Version 1.0.0

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/

Version History

Version #	Date	Reason for Change
Version 1.0.0	11/9/2022	Original Document

Table of Contents

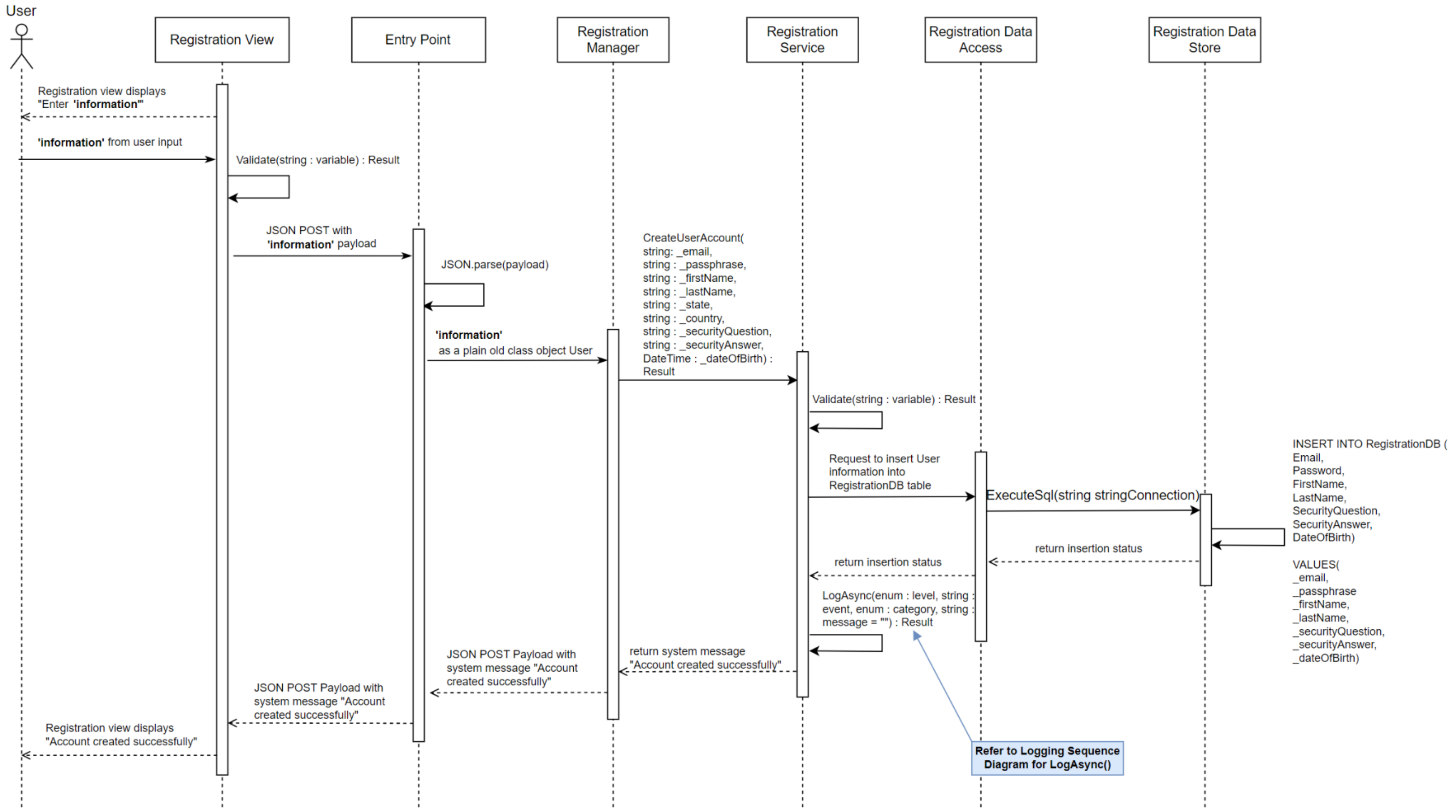
Overview

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

Registration Success Case


User
Registration View
Entry Point
Registration Manager
Registration Service
Registration Data Access
Registration Data Store
Registration view displays "Enter 'information'"
'information' from user input
Validate(string : variable) : Result
JSON POST with 'information' payload
JSON.parse(payload)
'information' as a plain old class object User
CreateUserAccount(string: _email, string : _passphrase, string : _firstName, string : _lastName, string : _state, string : _country, string : _securityQuestion, string : _securityAnswer, DateTime : _dateOfBirth) : Result
Validate(string : variable) : Result
Request to insert User information into RegistrationDB table
ExecuteSql(string stringConnection)
INSERT INTO RegistrationDB (Email, Password, FirstName, LastName, SecurityQuestion, SecurityAnswer, DateOfBirth)
VALUES(_email, _passphrase _firstName, _lastName, _securityQuestion, _securityAnswer, _dateOfBirth)
return insertion status
return insertion status
LogAsync(enum : level, string : event, enum : category, string : message = "") : Result
return system message "Account created successfully"
JSON POST Payload with system message "Account created successfully"
JSON POST Payload with system message "Account created successfully"
Registration view displays "Account created successfully"
Refer to Logging Sequence Diagram for LogAsync()

