

useListsData

Overview

The `useListsData` hook provides a comprehensive and centralized solution for managing hierarchical lists. Unlike individual hooks such as `useCreateList`, `useAddToList`, or `useFetchRootList`, this hook combines all list-related functionalities into a unified interface, withing the context of the logged in user. It enables creating, updating, deleting, and navigating through lists, as well as managing list entities. It is ideal for applications that need to handle complex list structures with parent-child relationships and caching mechanisms for efficient data access.

Usage Example

```
import { useListsData } from "@replyke/react-js";

function ListsComponent() {
  const {
    currentList,
    subLists,
    loading,
    openList,
    goBack,
    goToRoot,
    isEntityInList,
    createList,
    updateList,
    deleteList,
    addToList,
    removeFromList,
  } = useListsData();

  return (
    <div>
      <h1>Current List: {currentList?.name || "Root"}</h1>
      <ul>
        {subLists.map((list) => (
          <li key={list.id}>
            {list.name}
            <button onClick={() => openList(list)}>Open</button>
          </li>
        ))}
      </ul>
    </div>
  );
}
```

```

    ))}
  </ul>

  {loading && <p>Loading...</p>}}

  <button onClick={goBack}>Go Back</button>
  <button onClick={goToRoot}>Go to Root</button>

  <button
    onClick={async () => {
      await createList({ listName: "New List" });
    }}
  >
    Create List
  </button>

  <button
    onClick={async () => {
      if (currentList) {
        await deleteList({ list: currentList });
      }
    }}
  >
    Delete Current List
  </button>
</div>
);
}

```

Parameters & Returns

Parameters

This hook does not require any parameters.

Returns

The hook returns an object with the following fields:

Return Value	Type	Description
<code>currentList</code>	<code>List null</code>	The currently active list.
<code>subLists</code>	<code>List[]</code>	The sub-lists of the current list.

Return Value	Type	Description
<code>loading</code>	<code>boolean</code>	Indicates whether data is being fetched.
<code>openList</code>	<code>(list: List) => void</code>	Opens a specified list and sets it as the current list.
<code>goBack</code>	<code>() => void</code>	Navigates back to the previous list in the hierarchy.
<code>goToRoot</code>	<code>() => void</code>	Navigates directly to the root list.
<code>isEntityInList</code>	<code>(selectedEntityId: string) => boolean</code>	Checks if an entity exists in the current list.
<code>createList</code>	<code>(props: { listName: string }) => Promise<void></code>	Creates a new sub-list under the current list.
<code>updateList</code>	<code>(props: { listId: string; update: Partial<{ name: string }> }) => Promise<void></code>	Updates the name or other properties of a list.
<code>deleteList</code>	<code>(props: { list: List }) => Promise<void></code>	Deletes the specified list.
<code>addToList</code>	<code>(props: { entityId: string }) => Promise<void></code>	Adds an entity to the current list.
<code>removeFromList</code>	<code>(props: { entityId: string }) => Promise<void></code>	Removes an entity from the current list.

Key Features

- Hierarchical Navigation:** Easily navigate between parent and child lists with `openList`, `goBack`, and `goToRoot`.
- Entity Management:** Add or remove entities from lists using `addToList` and `removeFromList`.
- Cache Mechanism:** Caches sub-list data for each list to avoid redundant API calls and improve performance.
- CRUD Operations:** Perform create, update, and delete operations on lists.
- Root List Handling:** Automatically fetches the root list upon initialization.

Advanced Usage

Caching

The hook uses an internal cache (`subListCache`) to store sub-lists for each list by their ID. This prevents unnecessary API calls when navigating back to a previously accessed list.

State Management

- `loading` : Tracks whether any API operation is in progress.
 - `currentList` : Represents the currently selected list.
 - `subLists` : Contains the child lists of the current list.
 - `listHistory` : Maintains a stack of previously visited lists to enable the `goBack` functionality.
-

Common Errors

1. No Root List:

- Ensure the user is authenticated before invoking the hook.
- The root list is fetched automatically, but failures in this operation might result in `currentList` being `null`.

2. Cache Miss:

- If a list's sub-lists are not cached, the hook will fetch them from the server.
-

The `useListsData` hook is a comprehensive tool for managing lists and their entities, enabling developers to build robust hierarchical data management interfaces with minimal effort.

