Space Surfer: Waitlist

Developer: Sarah Phan

Developer Submitted: 03/16/2024

Peer Reviewed By: Brandon Galich

Review Complete On: 03/18/2024

Major Positives

1. Clear User Flow Interaction (Wailist-Join-Success.png, Wailist-Join-Failure.png, & Leave Waitlist-V2.png)

   The diagrams effectively show the process of the user's interactions with the system when joining or leaving the waitlist.

2. Includes An Important Check (Wailist-Join-Success.png & Wailist-Join-Failure.png)

   Validation check 'isUserOnWaitlist**'** ensures that the same user cannot enroll in the waitlist more than once.

3. Clear Success Leave Waitlist (Leave Waitlist-V2.png)

   The system provides clear understanding of the user leaving the waitlist after the action was completed. This also implies another user being considered in this action as the next user on the list will be notified about them being next on the waitlist.

4. MailSender Object(Wailist-Join-Success.png & Leave Waitlist-V2.png)

   Helps with sending the user a notifications about their action being completed with their own appropriate responses. 'Leave Wailist-V2.png' contains methods allowing to send separate emails to different users.

5. HTTP Responses(Wailist-Join-Success.png, Wailist-Join-Failure.png, & Leave Waitlist-V2.png)

   Contains the appropriate responses that are sent to the UI when the process was successfully completed.

6. Descriptive Purpose and Labels(Leave Waitlist-V2.png)

   Describes the purpose of the function when leaving the waitlist after being enlisted. As well as the two users being labeled with their roles in the sequence diagram.

Major Negatives

1. Undefined user (Wailist-Join-Success.png, Wailist-Join-Failure.png, & Leave Waitlist-V2.png)

   The diagrams do not indicate if the user is logged in or it can be any user, regardless if they have an account.

2. WaitlistEntry object vague(Wailist-Join-Success.png)

   Not much information about what the WaitlistEntry object will be doing. An instance is initiated, but what is being returned or executed.

3. Lack of Contextual information for the UI (Wailist-Join-Success.png, Wailist-Join-Failure.png, & Leave Waitlist-V2.png)

   Regarding the UI, it lacks clarity in a way of its purpose. The UI has no clear form of communicating between the user and the controller. There is no indication that the UI has any UIcomponents like a button, link, or form to receive the user's request to POST the user's input.

4. The Validation Check 'isUserOnWaitlist' Is Not In The Correct Step (Wailist-Join-Success.png & Wailist-Join-Failure.png)

   The validation check 'isUserOnWaitlist' checks to see if the user exists in the database, but it is called before the SQL command retrieves the information. The SQL command gets the full list depending on the 'SpaceID' and 'Time' indicating the result is always greater than 0 and therefore the 'hasError' is true. Either the step is incorrect, or the SQL command is built incorrectly where the 'username' is not used.

5. The Validation Check 'isUserOnWaitlist' Is Not In The Correct Step (Wailist-Join-Success.png)

   The validation check 'isUserOnWaitlist' checks to see if the user exists in the database, but it is called before the SQL command retrieves the information. The SQL command gets the full list depending on the 'SpaceID' and 'Time' indicating the result is always greater than 0 and therefore the 'hasError' is true. Either the step is incorrect, or the SQL command is built incorrectly where the 'username' is not used.

6. Clarify Email Sent From MailSender or UI (Wailist-Join-Success.png & Leave Waitlist-V2.png)

   It implies that the UI will be sending the email to the user. The MailSender object is used as a creator than sending an email to the user. As well as indicating if the MailSender will contain a response object if it has an error.

7. Sequence for Next User In Line (Leave Waitlist-V2.png)

   The sequence for the next user in line indicates that the user will be receiving an email from the user that cancelled their reservation.

Unmet Requirements

User's spot in the waitlist recorded in the data store – No indication of it being recorded

There is no indication where the user's spot change is being recorded in the sequence diagram. As well as the indication of which user is having their spot being changed.

Waitlists are removed from the database after the time and date for that reservation has passed – No diagram for this success and failure

In the BRD, it mentions that as a success the waitlist for the reservation will be removed after the time and date has expired. For the failure, the reservation is still shown after the time and date has expired. This can lead to users still able to join the waitlist, which can lead to problems.

Waitlist takes 3 seconds or less to update – No indication of this validation in any diagram

There is no indication showing how the waitlist would update less than 3 seconds. There is also no validation to check if the process took less than 3 seconds to complete. This would lead to believe that this process will always be less than 3 seconds.

Waitlist takes 3 seconds or less to update – No diagram for a failure

There is no diagram that shows the outcome of the process taking more than 3 seconds. The user should be displayed with a prompt of "Time Limit for Operation Exceeded."

Waitlist option appears even if that reservation is still available to the user – No diagram for this failure

From the BRD, it mentions that the option for the waitlist will not appear if the user is in the waitlist, but there is no diagram for this failure. A message of "Come back later" should be displayed if the user should request to join with the option still displayed.

User is not shown in the waitlist after joining – No diagram for this failure

There's no diagram to show the possible failure of the user not being added to the waitlist after an attempt was made. Two possible failures were described on the BRD. One failure involves the outcome to be considered as successful, but was the user was still not added to the waitlist. The other failure regards the outcome being a failure and the user was not added.

