Authentication Use Case Descriptions (With Feedback Highlights)

Use Case: User Registration

Iteration: 2

Primary Actor: User

Goal in Context: Allow a user to create an account by providing necessary credentials.

Preconditions:

- The user device is powered on.

- The user navigates to the registration page on the website.

- The website is operating without any technical issues.

- The user has a valid email address.

- The user agrees to the terms and conditions before registering.

- The user's device must have a stable internet connection.

Triggers:

- The user selects the 'Register' option from the website homepage.

Scenario:

1. The user selects the "Register" option on the website homepage.

2. The system prompts the user to enter the name and date of birth.

   (Clarify whether this is required or optional.)

3. The system prompts the user to enter a username, password, and email address.

4. The user inputs the required credentials and submits the form.

5. The system checks if the username is unique, and the password meets security requirements.

   (e.g., minimum length, special characters)

6. If valid, the system sends an account verification email with a one-time password.

   (Specify OTP valid duration, e.g., 10 minutes.)

7. Otherwise, user will have to choose another username or password.

8. The user enters the one-time password into the given textbox.

(Clarify multiple attempts or account lockout.)

9. The system activates the user account.

10. The user is redirected to the sign-in page.

Postconditions:

- The user successfully creates an account.

- The system stores user credentials securely.

- The user can log in with the registered credentials.

- The user receives a confirmation message upon successful registration.

Exception:

- The username is already taken.

- The email is already registered.

- The password does not meet security criteria.

- The email with one time password hasn't been received.

- The email system fails to send the OTP.

- The user's device loses internet connectivity during registration.

Open Issues:

- Preventing spam registrations.

- Consider adding CAPTCHA or anti-spam mechanism.

Use Case: User Login

Iteration: 1

Primary Actor: User/Gamer

Goal in context: Allow the user to login into their account using user ID and password

Preconditions:

- The user device is powered on

- The user navigated to the website on their browser using URL

- The user device is connected with stable internet

- The website is operating without any technical issues

Triggers:

The user initiates the login process by selecting the ‘Sign in’ option from the website homepage.

Scenario:

1. The user selects the “Sign in” from the website home page.

2. The system prompts user with two textboxes to enter the username and password.

3. Both textboxes are mandatory before the user can click the login button.

4. After entering credentials and clicking login, the system verifies them against the database.

   (Clarify how verification works: encryption or hashing?)

5. The user account will be temporarily blocked after 5 wrong password attempts.

   (Clarify block duration, e.g., 30 minutes.)

6. The system displays the 2FA page where the user must enter the OTP sent to their email.

   (Clarify how long OTP is valid, e.g., 5 minutes.)

7. The system will allow OTP resend if user didn’t receive it.

   (Clarify: resend to same email or alternate?)

8. The user will see menu options after successful login.

Postconditions:

- The user is able to sign into the system.

- The user can access/view their profile info.

- User login activity is logged for security purposes.

Exception:

- Website unresponsive.

- Website-server communication fails.

- OTP not received or fails even after correct entry.

- Device loses internet connectivity during login.

- Account permanently blocked after repeated failed attempts.

Open Issues:

- Handle temp blocks due to failed attempts.

- Handle OTP failures.

Use Case: Forget Password

Iteration: 1

Primary Actor: User/Gamer

Goal in context: Allow the user to reset their password

Preconditions:

- The user device is powered on

- The user navigated to the website on their browser using URL

- The user device is connected with stable internet

- The website is operating without any technical issues

Triggers:

The user forgets the password

Scenario:

1. The user selects ‘Sign in’ from homepage.

2. The user selects ‘forget password’ from sign in page.

3. The system prompts for the username.

   (Clarify behavior if username doesn’t exist.)

4. The user clicks the reset password button.

5. System asks user to input OTP sent to email.

   (Specify how long OTP is valid, e.g., 10 minutes.)

6. User can request OTP resend.

7. User inputs new password twice and clicks submit.

   (Clarify password complexity requirements.)

8. User redirected to sign-in page if reset is successful.

Postconditions:

- The user is able to reset their password.

- The user is able to log in again.

- User password reset activity is logged for security purposes.

Exception:

- Website unresponsive or server issues.

- OTP not received or incorrect.

- Device loses internet during reset process.

- Account locked due to multiple failed reset attempts.

Open Issues:

- How to help users who lost access to their email.

