Feature: Reputation

Feature Developer: Rhoy Oviedo
Date Model Submitted: 3/24/23
Reviewer: Ghabrille Ampo
Date Review Completed: 4/4/23

Major Positives

1. Implementing separate views for reputation functionalities is a positive since it allows for isolations between processes so that it does not unnecessarily affect each other. This also makes it clear to the user what they are capable of doing when interacting with a reputation's functionality.

2. Functionality flows are properly moving through our layers and it is clear what each layer is responsible for.

Major Negatives

1. Star allocations are not implemented properly. Currently your design says that the user who creates or completes the pin gets a 0.1 addition to their reputation. Only the user completing the pin would get that 0.1 addition and has to give a rating for the pin which the creator receives.

2. There is no validation of whether a user is authorized to be able to access a functionality. Saying that the preconditions are that the user is a certain role is ok to say on design, but there should be a validations check on the UI and Entry for the user's role. When our code will be deployed, we will not know the user's role unlike our designs so role validations are necessary. JWT should be used to get the user's role for validation.

3. The lack of input parameter validations and Input parameters for functions should be validated to make sure that it's not the reason for the functionality to not work. In the flow "As a user, I want to successfully report another user and affect their reputation score" the reputation object should be checked at the entry level if it's null or has unwanted attributes so it can exit early. This makes sure that the inputs don't break our functions and early exit validation fails.

4. Function output validations are also lacking. This is a negative since the lack of validating if the output is correct can lead to other layers getting the wrong data. In the scenario of a user getting a list of user reputation rankings there is a difference between a user getting an empty list since there are no users on the list or a user getting an empty list since from an error even though there are users on the list. Validations will handle which scenario happened and inform the user of the application what happened.

5. Views doesn't show or explain how it got to that view. There should be an arrow pointing to this view that leads back to the button that leads to the functionality view. This makes it unclear where the user is coming from before the view and where that view can lead next.

Unmet Requirements

1. User reports should be after a user completes the pin and gives that pin a rating on whether it was a properly created pin or not. Currently implementations of users reporting each other directly is wrong since our reports should be based on their pins.

2. Lacks checks on functional and non-functional requirements of the feature. This should be implemented in the manager layer where it can check if processes are being done within the requirements we specified.

Design Recommendations

1. Unclear on how sql statements will be created but recommends to build using a query builder instead of hardcoding.
    a. Positive: This will get rid of the hardcoded sql statements that are prone to sql injections.

2. Use stored procedures when inserting to the database to make sure that the data inputted is in the proper format.
    a. Positive: Stored procedures have baked in sanitizations and do not require a return value.

3. I noticed that the rating check is done through the front end but should be done in the backend. Maybe when in the reputation manager, after a process of changing someone's reputation you do a check on their reputation if they should be promoted. This way it's secure since it will be done in the back end and also saves another roundtrip since changes are done without needing to go back to the front end.
    a. Positive: Loses one round trip since checking to change a user's role to reputation is included in the flow of the user gaining or losing stars.
    b. Negative: Anything relating to a user gaining or losing stars would gain the extra overhead in the manager layer to check if their rating is enough to change their role.

4. When implementing users rating a user's pin after completing it. The view "UserReportCreationView" can be used to ask for the user's rating.
    a. Positive: The user reports view can be reused since the unmet requirements explain that user reports were not implemented properly.

Test Recommendations

1. System handling a user having 5000 reports. This can be done with creating a csv of 5000 user reports to put in the database. Then having an e2e test on whether the user loads and sees its rating and if the system handles large amounts of reports.

2. System being able to handle multiple users being displayed. This can be handled by having multiple users that are already in the database or are added in using a csv. Then an e2e test would check if the list of user rankings with multiple users on the list will properly load.

3. Show users move rankings within the ranking list. This e2e test can show that rating changes are properly working and that the user role change also works properly. This can be done by having a different user complete the pin then we check if the creator of the pin's rating changed and moved around the list.