- “RegistrationDB” is the table name within a database relating to user microservices.
- **‘Information’** is a general term used to prevent clutter of repetitive terms used in the diagrams.
- **‘Information’** contains:

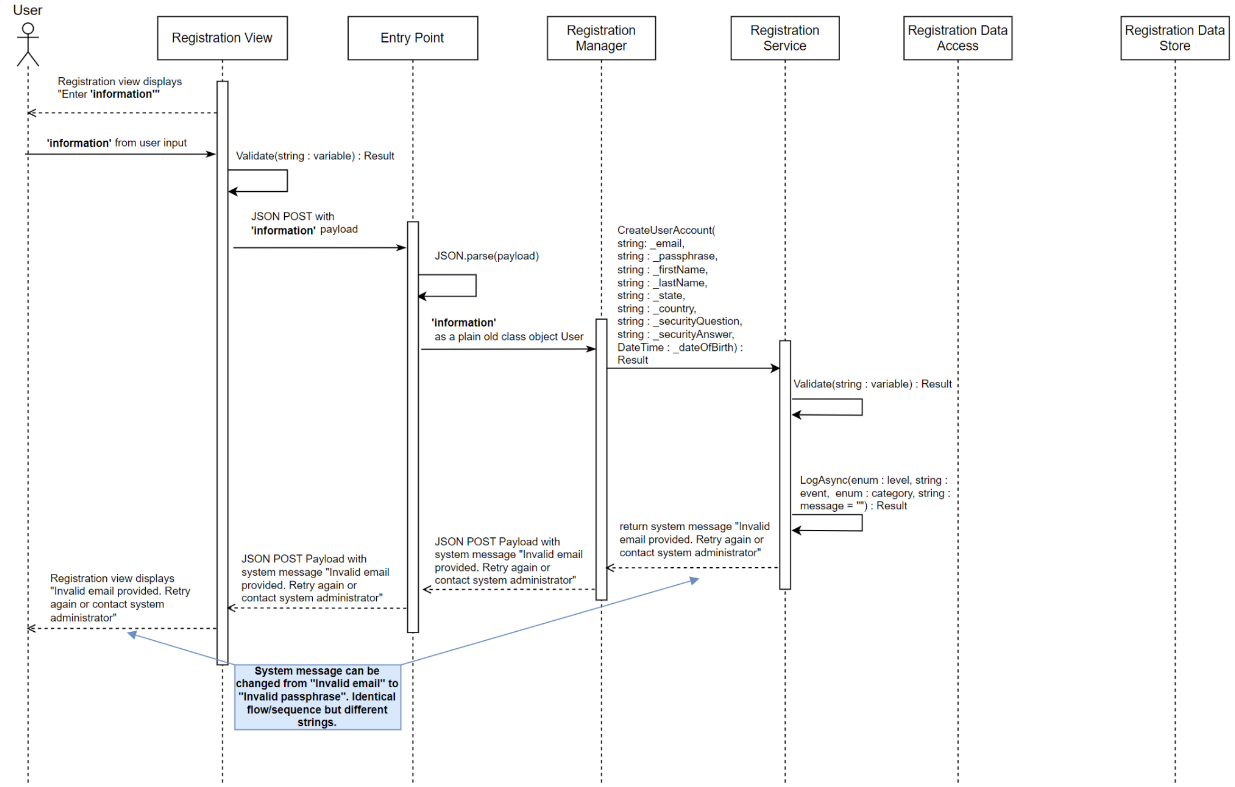
  string : _email
  string : _passphrase
  string : _firstName
  string : _lastName
  string : _state
  string : _country
  string : _securityQuestion
  string : _securityAnswer
  string : _dateOfBirth

- The method **Validate(string : variable) : Result** is a general term for all validation happening in Registration.
- The specific methods of **Validate(string : variable) : Result** include all of the following:

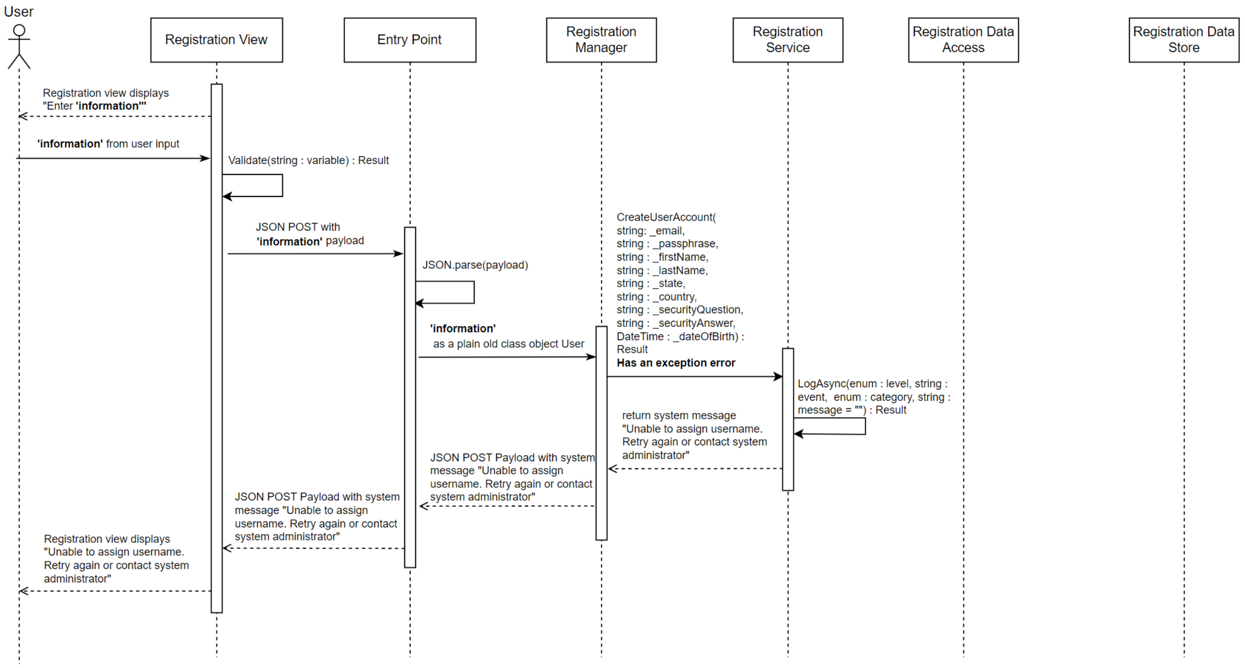
  CheckIfValidEmail(string email) : Result
  CheckPassphrase(string passphrase) : Result
  CheckUserAge(DateTime dateOfBirth) : Result
  CheckUserLocation(string state, string country) : Result
  CheckUserSecurityQA(string securityQuestion, string securityAnswer) : Result

