Peer Review

Feature Name: Service Management
Developer Name: Anh Huynh
Date Developer Submitted: 3/5/23
Reviewer Name: David Chan
Date Review Completed: 3/8/23

Issued by:

Algorithmic Alchemist

Team Lead

Sierra Harris

Team Members

Abhay Solanki
Anh Huynh
Aster Lee
David Chan
Sierra Harris

Github: https://github.com/abhay772/491B

Version History

Version #	Date	Reason for Change
Version 1	3/8/2023	Original Document

Table of Contents

Major Positives

- The design shows the type of HTTP request that is required and the expected HTTP response.
- It shows which calls will be asynchronous and which ones will not be.

Major Negatives

- The current design does not have any front end designs which means developing the front end will be left vague to the developers.
- The overall design is vague with bare minimum details.
- The design is more similar to a High Level Design showing only the bare minimum flow between layers.
- The design system offers no input validation for the client side or on the server side which can lead to invalid inputs or errors causing system failure.
- The design is completely missing the entry point / controller layer and shows the UI interacting directly to the manager layer meaning that the manager layer is the entry point in this design.
- It is not showing clear and distinct responsibility of each layer.
    - The UI Layer should be responsible for getting user input, doing client side validation, and updating the view to the user.
    - The Entry Point Layer which should be the controller should be the connection between frontend and backend.
    - The Manager Layer should be handling the business rules and validation checks for the backend.
    - The Service Layer is supposed to be handling a single unique responsibility.
    - The Data Access Layer should be handling all the interactions with the database.

- By looking at this design, A developer would not know what the function is supposed to do or what are the requirements. It does not further assist in explaining what should be coded any better than the BRD.
    - There are no details on how the user will be searching and showing the services that they will be shown.
    - The user would also be able to be shown a service already used by the user and they would be able to request to add a service already used.
- The design is not showing the requirements that need to be met.
    - The business rules are not being enforced in the sequence diagrams.
    - The preconditions are not being shown as required so it leaves open the user as anybody including unauthorized users.
    - The success cases are also not being enforced by the design or explain how the functional requirements are going to work which can lead to a wide range of interpretation by developers which could lead to a different function.
    - There is also no logging for successful action, a user attempting an unauthorized action, or in the case of a system failure.
- There are no failure cases being shown in the design.
    - If any failure happens in the system there would be no way of dealing with it in the current design. There is no error handling or security.
- The rating sequence diagram is not using the database layer but the function should be using the database to update the service rating in a persistent database.
    - Without the database there would be nowhere to store the review or to update existing reviews.
- There is also no data access object layer so each layer that calls the database has to do it directly which is not ideal as it will require it to have duplicate code. It will also be adding multiple responsibilities to a lot of functions and not following SOLID principles.

Unmet Requirements

Service Management Overall Missing Requirements

- The design shows nothing checking and enforcing that the user has to be a property manager even though that is a requirement.
    - There are no checks in the system design to reject a non authorized user or even a user that is a service provider from using this function.
    - There are also no checks to make sure the user is even logged in.

Service Request Missing Requirements

- There is no design showing that it is checking if the user is a property manager with a known address.
- It does not show that the user is allowed to make a request for services provided by service provider users.
- There are no checks if the user has successfully logged into the website and selected the service management feature.
- The design does not show a way to display services and service provider lists.
- There is no built in notification system for when a user requests a service so the service provider user will not be notified when they are getting requested.
- It does not provide the functional ability to leave additional comments when requesting a service.
- There is no update to the user service list to show a pending service request.
- The design has no mention of a search bar which should be required to complete the search function within 5 seconds.
- There are no notifications being sent to service providers being shown in the design flow which also should be sent within 5 seconds.
- The system does not restrict the completion time of updating the service data to being under 5 seconds.
- Additional comments added to a request are not restricted to being under 150 characters.
- There should be a restriction that only 5 services should be on display on the user screen at a time so only 5 services should be sent to the front end at a time.

