Team Big Data

U-tification

Logging Low Level Design

Date: 11/9/2022

Team Leader: Joseph Armas

Team Members: Joshua Gherman

Rhoy Oviedo

Frank Curry

Ghabrille Ampo

David DeGirolamo

Git Repository: https://github.com/JosephArmas/cecs-491A-Team-Big-Data

Version History

Current Version: V2

CHANGES

General

- Changed DataAccess Layer to be a TCP connection only
- Reformated each Diagram
- Added LogError method to Failure diagrams 1,2,and 3
- Modified diagram summaries to include:
    - More information at each layer
    - Return types mentioned
    - Creation of log methods
    - Success/Failure scenario for DataStore response
- Logging Non-Core Scenarios was added
    - Authentication at UI
    - Authentiaction without Log In
    - Multiple Log Writes

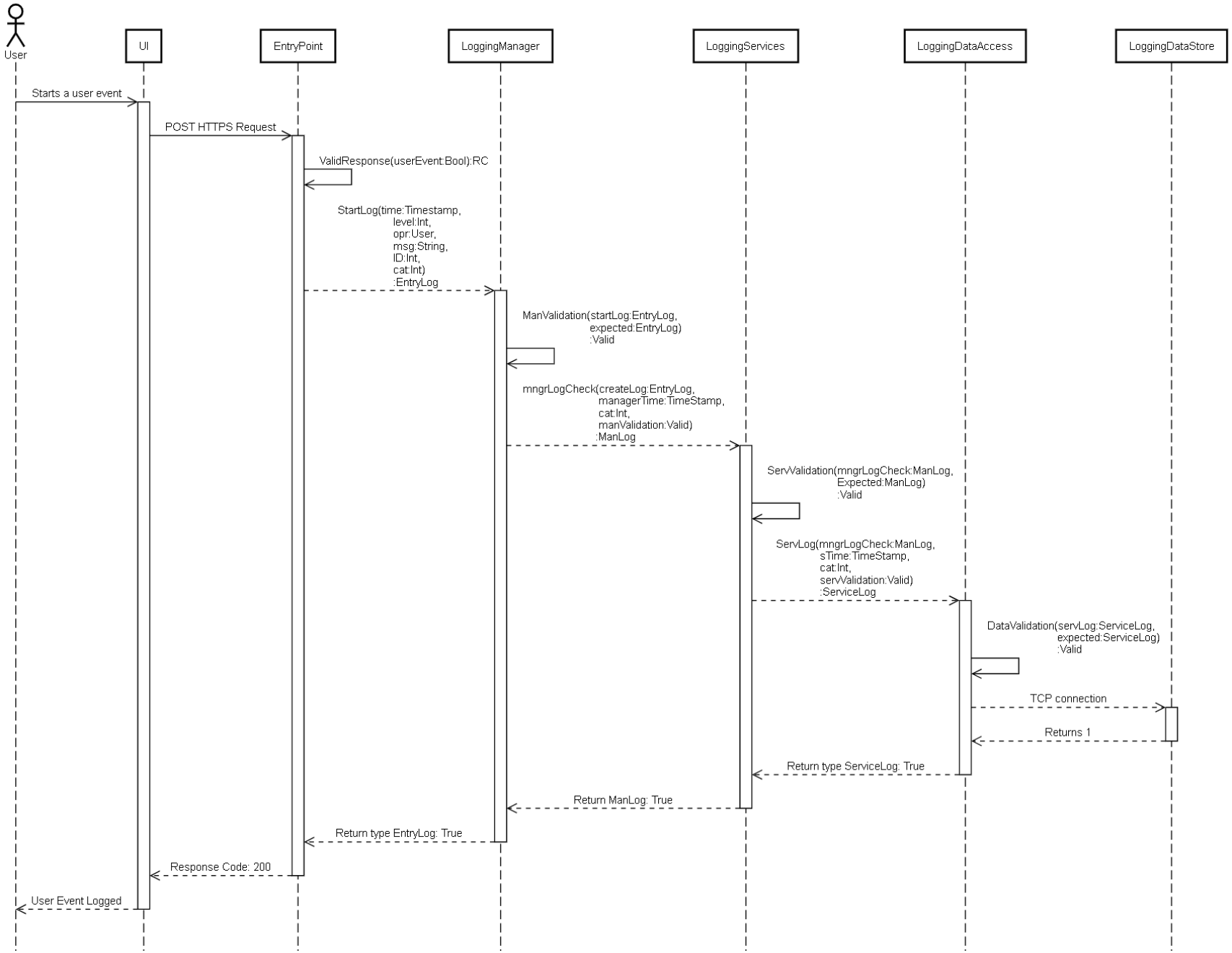
Previous Versions: V1, V2

Table of Contents

Logging Success:Successfully Logging User Success

- Objective: Show a successful user scenario of logging.
- UI: Sends a HTTPS POST AJAX request that is validated by a Response Code
    - Return: a string notification error.
- EntryPoint: The EntryPoint would have validated a request as RC 200. The Logging Process begins.
    - Return: a Response Code is expected to be returned.
- LoggingManager: Validates the log that it is passing along information. LoggingManager is made in Logger.cs file.
    - ManValidation is created in our LoggerTest.cs file to test if the log has information
    - Return: ManLog object that returns a boolean True
- LoggingServices: Log method is officially called to start sending information to the backend. LoggingServices is made in the Logger.cs file.
    - ServValidation is created in our LoggerTest.cs file to test the valid system or user information.
    - Return: ServiceLog object that returns a boolean value True
- LoggingDataAccess: Log is sent over TCP connection to reach the DataStore. LoggingDataAccess is made in the SqlDAO.cs file
    - DataValidation is created in DataAccessTest to validate the log before finally reaching the DataStore.
    - Return: LogResult that returns a boolean True
- LoggingDataStore: Data Store Returns scalar value (0 - unknown, 1- true, 2- false) of 1 as there is no error with logging to the DataStore. The User interaction is logged in the DataStore
- Valid is a custom object that returns a boolean value

As a User, I want to start a user event, so that the system may log my actions.

