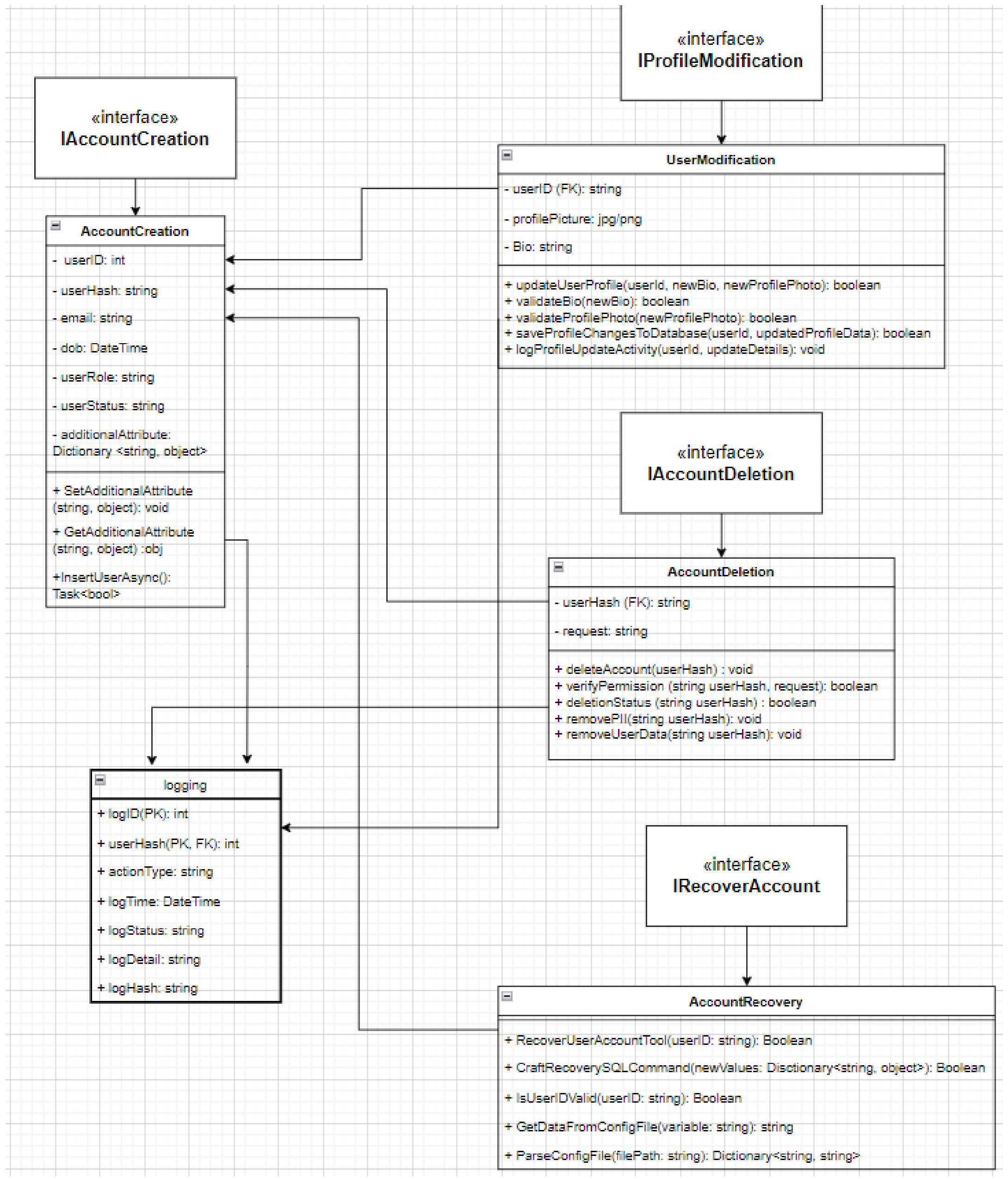
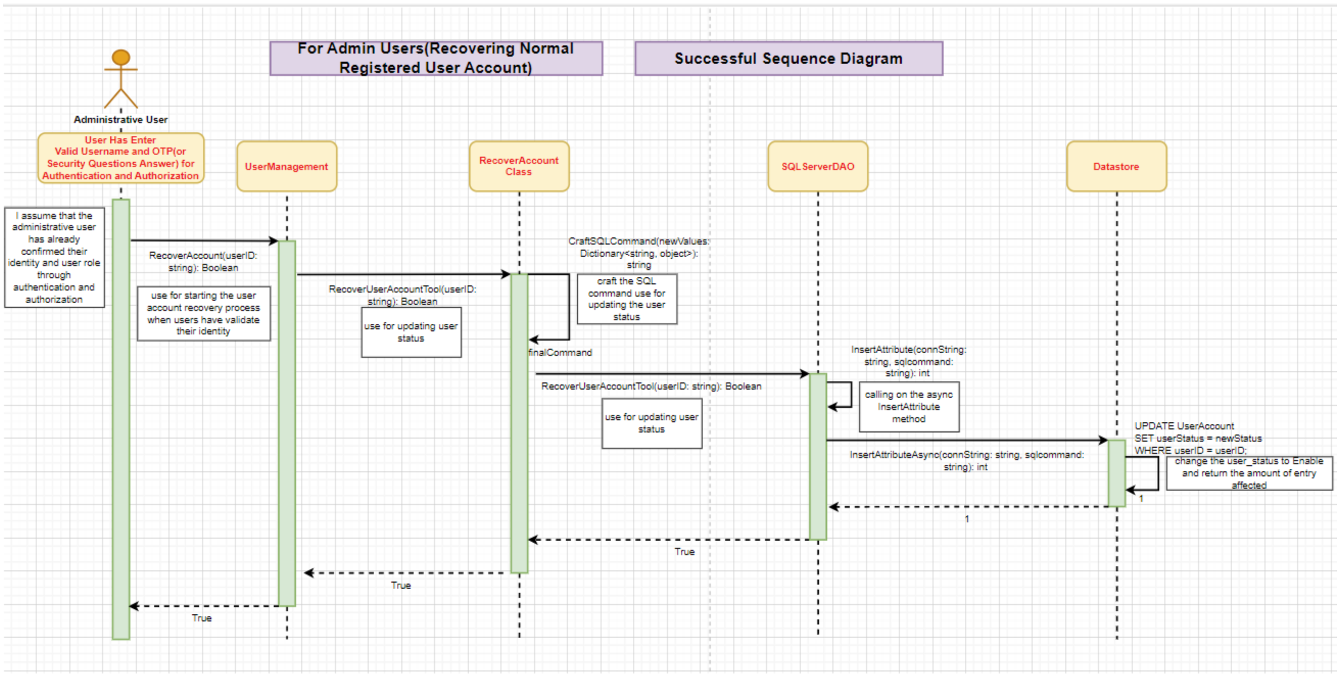
Account Recovery:

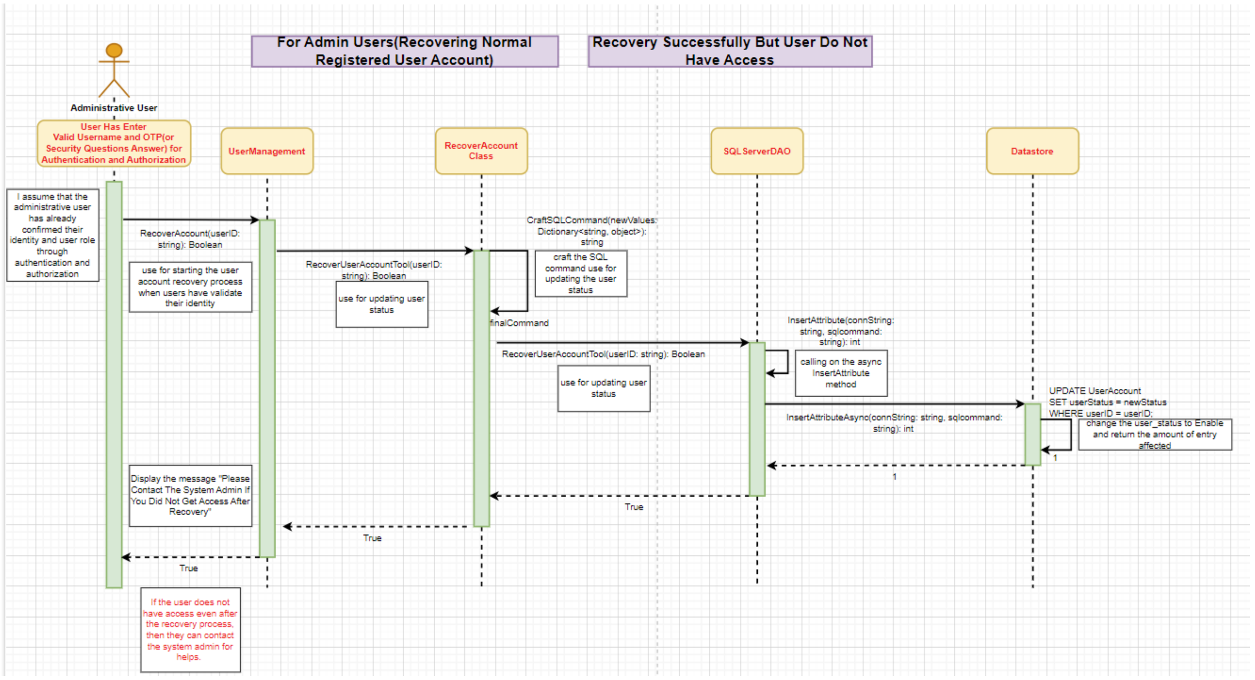
- **Class Diagram:**


«interface» IProfileModification
«interface» IAccountCreation
UserModification
- userID (FK): string
- profilePicture: jpg/png
- Bio: string
+ updateUserProfile(userId, newBio, newProfilePhoto): boolean
+ validateBio(newBio): boolean
+ validateProfilePhoto(newProfilePhoto): boolean
+ saveProfileChangesToDatabase(userId, updatedProfileData): boolean
+ logProfileUpdateActivity(userId, updateDetails): void
AccountCreation
- userID: int
- userHash: string
- email: string
- dob: DateTime
- userRole: string
- userStatus: string
- additionalAttribute: Dictionary <string, object>
+ SetAdditionalAttribute (string, object): void
+ GetAdditionalAttribute (string, object) :obj
+InsertUserAsync(): Task<bool>
«interface» IAccountDeletion
AccountDeletion
- userHash (FK): string
- request: string
+ deleteAccount(userHash) : void
+ verifyPermission (string userHash, request): boolean
+ deletionStatus (string userHash) : boolean
+ removePII(string userHash): void
+ removeUserData(string userHash): void
logging
+ logID(PK): int
+ userHash(PK, FK): int
+ actionType: string
+ logTime: DateTime
+ logStatus: string
+ logDetail: string
+ logHash: string
«interface» IRecoverAccount
AccountRecovery
+ RecoverUserAccountTool(userID: string): Boolean
+ CraftRecoverySQLCommand(newValues: Disctionary<string, object>): Boolean
+ IsUserIDValid(userID: string): Boolean
+ GetDataFromConfigFile(variable: string): string
+ ParseConfigFile(filePath: string): Dictionary<string, string>