User is not emailed – No failure diagrams

There are no diagrams that address the a failed scenarios of the current user or the 'next in line' user where the MailSender can result in an error. Therefore, not send any emails to the users.

Other failures diagrams – No failure diagrams

No diagrams that address if there is a database connection failure or WaitlistEntry failure. In the BRD, there is an indication of the database connection failure meaning that no data can be stored or retrieved. For the WaitlistEntry, there is no indication if it can be a point of failure or not.

Design Recommendations

1. Add a Stopwatch:

   In your diagrams, you could be able to add a stopwatch to verify if the action took more than 3 seconds. You can add a start stopwatch to start the timer at when your action gets executed and add an end stopwatch to verify if the action took more than 3 seconds. This would help verify and check if the action should not end in an error. If the action completes within the expected time, proceed as usual. If it exceeds the expected time, then consider logging an error or providing appropriate feedback to users.

   Positives:

   Allows to measure the elapsed time of the operations
   Can measure individual method execution time to use in different validation checks
   It provides a more precise measurement of the elapsed time

   Negatives:

   Stopwatch just adds some complexity to the code where you need to manage its lifecycle.
   Proper use of try-catch blocks or error handlings will be necessary to handle exceptions
   Synchronization will be needed to ensure thread safety

2. Get the waitlist from the database first:

   In your diagram, you have the validation before getting the waitlist so it would not be validating anything. You should get the waitlist from the database first, then you should validate if the user is in the list. If the list is empty or the user does not exist in the list, then it should call the WaitlistEntry object to add the user to the waitlist.

Positives:

Allows it to check if the user exists in an up-to-date data retrieval from the database
Reduces redundancy on validation checks
Saves process time and resources

Negatives:

Retrieving the entire waitlist might be resource intensive
Handling errors where the waitlist is empty or if the user doesn't exist requires additional logic

3. MailSender sends the email:

Your diagram shows that you only create instances of the MailSender. I would consider you to have arrows go all the way to the user and to have it say the email was sent. The frontend does not handle any email being sent and it should prompt the user that a "Confirmation email was sent".

Positives:

Provides a clearer feedback about the user receiving the email and understanding the user receives a prompt that a "confirmation email was sent" to their email/username.
It will assure the user that the request process was successful

Negatives:

SMTP service can be slow or unavailable
If email sending takes some time, it can impact other operations

4. Add response objects to sections where there can be an error:

By doing this, you can verify at what point in the process it encounters an error rather than relying on one response throughout. Adding a response object on the WaitlistEntry and the MailSender would allow us to check if there are no errors to append to the WailistService response object. This is also a way to verify if the mail was sent and verify a failure outcome.

Positives:

Explicit responses improve in clarification and communication

Responses provides feedback to the user about their actions or errors

Negatives:

Each response object consumes memory

Handling multiple responses in different scenarios

5. Providing a better flow of the frontend:

   This would help with identifying the successful outcomes when the action is completed. At times, it says “confirmation email”, but not much is mentioned about it. This assumes that the user just receives a confirmation email, but nothing happens in the UI portion since nothing is described. By indicating that you would have an HTML and a JavaScript class, you would be able to structure a foundation of how the frontend would interact with the user. This also shows what would be prompted to the user instead of an arrow indicating that an HTTP Post was requested. Also indicating if there are going to be any validation checks, calculations, or any UI updates that could help offload any work required in the backend reducing server load and creating updates to the browser directly.

   Positives:

   Help create a better understanding of how the UI would interact with the user.

   Validation checks,calculations, or UI updates in the frontend to help with offloading

   Creates a clearer view of the outcomes it can have

   Negatives:

   Javascript usage can impact loading times

   Proper security measures would be required

Test Recommendations

<u>Join Waitlist</u>

1. Same user joins the same waitlist: Checks to see if the same user can join the same waitlist. If they can, then it's a failure.
2. User joins the waitlist, but is added on every waitlist of that reservable space: Check to see if the user joined a different or multiple waitlists of the reservable space.
3. User can join multiple waitlists from the same reservable space: Check to see if the user is able to join the same waitlist for the reservable space at different time slots.
4. Validate the user does not cut the line Validate that the user did not cut in the waitlist.

5. Correct user was added: Check if the user requested to be added to the waitlist is the correct user being added.

<u>Leave Waitlist</u>

1. User leaves the waitlist, but is removed from a different waitlist: Make sure the system does not remove the user from their other waitlists they have joined.
2. User leaves the waitlist, but is removed from every waitlist from the same reservable space: The user must be in multiple waitlists of the same reservable space in different time slots to verify that they are only removed from the correct time slot.
3. Check if the correct user was removed: Other users should not be removed from the waitlist.
4. User not removed from the system: Verify that by removing the user from the waitlist does not remove him from the system entirely.
    - ➢ If a table is needed, make sure it is a child table in the case of Cascade Delete to have the user deleted from the waitlist if they decide to completely delete their account.
    - ➢ Consider updating the positioning of the users if this does happen.
5. Updated waitlist positioning: Check if the system correctly updates the waitlist position for the remaining users in the waitlist when the user leaves.

<u>General</u>

1. Concurrency: Multiple users try to join or leave the waitlist simultaneously ensuring the system can handle concurrent requests.
2. Waitlist update: Verify the waitlist updates within 3 seconds