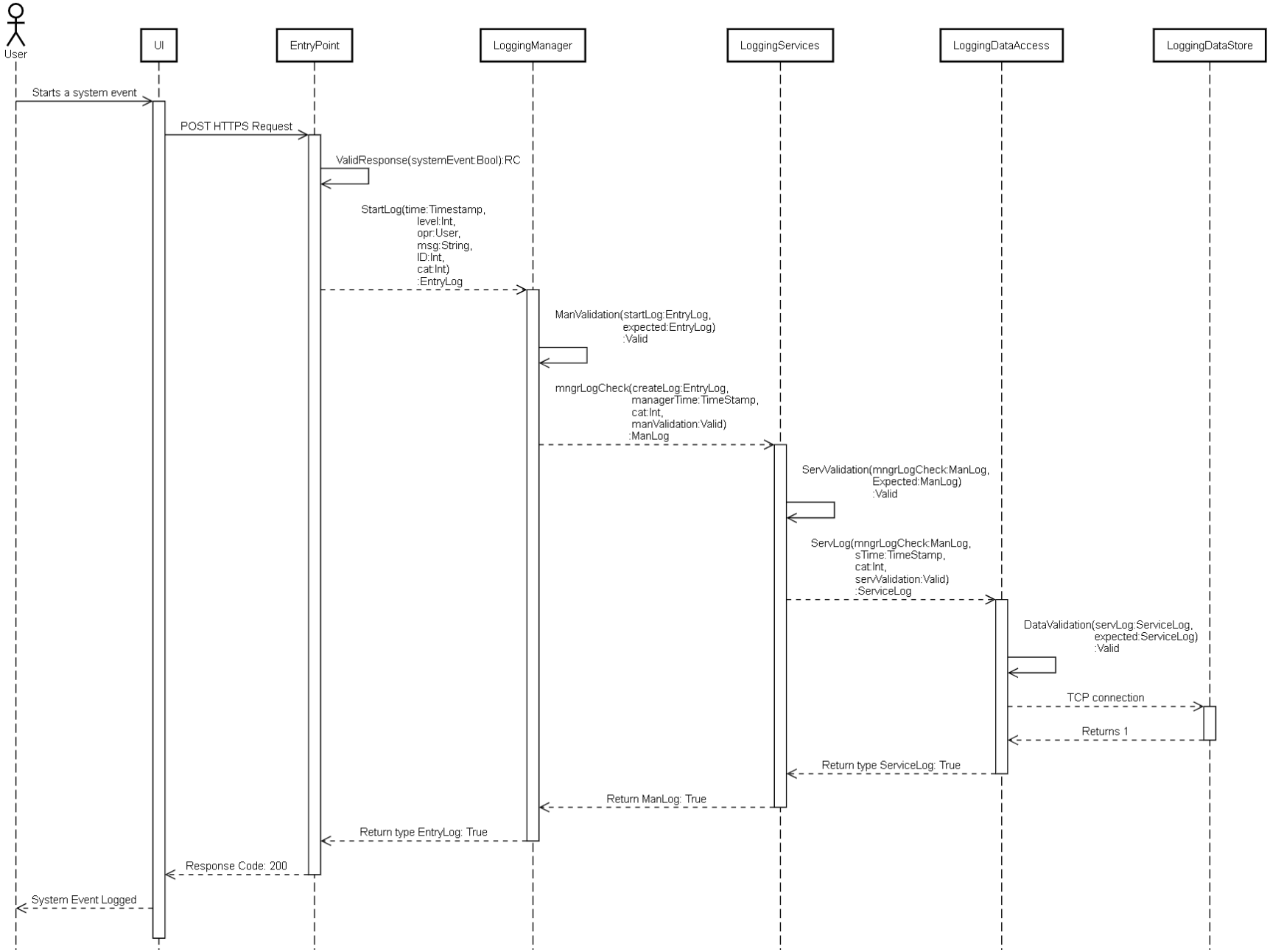


Successfully Logging System Success

The diagram below shows a successful system event that is being logged. The map of the system is updating automatically over a certain time interval.

- Objective: Show a successful system scenario of logging.
- UI: Sends a HTTPS POST AJAX request that is validated by a Response Code
    - Return: a string notification error.
- EntryPoint: The EntryPoint would have validated a request as RC 200. The Logging Process begins.
    - Return: a Response Code is expected to be returned.
- LoggingManager: Validates the log that it is passing along information. LoggingManager is made in Logger.cs file.
    - ManValidation is created in our LoggerTest.cs file to test if the log has information
    - Return: ManLog object that returns a boolean True
- LoggingServices: Log method is officially called to start sending information to the backend. LoggingServices is made in the Logger.cs file.
    - ServValidation is created in our LoggerTest.cs file to test the valid system or user information.
    - Return: ServiceLog object that returns a boolean value True
- LoggingDataAccess: Log is sent over TCP connection to reach the DataStore. LoggingDataAccess is made in the SqlDAO.cs file
    - DataValidation is created in DataAccessTest to validate the log before finally reaching the DataStore.
    - Return: LogResult that returns a boolean True
- LoggingDataStore: Data Store Returns scalar value (0 - unknown, 1- true, 2- false) of 1 as there is no error with logging to the DataStore. The System process is logged in the DataStore.
- Valid is a custom object that returns a boolean value

As a System, I want to log successful events, so that the system may log my actions.

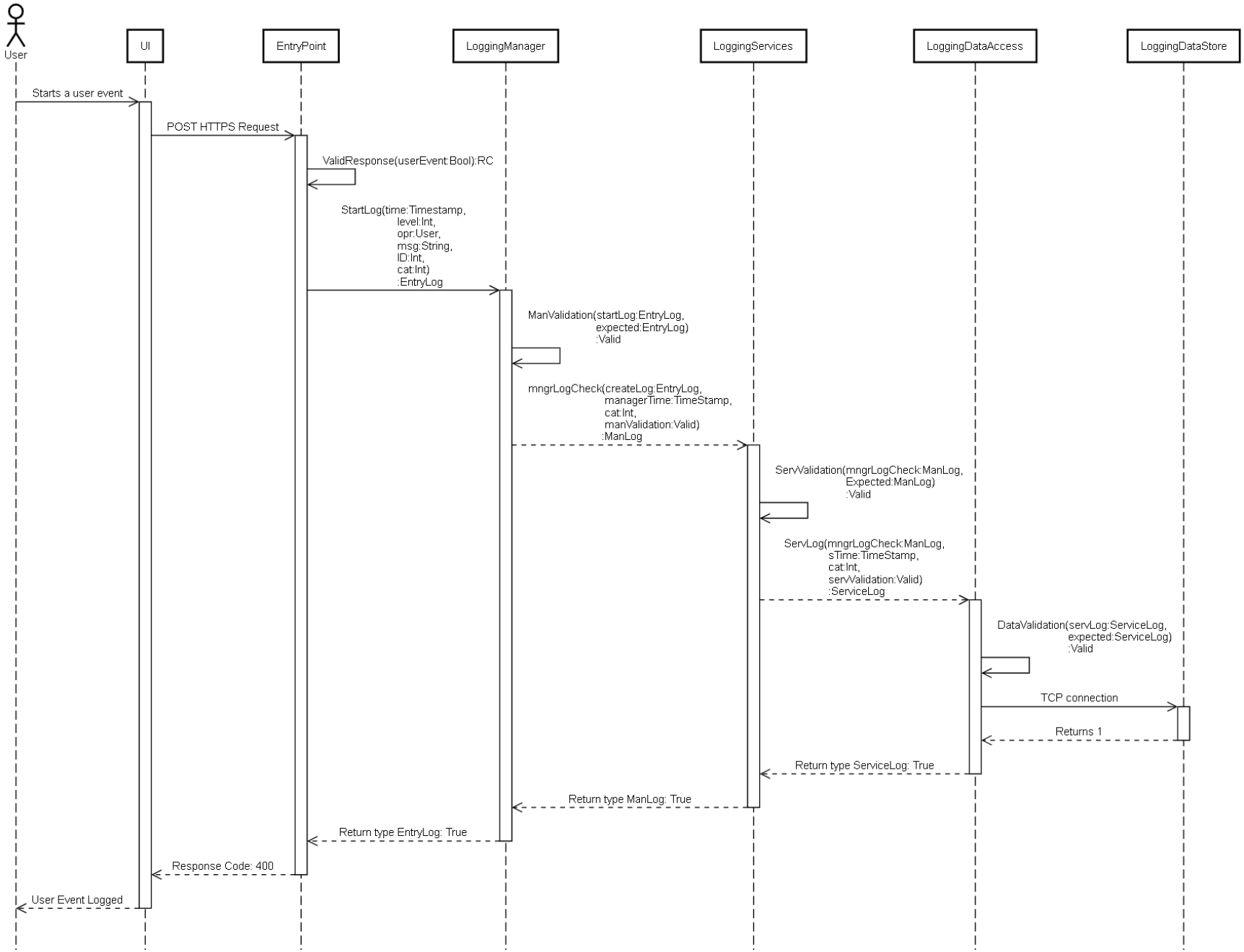


Successfully Logging User Failure

The diagram represents a user event failing and being logged. The example shown is of a pin attempting to be created but failing.

