SOFE 4790U: Distributed Systems (Fall 2023)

Instructor: Dr. Ahmed Badr

Assignment #1

Honour code: By submitting this assignment, I (name and banner id# below) affirm this is my own work, and I have not asked any of my fellow students or others for their source code or solutions to complete this assignment, and I have not offered my source code or solutions for this assignment to any of my fellow students.

Name: Alden O'Cain

Banner ID#: 100558599

1. Application idea

In an online world communication has become more important then ever to our personal and professional lives. For this assignment I built a real-time Chatroom Service using a Java-based Client-Server architecture. This service aims to provide clients with a platform for holding conversations.

2. Describe the two core functionalities

1. **Real-Time Message Broadcasting:** The server can facilitate real-time message broadcasting to ensure that messages sent by one client are quickly delivered to the server. This core functionality is essential for enabling conversations within the chatroom. The client server efficiently transfers messages back and forth, updating the chat window server side.
2. **User Registration and Authentication:** Implementing authentication functionalities is crucial for a chatroom service. The server can handle the creation and validation of user identities of users in chatrooms. This includes user registration, login. User authentication ensures that only authorized individuals can participate in the chat

3. Describe the two novel features

1. **SQLite Database Integration:** A novel feature of this chatroom service is the integration of an SQLite database, which enhances the data management of the distributed application. This technology is capable of handling message history storage, as well as database initialization, table creation, clearing data in the database's tables and more importantly, has functionality for pinning messages, adding banned words to the chat filtering system, storing user credentials, the existence of specific usernames during registration, and checking the validity of usernames and passwords.
2. **Chat Filtering:** A novel feature of this chatroom service is the integration of an content filtering system. This technology scans messages for offensive or inappropriate words based off a word list established in the SQLite Database and sanitizes the content, warning the user who violates the rules of the chatroom.

4. Challenges and solutions

1. **User Experience:** Ensuring a smooth and user-friendly experience is vital. So creating a clean command line interface to make the chatroom service intuitive and visually appealing is important to ensure a positive experience for all users.
2. **Cross-Platform Compatibility:** Another challenge is ensuring that the chatroom service is accessible across various devices and platforms. To address this, we will developed both the client and server applications in Java because it is portable and compatible with all major operating systems. Users can transition between devices using the same login credentials because they are stored server side.

5. Testing

```java
public class JavaDBTest {
    public static void main(String[] args) {
        try {
            // Create an instance of SQLiteCRUD
            SQLiteCRUD sqliteCRUD = new SQLiteCRUD();

            // Initialize the chat tables
            sqliteCRUD.clearDatabaseTables();
            sqliteCRUD.initializeDatabaseTables();

            // Create users
            sqliteCRUD.createUser( username: "user001",  password: "1234",  email: "john@gmail.com");
            sqliteCRUD.createUser( username: "user102",  password: "5678",  email: "david@hotmail.com");
            sqliteCRUD.createUser( username: "user203",  password: "9012",  email: "benjamin@yahoo.com");

            // Create a sample message
            sqliteCRUD.storeMessage( senderId: 1,  messageText: "Hello, SQLite!");

            // Retrieve and print chat messages
            System.out.println("\nChat Messages:");
            sqliteCRUD.printChatMessages();

            // Test pinning and unpinning a message
            System.out.println("\nPinned Messages:");
            sqliteCRUD.pinMessage( messageID: 1);
            sqliteCRUD.printPinnedMessages();
            System.out.println("Pinned Messages:");
            sqliteCRUD.unpinMessage( messageID: 1);
            sqliteCRUD.printPinnedMessages();

            // Test pinning and unpinning a message
            System.out.println("\nBanned Words:");
            sqliteCRUD.addBannedWord("DurhamCollege");
            sqliteCRUD.printBannedWords();
            // System.out.println("Banned Words:");
            // sqliteCRUD.removeBannedWord("DurhamCollege");
            // sqliteCRUD.printBannedWords();
        } catch (Exception e) {
            e.printStackTrace();
        }
```

The JavaDBTest class serves as a testing suite for the functions within the SQLiteCRUD class. It systematically assesses the validity of these functions by executing a series of operations and printing the results. Here's a description of the manual hard-coded tests performed in this class:

