

# Low Level Diagram

November 9, 2022

## Issued by:

Algorithmic Alchemist

## Team Lead

Sierra Harris

## Team Members

Bryant Lam

Abhay Solanki

Faisal Al Muharrami

David Chan

Github: [https://github.com/abhay772/AA\\_Senior\\_Project/](https://github.com/abhay772/AA_Senior_Project/)

# Version History

Version #	Date	Reason for Change
Version 1.0.0	11/9/2022	Original Document
Version 2.0.0	12/14/2022	Added Registration, Authentication, and Authorization

# Table of Contents

<b>Version History</b>	<b>2</b>
<b>Overview</b>	<b>4</b>
<b>Registration</b>	<b>4</b>
Registration Success Case 1: User registers with a valid email and valid passphrase.	4
Registration Failure Case 1 from Invalid Account Type	7
Registration Failure Case 2 from Invalid Email	8
Registration Failure Case 3 from Invalid Passphrase	9
Registration Failure Case 4 from Invalid Date of Birth	10
Registration Failure Case 5 from Unable to assign username with valid email and passphrase	10
Registration Failure Case 6 from _registrate.CreateUser() took longer than 5 seconds before logging	12
<b>Authentication</b>	<b>14</b>
Authentication Success Case 1 from user authenticated through valid credentials	14
Authentication Success Case 2 from user is already authenticated	15
<b>Authorization</b>	<b>16</b>
Authorization Success Case 1 - 4	16
Success Cases 1: User attempts to access a protected functionality within authorization scope. Access is granted to perform functionality.	16
Success Case 2: User attempts to access protected data within authorization scope. Access is granted to perform read operations.	16
Success Case 3: User attempts to modify protected data within authorization scope. Access is granted to perform write operations.	16
Success Case 4: User attempts to access protected views within authorization scope. Access is granted to the view. User is automatically navigated to view.	16
<b>Logging</b>	<b>16</b>
Logging Success Case	17
Logging Failure Case by whole process taking longer than 5 seconds	18
Logging Failure Case by preventing user from interacting with the system	18
Logging Failure Case by failed to save log in a persistent data store	19
Logging Failure Case by inaccurately saving the event to a persistent data store	19
Logging Failure Case by modifiable log entries	20
<b>Result</b>	<b>20</b>
<b>Glossary</b>	<b>20</b>

## Overview

The Low Level Design document will describe the layer interaction of functionalities visually through UML sequence diagrams. This document is designed to help developers understand the flow and interactions of functionality through abstract layers. Layers will mainly cover the system as a whole, from frontend to backend. There will be multiple sequence diagrams for a feature to cover the success case and all failure cases derived from business rules.

## Registration

From this point on, refer to the link below for naming conventions.

Ex. Class member field called “\_noun” will be private.

[https://github.com/v-vong3/csulb/blob/master/cecs\\_491/docs/cecs491-coding-standards.pdf](https://github.com/v-vong3/csulb/blob/master/cecs_491/docs/cecs491-coding-standards.pdf)

### **Registration Success Case 1: User registers with a valid email and valid passphrase.**

The system is able to assign a system-wide unique username. A system message displays “Account created successfully” within 5 seconds of invoking the registration process. The system provides the username to the user.

