Fpt University

TRƯỜNG ĐẠI HỌC FPT

A SOCIAL NETWORK IRCN V

Software Design Specification

– DaNang, June20233 –

RECORD OF CHANGES

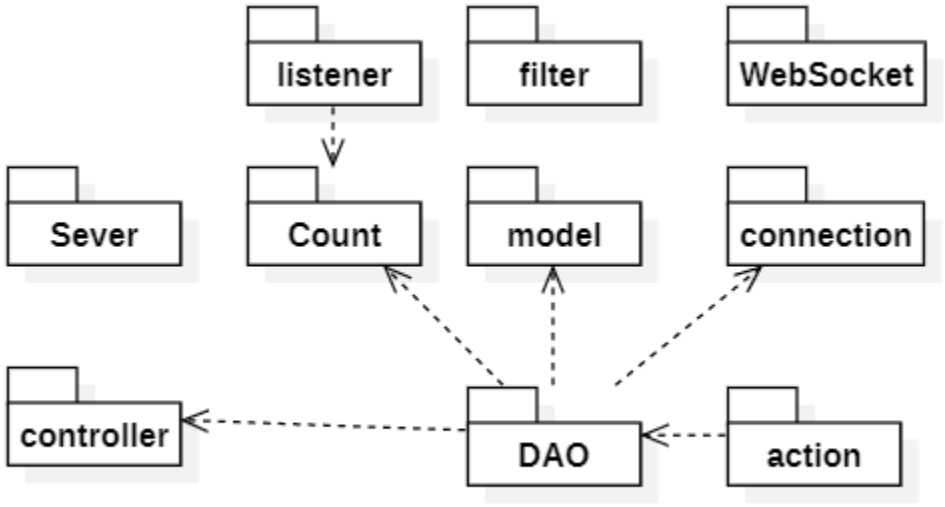
Date	A* M, D	In charge	Change Description
08-05-2023	A*	Nguyen Hong Linh	Drawing a general Use Case diagram and assigning tasks.
08-05-2023	A*	Nguyen Anh Viet, Nguyen Hong Linh, Nguyen Ho Ngoc an, Nguyen Duy Khanh, Nguyen Cong Thinh	Describing the structure of the report and outlining some functionalities for Guest, User, Admin, Master Admin
10-05-2023	A*	Nguyen Anh Viet, Nguyen Hong Linh,	Designing mockups for the Admin, Master Admin
10-05-2023	A*	Nguyen Ho Ngoc an, Nguyen Duy Khanh, Nguyen Cong Thinh	Designing mockups for the Guest, User
13-05-2023	M	Nguyen Ho Ngoc an, Nguyen Duy Khanh, Nguyen Cong Thinh	Editing and refining the mockups.
14-05-2023	M	Nguyen Anh Viet, Nguyen Hong Linh,	Finalizing the report.
15-05-2023	A*	Nguyen Ho Ngoc an, Nguyen Duy Khanh, Nguyen Cong Thinh	Designing mockups for business features of users.
03-07-2023	D	Nguyen Hong Linh	Deleted UseCase: UC-16, UC-18, UC-19, UC-25
03-07-2023	M	Nguyen Hong Linh	Revise the use case diagram
06-07-2023	A*	Nguyen Ho Ngoc an, Nguyen Anh Viet,	Add UseCase UC-35,UC-36,UC-37,UC-38,UC-39
07-07-2023	A*	Nguyen Hong Linh,	Add UseCase UC-40,UC-41,UC-42,UC-43,UC-44,UC-45
09-07-2023	A*	Nguyen Anh Viet, Nguyen Hong Linh, Nguyen Ho Ngoc an, Nguyen Duy Khanh, Nguyen Cong Thinh	Designing mockups for Report

*A - Added M - Modified D - Deleted

Table of Contents

I. Overview

1. Code Packages



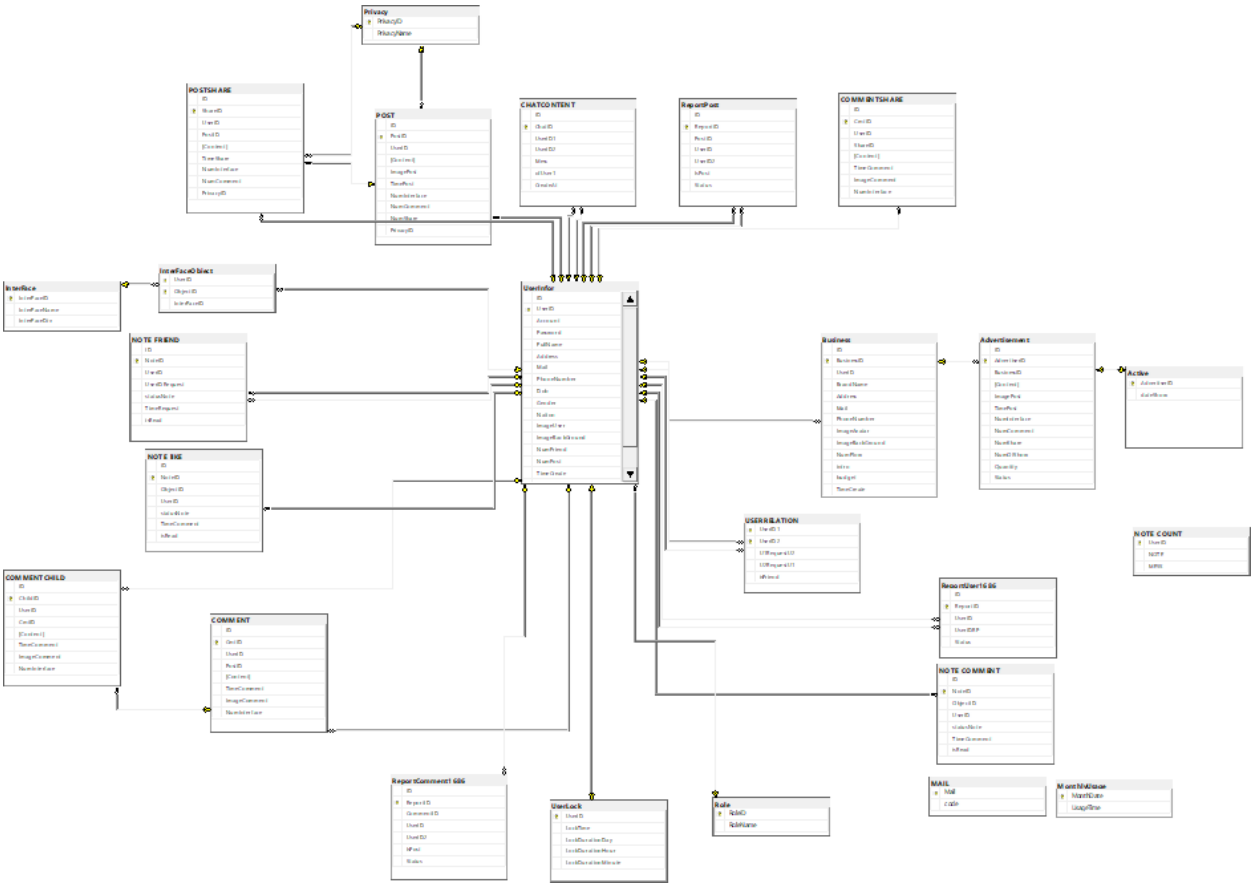
Package descriptions

No	Package	Description
01	*filter*	*This package contains filters that are applied when users access the system without logging in or attempt unauthorized access. These filters are responsible for checking and handling user requests to ensure system safety and security.*
02	*WebSocket*	*This package includes components related to building and managing WebSocket connections. A WebSocket server is implemented to provide real-time messaging capabilities for applications.*
03	*Sever*	*This package contains components related to storing server information. It may include classes or components to manage server details such as name, IP address, configuration information, and other properties.*
04	*Count*	*This package encompasses components related to storing and counting system usage time and the number of users. It may include classes or components to record and track usage time, access counts, and related statistics.*
05	*listener*	*This package consists of components that listen for login and logout events from users. The event information is stored and processed by sending the data to the Count package for storage and counting purposes.*
06	*controller*	*This package contains components related to handling business logic, including encoding, creating storage directories for web images, and sending emails to users. The classes or components in this package are*

		responsible for processing user requests and interacting with other components in the system.
07	*model*	*This package holds Class Objects representing objects in the system. The classes in this package may include attributes and methods for data manipulation and processing.*
08	*connection*	*This package includes components related to connecting to the database. It may consist of classes or components to establish and manage database connections.*
09	*DAO*	*This package contains classes or components related to database access. It utilizes the connection from the Connection package to perform communication operations between Servlets (or other components) and the database, including querying, inserting, updating, deleting data, and other interaction operations.*
10	*action*	*This package includes classes or components that call classes in the DAO package to retrieve data from the database. It serves as a bridge between Servlets and DAO classes to retrieve data from the database and provide it to the user interface (Web).*

2. Database Design

a. Database Schema



b. Table Description

MainTables

No	Table	Description
01	*DBO.Role*	*- This table contains information about user roles in the system. - The columns in this table include "RoleID" (role ID) and "RoleName" (role name).*
02	*DBO.UserInfor*	*- This table stores personal information of users. - The columns in this table include information such as "UserID" (user ID), "Account" (account), "Password" (password), "FullName" (full name), "Address" (address), "Mail" (email), "PhoneNumber" (phone number), "Dob" (date of birth), "Gender" (gender), "Nation" (nationality), "ImageUser" (user image), "ImageBackGround" (background image), "NumFriend" (number of friends), "NumPost" (number of posts), "TimeCreate" (creation time), "RoleID" (role ID), "intro" (introduction about the user). - It has a foreign key "RoleID" referencing the "RoleID" column in the "Role" table.*
03	*DBO.Privacy*	*- This table holds information about privacy settings. - The columns in this table include "PrivacyID" (privacy ID) and "PrivacyName" (privacy name).*
04	*DBO.POST*	*- This table stores information about user posts. - The columns in this table include "PostID" (post ID), "UserID" (user ID who created the post), "Content" (post content), "ImagePost" (post image), "TimePost" (post timestamp), "NumInterface" (number of interactions), "NumComment" (number of comments), "NumShare" (number of shares), "PrivacyID" (privacy ID). - It has a foreign key "UserID" referencing the "UserID" column in the "UserInfor" table, and a foreign key "PrivacyID" referencing the "PrivacyID" column in the "Privacy" table.*
05	*DBO.COMMENT*	*- This table contains information about comments on posts. - The columns in this table include "CmtID" (comment ID), "UserID" (user ID who made the comment), "PostID" (related post ID), "Content"*

		(comment content), "TimeComment" (comment timestamp), "ImageComment" (comment image), "NumInterface" (number of interactions). *- It has a foreign key "UserID" referencing the "UserID" column in the "UserInfor" table, and a foreign key "PostID" referencing the "PostID" column in the "POST" table.*
07	*DBO.COMMENTCHILD*	*- This table stores information about replies to comments.* *- The columns in this table include "ChildID" (child comment ID), "UserID" (user ID who made the child comment), "CmtID" (original comment ID), "Content" (child comment content), "TimeComment" (child comment timestamp), "ImageComment" (child comment image), "NumInterface" (number of interactions).* *- It has a foreign key "UserID" referencing the "UserID" column in the "UserInfor" table, and a foreign key "CmtID" referencing the "CmtID" column in the "COMMENT" table.*
08	*DBO.MAIL*	*- This table contains information about email addresses and their corresponding codes.* *- The columns in this table include "Mail" (email address, primary key) and "code" (code associated with the email).*
09	*DBO.USERRELATION*	*- This table stores information about relationships between users.* *- The columns in this table include "UserID1" (user ID 1) and "UserID2" (user ID 2), used to identify the relationship between two users.* *- It has foreign key constraints referencing the "UserInfor" table for both "UserID1" and "UserID2".*
10	*DBO.CHATCONTENT*	*- This table contains information about chat conversations between two users.* *- The columns in this table include "ChatID" (chat ID), "UserID1" (user ID 1), and "UserID2" (user ID 2) to identify the participating users.* *- The "Mess" column stores the chat content, while the "ofUser1" column determines whether the chat belongs to user 1 or not.*
11	*DBO.POSTSHARE*	*- This table stores information about user shares of posts.* *- The columns in this table include "ShareID" (share ID), "UserID" (user ID), "PostID" (post ID),*

		"Content" (share content), "TimeShare" (share timestamp), "NumInterface" (number of interactions), "NumComment" (number of comments), "PrivacyID" (privacy ID). *- It has a foreign key "UserID" referencing the "UserID" column in the "UserInfor" table and a foreign key "PostID" referencing the "PostID" column in the "POST" table.* *- The "PrivacyID" column has a foreign key referencing the "PrivacyID" column in the "Privacy" table.*
12	*DBO.INTERFACE*	*- This table contains information about the system interfaces.* *- The columns in this table include "InterFaceID" (interface ID) and "InterFaceName" (interface name).*
13	*DBO.INTERFACEOBJECT*	*- This table stores information about objects in the system interface.* *- The columns in this table include "UserID" (user ID) and "ObjectID" (object ID).* *- It has a foreign key "UserID" referencing the "UserID" column in the "UserInfor" table and a foreign key "InterFaceID" referencing the "InterFaceID" column in the "InterFace" table.*
14	*DBO.NOTE_COUNT*	*- This table contains information about the number of notifications for each user.* *- The columns in this table include "UserID" (user ID, primary key), "NOTE" (number of notifications), and "MESS" (number of messages).*
15	*DBO.NOTE_FRIEND*	*- This table stores information about friend requests and related notifications.* *- The columns in this table include "NoteID" (notification ID), "UserID" (user ID), "UserIDRequest" (user ID who sent the friend request), "statusNote" (notification status), "TimeRequest" (request timestamp), and "isRead" (read status).* *- It has foreign key constraints referencing the "UserInfor" table for both "UserID" and "UserIDRequest".* *- The unique constraint "uc_UserID_UserIDRequest" ensures the combination of "UserID" and "UserIDRequest" is unique.*

16	*DBO.NOTE_COMMENT*	*- This table contains information about notifications related to comments.* *- The columns in this table include "NoteID" (notification ID), "ObjectID" (related object ID, can be a post ID or a comment ID), "UserID" (user ID), "statusNote" (notification status), "TimeComment" (comment timestamp), and "isRead" (read status).* *- It has foreign key constraints referencing the "UserInfor" table for "UserID".*
17	*DBO.NOTE_LIKE*	*- This table contains information about notifications related to interactions (likes) on posts or comments.* *- The columns in this table include "NoteID" (notification ID), "ObjectID" (related object ID, can be a post ID or a comment ID), "UserID" (user ID), "statusNote" (notification status), "TimeComment" (interaction timestamp), and "isRead" (read status).* *- It has a unique constraint "uc_UserID_ObjectID" on the combination of "UserID" and "ObjectID".* *- The "ObjectID" column refers to the ID of the post or comment being interacted with.*

Business

No	Table	Description
01	*DBO.Business*	*The "Business" table stores information about businesses, including their ID, brand name, user ID, address, email, phone number, images, number of flows, introduction, budget, and creation timestamp.*
02	*DBO.Advertisement*	*The "Advertisement" table stores information about advertisements, including the advertiser ID, associated business ID, content, image, posting time, interaction counts, quantity, and status.*
03	*DBO.Active*	*The "Active" table tracks active advertisements by storing the advertiser ID and the date and time the advertisement was shown.*

dashBoard

No	Table	Description
01	*PostSummaryByMonth*	*The provided code creates a view named "PostSummaryByMonth" that aggregates the number of posts and post shares by month and year. It also creates a table named "MonthlyUsage" to store monthly usage data with the corresponding month and usage time.*
02	*DBO.ReportPost*	*The provided code creates a table named "ReportPost" to store reports on posts. It includes columns for the report ID, post ID, user IDs associated with the report, and flags indicating the type of item reported and its status.*
03	*ReportPostView*	*The "ReportPostView" view combines data from the "ReportPost," "Post," and "PostShare" tables. It retrieves the post ID, item type (post or post share), image, content, user count, time, user ID, and status for each reported item.*
04	*DBO.ReportComment1686*	*The provided code creates a table named "ReportComment1686" to store reports on comments. It includes columns for the report ID, comment ID, user IDs associated with the report, and flags indicating the type of item reported and its status.*
05	*ReportCommentView*	*The "ReportCommentView" view combines data from the "ReportComment1686," "COMMENT," and "COMMENTSHARE" tables. It retrieves the comment ID, item type (comment or shared comment), image, content, user count, time, user ID, and status for each reported item.*
06	*DBO.ReportUser1686*	*The provided code creates a table named "ReportUser1686" to store reports on user accounts. It includes columns for the report ID, user IDs associated with the report, and a status flag.*
07	*DBO.UserLock*	*The "UserLock" table manages user locks in a system. It includes columns for the user ID, lock time, lock duration in days, hours, and minutes. The user ID serves as the primary key, and it references the `UserID` column in the `UserInfor` table.*
08	*UserReportSummary*	*The "UserReportSummary" view retrieves user information from the "UserInfor" table and provides a summary of user reports. It includes the user ID, image, full name, account, email, phone number, address, number of comments reported, number of posts reported, and number of users who reported the user. The view filters the results based on the "ReportUser1686" table, considering only users with a report status of 1.*
09	*UserView*	*The "UserView" view retrieves user information from the "UserInfor" table, including the user ID, image, full name, address, email, account, phone number, date of birth, nationality, and role ID.*

II. Code Designs

1. Sign Up

a. Class Diagram

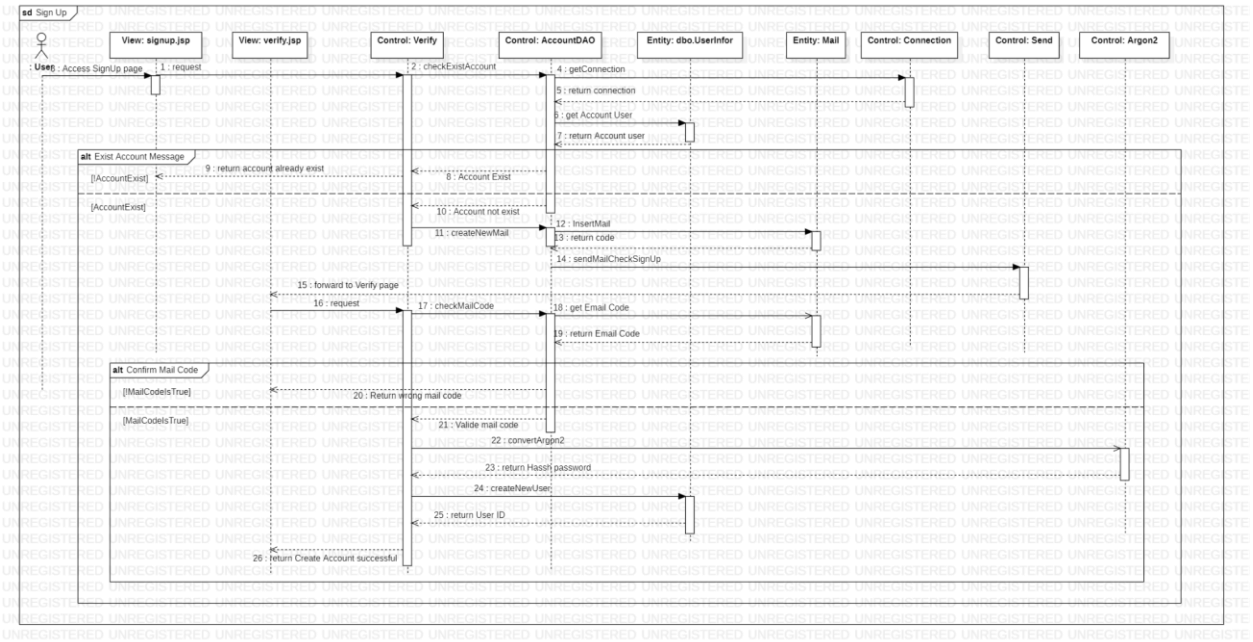
b. Class Specifications

AccountDAO

No	Method	Description
01	*AccountDAO()*	*creates a Connection object, and then the getConnection() method of the Connection Class is called to establish the connection to the database..*
02	*resetPass(String user, String pass)*	*The resetPass method is used to update the new password for the user in the database. It converts the password into an encrypted form using the Argon2 algorithm and updates the new password into the database. If an error occurs, an error message will be displayed.*
03	*getAccountByUserID(String UserID)*	*The getAccountByUserID method is used to get the user's account information based on the user ID. It queries the database for the account value and returns that value. If there is an error, an error message is displayed and an empty string is returned.*
04	*checkMailCode(String mail, String code)*	*The checkMailCode method is used to check the confirmation code matches the given email address. It queries the database and returns true if it matches, false otherwise. If there is an error, an*

No	Method	Description
		error message is displayed and false is returned.
05	*checkExistAccount(String user)*	*The checkExistAccount method is used to check the existence of an account based on the username. It queries the database and returns true if the account exists, false otherwise. If there is an error, an error message is displayed and false is returned.*
06	*getIdUser(String user)*	*The `getIdUser` method is used to get the user ID based on the username. If found, the user ID is returned. If not found or there is an error, return null.*
07	*createNewUser(String user, String pass, String name, String mail, String dob, String getGender)*	*The `createNewUser` method is used to create a new user in the database with the corresponding information. If successful, return true. If there is an error, return false.*

c. Sequence Diagram(s)



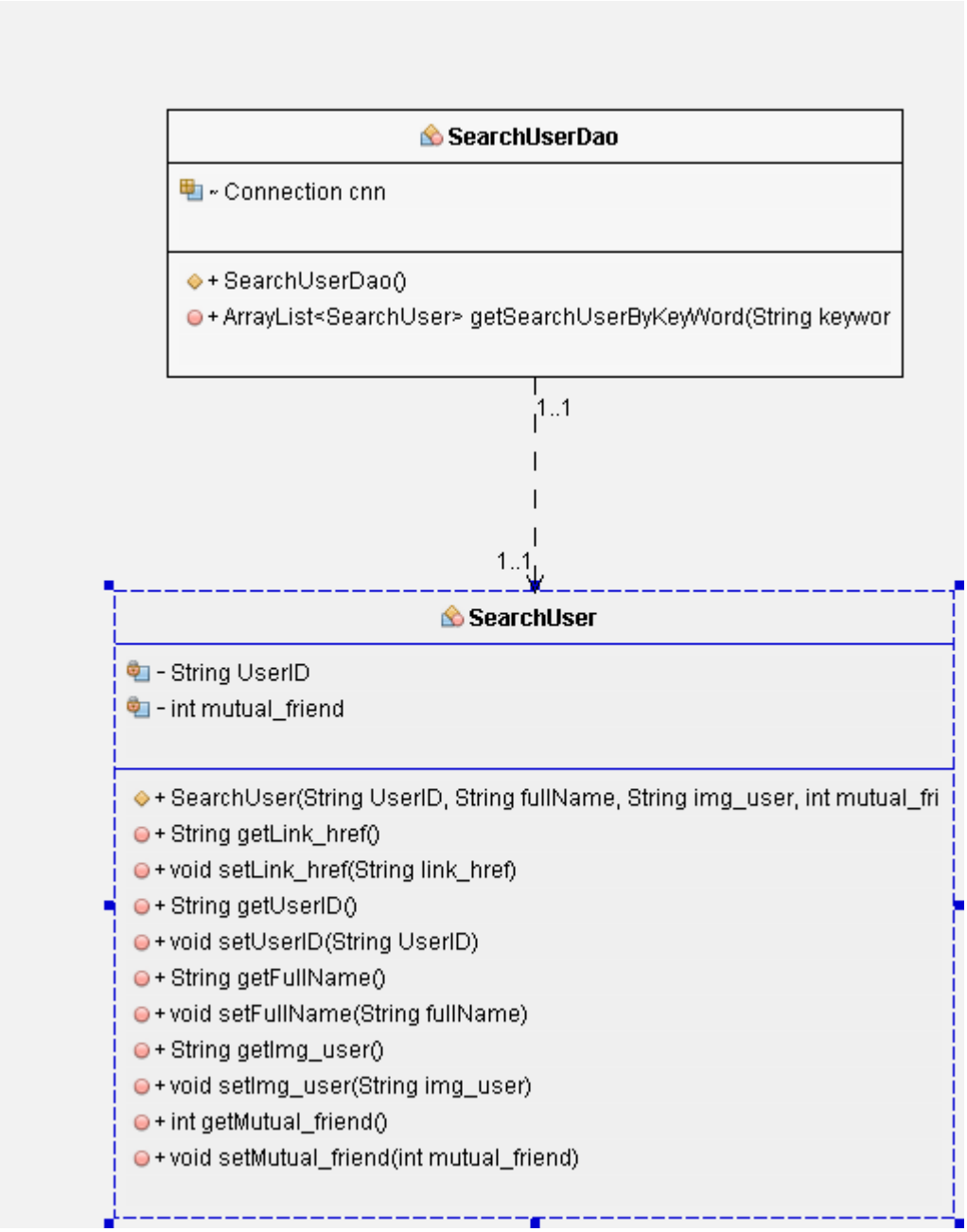
d. Database Queries

No	Queries	Description
01	*INSERT INTO dbo.UserInfor (Account, Password, FullName,*	*This INSERT INTO statement is used to add a new record to the "UserInfor" table*

No	Queries	Description
	Address, Mail, PhoneNumber, Dob, Gender, Nation, ImageUser, ImageBackGround, NumFriend, NumPost, TimeCreate, RoleID) VALUES (?, ?, ?, NULL, ?, NULL, ?, ?, NULL, NULL, NULL, DEFAULT, DEFAULT, DEFAULT, DEFAULT)	*in the database. The values of the fields are provided in the SQL statement, where some fields have specific values, while others will have default values.users based on the user ID. It queries the database for the account value and returns that value. If there is an error, an error message is displayed and an empty string is returned.*
02	*SELECT FROM dbo.UserInfor WHERE Account= ?*	*This SELECT query statement is used to get all columns and rows from the "UserInfor" table in the database, provided that the "Account" field must match the value provided in the SQL statement.*
03	*SELECT * FROM dbo.MAIL WHERE Mail= ? AND code= ?*	*This SELECT query statement is used to get all columns and rows from the "MAIL" table in the database, provided that the "Mail" field must match the value provided in the SQL statement and the " code" must match the value provided in the SQL statement.*
04	*SELECT UserID FROM dbo.UserInfor WHERE Account= ?*	*This SELECT query statement is used to get the value of the "UserID" field from the "UserInfor" table in the database, provided that the "Account" field must match the value provided in the SQL statement.*
05	*UPDATE dbo.UserInfor SET Password= ? WHERE Account= ?*	*This UPDATE query statement is used to update the value of the "Password" field in the "UserInfor" table in the database, provided that the "Account" field must match the value provided in the SQL statement.*

