Feature name : Listing Profile
Developer name : Jett Sonoda
Date developer submitted : 04/15/2023
Reviewer name : Bryan Tran
Date review completed : 04/17/2023

Major Positives and Negatives

- Top design qualities:
    - Wide coverage of failure cases. Went through all of the possible success and failure cases for each use case and even account for edge cases.
    - Design of relational tables are solid and have meaningful relationships that are easy to understand as well as reduces the amount of redundant data to be stored.
    - Layers are clear and separated with each layer serving a purpose and adding value to a method within the design. Flow and logic of methods within the layers are concise and reasonable.
- Top flaws:
    - Revisit and cleanup sequence diagrams to ensure lifecycles align with the layers (methods should end at the same layer it starts).
    - Results returned from the controller layer are vague and do not provide what objects are returned.
    - Missing labels on diagrams where it is split into separate parts to add clarity.
    - Missing bar to indicate lifecycles for the diagrams, making it hard to see where a method starts and ends.
    - Remove or condense database calls within certain use cases are needed to reduce the amount of database calls to the backend to speed up a request.

Unmet Requirements:

- **BRD Reference-Listing Initialization (Has an account)**
    - **Failure Criteria** not accounted for in wireframe and design document:
        - “Prompt user ‘Listing initialization error. Refresh page or try again later.’”
    - Failure Criteria not accounted for in wireframe and design document:
        - Prompt user “Listings failed to load. Refresh page or try again later.”
- **BRD Reference-View all owned Listings**
    - Success Criteria not accounted for in design document:
        - “If user has no listing, view is empty”
        - “Otherwise, a list of published and listing drafts will be displayed”
    - However, the design document vaguely states “All user listings displayed”. Unclear how the scenario where the database returns no listings is handled.
- **BRD Reference-Edit a Listing/Listing Draft**
    - Failure Criteria not accounted for in wireframe and design document:
        - “Prompt user ‘Listing edits error. Refresh page or try again later.’”
- **BRD Reference-Create a Listing Draft**
    - This requirement is missing entirely and not accounted for in the design document. The functionality requires the user to be able to create a Listing Draft where the listing is unpublished to the public and saves the information that they have inputted.
- **BRD Reference-Delete a Listing/Listing Draft**
    - Failure Criteria not accounted for in wireframe and design document:

        - "Prompt user 'Listing deletion error. Refresh page or try again later.'"
- **BRD Reference-Publish a Listing/Listing Draft**
    - Failure Criteria not accounted for in wireframe and design document:
        - "Prompt user 'Publishing listing error. Refresh page or try again later.'"
- **BRD Reference-Listing Rating and Commenting**
    - Precondition not accounted for in design document:
        - "User has not left a review on Listing previously".
        - This would require a database call to the backend to query the ListingRatings table within the database to check if a user has already left a review.
    - Failure Criteria not accounted for in wireframe and design document:
        - "Prompt user 'Listing rating and comment error. Ratings failed to upload. Refresh page or try again later.'"
- **BRD Reference-Edit Listing Review**
    - Prompt user "Listing review editing error. Refresh page or try again later."
- **BRD Reference – Delete Listing Review**
    - Prompt user "Review deletion error. Refresh page or try again later."

Document Clarity Recommendations:

- **LLD-Edit a Listing/Listing Draft**
    - Having labels on the diagrams that are broken up into different diagrams would be useful in immediately identifying each diagram rather than labeling them as "Part I", "Part II", etc…
- Since diagrams are very large horizontally, it is very difficult to read the small text.
- Other feature documents included the activity flow for each of the use cases. For consistency and for further elaboration and clarification, it would be useful to have these diagrams.
- **All Diagrams**
    - The "AJAX HTTP …" requests and response calls are vague and do not provide a lot of details on what objects are being returned from the backend. Provide what type of objects are being returned or what properties the data returned has.
    - In the sequence diagrams, they should show the lifeline of each of the method calls. Not showing the lifeline makes it harder to follow the lifecycle of the method starts and ends.

Design Recommendations:

- **LLD-View all my Listings**
    - Unnecessary database call to backend to get the email for the user. The email can be accessed from the access token as it is stored as part of the token.
    - The SELECT statement is invalid with the aggregate AVG function as it required you to have a GROUP BY clause.
    - Rather than looping and performing multiple database calls to grab a single listing's average rating, it can be performed in a single SQL statement as shown below. Doing so will reduce the amount of complexity of the code as well as speed up the performance of the application.

```
SELECT ListingId, AVG(CAST(Rating AS Float)) as AvgRating
FROM [DevelopmentHell.Hubba.ListingProfiles].[dbo].[ListingRatings]
WHERE UserId = 3
GROUP BY ListingId
```

    - This query will return rows of listing id's and calculated average ratings with the specified UserId.
- **LLD-Edit a Listing/Listing Draft**
    - Inconsistency in the View layers where the AJAX HTTP requests and responses occur. The requests and responses do not align with the correct layers for all parts.
    - Part II – Database calls
        - The for loop for each ListingAvailibilities object needs to encompass all three of the ListingAvailibility modifying methods. The way the sequence diagram currently shows how each ListingAvailibility object is modified, it is completely separate from the methods that are actually using the object.

Test Recommendations:

- **View all my Listings**
    - Test to ensure that all listings from a user are all visible on the Listings view. If there are none, display an empty view. Also, add tests to ensure cases where a listing does not have any ratings to display. These tests would satisfy the success criteria where the owner's listing or listing drafts are displayed if any exists.
        - This is high priority as a user would need to be able to see their listings to edit and publish them in the future.
- **Edit a Listing/Listing Draft**
    - Test for editing every parameter and field of a listing and saving the listing data. For the listing availabilities, there would need to be tests that cover the editing, deletion, and the creation of new time slots or availabilities for a listing. Also have test cases that would result in validation failures for any text input. These tests would ensure that the listings that are changed by the user are saved within the database upon saving.
        - This is high priority as a user needs to be able to edit their listings in order to publish them.