Table Management Canister Documentation

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

Type Definitions

Table

Represents a single collaborative table.

- **id**: Nat - A unique identifier for the table.
- **title**: Text - The name of the table.
- **creator**: Principal - The principal ID of the user who created the table.
- **description**: Text - A detailed description of the table's purpose.
- **tableCollaborators**: [Principal] - An array of principal IDs of all users currently collaborating on the table, including the creator.

UserTables

A structured type returned when querying a user's tables, separating them by ownership.

- **created**: [Table] - An array of Table objects that the user has created.
- **joined**: [Table] - An array of Table objects that the user has joined as a collaborator.

Public Functions

create_table(title : Text, description : Text)

- **Description**: Creates a new collaborative table. The caller of this function automatically becomes the creator and the first collaborator of the table. Requires the caller to be a registered user.
- **Arguments**:
  - title: Text - The desired title for the new table. Must not be empty.
  - description: Text - A description for the new table. Must not be empty.
- **Return Value**: async Result<Table, Text>
  - ok(Table): Returns the newly created Table object upon successful creation.
  - err(Text): Returns an error message if the title or description is empty, or if the caller's

principal is not associated with a registered user.

- **Dependent/Related Functions**:
  - Auth.get_user_by_principal: Used to verify the caller's registration status.
  - Auth.update_user_add_table: Updates the caller's record in the Authentication canister to add the new table to their tablesCreated list.

delete_table(tableId : Nat)

- **Description**: Deletes an existing table. Only the creator of the table can perform this action. Upon deletion, the table is removed from the system, it's removed from the tablesCreated list of the creator, and from the tablesJoined list of all its collaborators. Any pending join requests associated with this table are also removed.
- **Arguments**:
  - tableId: Nat - The ID of the table to be deleted.
- **Return Value**: async Result<Table, Text>
  - ok(Table): Returns the Table object that was deleted.
  - err(Text): Returns an error message if the table does not exist, the caller is not a registered user, or the caller is not authorized (i.e., not the table's creator).
- **Dependent/Related Functions**:
  - Auth.get_user_by_principal: Verifies the caller's registration.
  - Auth.update_user_remove_table: Removes the table from the tablesCreated list of the creator and tablesJoined lists of all collaborators.
  - arrayContains (helper): Checks if an array contains a specific element.

get_table(tableId : Nat)

- **Description**: Retrieves a single table by its unique ID. This is a query function, meaning it doesn't modify the canister state and is typically faster.
- **Arguments**:
  - tableId: Nat - The ID of the table to retrieve.
- **Return Value**: async ?Table
  - ?Table: Returns the Table object if found.
  - null: Returns null if no table with the given tableId exists.
- **Dependent/Related Functions**: None directly, interacts with internal storage.

get_user_tables()

- **Description**: Fetches all tables associated with the authenticated caller, categorized into tables they have created and tables they have joined.
- **Arguments**: None.
- **Return Value**: async Result<UserTables, Text>

    - ok(UserTables): Returns a UserTables object containing arrays of Table objects.
    - err(Text): Returns an error message if the caller's principal is not associated with a registered user.
- **Dependent/Related Functions**:
    - Auth.get_user_by_principal: Used to retrieve the caller's user record and their created/joined table IDs.

leave_table(tableId : Nat)

- **Description**: Allows an authenticated user to leave a table they are a collaborator on. A user cannot leave a table they have created. This removes the user from the table's tableCollaborators list and updates their tablesJoined list in the Authentication canister.
- **Arguments**:
    - tableId: Nat - The ID of the table the caller wishes to leave.
- **Return Value**: async Result<[Nat], Text>
    - ok([Nat]): Returns the updated list of Nat (table IDs) from the user's tablesJoined list on success.
    - err(Text): Returns an error message if the table does not exist, the caller is not a registered user, the caller is the table's creator, or the caller is not currently a collaborator.
- **Dependent/Related Functions**:
    - Auth.get_user_by_principal: Verifies caller's registration.
    - Auth.update_user_remove_table: Removes the table from the caller's tablesJoined list.
    - arrayContains (helper): Checks if the user is a creator or collaborator.

get_table_collaborators(tableId : Nat)

- **Description**: Retrieves a list of all User objects who are collaborating on a specific table. This is useful for displaying table members.
- **Arguments**:
    - tableId: Nat - The ID of the table whose collaborators are to be fetched.
- **Return Value**: async Result<[User], Text>
    - ok([User]): Returns an array of User objects (from the Authentication canister) representing the collaborators.
    - err(Text): Returns an error message if the table does not exist.
- **Dependent/Related Functions**:
    - Auth.get_user_by_principal: Used to fetch User details for each collaborator principal.

request_join_table(userPrincipal : Principal, tableId : Nat)

- **Description**: Initiates a request for a specific user (userPrincipal) to join a table. This function can only be called by the **creator** of the tableId. The request is stored as pending until

accepted or rejected.