- 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.

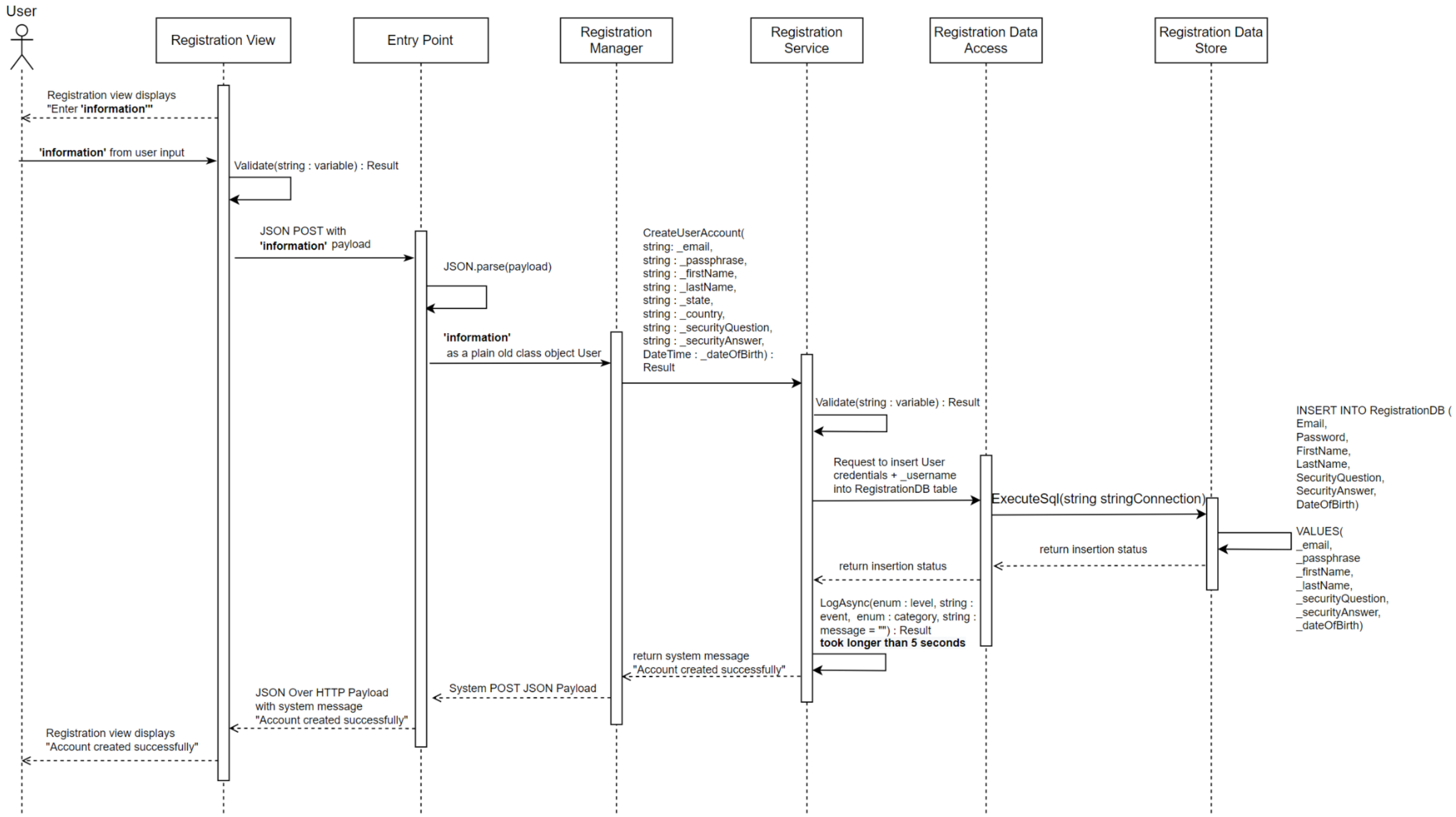
Registration Failure Case from Invalid Email/Passphrase


