Entity Provider and useEntity Hook

Introduction

The Entity Provider is the easiest way to interact with an entity in Replyke. It ensures that the entity remains in its most up-to-date state and exposes data and functions for seamless interaction.

Using the EntityProvider Component

To use the Entity Provider, wrap the relevant content with the `EntityProvider` component. This enables access to the entity context for all child components.

The `EntityProvider` expects at least one of the following properties. None are mandatory individually, but at least one must be provided:

1. **entityId**: The ID of the entity in Replyke's system.
2. **foreignId**: The original item's ID if integrated with an external dataset, or a static value (e.g., "about-page").
3. **shortId**: The unique short ID of the entity.
4. **entity**: A complete `Entity` object. This is useful when rendering entities in a feed where the data is already available, preventing redundant re-fetches by passing the existing entity into the provider.

Optional Flag: createIfNotFound

The `createIfNotFound` flag instructs Replyke to create an entity if one isn't found. This flag is relevant for integration with existing datasets or static apps. When used, this flag will only cause the creation of an entity once per item or entity. Subsequent requests will locate the previously created entity.

Accessing the Entity Context

Child components can access the Entity Provider's context using the `useEntity` hook. Each `EntityProvider` instance provides context for a specific entity, ensuring that components using

the `useEntity` hook access only the relevant data and functions for that entity.

Note:

Avoid nesting multiple `EntityProvider` inside of each other. Doing so can result in unexpected behavior.

Data and Functions Exposed by the Entity Context

The `useEntity` hook exposes the following values:

```
interface EntityContextValues {
  entity: Entity | undefined;
  setEntity: React.Dispatch<React.SetStateAction<Entity | null | undefined>>;
  userUpvotedEntity: boolean;
  userDownvotedEntity: boolean;
  upvoteEntity: () => void;
  removeEntityUpvote: () => void;
  downvoteEntity: () => void;
  removeEntityDownvote: () => void;
  updateEntity(props: Pick<UpdateEntityProps, "update">): Promise<void>;
  incrementEntityViews: () => Promise<void>;
  deleteEntity: () => Promise<void>;
}
```

- **entity**: The entity data. It will be `undefined` initially until the entity is fetched, if the provider was given an ID (e.g., `entityId`) and not a full entity object.
- **setEntity**: The setter function for the entity object.
- **userUpvotedEntity**: A flag indicating whether the user has already upvoted the entity. Use this to render the UI appropriately and decide the voting action.
- **userDownvotedEntity**: A flag indicating whether the user has already downvoted the entity.
- **upvoteEntity**: A function to upvote the entity by the logged-in user.
- **removeEntityUpvote**: A function to remove an upvote from the entity by the logged-in user.
- **downvoteEntity**: A function to downvote the entity by the logged-in user.
- **removeEntityDownvote**: A function to remove a downvote from the entity by the logged-in user.
- **updateEntity**: A function to update specific properties of the entity.
- **incrementEntityViews**: A function to increment the entity's view count. It's up to the developer to define what constitutes a "view."

- **deleteEntity**: A function to delete the entity if the logged-in user is its creator.

> ***Note:*** *If a user has previously upvoted an entity and now wishes to downvote it, there is no need to first remove the upvote. Simply calling the* `downvoteEntity` *function will automatically add the user as a downvoter and remove them from the upvotes. The same applies for upvoting after downvoting.*

Functionality Protection

All functions are protected by Replyke's backend to ensure data integrity. For instance, if a user has already upvoted an entity, they cannot upvote it again. However, to maintain smooth UX and avoid backend errors, developers should implement UI logic to prevent invalid actions before they are attempted.

Last updated on May 7, 2025