2. Find

a. Class Diagram



b. Class Specifications

SearchUser Class

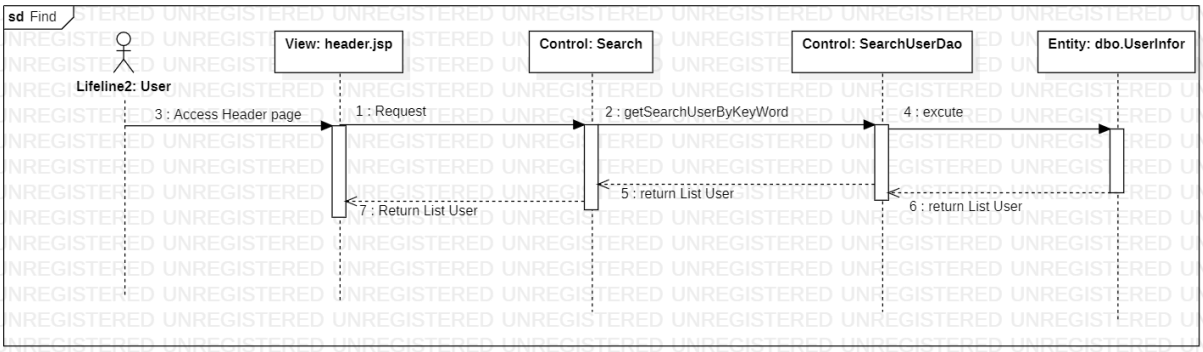
No	Method	Description
01	*SearchUser(String UserID, String fullName, String img_user, int mutual_friend)*	*This constructor is used to create a SearchUser object with the provided values. After calling this constructor, the properties of the SearchUser object will be set to the corresponding values.*

No	Method	Description
02	*Getter/Setter*	*Getter and Setter methods are used to access and set the value of properties in a class.*

SearchUserDAO

No	Method	Description
01	*SearchUserDao()*	*creates a Connection object, and then the getConnection() method of the Connection Class is called to establish the connection to the database..*
01	*ArrayList<SearchUser> getSearchUserByKeyWord(String keyword)*	*The `getSearchUserByKeyWord` method is used to search for users based on the keyword `keyword` in the database. It returns an `ArrayList<SearchUser>` list containing `SearchUser` objects corresponding to the search results. Each `SearchUser` object contains information such as the UserID, FullName, ImageUser, and NumFriend of the respective user.*

c. Sequence Diagram(s)



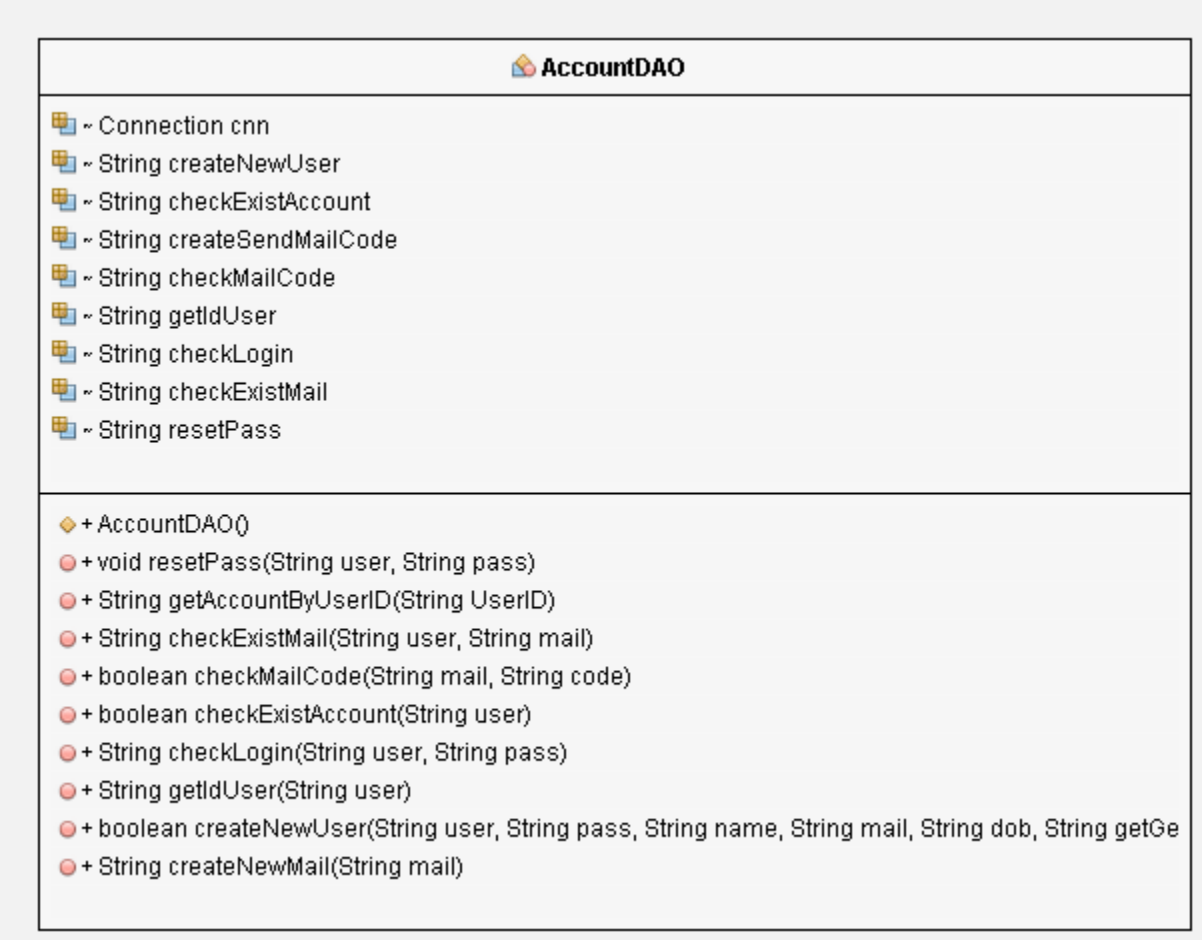
d. Database Queries

No	Queries	Description
01	*SELECT UserId,FullName, ImageUser, NumFriend FROM dbo.UserInfor WHERE FullName COLLATE Latin1_General_CI_AI LIKE '%' + ? + '%'*	*This SELECT query statement is used to search for users in the "UserInfor" table based on the keyword ? in the "FullName" field. The statement uses the LIKE pattern to search for "FullName" values containing the corresponding keyword.*

No	Queries	Description

3. Login, Logout And Forgot Password

a. Class Diagram



b. Class Specifications

AccountDAO

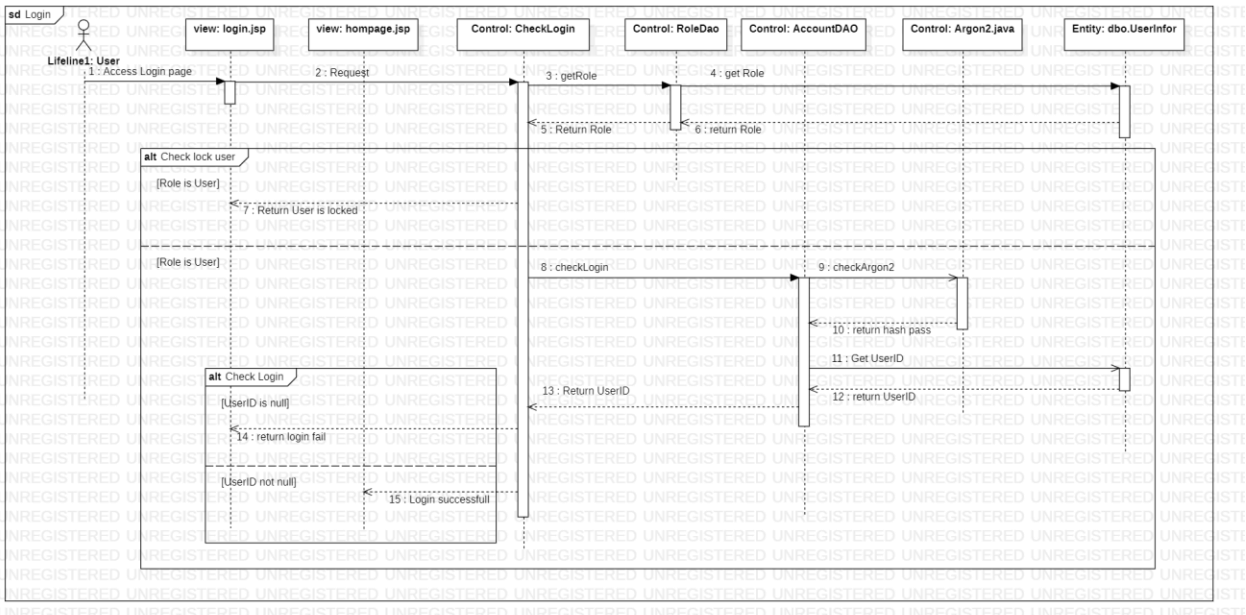
No	Method	Description
01	*checkLogin(String user, String pass)*	*The `checkLogin` method is used to verify the login by comparing the provided username and password with the account information in the database. If matched, the username is returned. If there is no match or error, return null.*
02	*checkExistMail(String user, String mail)*	*The checkExistMail method is used to check the existence of an email address in the database for a specific user. It queries the database and returns the corresponding email value if one exists. If*

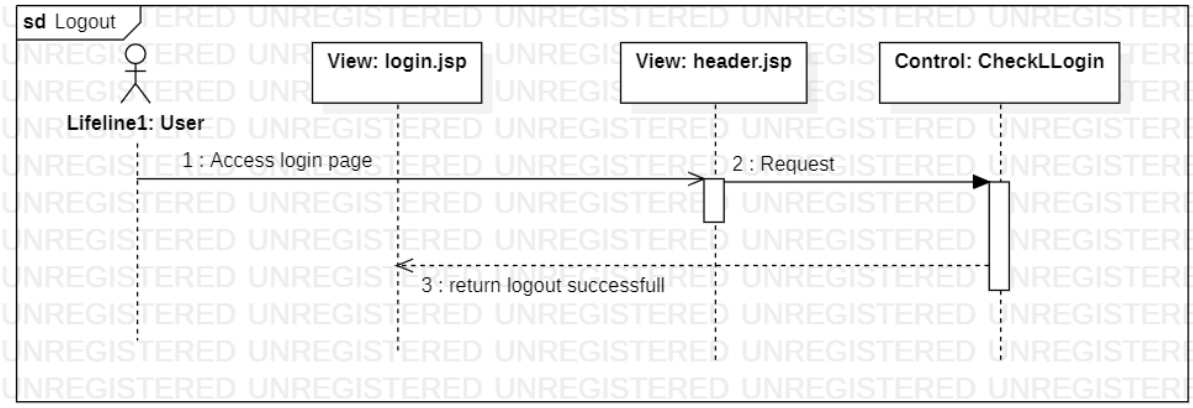
		there is an error, an error message is displayed and a null value is returned.
03	*createNewMail(String mail)*	*The `createNewMail` method is used to generate and store a new confirmation code for the provided email address in the database. If successful, a confirmation code is returned. If there is an error, return null.*

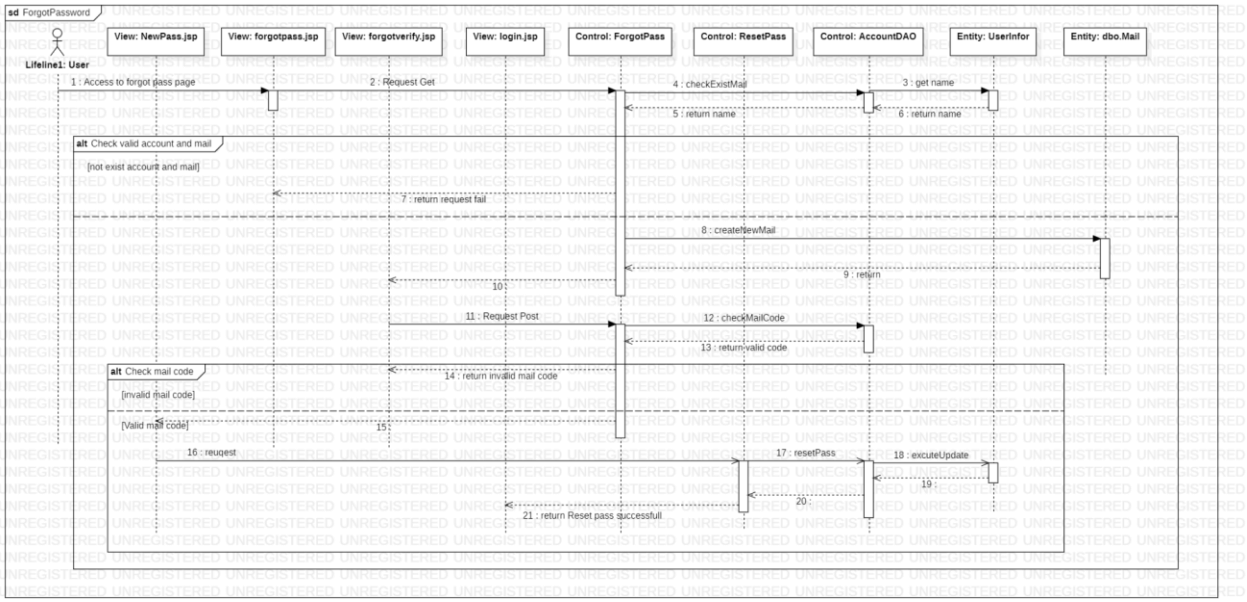
RoleDAO

No	Method	Description
01	*getRole(String UserID)*	*The `getRole` method is used to get information about the user's role based on the UserID. It checks if the user is locked and performs user role updates if necessary. It then returns a Role object containing information about the user's role and key status.*

c. Sequence Diagram(s)







d. Database Queries

AccountDAO

No	Queries	Description
01	*AccountDAO()*	*creates a Connection object, and then the getConnection() method of the Connection Class is called to establish the connection to the database..*
02	*SELECT UserID, Password FROM dbo.UserInfor WHERE Account= ?*	*This SELECT query statement is used to get the values of the "UserID" and "Password" fields from the "UserInfor" table in the database, provided that the "Account" field must match the value provided in the statement. SQL command.*

03	*SELECT FullName FROM dbo.UserInfor WHERE Account= ? AND Mail= ?*	*This SELECT query statement is used to get the value of the "FullName" field from the "UserInfor" table in the database, provided that the "Account" field must match the value provided in the SQL statement and the field "Mail" must also match the value provided.*
04	*DECLARE @mail VARCHAR(255);* *SET @mail = ? ;* *DECLARE @code CHAR(10);* *SET @code = ? ;* *IF NOT EXISTS (SELECT * FROM dbo.MAIL WHERE Mail = @mail)* *BEGIN* *INSERT INTO dbo.MAIL* *VALUES (@mail, @code)* *END* *ELSE* *BEGIN* *UPDATE dbo.MAIL* *SET code = @code* *WHERE Mail = @mail* *END*	*The above query statement is used to check the existence of an email address in the "MAIL" table of the database. Here is a brief description of how the statement works:* *- Declare the variable "@mail" with data type VARCHAR(255) and assign the value from the parameter provided in the SQL statement.* *- Declare the variable "@code" with data type char(10) and assign the value from the parameter provided in the SQL statement.* *- Use the IF NOT EXISTS statement to check if the email address "@mail" exists in the "MAIL" table.* *- If not exists, execute INSERT INTO statement to add a new record to the "MAIL" table with the value of the email address "@mail" and the confirmation code "@code".* *- If the email address already exists, execute the UPDATE statement to update the "@code" confirmation code value for the existing "@mail" email address in the "MAIL" table.*

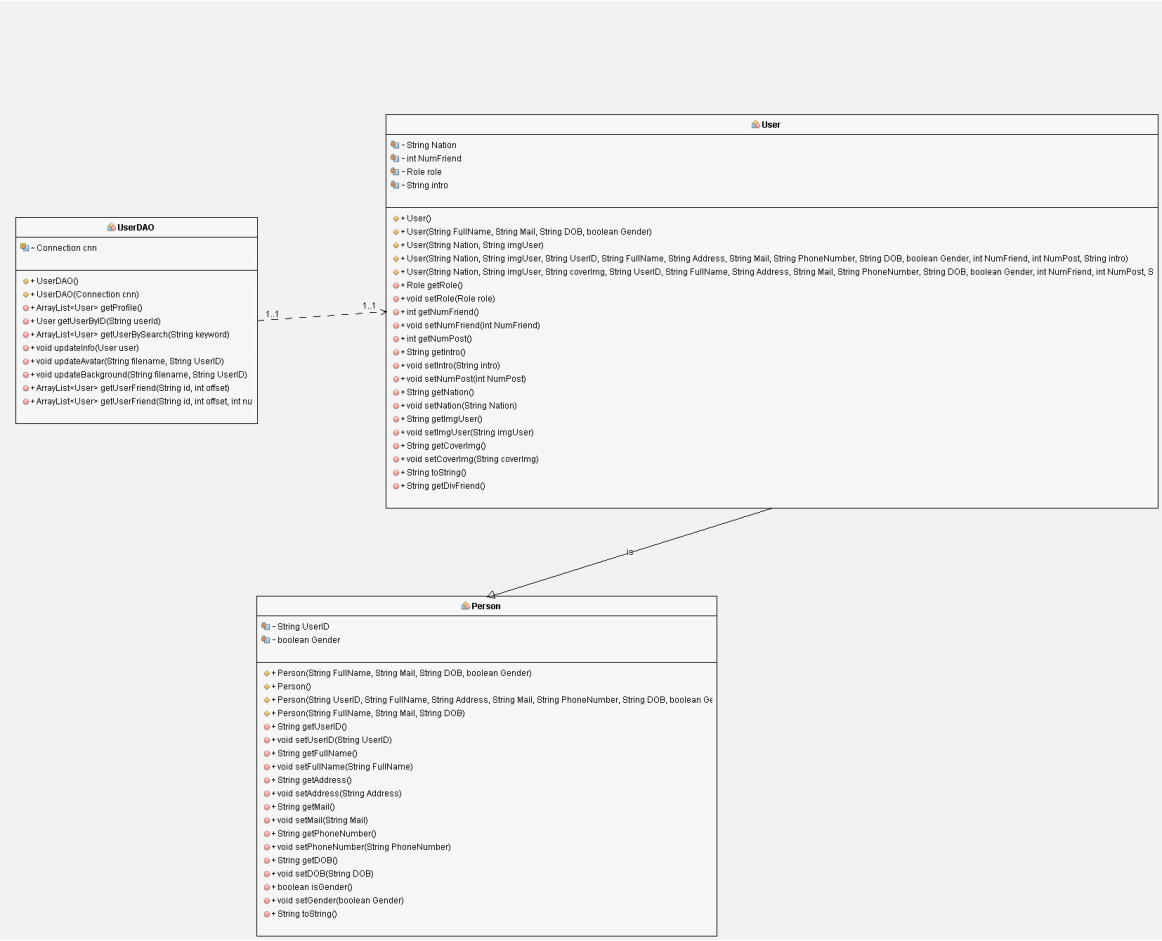
RoleDAO

No	Method	Description
01	*RoleDao()*	*creates a Connection object, and then the getConnection() method of the Connection Class is called to establish the connection to the database..*
02	*DECLARE @UserID NVARCHAR(11) = ?* *DECLARE @isLock BIT = CASE WHEN ((SELECT TOP(1)*	*This query statement is used to perform some operations on the user and role database. Here is the job description of each part of the query statement:*

	DATEADD(MINUTE, LockDurationMinute, DATEADD(HOUR, LockDurationHour, DATEADD(DAY, LockDurationDay, LockTime))) FROM UserLock WHERE UserID = @UserID) > GETDATE()) THEN 1 ELSE 0 END *DECLARE @Role VARCHAR(11)* *SELECT @Role = UserInfor.RoleID FROM dbo.UserInfor WHERE UserInfor.UserID = @UserID* *IF (@isLock = 0 AND @Role = 'LOCK') BEGIN UPDATE dbo.UserInfor SET UserInfor.RoleID = 'USER' WHERE UserInfor.UserID = @UserID DELETE FROM dbo.UserLock WHERE UserLock.UserID = @UserID END* *SELECT Role.RoleID, RoleName, @isLock FROM dbo.UserInfor INNER JOIN dbo.Role ON Role.RoleID = UserInfor.RoleID WHERE UserID = @UserID* *SELECT Role.RoleID, RoleName, @isLock FROM dbo.UserInfor INNER JOIN dbo.Role ON Role.RoleID = UserInfor.RoleID WHERE UserID = @UserID*	*1. Declare variables:* *- `@UserID` is a string variable (NVARCHAR) with a maximum length of 11 characters. The value of this variable needs to be provided before executing the query.* *- `@isLock` is a bit variable (BIT) used to store the user's lock state. The value of this variable is determined based on the user's lock time (calculated from the UserLock table) and the current time (using the GETDATE() function).* *2. Get the user's role:* *- This SELECT query retrieves the user's role (RoleID) from the UserInfor table based on the value of the @UserID variable.* *3. Check and update user if not locked and whose role is 'LOCK':* *- If the @isLock variable is 0 (not locked) and the value of the @Role variable is 'LOCK', execute the UPDATE statement to update the user's role to 'USER' in the UserInfor table. Also, delete the user's key information in the UserLock table.* *4. Get the user's key status and role information:* *- This SELECT query combines the UserInfor table and the Role table through an INNER JOIN to get information about the user's role, the role name, and the value of the @isLock variable. Returns the RoleID, RoleName columns and the value of the @isLock variable from the UserInfor table.* *5. Get the user's key status and role information (repeated):* *- SELECT query is similar to query 4, but without any changes.*

4. Manage Profile

a. Class Diagram



b. Class Specifications

Person Class

No	Method	Description
01	*Person(in String FullName, in String Mail, in String DOB, in boolean Gender)*	*This constructor is used to create a Person object with the provided values. After calling this constructor, the properties of the Person object will be set to the corresponding values.*
02	*Person(in String UserID, in String FullName, in String Address, in String Mail, in String PhoneNumber, in String DOB, in boolean Gender)*	*This constructor is used to create a Person object with the provided values. After calling this constructor, the properties of the Person object will be set to the corresponding values.*

No	Method	Description
02	*Getter/Setter*	*Getter and Setter methods are used to access and set the value of properties in a class.*

User Class

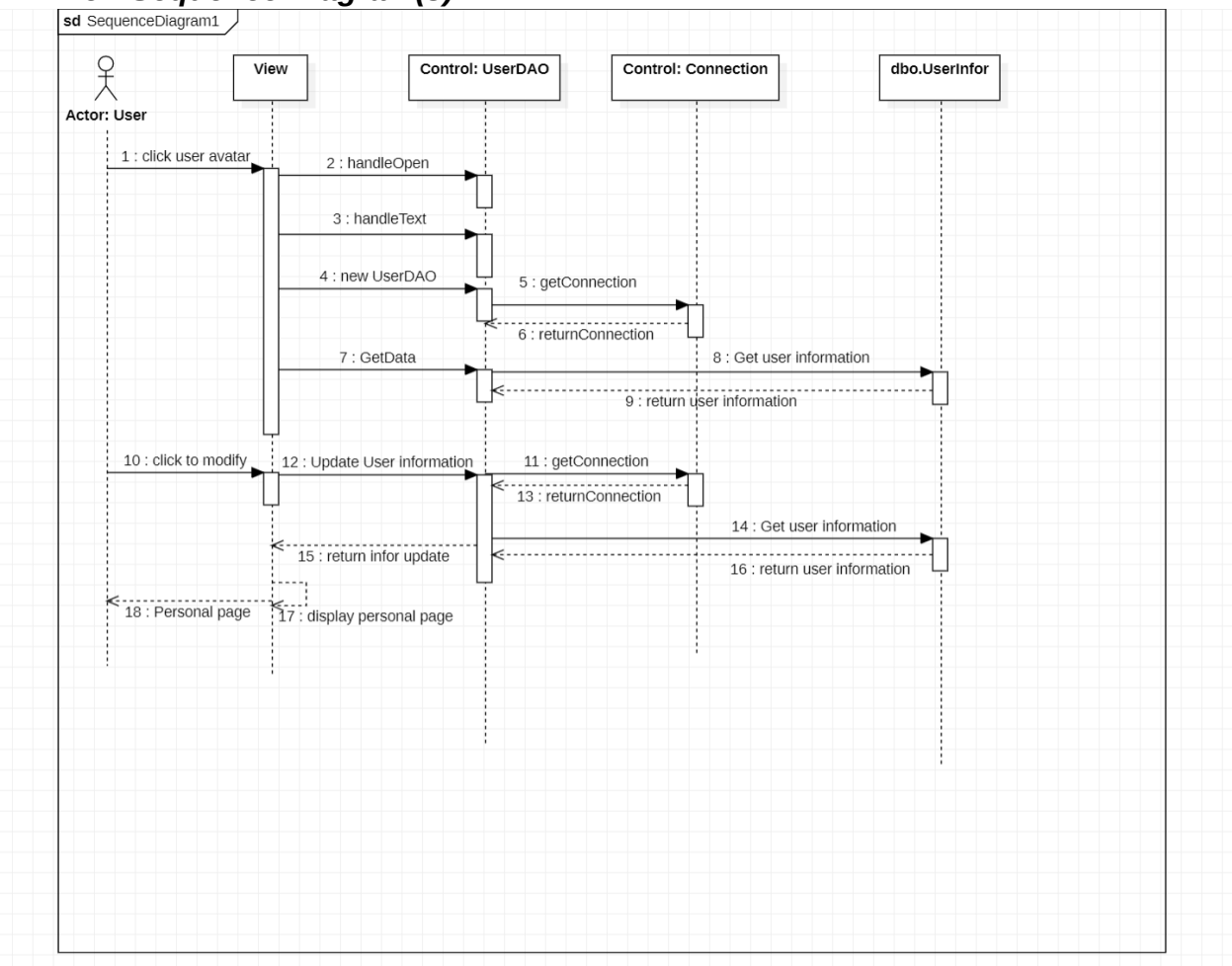
No	Method	Description
01	*User(in String Nation, in String imgUser, in String coverImg, in String UserID, in String FullName, in String Address, in String Mail, in String PhoneNumber, in String DOB, in boolean Gender)*	*This constructor is used to create a User object with the provided values. After calling this constructor, the properties of the User object will be set to the corresponding values.*
02	*public User(in String Nation, in String imgUser, in String UserID, in String FullName, in String Address, in String Mail, in String PhoneNumber, in String DOB, in boolean Gender)*	*This constructor is used to create a User object with the provided values. After calling this constructor, the properties of the User object will be set to the corresponding values.*
02	*Getter/Setter*	*Getter and Setter methods are used to access and set the value of properties in a class.*

UserDAO

No	Method	Description
01	*UserDAO()*	*creates a Connection object, and then the getConnection() method of the Connection Class is called to establish the connection to the database..*
02	*getProfile()*	*Method to query the database to get information about the users. This method returns a list of User objects containing user information*
03	*getUserByID(String userId)*	*The method executes an SQL query to get data from the UserInfor.table by User Id* *Finally, the method returns a User object containing that user's information. In case of an error, the method prints an error message and returns null.*
04	*updateAvatar(String filename, String UserID)*	*The method returns the filename with the specified UserID.*

No	Method	Description
		Method to execute SQL query to update data from UserInfor table according to user's Id *Then the implementation method updates the avatar image file according to the user's Id*
05	*updateBackground(String filename, String UserID)*	*The method returns the filename with the specified UserID.* *Method to execute SQL query to update data from UserInfor table according to user's Id* *Then the method updates the background image file according to the user's Id*

c. Sequence Diagram(s)