- Allow identity verification with secondary email or phone

Use Case: Reset Password

Iteration: 1
Primary Actor: User
Goal in context: Allow the user to reset the password from the user profile settings

Preconditions:

- The user is signed into the account.
- The user navigates to the profile settings.

Triggers:

- The user selects the reset password option from the profile setting page.

Scenario:

- The user selects the reset password from the profile settings page.
- The user is prompted with two textboxes to enter the new password and verify it.
- The user will receive a "password reset successfully" message when the password meets the complexity requirements. Otherwise, the system will ask the user to select another password. (in red: clarified complexity check and added fallback behavior)
- If the two passwords entered do not match, the system displays an error message and prompts the user to try again.

Postconditions:

- The password on the user account is reset.
- The user remains signed in to the account.
- The user is required to use the new password for future sign-in attempts.
- The system logs the password reset activity for security purposes.

Exceptions:

- The system stops responding once the user hits reset after entering the password in the textboxes.

- The system displays an error and asks the user to enter the value again.

- If the user's device loses internet connectivity during the reset process, the system displays a connection error and halts the process.

- If the user enters incorrect passwords too many times, the system temporarily locks the account.

- Priority: Low
  When Available: User is already signed in to the account.
  Frequency of Use: Rare
  Channel to Actors: Web browser, system UI.
  Secondary Actors: None
  Channel to Secondary Actors: None
  Open Issues: None

Use Case: Logout

- Iteration: 1
  Primary Actor: User
  Goal in context: Allow the user to securely log out of their account, ensuring the "Remember Me" mechanism is handled correctly.

Preconditions:

- The user is logged in.

- The website is operating normally.

- The user may have previously selected the "Remember Me" option.

Triggers:

- The user selects the "Logout" option.

Scenario:

- The user selects the "Logout" option from the dashboard.
- The system terminates the session by invalidating the session on the server and deleting session cookies.
- The system checks if the user had enabled the "Remember Me" feature.
- If "Remember Me" was enabled, the system deletes the stored authentication token from both the client side and server side.
- The system redirects the user to the login page.
- The system performs a server-side check to confirm that no active session or authentication token remains.

Postconditions:

- The user is successfully logged out.
- The session is securely terminated.
- Any "Remember Me" authentication token is invalidated, requiring re-authentication next time.
- The system logs the logout activity for auditing purposes.

Exceptions:

- The system fails to terminate the session properly.
- The "Remember Me" token persists after logout, allowing unintended access.
- The user session remains active beyond logout.
- If the user's device loses internet connectivity, the logout process may fail, prompting the user to retry.
- If the user's session is hijacked, the system should detect this and block the request.
- Priority: Medium
  When Available: Always available on the user dashboard.
  Frequency of Use: Once per session.
  Channel to Actors: Web browser, system UI.
  Secondary Actors: Session manager, authentication token management system.
  Channel to Secondary Actors: None

Open Issues:

- Ensuring all authentication tokens are invalidated properly upon logout.
- Handling cases where token deletion fails.
- Implementing forced logout for long-lived "Remember Me" sessions if necessary.
- Adding an option for the user to log out of all devices if they suspect unauthorized access.

Use Case: "Remember me"

Iteration: 1

Primary Actor: User

Goal in Context: Identify a user once, so that the user doesn't have to re-login every time a user is accessing the game. A token will be created as identification when the user logins, and a time to live will be associated with the token

Preconditions:

· The user is on the login page about to login.

· The user is typing their account credentials.

Triggers:

· The user ticks the “Remember me” checkbox.

· The user presses the login button to log in.

Scenario:

1. The user is in the login page.
2. The user types in their account credentials in the available input fields.
3. The user ticks the “Remember me” checkbox.
4. The user presses the login button, and logs in to their account.
5. If the user does not tick the “Remember me” checkbox, the system skips token creation. 6. If selected, the system creates an identification token on the server side and stores a reference on the client device.

Postconditions:

· The user entered the correct account credentials details.

· The user logins successfully without any errors.

· An identification token is created to identify the user, so that they don’t have to re-login each time. But a time to live is associated with the token – token will be invalid after certain amount of life span.

· The user’s login activity is logged for security purposes.

Exception:

· The user enters the wrong account credentials; thus, login is unsuccessful.

· The user attempts a failed login, and so an identification token is not created to identify the user – to prevent re-login every time the user accesses the game.

· The token has expired, and the user has to re-login.