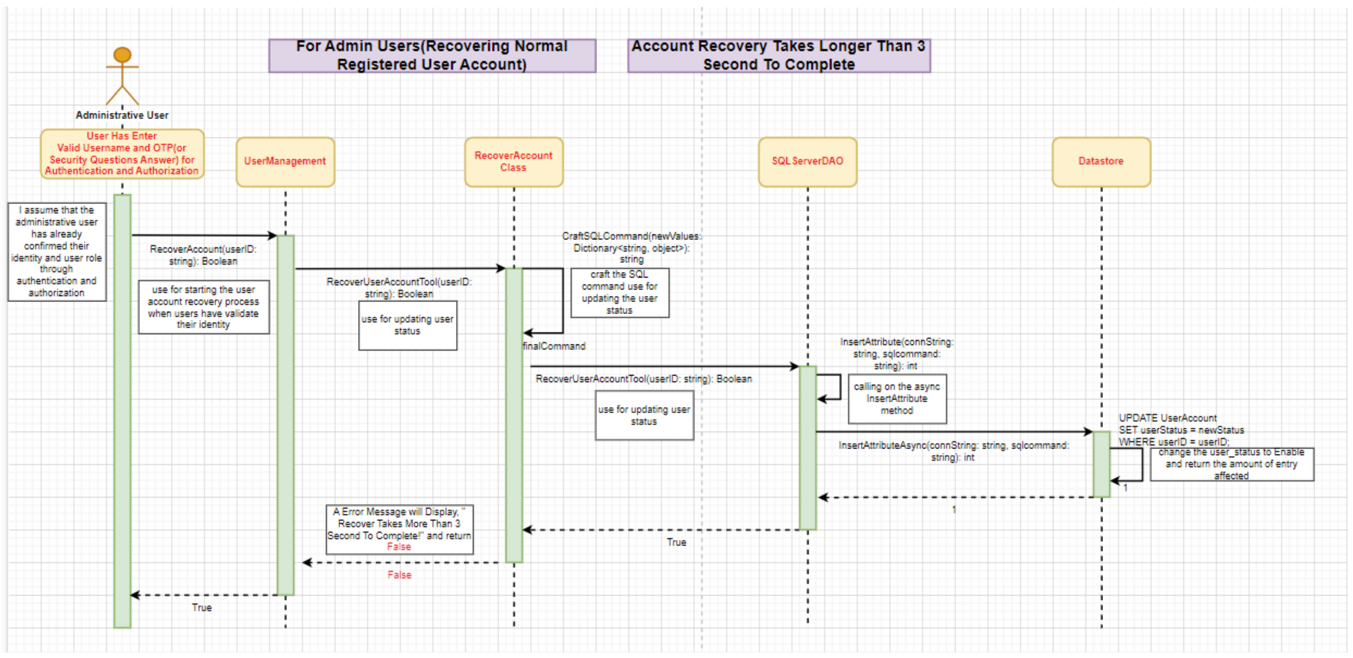
- **Success Sequence Diagram:**


For Admin Users(Recovering Normal Registered User Account)
Successful Sequence Diagram
Administrative User
User Has Enter Valid Username and OTP(or Security Questions Answer) for Authentication and Authorization
UserManagement
RecoverAccount Class
SQL ServerDAO
Datastore
I assume that the administrative user has already confirmed their identity and user role through authentication and authorization
RecoverAccount(userID: string): Boolean
use for starting the user account recovery process when users have validate their identity
RecoverUserAccountTool(userID: string): Boolean
use for updating user status