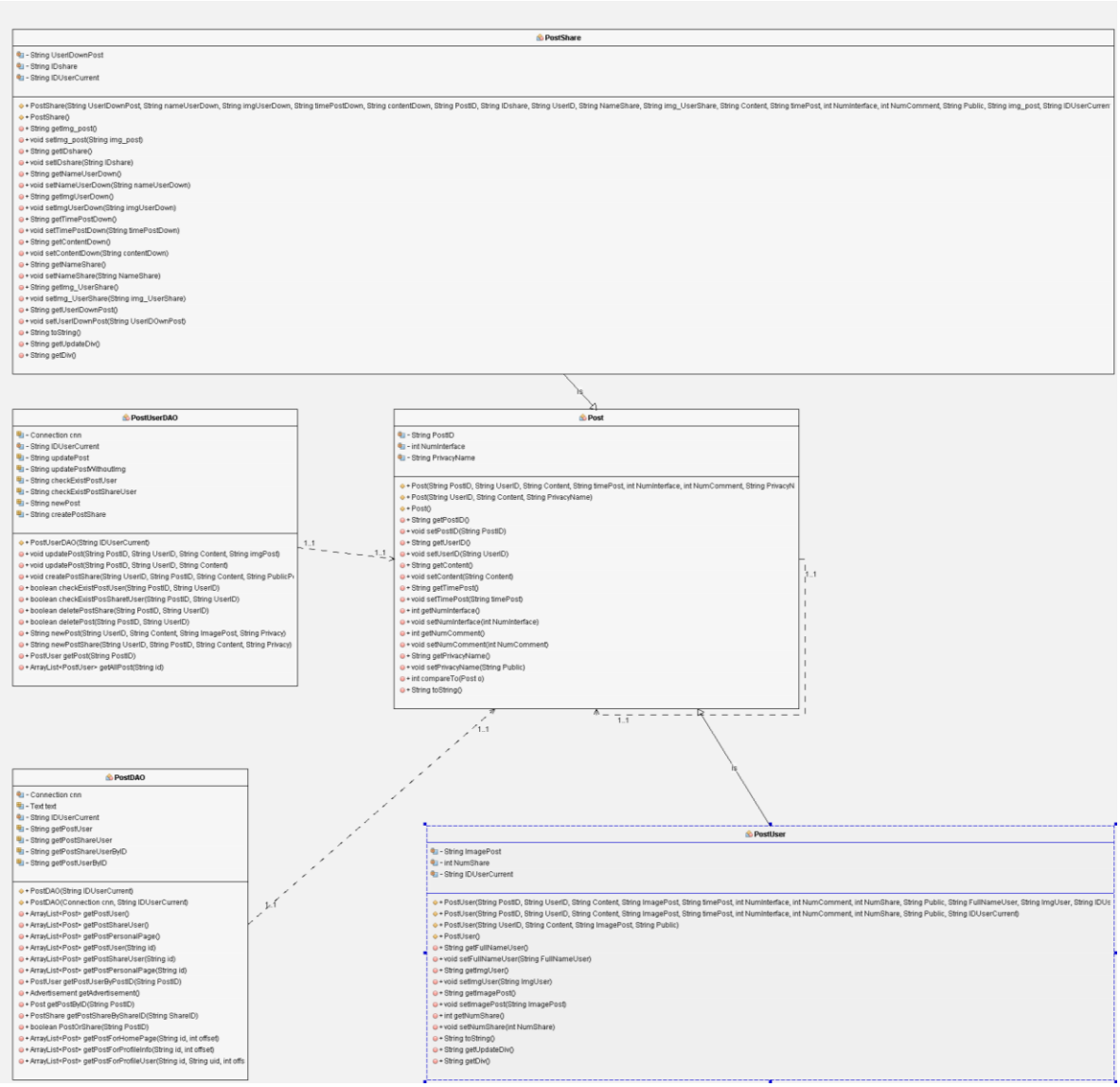


d. Database Queries

No	Queries	Description
01	*SELECT UserID, FullName, Address, Mail, PhoneNumber, Dob, Gender, Nation, ImageUser, ImageBackGround FROM dbo.UserInfor WHERE UserID = ?*	*This SQL query statement is used to get user information from the "UserInfor" table in the database. .*
02	*update UserInfor set FullName = ? , Address = ? ,Mail = ? ,PhoneNumber = ? ,Dob = ? ,Gender = ? ,Nation = ? where UserID = ?*	*This SQL query statement is used to update the user information in the "UserInfor" table of the database.*
03	*update UserInfor set ImageUser = ? where UserID = ?*	*This SQL query statement is used to update the "ImageUser" column in the "UserInfor" table of the database.*
04	*update UserInfor set ImageBackGround = ? where UserID = ?*	*This SQL query statement is used to update the "ImageBackGround" column in the "UserInfor" table of the database.*

5. Manage Post And Post Share

a. Class diagram



b. Class Specifications

Post class

No	Method	Description
01	*Post(String PostID, String UserID, String Content, String timePost, int NumInterface, int NumComment, String PrivacyName)*	*This initialization method is used to create a Post object with the provided values. After calling this initialization method, the properties of the Post object will be set with the corresponding values.*

02	*Post(String UserID, String Content,String PrivacyName)*	*This initialization method is used to create a Post object with the provided values for UserID, Content, and PrivacyName. After calling this initialization method, the properties UserID, Content, and PrivacyName of the Post object will be set with the corresponding values. Other properties of the Post object such as PostID, timePost, NumInterface, and NumComment may not be initialized and will retain their default values (if any) or be set to null (if no default value exists).*
03	*Getter; Setter()*	*Getter và Setter methods are used to access and modify the values of properties within a class.*
04	*Post()*	*The provided new Post constructor does not take any parameters. It is used to create a Post object without any initial values for its properties.*
05	*compareTo(Post o)*	*The compareTo method is overridden (@Override) from the Comparable<Post> interface. It compares two Post objects based on their timePost property and returns an integer value.*

PostUser class

No	Method	Description
01	*PostUser(String PostID, String UserID, String Content, String ImagePost, String timePost, int NumInterface, int NumComment, int NumShare, String Public, String FullNameUser, String ImgUser, String IDUserCurrent)*	*This initialization method is used to create a PostUser object with the provided values. After calling this initialization method, the properties of the PostUser object will be set with the corresponding values.*
02	*PostUser(String PostID, String UserID, String Content, String ImagePost, String timePost, int NumInterface, int NumComment, int NumShare, String Public, String IDUserCurrent)*	*This constructor method is used to create a PostUser object with the provided values. After calling this constructor method, the properties of the PostUser object will be set with the corresponding values.*

03	*Getter; Setter()*	*Getter and Setter methods are used to access and modify the values of properties within a class.*
04	*PostUser(String UserID, String Content, String ImagePost, String Public)*	*This constructor method is used to create a PostUser object with the provided values. The properties UserID, Content, and Public of the PostUser object will be set through the constructor method of the parent class Post. The ImagePost property will be directly set from the ImagePost parameter.*
05	*PostUser()*	*The provided new PostUser constructor does not take any parameters. It is used to create a PostUser object without any initial values for its properties.*
06	*getUpdateDiv()*	*The getUpdateDiv() method is used to generate an HTML string to update the user interface when there are changes in a post, such as when a user performs interactions like liking, commenting, or sharing the post.*
07	*getDiv()*	*The getDiv() method is used to generate an HTML string to display a post in the user interface.*

PostShare class

No	Method	Description
01	*PostShare(String UserIDownPost, String nameUserDown, String imgUserDown, String timePostDown, String contentDown, String PostID, String IDshare, String UserID, String NameShare, String img_UserShare, String Content, String timePost, int NumInterface, int NumComment, String Public, String img_post, String IDUserCurrent)*	*This constructor initializes a PostShare object with corresponding properties to store information about the shared post and related details of the original post.*
02	*PostShare()*	*The provided new PostShare constructor does not take any parameters.*
03	*Getter; Setter()*	*Getter and Setter methods are used to access and set the values of properties within a class..*

04	*getUpdateDiv()*	*The getUpdateDiv() method returns an HTML string representing the content section of the shared post. Ultimately, the returned HTML string is used to display the shared post on a website.*
05	*getDiv()*	*The getDiv() method is used to generate an HTML string to display the shared post in the user interface.*

PostDao class

No	Method	Description
01	*PostDAO(String IDUserCurrent)*	*This constructor allows access and use of the methods and properties of the PostDAO object for interacting with the database and processing text in the context of the current user.*
02	*PostDAO(Connection cnn, String IDUserCurrent)*	*This constructor enables access and utilization of the methods and properties of the PostDAO object for interacting with the database and processing text within the context of the current user.*
03	*getPostUser()*	*The getPostUser() method returns a list of Post objects representing the user's posts.* *This method allows to get a list of user posts from the database and return it for display or further processing in the application.*
04	*getPostShareUser()*	*The getPostShareUser() method returns a list of Post objects representing the user's shared posts.* *This method allows to get a list of user's shared posts from the database and return it for display or further processing in the application.*
05	*getPostPersonalPage()*	*This method allows to get a list of posts on a user's profile by combining a list of personal posts and a list of shared posts. This helps to display and further process the user's profile posts in the application.*
06	*getPostUser(String id)*	*The getPostUser(String id) method takes an id parameter representing the user's ID and returns a list of Post objects representing that user's posts.*
07	*getPostShareUser(String id)*	*The getPostShareUser(String id) method takes an id parameter representing the user's ID and returns a list of PostShare objects representing that user's shared posts.*
08	*getPostPersonalPage(String id)*	*The getPostPersonalPage(String id) method takes an id parameter representing the user's ID and returns a list of Post objects representing that user's personal and shared posts on the profile page. core.*
09	*getPostUserByPostID(String PostID)*	*The `getPostUserByPostID(String PostID)` method takes a `PostID` parameter representing the post's ID and returns a `PostUser` object representing the corresponding individual post.*

10	*getPostByID(String PostID)*	*The getPostByID(String PostID) method takes a PostID parameter representing the ID of a post or share and returns a Post object corresponding to that post or share.*
11	*getPostShareByShareID(String ShareID)*	*The getPostShareByShareID(String ShareID) method takes a ShareID parameter representing the ID of a share and returns a PostShare object corresponding to that share.*
12	*PostOrShare(String PostID)*	*The PostOrShare(String PostID) method takes a PostID parameter representing the ID of a post or share and returns a boolean to determine if the PostID is the ID of the post or share.*
13	*getPostForHomePage(String id, int offset)*	*The getPostForHomePage(String id, int offset) method is used to get the list of posts on the homepage. This method allows to get a list of posts on the homepage, sorted by time and containing both posts and shares. In addition, it can also insert ads into the list of posts.*
14	*getAdvertisement()*	*The getAdvertisement() method is used to get ad information. This method allows to get information about the advertisement from the database. It retrieves advertising information from the Advertisement table and updates the relevant information in the Active table. It then returns an Advertisement object containing the advertisement information.*
15	*getPostForProfileInfo(String id, int offset)*	*The getPostForProfileInfo() method is used to get the list of posts for the profile page. This method allows to get the list of articles from the database for the profile page. It retrieves a list of posts from the POST and POSTSHARE tables based on the UserID, and then returns a list of the corresponding Post objects.*
16	*getPostForProfileUser(String id, String uid, int offset)*	*The getPostForProfileUser() method is used to get a list of posts for another user's profile. This method allows to get a list of articles from the database for another user's profile. It retrieves the list of posts from the POST and POSTSHARE tables based on the UserID and PrivacyID, and then returns a list of the corresponding Post objects.*

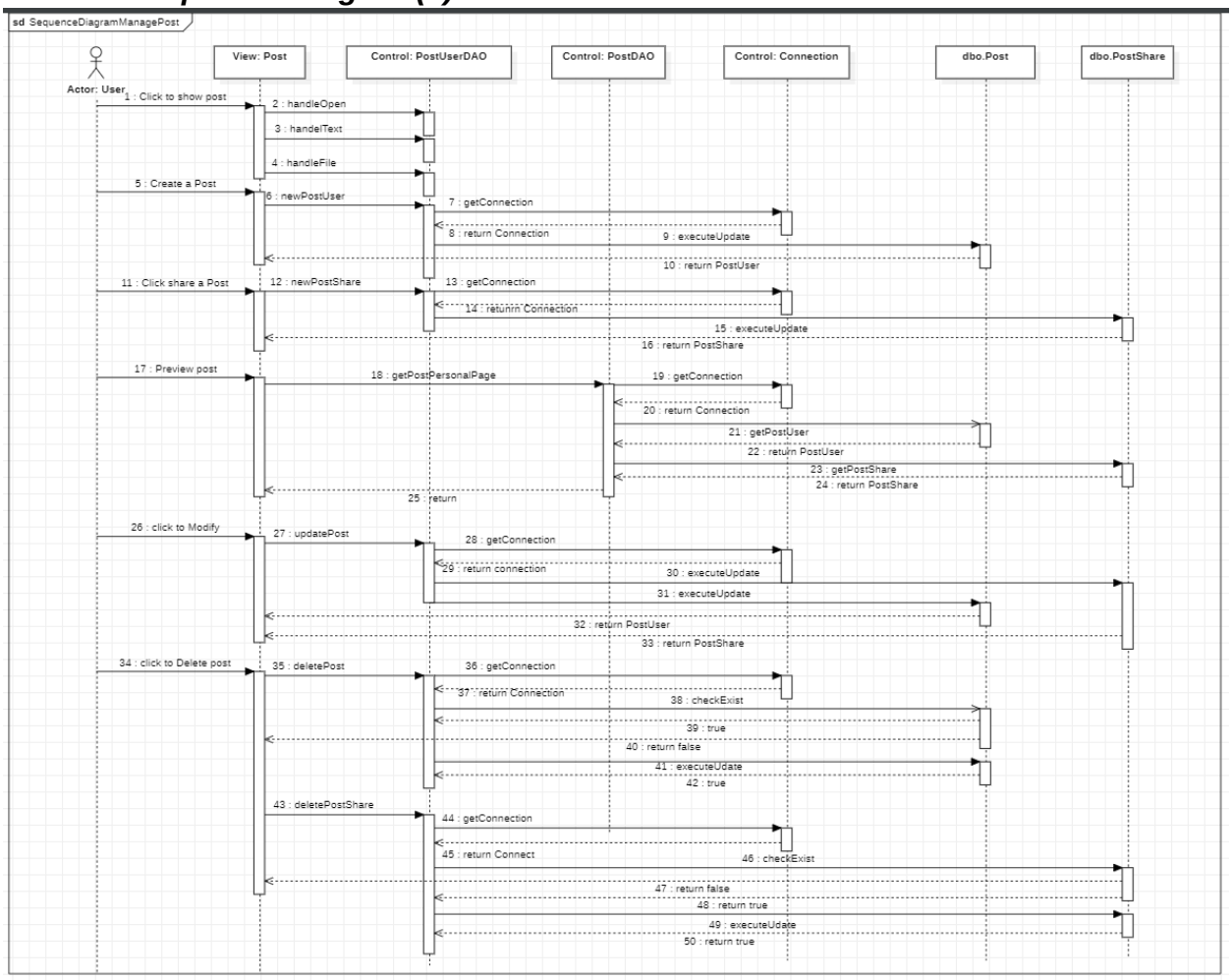
PostUserDao class

No	Method	Description
01	*PostUserDAO()*	*Creates a Connection object, and then the getConnection() method of the Connection Class is called to establish the connection to the database.*

02	*updatePost(String PostID, String UserID, String Content, String imgPost, String Public)*	*This method is used to update a post in the database based on input parameters.* *Specifically, the SQL query used to update the post is stored in the updatePost string. This query uses question mark (?) parameters to refer to the values to be updated.* *If any error occurs during processing, report the error and prevent it from spreading to the outside, by printing the stack trace of the exception.*
03	*updatePost(String PostID, String UserID, String Content, String Public)*	*In this case, the SQL query uses the updatePostWithoutImg string to update the post in the database.* *The method executes the query and updates the post in the database without resolving the ImagePost column values.* *If an error occurs during processing, report the error and print the stack trace of the exception.*
04	*createPostShare(String UserID, String PostID, String Content, String PublicPost)*	*This method is used to create a share post in the database based on the input parameters.* *In the createPostShare method, parameter values are used to assign question marks in the query.* *If an error occurs during processing, report the error and print the stack trace of the exception.*
05	*checkExistPostUser (String PostID, String UserID)*	*This method is used to check if a particular post exists in the database with a specific UserID, based on the input parameters.* *After setting the parameter values, the method executes the query and retrieves the results from the object* *If any error occurs during processing, report the error and print the stack trace of the exception. Finally, if no records are found, the method returns false to indicate that the post does not exist for the particular UserID.*
06	*checkExistPostShareUser(String PostID, String UserID)*	*This method is used to check if a particular shared post exists in the database with a specific UserID, based on the input parameters.* *After setting the parameter values, the method executes the query and retrieves the results from the ResultSet object. In the while loop, if any record is returned i.e. the share post exists with the corresponding UserID, the method returns true.*
07	*deletePostShare(String PostID, String UserID)*	*The deletePostShare method uses an SQL query to delete a shared post based on the PostID and UserID. Before deleting, the method checks the existence of the shared post by calling the checkExistPosSharetUser method. If not exists, the method returns false. If exists, the method executes the query to delete the shared post using*

		PreparedStatement and assigns the values of PostID and UserID to the query. If an error occurs, the method reports the error and prints the stack trace of the exception. Finally, if the delete is successful, the method returns true.
08	*deletePost(String PostID, String UserID)*	*Used to delete a post from the database based on the PostID and UserID parameters.* *Before performing the delete, the method checks if the post already exists with the corresponding PostID and UserID by calling checkExistPostUser. If the post does not exist, the method returns false to indicate that no posts have been deleted.*
09	*newPost(String UserID, String Content, String ImagePost, String PublicPost)*	*Use to create a new post in the database based on UserID, Content, ImagePost and PublicPost parameters.* *First, the method establishes a connection to the database through the connection.connection class. It then creates an SQL query that is stored in the newPost variable.*
10	*newPostShare(String UserID, String PostID, String Content, String PublicPost)*	*Use to create a new shared post in the database based on UserID, PostID, Content and PublicPost parameters.* *the newPostShare method first establishes a connection to the database through the connection.connection class. It then creates an SQL query that is stored in the createPostShare variable.*
11	*getPost(String PostID)*	*Used to get the details of a post based on the passed in PostID. If there is a result, the method returns a PostUser object containing the post information. If there is no result or an error occurs, the method returns null.*
12	*getAllPost(String id)*	*Use to get a list of posts by a user based on the user's id. The result is returned as an ArrayList<PostUser> containing PostUser objects representing posts.*

c. Sequence Diagram(s)



d. Database Queries

No	Queries	Description
01	*SELECT PostID, POST.UserID, Content, ImagePost, TimePost, NumInterface, NumComment, NumShare, PrivacyName, FullName, ImageUser FROM dbo.POST INNER JOIN dbo.UserInfor ON UserInfor.UserID = POST.UserID INNER JOIN dbo.Privacy ON Privacy.PrivacyID = POST.PrivacyID;*	*The statement executes a SELECT query from the POST table and joins the UserInfor and Privacy tables using join conditions. The columns selected are PostID, UserID, Content, ImagePost, TimePost, NumInterface, NumComment, NumShare, PrivacyName, FullName and ImageUser from the respective tables.*
02	*SELECT c.UserID, c.FullName, c.ImageUser,*	*The statement executes a SELECT query from the POSTSHARE table and joins the POST, UserInfor and Privacy*

	b.TimePost, b.Content, a.PostID, a.ShareID, a.UserID, d.FullName, d.ImageUser, a.Content, a.TimeShare, a.NumInterface, a.NumComment, e.PrivacyName, b.ImagePost FROM dbo.POSTSHARE a INNER JOIN dbo.POST b ON b.PostID = a.PostID INNER JOIN dbo.UserInfor c ON b.UserID = c.UserID INNER JOIN dbo.UserInfor d ON d.UserID = a.UserID INNER JOIN dbo.Privacy e ON e.PrivacyID = a.PrivacyID;	*tables using join conditions. Selected columns are UserID, FullName, ImageUser, TimePost, Content, PostID, ShareID, UserID, FullName, ImageUser, Content, TimeShare, NumInterface, NumComment, PrivacyName and ImagePost from the respective tables.*
03	*SELECT c.UserID, c.FullName, c.ImageUser, b.TimePost, b.Content, a.PostID, a.ShareID, a.UserID, d.FullName, d.ImageUser, a.Content, a.TimeShare, a.NumInterface, a.NumComment, e.PrivacyName, b.ImagePost FROM dbo.POSTSHARE a INNER JOIN dbo.POST b ON b.PostID = a.PostID INNER JOIN dbo.UserInfor c ON b.UserID = c.UserID INNER JOIN dbo.UserInfor d ON d.UserID = a.UserID INNER JOIN dbo.Privacy e ON e.PrivacyID = a.PrivacyID WHERE a.UserID = ?;*	*The statement executes a SELECT query from the linked tables and selects the corresponding columns. It gets the UserID, FullName, ImageUser from the UserInfor, TimePost, Content, PostID, ShareID, Content, TimeShare, NumInterface, NumComment, PrivacyName, ImagePost tables from the POST and POSTSHARE tables, based on the join conditions. WHERE condition a.UserID = ? is used to retrieve only articles of UserID that match the passed value.*
04	*SELECT PostID, POST.UserID, Content, ImagePost, TimePost, NumInterface, NumComment, NumShare, PrivacyName, FullName, ImageUser FROM dbo.POST INNER JOIN dbo.UserInfor ON UserInfor.UserID = POST.UserID*	*The statement executes a SELECT query from the linked tables and selects the corresponding columns. It retrieves the PostID, UserID, Content, ImagePost, TimePost, NumInterface, NumComment, NumShare, PrivacyName, FullName, ImageUser from the POST, UserInfor and Privacy tables, based on the binding conditions. Condition WHERE POST.UserID = ? is*

	INNER JOIN dbo.Privacy ON Privacy.PrivacyID = POST.PrivacyID *WHERE POST.UserID = ?;*	*used to retrieve only articles of UserID that match the passed value.*
05	*SELECT PostID, POST.UserID, Content, ImagePost, TimePost, NumInterface, NumComment, NumShare, PrivacyName, FullName, ImageUser* *FROM dbo.POST* *INNER JOIN dbo.UserInfor ON UserInfor.UserID = POST.UserID* *INNER JOIN dbo.Privacy ON Privacy.PrivacyID = POST.PrivacyID* *WHERE PostID = ?;*	*The statement executes a SELECT query from the linked tables and selects the corresponding columns. It retrieves the PostID, UserID, Content, ImagePost, TimePost, NumInterface, NumComment, NumShare, PrivacyName, FullName, ImageUser from the POST, UserInfor and Privacy tables, based on the binding conditions. Condition WHERE PostID = ? is used to retrieve only posts whose PostID matches the value passed in.*
06	*DECLARE @AdvertiserID VARCHAR(11);* *SET @AdvertiserID = (SELECT TOP (1) AdvertiserID FROM dbo.Active ORDER BY dateShow);* *UPDATE dbo.Advertisement* *SET Quantity -= 1* *WHERE AdvertiserID = @AdvertiserID;* *UPDATE dbo.Active* *SET dateShow = GETDATE()* *WHERE AdvertiserID = @AdvertiserID;* *SELECT AdvertiserID, BusinessID, Content, ImagePost, TimePost, NumInterface, NumComment, NumShare, NumOfShow, Status, Quantity* *FROM dbo.Advertisement* *WHERE AdvertiserID = @AdvertiserID;*	*The command does the following:* ● *Declare variable @AdvertiserID with data type VARCHAR(11).* ● *Assign the value to the @AdvertiserID variable with the first AdvertiserID in the Active table sorted by the dateShow field.* ● *Update the Quantity field in the Advertisement table by 1 for the record whose AdvertiserID matches the value of the @AdvertiserID variable.* ● *Update the dateShow field in the Active table with the current time value (GETDATE()) for the record whose AdvertiserID matches the value of the @AdvertiserID variable.* ● *Query and get the AdvertiserID, BusinessID, Content, ImagePost, TimePost, NumInterface, NumComment, NumShare, NumOfShow, Status, Quantity columns from the Advertisement table, retrieving only the records where the AdvertiserID matches*

		the value of the @AdvertiserID variable.
07	*SELECT c.UserID, c.FullName, c.ImageUser, b.TimePost, b.Content, a.PostID, a.ShareID, a.UserID, d.FullName, d.ImageUser, a.Content, a.TimeShare, a.NumInterface, a.NumComment, e.PrivacyName, b.ImagePost FROM dbo.POSTSHARE a INNER JOIN dbo.POST b ON b.PostID = a.PostID INNER JOIN dbo.UserInfor c ON b.UserID = c.UserID INNER JOIN dbo.UserInfor d ON d.UserID = a.UserID INNER JOIN dbo.Privacy e ON e.PrivacyID = a.PrivacyID WHERE a.ShareID = ?;*	*The statement queries data from the POSTSHARE, POST, UserInfor, and Privacy tables. It gets the UserID, FullName, ImageUser columns from the UserInfor table, the same TimePost, Content, PostID, ShareID, UserID columns from the POST table, the same FullName, ImageUser, Content, TimeShare, NumInterface, NumComment, PrivacyName, ImagePost columns from the POSTSHARE table . The tables are connected to each other via ON conditions to get information related to a particular ShareID.*
08	*DECLARE @id NVARCHAR(11) = ?; DECLARE @Offset INT = ?; DECLARE @FetchCount INT = 10; DECLARE @table TABLE (FriendID NVARCHAR(11)); INSERT INTO @table (FriendID) VALUES (@id); INSERT INTO @table (FriendID) SELECT CASE WHEN UserID1 = @id THEN UserID2 WHEN UserID2 = @id THEN UserID1 END AS FriendID FROM dbo.USERRELATION WHERE (UserID1 = @id OR UserID2 = @id) AND isFriend = 1; SELECT PostID, TimePost, PrivacyID FROM dbo.POST*	*The statement executes the query for data from the POST and POSTSHARE tables. Before executing the query, it creates a temporary table @table and inserts the @id value into the clipboard. It then inserts the corresponding FriendID values into the clipboard from the USERRELATION table based on the condition that UserID1 or UserID2 equals @id and isFriend equals 1. Finally, the query statement retrieves the PostID, TimePost, and PrivacyID columns from the POST table when the corresponding UserID in the POST table matches the FriendID in the @table clipboard and has a PrivacyID of 'FRIEND' or 'Public'. Similarly, it retrieves the ShareID, TimeShare and PrivacyID columns from the POSTSHARE table under the same conditions. The results are sorted by the TimePost field in descending order and only the number of posts by the given offset and fetch count is taken.*

