Feature: Scale Display

Feature Developer: Kihambo Muhumuza
Date Developer Finished: 3/21/2024

Review by: Jason Jitsiripol
Date Reviewer Completed: 3/21/2024

Major Positives:

1. I like the details given to the steps specifically given to each method such as getScaleType(nameOfScale, typeOfScale) It allows for easy design of how to implement each scale.
2. I like the parts that don’t need direct implementation through a method that has good brief descriptions about them. It allows anyone implementing these designs to know what is the expected way that all of the layers are supposed to communicate with each other.
3. I like that there are nice quantifiable ways to test these cases such as how it will fail if it takes more than 5 seconds to do. I particularly like how in both the success and fail case, the mention of this 5 second requirement such as how success has “all notes of the chosen scale are displayed in under 5 seconds” covers multiple layers. This clearly shows that this is an overarching business rule that should be enforced during each of those layers.

Major Negatives:

1. While there are nice quantifiable ways to test these cases, there’s not many quantifiable ways to test. This is an issue since some cases are not covered during the process of the scale display. For example, when validating the user’s credentials during the manager and service layer. What happens if the user’s credential is invalid? This failure test case does not show up during these test cases nor does it specify if another microservice/controller handles this. In addition, it is unclear if it is possible if it can fail during any of the methods in each of these layers. For example if getScaleType(nameOfScale, typeOfScale) fails, what happens?
2. Where’s the controller layer? We implement each of these services as a controller so it is important to consider having one. The lack of this layer also makes the UI layer unclear. I am not sure if the UI layer is the front-end or not? getScaleType(nameOfScale, typeOfScale) suggests it might be a controller, but the User interaction with it means it is probably the front-end. However, this causes that layer to be unclear.
3. While I like the details given in the methods, the return type is unclear. The return arrows also don't have enough information to determine what those types could be. While getScaleType(nameOfScale, typeOfScale) returns a “scale” type object. What datatype is a “scale” or if it is a custom type, what goes inside a scale type? Does a scale have a string inside it, or is it an array or dictionary? In addition, displayScaleNotes(notesOfScale) calls to display the notes, but what does this method return?
4. Continuing the point above, the return arrows could have some more granularity. More details could help you to know what is exactly being returned. For example, the return arrows currently aren’t even pointing in a direction. It is unclear if they

are even supposed to be return arrows. This point is assuming the dotted lines are return arrows. The return arrows could be interpreted as details for the things that are happening as it is progressing through the layers. In addition, each layer should properly return something and the return arrows currently do not reflect that fact.

5. Validation of credentials are probably not needed for this design. This is because the preconditions should make it so the user is already logged in and authenticated, so there is no reason to authenticate the user again. In addition to this point, you should probably label the preconditions at the start or else this authenticate/validation step can be confusing, especially if it is for something else.
6. Following that point, it is unclear what the manager and service layers are doing. The manager layer was already mentioned to be confusing earlier due to the fact you may not need to re-authenticate the user. If you do, you should clarify why you need to do so. In addition, the service layer is confusing as it tries to display the scale notes, but it is currently unclear what sort of business rules the service layer is enforcing. In general, it is unclear if either are enforcing any rules.
7. In addition the logging layer in this design currently doesn't communicate to the Data Access Layer to save the log into the data store. It should probably reference the successful logging LLD.
8. Compounding some of the points made above, assuming you do want an authentication/validation check, in the fail case of the LLD, the validation returns a successful authentication. This doesn't make sense as the way the design is currently structured, it looks like it returns that successful authentication after already failing the process. That step would probably be skipped. Maybe it has to authenticate before the process fails during the communication with the database, but the current LLD does not reflect that.

Unmet Requirements:

1. The preconditions are currently not labeled at the start of the transaction.

Design Recommendations:

1. I think more granularity in the design would help with coding the scale display. While you might not know exactly what you need to return in a method/function, a good LLD could help enforce the exact functionality. This is relevant since if you gave two people the same design, they might use different return types for functions like displayScaleNotes(notesOfScale). One person might return a string while another person may have that method return an array of characters.
2. I would add more quantifiable things to the methods in the LLD so you know what happens if a feature succeeds or if it fails. For example, does the validation of the user's credentials return a bool or is it some other data type?

3. I would also add some more failure cases. Other than the example given earlier, what happens if logging fails? Does the scale display feature stop and return a fail? Or does it do something else? In general, the current failure case is very unclear and I think adding more failure cases may clarify how a failure should be handled.

Test Recommendations:

1. Based on the given test cases, I would work on the Scale Display Success case first then the Scale Display failure second. By working on success first, you understand all of the failure points in your design which should allow you to properly test the failure case(s). By doing this, this should allow your test cases to properly align with the feature’s requirements.