Feature Name: Account System
Developer Name: Kevin Dinh
Date Developer Submitted: 4/16/2023
Reviewer Name: Jett Sonoda
Date Review Completed: 4/16/2023

Major Positives:

- Almost all major requirements are accounted for
    - All found in the BRD

Major Negatives:

- No input validation or sanitation, resulting in a huge vulnerability to the system from SQL Injections
- No authorization checks on the access token, resulting in a huge vulnerability to the system from query parameters
- Business rules not accounted for, resulting in an unfulfilled Business Requirements Document
- Requirement input/success criteria is not met from the Requirements, resulting in an unfulfilled Business Requirements Document
- SELECT * SQL statements are not scalable, datastore table columns may change in the future that would result in potentially ruining the logic in higher layers when handling data

Unmet requirements:

Reset Password

- User is not logged in (unauthenticated) trying to reset password
- Password stored is not identical to input password
- New password adhering to the business rules

Settings (Account Modification)

- Changing/updating the first name
- Changing/updating the last name

Rental History View

- Rental history is not in chronological order

Change Rental History View to Calendar Format

- All

High-Level Design Diagrams Recommendations

Change Password: Low Priority

- Does not follow the same 4-tier layered architecture as the rest of the diagrams since “Login Service” is in between the view and manager layer. ***Please remove the “Login Service” to keep this diagram consistent with the other diagrams and 4-tier layered architecture or explain this Login Service if you think this is appropriate to keep.***
- Text/check bubbles are doing more than one task. There is clearly more than one process occurring inside of this check. ***Please include the entire process for the checks currently inside of “Login Service”. For example, include the process for checking to make sure the user entered in a valid password. Then check the datastore to ensure that it is the previous password. Then show the process for sending the OTP and confirming it with the datastore. Each process should be in its own section/bubble***
    - Example: “User is prompted to enter password then OTP”. This should be broken up into at least two checks.

Change Account Settings: Low Priority

- After OTP is sent to the user, there should be a process from the view layer where the user is prompted to enter in the OTP and it is sent to the other layers for verification. Then the process may continue. ***Please include this input from the user.***

Search Rental History: Low Priority

- “Load Booking History View” moving into another process in the view is confusing since that isn’t the process to view the booking history. ***Please remove the “Load Booking History View” and either note that the user is already in the booking history view to begin this process or note that the “Load Booking History View” is an abstracted process for the other requirement “Get Booking History”.***

Rental Cancellation: Low Priority

- ***Same feedback for the previous bullet point (Search Rental History).***

Overall Low-Level Design Recommendations:

- DataAccess and Datastore layers need to be specified. What Datastore specifically are we accessing? Especially when the relational tables display that we are accessing more than one datastore. This is crucial when other developers will try to understand these designs with no background knowledge
- Make sure that there is authorization (for certain requirements) at the manager layer to ensure that the request was not from query parameters and that it is a real authenticated user instead. This could result in unauthenticated users accessing parts of the app, becoming a huge security threat.
- Include proper input validation to ensure that the input satisfies the business rules. Specifically, the password and email/username have specific rules that our system is required to satisfy. Reference the BRD User Management Business Rules on page 17
- Change all SELECT * SQL statements to SELECT (column_name1, column_name2, …) to be more specific and ensure that you know exactly what data you are looking for and handling.
- When handling failure cases, there should be no payload (as in passing data when something has failed). Instead, there should be error messages to know exactly where and when the process has failed which is extremely important to debugging for developers. For the user experience, it is important to handle these errors differently when displaying them to the user because it may be something that they can fix or an issue that is on our end/unfixable by the user.