	INNER JOIN @table ON UserID = FriendID *WHERE (PrivacyID = 'FRIEND' OR PrivacyID = 'Public')* *UNION ALL* *SELECT ShareID, TimeShare, PrivacyID* *FROM dbo.POSTSHARE* *INNER JOIN @table ON UserID = FriendID* *WHERE (PrivacyID = 'FRIEND' OR PrivacyID = 'Public')* *ORDER BY TimePost DESC* *OFFSET (@Offset-1) * @FetchCount ROWS* *FETCH NEXT @FetchCount ROWS ONLY;*	
09	*DECLARE @id NVARCHAR(11) = ?;* *DECLARE @Offset INT = ?;* *DECLARE @FetchCount INT = 5;* *SELECT PostID, TimePost, PrivacyID* *FROM dbo.POST* *WHERE UserID = @id* *UNION ALL* *SELECT ShareID, TimeShare, PrivacyID* *FROM dbo.POSTSHARE* *WHERE UserID = @id* *ORDER BY TimePost DESC* *OFFSET (@Offset-1) * @FetchCount ROWS* *FETCH NEXT @FetchCount ROWS ONLY;*	*The statement executes the query for data from the POST and POSTSHARE tables. It retrieves the PostID, TimePost and PrivacyID columns from the POST table when the UserID matches the @id value. Similarly, it retrieves the ShareID, TimeShare and PrivacyID columns from the POSTSHARE table under the same condition. The results are sorted by the TimePost field in descending order and only the number of posts by the given offset and fetch count is taken.*
10	*DECLARE @id NVARCHAR(11) = ?;* *DECLARE @uid NVARCHAR(11) = ?;* *DECLARE @u1 NVARCHAR(11), @u2 NVARCHAR(11), @isFriend BIT;* *DECLARE @Offset INT = ?;* *DECLARE @FetchCount INT = 5;*	*The statement executes the query for data from the POST and POSTSHARE tables. Based on the values of @id and @uid, it determines the values @u1 and @u2 to sort the order of the two UserIDs. It then checks if the two UserIDs are friends by querying the USERRELATION table. Based on this result, the statement returns posts under the right conditions: if friends,*

	IF (@id > @uid) *BEGIN* *SET @u1 = @id;* *SET @u2 = @uid;* *END* *ELSE* *BEGIN* *SET @u2 = @id;* *SET @u1 = @uid;* *END* *SET @isFriend = (SELECT isFriend* *FROM dbo.USERRELATION* *WHERE UserID1 = @u1 AND UserID2 = @u2);* *IF (@isFriend = 1)* *BEGIN* *SELECT PostID, TimePost, PrivacyID* *FROM dbo.POST* *WHERE UserID = @uid AND (PrivacyID = 'FRIEND' OR PrivacyID = 'Public')* *UNION ALL* *SELECT ShareID, TimeShare, PrivacyID* *FROM dbo.POSTSHARE* *WHERE UserID = @uid AND (PrivacyID = 'FRIEND' OR PrivacyID = 'Public')* *ORDER BY TimePost DESC* *OFFSET (@Offset - 1) * @FetchCount ROWS* *FETCH NEXT @FetchCount ROWS ONLY;* *END* *ELSE* *BEGIN* *SELECT PostID, TimePost, PrivacyID* *FROM dbo.POST* *WHERE UserID = @uid AND (PrivacyID = 'Public')* *UNION ALL*	*return posts with PrivacyID of 'FRIEND' or 'Public'; if not friends, return only posts with PrivacyID of 'Public'. The results are sorted by the TimePost field in descending order and only the number of posts by the given offset and fetch count is taken.*

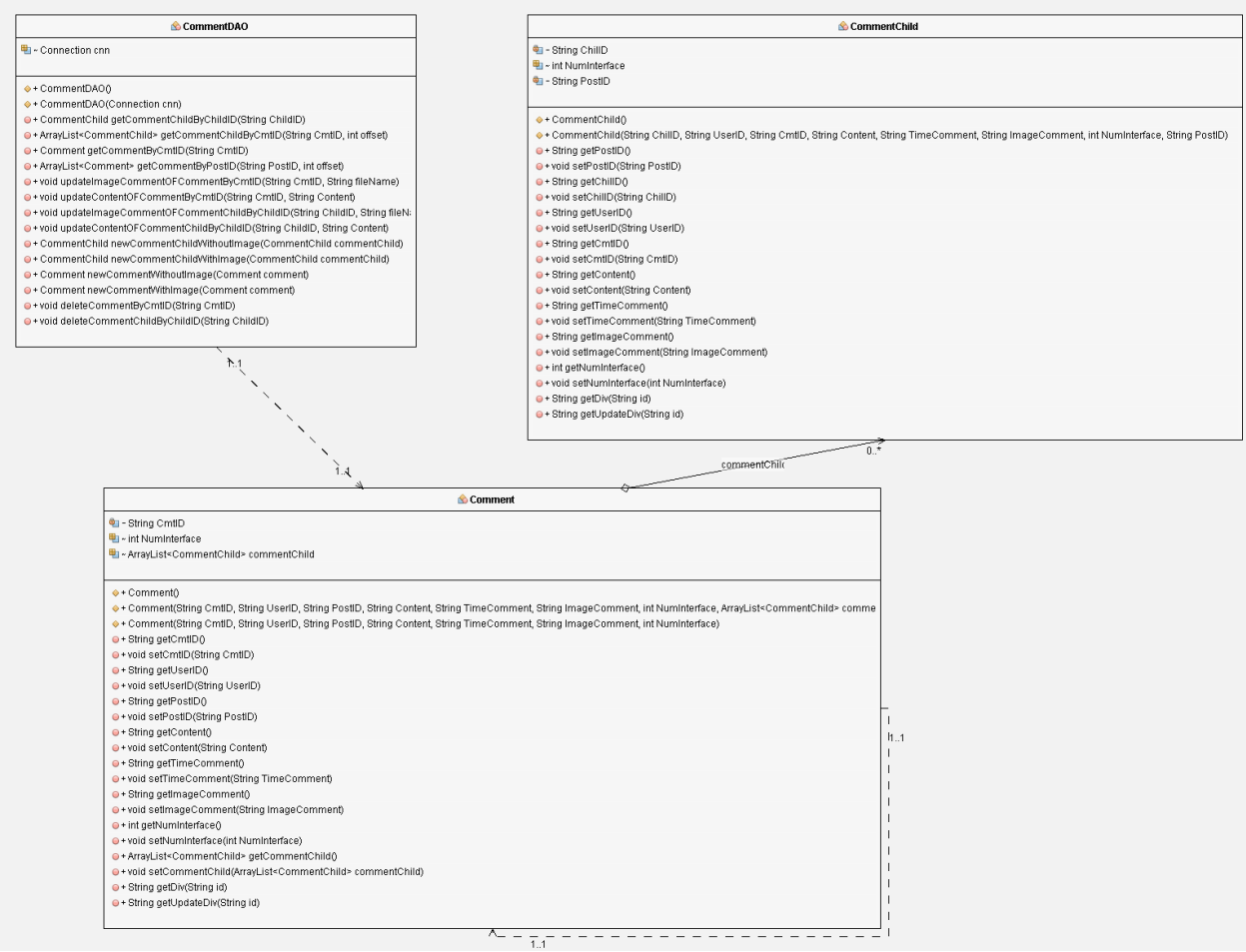
	SELECT ShareID, TimeShare, PrivacyID FROM dbo.POSTSHARE WHERE UserID = @uid AND (PrivacyID = 'Public') ORDER BY TimePost DESC OFFSET (@Offset - 1) * @FetchCount ROWS FETCH NEXT @FetchCount ROWS ONLY; END	
11	SELECT * FROM dbo.POSTSHARE WHERE ShareID= ? AND UserID= ?	The following SQL query is used to get the information of a shared post from the POSTSHARE table based on the ShareID and UserID
12	DECLARE @InsertedIDs TABLE (PostID VARCHAR(11)); INSERT INTO dbo.POST (UserID, Content, ImagePost, TimePost, NumInterface, NumComment, NumShare, PublicPost)\n" OUTPUT Inserted.PostID INTO @InsertedIDs VALUES (? , -- UserID - varchar(11) ? , -- Content - nvarchar(max) ? , -- ImagePost - nvarchar(255) DEFAULT, -- TimePost - datetime DEFAULT, -- NumInterface - int DEFAULT, -- NumComment - int DEFAULT, -- NumShare - int	SQL snippet to create a new post in POST table.

	? -- PublicPost - bit);\n" SELECT PostID FROM @InsertedIDs;	
13	SELECT PostID, UserID, Content, ImagePost, TimePost, NumInterface, NumComment, NumShare, PublicPost FROM dbo.POST WHERE PostID= ?	SQL query to get information of a post from POST table based on PostID.
14	DECLARE @InsertedIDs TABLE (ShareID VARCHAR(11)); INSERT INTO dbo.POSTSHARE (UserID, PostID, Content, TimeShare, NumInterface, NumComment, PublicPost) OUTPUT Inserted.ShareID INTO @InsertedIDs VALUES (? , -- UserID - varchar(11) ? , -- PostID - varchar(11) ? , -- Content - nvarchar(max) DEFAULT, -- TimeShare - datetime DEFAULT, -- NumInterface - int DEFAULT, -- NumComment - int ? -- PublicPost - bit) SELECT ShareID FROM @InsertedIDs;	An SQL query to add a new shared post to the POSTSHARE table and get the ShareID of the newly added post.

15	*SELECT * FROM dbo.POST WHERE PostID = ? AND UserID = ?;*	*This statement queries the POST table to get the information of a specific post based on the PostID and UserID. The results returned are all columns of that post.*
16	*UPDATE POST SET Content = ? WHERE PostID = ?;*	*This statement updates the content of the post whose PostID is the specified value. The new content value is set equal to the value specified in the statement.*
17	*UPDATE POST SET Content = ?, ImagePost = ? WHERE PostID = ?;*	*This statement updates the content (Content) and image (ImagePost) of the post whose PostID is the specified value. The new image and content values are set equal to the value specified in the statement.*

6. Manage Comment

a. Class Diagram



b. Class Specifications

Comment Class

No	Method	Description
01	*Comment()*	*The constructor is used to initialize the object of a class and perform initialization work. In this case, the empty constructor specifies that when a Comment object is created, no specific information is passed to it.*
02	*Comment(String CmtID, String UserID, String PostID, String Content, String TimeComment, String ImageComment, int NumInterface, ArrayList<CommentChild> commentChild)*	*In the constructor, the values passed in the parameters will be assigned to the corresponding properties of the Comment object. Also, if ImageComment is not empty, the path will be adjusted to point to the resource in the application.*
03	*Comment(String CmtID, String UserID, String PostID, String Content, String TimeComment, String ImageComment, int NumInterface)*	*In the constructor, the values passed in the parameters will be assigned to the corresponding properties of the Comment object. If ImageComment is not empty, the path will be adjusted to point to the resource in the application.*
04	*Getter; Setter()*	*Getter and Setter methods are used to access and set the value of properties in a class.*
05	*getDiv()*	*This getDiv() method is used to generate the HTML string to display the share post in the UI.*
06	*getUpdateDiv(String id)*	*The getUpdateDiv method is used to generate an HTML string representing a Comment object in the user interface.* *Information about the comment's image, the user who created the comment, the number of interactions, and the comment time are also obtained and used to generate the HTML string. User image and comment image (if any) are displayed. The "Delete", "Modify" and "Report" actions have a click event to perform their respective functions.*

CommentChild Class

No	Method	Description
01	*CommentChild()*	*The default constructor of the CommentChild class is defined to create a new CommentChild object. This method takes no parameters and has no internal function body. The purpose of this method is to initialize a CommentChild object with default properties or initial state defined in the class member variables.*
02	*CommentChild(String ChilID, String UserID, String CmtID, String Content, String TimeComment, String ImageComment, int NumInterface, String PostID)*	*The constructor of the CommentChild class has a parameter defined to create a CommentChild object with specific values for the properties.*
03	*Getter; Setter()*	*Getter and Setter methods are used to access and set the value of properties in a class.*
04	*getDiv(String id)*	*The getDiv method is defined to generate an HTML snippet representing the CommentChild object. This method takes an id parameter to identify the current user.*
05	*getUpdateDiv(String id)*	*The getUpdateDiv method is defined to generate an HTML snippet representing the CommentChild object in case an update is needed. This method takes an id parameter to identify the current user.*

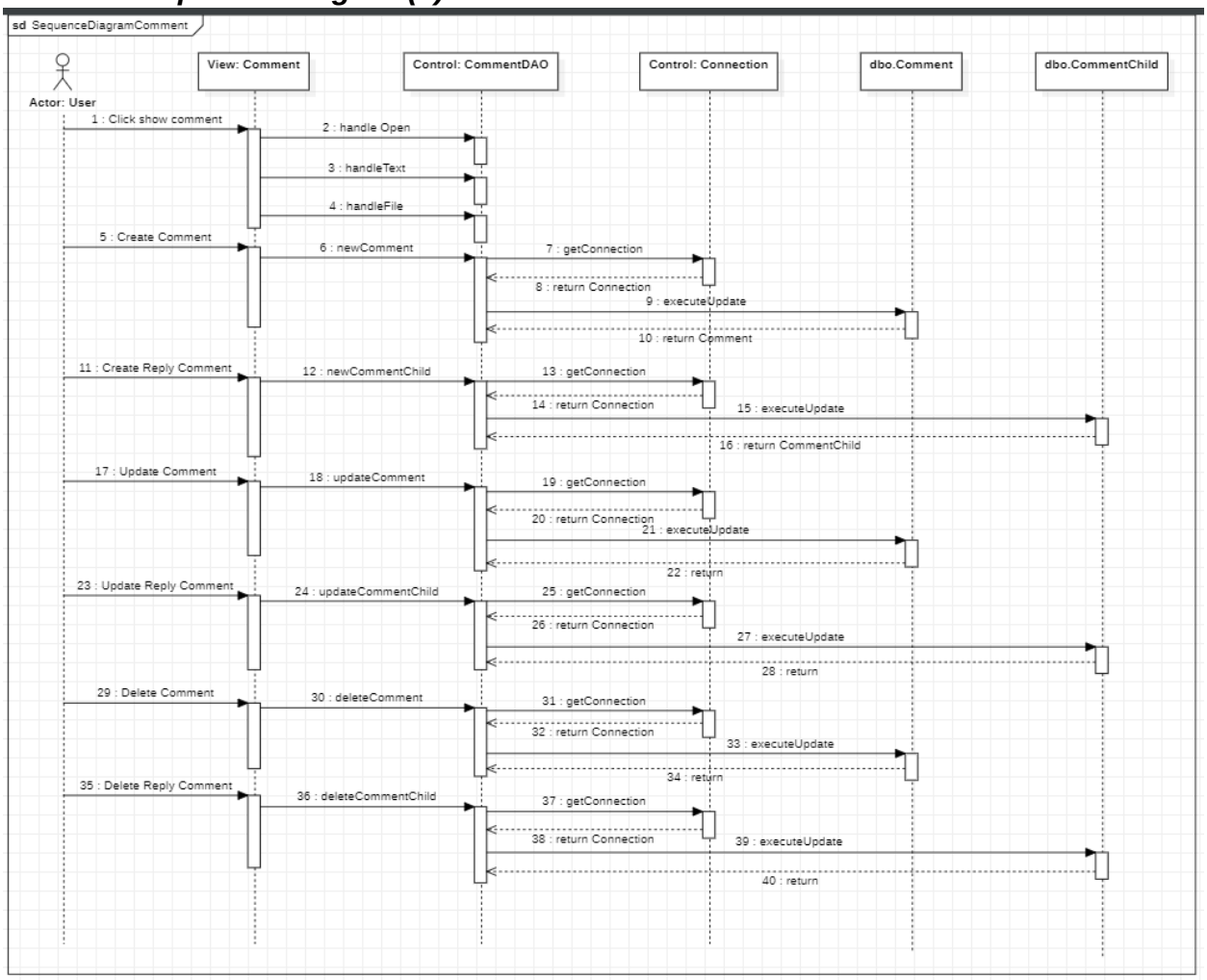
CommentDAO Class

No	Method	Description
01	*CommentDAO()*	*The CommentDAO constructor is defined to instantiate a CommentDAO object. In this constructor, a connection to the database is established by calling the getConnection() method from the connection.connection class.*
02	*CommentDAO(Connection cnn)*	*The CommentDAO constructor takes a cnn parameter of type Connection, which is used to establish the connection to the database.*
03	*getCommentChildByChildID(String ChildID)*	*The getCommentChildByChildID method in the CommentDAO class has the function to get the information of a CommentChild based on the ChildID from the database.*
04	*getCommentChildByCmtID(String CmtID, int offset)*	*The getCommentChildByCmtID method in the CommentDAO class has the function to get a list of CommentChild based on CmtID from the database.*

		The function of the method is to get a list of CommentChilds based on the CmtID to display, process or transmit data to other layers in the application.
05	*getCommentByCmtID(String CmtID)*	*The getCommentByCmtID method in the CommentDAO class takes a CmtID parameter of type String and returns a Comment object.* *In the above example, cnn is a previously established database connection object, and cmtID is the value of the CmtID to be queried.*
06	*getCommentByPostID(String PostID, int offset)*	*The getCommentByPostID method in the CommentDAO class takes a String PostID and an int offset, and returns an ArrayList<Comment> list.* *The getCommentByPostID method in the CommentDAO class is used to get the list of Comments based on the PostID and offset.*
07	*updateImageCommentOFCommentByCmtID(String CmtID, String fileName)*	*The updateImageCommentOFCommentByCmtID method in the CommentDAO class updates the Comment's ImageComment field based on the CmtID in the database.* *The function of the method is to update the ImageComment field of the Comment in the database. This update can be used to change the image associated with the Comment.*
08	*updateContentOFCommentByCmtID(String CmtID, String Content)*	*The updateContentOFCommentByCmtID method in the CommentDAO class has the function to update the Content field of the Comment based on the CmtID in the database.* *The function of the method is to update the Content field of the Comment in the database. This update can be used to edit the content of the Comment.*
09	*updateImageCommentOFCommentChildByChildID(String ChildID, String fileName)*	*The updateImageCommentOFCommentChildByChildID method in the CommentDAO class updates the CommentChild's ImageComment field based on the ChildID in the database.* *The function of the method is to update the ImageComment field of the CommentChild in the database. This update can be used to change CommentChild's attached image.*
10	*updateContentOFCommentChildByChildID(String ChildID, String Content)*	*The updateContentOFCommentChildByChildID method in the CommentDAO class updates the CommentChild's Content field based on the ChildID in the database.* *The function of the method is to update the Content field of the CommentChild in the database. This update can be used to edit the contents of the CommentChild.*

11	*newCommentChildWithoutImage(CommentChild commentChild)*	*The newCommentChildWithoutImage method in the CommentDAO class creates a new CommentChild in the database without the image.* *The function of the method is to create a new CommentChild in the database without the image. This CommentChild is inserted into the COMMENTCHILD table and then queried from the database to get the complete CommentChild information.*
12	*newCommentChildWithImage(CommentChild commentChild)*	*The newCommentChildWithImage method in the CommentDAO class has the function to create a new CommentChild in the database with the image attached.* *The function of the method is to create a new CommentChild in the database with the image. This CommentChild is inserted into the COMMENTCHILD table and then queried from the database to get the complete CommentChild information.*
13	*newCommentWithoutImage(Comment comment)*	*The newCommentWithoutImage method in the CommentDAO class has the function to create a new Comment in the database without an image.* *The function of the method is to create a new Comment in the database without the image. This comment is inserted into the COMMENT table and then queried from the database to get the complete comment information.*
14	*newCommentWithImage(Comment comment)*	*The newCommentWithImage method in the CommentDAO class has the function to create a new Comment in the database with the image.* *The function of the method is to create a new Comment in the database with the image. This comment is inserted into the COMMENT table and then queried from the database to get the complete comment information.*
15	*deleteCommentByCmtID(String CmtID)*	*The deleteCommentByCmtID method in the CommentDAO class has the function to delete a Comment based on the CmtID.* *The function of the method is to delete a Comment from the database based on CmtID. Comments are removed from the COMMENT panel.*
16	*deleteCommentChildByChildID(String ChildID)*	*The deleteCommentChildByChildID(String ChildID) method in the Dao package's CommentDAO class is used to delete a CommentChild based on the ChildID.*

c. Sequence Diagram(s)



d. Database Queries

No	Queries	Description
01	*SELECT ChildID, UserID, CmtID, Content, TimeComment, ImageComment, NumInterface, (SELECT TOP(1) PostID FROM dbo.COMMENTCHILD INNER JOIN dbo.COMMENT ON COMMENT.CmtID = COMMENTCHILD.CmtID WHERE ChildID = ?) AS PostID FROM dbo.COMMENTCHILD WHERE ChildID = ?*	*Gets CommentChild information based on ChildID. The query returns the columns ChildID, UserID, CmtID, Content, TimeComment, ImageComment, NumInterface, and PostID. PostID is obtained from COMMENTCHILD and COMMENT by combining the tables and the WHERE ChildID = ? condition.*
02	*DECLARE @CmtID VARCHAR(11) = ?;*	*Get a list of CommentChild based on CmtID and do pagination. The query returns the columns ChildID, UserID,*

	*DECLARE @Offset INT = ?; -- Số bài post đã hiển thị trước đó = @FetchCount * (offset-1)* *DECLARE @FetchCount INT = 5; -- Số bài post muốn lấy thêm* *SELECT ChildID, UserID, CmtID, Content, TimeComment, ImageComment, NumInterface,* *(SELECT TOP(1) PostID* *FROM dbo.COMMENTCHILD* *INNER JOIN dbo.COMMENT* *ON COMMENT.CmtID = COMMENTCHILD.CmtID* *WHERE ChildID = 'ILD00000009') AS PostID* *FROM dbo.COMMENTCHILD* *WHERE CmtID = @CmtID* *ORDER BY TimeComment DESC* *OFFSET (@Offset - 1) * @FetchCount ROWS* *FETCH NEXT @FetchCount ROWS ONLY;* *SELECT ** *FROM dbo.COMMENTCHILD;*	*CmtID, Content, TimeComment, ImageComment, NumInterface and PostID (taken from COMMENTCHILD and COMMENT). The posts are sorted in descending chronological order and take the posts from (@Offset - 1) * @FetchCount to (@Offset - 1) * @FetchCount + @FetchCount.*
03	*SELECT CmtID, UserID, PostID, Content, TimeComment, ImageComment, NumInterface* *FROM dbo.COMMENT* *WHERE CmtID = ?;*	*Get Comment information based on CmtID. The query returns columns CmtID, UserID, PostID, Content, TimeComment, ImageComment, NumInterface from the COMMENT table.*
04	*DECLARE @PostID VARCHAR(11) = ?;* *DECLARE @Offset INT = ?; -- Số bài post đã hiển thị trước đó = @FetchCount * (offset-1)* *DECLARE @FetchCount INT = 5; -- Số bài post muốn lấy thêm* *SELECT CmtID, UserID, PostID, Content, TimeComment, ImageComment, NumInterface* *FROM dbo.COMMENT* *WHERE PostID = @PostID* *ORDER BY TimeComment DESC*	*Get a list of Comments based on PostID and do pagination. The query returns columns CmtID, UserID, PostID, Content, TimeComment, ImageComment, NumInterface from the COMMENT table. The posts are sorted in descending chronological order and take the posts from (@Offset - 1) * @FetchCount to (@Offset - 1) * @FetchCount + @FetchCount.*

	*OFFSET (@Offset - 1) * @FetchCount ROWS FETCH NEXT @FetchCount ROWS ONLY;*	
05	*DECLARE @CmtId NVARCHAR(11) = ?; DECLARE @ImageComment VARCHAR(255) = ?;* *UPDATE dbo.COMMENT SET ImageComment = @ImageComment WHERE CmtID = @CmtId;*	*Update the Comment's ImageComment field based on the CmtID. The query performs updating ImageComment = @ImageComment for records with CmtID = @CmtId in the COMMENT table.*
06	*DECLARE @CmtId NVARCHAR(11) = ?; DECLARE @Content NVARCHAR(MAX) = ?;* *UPDATE dbo.COMMENT SET Content = @Content WHERE CmtID = @CmtId;*	*Update the Comment's Content field based on the CmtID. The query performs Content = @Content update for the record with CmtID = @CmtId in the COMMENT table.*
07	*DECLARE @ChildId NVARCHAR(11) = ?; DECLARE @ImageComment VARCHAR(255) = ?;* *UPDATE dbo.COMMENTCHILD SET ImageComment = @ImageComment WHERE ChildID = @ChildId;*	*Update the CommentChild's ImageComment field based on the ChildID. The query performs an update of ImageComment = @ImageComment for records with ChildID = @ChildId in the COMMENTCHILD table.*
08	*DECLARE @ChildId NVARCHAR(11) = ?; DECLARE @Content VARCHAR(255) = ?;* *UPDATE dbo.COMMENTCHILD SET Content = @Content WHERE ChildID = @ChildId;*	*Update the CommentChild's Content field based on the ChildID. The query performs an update of Content = @Content for records with ChildID = @ChildId in the COMMENTCHILD table.*

09	*DECLARE @InsertedIDs TABLE (ChildID VARCHAR(11));* *INSERT INTO dbo.COMMENTCHILD (UserID, CmtID, Content, TimeComment, ImageComment, NumInterface) OUTPUT Inserted.ChildID INTO @InsertedIDs VALUES (?, ?, ?, DEFAULT, '', DEFAULT);* *SELECT ChildID FROM @InsertedIDs;*	*Add a new CommentChild and return the ChildID. The query inserts a new record into the COMMENTCHILD table with the corresponding UserID, CmtID, Content, TimeComment, ImageComment and NumInterface values. Then return the ChildID of the inserted record.*
10	*DECLARE @InsertedIDs TABLE (ChildID VARCHAR(11));* *INSERT INTO dbo.COMMENTCHILD (UserID, CmtID, Content, TimeComment, ImageComment, NumInterface) OUTPUT Inserted.ChildID INTO @InsertedIDs VALUES (?, ?, ?, DEFAULT, ?, DEFAULT);* *SELECT ChildID FROM @InsertedIDs;*	*Add new CommentChild with ImageComment and return ChildID. The query inserts a new record into the COMMENTCHILD table with the corresponding UserID, CmtID, Content, TimeComment, ImageComment and NumInterface values. Then return the ChildID of the inserted record.*
11	*DELETE FROM dbo.COMMENT WHERE CmtID = ?;*	*Delete Comments based on CmtID. The query that deletes records with CmtID = ? in the COMMENT panel.*
12	*DELETE FROM dbo.COMMENTCHILD WHERE ChildID = ?;*	*Remove CommentChild based on ChildID. The query that deletes records with ChildID = ? in the COMMENTCHILD table.*

7. Report User

a. Class Diagram

b. Class Specifications

UserReport

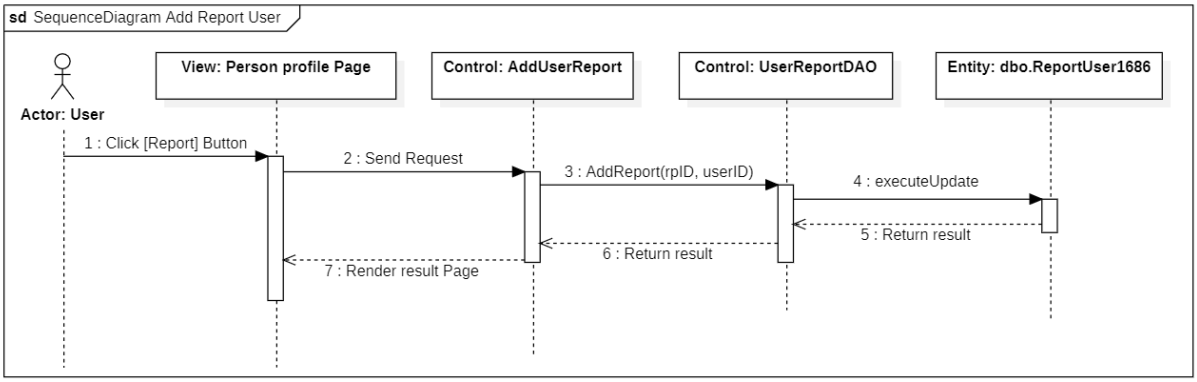
No	Method	Description
1	*UserReport()*	*Default constructor of UserReport . class*
2	*UserReport(String userID, String imageUser, String fullName, String account, String mail, String phoneNumber, String address, int numCommentReported, int numPostReported)*	*Constructor with parameters to initialize the UserReport object*
3	*getters and setters*	*Getter and setter methods for properties of UserReport . class*