- The design does not show specifically that the user chooses a service from a list of services stored in the database and then sends the request to the service provider which then sets the user's service request as pending.
- There are no failure cases in the design for when:
    - The service provider does not receive the request from a property manager
    - The service provider does receive the request but the information received is wrong
    - The search bar did not complete the search within 5 seconds.
    - It took more than 5 seconds for the property manager to get a notification
    - It took more than 5 seconds for the service provider to get a notification
    - Updating service data should take more than 5 seconds.
    - Comment text was more than 150 characters
    - More than 5 services were displayed per page
    - The users services list is not updated with the new pending service

Frequency Change Missing Requirements

- It does not allow the user to cancel a service that they already have attached to their account.
- There are no checks if the user has successfully logged into the website and selected the service management feature.
- It does not have a way to display the users current services.
- The design also does not notify the property manager of any changes in the service request and which user is the one requesting the change.
- There is no system in place to set a pending status after an update request.
- There is no design for a search bar in the Frequency Change design.
- The service provider is not being notified within 5 seconds.
- The completion time to update service data should be limited to 3 seconds.
- There is no design for a comment text box attached to the change request that should also be limited to 250 characters.

- There are no failure cases in the design for when:
    - User chooses a service and the system does not update their services with a pending service change request status.
    - The system takes longer than 5 seconds to notify service provider, the systems send an “Unsuccessful Service Change Request” error message
    - User service change update takes longer than 3 seconds, the system send an “Unable to update” error message

Rating Missing Requirements

- There is no design showing that it is checking if the user is a property manager with a known address.
- Since there is no interaction with the database there is no way to save the user input for a rating so the design does not allow the user to rate a completed or closed service. The rating would also not be saved within 5 seconds.
- There is no check if the user has successfully logged in and selected the rating feature from the homepage.
- The design does not show that it is displaying the rating as a likert scale
- There is no search bar for rating along with a time restriction of 5 seconds.
- The design does not show a way to notify the property manager and service provider with a time restriction of being completed within 5 seconds.
- It has no design implemented to add a comment with the rating that is also limited to 240 words.
- There are no failure cases in the design for when:
    - The user was not able to rate a service
    - A likert scale was not displayed
    - User rating was not stored in a database
    - Property Managers were not able to enter comments on the service provider.
    - The search bar did not complete the search within 5 seconds.
    - Property manager did not get notified in 5 seconds
    - Service provider did not get notified in 5 seconds
    - Updating service data took more than 5 seconds.
    - Comment text was more than 240 words

Design Recommendations

The overall design should be much more detailed in the flow and how each layer should have an unique function and reason behind each layer. Another way to improve the design is to closely follow the requirements first and then build up from there. For example a common mistake made was not following the timing restrictions made by the business rules. This resulted in a lot of times where the design needed to set timers to asynchronous calls to be able to cancel a function if it took too long. I would make a task call and a second task with a 5 second delay then use “completedTask = await Task.WhenAny” to check which task finished first by checking if the completedTask equals the first task call. I also recommend that after business rules are all satisfied then to focus on making sure the precondition is being met by checking the user’s IPrincipal file to see what user they are. This allows the system to verify who the user is and what access rights they have. This also could be reused in other parts of the feature to verify part of the preconditions.

A major flaw seen in the design is not setting a clear distinction between layers as each layer currently does not have its own responsibility. I would set up the UI/ Client side to do basic validation and for user interactions. The controller / entry point layer should be the firewall and also contain minimal validation to then send it to the manager. The manager should be the business rules and requirements so any business rules like timing requirements or validation should be handled in detail in this layer. The service layer should be handling a specific and unique function. An example of this would be in frequency change function the service layer should be the one that checks if the selected frequency change is valid and then it should be calling the data access object which would then access data in the database to be returned and processed by the service layer.

It would also be useful if there was a data access layer responsible for handling all interactions with the data storage to better follow SOLID principles. It would also be useful to put what the data access object will be doing to the database which could be a short description or a SQL command.

Including failure cases is also extremely important so when coding the system the programmers can know what to do if a failure case does happen. I also recommend logging every time a failure case happens as that could be useful analytical data.

Test Recommendations

Tests that I would recommend would be to test all of the time restrictions to check if it's true by setting a stopwatch and checking if it is under 5 seconds with “Assert.IsTrue(stopwatch.ElapsedMilliseconds >= 5000);”. After that I would recommend starting with adding validation tests. For example if a comment text has a limit of 150 characters then send in 151 characters. Another test would be validating if the user has the correct role to be using this feature so I would create an expected role principal object then create another principal object that has a valid role and test with principal.IsInRole() and then assert it is true. It could also be used to check if the user does not have a valid role if the second object is created with a nonvalid role and then assert is false.