User
Registration View
Entry Point
Registration Manager
Registration Service
Registration Data Access
Registration Data Store
Registration view displays "Enter 'information'"
'information' from user input
Validate(string : variable) : Result
JSON POST with 'information' payload
JSON.parse(payload)
'information' as a plain old class object User
CreateUserAccount(string: _email, string : _passphrase, string : _firstName, string : _lastName, string : _state, string : _country, string : _securityQuestion, string : _securityAnswer, DateTime : _dateOfBirth) : Result
Validate(string : variable) : Result
LogAsync(enum : level, string : event, enum : category, string : message = "") : Result
return system message "Invalid email provided. Retry again or contact system administrator"
JSON POST Payload with system message "Invalid email provided. Retry again or contact system administrator"
JSON POST Payload with system message "Invalid email provided. Retry again or contact system administrator"
Registration view displays "Invalid email provided. Retry again or contact system administrator"
System message can be changed from "Invalid email" to "Invalid passphrase". Identical flow/sequence but different strings.

Registration Failure Case from Unable to assign username

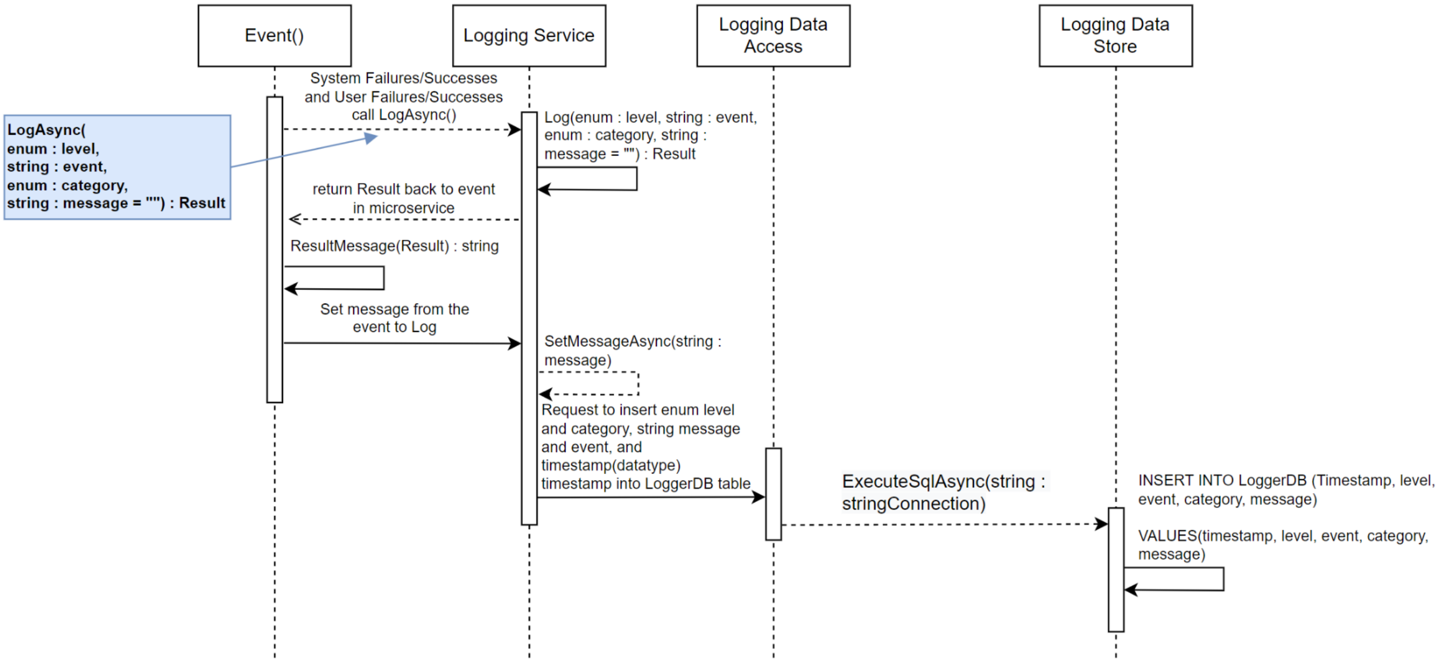

User
Registration View
Entry Point
Registration Manager
Registration Service
Registration Data Access
Registration Data Store
Registration view displays "Enter 'information'"
'information' from user input
Validate(string : variable) : Result
JSON POST with 'information' payload
JSON.parse(payload)
'information' as a plain old class object User
CreateUserAccount(string: _email, string : _passphrase, string : _firstName, string : _lastName, string : _state, string : _country, string : _securityQuestion, string : _securityAnswer, DateTime : _dateOfBirth) : Result
Has an exception error
LogAsync(enum : level, string : event, enum : category, string : message = "") : Result
return system message "Unable to assign username. Retry again or contact system administrator"
JSON POST Payload with system message "Unable to assign username. Retry again or contact system administrator"
JSON POST Payload with system message "Unable to assign username. Retry again or contact system administrator"
Registration view displays "Unable to assign username. Retry again or contact system administrator"