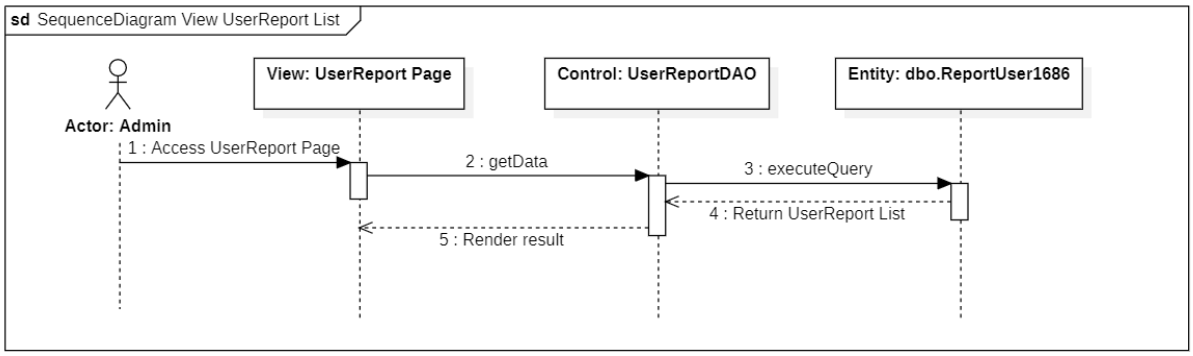
UserReportDAO

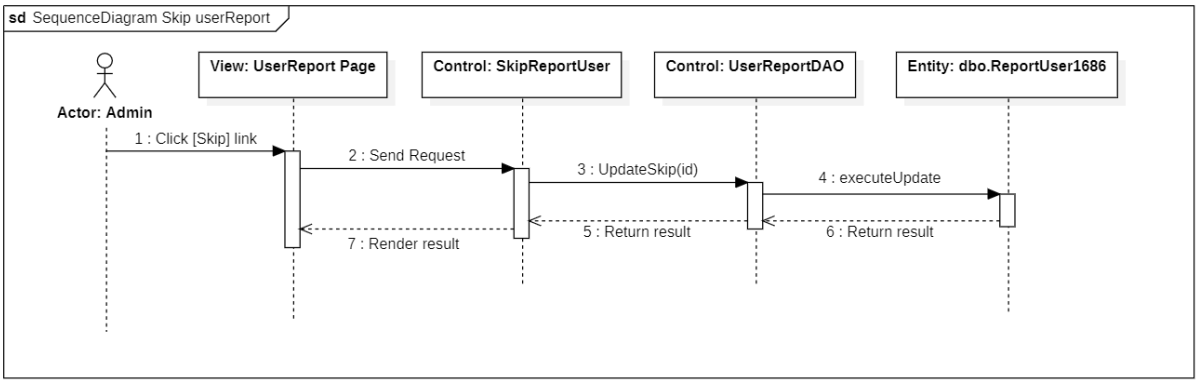
No	Method	Description
1	*UserReportDAO()*	*Constructor: Initializes UserReportDAO object and establishes connection*
2	*getData()*	*Get list of UserReports from database*
3	*UpdateSkip(String id)*	*Update "Skip" status for UserReport based on ID. Reported users are not affected*

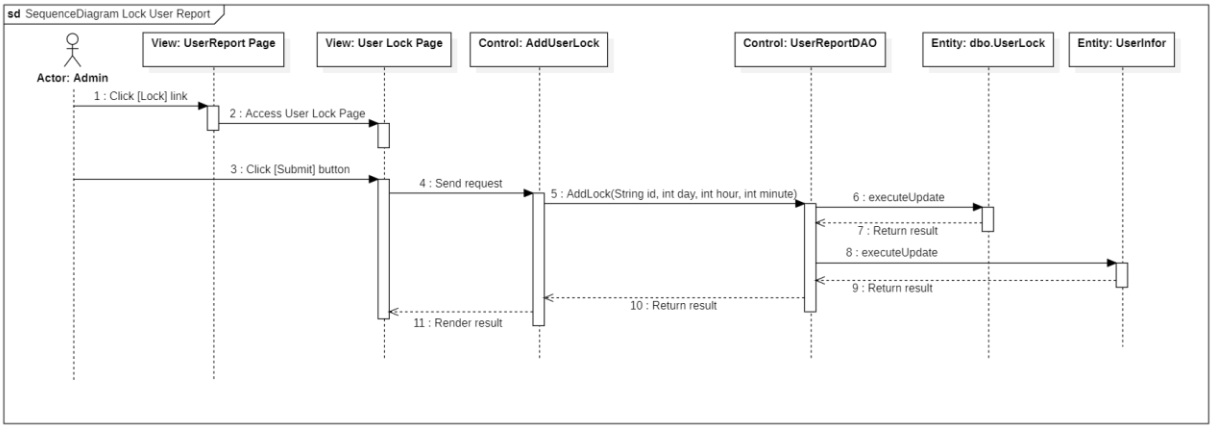
4	*AddReport(String urpid, String uid)*	*Add a new report to the ReportUser1686 table with UserID (Reported User) and UserIDRP (Reporter) information*
5	*AddLock(String id, int day, int hour, int minute)*	*Add a new record to the UserLock table to lock the User account for a specified period of time (Days - Hours - Minutes)*
6	*DeleteUser(String id)*	*Update the "Deleted" status for the User account in the database*

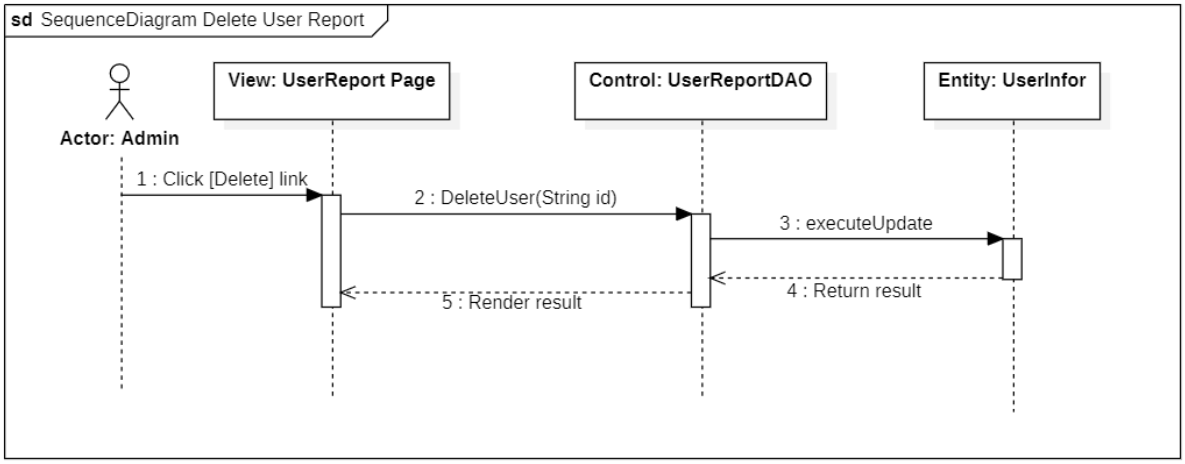
c. Sequence Diagram(s)











d. Database Queries

No	Queries	Description
01	*SELECT * FROM UserReportSummary*	*Query to get all unrated reported users from the "UserReportSummary" view.*
02	*UPDATE dbo.ReportUser1686 SET ReportUser1686.Status=0 WHERE ReportUser1686.UserID=?*	*Update the "Status" column value to 0 (Evaluated) in the "ReportUser1686" table for the User whose "UserID" corresponds to the value replaced by the "?"*
03	*INSERT INTO dbo.ReportUser1686 (UserID, UserIDRP, Status) VALUES (?, ?, 1)*	*Add a User to the "ReportUser1686" table with the corresponding values replaced by a "?" when that user is reported*
04	*INSERT INTO dbo.UserLock (UserID, LockTime, LockDurationDay, LockDurationHour,*	*Add a User to the "UserLock" table with the Date, Hour, and Minute values replaced by a "?" is the amount of time the User will not be able to log in*

	LockDurationMinute) VALUES (?, GETDATE(), ?, ?, ?)	
05	*UPDATE dbo.UserInfor SET UserInfor.RoleID = 'LOCK' WHERE UserID = ?*	*Update the "RoleID" column value to 'LOCK' in the "UserInfor" table for the record where "UserID" corresponds to the value replaced by the "?" to mark that user as account locked*
06	*UPDATE dbo.UserInfor SET UserInfor.RoleID = 'DELETED' WHERE UserID = ?*	*Update the "RoleID" column value to 'DELETED' in the "UserInfor" table for the record where "UserID" corresponds to the value replaced by the "?" to mark that user as Account Deleted and can't log in anymore*

8. Report Comment

a. Class Diagram

b. Class Specifications

CommentReport

No	Method	Description
01	*CommentReport(String commentID, boolean isPost, String img, String content, int reportCount, Date time)*	*The constructor to create a CommentReport object with information such as commentID, isPost, img, content, reportCount, time.*
02	*getters and setters*	*Getter and setter methods for properties of class CommentReport*

CommentReportDAO

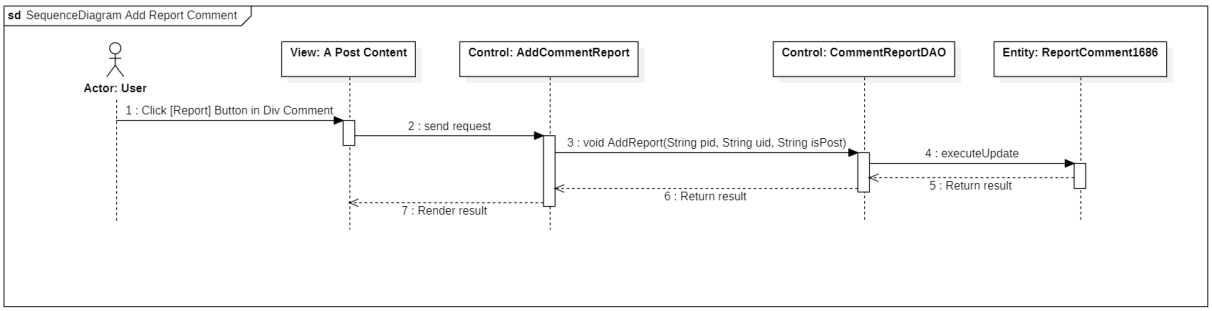
No	Method	Description
01	*CommentReportDAO()*	*This is the constructor of the class that establishes a connection to the database.*
02	*getData()*	*This method retrieves a list of CommentReport objects from the database. It executes a SQL query to fetch the required data from the ReportCommentView table. The retrieved data is used to create CommentReport objects, which are then added to the list and returned.*
03	*UpdateSkip(String id, boolean isPost)*	*This method updates the Status field of the ReportComment1686 table to mark a comment report as skipped. It takes the comment ID and a boolean flag indicating whether the reported item is a post or not. The method executes an SQL update statement to modify the corresponding row in the table.*
04	*AddReport(String pid, String uid, String isPost)*	*This method adds a new comment report to the ReportComment1686 table. It takes the comment ID, user ID, and a string representation of the isPost flag. The method converts the isPost flag to a boolean value and then executes an SQL insert statement to insert a new row in the table with the provided data.*

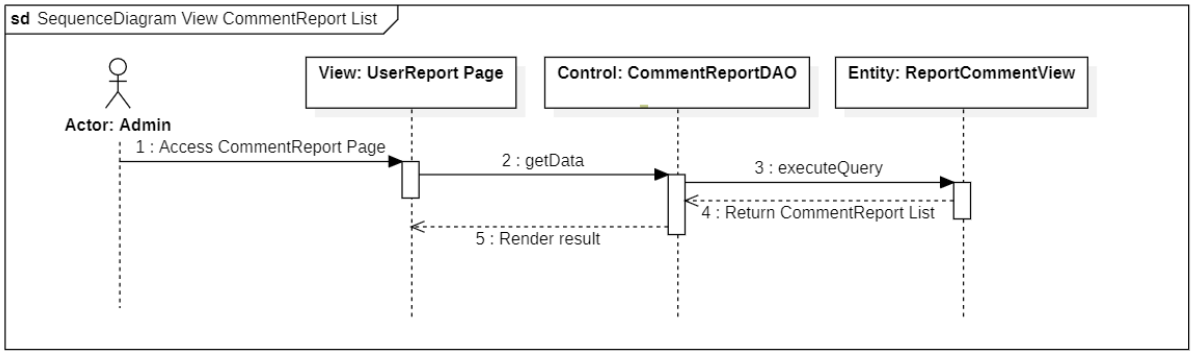
CommentDAO

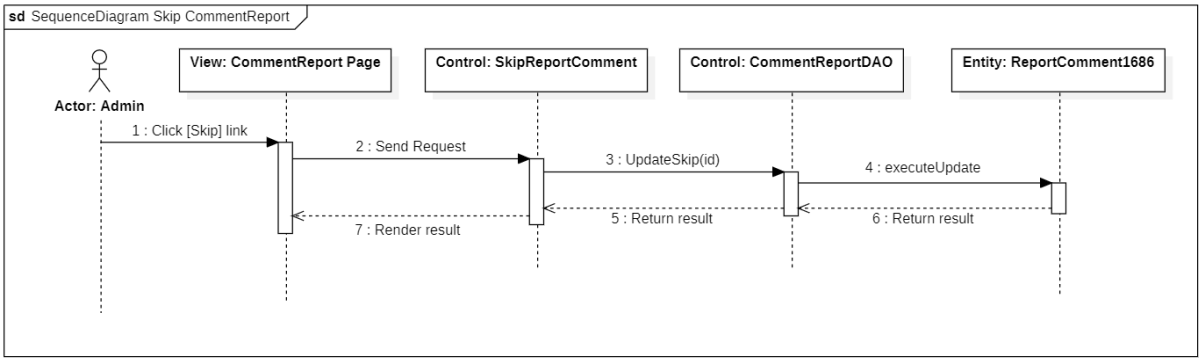
No	Method	Description
01	*CommentDAO()*	*Constructor mặc định khởi tạo một đối tượng CommentDAO và thiết lập kết nối đến cơ sở dữ liệu bằng cách gọi phương thức getConnection() từ lớp connection.connection.*
02	*deleteCommentByCmtID(String CmtID)*	*Phương thức này xóa bình luận dựa trên CmtID được truyền vào. Nó thực hiện câu truy vấn SQL*

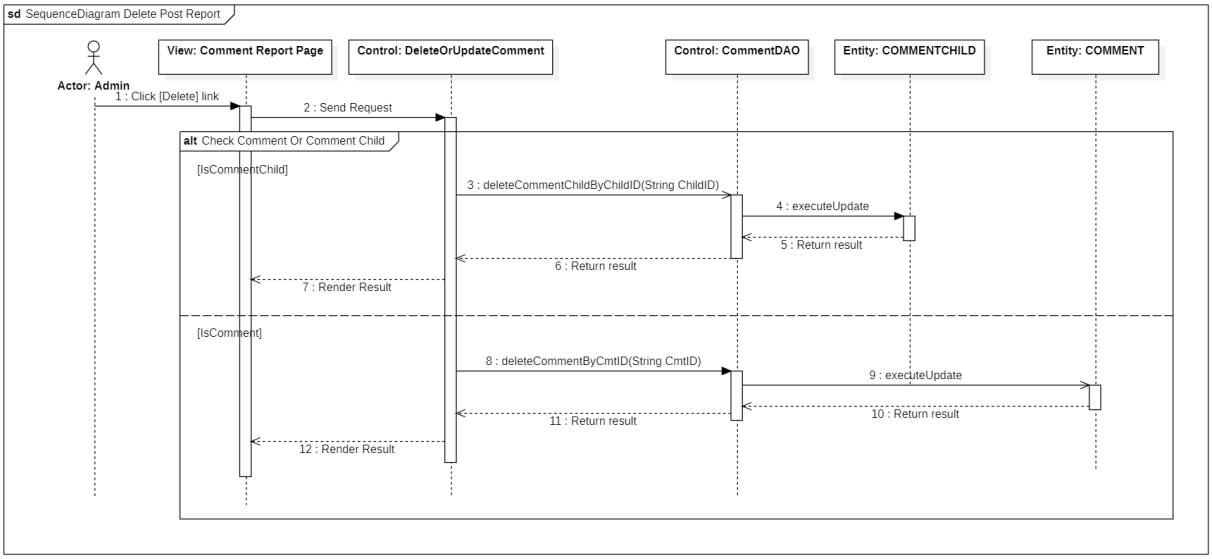
		DELETE để xóa bình luận tương ứng trong cơ sở dữ liệu.
03	*deleteCommentChildBy ChildID(String ChildID)*	*Phương thức này xóa bình luận con (comment child) dựa trên ChildID được truyền vào. Nó thực hiện câu truy vấn SQL DELETE để xóa bình luận con tương ứng trong cơ sở dữ liệu.*

c. Sequence Diagram(s)









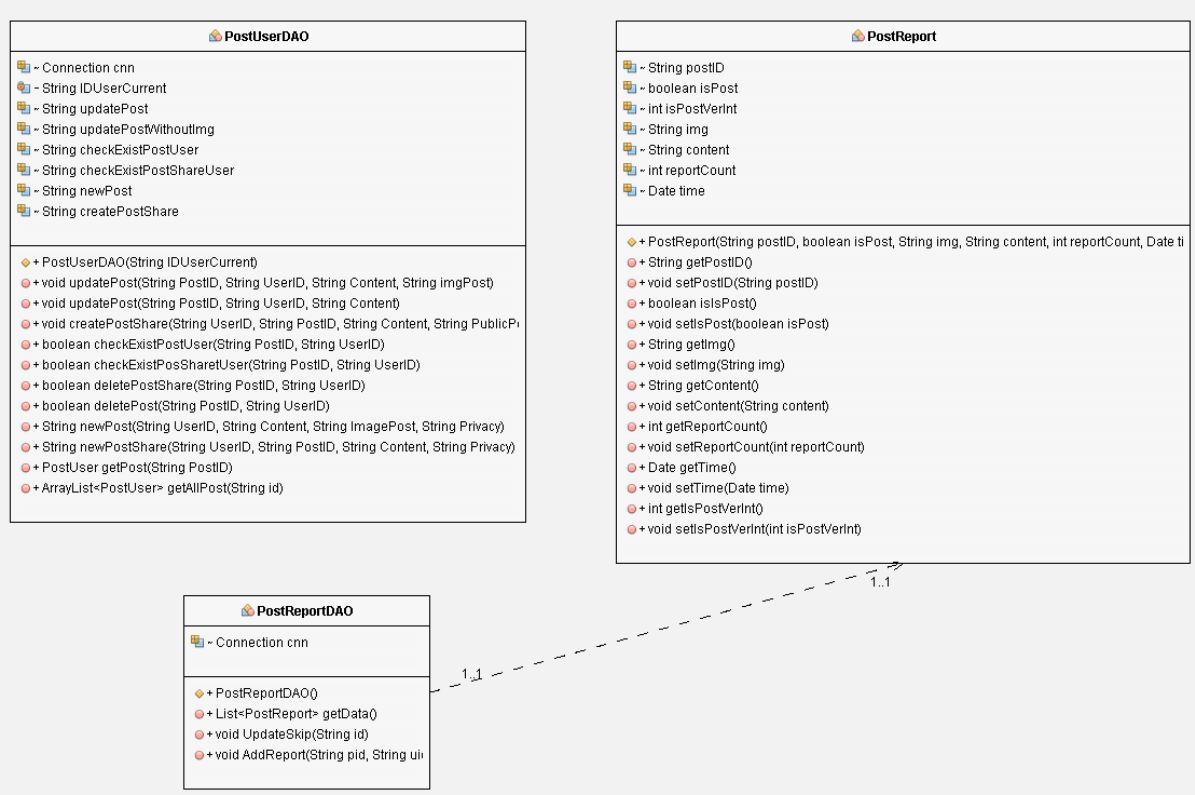
d. Database Queries

No	Queries	Description
01	*SELECT * FROM ReportCommentView*	*This query selects all columns from the ReportCommentView table. It retrieves all the data from this table.*
02	*UPDATE dbo.ReportComment1686 SET ReportComment1686.Status = 0 WHERE ReportComment1686.CommentID =? AND ReportComment1686.IsPost=?*	*This query updates the Status column of the ReportComment1686 table. It sets the status to 0 for a specific comment ID and IsPost flag.*
03	*INSERT INTO dbo.ReportComment1686 (CommentID, UserID, UserID2, IsPost, Status) VALUES (?, -- CommentID - varchar(11) ?, -- UserID - varchar(11) CASE WHEN ? = 1 THEN (SELECT UserID FROM dbo.COMMENT WHERE dbo.COMMENT.CmtID=?)*	*This query inserts a new row into the ReportComment1686 table. It inserts the provided CommentID, UserID, derived UserID2 based on the IsPost flag, IsPost flag itself, and sets the Status to 1.*

	ELSE (SELECT UserID FROM dbo.COMMENTSHARE WHERE dbo.COMMENTSHARE.CmtID=?) END, -- UserID2 - varchar(11) ?, -- IsPost - bit 1 -- Status - bit)	
04	*DELETE dbo.COMMENT WHERE CmtID= ?*	*Delete the Comment in the "COMMENT" table with the "CmtID" corresponding to the value replaced by "?".*
05	*DELETE dbo.COMMENTCHILD WHERE ChildID = ?*	*Delete the child Comments in the "COMMENTCHILD" table with the "ChildID" corresponding to the value replaced by "?".*

9. Report Post

a. Class Diagram



b. Class Specifications

PostUserDAO

No	Method	Description
01	*checkExistPostUser(String PostID, String UserID)*	*This method checks if a post is a shared post by the user.*
02	*checkExistPosSharetUser(String PostID, String UserID)*	*This method checks if a post is a shared post by the user.*

No	Method	Description
03	*deletePostShare(String PostID, String UserID)*	*This method deletes a shared post.*
04	*deletePost(String PostID, String UserID)*	*This method deletes a post.*

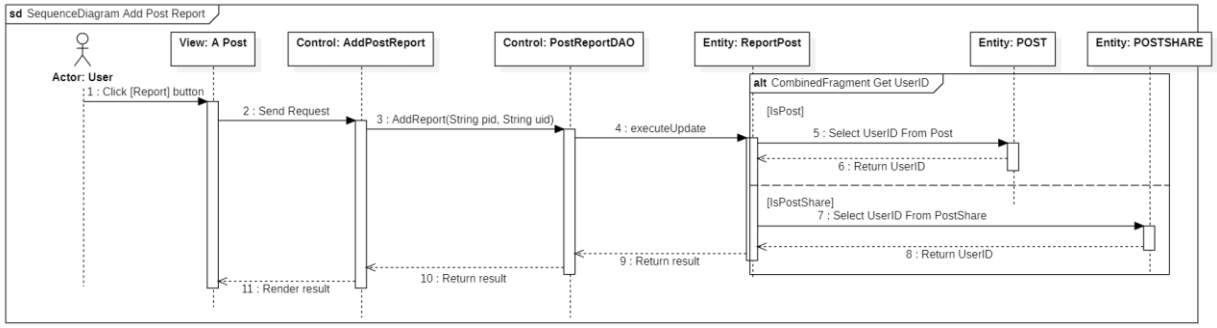
PostReport

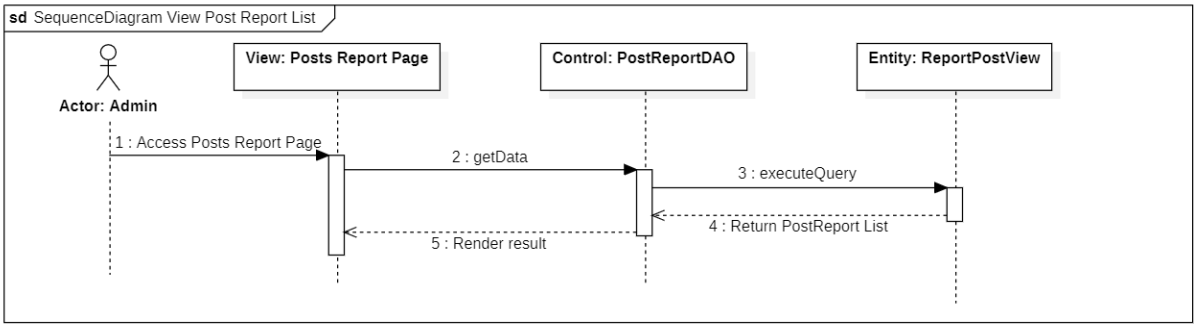
No	Method	Description
01	*PostReport(String postID, boolean isPost, String img, String content, int reportCount, Date time)*	*Initialize a PostReport object with information such as post code, report type, image, content, number of reports, and report time.*
02	*getters and setters*	*Getter and setter methods for properties of UserReport . class*