- Objective: Show an unsuccessful user event and log the event.
- UI: Sends a HTTPS POST AJAX request that is validated by a Response Code
    - Return: a string notification error.
- EntryPoint: The EntryPoint would have user invalidated a request as RC 400 The Logging Process begins.
    - Return: a Response Code is expected to be returned.
- LoggingManager: Validates the log that it is passing along information. LoggingManager is made in Logger.cs file.
    - ManValidation is created in our LoggerTest.cs file to test if the log has information
    - Return: ManLog object that returns a boolean True
- LoggingServices: Log method is officially called to start sending information to the backend. LoggingServices is made in the Logger.cs file.
    - ServValidation is created in our LoggerTest.cs file to test the valid system or user information.
    - Return: ServiceLog object that returns a boolean value True
- LoggingDataAccess: Log is sent over TCP connection to reach the DataStore. LoggingDataAccess is made in the SqlDAO.cs file
    - DataValidation is created in DataAccessTest to validate the log before finally reaching the DataStore.
    - Return: LogResult that returns a boolean True
- LoggingDataStore: Data Store Returns scalar value (0 - unknown, 1- true, 2- false) of 1 as there is no error with logging to the DataStore. The unsuccessful user event is logged in the DataStore.
- Valid is a custom object that returns a boolean value

As a User, I want my Unsuccessful user events logged, so that the system may record error.

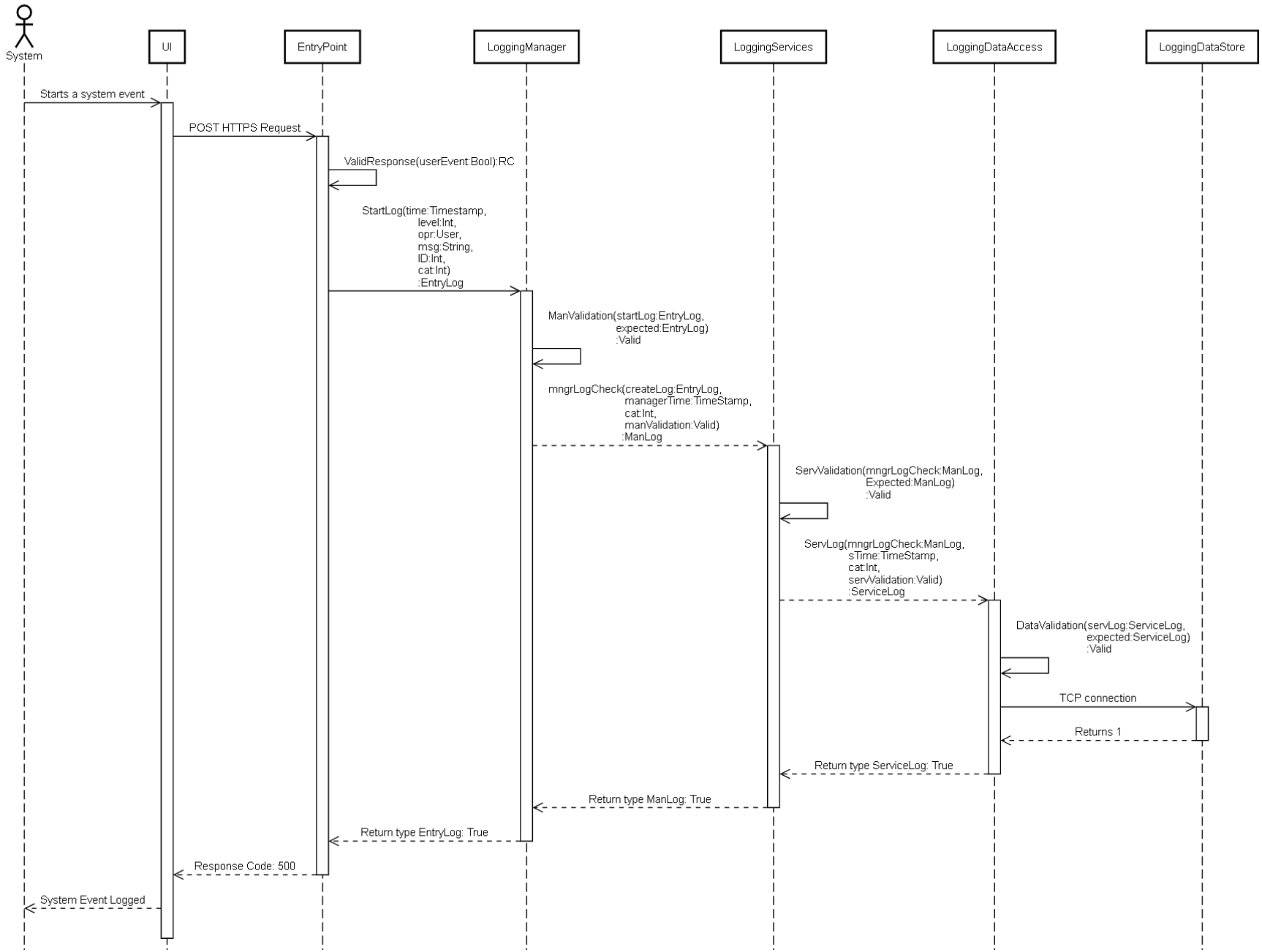


Successfully Logging System Failure

The system failure in this diagram represents a failure in the UI layer as an example. The failure accounts for an automatic update to the map but it fails to load. This is then logged to the database.

- Objective: Show an unsuccessful system event and log the event.
- UI: Sends a HTTPS POST AJAX request that is validated by a Response Code
    - Return: a string notification error.
- EntryPoint: The EntryPoint would have user invalidated a request as RC 500 The Logging Process begins.
    - Return: a Response Code is expected to be returned.
- LoggingManager: Validates the log that it is passing along information. LoggingManager is made in Logger.cs file.
    - ManValidation is created in our LoggerTest.cs file to test if the log has information
    - Return: ManLog object that returns a boolean True
- LoggingServices: Log method is officially called to start sending information to the backend. LoggingServices is made in the Logger.cs file.
    - ServValidation is created in our LoggerTest.cs file to test the valid system or user information.
    - Return: ServiceLog object that returns a boolean value True
- LoggingDataAccess: Log is sent over TCP connection to reach the DataStore. LoggingDataAccess is made in the SqlDAO.cs file
    - DataValidation is created in DataAccessTest to validate the log before finally reaching the DataStore.
    - Return: LogResult that returns a boolean True
- LoggingDataStore: Data Store Returns scalar value (0 - unknown, 1- true, 2- false) of 1 as there is no error with logging to the DataStore. The unsuccessful system event is logged in the DataStore.
- Valid is a custom object that returns a boolean value

As a System, I want to log unsuccessful events, so that the system may log my actions.



