Feature: Events

Feature Developer: Joseph Armas

Design Submission Date: 4/11/2023

Peer Reviewer: Rhoy Oviedo

Date Review Completed: 4/14/2023

Major Positives

1. Flow can be understood despite inconsistencies in the diagrams and some overlooked formatting issues such as not deactivating a lifeline. Notes that are on each layer represents an abstract idea of a functionality that the layer will have. Which is a good reference to look back at when doing the implementation.

2. Logging is clearly intended to be done in the service layer which is good to have listed down in terms of addressing cross cutting concerns but may need to reevaluate where it is implemented in to provide value to other layers like the manager.

Major Negatives

1. Declaring object instantiation is redundant in sequence diagram design and should be excluded to decrease diagram length. For example, in the event-create-success diagram, the DAO, manager, and service layers are shown to be instantiated in the entry point. While this is important to know during implementation, this is redundant in terms of sequence diagram design modelling since the representation of instantiation comes from the layers existing in the flow of the diagram. In other words, each layer represents an instantiated object which theoretically can be accessed by the layer before it.

2. The lifeline of the layers is represented incorrectly. For example, the design indicates that when the logger is instantiated in the service layer. The design indicates that this activates the logger layer's lifeline in the event-create-success diagram and that it returns the object as a result. As stated in the previous problem for sequence diagrams, the layer's existence in the diagram represents instantiation. While the lifeline represents the method's execution period that is activated when it is called from another layer.

3. The flow of interaction between layers is incorrectly represented. For example, the Log method belongs to the logger layer and is called later in the service layer in the event-create-success diagram. However, the design goes straight to the data store from the service layer, skipping over the logger and data access layers. This is incorrect because

the Log function comes from the logger layer that accesses a data access layer method. Then the data access method accesses the data store and returns a result back through each mentioned layer in reverse order.

4. Inconsistency using the arrow lines that represent the interactions between layers. Solid line arrows should be used to represent a method call and activating a lifeline, while dashed line arrows should be used at the lifeline's deactivation to represent returns. Using dashed line arrows as a method call can make design reading difficult for developers that are familiar with the sequence diagram lingo despite writing the method call on the line. The expectation would be that a return should be on a dashed line arrow.

5. The manager layer's lifeline does not activate through app.Run(), the lifeline starts when the method CreateNewEvent() is called through the controller. This is important to mention because the design indicates that the manager layer called its own method which is incorrect and that the data coming from the frontend and the configuration file needs to go through the controller to get to the manager layer.

Unmet Requirements

1. Several of the diagrams' titles state that a "regular user can modify {something} of an event". However, on the BRD, it is stated that only a reputable user has the ability to create, modify, or delete their own events, not regular users.

2. The BRD states that the attendance viewability can be toggled by the event creator. However, the design that describes the flow of users viewing the attendance of an event doesn't have a pre-requisite that the event attendance should be toggled on for other users to view the attendance.

3. Missing the failure case design for error handling "A Non-Reputable type user is able to create an event" where the result should return "Event Creation is Inaccessible to Non-

Reputable Users". This is critical since we want to restrict regular users from being able to create, modify, or delete events.

Design Recommendations

1. To address "unmet requirement 2", I recommend implementing the "View Attendance" button to be transparent specifically when an event's attendance viewability is set to off. Perhaps when a user attempts to use the button or has their mouse over the transparent button, a message box can appear to the user that the event creator has disabled viewability. This way users have context as to why they do not have access to view the event's attendance and decreases confusion.

2. The "Events-modify-description" view and "Events-modify-title" view are exactly the same. Just need to change the title of the "Events-modify-title" view.

3. The CreateEvent() method and IsSevenDayLimit() method are both service layer methods according to the design. However, the IsSevenDayLimit() method is called inside the CreateEvent() method also according to the design. Therefore, in order to allocate more value to the manager layer, I recommend splitting up the functionality of the CreateEvent() method by calling both service methods inside the CreateNewEvent() method in the manager. For example, the first thing the CreateNewEvent() method does is call the IsSevenDayLimit() method which will use the GetLastEventByUser() data access method and then return to the manager. If the result is permissible, then call the CreateEvent() method afterwards inside the CreateNewEvent() method. If any validation is wrong or the IsSevenDayLimit() function returns true then logging can happen straight from the manager layer and the flow doesn't need to go through all the layers again. Just the entrypoint and back to the frontend.

4. Consider using the asynchronous functions for the DAO and all the rest of the methods in each layer so that the system is completely asynchronous. Just as the professor advised during the first code review.

Test Recommendations

1. Since the design already includes having timers then it will be easy to implement the "system shall take no longer than 7 seconds" tests for all event functionalities that require it. However, I would also recommend using a CancellationToken if you use asynchronous data access methods and so another timer can be set using TimeSpan.FromSeconds()

2. To test that an event can have up to a maximum of 100 users, I would recommend int the backend to create a random generator function that inserts 100 new users to the database by randomly generating characters for both username and password using a loop. Once the users have been added to the database, I would then recommend using another loop to have the 100 users join a test event. Once this test has been completed, I would then use an e2e test to display the users in the frontend.