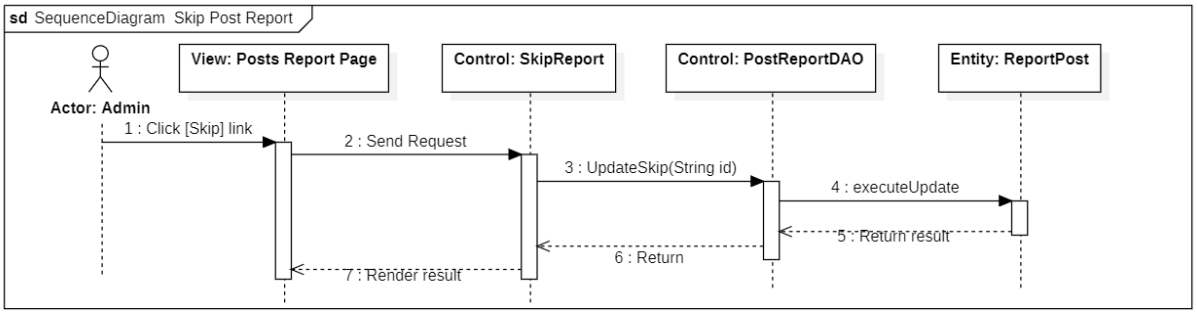
PostReportDAO

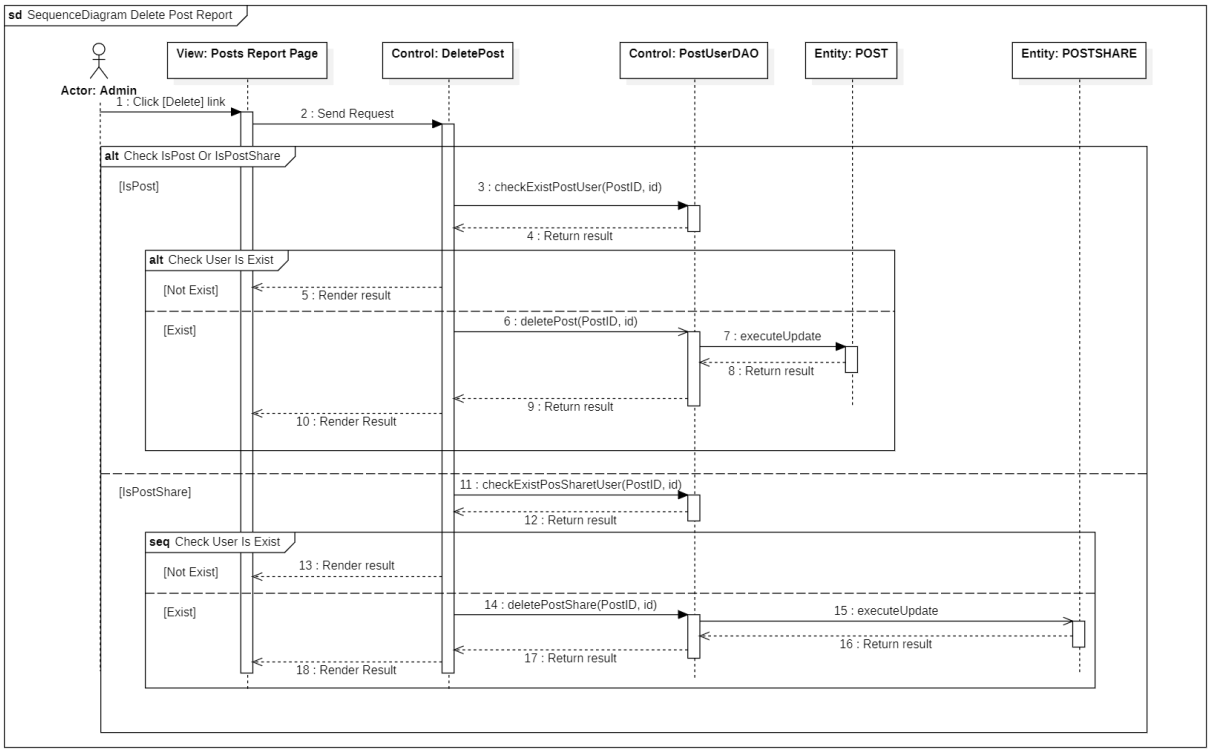
No	Method	Description
01	*PostReportDAO()*	*Initialize a PostReportDAO object and establish a connection to the database by calling the getConnection() method from the connection.connection class*
02	*getData()*	*Query and retrieve data from the ReportPostView table in the database. The method returns a list (List) of PostReport objects. The data is retrieved from the columns in the table including PostID, IsPost, Image, Content, ReportCount, and Time. PostReport objects are only added to the list if the Status field has the value 1 (Unapproved).*
03	*UpdateSkip(String id)*	*Update the Status field of the post report in the ReportPost table to 0(approved) based on the PostID passed in.*
04	*AddReport(String pid, String uid)*	*Add a new report to the ReportPost table. The method takes two parameters, pid (post code) and uid (user code). The IsPost field is set based on the value of the pid. If the pid contains the string "PID", then IsPost is set to true, otherwise false. For the UserID2 field, if IsPost is true, then the value will be queried from the Post table based on the PostID, otherwise it will be queried from the PostShare table based on the ShareID.*

c. Sequence Diagram(s)









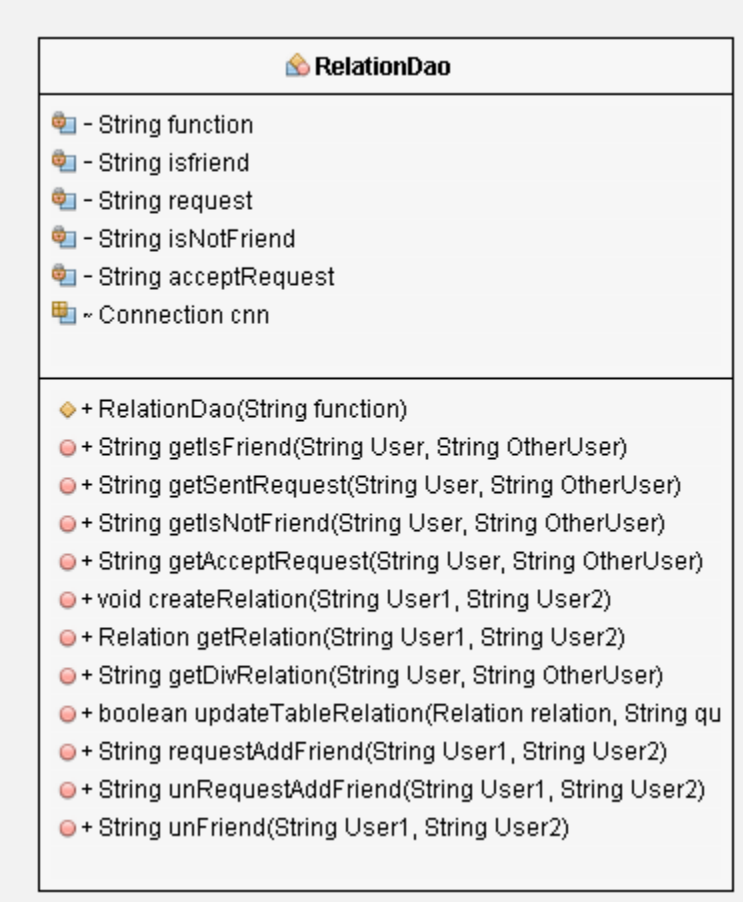
d. Database Queries

No	Queries	Description
01	*SELECT * FROM dbo.POST WHERE PostID= ? AND UserID= ?*	*Query to get all fields of records in table "POST" with "PostID" and "UserID" respectively with value replaced by "?" to check if the Post is the UserID's own or not*
02	*SELECT * FROM dbo.POSTSHARE WHERE ShareID= ? AND UserID= ?*	*Query to get all fields of records in table "POSTSHARE" with "ShareID" and "UserID" respectively with value replaced by "?". to check if that PostShare is the UserID's own or not*
03	*DELETE dbo.POSTSHARE WHERE ShareID= ?*	*Delete a Post in the "POSTSHARE" table with the corresponding "ShareID" value replaced by a "?".*
04	*DELETE FROM dbo.POST WHERE PostID= ?*	*Delete the Posts in the "POST" table whose "PostID" corresponds to the value replaced by a "?"*
05	*SELECT * FROM ReportPostView*	*Query to get reported Posts from View "ReportPostView".*
06	*UPDATE ReportPost SET Status = 0 WHERE PostID = ?*	*Update the "Status" column value to 0 (Browse) in the "ReportPost" table for*

		the record whose "PostID" corresponds to the value replaced by the "?"
07	*INSERT INTO dbo.ReportPost (IsPost, PostID, UserID, UserID2, Status) VALUES (?, ?, ?, CASE WHEN ? = 1 THEN (SELECT UserID FROM Post WHERE POST.PostID=?) ELSE (SELECT UserID FROM PostShare WHERE dbo.POSTSHARE.ShareID=?) END, 1)*	*Add a new Post report to the "ReportPost" table with the corresponding values replaced by a "?"*

10. Manage Friend

a. Class Diagram



b. Class Specifications

RelationDAO

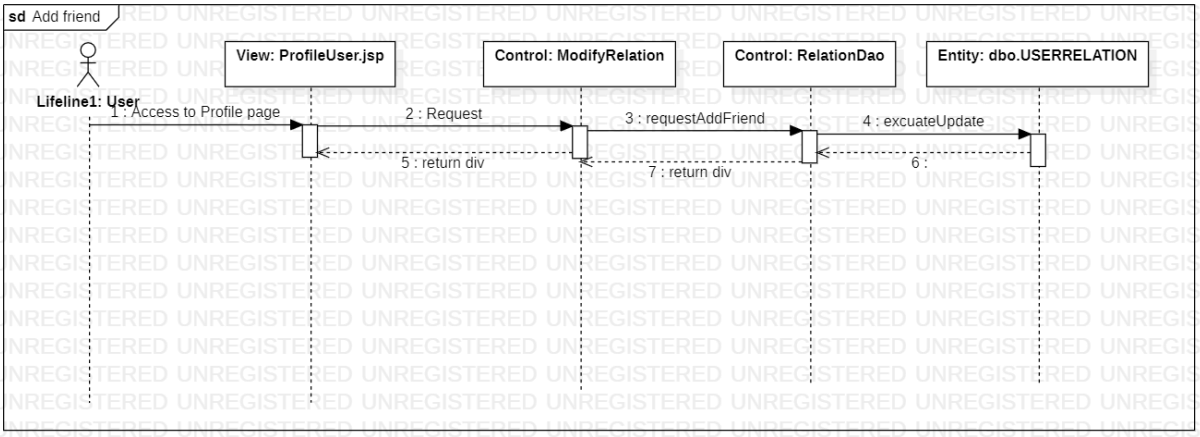
No	Method	Description
01	*createRelation(String User1, String User2)*	*The `createRelation` method is used to create a relationship between two users in the database. If the relationship*

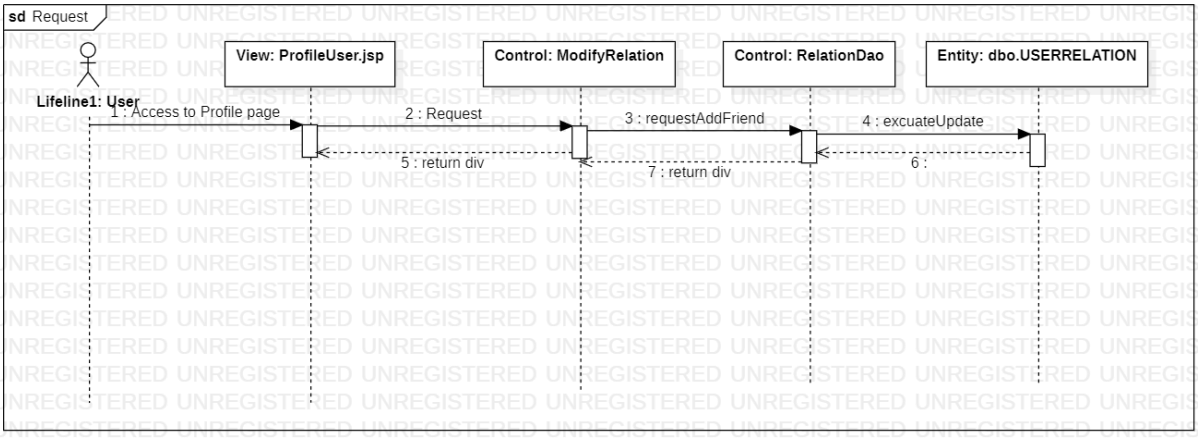
No	Method	Description
		doesn't already exist, the method will add a new record to the USERRELATION table.
02	*getRelation(String User1, String User2)*	*The `getRelation` method is used to get information about the relationship between two users from the database. It returns a `Relation` object containing information about the relationship, or null if the corresponding relationship is not found.*
03	*getDivRelation(String User, String OtherUser)*	*The `getDivRelation` method is used to get information about the relationship between two users and returns a string describing the relationship.* *- If the relationship is "isFriend", return the result from the `getIsFriend` method.* *- If not friends:* *- If there is no request from either side, return the result from the `getIsNotFriend` method.* *- If `User` has an ID less than `OtherUser`:* *- If there is an accept request from `User` to `OtherUser`, return the result from the `getAcceptRequest` method.* *- If there is a request sent from `User` to `OtherUser`, return the result from the `getSentRequest` method.* *- If `User` has an ID greater than `OtherUser`:* *- If a request is sent from `OtherUser` to `User`, return the result from the `getSentRequest` method.* *- Otherwise, return the result from the `getAcceptRequest` method.*
04	*String getIsFriend(String User, String OtherUser)*	*The getIsFriend method is used to determine the relationship between two users who are friends*
05	*String getSentRequest(String User, String OtherUser)*	*The getSentRequest method is used to send requests from one user to another*
06	*getIsNotFriend(String User, String OtherUser)*	*The getIsNotFriend method is used to determine the relationship between two users who are not friends*

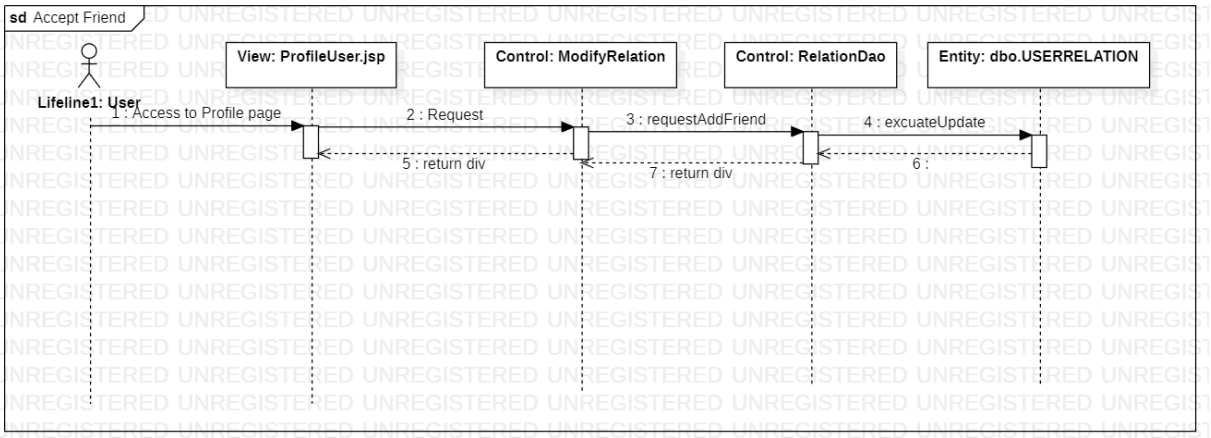
No	Method	Description
07	*getAcceptRequest(String User, String OtherUser)*	*The getAcceptRequest method is used to confirm that the other person has accepted the user's friend request*
08	*requestAddFriend(String User1, String User2)*	*The `requestAddFriend` method is used to make a friend request between two users and returns a string describing the relationship after making the request.* *- Build a query to update the relationship between `User1` and `User2`.* *- Get information about the relationship between `User1` and `User2`.* *- Based on the IDs of `User1` and `User2`, assign the corresponding query to the variable `query`.* *- Update the USERRELATION table with the changed relationship and check the update result.* *- If the update is successful, return the string describing the new relationship by calling the `getDivRelation` method with `User1` and `User2`.* *- If update fails, return null.*
09	*unRequestAddFriend(String User1, String User2)*	*The `unRequestAddFriend` method is used to cancel a friend request between two users and returns a string describing the relationship after canceling the request.* *- Build a query to cancel the friend request between `User1` and `User2`.* *- Get information about the relationship between `User1` and `User2`.* *- Based on the IDs of `User1` and `User2`, assign the corresponding query to the variable `query`.* *- Update the USERRELATION table with the changed relationship and check the update result.* *- If the update is successful, return the string describing the new relationship by calling the `getDivRelation` method with `User1` and `User2`.* *- If update fails, return null.*

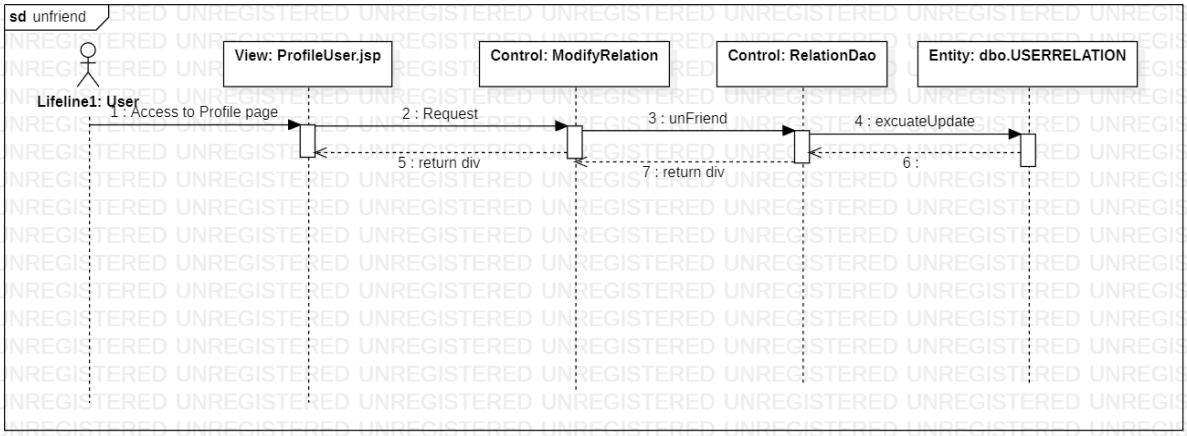
No	Method	Description
10	*unFriend(String User1, String User2)*	*The `unFriend` method is used to unfriend two users and returns a string describing the relationship after unfriending.* *- Construct the query `unfriend` to cancel the friendship relationship between `User1` and `User2`.* *- Get information about the relationship between `User1` and `User2`.* *- Update USERRELATION table with status not friend (isFriend=0) and check update result.* *- If the update is successful, return the string describing the new relationship by calling the `getDivRelation` method with `User1` and `User2`.* *- If update fails, return null.*

c. Sequence Diagram(s)









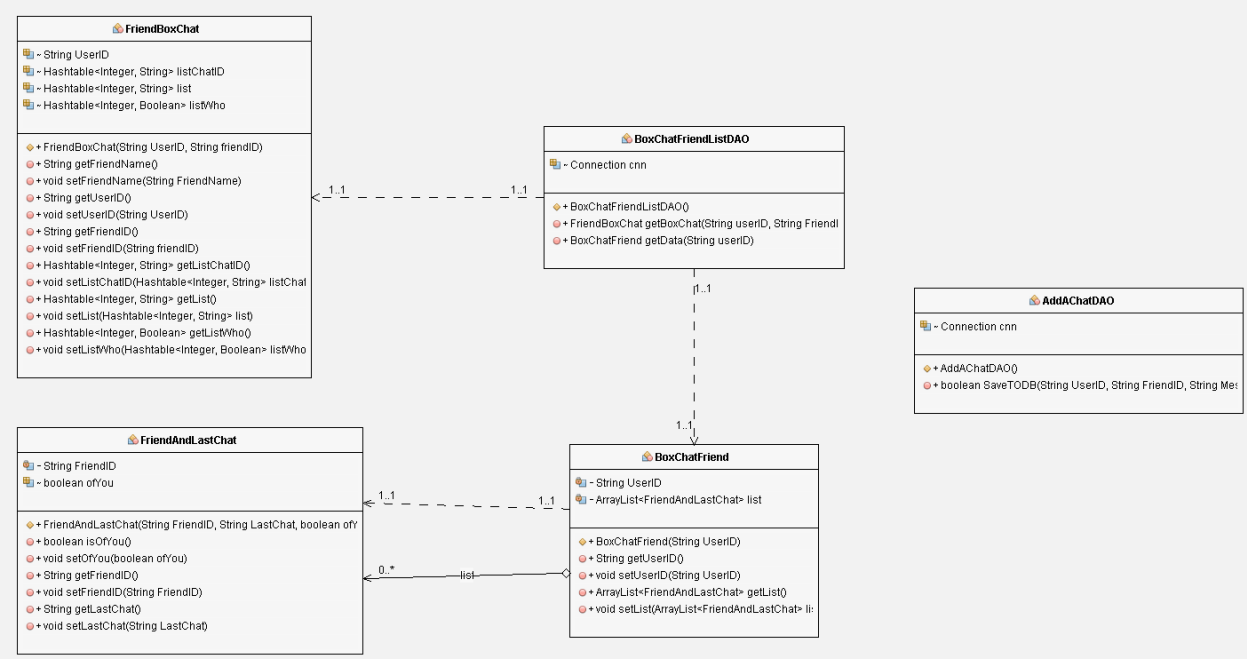
d. Database Queries

No	Queries	Description
01	*DECLARE @User1 NVARCHAR(11), @User2 NVARCHAR(11);* *SET @User1 = ? ;* *SET @User2 = ? ;* *IF NOT EXISTS (* *SELECT ** *FROM dbo.USERRELATION* *WHERE UserID1 = @User1 AND UserID2 = @User2* *)* *BEGIN* *INSERT INTO dbo.USERRELATION* *(* *UserID1,* *UserID2,* *U1RequestU2,*	*The SQL statement is used to create a relationship between two users in the USERRELATION table of the database. Specifically, the statement performs the following actions:* *1. Declare and initialize two variables `@User1` and `@User2` to store the values of the two users passed in the statement.* *2. Check if a relationship exists between `User1` and `User2` in the USERRELATION table.* *3. If the relationship does not exist, execute the INSERT statement to insert a new record into the USERRELATION table with the corresponding values `User1`, `User2`, and default values for*

	U2RequestU1, isFriend) VALUES (@User1, @User2, DEFAULT, DEFAULT, DEFAULT) END;	columns U1RequestU2, respectively. U2RequestU1, isFriend.
02	SELECT U1RequestU2, U2RequestU1, isFriend FROM dbo.USERRELATION WHERE UserID1= ? AND UserID2= ?	The SQL statement retrieves information about the relationship between two users (User1 and User2) from the USERRELATION table.
03	UPDATE dbo.USERRELATION SET U1RequestU2 = 1 WHERE UserID1 = ? AND UserID2 = ?	The SQL statement is used to update the field U1RequestU2 in the USERRELATION table to the value 1 for records where UserID1 equals the passed value and UserID2 equals the passed value.
04	UPDATE dbo.USERRELATION SET U2RequestU1 = 1 WHERE UserID1 = ? AND UserID2 = ?	The SQL statement is used to update the field U2RequestU1 in the USERRELATION table to value 1 for records where UserID1 equals the passed value and UserID2 equals the passed value.
05	UPDATE dbo.USERRELATION SET U1RequestU2 = 00 WHERE UserID1 = ? AND UserID2 = ?	The SQL statement is used to update the U1RequestU2 field in the USERRELATION table to the value 00 for records where UserID1 equals the passed value and UserID2 equals the passed value.
06	UPDATE dbo.USERRELATION SET U1RequestU2 = 00 WHERE UserID1 = ? AND UserID2 = ?	The SQL statement is used to update the field U2RequestU1 in the USERRELATION table to the value 0 for records where UserID1 equals the passed value and UserID2 equals the passed value.
07	UPDATE dbo.USERRELATION SET isFriend= 0 WHERE UserID1 = ? AND UserID2 = ?	The SQL statement is used to update the isFriend field in the USERRELATION table to the value 0 (not friends) for records where UserID1 equals the passed value and UserID2 equals the passed value.

11. Manage Chat

a. Class Diagram



b. Class Specifications

FriendBoxChat Class

No	Method	Description
01	*FriendBoxChat(in String UserID, in String friendID)*	*Set initial values for attributes.*
02	*Getter/Setter*	*access and update the value of attributes*

FriendAndLastChat Class

No	Method	Description
01	*FriendAndLastChat(in String FriendID, in String LastChat, in boolean ofYou)*	*Set initial values for attributes.*
02	*Getter/Setter*	*access and update the value of attributes*

BoxChatFriend Class

No	Method	Description
01	*BoxChatFriend(String UserID)*	*Set initial values for attributes.*
02	*Getter/Setter*	*access and update the value of attributes*

BoxChatFriendListDAO

No	Method	Description

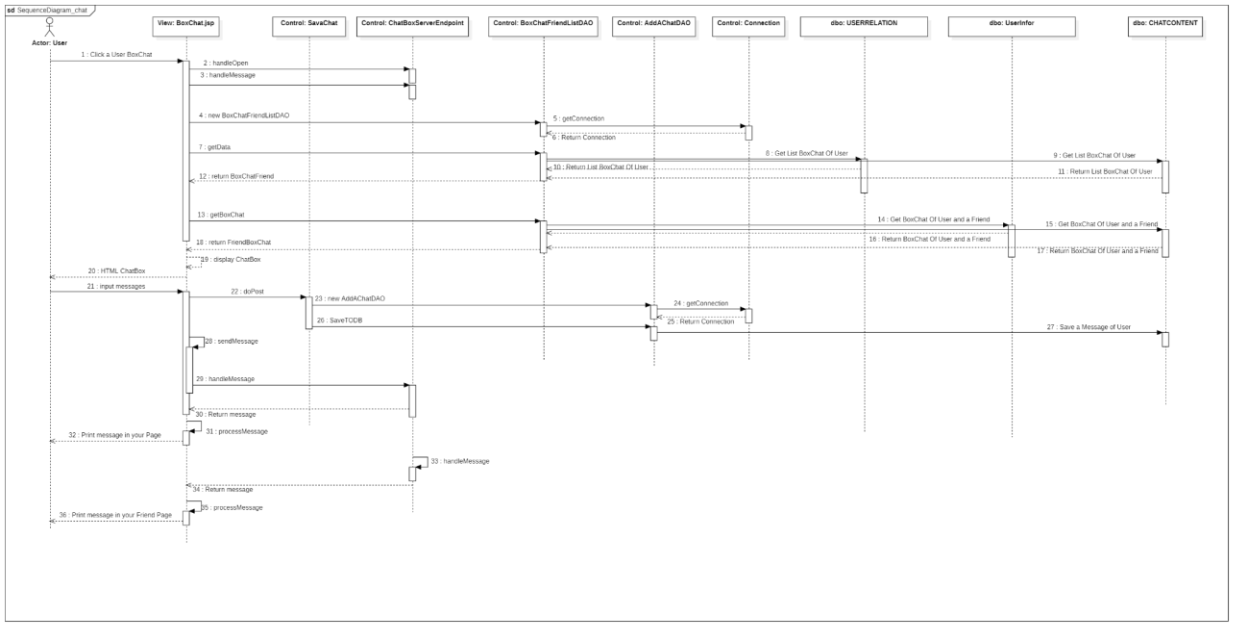
01	*BoxChatFriendListDAO()*	*creates a Connection object, and then the getConnection() method of the Connection Class is called to establish the connection to the database.*
02	*getBoxChat(String userID, String FriendID)*	*This method returns a Friend BoxChat object containing your chat box information with the specified friends. If FriendID is null or empty, the method will return null.* *The method starts by creating a new instance of the Friend Box Chat class with the provided user ID and FriendS.* *The method then executes an SQL query to get data from the CHAT CONTENT table and related UserInfo tables. Query results are stored in ResultSet rs.* *Next, the method takes the values from the SQL query to create the message objects (chatID, message, of You) and determines the friend's name (Friend Name).* *The method then uses FriendBoxChat's setter methods to set the corresponding values.*
03	*getData(in String userID)*	*Method that executes an SQL query to get data from the CHATCONTENT and USERRELATION tables.* *The method takes the values from the SQL query to create FriendAndLastChat objects and add them to the list of BoxChatFriend objects.* *Finally, the method returns a BoxChatFriend object containing information about the user's chat box. In case of an error, the method prints an error message and returns null.*

