Callbacks for the Comment Section

Replyke provides a robust callback system to give developers control over user interactions in the comment section. These callbacks allow you to define specific behaviors for various scenarios, enhancing the user experience and ensuring consistency with your app's requirements.

Below is the interface for available callbacks:

```
export type SocialStyleCallbacks = {
  loginRequiredCallback?: () => void;
  // Executed when an unauthenticated user tries to interact with the comments
  // (like/comment/reply). Defaults to a simple alert message.

  usernameRequiredCallback?: () => void;
  // Executed when a user with no username tries to interact with the comments
  // (like/comment/reply). Useful for enforcing usernames (e.g., directing users to
  // set up a username). If not provided, the action will be submitted, and a
  // generic username (e.g., user-85h344) will be used in the UI.

  commentTooShortCallback?: () => void;
  // Executed if the user attempts to submit an empty or too-short comment/reply.
  // Defaults to a simple alert message.

  currentUserClickCallback?: () => void;
  // Defines what happens when a user clicks their own avatar, name, or mention in the
  // comment section (e.g., redirecting them to their profile). Defaults to no action.

  otherUserClickCallback?: (userId: string) => void;
  // Defines what happens when a user clicks another user's avatar, name, or mention
  // in the comment section (e.g., navigating to that user's profile). Defaults to no act

  userCantBeMentionedCallback?: () => void;
  // What should happen when a user tries to mention another user but that other user
  // has no username set, which is a prequisite to being mentioned. Defaults to basic ale
};
```

Callback Descriptions

1. `loginRequiredCallback`

   - **Purpose**: Handles scenarios where an unauthenticated user attempts to interact with the comment section.

- **Default Behavior**: Shows a simple alert message.
- **Suggested Use**: Direct the user to a login or signup page to enable interaction.

2. `usernameRequiredCallback`

   - **Purpose**: Enforces username setup for users without a defined username.
   - **Default Behavior**: The interaction proceeds, and a generic username (e.g., `user-85h344`) is displayed.
   - **Suggested Use**: Prompt the user to set up a username, ideally during signup or through a dedicated flow.

3. `commentTooShortCallback`

   - **Purpose**: Prevents submission of empty or too-short comments/replies.
   - **Default Behavior**: Alerts the user with a simple message.
   - **Suggested Use**: Display a friendly message explaining why the comment cannot be submitted.

4. `currentUserClickCallback`

   - **Purpose**: Handles clicks on the current user's avatar, name or mention in the comment section.
   - **Default Behavior**: No action.
   - **Suggested Use**: Redirect the user to their profile or account settings page.

5. `otherUserClickCallback`

   - **Purpose**: Handles clicks on other users' avatars, names, or mentions.
   - **Default Behavior**: No action.
   - **Suggested Use**: Navigate to the clicked user's profile or show a modal with their information.

6. `userCantBeMentionedCallback`

   - **Purpose**: Handles cases where a user tries to mention another user who does not have a username set.
   - **Default Behavior**: Shows a basic alert message.
   - **Suggested Use**: You can notify the current user that mentioning of that user is not available. To avoid that, require all new users to set a unique username.

Example Usage

Here's an example of how you can define and pass these callbacks to the comment section:

```
import React from "react";
import { useSocialComments, useSocialStyle } from "@replyke/comments-social-react-native"
import { showMessage } from "react-native-flash-message";

const CommentSectionWithCallbacks = ({ entityId }: { entityId: string }) => {
  const styleConfig = useSocialStyle();

  const callbacks = {
    loginRequiredCallback: () => {
      showMessage({
        message: "Login Required",
        description: "Please log in to interact with the comment section.",
        type: "warning",
      });
    },
    usernameRequiredCallback: () => {
      showMessage({
        message: "Username Required",
        description: "Please set up a username to continue.",
        type: "warning",
      });
    },
    commentTooShortCallback: () => {
      showMessage({
        message: "Comment Too Short",
        description: "Your comment must contain at least one character.",
        type: "warning",
      });
    },
    currentUserClickCallback: () => {
      console.log("Navigating to current user's profile...");
    },
    otherUserClickCallback: (userId) => {
      console.log(`Navigating to profile of user with ID: ${userId}`);
    },
    userCantBeMentionedCallback: (user) => {
      showMessage({
        message: "User Can't Be Mentioned",
        description: `${user.name} cannot be mentioned because they don't have a username
        type: "warning",
      });
    },
  };

  const { CommentSectionProvider, CommentsFeed, NewCommentForm } = useSocialComments({
    entityId,
    styleConfig,
    callbacks,
  });
```

```
  return (
    <CommentSectionProvider>
      <CommentsFeed />
      <NewCommentForm />
    </CommentSectionProvider>
  );
};

export default CommentSectionWithCallbacks;
```

Conclusion

These callbacks give you granular control over the behavior of your comment section. By customizing them, you can create a more cohesive and user-friendly experience that aligns with your application's goals. For more advanced customization options, see the next section.

Last updated on May 6, 2025