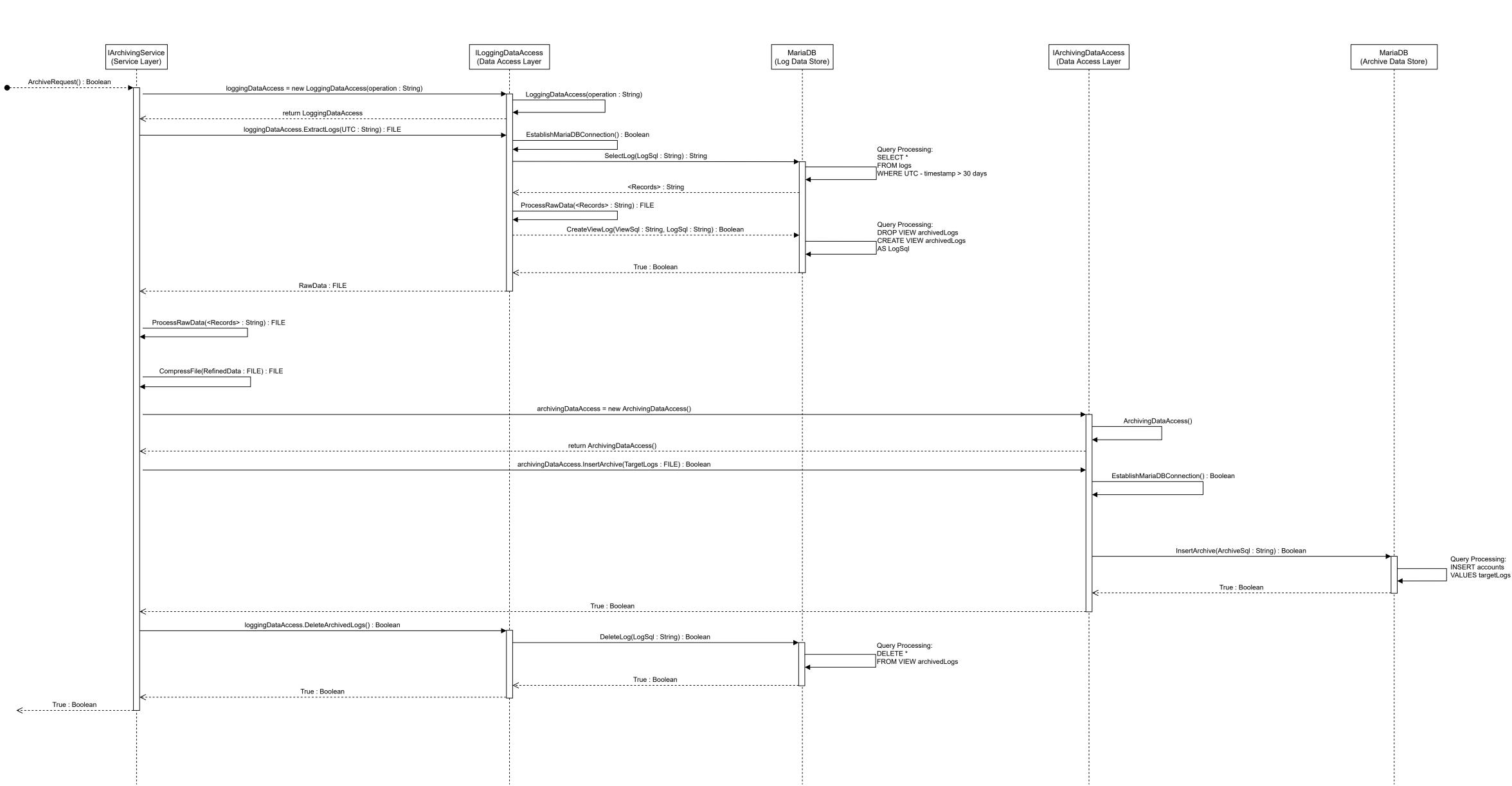
WHERE username LIKE usernameInput Return TRUE if found