Registration Failure Case from Process took longer than 5 seconds before logging


User
Registration View
Entry Point
Registration Manager
Registration Service
Registration Data Access
Registration Data Store
Registration view displays "Enter 'information'"
'information' from user input
Validate(string : variable) : Result
JSON POST with 'information' payload
JSON.parse(payload)
'information' as a plain old class object User
CreateUserAccount(string: _email, string : _passphrase, string : _firstName, string : _lastName, string : _state, string : _country, string : _securityQuestion, string : _securityAnswer, DateTime : _dateOfBirth) : Result
Validate(string : variable) : Result
Request to insert User credentials + _username into RegistrationDB table
ExecuteSql(string stringConnection)
INSERT INTO RegistrationDB (Email, Password, FirstName, LastName, SecurityQuestion, SecurityAnswer, DateOfBirth)
VALUES(_email, _passphrase _firstName, _lastName, _securityQuestion, _securityAnswer, _dateOfBirth)
return insertion status
return insertion status
LogAsync(enum : level, string : event, enum : category, string : message = "") : Result
took longer than 5 seconds
return system message "Account created successfully"
System POST JSON Payload
JSON Over HTTP Payload with system message "Account created successfully"
Registration view displays "Account created successfully"

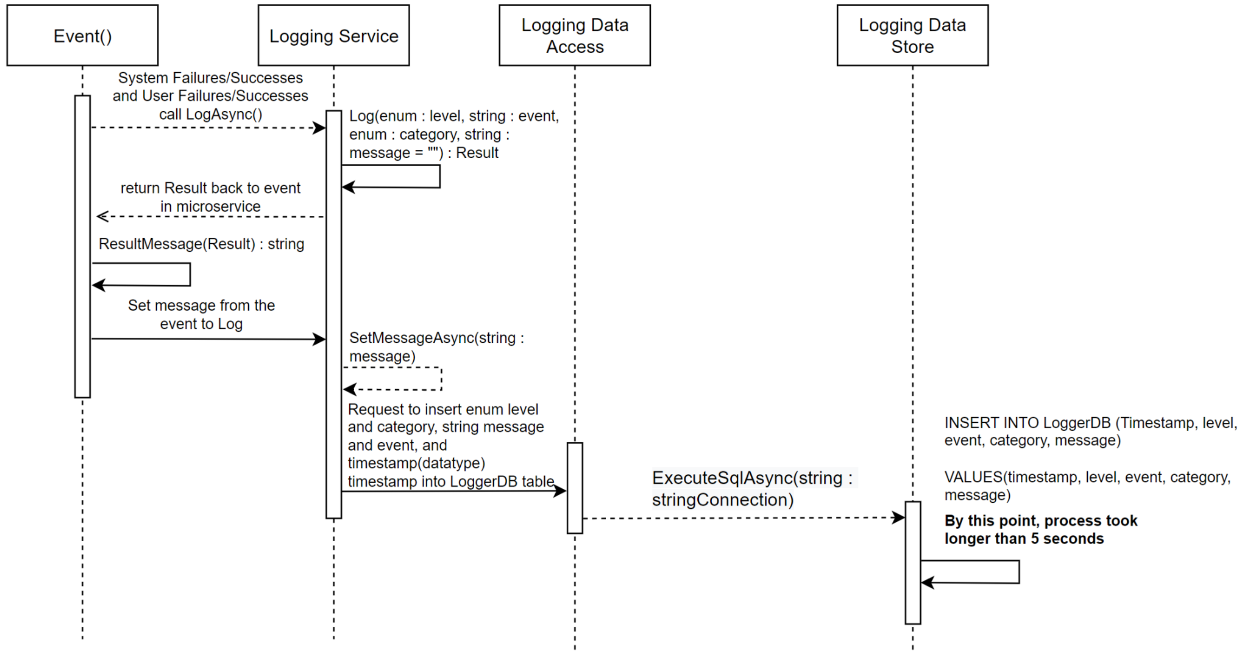
Logging

Logging Success Case

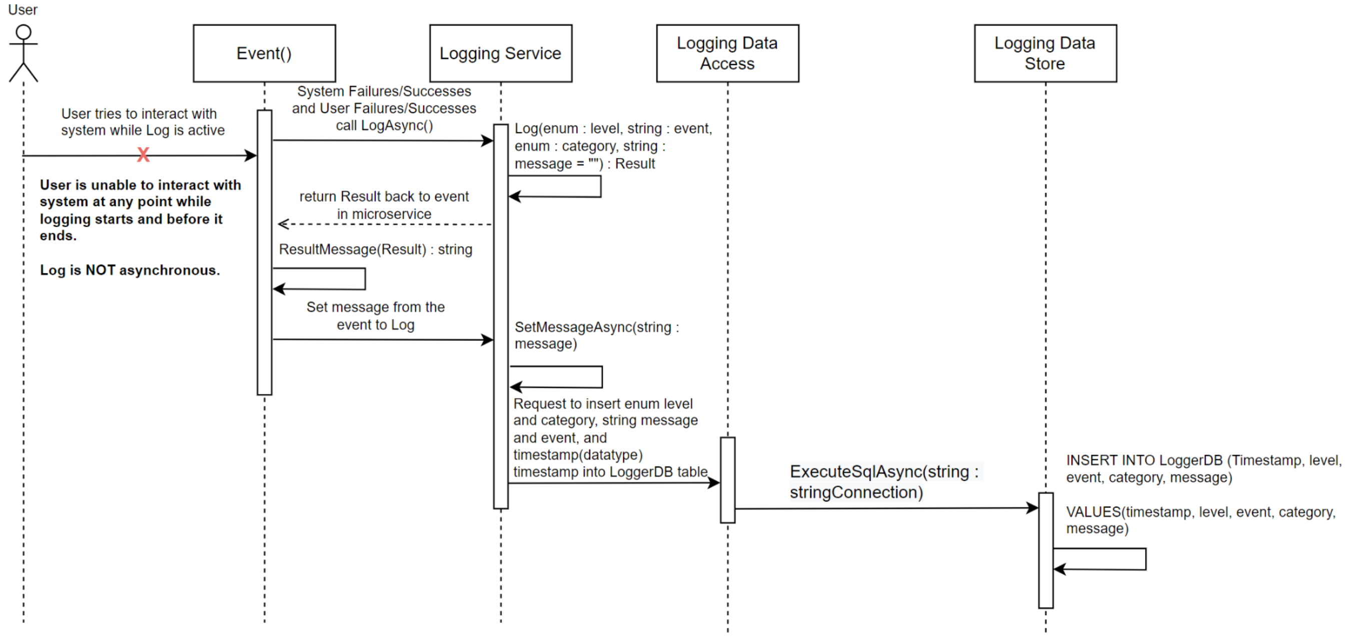

Event()
Logging Service
Logging Data Access
Logging Data Store
LogAsync(
enum : level,
string : event,
enum : category,
string : message = "") : Result
System Failures/Successes and User Failures/Successes call LogAsync()
Log(enum : level, string : event, enum : category, string : message = "") : Result
return Result back to event in microservice
ResultMessage(Result) : string
Set message from the event to Log
SetMessageAsync(string : message)
Request to insert enum level and category, string message and event, and timestamp(datatype) timestamp into LoggerDB table
ExecuteSqlAsync(string : stringConnection)
INSERT INTO LoggerDB (Timestamp, level, event, category, message)
VALUES(timestamp, level, event, category, message)

- Event() is the actor and can be considered as any functionality that requires Logging. Functionality can 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.

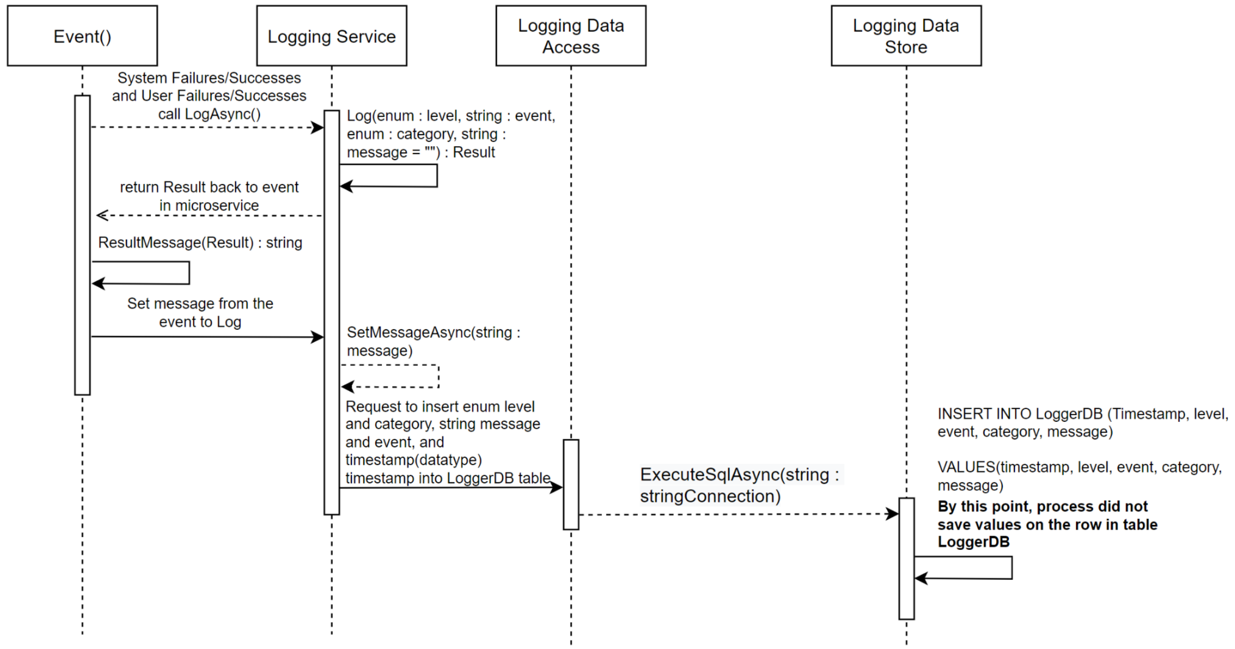
Logging Failure Case by whole process taking longer than 5 seconds