· The user’s device loses internet connectivity during login.

· The token is compromised and used by another user.

Priority: Medium

When Available: Always available in the login screen as an option.

Frequency of Use: Every new login attempt is made.

Channel to Actors: System login page.

Secondary Actors: Token identification management

Open Issues:

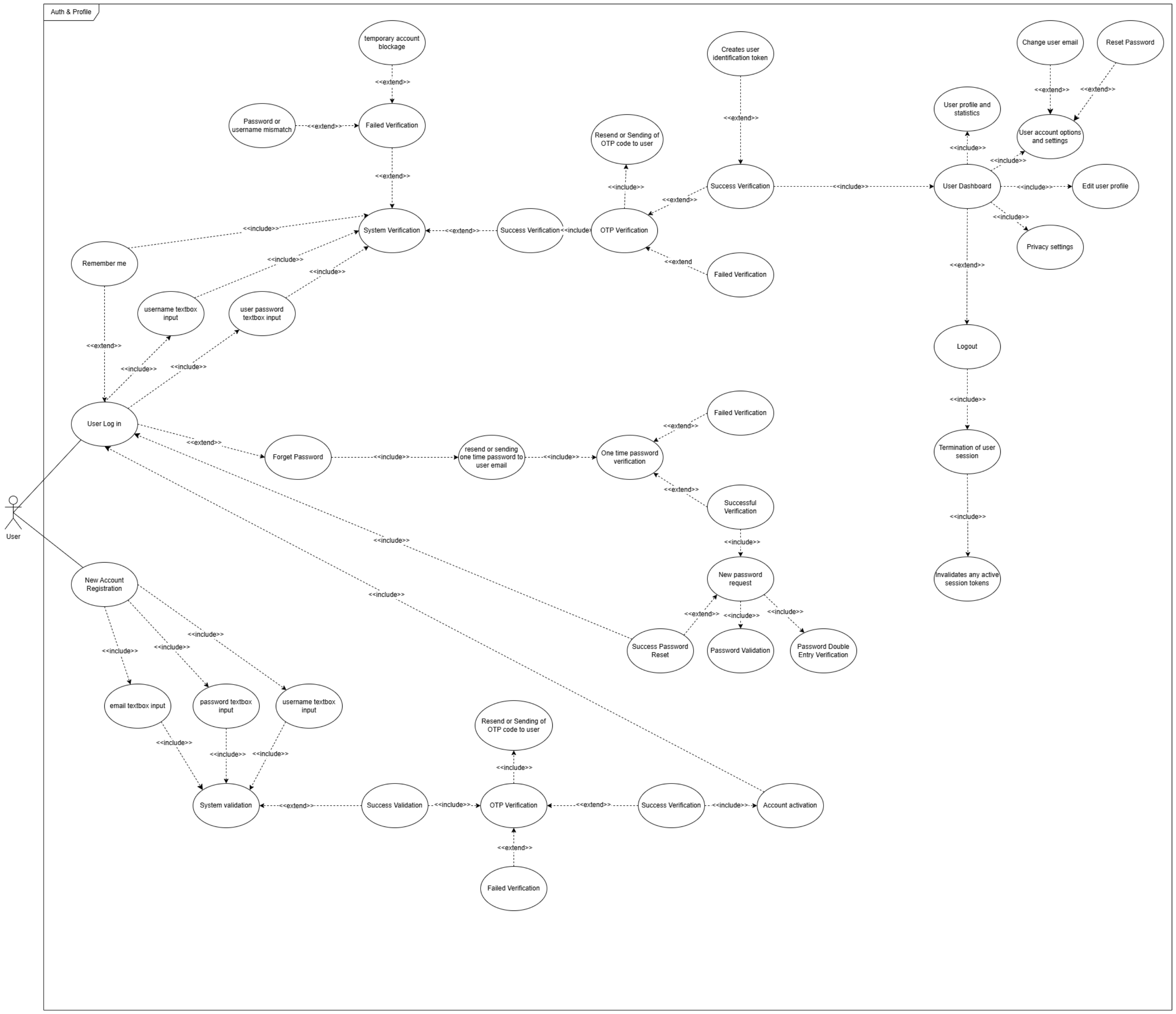
· Creating a mechanism on implementing the identification token.

· Ensuring that the identification works as intended.

· Making sure that the tokens are destroyed when the user logs out.