# Logging s

ina Success Diagram



# **Archiving Success Diagram**

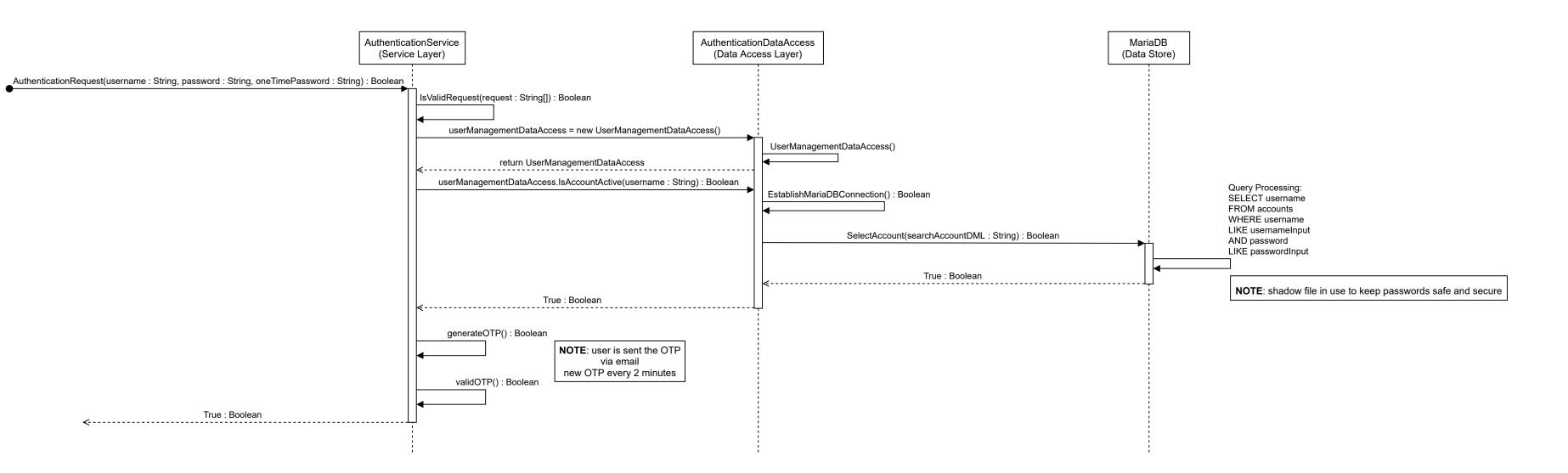


# **Access Control List**

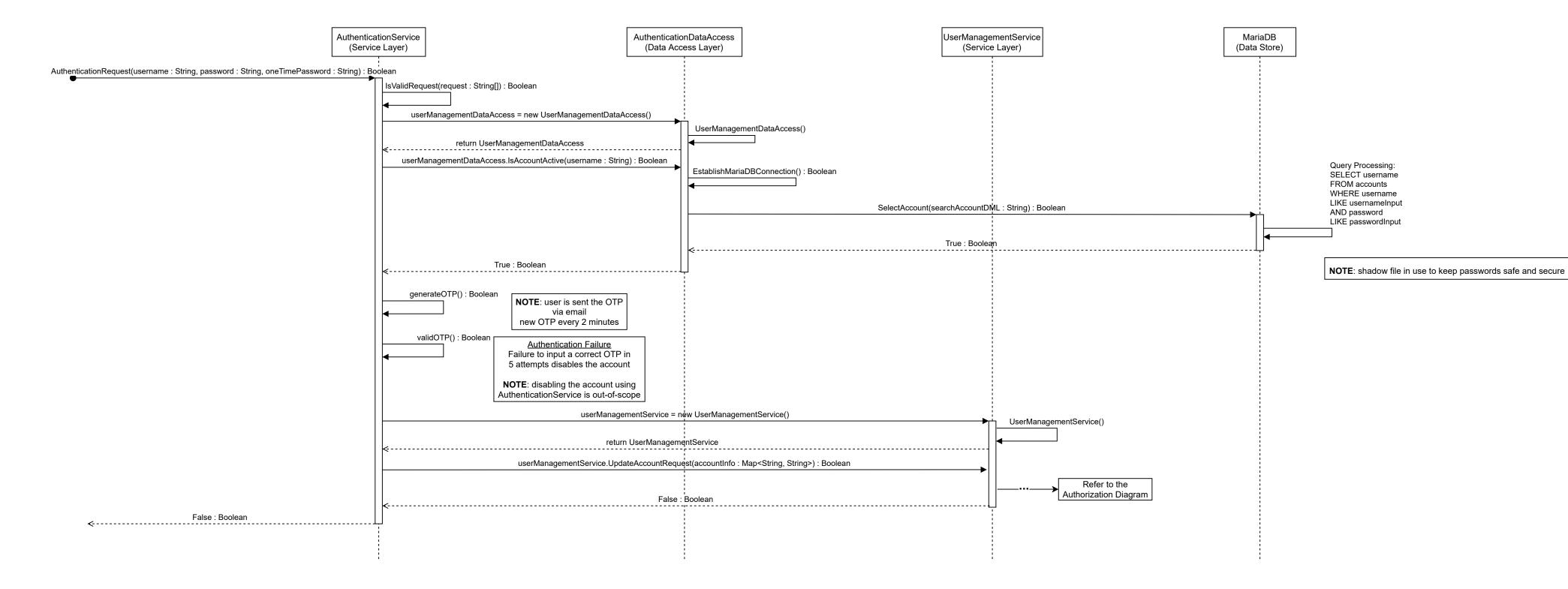
		User Management			Data Store		
User Type	Privileges	CREATE	UPDATE	DISABLE/ENABLE	DELETE	READ	WRITE
System Admin	ALL	Yes	Yes	Yes	Yes	Yes	Yes
Logged Out User Account	LOW	Yes	No	No	No	No	No
Logged In User Account	SOME	Yes	Yes	No	Yes	No	Yes
Logged In Event Account	SOME	Yes	Yes	No	Yes	No	No