CraftSQLCommand(newValues: Dictionary<string, object>): string
craft the SQL command use for updating the user status
finalCommand
RecoverUserAccountTool(userID: string): Boolean
use for updating user status
InsertAttribute(connString: string, sqlcommand: string): int
calling on the async InsertAttribute method
InsertAttributeAsync(connString: string, sqlcommand: string): int
UPDATE UserAccount SET userStatus = newStatus WHERE userID = userID;
change the user_status to Enable and return the amount of entry affected
1
1
True
True
True

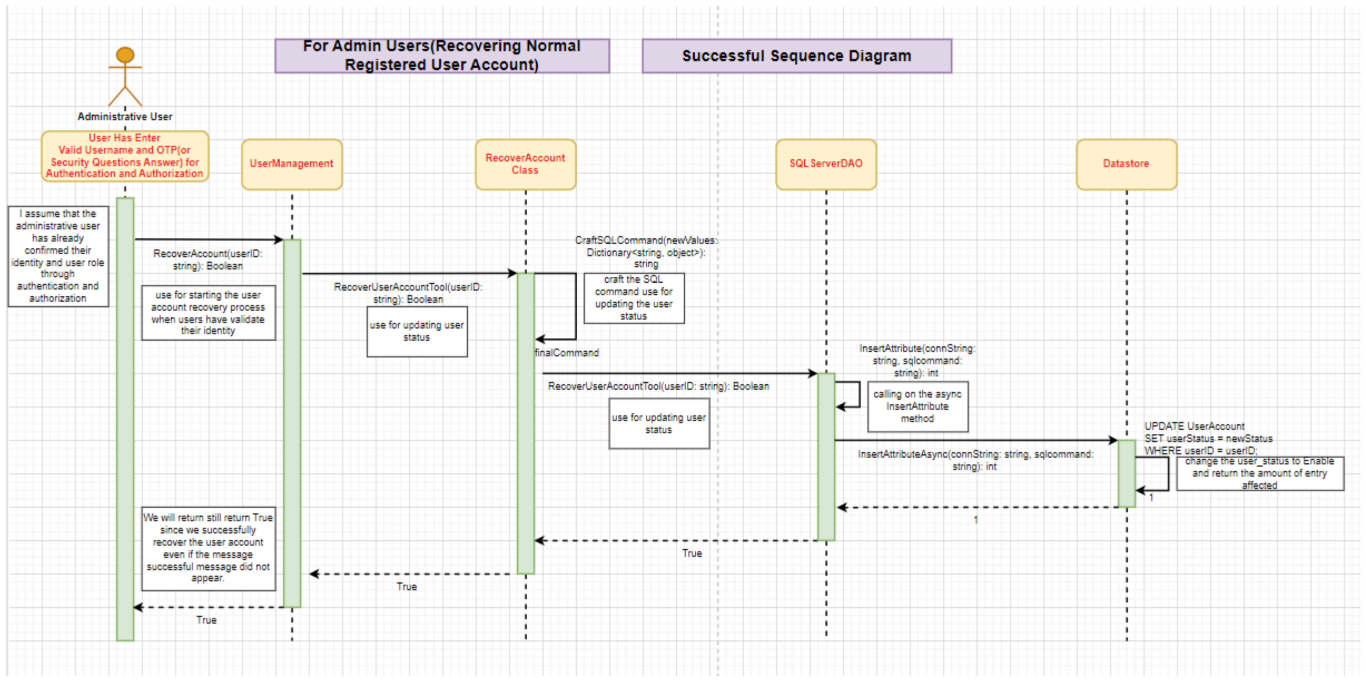
- **Failure Sequence Diagram:**
    - Account Recovered Successfully But User Have No Access