· Consider adding an option for the user to revoke the “Remember me” token (e.g., if they suspect unauthorized access).

Use Case diagram updated based on p2 feedback



Authentication Class Structure Feedback

1. AuthManager Password Handling

- The `hashPassword(password, salt)` method should return a `byte[]` instead of `String`, as password hashes should not be stored as plain strings.

- Consider adding a `verifyPassword(inputPassword, storedHash, salt)` method for authentication instead of directly comparing hashes.

Reply: We agree that password hashes should not be stored or compared as plaintext, and ideally `hashPassword(password, salt)` should return a `byte[]` with secure comparison using a `verifyPassword(inputPassword, storedHash, salt)`

However, Since implementing a secure database and persistent storage was outside of the scope of our assignment, we chose to store the password hash as a String for simplicity during development and testing

That being said, this is valuable feedback and aligns with standard security practices. It is high up on our project improvement list.

---

2. Lack of Associations in ProfileManager

- `ProfileManager` relies only on usernames. If `User` objects need to be retrieved often, consider maintaining a reference to `DatabaseStub` instead of just storing usernames.

Reply: `ProfileManager` was designed to manage user-specific game data, primarily rankings and history. It intentionally stores only `Username` references.

We reasoned that since usernames are guaranteed to be unique, they could serve as reliable keys for tracking profile data. Direct access to `User` objects or `DatabaseStub` objects was avoided to preserve the separation of concerns as database-level functionality is outside of our scope of responsibility

Any further `User` retrieval is handled at a higher level (e.g `AuthManager`), which helps us keep `ProfileManager` focused and modular.