# **Authentication Diagrams**

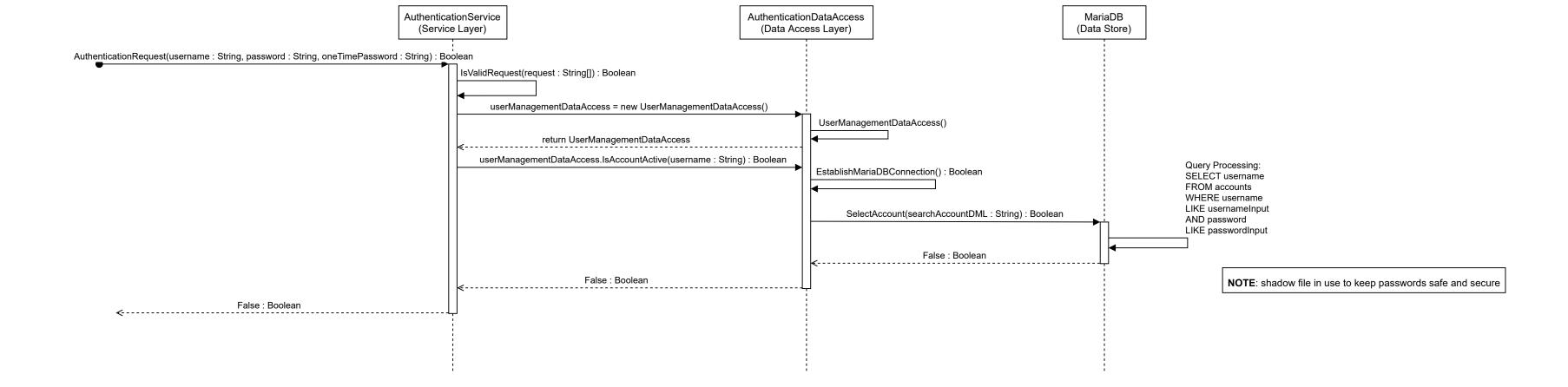
## Success Diagram



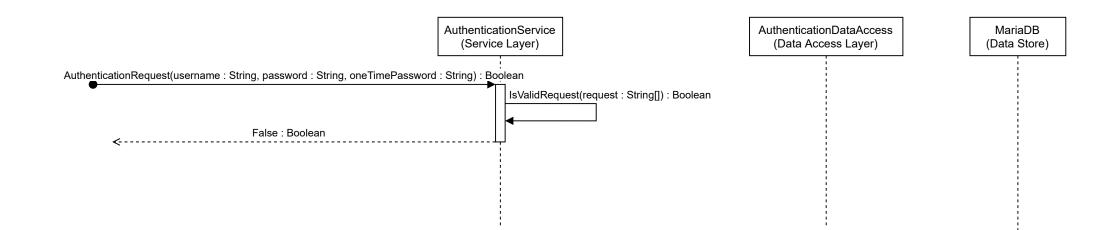
### Failure Diagram: OTP Failure



### Failure Diagram: Invalid Credentials

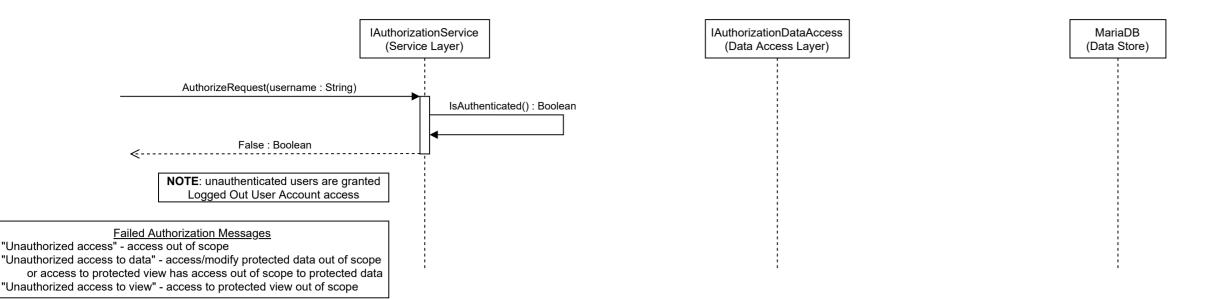