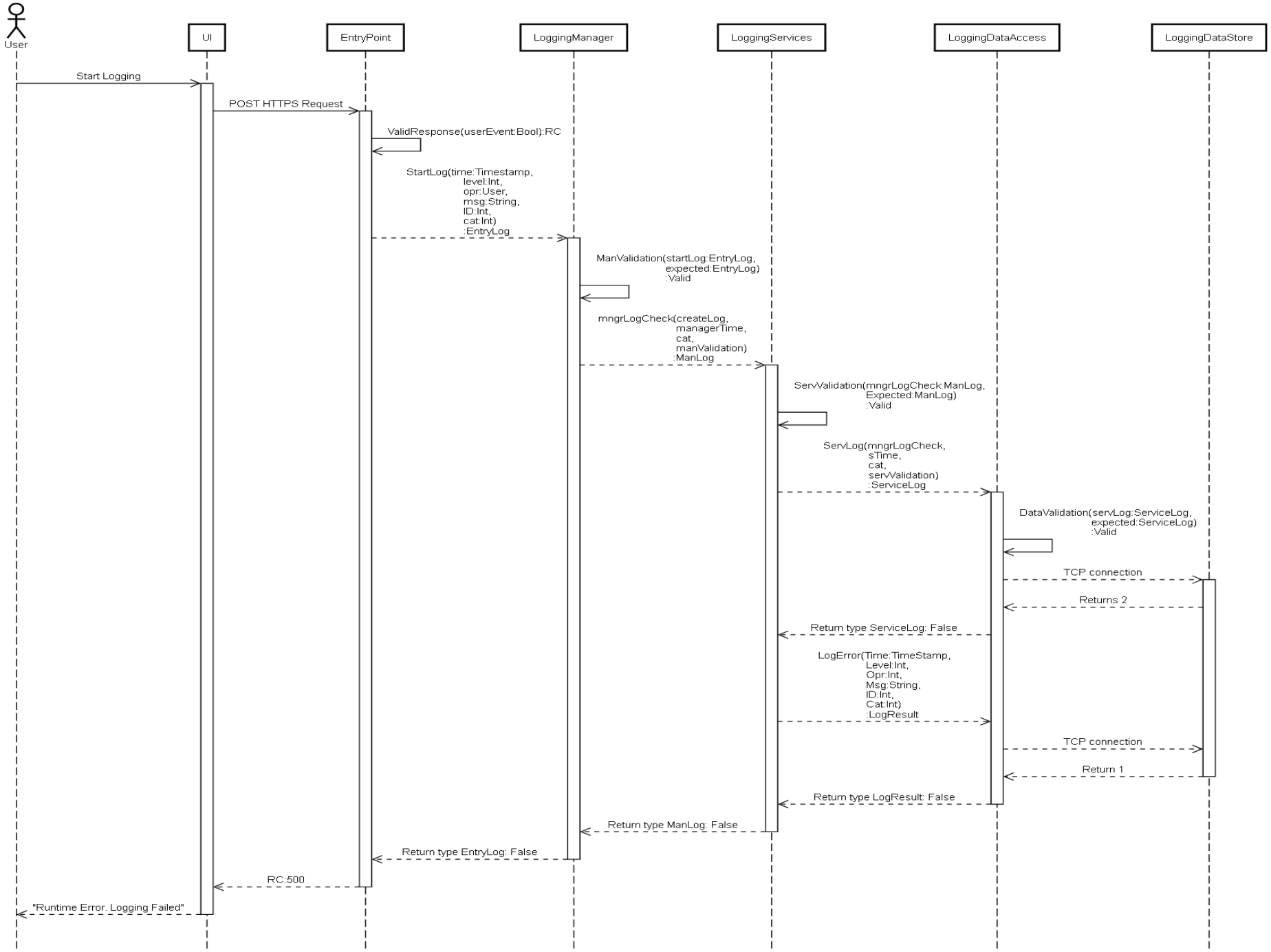
Logging Failure

System Has A Log Interaction Take Longer Than 5 Seconds

The Response Code (RC) is returned as a System Error, as User interaction was not the cause of failure.

- Objective: Catch an instance of a log taking longer than 5 seconds to complete upon invocation.
- UI: Sends a HTTPS POST AJAX request that is validated by a Response Code
    - Return: a string notification error.
- EntryPoint: The EntryPoint would have validated a request as RC 200. The Logging Process begins.
    - Return: a Response Code is expected to be returned.
- LoggingManager: Validates the log that it is passing along information. LoggingManager is made in Logger.cs file.
    - ManValidation is created in our LoggerTest.cs file to test if the log has information
    - Return: ManLog object that returns a boolean false
- LoggingServices: Log method is officially called to start sending information to the backend. LoggingServices is made in the Logger.cs file.
    - ServValidation is created in our LoggerTest.cs file to test the valid system or user information.
    - Return: ServiceLog object that returns a boolean value False
- LoggingDataAcces: Log is sent over TCP connection to reach the DataStore. LoggingDataAccess is made in the SqlDAO.cs file
    - DataValidation is created in DataAccessTest to validate the log before finally reaching the DataStore.
    - Return: LogResult that returns a boolean True
- LoggingDataStore: Data Store Returns scalar value (0 - unknown, 1- true, 2- false) of 2 as there is an error with logging to the DataStore. The Log took longer than 5 seconds. The log failed to be stored.
- LoggingServices request a new log as LogError to record the earlier invalid logging request.
- Valid is a custom object that returns a boolean value