Event()
Logging Service
Logging Data Access
Logging Data Store
System Failures/Successes and User Failures/Successes call LogAsync()
Log(enum : level, string : event, enum : category, string : message = "") : Result
return Result back to event in microservice
ResultMessage(Result) : string
Set message from the event to Log
SetMessageAsync(string : message)
Request to insert enum level and category, string message and event, and timestamp(datatype) timestamp into LoggerDB table
ExecuteSqlAsync(string : stringConnection)
INSERT INTO LoggerDB (Timestamp, level, event, category, message)
VALUES(timestamp, level, event, category, message)
By this point, process took longer than 5 seconds

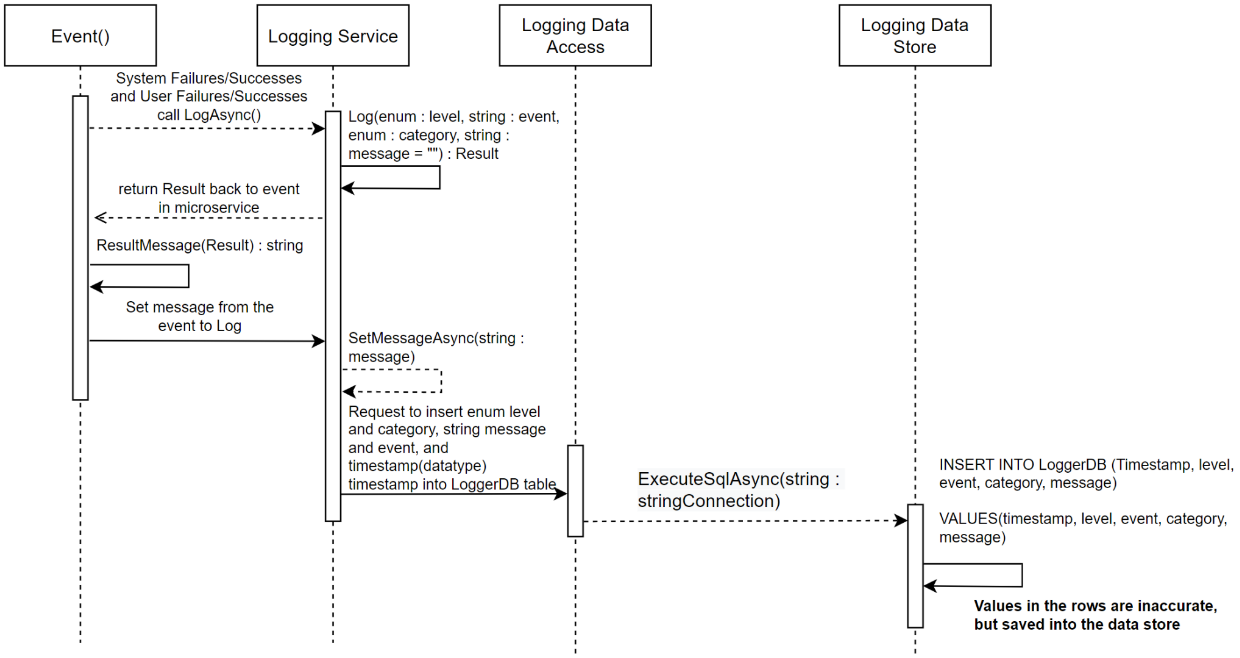
Logging Failure Case by preventing user from interacting with the system


User
Event()
Logging Service
Logging Data Access
Logging Data Store
User tries to interact with system while Log is active
X
User is unable to interact with system at any point while logging starts and before it ends.
Log is NOT asynchronous.
System Failures/Successes and User Failures/Successes call LogAsync()
Log(enum : level, string : event, enum : category, string : message = "") : Result
return Result back to event in microservice
ResultMessage(Result) : string
Set message from the event to Log
SetMessageAsync(string : message)
Request to insert enum level and category, string message and event, and timestamp(datatype) timestamp into LoggerDB table
ExecuteSqlAsync(string : stringConnection)
INSERT INTO LoggerDB (Timestamp, level, event, category, message)
VALUES(timestamp, level, event, category, message)

