Chores List

Developer: Githel Suico

Submitted: 4/5/23

Reviewer: Haley Nguyen

Reviewed: 4/10/23

Criteria

Major Positives & Negatives

Positives

- Sequence diagrams follow a consistent formatting with cascading request/response structure which makes it easy to read. Each object has a lifeline for its constructor and the return value to the correct dependent object.
- Each of the sequence diagrams includes method signatures, queries, HTTP response codes, and messages. It's very clear what methods are being called, each one detailing the function name, and when it's called.

Negatives

- The sequence diagrams are missing logging. This is important to have as it would help the developer to know what information goes into the log when integrating logging into the feature.
- The sequence diagrams are also missing input validation. In the BRD, the bill has valid characters and length limits for the “name” and “time to reset”. The sequence diagrams do not include these checks and which components are responsible for carrying out those checks.
- The designs do not necessarily follow MVC architecture. While the designs do include a controller (ChoreController) and a view (ChoreListToDoTab & ChoreCreationModal), the sequence diagrams are missing a model to define the data structure for a chore.
- The sequence diagrams are also missing a service layer to encapsulate the business logic. Without the service layer, it would be more difficult to write unit and integration tests later on.

Unmet Requirements

- The user can enter a new chore into the list with the following required inputs:
    - (Required User Input) Input chore name
        - Valid input: a-z, A-Z, 0-9, 60 character limit
    - (Required User Input)Input time to reset
        - Valid input: Integer values only, must be larger than current date
    - The sequence diagrams are missing input validation listed above. Hence, this requirement is not met by the design.
- Changes to the chore list will be logged to a database in the form of activity and/or errors (if any).
    - The sequence diagrams do not include logging, therefore this requirement is not met by the design.

Design Recommendations

- Logging is required by the BRD and in general is necessary for the developer to understand what goes into the logging when implementing
    - To implement logging, I would suggest creating a flow between the ChoreManager through the ChoreDBDAO to a Logger layer after the insertion of data into the Chore DB. Where the ChoreDBDAO should receive a successful/failure status in order for the ChoreManager to send the result to the Logger layer.
        - Positives: Debugging, monitoring, auditing, and performance tuning of the system will be a lot easier with logging. Implementing logging can provide valuable insights into the systems behavior and performance.
        - Negatives: Logging can potentially expose sensitive data or information such as user credentials if not implemented correctly.
- Input validation should be implemented both in the frontend (by the view) and backend (by the service layer or manager). The frontend validation should include a use case of the user adding a chore with an invalid input for the name and/or the time to reset.
    - You can do so by adding an if statement at the beginning of the sequence diagram between the user and ChoreListTodoTab to detail what would happen if the user inputs an invalid input for name and the time to reset.

        - Positives: This also creates a better user experience as it can reduce frustration and increase user satisfaction. It also prevents bypassing validation. If you only implement input validation in the backend, an attacker may be able to bypass validation by sending a malicious input directly to the backend.
        - Negatives: Additional development time, and if input validation is not implemented correctly, it can introduce a new security vulnerability.
- Additionally, there should also be backend validation.
    - You can implement this validation by either making your manager layer or service layer do the check. Validation at the service layer can help to catch errors early before the data is processed by the system.
        - Positives: This would keep data integrity. Validating input at the backend ensures that only valid data is stored in the database. This can help prevent data corruption and any other issues that could arise from storing invalid data. Additionally, Implementing both validations creates multiple layers of defense, so if one layer fails, the other layer can catch it.
        - Negatives: Additional development time, and increase in system complexity which could make it more difficult to debug the system.
- To follow the MVC architecture, I would suggest adding a model into the design.
    - To add a model into the existing design, I would suggest creating a ChoreModel layer between the ChoreController and ChoreManager. The ChoreModel should specify the components of a chore (ex. Name, reset_time, notes, etc) in order to prepare it to be sent to the database.
        - Positives: A model provides a better way to structure and manipulate data within the system. The separation between data manipulation and business logic into different layers can help developers maintain a clear separation of concerns which in return make it easier to maintain overtime.
- In order to make the design more modular, I suggest adding in a service layer in order to encapsulate the business logic.
    - To add a service layer into the existing design, I would suggest adding the new layer before the ChoreDBDAO. This would also be the perfect

place to validate the data once more in the backend before it reaches the database.

        - Positives: Separating the business logic into the service layer can make it easy to change or expand upon the application's functionality in the future. This can allow us to modify without having to touch the underlying data access layer. Additionally, having this separation can make it easier to write unit tests for the service layer.

Test Recommendations

Integration Tests

- To test updating/deleting a chore, I would manually create, update and delete chores and make sure that the chore gets updated and/or deleted for every user.
    - Arrange the test chore to update and delete. Make sure that all the users in the same group can see that chore.
    - After updating the chore, check if the chore objects are the same across all users.
    - Delete the chore and call GetChores on the group_id
    - Check that the deleted attribute is the same for all users.
    - In regards to the requirements, this test ensures that the chores are consistent for all users within the same group. Without a passing mark for this test, chores might be displayed differently for different users. Which defeats the purpose of keeping track of what chores are being done and if they are completed.
- To test for old/deleted chores being properly removed from the database, you can
    - Arrange a chore to be deleted. And check it is added to the database
    - After deleting the chore, assert that the chore is no longer in the chore list and does not appear in the to-do list nor the to-do history list. Additionally, check that it is removed from the database.

Unit Tests

- To test input validation, I would attempt to create a chore with invalid input fields for the name, time to reset, and notes. I would check the string length and valid characters for each field. It is also important to note that the error should not come from the data access layer but from the view/controller layers.

- I would also test if the chore lists start on Monday (BRD requirement).

References

[1] githelsui/WeCasa: @ Team HAGS JP. (2023, Feb 17) *GitHub*
https://github.com/githelsui/WeCasa/blob/main/docs/BRD%20v2.0.pdf