- **Arguments**:
  - userPrincipal: Principal - The principal ID of the user being invited to join the table.
  - tableId: Nat - The ID of the table for which the join request is being made.
- **Return Value**: async Result<Text, Text>
  - ok(Text): Returns a success message ("Join request sent successfully.")
  - err(Text): Returns an error message if the caller or recipient is not a registered user, the table does not exist, the caller is not the table's creator, the invited user is already a collaborator, or a join request for this user and table already exists.
- **Dependent/Related Functions**:
  - Auth.get_user_by_principal: Verifies both caller and recipient registration.
  - arrayContains (helper): Checks if the caller is the creator and if the user is already a collaborator.

accept_join_table(tableId : Nat)

- **Description**: Allows an authenticated user to accept a pending join request for a specific table. Upon acceptance, the user is added to the table's tableCollaborators list, the table is added to their tablesJoined list in the Authentication canister, and the pending request is removed.
- **Arguments**:
  - tableId: Nat - The ID of the table for which the join request is being accepted.
- **Return Value**: async Result<[Nat], Text>
  - ok([Nat]): Returns the updated list of Nat (table IDs) from the user's tablesJoined list on success.
  - err(Text): Returns an error message if the caller is not a registered user, no pending request exists for the caller and this table, the table does not exist, or the user is already a collaborator.
- **Dependent/Related Functions**:
  - Auth.get_user_by_principal: Verifies caller's registration.
  - Auth.update_user_add_table: Adds the table to the caller's tablesJoined list.
  - arrayContains (helper): Checks if the user is already a collaborator.

cancel_join_request(userPrincipal : Principal, tableId : Nat)

- **Description**: Allows the table creator to cancel a previously sent join request for a specific user and table.
- **Arguments**:
  - userPrincipal: Principal - The principal ID of the invited user whose request is being cancelled.
  - tableId: Nat - The ID of the table associated with the request.

- **Return Value**: async Result<Text, Text>
  - ok(Text): Returns a success message ("Join request cancelled.")
  - err(Text): Returns an error message if the caller is not a registered user, the caller is not the table's creator, or the pending request does not exist.
- **Dependent/Related Functions**:
  - Auth.get_user_by_principal: Verifies caller's registration.
  - arrayContains (helper): Checks if the caller is the table creator.

reject_join_request(tableId : Nat)

- **Description**: Allows a user to reject a pending join request they have received for a specific table.
- **Arguments**:
  - tableId: Nat - The ID of the table for which the join request is being rejected.
- **Return Value**: async Result<Text, Text>
  - ok(Text): Returns a success message ("Join request rejected.")
  - err(Text): Returns an error message if the caller is not a registered user or no pending request exists for the caller and this table.
- **Dependent/Related Functions**:
  - Auth.get_user_by_principal: Verifies caller's registration.

get_pending_sent_requests(tableId : Nat)

- **Description**: Retrieves the usernames of all users for whom the caller (as the table creator) has sent pending join requests for a specific table.
- **Arguments**:
  - tableId: Nat - The ID of the table to check for sent requests.
- **Return Value**: async Result<[Text], Text>
  - ok([Text]): Returns an array of usernames (Text) of the users with pending sent requests.
  - err(Text): Returns an error message if the caller is not a registered user or is not authorized (i.e., not the table's creator).
- **Dependent/Related Functions**:
  - Auth.get_user_by_principal: Fetches the caller's user record and recipient usernames.
  - arrayContains (helper): Checks if the caller is the table creator.

get_pending_recieved_requests()

- **Description**: Retrieves the IDs of all tables for which the authenticated caller has received pending join requests. This function is useful for a user to see invitations they need to respond to.
- **Arguments**: None.

