Authentication Canister Documentation

This document provides a detailed overview of the Authentication Motoko backend canister, outlining its public functions, data types, and typical usage flows.

Type Definitions

User

Represents a user's profile information.

- **username**: Text - The user's chosen username.
- **email**: Text - The user's email address.
- **github**: Text - (Optional) The user's GitHub username.
- **slack**: Text - (Optional) The user's Slack username.
- **principal**: Principal - The unique Internet Identity principal associated with the user.
- **tablesCreated**: [Nat] - An array of Nat (IDs) representing tables the user has created.
- **tablesJoined**: [Nat] - An array of Nat (IDs) representing tables the user has joined as a collaborator.

Public Functions

register_or_update(username : Text, email : Text, github : Text, slack : Text)

- **Description**: Registers a new user or updates an existing user's profile. If the msg.caller (Internet Identity principal) does not have a registered profile, a new one is created. If a profile already exists for the msg.caller, its details are updated.
- **Arguments**:
  - username: Text - The desired username. Must not be empty.
  - email: Text - The user's email address. Must not be empty.
  - github: Text - The user's GitHub username (can be empty if not provided).
  - slack: Text - The user's Slack username (can be empty if not provided).
- **Return Value**: async Result<Text, Text>
  - ok("Profile registered/updated successfully."): Returns a success message.
  - err("Username and email are required."): Returns an error if username or email is empty.
- **Dependent/Related Functions**: None directly, interacts with internal storage.

update_user_add_table(principal : Principal, tableId : Nat, tableType:

Text)

- **Description**: This function is primarily intended for **inter-canister communication** (e.g., from the TableManagement canister) to update a user's record by adding a tableId to either their tablesCreated or tablesJoined list.
- **Arguments**:
  - principal: Principal - The principal ID of the user whose record is to be updated.
  - tableId: Nat - The ID of the table to add.
  - tableType: Text - A string indicating which list to update: "tablesCreated" or "tablesJoined".
- **Return Value**: async Result<Text, Text>
  - ok("User successfully updated"): Returns a success message.
  - err("Principal not a registered user"): Returns an error if the provided principal does not correspond to a registered user.
- **Dependent/Related Functions**:
  - Used by TableManagement.create_table and TableManagement.accept_join_table.

update_user_remove_table(principal : Principal, tableId : Nat, tableType: Text)

- **Description**: Similar to update_user_add_table, this function is primarily for **inter-canister communication** to update a user's record by removing a tableId from either their tablesCreated or tablesJoined list.
- **Arguments**:
  - principal: Principal - The principal ID of the user whose record is to be updated.
  - tableId: Nat - The ID of the table to remove.
  - tableType: Text - A string indicating which list to update: "tablesCreated" or "tablesJoined".
- **Return Value**: async Result<Text, Text>
  - ok("User successfully updated"): Returns a success message.
  - err("Principal not a registered user"): Returns an error if the provided principal does not correspond to a registered user.
- **Dependent/Related Functions**:
  - Used by TableManagement.delete_table and TableManagement.leave_table.

get_profile()

- **Description**: Retrieves the profile details of the currently authenticated caller.
- **Arguments**: None.
- **Return Value**: async ?User
  - ?User: Returns the User object of the caller if found.
  - null: Returns null if the caller's principal is not associated with a registered user.
- **Dependent/Related Functions**: None directly, interacts with internal storage.

get_all_users()

- **Description**: Retrieves a list of all registered users in the canister. This is a query function, meaning it doesn't modify the canister state and is typically faster.
- **Arguments**: None.
- **Return Value**: async [User]
  - [User]: Returns an array of all User objects.
- **Dependent/Related Functions**: None directly, iterates over internal storage.

get_user_by_principal(userPrincipal : Principal)

- **Description**: Retrieves a single user's profile details by their principal ID. This is a query function.
- **Arguments**:
  - userPrincipal: Principal - The principal ID of the user to retrieve.
- **Return Value**: async ?User
  - ?User: Returns the User object if found.
  - null: Returns null if no user with the given userPrincipal exists.
- **Dependent/Related Functions**:
  - Used extensively by the TableManagement canister for various operations requiring user details (e.g., create_table, delete_table, get_user_tables, leave_table, get_table_collaborators, request_join_table, accept_join_table, cancel_join_request, reject_join_request, get_pending_sent_requests, get_pending_recieved_requests).

User Story

Here's a user story illustrating a typical flow of data and interactions using the Authentication canister functions:

1. **Alice Registers for the First Time**: Alice, a new user, visits the application and decides to sign up. She provides her username and email, and optionally her GitHub and Slack handles.
   - *Frontend Action*: Calls Auth.register_or_update("AliceUser", "alice@example.com", "alicegh", "aliceslack").
   - *Canister Response*: The canister checks if Alice's Internet Identity principal is already registered. Since it's her first time, a new User record is created with her provided details and empty tablesCreated and tablesJoined lists. A success message is returned.
   - *Frontend Update*: Displays a welcome message and navigates Alice to her dashboard.
2. **Alice Views Her Profile**: Alice wants to see her registered details.
   - *Frontend Action*: Calls Auth.get_profile().
   - *Canister Response*: The canister uses msg.caller (Alice's principal) to retrieve her User

object.
    - *Frontend Update*: Displays Alice's username, email, GitHub, and Slack details.
3. **Alice Creates a Table (Inter-canister interaction)**: Alice uses the TableManagement canister to create a new project table.
    - *Frontend Action (indirect)*: Calls TableManagement.create_table("My First Project", "A place to organize tasks.").
    - *Canister Interaction (behind the scenes)*: The TableManagement canister, after creating the table, calls Auth.update_user_add_table(Alice's Principal, newTableId, "tablesCreated") to update Alice's profile in the Authentication canister.
    - *Authentication Canister Response*: Alice's User record is updated to include the newTableId in her tablesCreated array.
    - *Frontend Update*: The TableManagement canister returns the new table, which is then displayed in Alice's created tables list.

4. **Alice Updates Her Slack Handle**: Alice recently changed her Slack username.
    - *Frontend Action*: Calls Auth.register_or_update("AliceUser", "alice@example.com", "alicegh", "new_alice_slack").
    - *Canister Response*: The canister recognizes Alice's principal, retrieves her existing User record, updates the slack field (and any other provided fields), while preserving her existing tablesCreated and tablesJoined lists.
    - *Frontend Update*: Confirms the profile update and displays the new Slack handle.
5. **Admin Views All Users**: An administrator needs a list of all users registered in the system.
    - *Frontend Action*: Calls Auth.get_all_users().
    - *Canister Response*: The canister retrieves and returns an array of all User objects stored within it.
    - *Frontend Update*: Displays a comprehensive list of all registered users and their details.