useEntityData

Overview

The `useEntityData` hook is a centralized utility for managing and interacting with a single entity in your application. Unlike the individual hooks such as `useFetchEntity` or `useUpdateEntity`, this hook integrates several functionalities to provide a streamlined approach for fetching, updating, deleting, and voting on a specific entity. It also includes logic to increment views and handle optimistic updates.

Usage Example

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

function EntityDetails({ entityId }: { entityId: string }) {
  const {
    entity,
    userUpvotedEntity,
    userDownvotedEntity,
    upvoteEntity,
    removeEntityUpvote,
    downvoteEntity,
    removeEntityDownvote,
    incrementEntityViews,
    deleteEntity,
  } = useEntityData({ entityId });

  useEffect(() => {
    incrementEntityViews();
  }, [incrementEntityViews]);

  if (!entity) return <p>Loading...</p>;

  return (
    <div>
      <h1>{entity.title}</h1>
      <p>{entity.content}</p>

      <button onClick={userUpvotedEntity ? removeEntityUpvote : upvoteEntity}>
        {userUpvotedEntity ? "Remove Upvote" : "Upvote"}
      </button>
```

```
      <button
        onClick={userDownvotedEntity ? removeEntityDownvote : downvoteEntity}
      >
        {userDownvotedEntity ? "Remove Downvote" : "Downvote"}
      </button>

      <button onClick={deleteEntity}>Delete Entity</button>
    </div>
  );
}
```

Parameters & Returns

Parameters

The hook accepts an object with the following fields. None are mandatory individually, but at least one of the ID properties, or a complete entity object, must be provided:

Parameter	Type	Required	Description
`entity`	`Entity`	No	If provided, skips fetching and uses this entity as the initial value.
`entityId`	`string`	No	The ID of the entity to fetch.
`foreignId`	`string`	No	A foreign ID to fetch the entity.
`shortId`	`string`	No	A short ID to fetch the entity.
`createIfNotFound`	`boolean`	No	If `true`, creates the entity if not found during fetching.

Returns

The hook returns an object with the following fields:

Return Value	Type	Description
`entity`	`Entity | null | undefined`	The fetched entity.

Return Value	Type	Description
`setEntity`	`React.Dispatch<React.SetStateAction<Entity | null | undefined>>`	The entity setter function.
`userUpvotedEntity`	`boolean`	Indicates if the current user has upvoted the entity.
`userDownvotedEntity`	`boolean`	Indicates if the current user has downvoted the entity.
`upvoteEntity`	`() => void`	Function to upvote the entity.
`removeEntityUpvote`	`() => void`	Function to remove the user's upvote.
`downvoteEntity`	`() => void`	Function to downvote the entity.
`removeEntityDownvote`	`() => void`	Function to remove the user's downvote.
`updateEntity`	`(props: Pick<UpdateEntityProps, "update">) => Promise<Entity | undefined>`	Function to update the entity.
`incrementEntityViews`	`() => Promise<void>`	Function to increment the entity's view count.
`deleteEntity`	`() => Promise<void>`	Function to delete the entity.

Features

Fetching Entity

The hook automatically fetches the entity using one of the provided identifiers (`entityId` , `foreignId` , or `shortId`). If an `entity` object is passed as a parameter, fetching is skipped.

Voting Functionality

The hook integrates voting logic, allowing users to upvote, remove upvotes, downvote, and remove downvotes on the entity. It tracks the user's current vote state (`userUpvotedEntity` , `userDownvotedEntity`).

Updating Entity

The `updateEntity` function simplifies updating the entity's data and optimistically updates the local state upon success.

Incrementing Views

The `incrementEntityViews` function ensures that views are incremented only once per session.

Deleting Entity

The `deleteEntity` function allows deleting the entity and updates the local state to remove it.

Best Practices

- **Use `incrementEntityViews` in `useEffect`**: Ensure that the view count increments only once when the entity details are rendered.
- **Optimistic Updates**: Leverage the optimistic updates provided by `updateEntity` to improve user experience.
- **Error Handling**: Handle errors gracefully using the provided `handleError` utility.

This comprehensive utility is ideal for managing individual entities in dynamic applications, providing a rich set of features out of the box.

Last updated on May 7, 2025