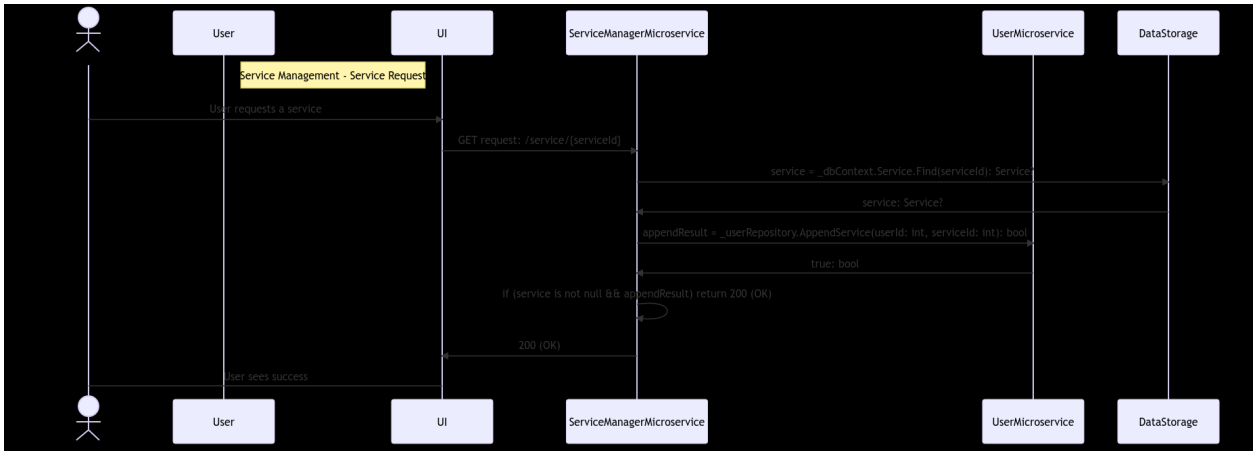
I would then recommend using NightWatch to check end to end testing for the front end. I would make 1 end to end test for each of the Success cases and failure cases. An example of that would be to make a test that automatically selects a service and then clicks a rating on the service. First you would set NightWatch.js to navigate to the service and then use browser.click on the button for rating. Then use browser.setValue to fill out the rating form. Then to check if its successful test with browser.assert to check popup status.

Ahn's Service Management Design

https://github.com/abhay772/491B/blob/main/Designs/ServiceManagement%20LLD.pdf

Use Case ID:	**42**
Use Case Name:	**Service Management - Service Request**
Actors:	- Property Manager
Business Rules:	- User must be a property manager with a known address.
Description:	- This feature will allow the user to requesting a service from a service provider
Preconditions:	- The user has successfully logged in to the website, and selected the Service Management feature from the homepage.
Service Request Functional	- Display services and service providers list - Allow the user to request services - Notifies service provider of request - Allow users to enter additional comments
Postconditions:	- A service request is created and sent, waiting for the service provider's response. - The user service list is updated to contain the pending confirmation service request
Nonfunctional	- The search bar should complete the search within 5 seconds. - Notification to property manager should be sent within 5 seconds - Notification to service provider should be sent within 5 seconds - Updating service data should take no more than 5 seconds. - Comment text allows only 150 characters - Only 5 services should be displayed on the view
Success Cases:	- User chooses a service and the system updates the service request of the service provider. - The system updates the users services with the pending service
Failure Cases:	- The service provider does not receive the request from a property manager - The service provider does receive the request but the information received is wrong - The search bar did not complete the search within 5 seconds. - It took more than 5 seconds for the property manager to get a notification - It took more than 5 seconds for the service provider to get a notification