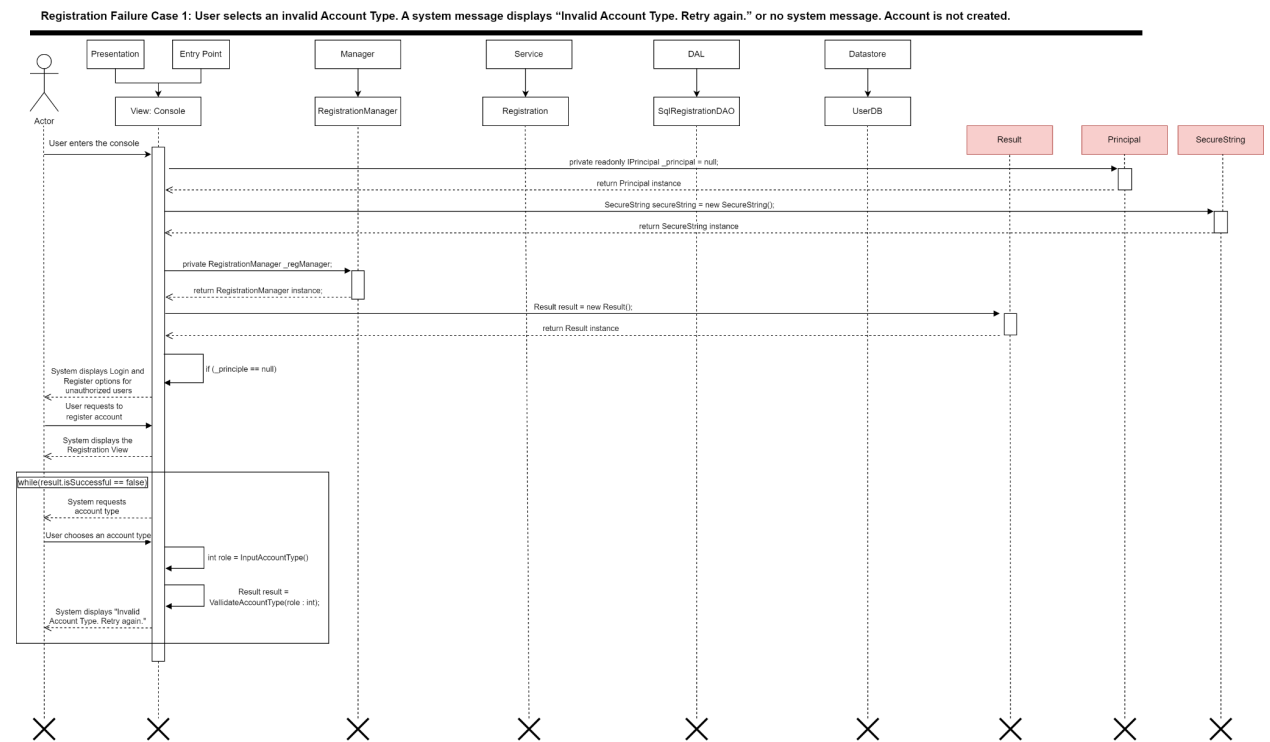
```

sequenceDiagram
    actor Actor
    participant Presentation
    participant EntryPoint as Entry Point
    participant Manager as Manager
    participant Service as Service
    participant DAL as DAL
    participant Database as Database

    Note over Actor: User enters the console
    Note over Presentation: private readonly IPincipal principal = null;
    Note over EntryPoint: return IPincipal instance
    Note over EntryPoint: SecureString secureString = new SecureString();
    Note over EntryPoint: return SecureString instance
    Note over Manager: private RegistrarManager _regManager;
    Note over Manager: return RegistrarManager instance;
    Note over Presentation: Result result = new Result();
    Note over Presentation: return Result instance
    Note over Presentation: if (_principal == null)
    Note over Presentation: System displays Login and Register options for unauthorized users
    Note over Presentation: User requests to register account
    Note over Presentation: System displays the Registration view
    Note over Presentation: System requests account type
    Note over Presentation: User chooses an account type
    Note over Presentation: int row = InputAccountType();
    Note over Presentation: Result result = ValidateAccountType(row);
    Note over Presentation: System requests User's email
    Note over Presentation: User enters email in System
    Note over Presentation: string email = InputEmail();
    Note over Presentation: Result result = ValidateEmail(email);
    Note over Presentation: System requests User's password
    Note over Presentation: User enters password in System
    Note over Presentation: SecureString password = InputPassword();
    Note over Presentation: return SecureString instance
    Note over Presentation: Result result = ValidatePassword(password);
    Note over Presentation: System requests User's date of birth
    Note over Presentation: User enters date of birth in System
    Note over Presentation: string dateOfBirth = InputBirthDay();
    Note over Presentation: Result result = ValidateDateOfBirth(dateOfBirth);
    Note over Presentation: System requests user's location
    Note over Presentation: User enters location in System
    Note over Presentation: string location = InputAddress();
    Note over Presentation: Result result = ValidateLocation(location);
    Note over Presentation: Result result = _regManager.RegistrateUser(email, string, password, string, dateOfBirth, string, location, string, role, string);
    Note over Manager: byte[] salt = GenerateSalt();
    Note over Manager: byte[] encryptedPassword = EncryptPassword(password, secureString, salt);
    Note over Manager: private readonly Result _result;
    Note over Manager: return Result instance
    Note over Manager: private Registrar _registration;
    Note over Manager: return Registrar instance;
    Note over Manager: Result registrate = _registration.CreateUser(email, string, encryptedPassword, byte[], salt, byte[], dateOfBirth, string, location, string, role, string);
    Note over Manager: private Result _result;
    Note over Manager: return Result instance;
    Note over DAL: var DAO = new SqlRegistrationDAO();
    Note over DAL: return SqlRegistrationDAO instance;
    Note over DAL: Result SaveUserAccount = DAO.SaveUserAccount(email, string, encryptedPassword, byte[], salt, byte[], dateOfBirth, string, location, string, role, string);
    Note over Database: var command = new SqlCommand(SqlQuery.connection);
    Note over Database: bool isSuccess = command.ExecuteNonQuery();
    Note over Database: return true;
    Note over Database: INSERT INTO UserAccountDB (Email, Password, Salt) VALUES (email, encryptedPassword, salt);
    Note over DAL: return SaveUserAccount.isSuccess = true;
    Note over DAL: Result SaveUserProfile = DAO.SaveUserProfile(email, string, dateOfBirth, string, location, string, role, string);
    Note over Database: var command = new SqlCommand(SqlQuery.connection);
    Note over Database: bool isSuccess = command.ExecuteNonQuery();
    Note over Database: return true;
    Note over Database: INSERT INTO UserProfileDB (Email, DateOfBirth, Location, Role) VALUES (email, dateOfBirth, location, role);
    Note over DAL: return SaveUserProfile.isSuccess = true;
    Note over Manager: if (SaveUserAccount.isSuccess == true && SaveUserProfile.isSuccess == true)
    Note over Manager: return registrateUser.isSuccess = true;
    Note over Presentation: return registrateUser.isSuccess = true;
    Note over Presentation: System displays 'Account created successfully'
    Note over Presentation: Command = email
  
```

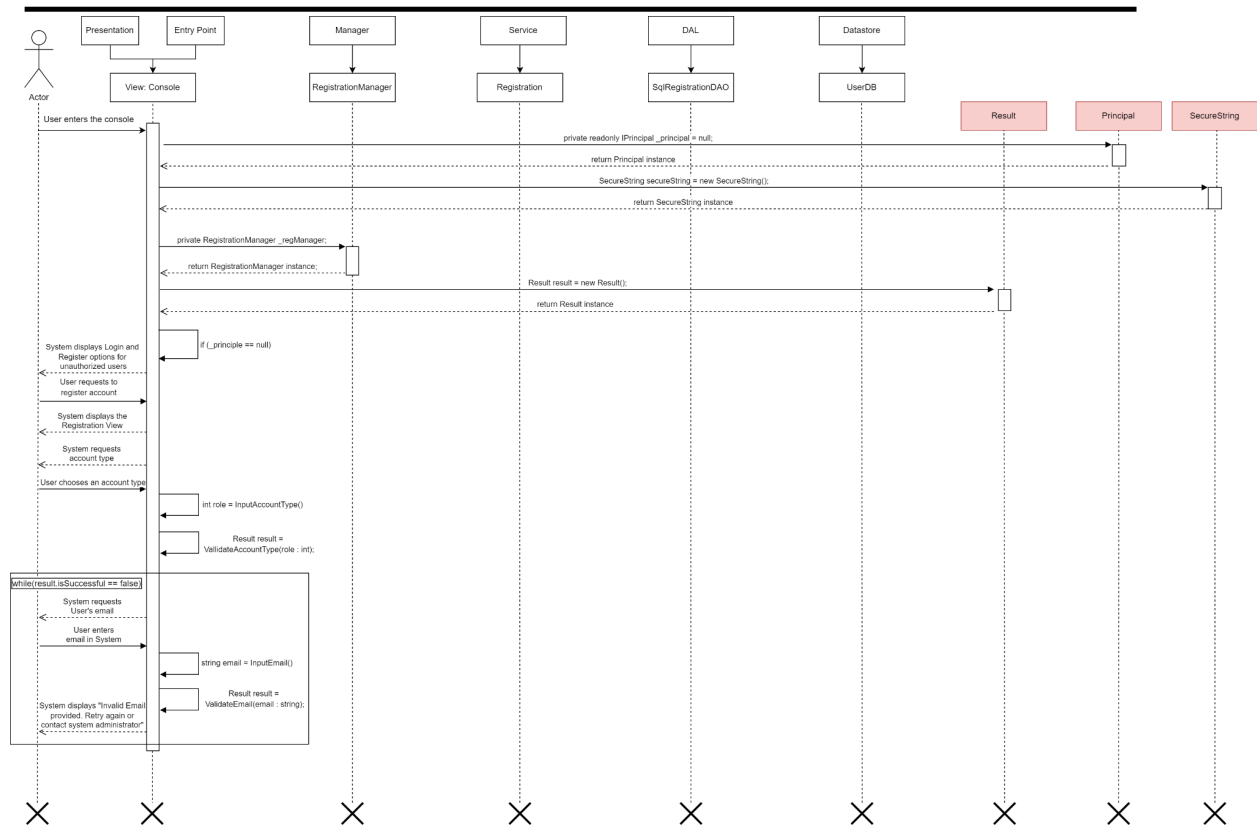
- “RegistrationDB” is the table name within a database relating to user microservices.
- Please refer to [Logging Success Case](#) for LogAsync().
- ExecuteSql(string stringConnection) will connect to the database relating to user microservices and only perform insert statements to RegistrationDB for the case of Registration.

