ListsProvider and useLists Hook Documentation

Replyke's **ListsProvider** and **useLists** hook offer a simple yet powerful way to manage the context and functionality of lists within your application. These tools provide developers with seamless integration into their UI, ensuring users can create, manage, and interact with lists efficiently.

ListsProvider

The **ListsProvider** acts as the context provider for lists. It wraps any content where access to list management is required. Developers can use it flexibly, wrapping:

- A drawer or sheet where users manage their lists.
- An entire screen dedicated to list management.
- Multiple areas of the UI as needed—there is no limit to how many times it can be used.

useLists Hook

Any component within the **ListsProvider** can utilize the **useLists** hook to access the lists context, which includes data and functions to manage lists. Here is the interface for the context values:

```
interface ListsContextValues {
  currentList: List | null;
  subLists: List[];
  loading: boolean;

  openList: (list: List) => void;
  goBack: () => void;
  goToRoot: () => void;

  isEntityInList: (selectedEntityId: string) => boolean;

  createList: (props: { listName: string }) => Promise<void>;
  updateList: (props: {
    listId: string;
    update: Partial<{ name: string }>;
  }) => Promise<void>;
  deleteList: (props: { list: List }) => Promise<void>;
  addToList: (props: { entityId: string }) => Promise<void>;
  removeFromList: (props: { entityId: string }) => Promise<void>;
}
```

Context Values and Functions

Data

1. **currentList**: The currently active list in context. If a user opens a sublist, this value updates to the sublist. If they navigate back, it updates to the previous list. Initially set to `null`, it should quickly resolve to the user's default root list.
2. **subLists**: The sublists of the current list.
3. **loading**: A boolean flag indicating if sublists are being fetched. It is only `true` when navigating to a new list, as fetched data is cached.

Functions

1. **openList**: Sets a specified list as the current list and fetches its sublists. Typically used when a user selects a sublist in the UI.
2. **goBack**: Returns to the previous list. Can be used for any level above the root list.
3. **goToRoot**: Navigates directly back to the root list in a single action.
4. **isEntityInList**: Accepts an `entityId` and returns `true` if the entity is bookmarked in the current list, and `false` otherwise.
5. **createList**: Creates a new sublist under the current list. Expects an object with a `listName` string.
6. **updateList**: Updates the name of a list. Expects an object containing:
    - `listId`: The ID of the list to update.
    - `update`: An object with the new `name` for the list.
7. **deleteList**: Deletes a specified list. Expects an object with the full `list` object to be deleted.
8. **addToList**: Adds an entity to the current list. Expects an object containing `entityId`.
9. **removeFromList**: Removes an entity from the current list. Expects an object containing `entityId`.

Example Usage

```
import { ListsProvider, useLists } from '@replyke/react-js';

function ListsDrawer() {
  const {
    currentList,
```

```
    subLists,
    openList,
    goBack,
    createList,
    addToList,
  } = useLists();

  return (
    <div>
      <h2>{currentList?.name || 'Root List'}</h2>
      <ul>
        {subLists.map((subList) => (
          <li key={subList.id} onClick={() => openList(subList)}>
            {subList.name}
          </li>
        ))}
      </ul>
      <button onClick={goBack}>Go Back</button>
      <button
        onClick={() =>
          createList({ listName: 'New Sublist' }).then(() => alert('List Created'))
        }
      >
        Create Sublist
      </button>
    </div>
  );
}

function App() {
  return (
    <ListsProvider>
      <ListsDrawer />
    </ListsProvider>
  );
}
```

The **ListsProvider** and **useLists** hook make it easy to integrate and manage list functionality, providing developers with the tools needed to create a dynamic and interactive user experience.

Last updated on May 6, 2025