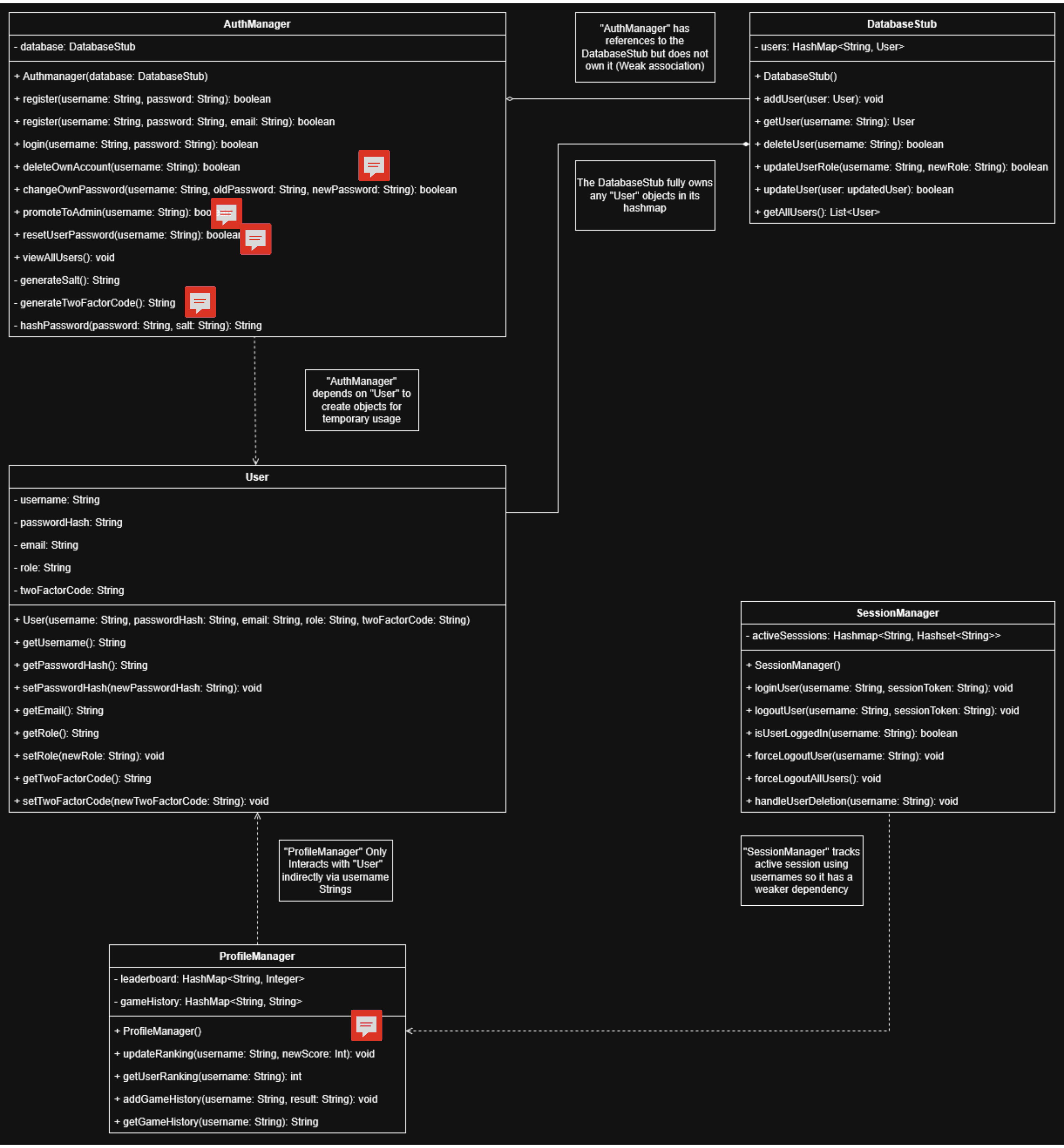
---

3. Missing Constraints in SessionManager

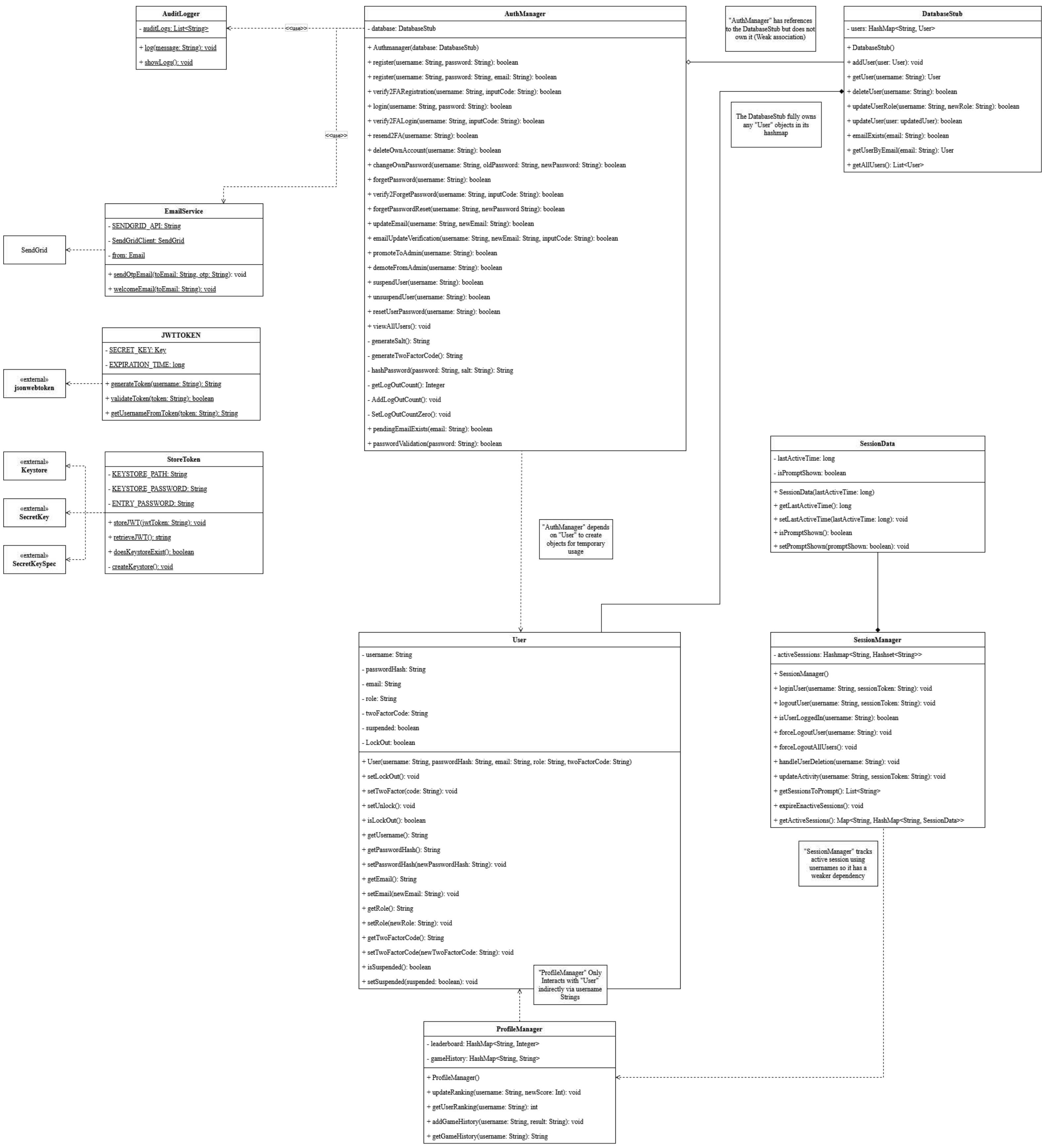
- Does `SessionManager` limit the number of concurrent sessions per user?
- Adding a max session limit per user would improve security.