# Registration Failure Case 1 from Invalid Account Type



## Registration Failure Case 2 from Invalid Email

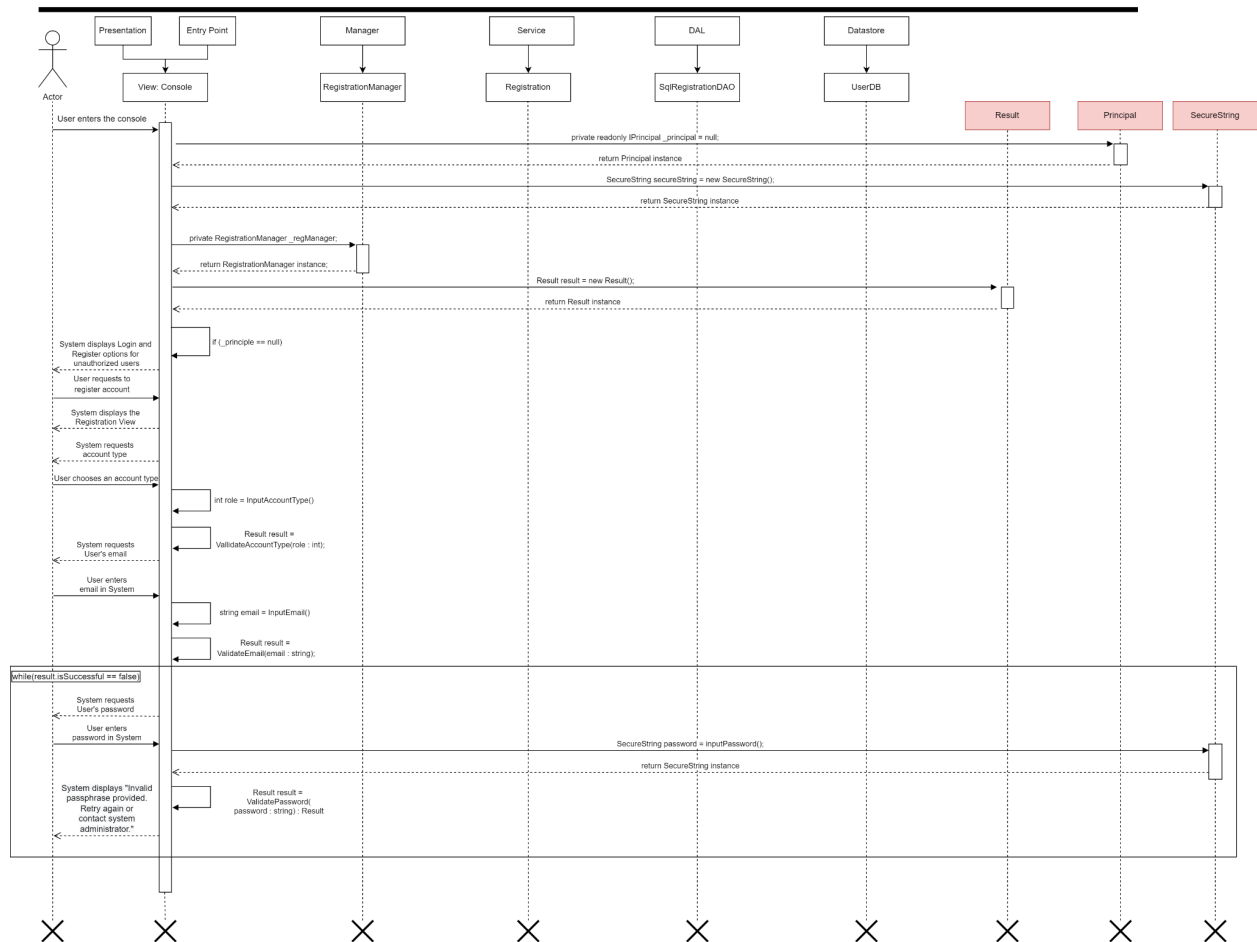
Registration Failure Case 2: User registers with an invalid email. A system message displays "Invalid email provided. Retry again or contact system administrator" or no system message. Account is not created.





### Registration Failure Case 3 from Invalid Passphrase

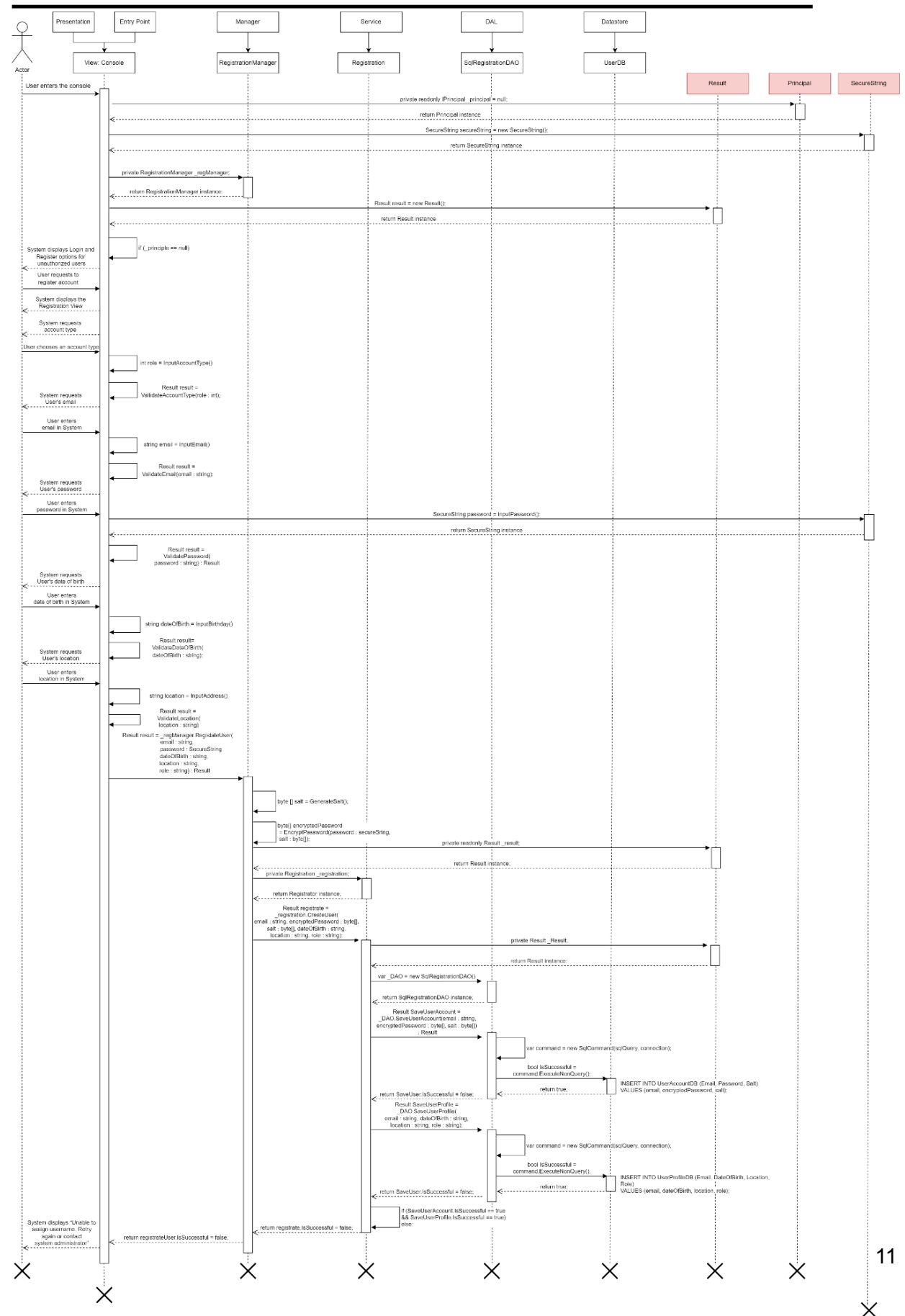
Registration Failure Case 3: User registers with an invalid passphrase. A system message displays "Invalid passphrase provided. Retry again or contact system administrator" or no system message, Account is not created.