As a System, I want to Log, but the process took longer than 5 seconds

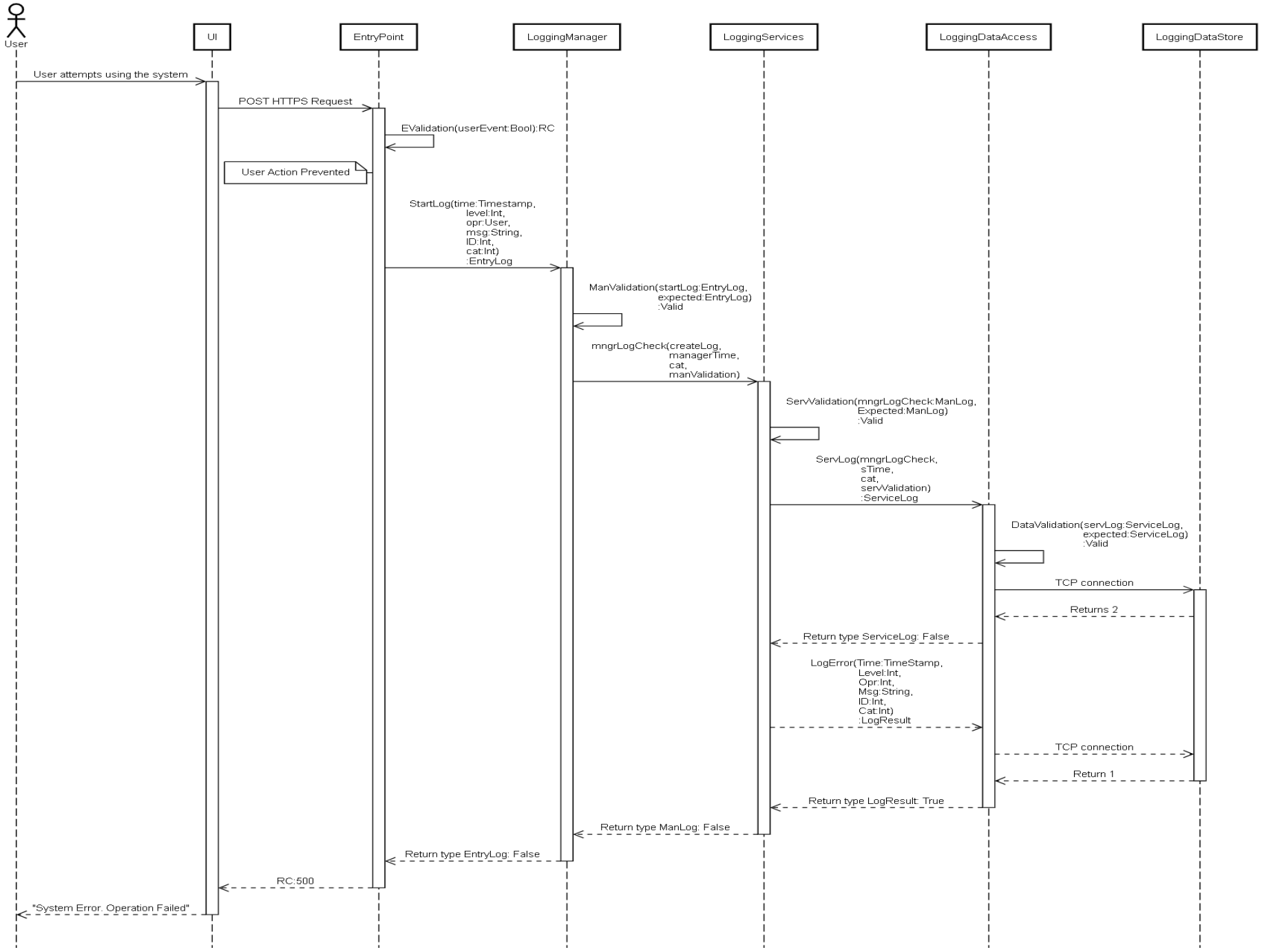


System Logging Does Not Allow User Interaction

Upon Log Creation, the system does not persist and fails to record User Interaction.

- Objective: Catch an instance of logging interfering with User interaction.
- UI: Sends a HTTPS POST AJAX request that is validated by a Response Code
    - Return: a string notification error.
- EntryPoint: The EntryPoint would have an invalidated a request as RC 500. The Logging Process begins.
    - Return: a Response Code is expected to be returned.
- User Interaction is blocked so the system fails to validate the logging process that caused the error.
- LoggingDataStore: Data Store Returns scalar value (0 - unknown, 1- true, 2- false) of 2 as there is an error with logging to the DataStore. The Log was detrimental to the system. The log failed to be stored.
- LoggingServices request a new log as LogError to record the earlier invalid logging request.
- Valid is a custom object that returns a boolean value

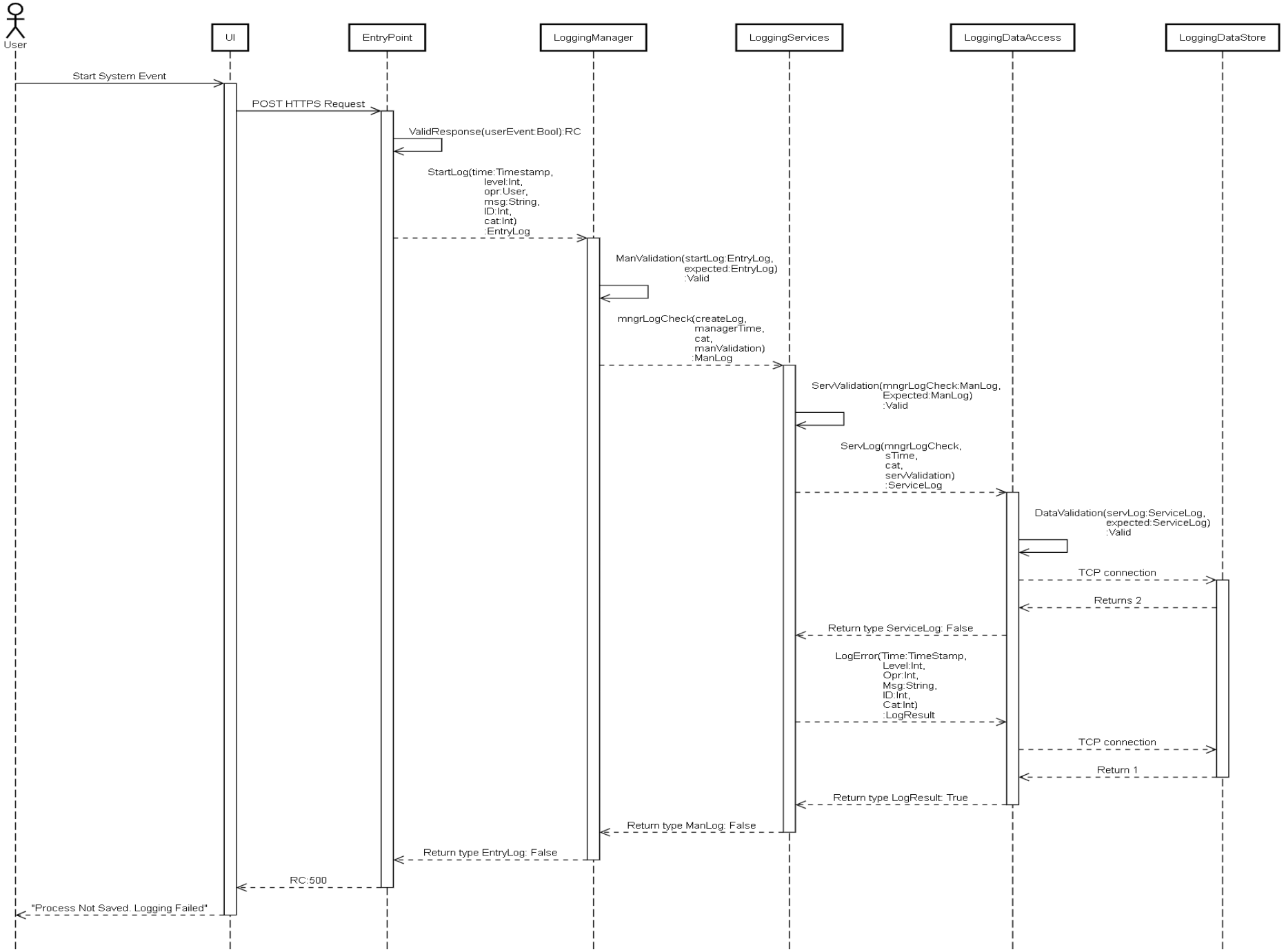
As a User, I want to interact, but I cannot interact with the system.



System Does Not Save Logs In A Persistent Storage

- Objective: Catch an instance of a log not being stored in the DataStore. Log the error.
- UI: Sends a HTTPS POST AJAX request that is validated by a Response Code
    - Return: a string notification error.
- EntryPoint: The EntryPoint would have validated a request as RC 200. The Logging Process begins.
    - Return: a Response Code is expected to be returned.
- LoggingManager: Validates the log that it is passing along information. LoggingManager is made in Logger.cs file.
    - ManValidation is created in our LoggerTest.cs file to test if the log has information
    - Return: ManLog object that returns a boolean false
- LoggingServices: Log method is officially called to start sending information to the backend. LoggingServices is made in the Logger.cs file.
    - ServValidation is created in our LoggerTest.cs file to test the valid system or user information.
    - Return: ServiceLog object that returns a boolean value False
- LoggingDataAcces: Log is sent over TCP connection to reach the DataStore. LoggingDataAccess is made in the SqlDAO.cs file
    - DataValidation is created in DataAccessTest to validate the log before finally reaching the DataStore.
    - Return: LogResult that returns a boolean True
- LoggingDataStore: Data Store Returns scalar value (0 - unknown, 1- true, 2- false) of 2 as there is an error with logging to the DataStore. The Log did not match the expected value. The log failed to be stored.
- LoggingServices request a new log as LogError to record the earlier invalid logging request.
- Valid is a custom object that returns a boolean value