Specific Low-Level Design Diagram Recommendations:

Change Password: **Critical Priority**

- *As mentioned in previous sections*, there is no validation of the new password meeting the password requirements (minimum character length and valid characters). ***Please include at the manager layer using the ValidationService.***
- There is no validation of the new password being identical to the old password. This validation needs to be included as it is part of a requirement in the BRD. ***Please include at the manager layer once the old password was received from the GetPasswordData(string email) was called (step 9 of Pt. 2 diagram)***
- Failure Diagrams:
    - Failure case of OTP service being unable to create an OTP not accounted for. An error may occur here and these diagrams don't account for it, resulting in a critical unhandled error for the user. This error should be represented and handled. ***Please include a diagram and error message to handle this error at the manager layer along with the error that will display to the user's view.***
    - *As mentioned in previous sections*, there are no error messages being passed from layer to layer when result.IsSuccessful = false. How will the developer know when or where the backend failed?
        - Note on the view when displaying the error to the user, it says "notify either old password is incorrect or the new password was not entered twice correctly". How will you know which error it is without error messages?
        - This should be handled by passing (result.ErrorMessage = "..."): string. ***Please include this for all failure cases once (result.IsSuccessful = false): bool.***
    - Failure Diagram User's new passwords do not match: Since this failure check doesn't require any involvement in the datastore, it should be done before proceeding in the process and advancing through the layers. ***Please move this validation check higher up in the process.***

Get Account Settings: **High Priority**

- *As mentioned in previous sections*, there is no authorization check for the access token at the manager layer.
    - With this check, you will also be passing the userId from manager to service layer in the method signature. ***Please include this Authorization check in the manager layer before proceeding with any service calls.***
- Select statement to the data store is only pulling the UserEmail, this requirement seems like it will need more information than just the email. ***Please include all information that you will want to display to the user about their Account Settings (First name? Last name? etc.)***

- The Payload passed back up from each layer is a List<Dictionary<string, object>>. It might be better to create a DTO object to make the data cleaner to pass through layers. ***Please create a DTO class object in Models that will have any information you plan on showing to the user.***
- Failure Diagrams:
    - Failure Diagram Request is unsuccessful: *As mentioned in previous sections*, there should be no payload and instead an error message being passed back up the layers. How can you pass data if the system failed? ***Please include error messages.***
    - Failure Diagram Response package is empty: Assuming that you change the select statement to be more information than the user email, can nothing in the datastore be null? For example, if they have never provided their first name before and this is null, this will respond with a failure, creating an infinite loop where the user is never able to enter in their first name because it is null in the datastore. *As mentioned in previous sections*, ***please create the DTO object and allow null values for when the response package is empty.***

Change Account Settings (email): **Critical Priority**

- *As mentioned in previous sections*, there is no authorization check for the access token at the manager layer after step 4, step 12, or step 20. Unauthenticated users could attempt to send a request and proceed based on this design. ***Please include authorization checks at each of these steps.***
- This requirement is solely for changing email (different from changing first name or last name), why is the user being asked to enter their first name and last name in the process of changing their email?
    - Note that in the UPDATE SQL statement, it is only updating their email. The first name and last name have no purpose of being passed down in the previous layers. ***Either include the first name and last name in the update statement or create other designs to handle such.***
- The Datastore should be responding to the Abstracted CRUD Repository which then responds to the Data Access layer. Step 25 displays that the Datastore is responding to the Data Access layer. ***Please fix this.***
- How can the user's view be updated when the payload that is passed up from the datastore all the way to the view is null? The view won't be updated with no information. However, there should be a system message to the user that the email change was successful. ***Please include a system message.***
- Failure Diagrams:
    - *As mentioned in previous sections*, the failure case of OTP service being unable to create an OTP not accounted for. ***Please include a failure diagram to handle such errors.***
    - *As mentioned in previous sections*, there are no error messages being passed from layer to layer when result.IsSuccessful = false. How will the developer know when or where the backend failed. ***Please include error messages.***

    - *As mentioned in previous sections*, there are no validation checks for a valid email being entered (minimum character length or valid characters). ***Please review the BRD Business Rules for User Management section and create the validation checks accordingly.***
    - No check for the email provided already existing in the system. ***Please include some sort of check by creating a function that will see if this email already exists in the data store or handle the error of the UPDATE SQL statement accordingly.***

Get Rental History: **High priority**

- *As mentioned in previous sections*, SELECT * is not recommended when calling to the datastore in terms of scalability (table columns might change in the future). ***Please write out the exact columns you will be selecting.***
- *As mentioned in previous sections*, the Payload passed back up from each layer is a List<Dictionary<string, object>>. It might be better to create a DTO object to make the data cleaner to pass through layers. ***Please create an object class in Models to use instead.***
- *As mentioned in previous sections*, there is no authorization check for the access token at the manager layer. ***Please include this check.***
- No mention of the chronological order of bookings displayed. Assuming this is going to be handled at frontend?