[illegible]

10

**Registration Failure Case 5: User registers with a valid email and valid passphrase. The system was unable to assign a system-wide username.**  
A system message displays "Unable to assign username. Retry again or contact system administrator". Account is not created.



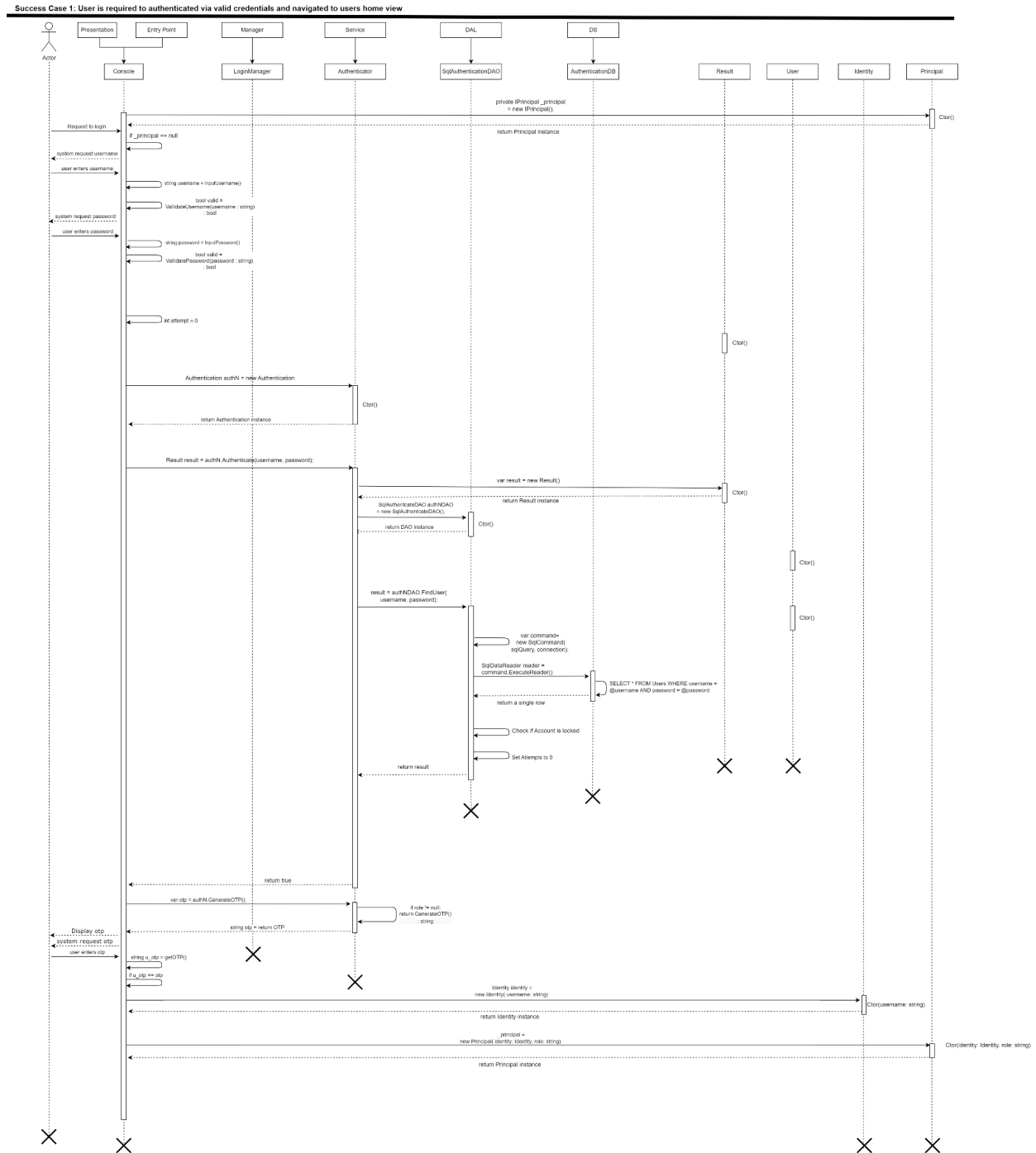
**Registration Failure Case 6 from \_registrate.CreateUser() took longer than 5 seconds before logging**