For Admin Users(Recovering Normal Registered User Account)
Recovery Successfully But User Do Not Have Access
Administrative User
User Has Enter Valid Username and OTP(or Security Questions Answer) for Authentication and Authorization
UserManagement
RecoverAccount Class
SQL ServerDAO
Datastore
I assume that the administrative user has already confirmed their identity and user role through authentication and authorization
RecoverAccount(userID: string): Boolean
use for starting the user account recovery process when users have validate their identity
RecoverUserAccountTool(userID: string): Boolean
use for updating user status
CraftSQLCommand(newValues: Dictionary<string, object>): string
craft the SQL command use for updating the user status
finalCommand
RecoverUserAccountTool(userID: string): Boolean
use for updating user status
InsertAttribute(connString: string, sqlcommand: string): int
calling on the async InsertAttribute method
InsertAttributeAsync(connString: string, sqlcommand: string): int
UPDATE UserAccount SET userStatus = newStatus WHERE userID = userID;
change the user_status to Enable and return the amount of entry affected
1
1
True
Display the message "Please Contact The System Admin If You Did Not Get Access After Recovery"
True
True
If the user does not have access even after the recovery process, then they can contact the system admin for helps.

    - Account Recovery Takes Longer Than 3 Seconds to Complete:


For Admin Users(Recovering Normal Registered User Account)
Account Recovery Takes Longer Than 3 Second To Complete
Administrative User
User Has Enter Valid Username and OTP(or Security Questions Answer) for Authentication and Authorization
UserManagement
RecoverAccount Class
SQL ServerDAO
Datastore
I assume that the administrative user has already confirmed their identity and user role through authentication and authorization
RecoverAccount(userID: string): Boolean
use for starting the user account recovery process when users have validate their identity
RecoverUserAccountTool(userID: string): Boolean
use for updating user status
CraftSQLCommand(newValues: Dictionary<string, object>): string
craft the SQL command use for updating the user status
finalCommand
RecoverUserAccountTool(userID: string): Boolean
use for updating user status
InsertAttribute(connString: string, sqlcommand: string): int
calling on the async InsertAttribute method
InsertAttributeAsync(connString: string, sqlcommand: string): int
UPDATE UserAccount SET userStatus = newStatus WHERE userID = userID;
change the user_status to Enable and return the amount of entry affected
1
1
True
A Error Message will Display, " Recover Takes More Than 3 Second To Complete!" and return False
False
True

- The Account Is Successfully Recover But the Message Takes Longer Than 3 Seconds to Display:


For Admin Users(Recovering Normal Registered User Account)
Successful Sequence Diagram
Administrative User
User Has Enter Valid Username and OTP(or Security Questions Answer) for Authentication and Authorization
UserManagement
RecoverAccount Class
SQL ServerDAO
Datastore
I assume that the administrative user has already confirmed their identity and user role through authentication and authorization
RecoverAccount(userID: string): Boolean
use for starting the user account recovery process when users have validate their identity
RecoverUserAccountTool(userID: string): Boolean
use for updating user status
CraftSQLCommand(newValues: Dictionary<string, object>): string
craft the SQL command use for updating the user status
finalCommand
RecoverUserAccountTool(userID: string): Boolean
use for updating user status
InsertAttribute(connString: string, sqlcommand: string): int
calling on the async InsertAttribute method
InsertAttributeAsync(connString: string, sqlcommand: string): int
UPDATE UserAccount SET userStatus = newStatus WHERE userID = userID;
change the user_status to Enable and return the amount of entry affected
1
1
True
We will return still return True since we successfully recover the user account even if the message successful message did not appear.
True
True

Low-Level Design Document for Registration: Account Creation

1. Introduction

1.1 Purpose

We will provide clarification of our design based on the sequence, ER, and Class diagram, and test case validation

1.2 Scope

The scope of this document includes the UserAccount and UserProfile class, sequence diagrams for the creation process, and database schema for storing account and profile data.