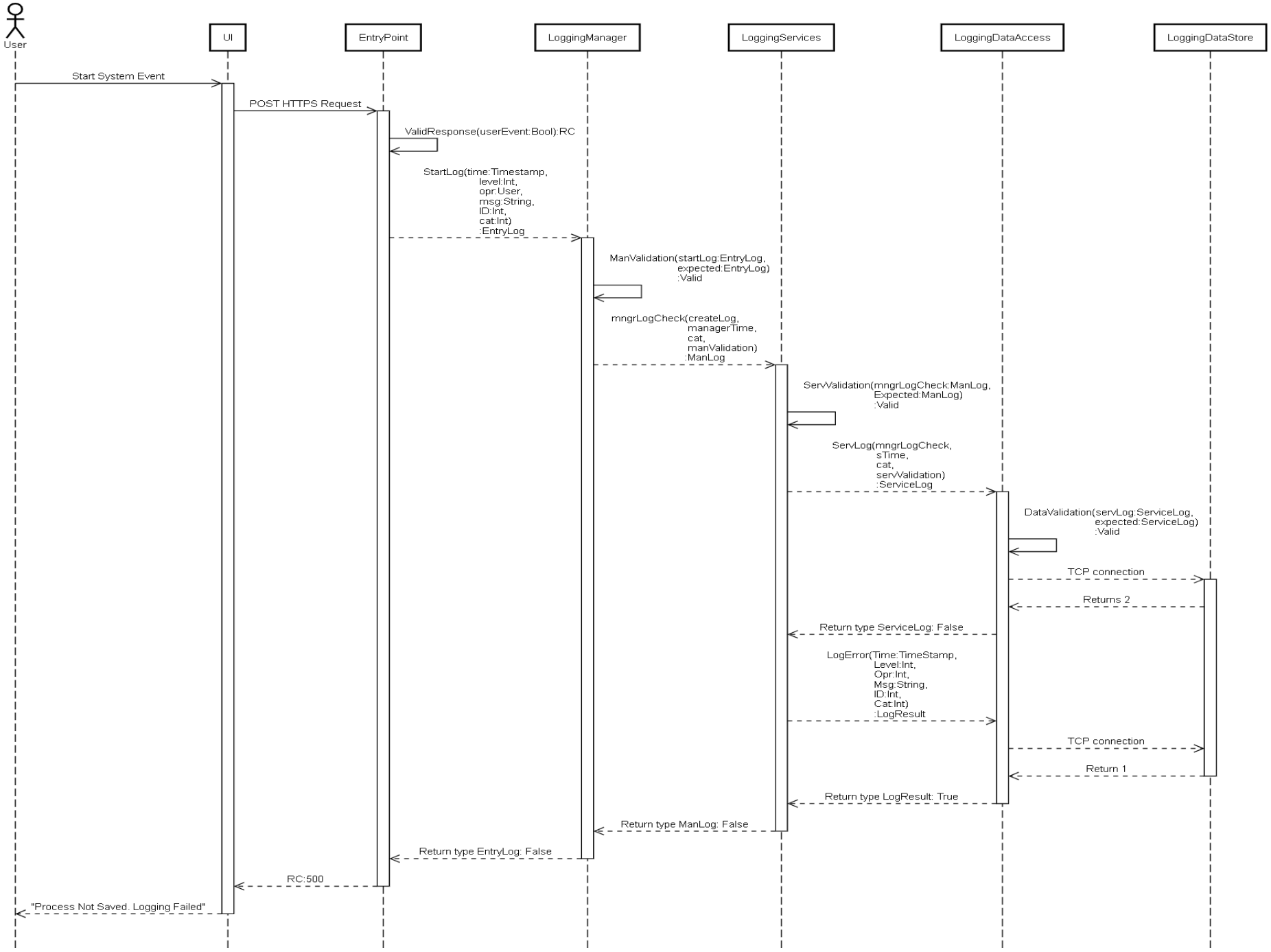
As a System, I want to Log, but the process did not save to the data store.



System Does Not Save All of Logs Data In A Persistent Storage

- Objective: Catch an instance of a log only being partially stored in the DataStore. Log the error.
- UI: Sends a HTTPS POST AJAX request that is validated by a Response Code
    - Return: a string notification error.
- EntryPoint: The EntryPoint would have validated a request as RC 200. The Logging Process begins.
    - Return: a Response Code is expected to be returned.
- LoggingManager: Validates the log that it is passing along information. LoggingManager is made in Logger.cs file.
    - ManValidation is created in our LoggerTest.cs file to test if the log has information
    - Return: ManLog object that returns a boolean false
- LoggingServices: Log method is officially called to start sending information to the backend. LoggingServices is made in the Logger.cs file.
    - ServValidation is created in our LoggerTest.cs file to test the valid system or user information.
    - Return: ServiceLog object that returns a boolean value False
- LoggingDataAcces: Log is sent over TCP connection to reach the DataStore. LoggingDataAccess is made in the SqlDAO.cs file
    - DataValidation is created in DataAccessTest to validate the log before finally reaching the DataStore.
    - Return: LogResult that returns a boolean True
- LoggingDataStore: Data Store Returns scalar value (0 - unknown, 1- true, 2- false) of 2 as there is an error with logging to the DataStore. The Log only partially matched the expected value. The log failed to be stored.
- LoggingServices request a new log as LogError to record the earlier invalid logging request.
- Valid is a custom object that returns a boolean value

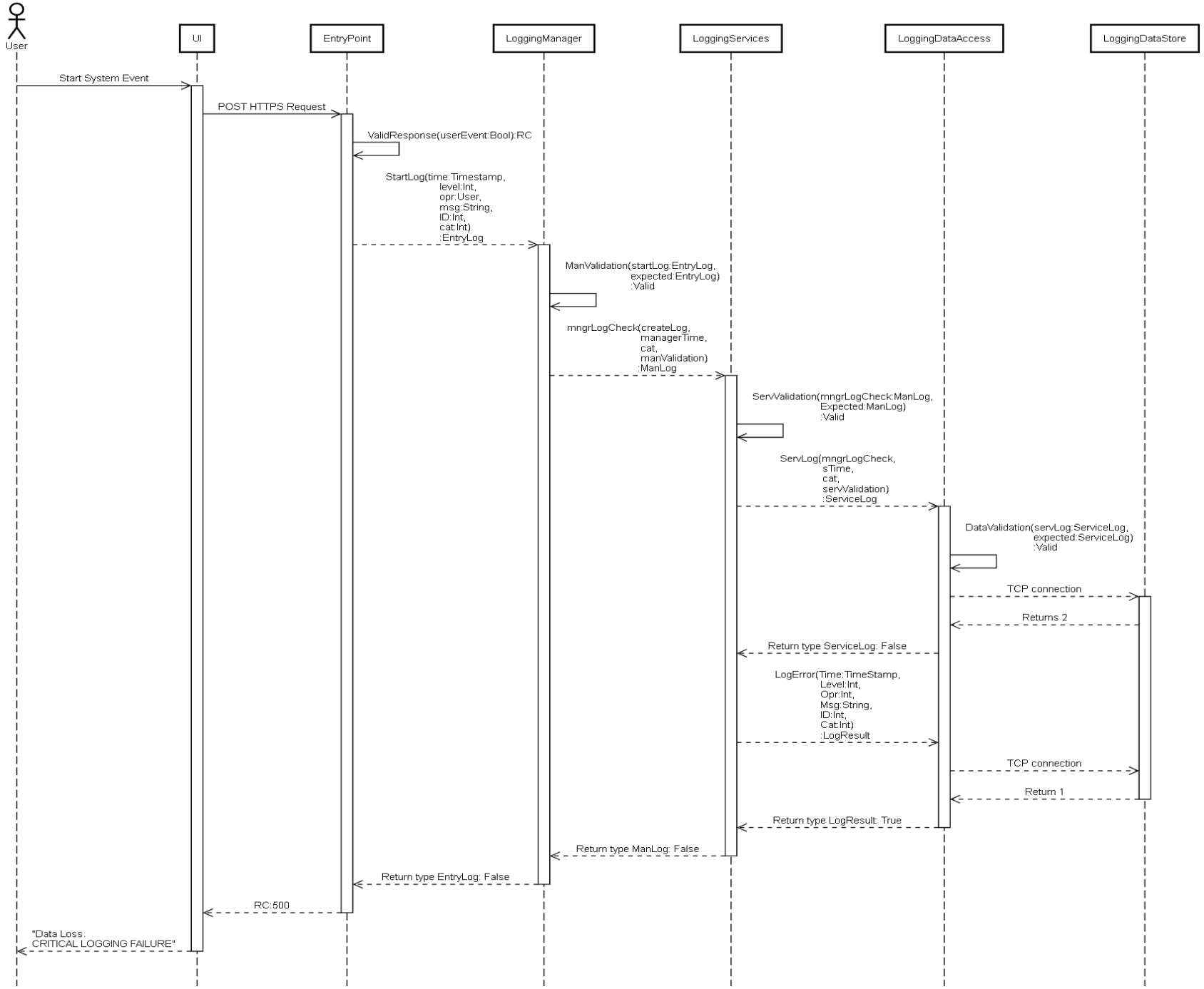
As a System, I want to Log, but the process did not accurately save all data.