Year	2000	2001	2002	2003	2004	2005
1	100	100	100	100	100	100
2	100	100	100	100	100	100
3	100	100	100	100	100	100
4	100	100	100	100	100	100
5	100	100	100	100	100	100
6	100	100	100	100	100	100
7	100	100	100	100	100	100
8	100	100	100	100	100	100
9	100	100	100	100	100	100
10	100	100	100	100	100	100
11	100	100	100	100	100	100
12	100	100	100	100	100	100
13	100	100	100	100	100	100
14	100	100	100	100	100	100
15	100	100	100	100	100	100
16	100	100	100	100	100	100
17	100	100	100	100	100	100
18	100	100	100	100	100	100
19	100	100	100	100	100	100
20	100	100	100	100	100	100
21	100	100	100	100	100	100
22	100	100	100	100	100	100
23	100	100	100	100	100	100
24	100	100	100	100	100	100
25	100	100	100	100	100	100
26	100	100	100	100	100	100
27	100	100	100	100	100	100
28	100	100	100	100	100	100
29	100	100	100	100	100	100
30	100	100	100	100	100	100
31	100	100	100	100	100	100
32	100	100	100	100	100	100
33	100	100	100	100	100	100
34	100	100	100	100	100	100
35	100	100	100	100	100	100
36	100	100	100	100	100	100
37	100	100	100	100	100	100
38	100	100	100	100	100	100
39	100	100	100	100	100	100
40	100	100	100	100	100	100
41	100	100	100	100	100	100
42	100	100	100	100	100	100
43	100	100	100	100	100	100
44	100	100	100	100	100	100
45	100	100	100	100	100	100
46	100	100	100	100	100	100
47	100	100	100	100	100	100
48	100	100	100	100	100	100
49	100	100	100	100	100	100
50	100	100	100	100	100	100
51	100	100	100	100	100	100
52	100	100	100	100	100	100
53	100	100	100	100	100	100
54	100	100	100	100	100	100
55	100	100	100	100	100	100
56	100	100	100	100	100	100
57	100	100	100	100	100	100
58	100	100	100	100	100	100
59						



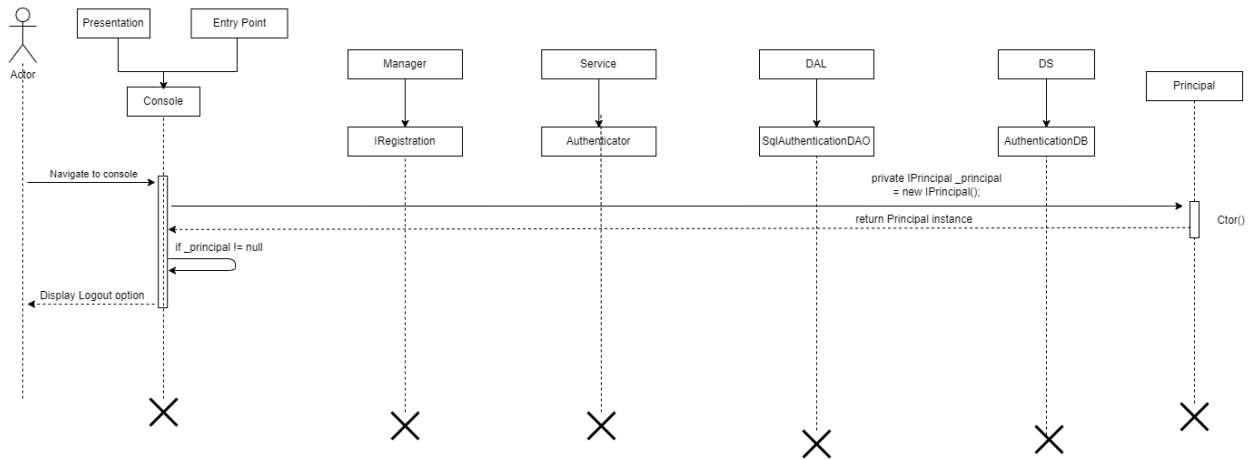
# Authentication

## Authentication Success Case 1 from user authenticated through valid credentials



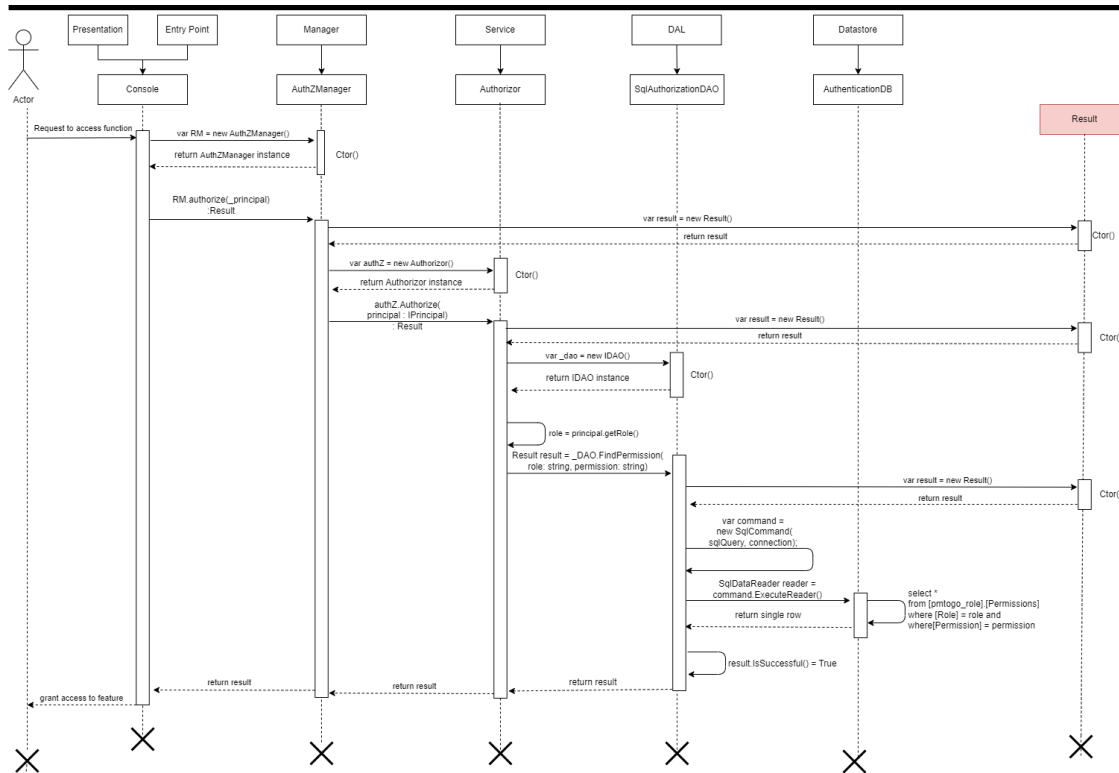
## Authentication Success Case 2 from user is already authenticated

Success Case 2: If user is already authenticated, the user should not be able to reach login view.



# Authorization

## Authorization Success Case 1 - 4



The Authorization Success Case encompasses all 4 cases:

**Success Cases 1:** User attempts to access a protected functionality within authorization scope. Access is granted to perform functionality.

**Success Case 2:** User attempts to access protected data within authorization scope. Access is granted to perform read operations.