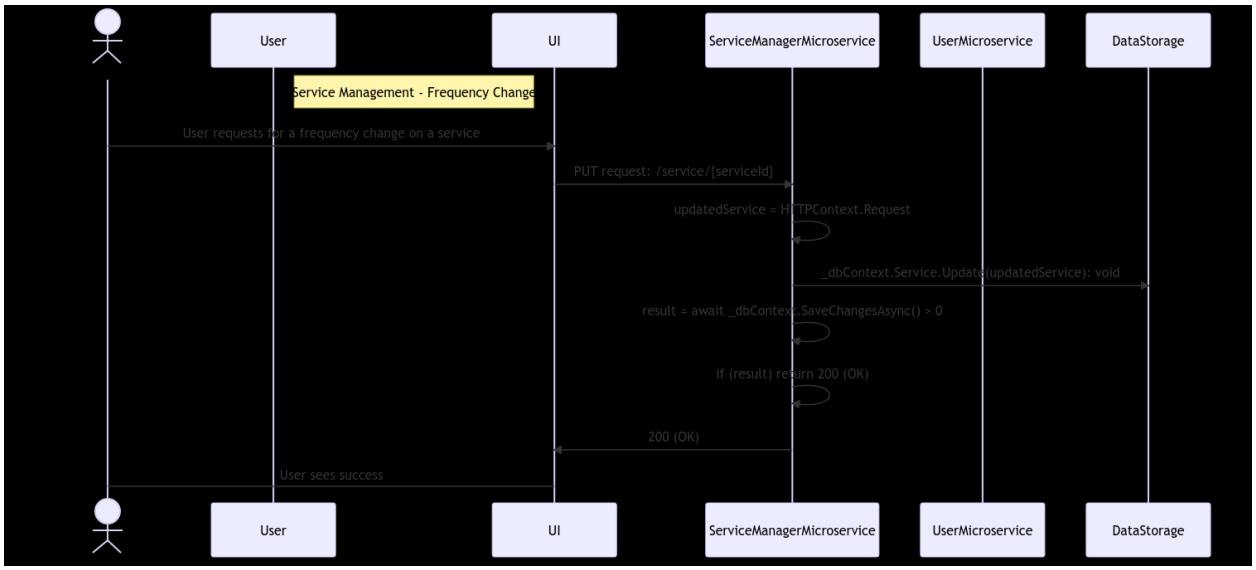
	- Updating service data should take more than 5 seconds. - Comment text was more than 150 characters - More than 5 services were displayed per page - The users services list is not updated with the new pending service




```
sequenceDiagram
    actor U as User
    Note right of User: Service Management - Service Request
    participant UI
    participant SMM as ServiceManagerMicroservice
    participant UM as UserMicroservice
    participant DS as DataStorage
    U->>UI: User requests a service
    UI->>SMM: GET request: /service/{serviceId}
    SMM->>DS: service = _dbContext.Service.Find(serviceId): Service?
    DS->>SMM: service: Service?
    SMM->>UM: appendResult = _userRepository.AppendService(userId: int,
serviceId: int): bool
    UM->>SMM: true: bool
    SMM->>SMM: if (service is not null && appendResult) return 200 (OK)
    SMM->>UI: 200 (OK)
    UI->>U: User sees success
```

Use Case ID:	**43**
Use Case Name:	**Service Management - Frequency Change**
Actors:	- Property Manager