AddAChatDAO Class

No	Method	Description

01	*AddAChatDAO()*	*creates a Connection object, and then the getConnection() method of the Connection Class is called to establish the connection to the database.*
02	*SaveTODB(String UserID, String FriendID, String Mess)*	*The method first compares UserID and FriendID to determine the order of the two IDs. If UserID comes before FriendID in lexicographic order, the value of check is set to true. Otherwise, the value of check is set to false and the two IDs are swapped.* *The method then executes an SQL query to insert the data into the CHATCONTENT table. The values of UserID1, UserID2, Mess, and ofUser1 are set in the query.* *Finally, the method returns true if the query is executed successfully, and false if an error occurs. In case of an error, the method prints an error message and returns false.*

c. Sequence Diagram(s)



d. Database Queries

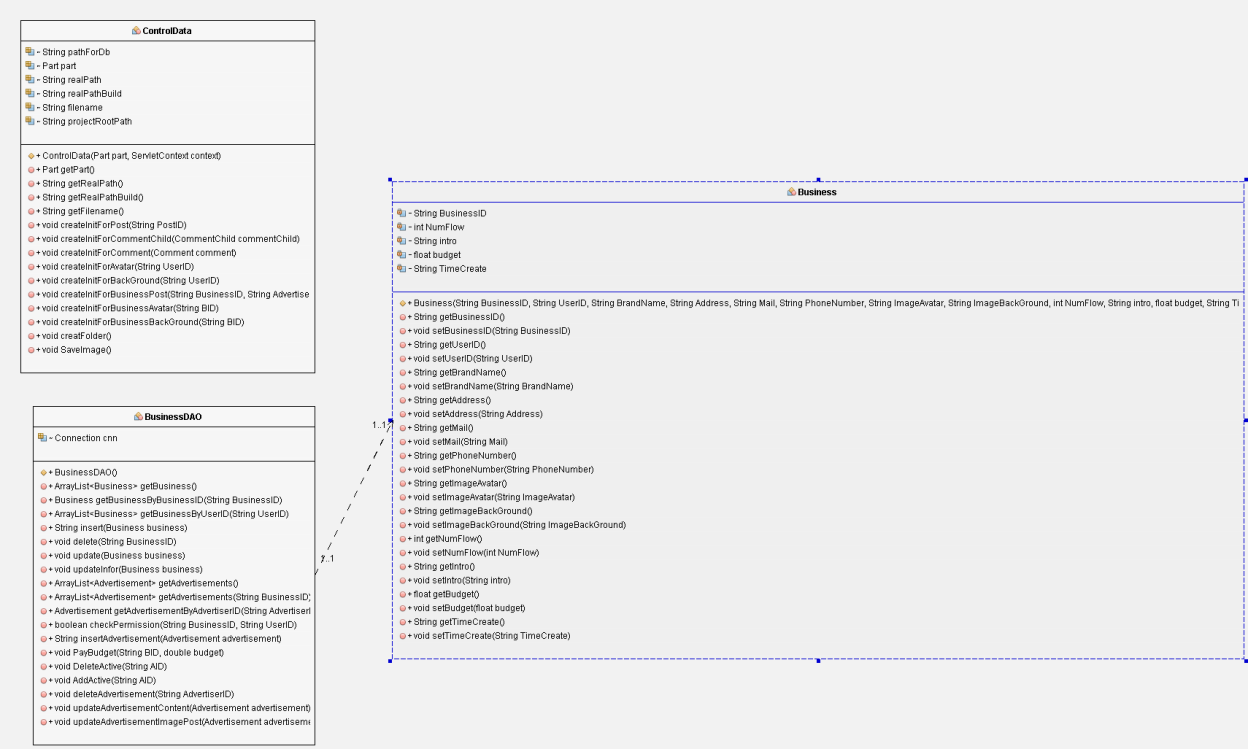
No	Queries	Description
01	*SELECT CHATCONTENT.ChatID, U1.Account AS UserID1, U2.Account AS UserID2, CHATCONTENT.Mess, CHATCONTENT.ofUser1 FROM dbo.CHATCONTENT*	*Câu truy vấn này lấy ra các thông tin về nội dung cuộc trò chuyện từ bảng CHATCONTENT kết hợp với thông tin người dùng từ bảng UserInfor. Câu truy vấn này lấy ra các trường thông tin bao gồm ChatID, UserID1, UserID2, Mess,*

	JOIN dbo.UserInfor U1 ON CHATCONTENT.UserID1 = U1.UserID *JOIN dbo.UserInfor U2 ON CHATCONTENT.UserID2 = U2.UserID* *WHERE U1.UserID = '?' AND U2.UserID = '?'* *ORDER BY CHATCONTENT.CreateAt ASC;*	*và ofUser1. Điều kiện WHERE được sử dụng để lọc kết quả dựa trên UserID1 và UserID2. Kết quả được sắp xếp theo thứ tự tăng dần của trường CreateAt.*
	SELECT T.UserID1, T.UserID2, T.Mess, T.ofUser1, T.CreateAt *FROM (* *SELECT* *ID.UserID1,* *ID.UserID2,* *CHATCONTENT.Mess,* *CHATCONTENT.ofUser1,* *CHATCONTENT.CreateAt,* *RANK() OVER (PARTITION BY ID.UserID1, ID.UserID2 ORDER BY CHATCONTENT.CreateAt DESC) AS Ranking* *FROM dbo.CHATCONTENT* *RIGHT JOIN (* *SELECT USERRELATION.UserID1, USERRELATION.UserID2* *FROM dbo.USERRELATION* *WHERE USERRELATION.UserID1 = ? OR USERRELATION.UserID2 = ?* *) ID ON ID.UserID1 = CHATCONTENT.UserID1 AND ID.UserID2 = CHATCONTENT.UserID2* *) AS T* *WHERE T.Ranking = 1* *ORDER BY T.CreateAt DESC;*	*Câu truy vấn này lấy ra thông tin cuộc trò chuyện gần nhất giữa người dùng và bạn bè từ bảng CHATCONTENT. Câu truy vấn này sử dụng cấu trúc con để tạo ra một bảng tạm gọi là T, trong đó mỗi cuộc trò chuyện được xếp hạng dựa trên trường CreateAt. Kết quả cuối cùng chỉ chọn những cuộc trò chuyện có xếp hạng là 1, và được sắp xếp theo thứ tự giảm dần của trường CreateAt.*
	INSERT INTO dbo.CHATCONTENT *(* *UserID1,* *UserID2,* *Mess,* *ofUser1* *)*	*Câu truy vấn này chèn một dòng mới vào bảng CHATCONTENT để lưu trữ thông tin về một cuộc trò chuyện mới. Các trường UserID1, UserID2, Mess, và ofUser1 được cung cấp giá trị từ tham số của câu truy vấn.*

	VALUES *(* *?, -- UserID1 - varchar(11)* *?, -- UserID2 - varchar(11)* *?, -- Mess - nvarchar(500)* *? -- ofUser1 - bit* *)*	

12. Create Brand

a. Class diagram



b. Class Specifications

Business Class

No	Method	Description
01	*Business(String BusinessID, String UserID, String BrandName, String Address, String Mail, String PhoneNumber, String ImageAvatar, String ImageBackGround, int NumFlow, String intro, float budget, String TimeCreate)*	*The constructor initializes the Business object with the provided values, assigns default values for certain fields if necessary, and sets the ImageAvatar variable with either a default image or the provided image path..*

02	*getter and setter()*	*Getter and setter methods for properties of class CommentReport*

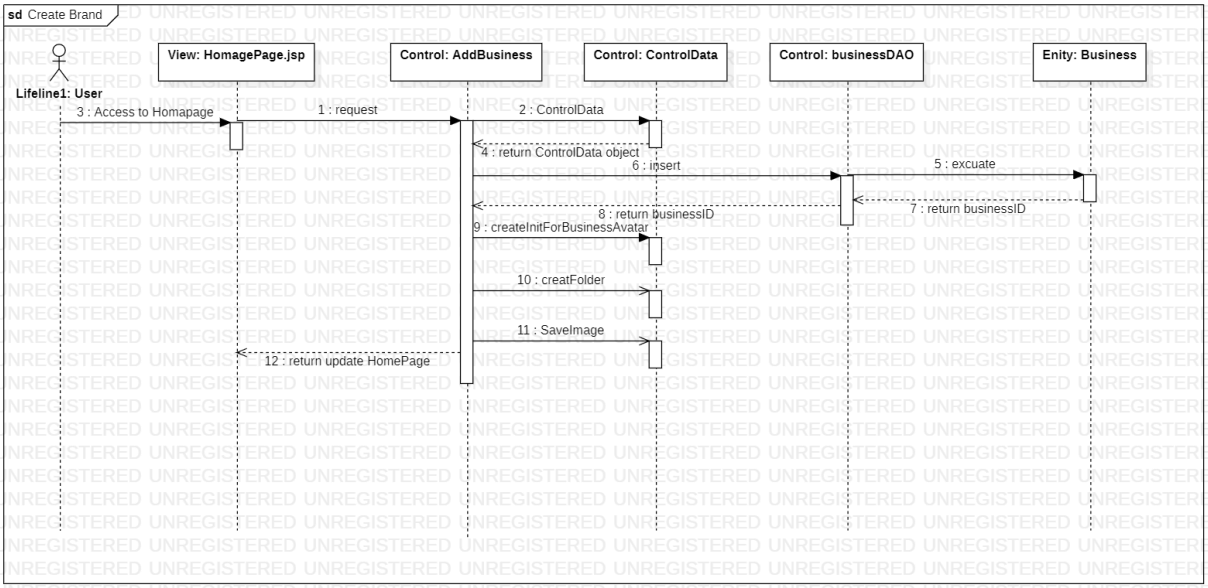
BusinessDAO Class

No	Method	Description
01	*getBusiness()*	*This method retrieves all businesses from the database by executing a SELECT query on the "Business" table. The retrieved data is mapped to `Business` objects, which are then added to an `ArrayList` and returned.*
02	*BusinessDAO()*	*The public BusinessDAO() method is a constructor of the BusinessDAO class. It is used to initiate a connection to the database.*
03	*getBusinessByBusinessID(String BusinessID)*	*This method retrieves a specific business from the database based on the provided BusinessID. It executes a SELECT query with a WHERE clause to filter the results by BusinessID. The retrieved data is used to create a `Business` object, which is then returned.*
04	*getBusinessByUserID(String UserID)*	*This method retrieves businesses associated with a specific UserID. It executes a SELECT query with a WHERE clause to filter the results by UserID. The retrieved data is mapped to `Business` objects, added to an `ArrayList`, and returned.*
05	*insert(Business business)*	*This method inserts a new business record into the database. It executes an INSERT query on the "Business" table, providing the necessary values from the `Business` object. The generated BusinessID is then returned.*
06	*delete(String BusinessID)*	*This method deletes a business record from the database based on the provided BusinessID. It executes a DELETE query on the "Business" table with a WHERE clause to specify the BusinessID.*
07	*update(Business business)*	*This method updates an existing business record in the database. It executes an UPDATE query on the "Business" table, setting the new values based on the provided `Business` object, using the BusinessID as a reference.*

08	*getAdvertisements()*	*This method retrieves all advertisements from the database by executing a SELECT query on the "Advertisement" table. The retrieved data is mapped to `Advertisement` objects, which are then added to an `ArrayList` and returned.*
09	*getAdvertisements(String BusinessID)*	*This method retrieves advertisements associated with a specific BusinessID. It executes a SELECT query with a WHERE clause to filter the results by BusinessID. The retrieved data is mapped to `Advertisement` objects, added to an `ArrayList`, and returned.*
10	*getAdvertisementByAdvertiserID(String AdvertiserID)*	*This method retrieves a specific advertisement from the database based on the provided AdvertiserID. It executes a SELECT query with a WHERE clause to filter the results by AdvertiserID. The retrieved data is used to create an `Advertisement` object, which is then returned.*
11	*checkPermission(String BusinessID, String UserID)*	*This method checks if a user has permission to access a specific business. It executes a SELECT query on the "Business" table with a WHERE clause to check if the BusinessID and UserID match any records in the database. It returns `true` if a match is found, indicating permission, and `false` otherwise.*
12	*insertAdvertisement(Advertisement advertisement)*	*This method inserts a new advertisement record into the database. It executes an INSERT query on the "Advertisement" table, providing the necessary values from the `Advertisement` object. The generated AdvertiserID is then returned.*
13	*DeleteActive(String AID)*	*This method deletes an active advertisement record from the "Active" table based on the provided AdvertiserID.*
14	*AddActive(String AID)*	*This method adds an active advertisement record to the "Active" table with the provided AdvertiserID.*
15	*deleteAdvertisement(String AdvertiserID)*	*This method deletes an advertisement record from the "Advertisement" table based on the provided AdvertiserID.*

16	*updateAdvertisement Content(Advertisement advertisement)*	*This method updates the content, quantity, and status of an advertisement record in the "Advertisement" table. It executes an UPDATE query, setting the new values based on the provided `Advertisement` object, using the AdvertiserID as a reference.*
17	*updateAdvertisementImagePost(Advertisement advertisement)*	*This method updates the image post of an advertisement record in the "Advertisement" table. It executes an UPDATE query, setting the new image post value based on the provided `Advertisement` object, using the AdvertiserID and BusinessID as references.*

c. Sequence Diagram(s)



d. Database Queries

No	Queries	Description
01	*SELECT BusinessID, UserID, BrandName, Address, Mail, PhoneNumber, ImageAvatar, ImageBackGround, NumFlow, intro, budget, TimeCreate FROM dbo.Business*	*This query retrieves information about all businesses from the "Business" table, including columns: BusinessID, UserID, BrandName, Address, Mail, PhoneNumber, ImageAvatar, ImageBackGround, NumFlow, intro, budget, TimeCreate.*

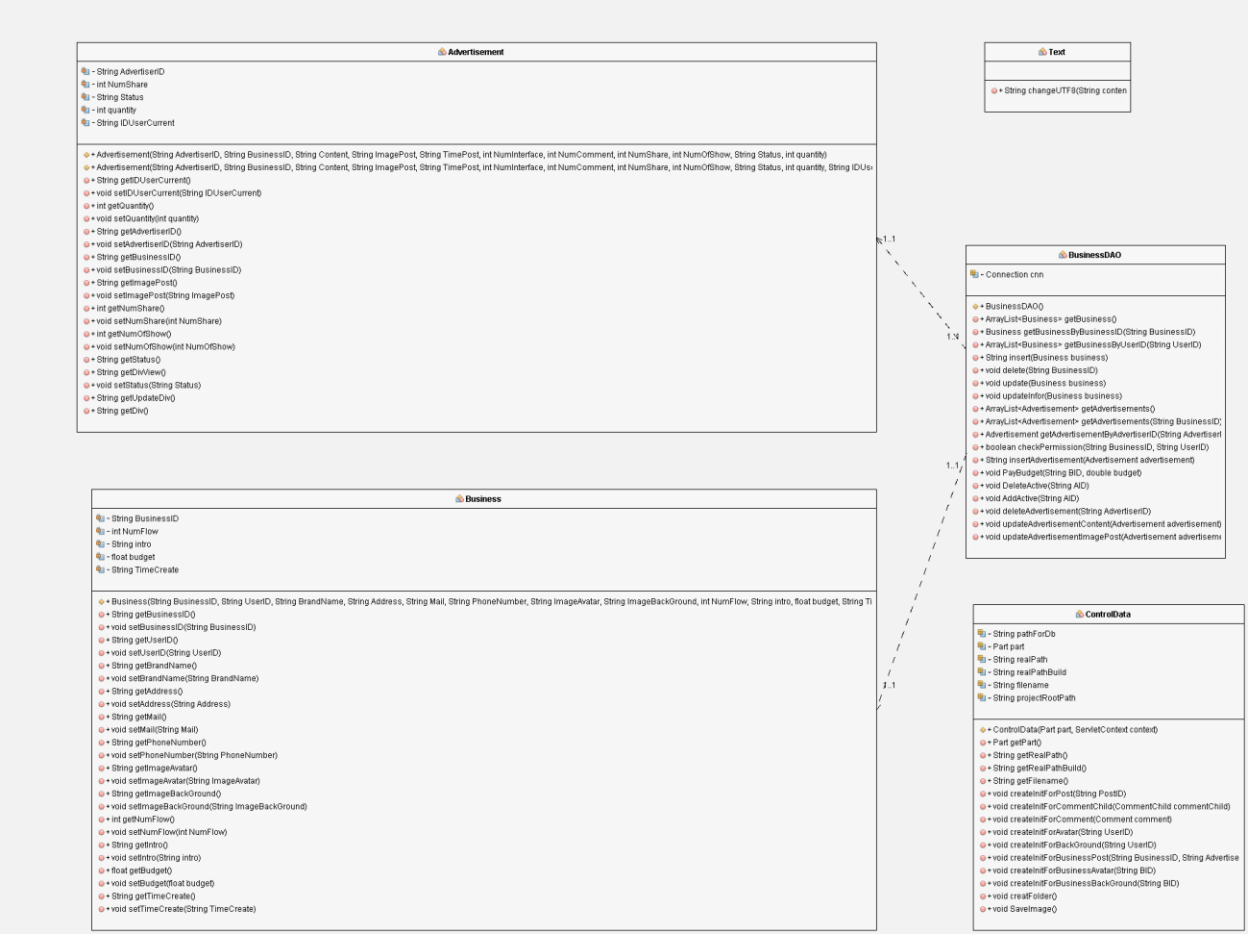
02	*SELECT BusinessID, UserID, BrandName, Address, Mail, PhoneNumber, ImageAvatar, ImageBackGround, NumFlow, intro, budget, TimeCreate FROM dbo.Business WHERE BusinessID = ?*	*This query retrieves the information of a specific business based on the specified BusinessID.*
03	*SELECT BusinessID, UserID, BrandName, Address, Mail, PhoneNumber, ImageAvatar, ImageBackGround, NumFlow, intro, budget, TimeCreate FROM dbo.Business WHERE UserID = ?*	*This query retrieves a list of businesses based on the specified UserID.*
04	*DECLARE @InsertedIDs TABLE (BusinessID VARCHAR(11)); INSERT INTO dbo.Business (UserID, BrandName, Address, Mail, PhoneNumber, ImageAvatar, ImageBackGround, NumFlow, intro, budget, TimeCreate) OUTPUT Inserted.BusinessID INTO @InsertedIDs VALUES (?, ?, ?, ?, ?, ?, ?, DEFAULT, ?, DEFAULT, DEFAULT); SELECT BusinessID FROM @InsertedIDs;*	*This statement inserts a new record into the "Business" table and returns the BusinessID of the inserted record. The values for the columns are passed as parameters.*
05	*DELETE FROM dbo.Business WHERE BusinessID = ?*	*This statement deletes a business record based on the specified BusinessID.*
06	*UPDATE dbo.Business SET UserID = ?, BrandName = ?, Address = ?, Mail = ?, PhoneNumber = ?, intro = ?, budget = ? WHERE BusinessID = ?*	*This statement updates the information of a business based on the specified BusinessID. The new values for the columns are passed as parameters.*
07	*SELECT AdvertiserID, BusinessID, Content, ImagePost, TimePost, NumInterface, NumComment,*	*This query retrieves information of all advertisements from the "Advertisement" table, including columns: AdvertiserID, BusinessID,*

	NumShare, NumOfShow, Status, Quantity *FROM dbo.Advertisement*	*Content, ImagePost, TimePost, NumInterface, NumComment, NumShare, NumOfShow, Status, Quantity.*
08	*SELECT AdvertiserID, BusinessID, Content, ImagePost, TimePost, NumInterface, NumComment, NumShare, NumOfShow, Status, Quantity* *FROM dbo.Advertisement* *WHERE BusinessID = ?*	*This query retrieves a list of advertisements for a specific business based on the specified BusinessID.*
09	*SELECT AdvertiserID, BusinessID, Content, ImagePost, TimePost, NumInterface, NumComment, NumShare, NumOfShow, Status, Quantity* *FROM dbo.Advertisement* *WHERE AdvertiserID = ?*	*This query retrieves the information of an advertisement based on the specified AdvertiserID.*
10	*SELECT ** *FROM dbo.Business* *WHERE BusinessID = ? AND UserID = ?*	*This query checks the existence of a record in the "Business" table based on the specified BusinessID and UserID.*
11	*DECLARE @inserted TABLE (AdvertiserID NVARCHAR(11));* *INSERT INTO dbo.Advertisement (BusinessID, Content, ImagePost, TimePost, NumInterface, NumComment, NumShare, NumOfShow, Status)* *OUTPUT Inserted.AdvertiserID INTO @inserted* *VALUES (?, ?, ?, DEFAULT, DEFAULT, DEFAULT, DEFAULT, DEFAULT, 'inactive');* *SELECT AdvertiserID FROM @inserted;*	*This statement inserts a new advertisement into the "Advertisement" table and returns the AdvertiserID of the inserted advertisement. The values for the columns are passed as parameters.*
12	*DELETE FROM dbo.Active* *WHERE AdvertiserID = ?*	*This statement deletes a record in the "Active" table based on the specified AdvertiserID.*

13	*INSERT INTO dbo.Active (AdvertiserID, dateShow) VALUES (?, DEFAULT)*	*This statement inserts a new record into the "Active" table with the specified AdvertiserID and default value for the dateShow column.*
14	*DELETE FROM dbo.Advertisement WHERE AdvertiserID = ?*	*This statement deletes an advertisement based on the specified AdvertiserID.*
15	*UPDATE dbo.Advertisement SET Content = ?, Quantity = ?, Status = ? WHERE AdvertiserID = ?*	*This statement updates the information of an advertisement based on the specified AdvertiserID. The new values for the columns are passed as parameters.*
16	*UPDATE dbo.Advertisement SET ImagePost = ? WHERE AdvertiserID = ? AND BusinessID = ?*	*This statement updates the image path of an advertisement based on the specified AdvertiserID and BusinessID.*

13. Manage ADS And Pay For ADS

a. Class diagram



b. Class Specifications

Advertisement

No	Method	Description
01	*Advertisement(String AdvertiserID, String BusinessID, String Content, String ImagePost, String TimePost, int NumInterface, int NumComment, int NumShare, int NumOfShow, String Status, int quantity)*	*This initialization method is used to create a Advertisement object with the provided values. After calling this initialization method, the properties of the Advertisement object will be set with the corresponding values.*
02	*Advertisement(String AdvertiserID, String BusinessID, String Content, String ImagePost, String*	*This initialization method is used to create a Advertisement object with the provided values. After calling this initialization method, the properties of the Advertisement object will be set with the corresponding values.*

	TimePost, int NumInterface, int NumComment, int NumShare, int NumOfShow, String Status, int quantity, String IDUserCurrent)	
03	*getter and setter()*	*Getter and setter methods for properties of class CommentReport*

Business

No	Method	Description
01	*Business(String BusinessID, String UserID, String BrandName, String Address, String Mail, String PhoneNumber, String ImageAvatar, String ImageBackGround, int NumFlow, String intro, float budget, String TimeCreate)*	*This initialization method is used to create a Business object with the provided values. After calling this initialization method, the properties of the Business object will be set with the corresponding values.*
02	*getter and setter()*	*Getter and setter methods for properties of class CommentReport*

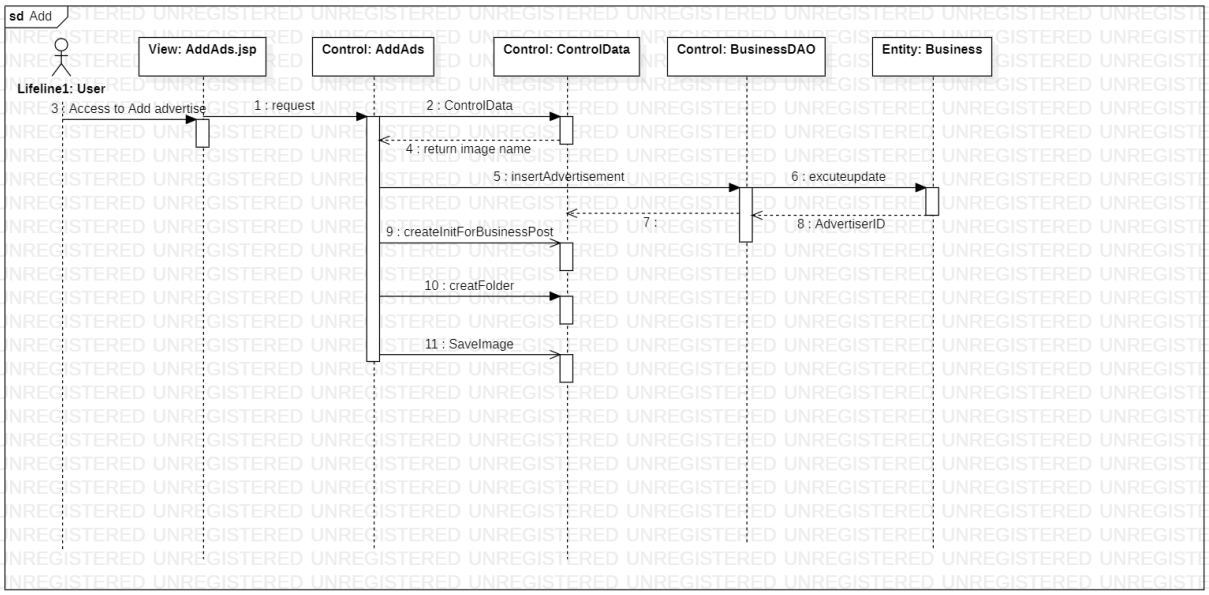
BusinessDAO

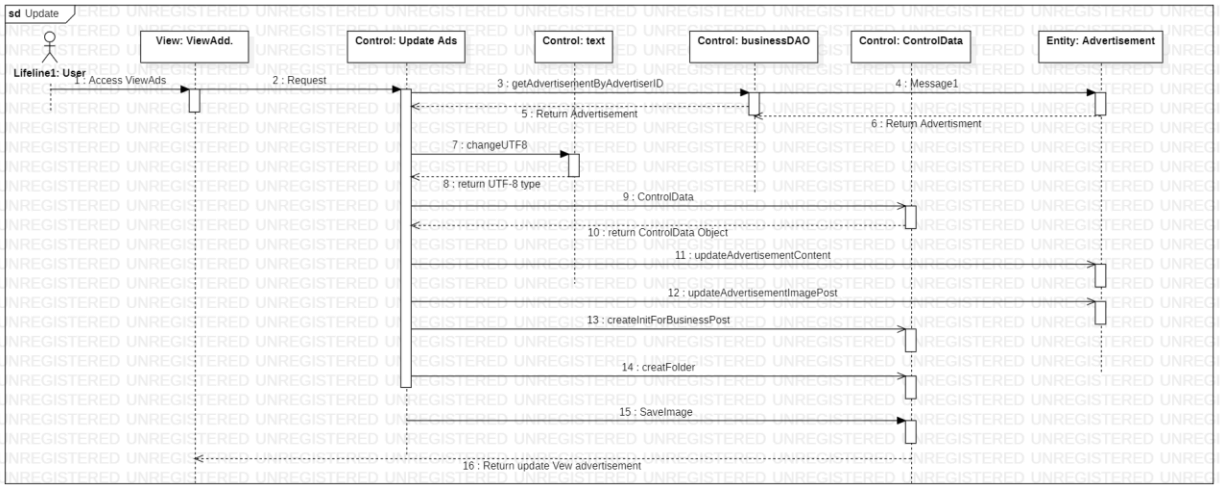
No	Method	Description
01	*getBusiness()*	*This method retrieves all businesses from the database by executing a SELECT query on the "Business" table. The retrieved data is mapped to `Business` objects, which are then added to an `ArrayList` and returned.*
02	*BusinessDAO()*	*The public BusinessDAO() method is a constructor of the BusinessDAO class. It is used to initiate a connection to the database.*
03	*insertAdvertisement(Advertisement advertisement)*	*This method inserts a new advertisement record into the database. It executes an INSERT query on the "Advertisement" table, providing the necessary values from the `Advertisement` object. The generated AdvertiserID is then returned.*

