useCommentSectionData

Overview

The `useCommentSectionData` hook provides a comprehensive solution for managing comment sections associated with entities. Unlike individual hooks such as `useFetchComments` or `useCreateComment`, this hook combines functionalities like fetching, creating, updating, deleting, and managing comment states (e.g., sorting, replying). It is ideal for implementing fully-featured comment sections with advanced state management.

Usage Example

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

function CommentSection({ entityId }: { entityId: string }) {
  const {
    comments,
    entityCommentsTree,
    loading,
    hasMore,
    sortBy,
    setSortBy,
    loadMore,
    createComment,
    updateComment,
    deleteComment,
    repliedToComment,
    setRepliedToComment,
    showReplyBanner,
    setShowReplyBanner,
  } = useCommentSectionData({ entityId, limit: 15 });

  const handleAddComment = async () => {
    try {
      await createComment({
        content: "This is a new comment!",
        mentions: [],
      });
    } catch (error) {
      console.error("Failed to add comment:", error.message);
    }
```

```
  };

  return (
    <div>
      <select
        value={sortBy}
        onChange={(e) => setSortBy(e.target.value as "new" | "top")}
      >
        <option value="new">Newest</option>
        <option value="top">Top</option>
      </select>

      <ul>
        {comments.map((comment) => (
          <li key={comment.id}>{comment.content}</li>
        ))}
      </ul>

      {loading && <p>Loading...</p>}
      {hasMore && !loading && (
        <button onClick={loadMore}>Load More</button>
      )}

      {showReplyBanner && <p>Replying to {repliedToComment?.id}</p>}
      <button onClick={handleAddComment}>Add Comment</button>
    </div>
  );
}
```

Parameters & Returns

Parameters

The hook accepts an object with the fields below. At least one of the first 4 fields must be provided (`entityId` / `foreignId` / `shortId` / `entity`):

Parameter	Type	Required	Description
`entityId`	`string | undefined | null`	No	The ID of the entity in Replyke's system.
`foreignId`	`string | undefined | null`	No	The original item's ID if integrated with an external dataset, or a static value (e.g., "about-page").

Parameter	Type	Required	Description
`shortId`	`string | undefined | null`	No	The unique short ID of the entity.
`entity`	`Entity | undefined | null`	No	A complete `Entity` object.
`createIfNotFound`	`string | undefined | null`	No	An optional flag which instructs Replyke to create an entity if one isn't found.
`limit`	`number`	No	The number of comments to fetch per page. Default is `15`.
`defaultSortBy`	`CommentsSortByOptions`	No	The default sorting criteria for comments (e.g., `new`, `top`).
`callbacks`	`SocialStyleCallbacks`	No	Optional callbacks for handling events like login requirements.
`highlightedCommentId`	`string | null`	No	The ID of a comment to highlight (e.g., for linking or deep linking).

Returns

The hook returns an object with the following fields:

Return Value	Type	Description
`entity`	`Entity | undefined | null`	The associated entity for the comment section.

Return Value	Type	Description
`callbacks`	`Record<string, (...args: any[]) => void> | undefined`	Optional callbacks passed into the hook.
`entityCommentsTree`	`EntityCommentsTree`	Tree structure representing threaded comments.
`comments`	`Comment[]`	The list of root-level comments currently loaded.
`newComments`	`Comment[]`	Comments that were recently added during the session.
`highlightedComment`	`{ comment: Comment; parentComment: Comment | null } | null`	A highlighted comment and its parent, useful for deep linking.
`loading`	`boolean`	Indicates if comments are currently loading.
`hasMore`	`boolean`	Whether there are more comments available for pagination.
`submittingComment`	`boolean`	Whether a comment is currently being submitted.
`loadMore`	`() => void`	Loads the next page of comments.
`sortBy`	`CommentsSortByOptions | null`	Current sorting method for the comments.
`setSortBy`	`(newSortBy: CommentsSortByOptions) => void`	Updates the comment sort order.
`pushMention`	`UserLean | null`	User that is being mentioned in a reply.
`selectedComment`	`Comment | null`	The comment currently selected

Return Value	Type	Description
		(e.g., for editing or context menu).
`setSelectedComment`	`(newSelectedComment: Comment | null) => void`	Sets the selected comment.
`repliedToComment`	`Partial<Comment> | null`	The comment being replied to.
`setRepliedToComment`	`(newRepliedToComment: Comment | null) => void`	Sets the comment being replied to.
`showReplyBanner`	`boolean`	Whether the UI banner indicating a reply is visible.
`setShowReplyBanner`	`(newState: boolean) => void`	Controls the reply banner visibility.
`addCommentsToTree`	`(newComments: Comment[] | undefined, newlyAdded?: boolean) => void`	Adds comments to the threaded comment tree.
`removeCommentFromTree`	`(commentId: string) => void`	Removes a comment from the tree structure.
`handleShallowReply`	`(comment: Comment) => void`	Handles a shallow reply (mention-only, not nested).
`handleDeepReply`	`(comment: Comment) => void`	Handles a deep reply (visibly nested under the parent comment).
`createComment`	`(props: { content?: string; gif?: GifData; mentions: Mention[] }) => Promise<void>`	Submits a new comment. Includes temporary rendering and error fallback.
`updateComment`	`(props: { commentId: string; content: string }) => Promise<void>`	Updates an existing comment.
`deleteComment`	`(props: { commentId: string }) => Promise<void>`	Deletes a comment from both the UI and backend.

© Replyke 2025