Space Surfer : Space Manager

Developer: Kay Kayale

Developer Submitted: 03/07/2024

Peer Reviewer: Carine Gordillo

Review Completed :03/11/2024

Major Positives

- Ensure that the user must be a specific role
    - the designs have a note that sate only managers of companies and facilities are able to access these operations
- Includes some important checks
    - Checks such as file type and size for images and character limits for space IDs, are included. These checks are essential for preventing common input-related vulnerabilities.
- Clear success feedback to user
    - The system provides clear feedback to users upon successful completion of an action. This clarity enhances the user experience by confirming that their actions have been successful.
- SpaceInfo object
    - Helps as a standardized data structure to keep the data on the space. This also helps with maintainability and readability.

Major Negatives

- Undefined objects being used
    - The diagrams lack clarity on the origin and attributes of some objects, leading to some confusion about the data flow and usage.
- Redefined SqlCommandBuilder Responsibilities
    - this is not how our design to execute sql command from a service works and it is a violation of single responsibility principle anyway
- Misuse of SqlCommands
    - In the Modify Space it calls the SqlCommand Builder to create an insert command when it should be creating an update command
- Non Efficient SpaceInfo attributes
    - In Modify space diagram it states that space info would contain email, firstName and lastName attribute. I don’t see this as efficient, I believe this might be an accident but the space info object should most likely contain the spaceID, timeLimit, and another attribute that lets us know which floor plan it belongs to

Unmet Requirements

Error after 5 seconds – no error message for time outs

Some of the designs indicate a timeout error but the diagrams do not show any indication of an error occurring anywhere during the sequence. We indicate we would only consider a timeout error if the operation took more than 5 seconds with an error message “System timeout error”, if the operation took between 3.1 – 5 seconds we would only consider it a warning.

No sequence for Deleting a floor plan

Our BRD states that in the modification functionality, a floor plan could be deleted but there isn’t any diagram showing that sequence of deleting a floor plan. If we can delete it, we should also delete the list of reservable spaces.

No checks for current reservations on Seats

While modifying a seat/floor plan we indicate our system must first check if there are any reservations for any of the seats in that floor plan. In the diagram there is no indication of any logic that checks for that.

No checks for Floor plan before allowing Managers to create a list of reservable spaces

The system currently lacks a mechanism to verify that a valid floor plan image has been uploaded before managers can add reservable spaces. This can lead to data inconsistencies and operational issues if space reservations are attempted without an associated floor plan.

No checks for how many reservable spaces a floor plan has

There's no validation to enforce the maximum limit of 50 reservable spaces per floor plan, as outlined in the BRD. Without this check users might unintentionally (or intentionally try to) exceed this limit and might lead to issues with the system performance.

Various Failure Scenarios

The current diagrams do not address various failure scenarios, such as data store connectivity issues, trying to upload reservable spaces without uploading a floor plan, trying to delete/modify a floor plan or a reservable space that has a pending reservation and more.

Design Recommendations

1. Execution once all information is received:
   Your sequence diagrams indicate that each process of creating one complete "space" (image, seats, times available) is done through different processes and forms. This would also mean that there are many calls and roundtrips to our backend and database that could most likely be made with one call rather than multiple. I would suggest making the space creator one form and submit the request to insert that that data into the database once all the valid information needed is received.

   Pros

   - Less runs to the database
   - Less risk of partial data – we either want a complete "space" or nothing. Without either component (image or list of reservable spaces) our space wouldn't really be functional anyway
   - Could check for duplicate spaceIDs within a space within on sequence before sending it into the database.

   Cons

   - If one input is invalid it would prevent the others from being submitted, you would need to be more specific in your error messages.

2. Inserting spaces one at a time:
   Your sequence diagram indicates you are adding spaces one at a time and each time you are going through the process of adding the seat into the database one by one. Similarly to the previous Recommendation, that is many unnecessary calls and checks through our backend and there could be a call for duplicate values which are invalid. I suggest allowing the manager to add a space one by one but to a dictionary rather than directly to a database as spacesDictionary< string spaceID, int timeLimit>.
   Pros
   - By nature, keys of dictionaries must be unique so there would not be any duplicates in that single sequence of inserting reservable spaces.
   - If there are duplicates added within that sequence of adding reservable spaces, you can inform the user in the front end rather than wasting a call to the backend.
   - Average time complexity for a dictionary search and delete is O(1) which again, is faster than a call to the database
   - You can give the manager an option to remove a seat just added within that sequence and easily remove it from the dictionary in O(1) time
   - You would be able to have one call to the backend service rather than 50 (max per floor plan) which would reduce risk of errors.

Cons

- Might have to redesign or add methods to handle the batch operation
- Error handling would become more complex because you would need to validate the success of every item in the dictionary
- Risk that error could occur hallway into inserting the items in the dictionary and you would need proper recovery methods, possibly return what items were a failure to insert

3. Maximizing the use of a manager (dependency injection and data validation):
   Use dependency injection when you are using the service from the manger layer. This helps with loose coupling, single responsibility principle and the injected services are easier to test. The manager that contains the service should also be the one that enforces the business rules checks, for example, many of our names should not be shorter than 2 characters, this is a rule that would be implied by our manager of that service.

