Feature Name: Request Management
Developer Name: Sierra Harris
Developer Submitted: 3/3/2023
Reviewer Name: Abhay Solanki
Review Completed: 3/7/2023

Major Positives and Negatives:

- Positives:
    - The Design is following the layer design set in high-level design.
    - It is a very simple and understandable design that can be read and implemented by non-team or new team members easily.
    - SQL procedures are written down exactly which makes the design very clear and prevents ambiguity.
- Negatives
    - One function is executing three different SQL procedures RequestDAO.getUserRequest in:
        - Accept Request
        - Decline Request
        - View Request
    - In the failure cases, the email with the issue description is sent only to the user, not an admin.
    - The feature is simple enough to have no business rule, and yet there is the unnecessary transfer of power from Manager to Service and then to DAO.

Unmet Requirement:

- Forgot to check the role (authorization) in each design.
- Forgot to dedicate or create a case for the following Failure Cases:
    - A service request is not displayed.
    - The request is not accepted/declined.
    - A notification is sent to the requestor, but with the wrong date.

Design Recommendation:

- The wireframe is missing page scrolling buttons which would be important in the case of overflowing requests.
- Keep the side navigation bar enabled so users can freely move between services.
- For each part of the feature, the use cases are simple enough that they do not have any business rules. This means that you are passing control from the Controller to the Manager and then to the Service, with the Manager only transferring power to the Service. Therefore, I suggest calling the DAO directly if the returned list does not need to change. Alternatively, you can directly call the Service, which can handle minor business rules if there are any.
- Make different methods for calling different SQL procedures, as having the same name will be an error.
- Add authentication and authorization components or references in the sequence diagram to meet the requirement.

Test Recommendation:

- Frontend Testing:
    - Test if the accept button adds the service to the user's accepted service list, and is deleted from the service request list for the particular user.
    - Test if the decline button deletes a service request from the service request list for the particular user.
- Backend Testing:
    - Test all the HTTP requests to the controllers, and check for the correct type of response/return.
    - Create a test user, and test if an API call to the controller can accept or decline requests.

Feedback:

- The layered structure is following the high-level design, but here the feature is very simple with almost no business rules needed. So as I mentioned earlier, in Design Recommendation, remove unnecessary transfer of power from Controller to Manager.
- As mentioned in High-level design we are using a relational database. So, putting the exact SQL procedure removes any confusion in implementing the feature. As that is the most important part of the flow. Which makes it easier to implement this design by another team or developer.
- Minor thing the requirement for authentication is included to the side as a note, rather than shown in the sequence diagram. I think it will be a good idea to introduce it in the design.