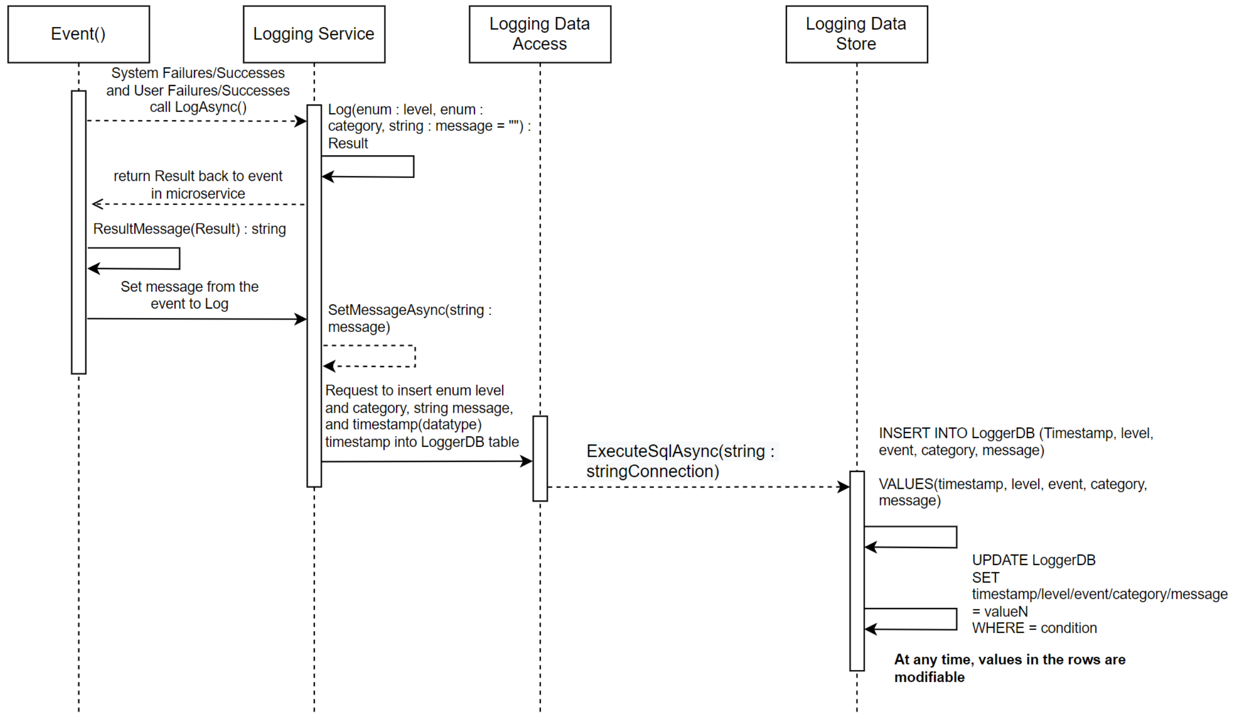
Logging Failure Case by failed to save log in a persistent data store


Event()
Logging Service
Logging Data Access
Logging Data Store
System Failures/Successes and User Failures/Successes call LogAsync()
Log(enum : level, string : event, enum : category, string : message = "") : Result
return Result back to event in microservice
ResultMessage(Result) : string
Set message from the event to Log
SetMessageAsync(string : message)
Request to insert enum level and category, string message and event, and timestamp(datatype) timestamp into LoggerDB table
ExecuteSqlAsync(string : stringConnection)
INSERT INTO LoggerDB (Timestamp, level, event, category, message)
VALUES(timestamp, level, event, category, message)
By this point, process did not save values on the row in table LoggerDB

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

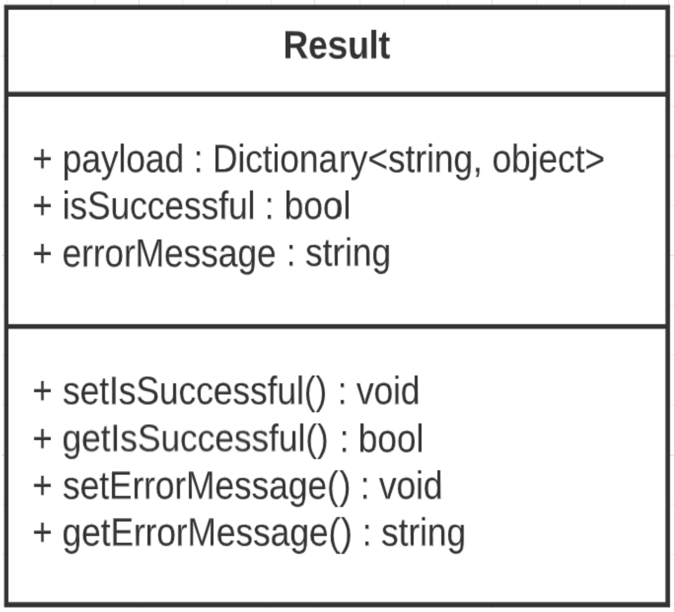

Event()
Logging Service
Logging Data Access
Logging Data Store
System Failures/Successes and User Failures/Successes call LogAsync()
Log(enum : level, string : event, enum : category, string : message = "") : Result
return Result back to event in microservice
ResultMessage(Result) : string
Set message from the event to Log
SetMessageAsync(string : message)
Request to insert enum level and category, string message and event, and timestamp(datatype) timestamp into LoggerDB table
ExecuteSqlAsync(string : stringConnection)
INSERT INTO LoggerDB (Timestamp, level, event, category, message)
VALUES(timestamp, level, event, category, message)
Values in the rows are inaccurate, but saved into the data store

Logging Failure Case by modifiable log entries


Event()
Logging Service
Logging Data Access
Logging Data Store
System Failures/Successes and User Failures/Successes call LogAsync()
Log(enum : level, enum : category, string : message = "") : Result
return Result back to event in microservice
ResultMessage(Result) : string
Set message from the event to Log
SetMessageAsync(string : message)
Request to insert enum level and category, string message, and timestamp(datatype) timestamp into LoggerDB table
ExecuteSqlAsync(string : stringConnection)
INSERT INTO LoggerDB (Timestamp, level, event, category, message)
VALUES(timestamp, level, event, category, message)
UPDATE LoggerDB SET timestamp/level/event/category/message = valueN WHERE = condition
At any time, values in the rows are modifiable

- **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/oXNslgIXIbQ. 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.