2. UserProfile Class Design

2.1 Attributes:- (At Top Class Diagram)

2.2 Methods:- (At Top Class Diagram)

3. Sequence Diagrams for Profile Modification

3.1 Success Scenario:- (At Top Success Outcome Diagram)

3.2 Failure Scenario:- (At Top Failure Outcome Diagram)

4. Database Design

4.1 Profile Table:- (At Top ER Diagram)

4.2 Logging Table:- (At Top ER Diagram)

5. Error Handling and Logging

5.1 Error Handling

- If the email is invalid in format or exists in our database, the transaction of inserting data into the database will be rollbacked.
- If the format of DOB and user's role are invalid, the account creation will stop and return false.
- If the userID and userhash process are not working, it will automatically terminate at the account object creation.
- If any of data insertion to 2 tables are not success the transaction will rollback.

5.2 Logging

- We will log all the success and failure processes.

6. Test Cases
6.1 Unit Tests
- IsValidEmail Tests:
  - Valid Emails: Tests that IsValidEmail returns true for valid email formats, which more than 8 chars, less than 64 chars, contains a-Z,0-9,@,.,-
  - Invalid Emails: Tests that IsValidEmail returns false for invalid email formats or violations of the specified rules.
- IsValidDate Tests:
  - Valid Dates: Tests that IsValidDate returns true for dates that are in the correct format and within the specified date range. Which has less than 10 digits, between 1970-1-1 and the current date.
  - Invalid Dates: Tests that IsValidDate returns false for dates outside the range, in incorrect format, or over the maximum length.
- IsNullString Tests:
  - Null and Empty Strings: Tests that IsNullString returns true for null or empty strings.
  - Non-Empty Strings: Tests that IsNullString returns false for non-empty strings.
- IsValidLength Tests:
  - Valid Length: Tests that IsValidLength returns true for strings within the specified length range.
  - Invalid Length: Tests that IsValidLength returns false for strings outside the length range.
- IsValidDigit Tests:
  - Valid Digits: Tests that IsValidDigit returns true for strings matching the specified pattern.
  - Invalid Digits: Tests that IsValidDigit returns false for strings not matching the pattern.
- IsValidPosition Tests:
  - Valid Position: Tests that IsValidPosition returns true for emails where '@' is correctly positioned. Which is not at the front, the end of the string, and the string has at least 1 @.

    - Invalid Position: Tests that IsValidPosition returns false for emails where '@' is incorrectly positioned.
- GenerateUserID Tests:
    - Valid ID Generation: Tests that GenerateUserID always generates a non-negative, random 10-digit ID in int.
- HashSHA256 and CreateEmailPepperHash Tests:
    - Expected Hash Generation: Tests that CreateEmailPepperHash generates the expected hash for a given email.
- IsValidUserRole Tests:
    - Valid Roles: Tests that IsValidUserRole returns true for valid user roles, which are more than 5 and less than 10 chars, a-Z, 0-9
    - Invalid Roles: Tests that IsValidUserRole returns false for invalid user roles.
- IsValidSecureAnswers test:
    - Valid Answers: will return true if the answer is more than 8 and less than 50 characters, a-Z, 0-9 includes white space.
    - Invalid Roles: Tests that IsValidSecureAnswers returns false for invalid security answers .
- CreateEmailPepperHash test:
    - Valid Hash will have fixed number of characters as 64, a-Z, 0-9.
    - Invalid hash will have less than 64 chars or null.
- GetDataFromConfigFile test:
    - Valid result will be a string that stores the address of the database.
    - Invalid result will be null or wrong address.
- ParseConfigFile test:
    - Valid result open an existing Config file and read all the data from it, separate the key and value by recognize the “:=”.
    - Invalid result is invalid address of Config file, empty file.

6.2 Integration test: testing all methods, classes, dependencies, databases together.

- InsertAccountToDatabase:
    - Valid result is the success of transaction commits to 2 tables, return true
    - Invalid result: return false if any insertion from any table is audit by the database. Transaction will rollback.

