Session Management Functions Documentation

1. `create_session(principal: Text) : async Types.SessionResult`

Description

Creates a new session for a user identified by their **principal**. This function generates a unique session ID, sets creation and expiration times, and stores the session in memory.

Arguments

- `principal` (Text): The unique identifier (principal) of the user.

Return Value

- `Types.SessionResult`: An object containing:
    - `success` (Bool): Indicates whether the session was successfully created.
    - `message` (?Text): An optional message providing details about the operation.
    - `session` (?Session): The created session object if the operation was successful.

Associated Types

- `Types.Session`
- `Types.SessionResult`

2. `validate_session(sessionId: Text) : async Types.SessionResult`

Description

Validates a session using its **session ID**. The function checks if the session exists and if it has not expired. If a session is found to be expired, it is automatically deleted.

Arguments

- `sessionId` (Text): The unique session identifier.

Return Value

- `Types.SessionResult`: An object containing:
    - `success` (Bool): Indicates whether the session is currently valid.
    - `message` (?Text): An optional message providing details about the operation.

- `session` (?Session): The session object if it is valid.

Associated Types

- `Types.Session`
- `Types.SessionResult`

3. `logout(sessionId: Text) : async Types.SessionResult`

Description

Deletes (logs out) a session identified by its **session ID**.

Arguments

- `sessionId` (Text): The unique session identifier.

Return Value

- `Types.SessionResult`: An object containing:
  - `success` (Bool): Indicates whether the session was successfully deleted.
  - `message` (?Text): An optional message providing details about the operation.
  - `session` (?Session): This field is always `null` for a logout operation.

Associated Types

- `Types.SessionResult`

User Story: Frontend Usage

As a user of the Cafe application, I want to be able to log in and have my session managed securely, so that I can access protected features and my session can expire or be invalidated as needed.

Frontend Flow Example

1. **Login:**
   - The frontend collects the user's principal (e.g., after Internet Identity authentication).
   - It then calls the **`create_session`** function on the authentication canister.
   - On successful session creation, the frontend stores the returned **`sessionId`** (e.g., in `localStorage`).
2. **Authenticated Requests:**
   - For any protected API call, the frontend includes the **`sessionId`**.

- The backend validates this session using the **validate_session** function.
- If the session is valid, the request proceeds; otherwise, the user is prompted to log in again.

3. **Logout:**

- When the user logs out, the frontend calls the **logout** function with the current **sessionId**.
- The session is deleted from the canister, and the frontend removes the **sessionId** from its local storage.

This ensures:

- Only authenticated users can access protected resources.
- Sessions can expire or be invalidated for enhanced security.
- The frontend and backend remain synchronized regarding the user's authentication state.