Search Rental History: **Critical Priority**

- *As mentioned in previous sections*, there is no authorization check for the access token at the manager layer.
    - With this check, you will also be passing the userId from manager to service layer in the method signature. ***Please include this authorization check as well as retrieving and parsing the access token to pull the userId from.***
- *As mentioned in previous sections*, there is no input validation/sanitation of the keywords the user enters before it reaches the datastore. This could result in malicious SQL injections. ***Please include these checks using Validation Service.***
- *As mentioned in previous sections*, SELECT * is not recommended when calling to the datastore in terms of scalability (table columns might change in the future). ***Please write out the exact columns you will be selecting.***
- *As mentioned in previous sections*, the Payload passed back up from each layer is a List<Dictionary<string, object>>. It might be better to create a DTO object to make the data cleaner to pass through layers. ***Please create an object class in Models to use instead.***
- Failure Diagrams:
    - Failure Diagram Request is unsuccessful: *As mentioned in previous sections*, there should be no payload and instead an error message being passed back up the layers. How can you pass data if the system failed? ***Please include error messages and remove the payload field from these diagrams.***

Booking Cancellation: **Critical Priority**

- Diagram looks unfinished. How can the booking be canceled if the process never reaches the datastore? ***Please include the rest of this process in the diagram or include a reference to where this diagram is completed.***
- Only listingId is being passed down, what happens when a user has multiple bookings with that one listing? Will this process cancel all bookings with this listing or what happens? ***Please include a bookingId or some sort of unique identifier to avoid these edge cases.***
- *As mentioned in previous sections*, there is no authorization check for the access token at the manager layer. ***Please include this check.***

Relational Tables: **High Priority**

- How can you display the rental history in chronological order if the Bookings table contains no dates? ***Please include a date of some sort in order to display the order properly.***

Wireframe: **Medium Priority**

- Canceling bookings suggests that you may cancel more than one at the same time, the backend does not account for a list of listingIds being passed as a parameter in the method signatures. ***Please allow a list of bookings to be canceled in the CancelAppointment method in the backend.***
- Resetting password is only available once the user is authenticated. ***Please include options for when the user is not logged in in order to satisfy the BRD requirements.***
- On-click of “Notification Settings” navigates the user to Booking History instead of the “Booking History” doing so. ***Please move the arrow accordingly.***

Test Recommendations:

Change Password:

- Create test cases for having invalid passwords for character length and invalid characters
- Create a test case where the new password is the same as the old password
- Create test case where the user is unauthenticated trying to reset password
- Create test cases that expect and handle certain errors

Get Account Settings:

- Create test case where there is no access token
- Create test cases where you want to ensure that the first name, last name, and email are displayed to the front end. Make an expected first name and ensure that the datastore matches that
- Create test cases where the first name and last name are null
- Create test cases that expect and handle certain errors

Change Account Settings:

- Create test case where there is no access token
- Create test cases where the user wants to update only their first name or only their last name
- Create test cases where the email does not satisfy the business rules in either the format, character length, or valid characters
- Create a test case where the email that the user requests to change to is already in the datastore saved to another account
- Create test cases that expect and handle certain errors

Get Rental History:

- Create test case where there is no access token
- Create a test case to retrieve the rental history and insert it into a list and ensure it is in chronological order

Search Rental History

- Create test case where there is no access token
- Create a test case where the user attempts to SQL inject using the search key words
- Create test cases that expect and handle certain errors

Booking Cancellation

- Create test cases where the user wants to delete more than one booking at the same time
- Create test case where the user has multiple bookings with a single listing and wants to cancel only one of them
- Create test case where there is no access token
- Create test cases that expect and handle certain errors