Low-Level Design Document for User Modification

1. Introduction

1.1 Purpose

To provide robust and efficient functionality, the purpose of this document is to outline a low-level design for Profile Modification features in our application, describing classes, methods, database designs, error handling, or test cases.

1.2 Scope

The scope of this document includes the UserProfile class, sequence diagrams for the modification process, database schema for storing profile data, logging activity, and defining error handling strategies along with test cases for validation.

2. UserProfile Class Design

2.1 Attributes:- (At Top Class Diagram Under User Modification)

2.2 Methods:- (At Top Class Diagram Under User Modification)

3. Sequence Diagrams for Profile Modification

3.1 Success Scenario:- (At Top Success Outcome Diagram Under User Modification)

3.2 Failure Scenario:- (At Top Failure Outcome Diagram Under User Modification)

4. Database Design

4.1 Profile Table:- (At Top ER Diagram Under User Modification)

4.2 Logging Table:- (At Top ER Diagram Under User Modification)

5. Error Handling and Logging

5.1 Error Handling

- Bio Validation Failure: If the bio does not meet the specified criteria (e.g., length less than or equal to 200 words), the validateBio method will return false, and the update process will terminate, informing the user of the failure.
- Profile Photo Validation Failure: Similar to bio validation, if the new profile photo does not adhere to the format (JPEG/PNG/HEIF) or size restrictions (< 5MB), the validateProfilePhoto method will return false, and the update process will be halted with an appropriate message to the user.
- Database Update Failure: If the update process encounters an issue while saving the data to the database (e.g., connection issues, constraint violations), the saveProfileChangesToDatabase method will return false. The system will log the failure and notify the user that the update was unsuccessful.

5.2 Logging

- Logging is handled by the Logger class which records each significant event during the profile modification process. The following events are logged:
- Successful bio validation.
- Successful profile photo validation.
- Successful database update.
- Bio validation failure.
- Profile photo validation failure.
- Database update failure.
- The log records will include the user ID, the type of event (validation success, validation failure, update success, update failure), and a timestamp. This will help in auditing and troubleshooting the system.

6. Test Cases

6.1 Unit Tests

- UpdateUserProfile_ValidBioAndPhoto_ReturnsTrue: Verifies that the method returns true when valid bio and JPEG photos are provided.
- UpdateUserProfile_InvalidBio_ReturnsFalse: Checks that the method returns false when the bio exceeds the length restriction.

- UpdateUserProfile_ValidBioAndPNGPhoto_ReturnsTrue: Confirms that the method returns true when a valid bio and a PNG photo are provided.
- UpdateUserProfile_ValidBioAndInvalidPNGPhoto_ReturnsFalse: Ensures the method returns false when an incorrect PNG photo header is provided.
- UpdateUserProfile_ValidBioAndHEIFPhoto_ReturnsTrue: Tests that the method returns true for a valid bio and HEIF photo.
- UpdateUserProfile_ValidBioAndInvalidHEIFPhoto_ReturnsFalse: Verifies that the method returns false when an incorrect HEIF photo header is provided.
- UpdateUserProfile_InvalidPhoto_ReturnsFalse: Asserts that the method returns false when the photo data is invalid.
- ValidateBio Method Tests
- ValidateBio_LessThan200Words_ReturnsTrue: Tests that the ValidateBio method returns true when the bio is within the acceptable length.
- ValidateBio_MoreThan200Words_ReturnsFalse: Ensures that the method returns false when the bio exceeds the acceptable length.
- ValidateProfilePhoto Method Tests
- ValidateProfilePhoto_ValidJPEG_ReturnsTrue: Confirms that the method returns true when a valid JPEG photo header is provided.
- ValidateProfilePhoto_ExceedsSizeLimit_ReturnsFalse: Ensures that the method returns false when the photo exceeds the size limit.
- SaveProfileChangesToDatabase Method Tests
- SaveProfileChangesToDatabase_SuccessfulUpdate_ReturnsTrue: Verifies that the method returns true when the database update is successful.
- SaveProfileChangesToDatabase_UnsuccessfulUpdate_ReturnsFalse: Checks that the method returns false when the database update is unsuccessful.