System Does Not Allow Immutable Logs

- Objective: catch an instance of a Log being overwritten in the DataStore
- UI: Sends a HTTPS POST AJAX request that is validated by a Response Code
    - Return: a string notification error.
- EntryPoint: The EntryPoint would have validated a request as RC 200. The Logging Process begins.
    - Return: a Response Code is expected to be returned.
- LoggingManager: Validates the log that it is passing along information. LoggingManager is made in Logger.cs file.
    - ManValidation is created in our LoggerTest.cs file to test if the log has information
    - Return: ManLog object that returns a boolean false
- LoggingServices: Log method is officially called to start sending information to the backend. LoggingServices is made in the Logger.cs file.
    - ServValidation is created in our LoggerTest.cs file to test the valid system or user information.
    - Return: ServiceLog object that returns a boolean value False
- LoggingDataAcces: Log is sent over TCP connection to reach the DataStore. LoggingDataAccess is made in the SqlDAO.cs file
    - DataValidation is created in DataAccessTest to validate the log before finally reaching the DataStore.
    - Return: LogResult that returns a boolean True
- LoggingDataStore: Data Store Returns scalar value (0 - unknown, 1- true, 2- false) of 2 as there is an error with logging to the DataStore. A User was able to modify a table in the DataStore.
- LoggingServices request a new log as LogError to record the earlier invalid logging request.
- Valid is a custom object that returns a boolean value

As a System, I want to Log, but previous logs are modifiable.

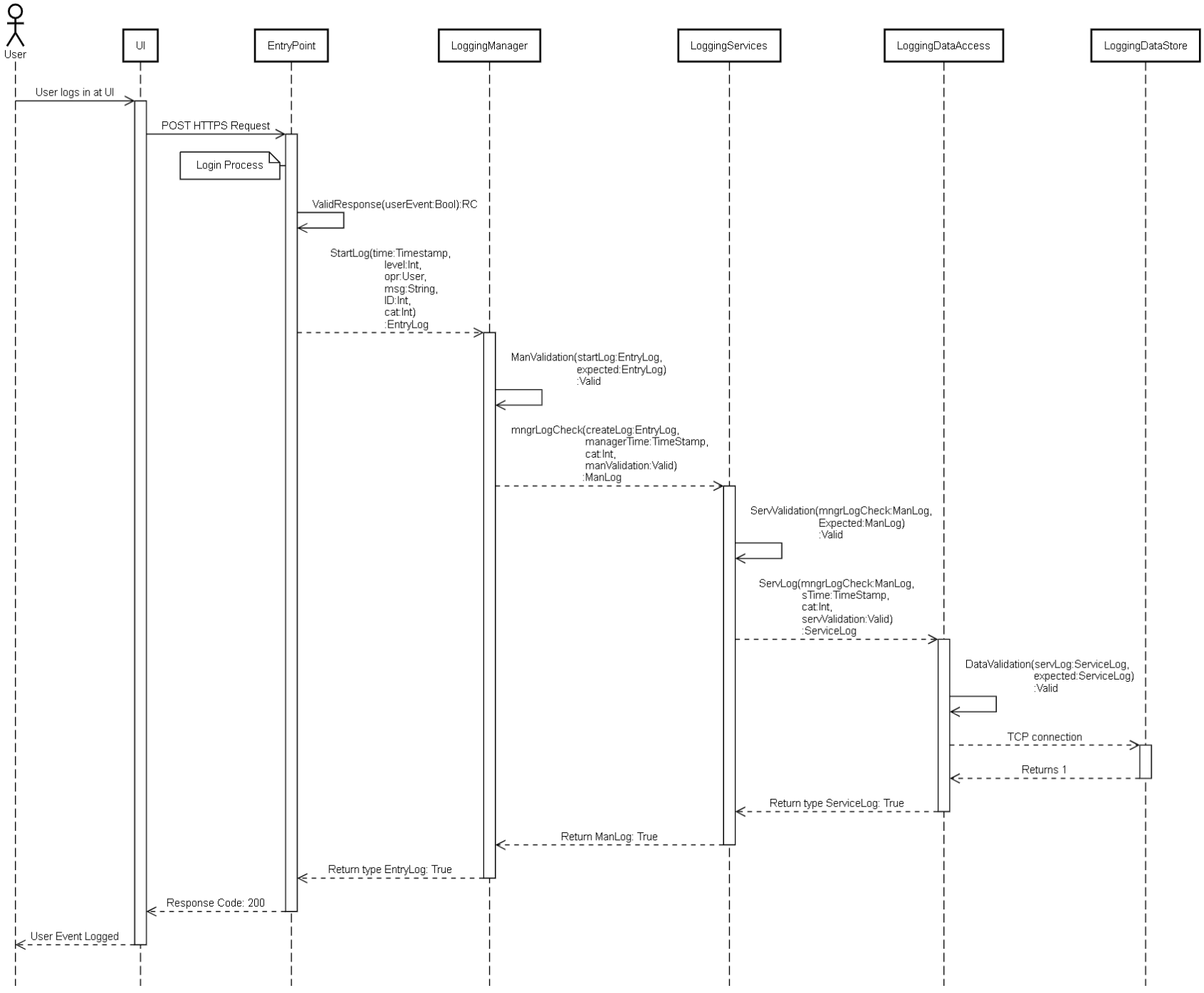


Logging Non-Core Logging Scenarios

User attempts Authentication by Logging In

- Authentication of a User is done at UI level when a User logs In.
- Process is recorded and passed along as a Successful Scenario