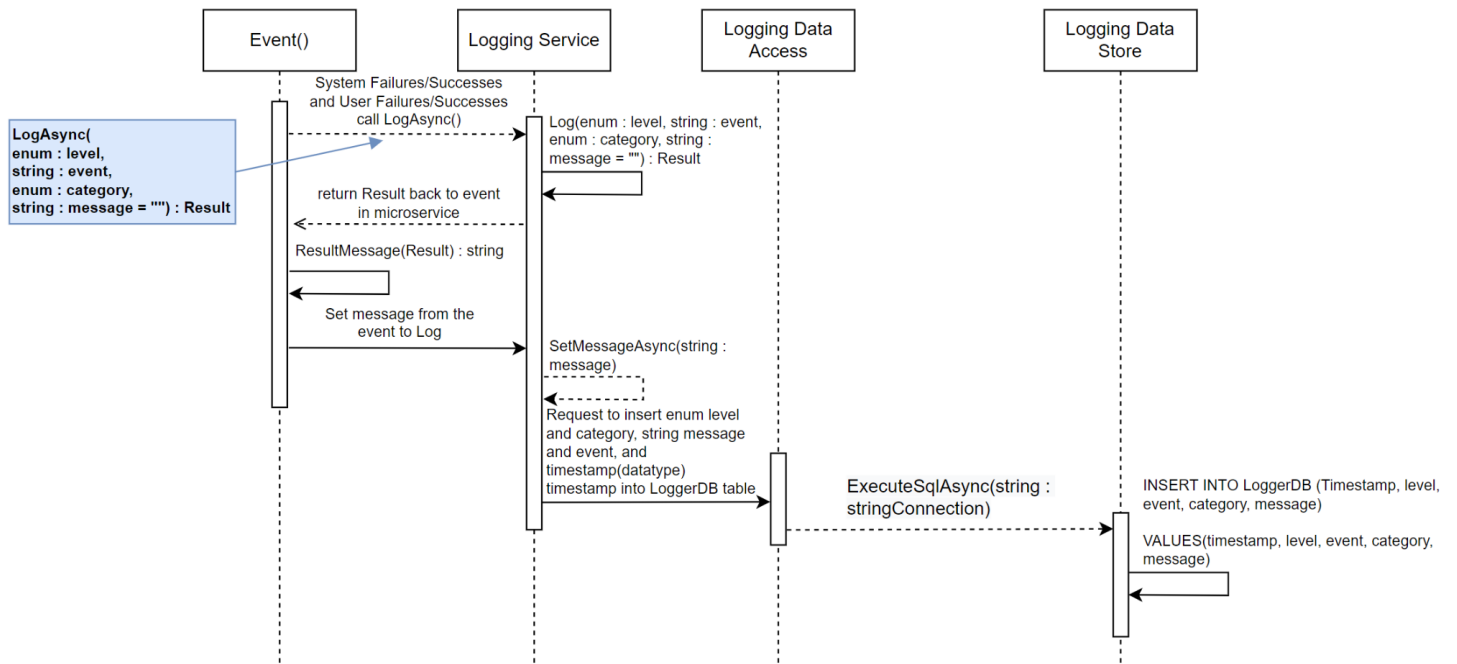
**Success Case 3:** User attempts to modify protected data within authorization scope. Access is granted to perform write operations.

**Success Case 4:** User attempts to access protected views within authorization scope. Access is granted to the view. User is automatically navigated to view.