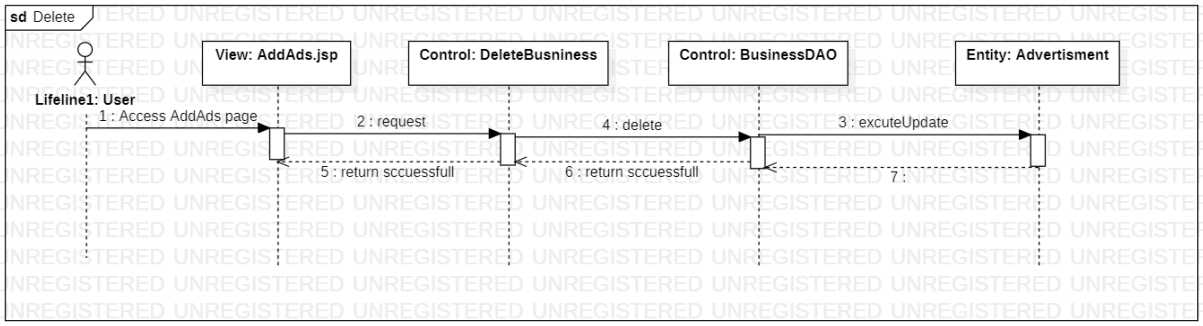
04	*getBusinessByBusinessID(String BusinessID)*	*This method retrieves a specific business from the database based on the provided BusinessID. It executes a SELECT query with a WHERE clause to filter the results by BusinessID. The retrieved data is used to create a `Business` object, which is then returned.*
05	*getBusinessByUserID(String UserID)*	*This method retrieves businesses associated with a specific UserID. It executes a SELECT query with a WHERE clause to filter the results by UserID. The retrieved data is mapped to `Business` objects, added to an `ArrayList`, and returned.*
06	*deleteAdvertisement(String AdvertiserID)*	*This method deletes an advertisement record from the "Advertisement" table based on the provided AdvertiserID.*
07	*delete(String BusinessID)*	*This method deletes a business record from the database based on the provided BusinessID. It executes a DELETE query on the "Business" table with a WHERE clause to specify the BusinessID.*
08	*getAdvertisementByAdvertiserID(String AdvertiserID)*	*This method retrieves a specific advertisement from the database based on the provided AdvertiserID. It executes a SELECT query with a WHERE clause to filter the results by AdvertiserID. The retrieved data is used to create an `Advertisement` object, which is then returned.*
09	*DeleteActive(String AID)*	*This method deletes an active advertisement record from the "Active" table based on the provided AdvertiserID.*
10	*AddActive(String AID)*	*This method adds an active advertisement record to the "Active" table with the provided AdvertiserID.*
11	*update(Business business)*	*This method updates an existing business record in the database. It executes an UPDATE query on the "Business" table, setting the new values based on the provided `Business` object, using the BusinessID as a reference.*

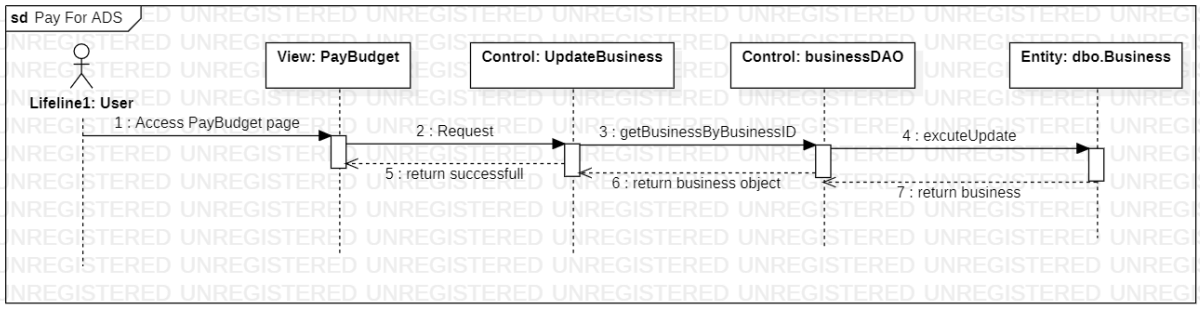
12	*updateAdvertisementContent(Advertisement advertisement)*	*This method updates the content, quantity, and status of an advertisement record in the "Advertisement" table. It executes an UPDATE query, setting the new values based on the provided `Advertisement` object, using the AdvertiserID as a reference.*
13	*updateAdvertisementImagePost(Advertisement advertisement)*	*This method updates the image post of an advertisement record in the "Advertisement" table. It executes an UPDATE query, setting the new image post value based on the provided `Advertisement` object, using the AdvertiserID and BusinessID as references.*

c. Sequence Diagram(s)









d. Database Queries

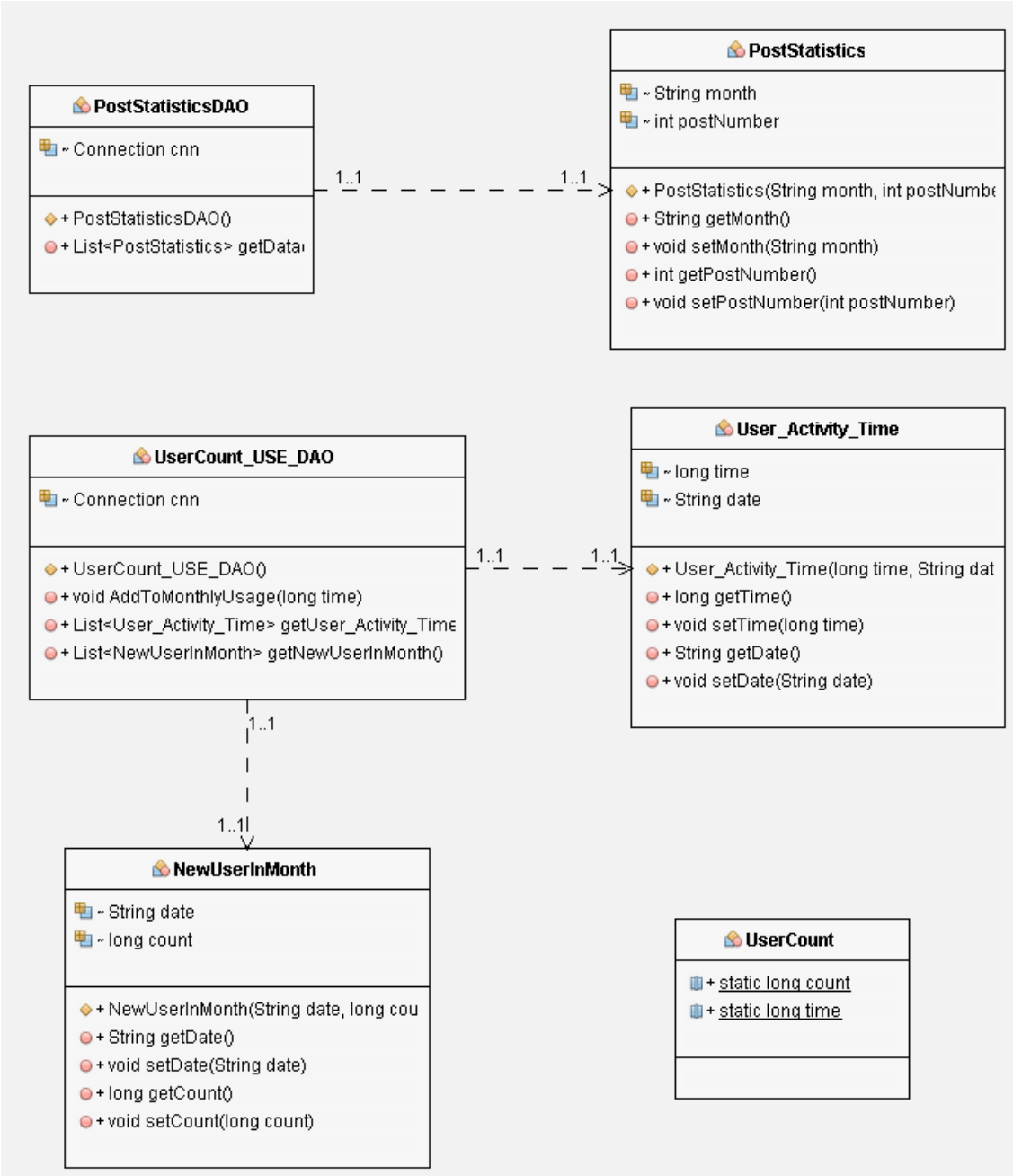
No	Queries	Description
01	*SELECT BusinessID, UserID, BrandName, Address, Mail, PhoneNumber, ImageAvatar, ImageBackGround, NumFlow, intro, budget, TimeCreate FROM dbo.Business*	*This query retrieves information about all businesses from the "Business" table, including columns: BusinessID, UserID, BrandName, Address, Mail, PhoneNumber, ImageAvatar, ImageBackGround, NumFlow, intro, budget, TimeCreate.*
02	*SELECT BusinessID, UserID, BrandName, Address, Mail, PhoneNumber, ImageAvatar, ImageBackGround, NumFlow, intro, budget, TimeCreate FROM dbo.Business WHERE BusinessID = ?*	*This query retrieves the information of a specific business based on the specified BusinessID.*
03	*SELECT BusinessID, UserID, BrandName, Address, Mail, PhoneNumber, ImageAvatar, ImageBackGround, NumFlow, intro, budget, TimeCreate FROM dbo.Business WHERE UserID = ?*	*This query retrieves a list of businesses based on the specified UserID.*
04	*DECLARE @InsertedIDs TABLE (BusinessID VARCHAR(11)); INSERT INTO dbo.Business (UserID, BrandName, Address, Mail, PhoneNumber, ImageAvatar, ImageBackGround, NumFlow, intro, budget, TimeCreate) OUTPUT Inserted.BusinessID INTO @InsertedIDs*	*This statement inserts a new record into the "Business" table and returns the BusinessID of the inserted record. The values for the columns are passed as parameters.*

	VALUES (?, ?, ?, ?, ?, ?, ?, DEFAULT, ?, DEFAULT, DEFAULT); *SELECT BusinessID FROM @InsertedIDs;*	
05	*DELETE FROM dbo.Business* *WHERE BusinessID = ?*	*This statement deletes a business record based on the specified BusinessID.*
06	*UPDATE dbo.Business* *SET UserID = ?, BrandName = ?, Address = ?, Mail = ?, PhoneNumber = ?, intro = ?, budget = ?* *WHERE BusinessID = ?*	*This statement updates the information of a business based on the specified BusinessID. The new values for the columns are passed as parameters.*
07	*SELECT AdvertiserID, BusinessID, Content, ImagePost, TimePost, NumInterface, NumComment, NumShare, NumOfShow, Status, Quantity* *FROM dbo.Advertisement*	*This query retrieves information of all advertisements from the "Advertisement" table, including columns: AdvertiserID, BusinessID, Content, ImagePost, TimePost, NumInterface, NumComment, NumShare, NumOfShow, Status, Quantity.*
08	*SELECT AdvertiserID, BusinessID, Content, ImagePost, TimePost, NumInterface, NumComment, NumShare, NumOfShow, Status, Quantity* *FROM dbo.Advertisement* *WHERE BusinessID = ?*	*This query retrieves a list of advertisements for a specific business based on the specified BusinessID.*
09	*SELECT AdvertiserID, BusinessID, Content, ImagePost, TimePost, NumInterface, NumComment, NumShare, NumOfShow, Status, Quantity* *FROM dbo.Advertisement* *WHERE AdvertiserID = ?*	*This query retrieves the information of an advertisement based on the specified AdvertiserID.*
10	*SELECT ** *FROM dbo.Business*	*This query checks the existence of a record in the "Business" table based on the specified BusinessID and UserID.*

	WHERE BusinessID = ? AND UserID = ?	
11	*DECLARE @inserted TABLE (AdvertiserID NVARCHAR(11)); INSERT INTO dbo.Advertisement (BusinessID, Content, ImagePost, TimePost, NumInterface, NumComment, NumShare, NumOfShow, Status) OUTPUT Inserted.AdvertiserID INTO @inserted VALUES (?, ?, ?, DEFAULT, DEFAULT, DEFAULT, DEFAULT, DEFAULT, 'inactive'); SELECT AdvertiserID FROM @inserted;*	*This statement inserts a new advertisement into the "Advertisement" table and returns the AdvertiserID of the inserted advertisement. The values for the columns are passed as parameters.*
12	*DELETE FROM dbo.Active WHERE AdvertiserID = ?*	*This statement deletes a record in the "Active" table based on the specified AdvertiserID.*
13	*INSERT INTO dbo.Active (AdvertiserID, dateShow) VALUES (?, DEFAULT)*	*This statement inserts a new record into the "Active" table with the specified AdvertiserID and default value for the dateShow column.*
14	*DELETE FROM dbo.Advertisement WHERE AdvertiserID = ?*	*This statement deletes an advertisement based on the specified AdvertiserID.*
15	*UPDATE dbo.Advertisement SET Content = ?, Quantity = ?, Status = ? WHERE AdvertiserID = ?*	*This statement updates the information of an advertisement based on the specified AdvertiserID. The new values for the columns are passed as parameters.*
16	*UPDATE dbo.Advertisement SET ImagePost = ? WHERE AdvertiserID = ? AND BusinessID = ?*	*This statement updates the image path of an advertisement based on the specified AdvertiserID and BusinessID.*

14. View Statistical

a. Class diagram



b. Class Specifications

User_Activity_Time

No	Method	Description
01	*User_Activity_Time(long time, String date)*	*Initializes a User_Activity_Time object with the parameter time and date (Time-time used by all users from month date to the beginning of month date+1) passed.*
02	*Getters and Setters*	*Getter and setter methods for properties of User_Activity_Time . class*

NewUserInMonth

No	Method	Description
01	*NewUserInMonth(String date, long count)*	*Initializes a NewUserInMonth object with the date and count parameters passed (month and number of accounts created in that month).*
02	*Getters and Setters*	*Getter and setter methods for the property properties of the NewUserInMonth . class*

PostStatistics

No	Method	Description
01	*PostStatistics(String month, int postNumber)*	*Instantiate a PostStatistics object with the parameter month and postNumber passed (month and number of posts in that month).*
02	*Getters and Setters*	*Getter and setter methods for properties of the PostStatistics class*

UserCount_USE_DAO

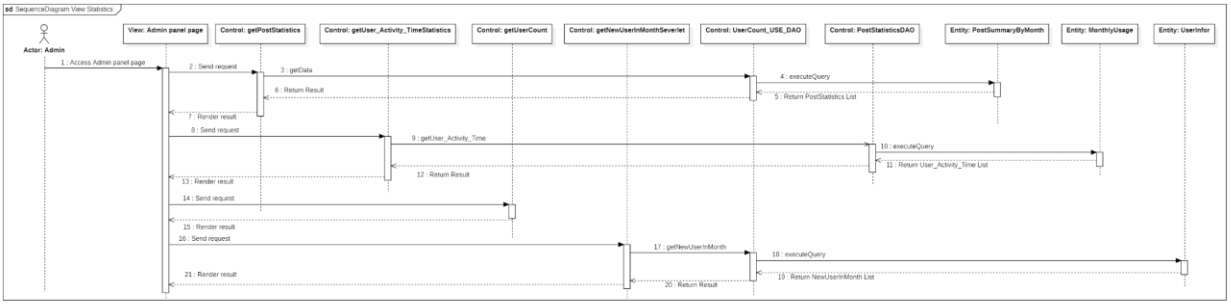
No	Method	Description
01	*UserCount_USE_DAO()*	*Instantiate a UserCount_USE_DAO object and establish a connection to the database by calling the getConnection() method from the connection.connection class.*
02	*AddToMonthlyUsage(long time)*	*Add usage time to the MonthlyUsage table. This method performs the check and adds or updates the usage time in the current month. The time parameter represents the usage time that needs to be added.*
03	*getUser_Activity_Time()*	*Query and return a list of User_Activity_Time objects. This method retrieves information about the user's uptime in recent months. The result is a list of User_Activity_Time objects, where each object contains information about the activity time and the corresponding month.*
04	*getNewUserInMonth()*	*Query and return a list of NewUserInMonth objects. This method retrieves information about the number of new users registered in recent months. The result is a list of NewUserInMonth objects, where each object contains*

No	Method	Description
		information about the number of users and the corresponding month.

PostStatisticsDAO

No	Method	Description
01	*PostStatisticsDAO()*	*Instantiate a PostStatisticsDAO object and establish a connection to the database by calling the getConnection() method from the connection.connection class.*
02	*getData()*	*Query and return a list of PostStatistics objects. This method gets information about the number of posts in recent months. The result is a list of PostStatistics objects, where each object contains information about the month and the corresponding post count.*

c. Sequence Diagram(s)



d. Database Queries

PostStatisticsDAO

No	Queries	Description
01	*SELECT M.CreatedMonth AS Month, M.CreatedYear AS Year, COALESCE(P.TotalPosts, 0) AS TotalPosts FROM (SELECT MONTH(MonthDate) AS CreatedMonth, YEAR(MonthDate) AS CreatedYear FROM (SELECT TOP 11 DATEADD(MONTH, -ROW_NUMBER() OVER (ORDER*	*This query statement retrieves the total number of posts by month and year from the table "PostSummaryByMonth". The results return columns "Month", "Year", and "TotalPosts". The results are sorted in ascending order of year and month..*

	BY (SELECT NULL)), GETDATE()) AS MonthDate *FROM* *sys.objects* *UNION ALL* *SELECT* *DATEADD(MONTH, 0, GETDATE()) AS MonthDate) AS Subquery) AS M* *LEFT JOIN* *(SELECT TOP 12* *Month,* *Year,* *TotalPosts* *FROM* *PostSummaryByMonth* *ORDER BY* *Year DESC,* *Month DESC) AS P* *ON* *M.CreatedMonth = P.Month* *AND M.CreatedYear = P.Year* *ORDER BY* *M.CreatedYear ASC,* *M.CreatedMonth ASC*	

UserCount_USE_DAO

No	Queries	Description
01	*INSERT INTO dbo.MonthlyUsage* *(* *MonthDate,* *UsageTime* *)* *VALUES* *(* *DATEFROMPARTS(YEAR(GETDATE()), MONTH(GETDATE()), 1),* *?* *)*	*The SQL statement is used to add a new record to the Monthly Usage table with the month and year from the current date, along with the usage time value (which should be replaced with the actual value).*
02	*UPDATE MonthlyUsage* *SET UsageTime = MonthlyUsage.UsageTime + ?*	*This statement is used to update the value of the "UsageTime" field in the "MonthlyUsage" table. Specifically, it increments the value of "UsageTime" by a specific value (which should be*

	WHERE MonthDate = DATEFROMPARTS(YEAR(GETDATE()), MONTH(GETDATE()), 1)	*replaced with the actual value in the SQL statement). The WHERE condition in the statement specifies that only records with the field "MonthDate" equal to the current month and year will be updated.*
03	*SELECT* *DATEFROMPARTS(M.CreatedYear, M.CreatedMonth, 1) AS MonthDate,* *ISNULL(T.UsageTime, '') AS UsageTime* *FROM* *(* *SELECT* *MONTH(MonthDate) AS CreatedMonth,* *YEAR(MonthDate) AS CreatedYear* *FROM* *(* *SELECT TOP 11* *DATEADD(MONTH, -ROW_NUMBER() OVER (ORDER BY (SELECT NULL)), GETDATE()) AS MonthDate* *FROM* *sys.objects* *UNION ALL* *SELECT* *DATEADD(MONTH, 0, GETDATE()) AS MonthDate* *) AS Subquery* *) AS M* *LEFT JOIN* *(* *SELECT TOP 12* *MonthDate,* *UsageTime* *FROM* *MonthlyUsage* *ORDER BY* *MonthDate DESC* *) AS T* *ON* *M.CreatedMonth = MONTH(T.MonthDate)*	*The SQL statement is used to query data from the table "MonthlyUsage" and return the columns "MonthDate" and "UsageTime". It combines subqueries to generate a list of recent months and then does a LEFT JOIN with the "MonthlyUsage" table to get the corresponding usage time information. The results are sorted by ascending time.*

	AND M.CreatedYear = YEAR(T.MonthDate) *ORDER BY* *M.CreatedYear ASC,* *M.CreatedMonth ASC*	
04	*SELECT* *ISNULL(T.NumUsers, 0) AS NumUsers,* *M.CreatedMonth,* *M.CreatedYear* *FROM* *(* *SELECT* *MONTH(MonthDate) AS CreatedMonth,* *YEAR(MonthDate) AS CreatedYear* *FROM* *(* *SELECT TOP 11* *DATEADD(MONTH, -ROW_NUMBER() OVER (ORDER BY (SELECT NULL)), GETDATE()) AS MonthDate* *FROM* *sys.objects* *UNION ALL* *SELECT* *DATEADD(MONTH, 0, GETDATE()) AS MonthDate* *) AS Subquery* *) AS M* *LEFT JOIN* *(* *SELECT* *COUNT(*) AS NumUsers,* *MONTH(TimeCreate) AS CreatedMonth,* *YEAR(TimeCreate) AS CreatedYear* *FROM* *UserInfor* *GROUP BY* *MONTH(TimeCreate),* *YEAR(TimeCreate)* *) AS T* *ON*	*The SQL statement is used to query the number of users ("NumUsers") from the "UserInfor" table for each month and year generated from the "M" subquery. The final results are sorted by ascending time.*

	M.CreatedMonth = T.CreatedMonth *AND M.CreatedYear = T.CreatedYear* *ORDER BY* *M.CreatedYear ASC,* *M.CreatedMonth ASC*	

15. Grant Admin Rights

a. Class diagram

b. Class Specifications

Role

No	Method	Description
01	*Role(String roleID, String roleName, boolean isLock)*	*This constructor is used to create a SearchUser object with the provided values. After calling this constructor, the properties of the SearchUser object will be set to the corresponding values.*
02	*Getters và Setters*	*Getter and Setter methods are used to access and set the value of properties in a class.*

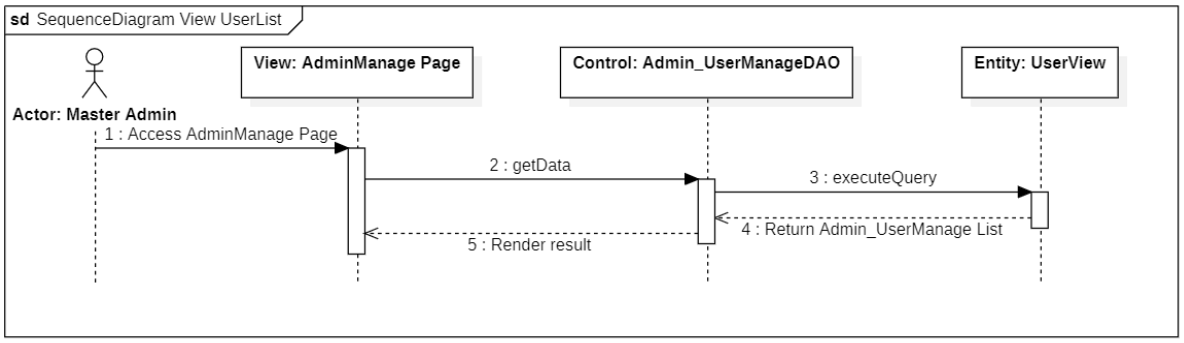
RoleDAO

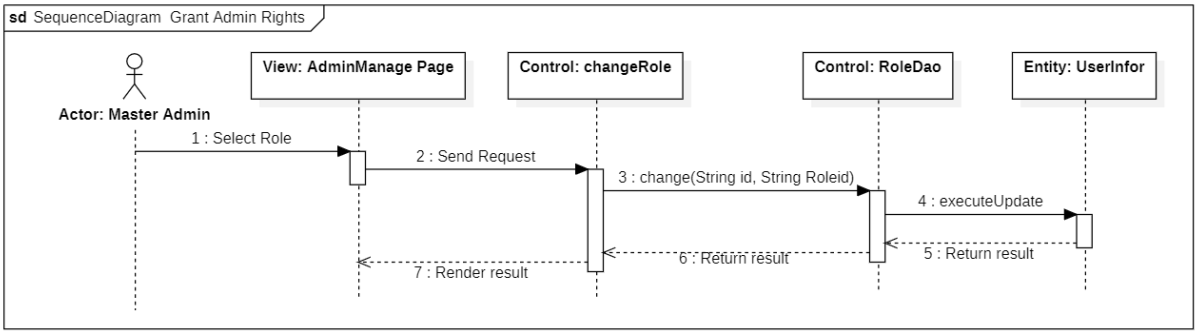
No	Method	Description
01	*RoleDao()*	*The public RoleDao() method is a constructor of the RoleDao class. It is used to initiate a connection to the database.*
02	*getRole(String UserID)*	*The `getRole(String UserID)` function is used to get the user's role information from the database. The function performs the following steps:* *- Execute SQL query to get user role from `UserInfor` table based on `UserID`.* *- Check if role is "LOCK" and user is not locked, perform role update to "USER" and delete record in `UserLock` table.* *- Returns the user's key status and role information.*
03	*change(String id, String Roleid)*	*The function `change(String id, String Roleid)` is used to change a user's role in the database.* *The function performs the following steps:* *- Prepare and execute SQL UPDATE statement to update user roles in `UserInfor` table.* *- Returns `true` if the statement is executed successfully, otherwise returns `false` in case of an error.*

Account_UserManagerDAO

No	Method	Description
01	*Admin_UserManageDAO()*	*The public Admin_UserManageDAO() method is a constructor of the Admin_UserManageDAOclass. It is used to initiate a connection to the database.*
02	*getData()*	*The getData() function is used to get user data from the database and return a list of Admin_UserManage objects.*

c. Sequence Diagram(s)





d. Database Queries

RoleDAO

No	Queries	Description
01	*DECLARE @UserID NVARCHAR(11) = ?* *DECLARE @isLock BIT = CASE* *WHEN ((SELECT TOP(1)* *DATEADD(MINUTE, LockDurationMinute,* *DATEADD(HOUR,*	*This query is used to check and update the user's role and return information about the user's role and key status.*

```
LockDurationHour,
DATEADD(DAY,
LockDurationDay, LockTime)))
        FROM UserLock
        WHERE UserID =
@UserID) > GETDATE()) THEN 1
   ELSE 0
END

DECLARE @Role VARCHAR(11)

SELECT @Role =
UserInfor.RoleID
FROM dbo.UserInfor
WHERE UserInfor.UserID =
@UserID

IF (@isLock = 0 AND @Role =
'LOCK')
BEGIN
   UPDATE dbo.UserInfor
   SET UserInfor.RoleID = 'USER'
   WHERE UserInfor.UserID =
@UserID
        DELETE FROM
dbo.UserLock WHERE
UserLock.UserID=@UserID
END

SELECT Role.RoleID, RoleName,
@isLock
FROM dbo.UserInfor
INNER JOIN dbo.Role ON
Role.RoleID = UserInfor.RoleID
WHERE UserID = @UserID


SELECT Role.RoleID, RoleName,
@isLock
FROM dbo.UserInfor
INNER JOIN dbo.Role ON
Role.RoleID = UserInfor.RoleID
WHERE UserID = @UserID
```

02	*UPDATE dbo.UserInfor SET UserInfor.RoleID=? WHERE UserInfor.UserID=?*	*This query is used to update the user's role in the UserInfor table based on the UserID.*

RoleDAO

No	Queries	Description
01	*SELECT * FROM UserView*	*This query is used to get all the data from the UserView table.*