- **Return Value**: async Result<[Nat], Text>
  - ok([Nat]): Returns an array of Nat (table IDs) for which the caller has received requests.
  - err(Text): Returns an error message if the caller is not a registered user.
- **Dependent/Related Functions**:
  - Auth.get_user_by_principal: Verifies caller's registration.

get_all_tables()

- **Description**: Retrieves a list of all tables currently stored in the canister. This function can be used for public table discovery or administrative purposes. This is a query function.
- **Arguments**: None.
- **Return Value**: async Result<[Table], Text>
  - ok([Table]): Returns an array of all Table objects.
  - err(Text): Returns an error message if no tables are found in the canister.
- **Dependent/Related Functions**: None directly, iterates over internal storage.

User Story

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

1. **Sarah Creates a New Project Table**: Sarah, a registered user, logs into the frontend application. She needs a new table to manage her "Website Redesign" project.
   - *Frontend Action*: Calls create_table("Website Redesign", "Tasks and progress for the company's new website.").
   - *Canister Response*: The canister processes the request, assigns a unique ID (e.g., 101), stores the new table, and automatically adds Sarah as the creator and initial tableCollaborator. It also updates Sarah's user record in the Auth canister via Auth.update_user_add_table to include this table in her tablesCreated list.
   - *Frontend Update*: Displays the newly created table in Sarah's "Created Tables" section.
2. **Sarah Views Her Tables**: Sarah wants to see all the tables she's involved with.
   - *Frontend Action*: Calls get_user_tables().
   - *Canister Response*: The canister fetches Sarah's user data from Auth and then retrieves all tables corresponding to the IDs in her tablesCreated and tablesJoined lists. It returns a UserTables object.
   - *Frontend Update*: Shows "Website Redesign" under "Tables I Created".
3. **David Discovers Tables**: David, another registered user, is looking for interesting projects.
   - *Frontend Action*: Calls get_all_tables().
   - *Canister Response*: The canister returns a list of all publicly available tables, including

Sarah's "Website Redesign" table.
    - *Frontend Update*: Displays a list of tables David can potentially join.
4. **Sarah Invites David to Collaborate**: Sarah needs David's expertise on the "Website Redesign" project. She finds David's principal ID.
    - *Frontend Action*: Calls request_join_table(David's Principal ID, 101).
    - *Canister Response*: The canister verifies Sarah is the creator of Table 101 and that David is a registered user not already a collaborator. It then stores a pending join request.
    - *Frontend Update*: Provides a confirmation that the invitation was sent. Sarah can later check get_pending_sent_requests(101) to see David's username.
5. **David Sees Pending Invitations**: David logs in and checks for new notifications.
    - *Frontend Action*: Calls get_pending_recieved_requests().
    - *Canister Response*: The canister iterates through all pending requests and identifies those addressed to David's principal. It returns the ID of Table 101.
    - *Frontend Update*: Shows a notification that David has a pending invitation for "Website Redesign".
6. **David Accepts the Invitation**: David reviews the invitation and decides to join the project.
    - *Frontend Action*: Calls accept_join_table(101).
    - *Canister Response*: The canister removes the pending request, adds David's principal to Table 101's tableCollaborators list, and updates David's user record in the Auth canister via Auth.update_user_add_table to include Table 101 in his tablesJoined list.
    - *Frontend Update*: Displays "Website Redesign" now under David's "Tables I Joined".
7. **Sarah Confirms David as a Collaborator**: Sarah wants to verify who is on the team.
    - *Frontend Action*: Calls get_table_collaborators(101).
    - *Canister Response*: The canister retrieves Table 101's collaborator principals and fetches their full User details from the Auth canister.
    - *Frontend Update*: Displays both Sarah's and David's names as collaborators on the "Website Redesign" table.
8. **David Leaves the Project**: After his part is done, David leaves the project.
    - *Frontend Action*: Calls leave_table(101).
    - *Canister Response*: The canister removes David's principal from Table 101's tableCollaborators list and updates David's user record in the Auth canister via Auth.update_user_remove_table.
    - *Frontend Update*: "Website Redesign" is removed from David's "Tables I Joined" list.
9. **Sarah Deletes the Project Table**: The website redesign is complete, and the table is no longer needed.
    - *Frontend Action*: Calls delete_table(101).
    - *Canister Response*: The canister verifies Sarah is the creator. It then removes Table 101 from the system, removes it from Sarah's tablesCreated list in Auth, and also cleans up any

remaining related pending requests.

- *Frontend Update*: The "Website Redesign" table is no longer visible in Sarah's "Created Tables" list.