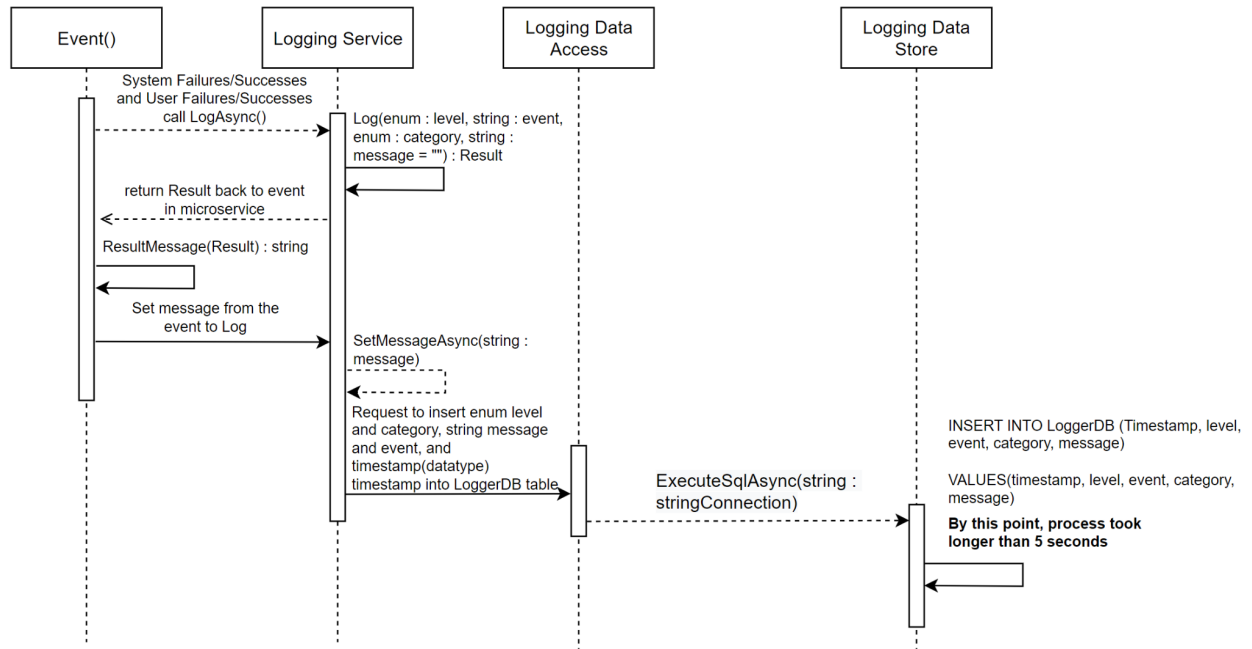
# Logging

## Logging Success Case

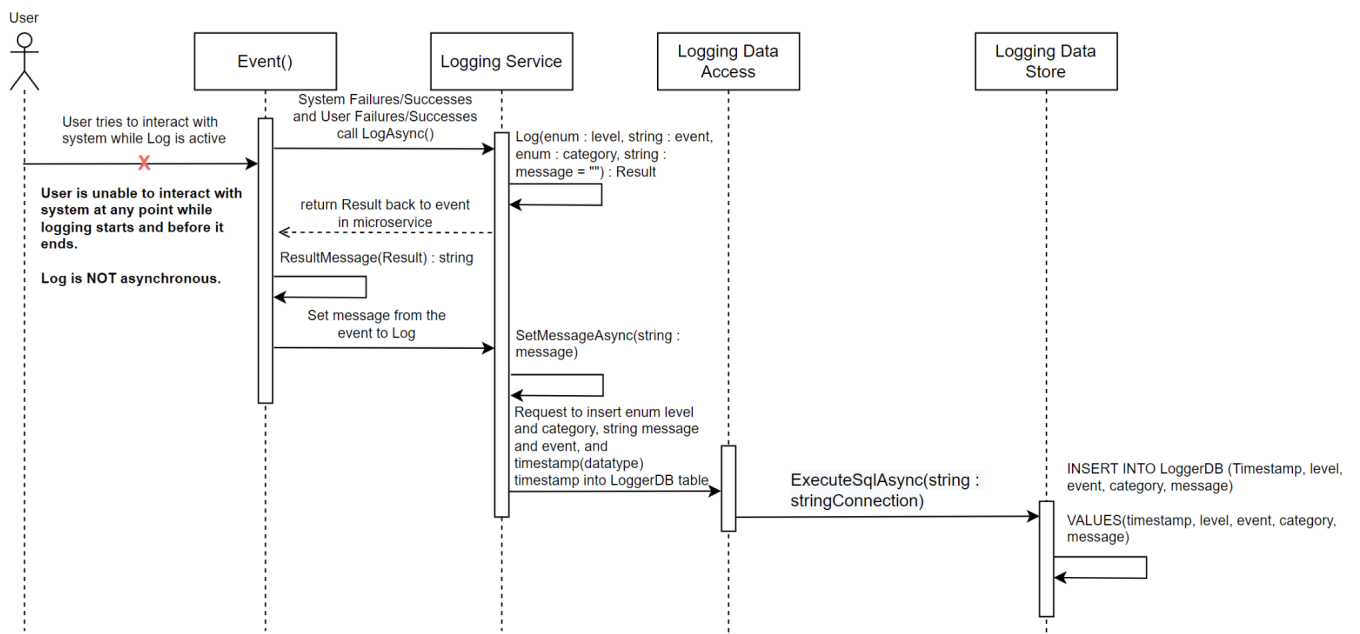


- Event() is the actor and can be considered as any functionality that requires Logging. Functionality can be determined by a user request or system functions and features.
- LoggerDB will be the name of the table AND in a separate database dedicated to logging.
- ExecuteSqlAsync() will be asynchronous because it only contains executing insert statements to tablename LoggerDB.

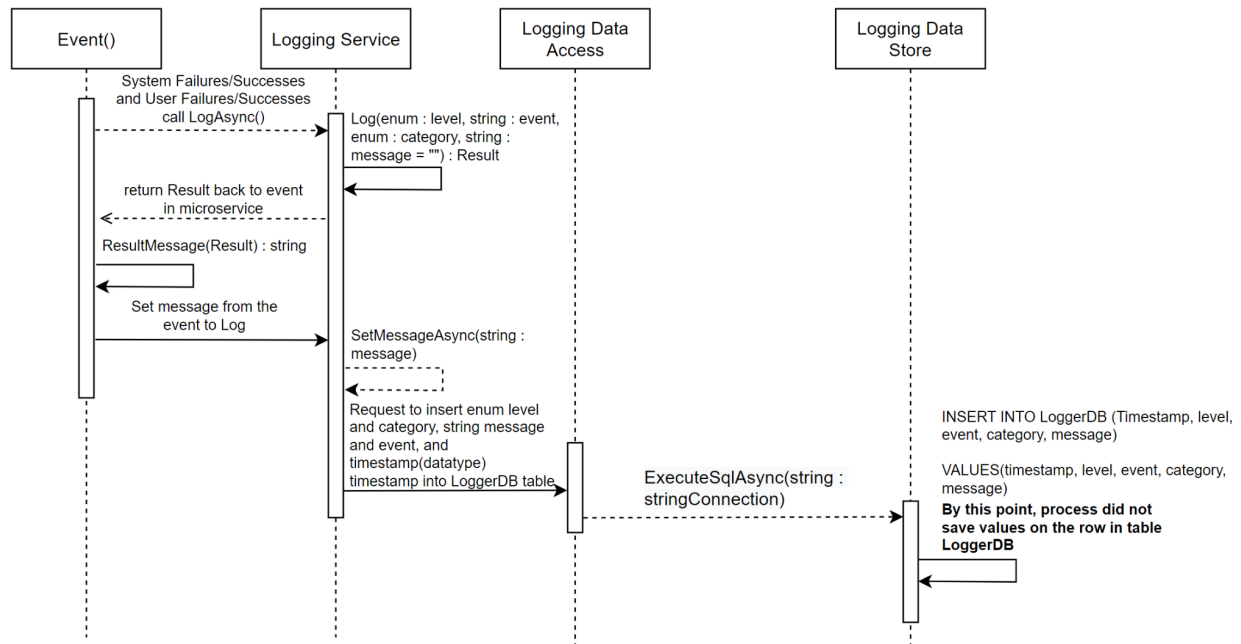
## Logging Failure Case by whole process taking longer than 5 seconds



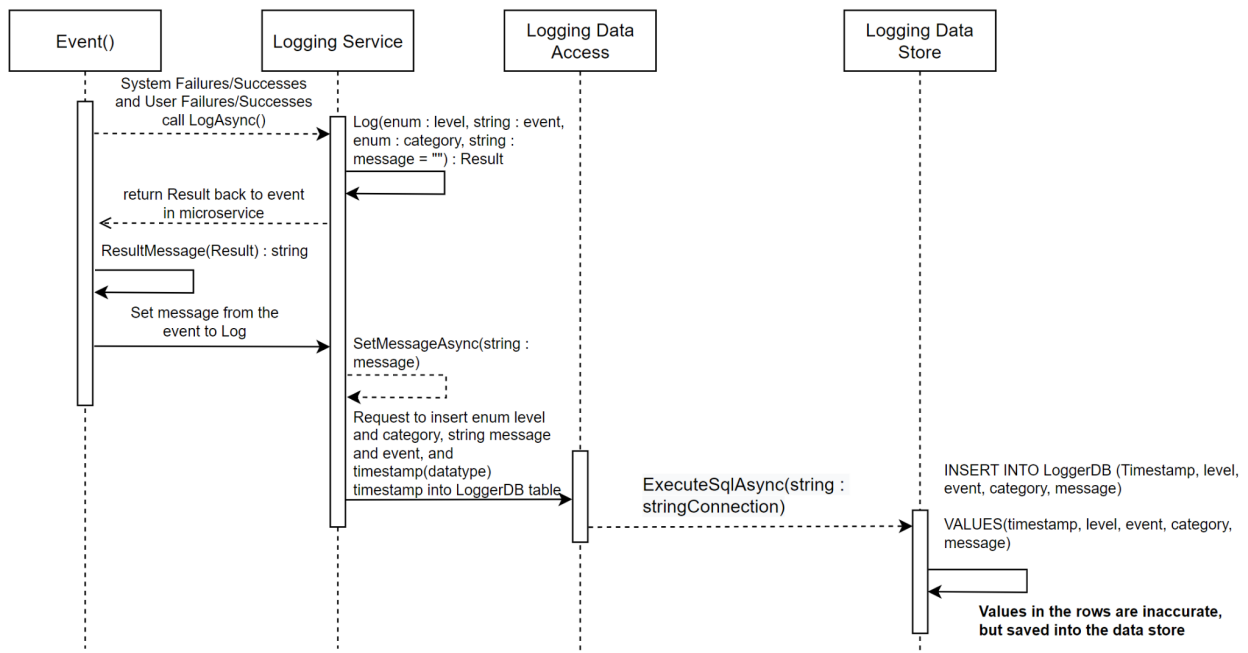
## Logging Failure Case by preventing user from interacting with the system



