Feature: Personal Overview Center

Developer: Brandon Galich

Submission Date: 03/16/24

Reviewer: Sarah Santos

Review Date: 03/17/24

Major Positives

- Comprehensive and detailed diagrams. Diagrams outline sequential steps of an authenticated user navigating the Personal Overview Center. This demonstrates a clear comprehension of how the Personal Overview Center works within the front end and back end of the system. In addition, the return response objects of each class are detailed throughout all success and failure cases of the feature.
- Various diagrams for specific success and failure scenarios. Each diagram focuses on a specific outcome produced by the feature. Demonstrates thoughtful comprehension of any possible outcomes such as Partial Filtering, No Filtering, and Outside Criteria of data.
- Demonstrated separation of concerns. Design displays organized sections of each class and their function within the feature. Microservices such as sorting by descending or ascending were displayed separately. If any maintenance or additions were necessary, the design would only need to be changed within a particular class/function.
- Distinction of filtering reservations by date. Diagrams are distinguishable when the date filter is applied by the user and how each class functions differently, such as the PersonalOverview class needing to check the filtered dates within the data store.

Major Negatives

- Ambiguous sorting list diagram: The success design for sorting a list by ascending or descending order does not provide a clear distinction that the dates are what is being sorted. In the future, it may be possible to have the option to sort them by location or company name. Therefore, it is important to display what exactly you are sorting in your diagram.

- Missing possible outcome of failure of No Data Filtering Diagram. What if the user uses the date to date filter and inputs the date of their first reservation (or earlier) to the present? Theoretically, the filter should still display the reservations even if the rows affected are the same amount as the non-filtered query. Yes, the default display already shows all the reservations, but considering this case will help prevent the feature from breaking.

- Not major, but frequent minor typos: Throughout a few of the diagrams some typos must be edited in order for them to make sense and follow the requirements. One major typo I found was in Success Display for Calendar view - From Date and To Date where actions 31 and 32 describe the “listView” and “list format”. A minor one is that all diagrams (except timeout failure) should display “stopwatch <= 3 seconds” instead of “< 3” to ensure that the feature will not time out if the stopwatch reaches 3 seconds. It is important to review that each action line corresponds to the requirements and respective diagram outcomes.

Unmet Requirements

- Confirmation of viewing selection. According to the BRD, the user must confirm their viewing selection after choosing their desired viewing option. Ensure that you add a confirmation button before proceeding to fetch the reservations and display this in all necessary diagrams.
    - This confirmation button should also be implemented after the user inputs the dates to filter their reservations by.
- Display Overview Categories. According to the BRD, the user must be given the option to select what category they can overview (All, Spaces, Items). Although we are currently only implementing space reservations, it is important to consider this under the open-closed principle in which your design can be modified in the future.
- Failure of sorting list view. Provide a diagram of the possible case of the sorting function failing. How will your system check the reservations are sorted properly and how will it raise an error if it fails? It is important to test that the sorting function works correctly.

Design Recommendations

- Detail list and calendar view functions. Consider displaying the details of how the listView() and calendarView() functions will take into account all reservation info data in your diagram. This will provide a clear understanding of how those functions will read in the data.
    - Positives:
        - Display any necessary parameters for the view functions
        - Demonstrate how the .getAllReservations call from Controller to PersonalOverview impacts the view functions
    - Negatives:
        - Limits implementation flexibility, causing difficulties in future modifications.
        - Possible overengineering. This addition to the design may be too detailed and may add unnecessary complexity to the design, wasting efforts on noncritical elements of the feature.
- Implementing a confirmation button after the user's filter inputs. As stated in the unmet requirements and major negatives section, this integration can help the functionality of filtering the dates. If you still want to raise a no filter failure error, the error can be raised after the user confirms the dates in the case where the user inputs the date of their first reservation (or earlier) to the present.
    - Positives:
        - This is a more functional approach to raising the no-filter error. The user can be notified that their input dates are invalid or that the feature cannot properly filter them.
        - Reduces user error by having them confirm they imputed the correct dates to filter in their desired view.
    - Negatives:

        - UI design complexity. This adds more complexity to the design and logic of the UI. It is important to consider whether the cost is worth implementing.
        - Delay in response. The response to the filter may also be delayed because of the additional resources used to confirm both the viewing option and the filtered dates.

- Combine timeout failure diagrams. To reduce the number of failure diagrams you can combine your timeout failure diagrams to one or two. Either have one diagram to represent how your feature handles a timeout, or two diagrams to display one without a filter and one with a filter.
    - Positives:
        - Reduces repetitive diagrams for the same error
        - Same comprehension of error under one (or two) diagram(s) makes it easier for the reader to understand and identify the main issue
    - Negatives:
        - The reader may not comprehend the different test cases that should be implemented to ensure the feature fully functions within three seconds
        - Lack of distinction. Possible test cases may pass under a list view but not a calendar view, or pass without the filter and fail with the filter

Test Recommendations

- End-to-end testing is important to ensure the user will have a seamless interaction with your feature.
    - Test reservation data is being properly fetched from the data store
        - Ensure all reservations are displayed from earliest to latest according to the preset requirements.
        - Ensure correct reservations are fetched after filtering dates
        - Ensure all reservation details are accurate
    - Test switching views from calendar to list is seamless
        - Ensure user can switch view options of reservations without any errors (or properly raised errors)
        - Ensure user can switch view options while under filtered dates
            - User should not need to filter the same dates twice in order to switch views
    - Test the sorting functions
        - Test with and without the date filter
        - Ensure all reservations are properly sorted
- Unit testing variations of filter inputs (out of order, in order, before first reservation to present, NA)
    - Test valid input dates (ex. 2023-12-31 to 2024-03-10)
    - Test input dates that are valid but out of order (ex. 2024-11-17 to 2023-02-24)
    - Test invalid dates (ex. 2022-31-01 to 2024-02-30)
    - Test first reservation date to present
    - Test filter with no input dates
- Stress testing the feature's capabilities
    - Determine the maximum number of reservations the system can handle. Observe when the system’s performance declines after a certain number of reservations.
    - Determine how far into the past and future the views can be displayed. Observe when the system’s performance declines after a certain range of dates.