6.2 Integration Tests

- TestCompleteProfileUpdateProcess: This test will simulate a complete profile update operation including validation of input and saving changes to the database. It will assert that the correct methods are called and that the database reflects the new profile data when the operation is successful.
- TestProfileUpdateWithInvalidData: To test the robustness of the system, this test will pass invalid data to the update operation and assert that the system

does not make changes to the database and correctly handles the invalid input.

- TestLoggingDuringProfileUpdate: This test will verify that all steps of the profile update process are logged correctly, especially in failure cases.

Low-Level Design Document for Profile Account Recovery

- **Design Goals:**
    - Security: we want to make sure that account recovery will protect our SQL server by making sure every string input prevents attackers from exploiting the inputs to take control or extract data from our database.
    - Maintainability: we want to ensure that our Account Recovery code is as clean and well-documented as possible for future updates and fixes.
    - Scalability: we want to make sure our account recovery function can function efficiently when multiple users want to recover their accounts.

1. Introduction

1.1 Purpose

The purpose of the Low-Level Design of Account Recovery is to present the technical information and the process behind account recovery for CraftVerify registered users or admins.

1.2 Scope

The scope of the Low-Level Design of Account Recovery includes the IRecover Account interface, the Account Recovery class, the sequence diagrams for the account recovery process, the Class diagram, the ER diagram, logging recovery requests, and defining the error handling for each of the fail cases.

2. AccountRecovery Class Design

2.1 Attributes:- (At Top Class Diagram)

2.2 Methods:- (At Top Class Diagram)

3. Sequence Diagrams for Account Recovery

3.1 Success Scenario:- (At Top Success Outcome Diagram)

3.2 Failure Scenario:- (At Top Failure Outcome Diagram)

4. Database Design
4.1 Profile Table:- (At Top ER Diagram)
4.2 Logging Table:- (At Top ER Diagram)
4.3 Account Table:- (At Top ER Diagram)

5. Error Handling
5.1 Error Handling
- The system will display a message to state the account recovery has failed if:
  - Account Recovery takes longer than 3 seconds to complete
  - Account Recovery takes longer than 3 seconds to display the account recovery successful message.
  - Account Recover Successfully but the user is not able to access their account.

5.2 Logging
- Our system will log all successes, fails, and Recovery Attempts.

6. Test Cases
6.1 Unit Tests
- RecoverUserAccountTool_ValidUserId_ReturnsTrue(): Check If the Valid userID makes Recover Account will return True.
- RecoverUserAccountTool_InvalidUserId_ReturnsFalse(): Check If Invalid userID will return False.
- GetDataFromConfigFile_ValidVariable_ReturnsValue(): Check If a Valid variable name is able to retrieve data from the config file.
- ParseConfigFile_ValidFilePath_ReturnsDictionary(): Check If Valid filePath will let Parse Config File open and read from the file.
- CraftRecoverySQLCommand_ValidData_ReturnsValidQuery(): Check If given an object of Dictionary<string, object> it could create a SQL Query.
- IsUserIDValid_ValidUserId_ReturnsTrue(): Check if the checks will return true if provided a valid userID.
- IsUserIDValid_NullUserId_ReturnsFalse(): Check if a null userID is entered it will return false.
- IsUserIDValid_ShortUserId_ReturnsFalse(): Check if a short userID is entered it will return false.

- IsUserIDValid_NonIntegerUserId_ReturnsFalse(): Check if a userID consists of characters other than an integer that will return false when checking for validity.
- IsUserIDValid_EmptySpacesUserId_ReturnsFalse(): Check if a userID consists of Empty Spaces will return false when checking for validity.

Low-Level Design Document for Profile Account Deletion

- **Design Goals:**
  - Security: We want to make the process of deletion reliably secure to where unauthorized access to deletion is properly prevented, ensuring full integrity for the user data.
  - Performance: We want to have our deletion process be efficient to the extent of minimizing system load and removing user data without major performance effects.

1. Introduction

The low-level design doc for the Account Deletion feature within the User Management system of CraftVerify will define user account removal and remove all associated data.

1.1 Purpose

The purpose of this document is to present a technical look into the Account Deletion feature, which will ensure the irreversible removal of personal data and user accounts from the CraftVerify system.

1.2 Scope

For any authenticated regular user/admin, the account deletion feature will allow them to delete their account with CraftVerify permanently, and all Personally Identifiable Information from the system.

2. Account Deletion Service Class Design

2.1 Attribute(s):- (At the Top Under Account Deletion)

2.2 Methods:- (At the Top Under Account Deletion)

3. Sequence Diagrams for Account Deletion