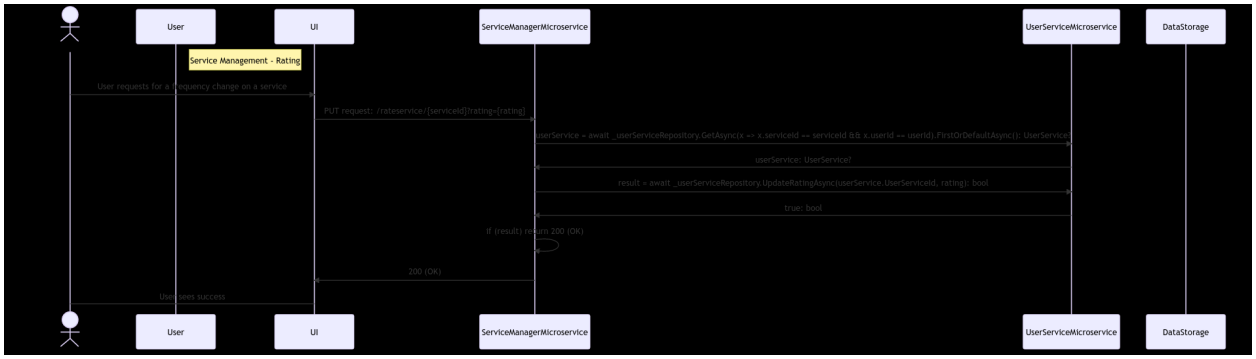
Business Rules:	- User should be able to request a frequency change of their services
Description:	- This feature will allow the user to manage all their services with service providers such as canceling, or changing the frequency of service.
Preconditions:	- The user has successfully logged in to the website, was directed to the homepage and selected the Service Management feature from the homepage.
Functional	- Display the users current services - Allow the user to Cancel or change the frequency of an service - Notify the property manager of service changes request - Update service database with a pending status for the users service change
Postconditions:	- The service change request is sent to the service provider and the service rating is uploaded.
Nonfunctional	- The search bar should complete the search within 5 seconds. - Notification to service provider should be within 5 seconds - Updating service data should take within 3 seconds. - Comment text box should only allow 250 characters
Success Cases:	- User navigates to the service they want to change and the service change request is sent to the service provider. - System updates user services with pending service change requests status.
Failure Cases:	- User chooses a service and the system does not update their services with a pending service change request status. - The system takes longer than 5 seconds to notify service provider, the systems send an “Unsuccessful Service Change Request” error message - User service change update takes longer than 3 seconds, the system send an “Unable to update” error message




```
sequenceDiagram
    actor U as User
    Note right of User: Service Management - Frequency Change
    participant UI
    participant SMM as ServiceManagerMicroservice
    participant UM as UserMicroservice
    participant DS as DataStorage
    U->>UI: User requests for a frequency change on a service
    UI->>SMM: PUT request: /service/{serviceId}
    SMM->>SMM: updatedService = HTTPContext.Request
    SMM->>DS: _dbContext.Service.Update(updatedService): void
    SMM->>SMM: result = await _dbContext.SaveChangesAsync() > 0
    SMM->>SMM: if (result) return 200 (OK)
    SMM->>UI: 200 (OK)
    UI->U: User sees success
```

Use Case ID:	**44**
Use Case Name:	**Service Management - Rating**
Actors:	- Property Manager
Business Rules:	- User must be a property manager with a known address.
Description:	- This feature will allow the user to rate a completed or closed service
Preconditions:	- The user has successfully logged in to the website, was directed to the homepage and selected the Rating feature from the

	homepage.
Functional	- Allow the user rate a service - Display a likert scale - Record user rating in a database - Property Managers can enter comments on the service provider.
Postconditions:	- The service request or change is sent to the service provider and the service rating is uploaded.
Nonfunctional	- The search bar should complete the search within 5 seconds. - Database should record user rating within 5 seconds. - Notification to property manager should be within 5 seconds - Notification to service provider should be within 5 seconds - Updating service data should take within 5 seconds. - Comment is limited to 240 words.
Success Cases:	- User is successfully able to rate the service that was performed. System updates the service provider's rating.
Failure Cases:	- The user was not able to rate a service - A likert scale was not displayed - User rating was not stored in a database - Property Managers were not able to enter comments on the service provider. - The search bar did not complete the search within 5 seconds. - Property manager did not get notified in 5 seconds - Service provider did not get notified in 5 seconds - Updating service data took more than 5 seconds. - Comment text was more than 240 words




```
sequenceDiagram
    actor U as User
    Note right of User: Service Management - Rating
    participant UI
    participant SMM as ServiceManagerMicroservice
    participant USM as UserServiceMicroservice
```

```
    participant DS as DataStorage
    U->>UI: User requests for a frequency change on a service
    UI->>SMM: PUT request: /rateservice/{serviceId}?rating={rating}
    SMM->>USM: userService = await _userServiceRepository.GetAsync(x =>
x.serviceId == serviceId && x.userId == userId).FirstOrDefaultAsync():
UserService?
    USM->>SMM: userService: UserService?
    SMM->>USM: result = await
_userServiceRepository.UpdateRatingAsync(userService.UserServiceId,
rating): bool
    USM->>SMM: true: bool
    SMM->>SMM: if (result) return 200 (OK)
    SMM->>UI: 200 (OK)
    UI->U: User sees success
```