   Pros

   - If you need to adjust the service, it would be easier to update or switch implementations.
   - Having a manager layer apply business rules provides more extensibility and consistency in case we were to ever switch to a different type of application, the service would still check for the same things
   - Makes unit testing more straight forward

   Cons
   - Can increase complexity for people who are initially understanding how do use dependency injection
   - Requires more initial setup when dealing with dependency injection and can be time consuming.

4. Role based access checks:
   There is not indication of where or when the system would check the user's role for our role based access control to ensure regular users are not able to have access to this feature, especially since this feature should only be available to mangers. I suggest discussing with the team about having a service that checks the role of this user. We could then use that service in me manager layer.
   Pros
   - Increased security by making sure users aren't given access to methods they aren't supposed to have
   - Debugging RBAC issues would be easier because the operations that check RBAC and give the permission would all be in one class.

- Having a class that provides the service allows for easier maintainability in case new requirements or roles are created.
- less redundancy in the implementation to check the role which would also lead to less potential repeated errors and maintainability.

Cons

- Could cause more latency because we will have an additional layer of role checks for each method.

5. More clear error message
   (in system)
   The Error Message in the response object is not capturing the actions of the operation and is very generic. Having generic messages could make it difficult to figure out where and why the error initially occurred. I would suggest appending the error message with which function the failure began and why.
   (to user)
   The error message displayed to the user should follow the error messages we discussed in the BRD so that the issue is clear.
   Pros
   - When you have a detailed error message, it enhances troubleshooting and help identify the issue more quickly and save time.
   - Solving bug much quicker will lead to a better user experience.

   Cons

   - Could cause performance overhead because of how big the message might be depending on the operation (how many methods it must go through)
   - Could maybe possibly lead to security risks if a user knows the inner system workings

6. Reevaluation of Service -> Data Access
   Some of the designs use functionalities from the wrong classes and restructure the design of our data access sequence. As I mentioned in the negatives, the SqlCommand Builder should not have access to the DAO and only returns a Sql Command, it will never execute because that violates single responsibility principle. I would suggest to review the sequence and methods in our data access diagrams to understand by to integrate them when designing your own feature.
   Pros
   - As in our original design, our sqlCommand builder class is a helper class that helps prevent sql injection

- By not making the sqlCommand builder. Execute the sqlCommnd itself we help maintain the single responsibility of building and returning the desired sql command.

Cons

- Redundant commands
- Possible error in sql command builder would not be known since it doesn't return a response.

Test Recommendations

Space Creator

1. Invalid image : not a Jpeg or png, over .5 MB – makes sure the system correctly rejects incorrectly formatted images and only compatible images are stored.
2. Invalid File Name : invalid characters, less than 2 characters - Validates the system's ability to enforce naming conventions
3. Inserting into DB successful but uploading the image to the S3 bucket was a failure – test the systems error handling and decision whether it will reject the whole process or keep what has been done
4. Company/Facility that has multiple floor plans - checks the system's capacity to handle multiple floor plans per entity without data overlap or integrity issues
5. Inserting floor plans with duplicate names within one facility - Ensures the system prevents name conflicts within a single facility
6. Inserting floor plans with duplicate names in 2 different facilities - ensures that the system allows for the reuse of floor plan names across different facilities

List of Reservable Spaces

1. A space with a time limit under 1 hour or exceeding 12 hours. – make sure the system enforces the boundaries as discussed in the brd
2. Duplicate space name but within two different floor plans for the same company/facility- Validates the system's ability to enforce naming conventions, space ID must be unique across the company/facility
3. Trying to have a list of 0 or null reservable spaces - Tests the system's response to attempts at creating empty space lists
4. Upload a combined number more than 50 reservable spaces on more than one floor plan. - Ensures that system-wide limits on reservable spaces are enforced
5. Trying to upload a list of reservable spaces before a floor plan is submitted. -Tests that spaces cannot be reserved without a corresponding floor plan, as stated in our BRD

Modify Reservable Space

1. Modify/delete a space with reservation that has not passed - Tests that spaces cannot be modified if they still have pending reservation, as stated in our BRD
2. Replace a floor plan image in a company/ facility that has a space with one or more reservations that have not passed. - Tests that floor plans cannot be modified if they have a reservable space that has one or more pending reservation, as stated in our BRD
3. Delete a floor plan image and make sure the list of spaces is also deleted – our BRD states that once a floor plan image is deleted, the list of reservable spaces will also be deleted
4. Deleting a floor plan image from a company that has more than one floor plan registered.

- Ensures the system handles deletions appropriately in complex scenarios

Generic

1. Invalid input types – verifies the system's ability to give an error response with invalid inputs and datatypes depending on the requirement
2. Inputs with just white spaces, null values, special characters, less than 2 characters and exceeding a character limit - other different types of valid inputs that can occur with strings