Reply: We considered the possibility of introducing a maximum concurrent session limit, but ultimately decided against it. We reasoned that a strict session limit could negatively impact user experience, particularly for users who access the platform across multiple devices or frequently switch between devices

To mitigate potential security risks, we implemented two-factor Authentication and a feature that detects and expires any inactive sessions. Given our current scope and existing security measures, we prioritized user flexibility over enforcing a session cap.


AuthManager
- database: DatabaseStub
+ Authmanager(database: DatabaseStub)
+ register(username: String, password: String): boolean
+ register(username: String, password: String, email: String): boolean
+ login(username: String, password: String): boolean
+ deleteOwnAccount(username: String): boolean
+ changeOwnPassword(username: String, oldPassword: String, newPassword: String): boolean
+ promoteToAdmin(username: String): boo
+ resetUserPassword(username: String): boolear
+ viewAllUsers(): void
- generateSalt(): String
- generateTwoFactorCode(): String
- hashPassword(password: String, salt: String): String
"AuthManager" has references to the DatabaseStub but does not own it (Weak association)
DatabaseStub
- users: HashMap<String, User>
+ DatabaseStub()
+ addUser(user: User): void
+ getUser(username: String): User
+ deleteUser(username: String): boolean
+ updateUserRole(username: String, newRole: String): boolean
+ updateUser(user: updatedUser): boolean
+ getAllUsers(): List<User>
The DatabaseStub fully owns any "User" objects in its hashmap
"AuthManager" depends on "User" to create objects for temporary usage
User
- username: String
- passwordHash: String
- email: String
- role: String
- twoFactorCode: String
+ User(username: String, passwordHash: String, email: String, role: String, twoFactorCode: String)
+ getUsername(): String
+ getPasswordHash(): String
+ setPasswordHash(newPasswordHash: String): void
+ getEmail(): String
+ getRole(): String
+ setRole(newRole: String): void
+ getTwoFactorCode(): String
+ setTwoFactorCode(newTwoFactorCode: String): void
SessionManager
- activeSesssions: Hashmap<String, Hashset<String>>
+ SessionManager()
+ loginUser(username: String, sessionToken: String): void
+ logoutUser(username: String, sessionToken: String): void
+ isUserLoggedIn(username: String): boolean
+ forceLogoutUser(username: String): void
+ forceLogoutAllUsers(): void
+ handleUserDeletion(username: String): void
"ProfileManager" Only Interacts with "User" indirectly via username Strings
"SessionManager" tracks active session using usernames so it has a weaker dependency
ProfileManager
- leaderboard: HashMap<String, Integer>
- gameHistory: HashMap<String, String>
+ ProfileManager()
+ updateRanking(username: String, newScore: Int): void
+ getUserRanking(username: String): int
+ addGameHistory(username: String, result: String): void
+ getGameHistory(username: String): String