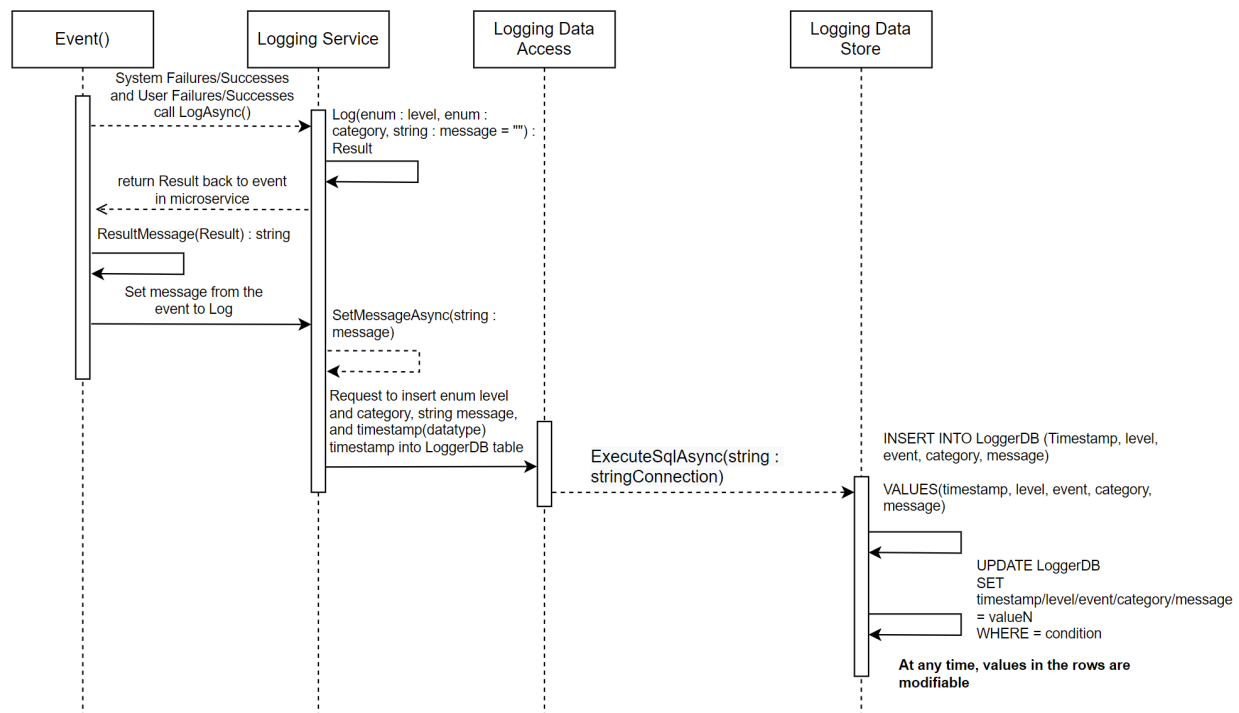
## Logging Failure Case by failed to save log in a persistent data store



## Logging Failure Case by inaccurately saving the event to a persistent data store



## Logging Failure Case by modifiable log entries



- **valueN** is defined as any value that will modify either of the columns shown.
- Ex. SET level = value1, event = value2, category = value3, etc.
- Modifying log entries at any time means it could be outside the sequence diagram as well. It can happen when log is inactive. As long as the persistent storage is on, modifying log entries is **NOT** a success case.

## Result

Result
+ payload : Dictionary<string, object> + isSuccessful : bool + errorMessage : string
+ setIsSuccessful() : void + getIsSuccessful() : bool + setErrorMessage() : void + getErrorMessage() : string

- Result is a class that will be used in every Event() or feature class. The purpose of Result is to confirm with boolean and send an error message if false.
  - Ex. Validation to see if something is true, if not then set and display a corresponding error message.
  - In C#, use the simplified form :  
**var { get; set; }**

# Glossary

Async (Asynchronous method)	Methods that do not have to wait for an answer

# References

- Adegeo, et al. "How to Verify That Strings Are in Valid Email Format." *Microsoft Learn*, 4 Oct. 2022,  
<https://learn.microsoft.com/en-us/dotnet/standard/base-types/how-to-verify-that-strings-are-in-valid-email-format>.
- "Asynchronous Method Call." *Techopedia.com*, 18 Aug. 2011,  
<https://www.techopedia.com/definition/25584/asynchronous-method-call#:~:text=An%20asynchronous%20method%20runs%20in,resources%20resulting%20in%20scalable%20application>
- Corey, Tim. *Logging in .NET Core 3.0 and Beyond - Configuration, Setup, and More. YouTube*, 26 Aug. 2019, <https://youtu.be/oXNslqIXIbQ>. Accessed 1 Nov. 2022
- "How to Send and Receive JSON Data to and from the Server." *Webucator*,  
<https://www.webucator.com/article/how-to-send-and-receive-json-data-to-and-from-the/>.
- "Low Level Design Template." *Government of Nepal Department of Information Technology*,  
<https://doit.gov.np/ckfinder/userfiles/files/GEA/Additional%20Aritfacts/Additional%20Aritfacts%201/Low%20Level%20Design%20Template.pdf>.
- Malek, Piotr. "How to Validate an Email Address in C#." *Mailtrap*, 28 Feb. 2022,  
<https://mailtrap.io/blog/validate-email-address-c/>.
- "SQL - Update Query." *Tutorials Point*,  
<https://www.tutorialspoint.com/sql/sql-update-query.htm>.
- Vatanik Vong, Lecture on Logging, CECS 491A Sec 04, CSULB, October 26, 2022.
- Vatanik Vong, Lecture on Web and UML, CECS 491A Sec 04, CSULB, October 10, 2022.