Comment Section: Implementation Guide

The Replyke comment section is designed to deliver a rich, social-media-style user experience with minimal setup. By using the components and hooks provided, you can quickly integrate a fully functional comment system into your application.

Key Components

To implement the comment section, four components come into play, three of which are required, and one optional:

1. **CommentSectionProvider** *(Required)*
   - Wraps the entire comment section and provides the necessary context to its child components.
   - This is the root of your comment section, ensuring all other elements can communicate seamlessly.
2. **CommentsFeed** *(Required)*
   - Displays the list of comments for the associated entity.
   - Supports nested replies and provides a social-style comment feed.
3. **NewCommentForm** *(Required)*
   - Provides the interface for users to submit new comments or replies.
4. **SortByButton** *(Optional)*
   - Allows users to sort comments by criteria such as “new,” “old,” or “top.”
   - Adds a layer of interactivity for users to customize their view of the comment feed.

Getting Started with `useSocialComments`

The `useSocialComments` hook provides these components and ensures they work together seamlessly. This hook requires two properties and allows four optional ones:

Required Properties

1. In order for the comment section to function properly, it need to be aware of the entity it related to. Therefore, the hook expects at least one of the following properties. None are mandatory individually, but at least one must be provided:

- `entityId`: The ID of the entity in Replyke's system.
- `foreignId`: The foreign/external ID of the entity. Only relevant when Replyke is integrated on top of an external dataset, or a static value (e.g., "about-page").
- `shortId`: The unique short ID of the entity.
- `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.

2. **styleConfig**: A configuration object created using the `useSocialStyle` hook. It allows for basic style customization of the comment section.

Optional Properties

1. **callbacks**: An object containing functions to handle specific interactions, such as what happens when a user needs to log in or clicks on an author or mention.
2. **defaultSortBy**: Sets the initial sort order of comments. Options are "top" (default), "new," or "old."
3. **limit**: Controls how many comments are fetched as the user scrolls. Defaults to 15. Adjusting this is possible but should balance efficiency and API costs—15 is typically ideal for most use cases.
4. **highlightedCommentId**: Highlights a specific comment. This is particularly useful when directing users to a comment via notifications. For instance:
    - If a user is notified that someone liked their comment, set `highlightedCommentId` to the ID of the liked comment when opening the comment section from their notifications.
    - If a user is notified about a reply to their comment, set `highlightedCommentId` to the ID of the **reply**, not the original comment. This distinction ensures that users are directed to the most relevant interaction in the thread.
5. **createIfNotFound**: An optional flag which instructs Replyke to create an entity if one isn't found.

Basic Implementation

Below is an example of how to set up the comment section in a React Native app (similar implementation should take place in a React JS app).

```
import React, { useEffect, useMemo, useRef } from "react";
import { View, Text, Keyboard } from "react-native";
import {
  useSocialComments,
  useSocialStyle,
  UseSocialStyleProps,
} from "@replyke/comments-social-react-native";
import { showMessage } from "react-native-flash-message";

const CommentsSection = ({
  entityId,
  navigateToAccount,
}: {
  entityId: string;
  navigateToAccount: (accountId: string) => void;
}) => {
  // Custom styles (optional)
  const customStyles = useMemo<Partial<UseSocialStyleProps>>(
    () => ({
      newCommentFormProps: {
        verticalPadding: 24,
        paddingLeft: 24,
        paddingRight: 24,
      },
    }),
    []
  );

  const styleConfig = useSocialStyle(customStyles);

  // Initialize comments with useSocialComments
  const { CommentSectionProvider, CommentsFeed, NewCommentForm, SortByButton } =
    useSocialComments({
      entityId,
      styleConfig,
      highlightedCommentId: "some-comment-uuid", // Should be an actual UUID if used
      callbacks: {
        currentUserClickCallback: () => {
          Keyboard.dismiss();
          navigateToProfile();
        },
        otherUserClickCallback: (userId: string) => {
          Keyboard.dismiss();
          navigateToAccount(userId);
        },
        loginRequiredCallback: () => {
          showMessage({
            message: "Oops! Login Required",
```

```
            description: "Please sign in or create an account to continue.",
            type: "warning",
          });
        },
      },
    });

  const commentFormRef = useRef<{ focus: () => void } | null>(null);

  // Focus the form when entityId changes
  useEffect(() => {
    if (!entityId) return;
    const timeout = setTimeout(() => {
      commentFormRef.current?.focus();
    }, 1000);
    return () => clearTimeout(timeout);
  }, [entityId]);

  return (
    <CommentSectionProvider>
      <View className="flex-1 h-full">
        <View className="flex-row gap-2 px-4 items-center mb-2">
          <View className="flex-1" />
          <SortByButton
            priority="top"
            activeView={
              <Text className="bg-black py-2 px-3 rounded-md text-white text-sm">
                Top
              </Text>
            }
            nonActiveView={
              <Text className="bg-gray-200 py-2 px-3 rounded-md text-sm">
                Top
              </Text>
            }
          />
          <SortByButton
            priority="new"
            activeView={
              <Text className="bg-black py-2 px-3 rounded-md text-white text-sm">
                New
              </Text>
            }
            nonActiveView={
              <Text className="bg-gray-200 py-2 px-3 rounded-md text-sm">
                New
              </Text>
            }
          />
        </View>
        <CommentsFeed />
        <NewCommentForm ref={commentFormRef} />
      </View>
```

```
    </CommentSectionProvider>
  );
};

export default CommentsSection;
```

Explanation of the Example

1. `useSocialStyle`: Customizes the style of the comment section, here adjusting paddings for the `NewCommentForm`.
2. `useSocialComments`: Provides the components required for the comment section.
3. `highlightedCommentId`: Highlights a specific comment, making it easier for users to locate important interactions.
    - If a notification relates to a liked comment, pass the liked comment's ID.
    - If it relates to a reply, pass the reply's ID instead to direct users to the reply in context.
4. `SortByButton`: Adds sorting options for comments (optional).
5. **Keyboard Interaction**: Demonstrates how to handle focus and navigation for comment interactions.

This example showcases the ease of integrating Replyke's comment system while still allowing you to make meaningful customizations.

In the next chapter, we'll dive into advanced configurations, callbacks, and styling options for creating a more tailored comment experience.

Last updated on May 7, 2025