1. **SQLiteCRUD Initialization:** An instance of the SQLiteCRUD class is created, allowing for interaction with the SQLite database.
2. **Database Setup:** The database tables are cleared and then initialized. This ensures a clean and consistent starting point for the tests.
3. **User Creation:** Three user accounts are created using the createUser function. These users are identified by unique usernames, passwords, and email addresses.
4. **Message Storage:** A sample chat message is stored in the database using the storeMessage function, associating it with a specific user.
5. **Chat Message Retrieval:** The stored chat messages are retrieved and printed to the console, providing a visual check to confirm successful message storage and retrieval.
6. **Message Pinning and Unpinning:** The ability to pin and unpin a message is tested. A message is pinned and then immediately unpinned, and both pinned and unpinned messages are printed separately. This test checks the functionality of these operations.
7. **Banned Words Management:** The process of adding and viewing banned words is evaluated. A banned word, "DurhamCollege," is added to the list of banned words, and the list is printed. Note that there are commented-out lines for removing banned words; however, this part of the test is currently disabled.

```
public class UsersTest {
    public static void main(String[] args) {
        // Initialize the SQLiteCRUD instance
        SQLiteCRUD sqliteCRUD = new SQLiteCRUD();

        // Create an instance of Users and pass the SQLiteCRUD instance
        Users users = new Users(sqliteCRUD);

        // Check if a username exists
         boolean usernameExists = users.checkUsername("freeUsername");
         System.out.println("[checkUsername, freeUsername] usernameExists=" + usernameExists + " (expected false)");
         usernameExists = users.checkUsername("user001");
         System.out.println("[checkUsername, user001] usernameExists=" + usernameExists + " (expected true)");

        // Authenticate a username and password
         boolean authenticated = users.authenticateUsername( username: "user001",  password: "password123");
         System.out.println("[authenticateUsername, user001/badpass] authenticated=" + authenticated + " (expected false)");
         authenticated = users.authenticateUsername( username: "user001",  password: "1234");
         System.out.println("[authenticateUsername, user001/1234] authenticated=" + authenticated + " (expected true)");

        System.out.println("'Test' results will not be correct unless data is configured correctly prior to execution..");
        // Test creating a new user
        boolean userCreated = users.createUser( username: "user001",  password: "1234",  email: "john@gmail.com");
        System.out.println("[createUser, user001] userCreated=" + userCreated + " (expected false)");
        userCreated = users.createUser( username: "user304",  password: "3456",  email: "susan@msn.com");
        System.out.println("[createUser, user304] userCreated=" + userCreated + " (expected true)");

        // Test logging into a user
        boolean loggedIn = users.loginUser( username: "user001",  password: "badpass");
        System.out.println("[loginUser, user001/badpass] authenticated=" + loggedIn + " (expected false)");
        loggedIn = users.loginUser( username: "user304",  password: "3456");
        System.out.println("[loginUser, user304/3456] authenticated=" + loggedIn + " (expected true)");
    }
}
```

The UsersTest class serves as a comprehensive testing suite for the functions within the Users class. These manual hard-coded tests systematically assess the validity of these functions by executing various scenarios and printing the results. Here's a description of the manual tests performed in this class:

1. **SQLiteCRUD Initialization:** An instance of the SQLiteCRUD class is created, enabling interactions with the SQLite database.
2. **Users Initialization:** An instance of the Users class is created, passing the SQLiteCRUD instance as a parameter. This sets up the user management system and associates it with the database.
3. **Username Existence Check:** The checkUsername function is tested to determine whether it correctly identifies the existence of a username. First, a non-existent username, "freeUsername," is checked, and the result (false) is printed. Then, an existing username, "user001," is checked, and the result (true) is printed. These tests ensure the accuracy of the username verification process.
4. **Username and Password Authentication:** The authenticateUsername function is tested to verify whether it correctly authenticates a combination of username and password. Initially, an incorrect password is provided for "user001," and the result (false) is printed. Then, the correct password for "user001" is used, resulting in an authentication (true) and is printed. These tests validate the proper functioning of the username and password authentication system.
5. **User Creation:** The createUser function is tested to assess its ability to create new user accounts. An attempt to create a user with an already existing username ("user001") is made, and the result (false) is printed. Subsequently, a new user with a unique username ("user304") is created, and the result (true) is printed. These tests confirm the user creation process's correctness.
6. **User Login:** The loginUser function is tested to evaluate its capacity to log in users. An attempt is made to log in with the correct username but the wrong password for "user001," resulting in failure (false) and is printed. Next, successful login with the correct username and password for the newly created user "user304" is verified, resulting in successful authentication (true) and is printed.

```java
public class MessagesTest {
    public static void main(String[] args) {
        // Assume you have an SQLiteCRUD instance
        SQLiteCRUD sqliteCRUD = new SQLiteCRUD();

        // Create an instance of Messages with the SQLiteCRUD instance
        Messages messages = new Messages(sqliteCRUD);

        // Send a message
        boolean messageSent = messages.sendMessage( senderId: 4,  messageText: "This is a message that contains a banned word.. DurhamCollege");
        // System.out.println("[sendMessage] messageSent=" + messageSent);

        // Get chat messages
        System.out.println("Chat Messages:");
        messages.getMessages();

        // Pin a message
        boolean messagePinned = messages.pinMessage( messageID: 3);
        // System.out.println("[pinMessage] messagePinned=" + messagePinned);

        // Get pinned messages
        System.out.println("Pinned Messages:");
        messages.getPinnedMessages();

        // Unpin a message
        boolean messageUnpinned = messages.unpinMessage( messageID: 3);
        // System.out.println("[pinMessage] unpinMessage=" + messagePinned);

        // Get pinned messages
        System.out.println("Pinned Messages:");
        messages.getPinnedMessages();

        // Check for banned words
        System.out.println("Check for banned words:");
        boolean bannedWordsCheck = messages.checkMsgForBannedWords(messages.getIndividualMsg( messageID: 1));
        System.out.println("[checkMsgForBannedWords, clean message] bannedWordsCheck=" + bannedWordsCheck + " (expected false)");
        bannedWordsCheck = messages.checkMsgForBannedWords(messages.getIndividualMsg( messageID: 4));
        System.out.println("[checkMsgForBannedWords, dirty message] bannedWordsCheck=" + bannedWordsCheck + " (expected true)");
    }
}
```

The MessagesTest class contains manual hard-coded tests to assess the functionality of the functions within the Messages class. Here's a description of the tests conducted in this class:

1. **SQLiteCRUD Initialization:** An instance of the SQLiteCRUD class, assumed to be previously initialized, is used for interacting with the SQLite database.
2. **Messages Initialization:** An instance of the Messages class is created, taking the SQLiteCRUD instance as a parameter. This sets up the message management system and associates it with the database.
3. **Message Saving:** The sendMessage function is tested by attempting to save a message. The test message includes a banned word, "DurhamCollege.".
4. **Chat Message Retrieval:** The getMessages function is used to retrieve and print chat messages. This provides visibility into the current chat message history.
5. **Message Pinning:** The pinMessage function is tested to verify if it successfully pins a specific message.
6. **Pinned Message Retrieval:** The getPinnedMessages function is employed to retrieve and print pinned messages. This helps verify if message pinning is functioning correctly.

7. **Message Unpinning:** The unpinMessage function is tested to ascertain its ability to unpin a message.
8. **Pinned Message Retrieval After Unpinning:** The getPinnedMessages function is used again to retrieve and print pinned messages after attempting to unpin a message. This step verifies the success of the unpinning operation.
9. **Banned Words Check:** The checkMsgForBannedWords function is tested to determine if it correctly checks for banned words within a message. First, a clean message is checked for banned words, and the result (false) is printed. Next, a dirty message containing a banned word is checked, and the result (true) is printed. These tests validate the proper functioning of the banned words detection and filtering process.

   These manual hard-coded tests provide a comprehensive assessment of the functions within the SQLiteCRUD, Users and Messages classes, helping ensure that the functionalities used by the client server application are working as expected.

6. Conclusion

In conclusion, our Java-based Client-Server Chatroom Service brings two core functionalities of real-time chat and user accounts. It also introduces novel features like SQLite database integration and chat filtering. While addressing challenges related to cross platform compatibility and user experience, the chat service service aims to offer a reliable and secure platform for people to connect to a chatroom.