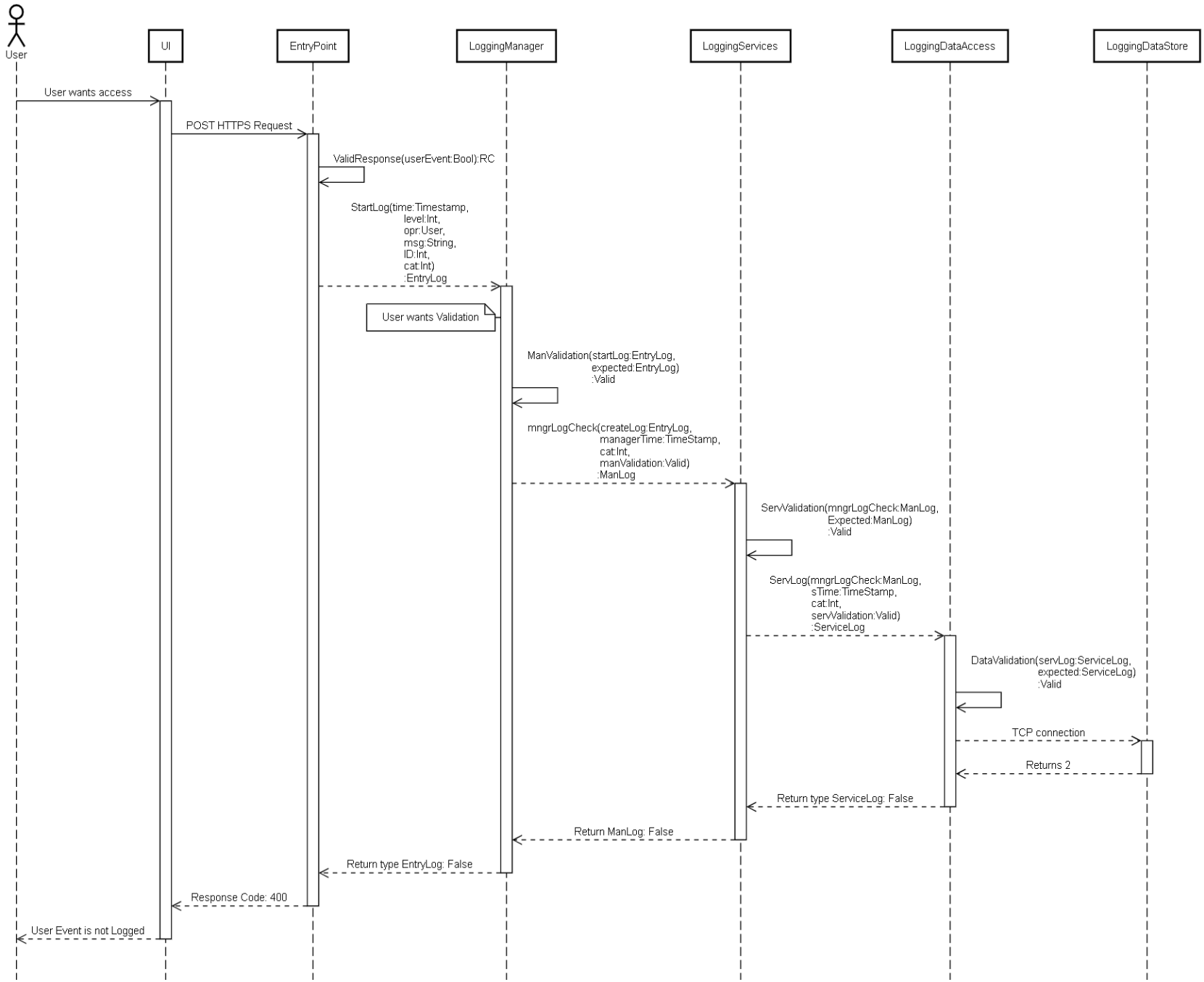
As a User, I want to login at UI, and have my authentication recorded



User attempts Authentication at Different Level

- User is not authenticated when logging in yet wants access to private data.
- User process is caught at the Manager level
- User interaction is not logged

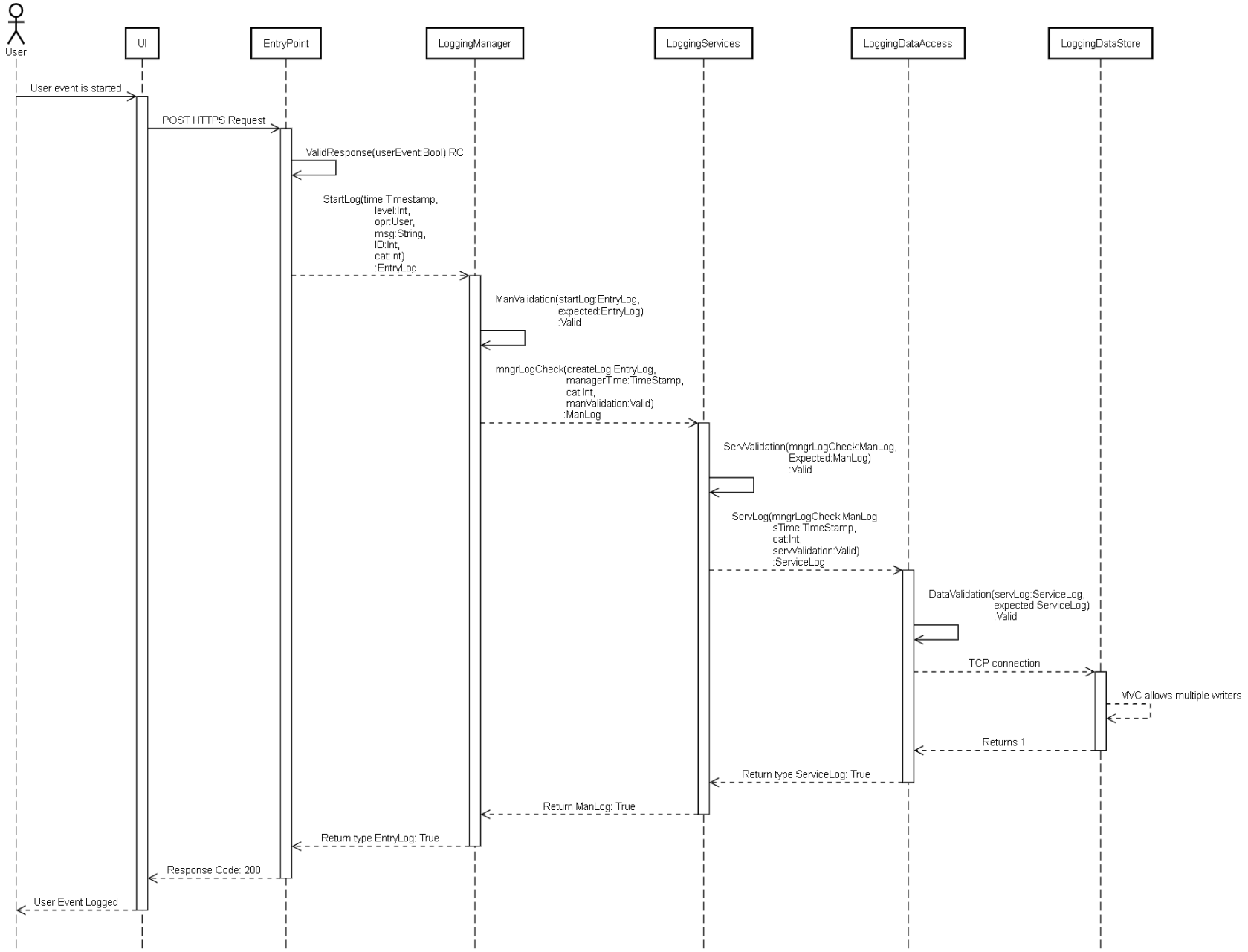
As a User, I want to access protected information, but I am not authenticated



A System wants Multiple writes at the same time to the DataStore

- Multiple writers are dealt with using the MVC model
- The controller dictates the order in which logs are dealt with

As a System, I want to handle multiple log requests, so that data is stored properly



References

https://sequencediagram.org/