#### Failure Diagram: Invalid Request

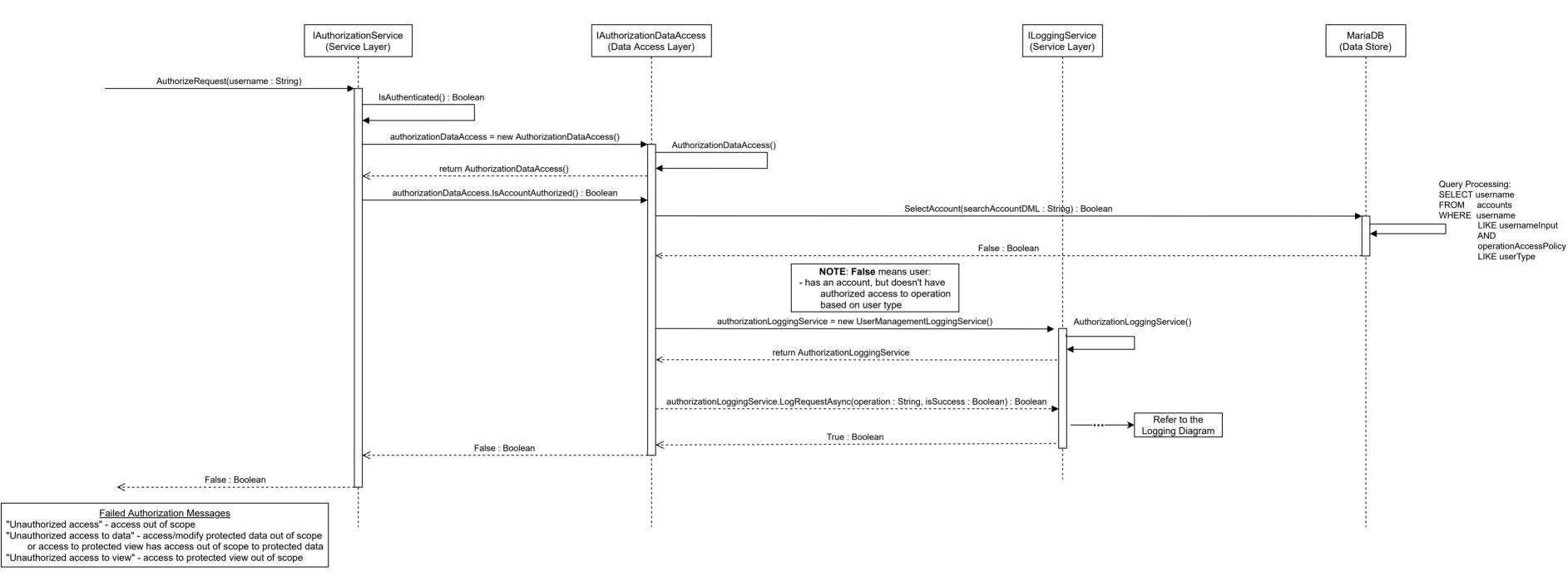


# **Authorization Diagrams**

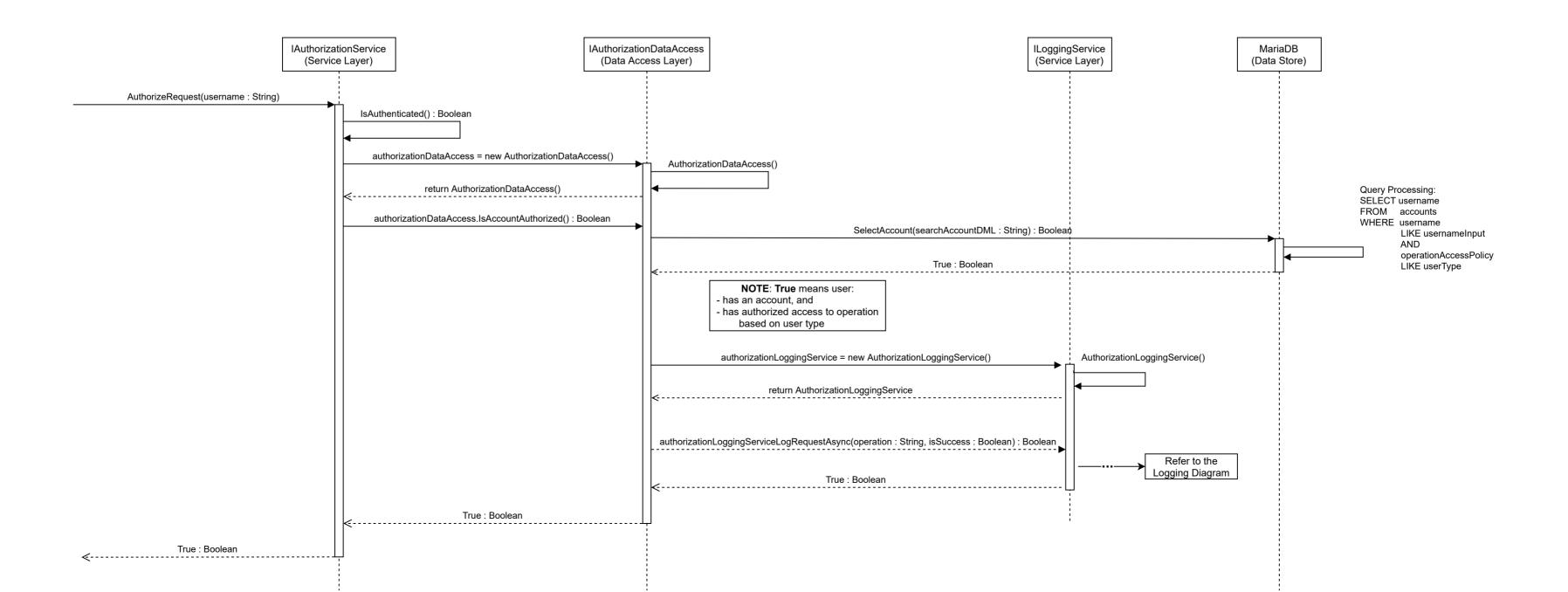
### **Unauthenticated User**



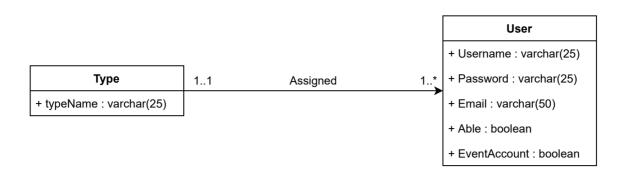
### Authenticated User w/o Proper Authorization

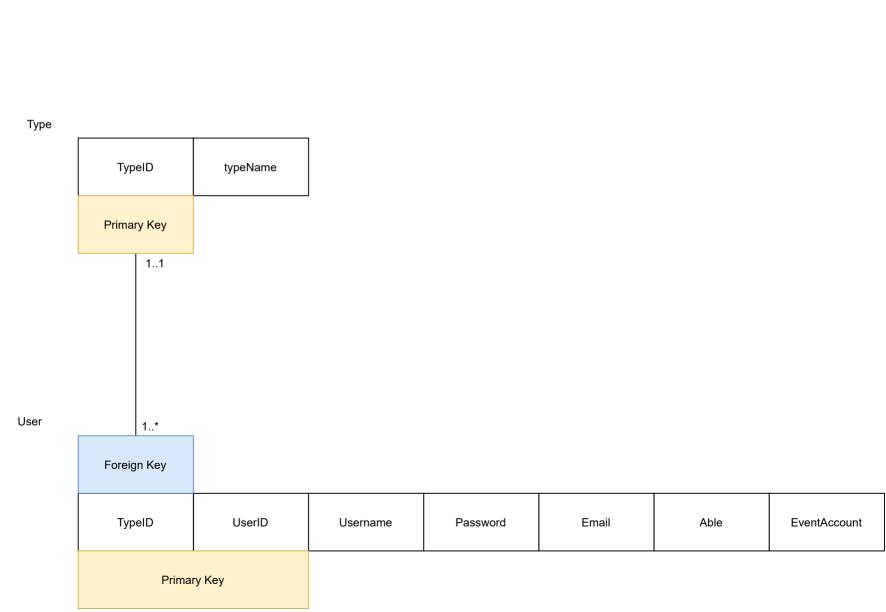


# Authenticated User w/ Proper Authorization

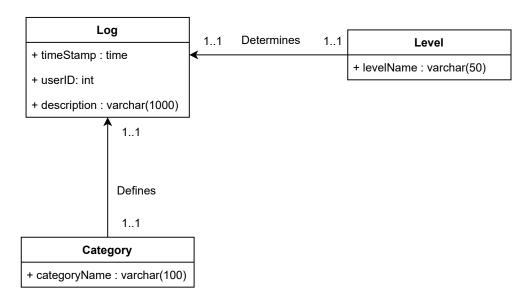


## **USER MANAGEMENT DATA STORE UML and RELATION SCHEME**





# **DATA STORE LOG UML and RELATION SCHEME**



#### Category

LevelID

levelName