Updated Class Structure Diagram including P2 Feedback


AuditLogger
- auditLogs: List<String>
+ log(message: String): void
+ showLogs(): void
<<use>>
<<use>>
AuthManager
- database: DatabaseStub
+ Authmanager(database: DatabaseStub)
+ register(username: String, password: String): boolean
+ register(username: String, password: String, email: String): boolean
+ verify2FARegistration(username: String, inputCode: String): boolean
+ login(username: String, password: String): boolean
+ verify2FALogin(username: String, inputCode: String): boolean
+ resend2FA(username: String): boolean
+ deleteOwnAccount(username: String): boolean
+ changeOwnPassword(username: String, oldPassword: String, newPassword: String): boolean
+ forgetPassword(username: String): boolean
+ verify2ForgetPassword(username: String, inputCode: String): boolean
+ forgetPasswordReset(username: String, newPassword String): boolean
+ updateEmail(username: String, newEmail: String): boolean
+ emailUpdateVerification(username: String, newEmail: String, inputCode: String): boolean
+ promoteToAdmin(username: String): boolean
+ demoteFromAdmin(username: String): boolean
+ suspendUser(username: String): boolean
+ unsuspendUser(username: String): boolean
+ resetUserPassword(username: String): boolean
+ viewAllUsers(): void
- generateSalt(): String
- generateTwoFactorCode(): String
- hashPassword(password: String, salt: String): String
- getLogOutCount(): Integer
- AddLogOutCount(): void
- SetLogOutCountZero(): void
+ pendingEmailExists(email: String): boolean
+ passwordValidation(password: String): boolean
"AuthManager" has references to the DatabaseStub but does not own it (Weak association)
DatabaseStub
- users: HashMap<String, User>
+ DatabaseStub()
+ addUser(user: User): void
+ getUser(username: String): User
+ deleteUser(username: String): boolean
+ updateUserRole(username: String, newRole: String): boolean
+ updateUser(user: updatedUser): boolean
+ emailExists(email: String): boolean
+ getUserByEmail(email: String): User
+ getAllUsers(): List<User>
The DatabaseStub fully owns any "User" objects in its hashmap
EmailService
- SENDGRID_API: String
- SendGridClient: SendGrid
- from: Email
+ sendOtpEmail(toEmail: String, otp: String): void
+ welcomeEmail(toEmail: String): void
SendGrid
JWTTOKEN
- SECRET_KEY: Key
- EXPIRATION_TIME: long
+ generateToken(username: String): String
+ validateToken(token: String): boolean
+ getUsernameFromToken(token: String): String
«external» jsonwebtoken
StoreToken
- KEYSTORE_PATH: String
- KEYSTORE_PASSWORD: String
- ENTRY_PASSWORD: String
+ storeJWT(jwtToken: String): void
+ retrieveJWT(): string
+ doesKeystoreExist(): boolean
- createKeystore(): void
«external» Keystore
«external» SecretKey
«external» SecretKeySpec
SessionData
- lastActiveTime: long
- isPromptShown: boolean
+ SessionData(lastActiveTime: long)
+ getLastActiveTime(): long
+ setLastActiveTime(lastActiveTime: long): void
+ isPromptShown(): boolean
+ setPromptShown(promptShown: boolean): void
"AuthManager" depends on "User" to create objects for temporary usage
User
- username: String
- passwordHash: String
- email: String
- role: String
- twoFactorCode: String
- suspended: boolean
- LockOut: boolean
+ User(username: String, passwordHash: String, email: String, role: String, twoFactorCode: String)
+ setLockOut(): void
+ setTwoFactor(code: String): void
+ setUnlock(): void
+ isLockOut(): boolean
+ getUsername(): String
+ getPasswordHash(): String
+ setPasswordHash(newPasswordHash: String): void
+ getEmail(): String
+ setEmail(newEmail: String): void
+ getRole(): String
+ setRole(newRole: String): void
+ getTwoFactorCode(): String
+ setTwoFactorCode(newTwoFactorCode: String): void
+ isSuspended(): boolean
+ setSuspended(suspended: boolean): void
"ProfileManager" Only Interacts with "User" indirectly via username Strings
SessionManager
- activeSesssions: Hashmap<String, Hashset<String>>
+ SessionManager()
+ loginUser(username: String, sessionToken: String): void
+ logoutUser(username: String, sessionToken: String): void
+ isUserLoggedIn(username: String): boolean
+ forceLogoutUser(username: String): void
+ forceLogoutAllUsers(): void
+ handleUserDeletion(username: String): void
+ updateActivity(username: String, sessionToken: String): void
+ getSessionsToPrompt(): List<String>
+ expireEnactiveSessions(): void
+ getActiveSessions(): Map<String, HashMap<String, SessionData>>
"SessionManager" tracks active session using usernames so it has a weaker dependency
ProfileManager
- leaderboard: HashMap<String, Integer>
- gameHistory: HashMap<String, String>
+ ProfileManager()
+ updateRanking(username: String, newScore: Int): void
+ getUserRanking(username: String): int
+ addGameHistory(username: String, result: String): void
+ getGameHistory(username: String): String