Comment Section: Using the SocialCommentSection Component

The `SocialCommentSection` component offers the simplest way to integrate a fully functional, social-style comment section into your application. It encapsulates the core Replyke comment components and allows basic customization via props.

This component is ideal for developers who want to implement a complete comment section with minimal code while retaining flexibility.

Integration Example

If your app already wraps its content in a `ReplykeProvider`, you can use the component like so:

```
const DUMMY_POST = {
  id: "post_1234",
  title: "Replyke Demo",
  content: "Adding comment sections has never been so easy!",
};

<SocialCommentSection foreignId={DUMMY_POST.id} />
```

Here is a complete example:

```
import { useEffect, useState } from "react";
import { ReplykeProvider, useSignTestingJwt } from "@replyke/react-js";
import { SocialCommentSection } from "@replyke/comments-social-react-js";

const PROJECT_ID = import.meta.env.VITE_PUBLIC_REPLYKE_PROJECT_ID;
const PRIVATE_KEY = import.meta.env.VITE_PUBLIC_REPLYKE_SECRET_KEY;

const DUMMY_USER = { id: "user1", username: "lionel_messi10" };
const DUMMY_POST = {
  id: "post_1234",
  title: "Replyke Demo",
  content: "Adding comment sections has never been so easy!",
};
```

```
function App() {
  const signTestingJwt = useSignTestingJwt();
  const [signedToken, setSignedToken] = useState<string>();

  useEffect(() => {
    const handleSignJwt = async () => {
      const token = await signTestingJwt({
        projectId: PROJECT_ID,
        privateKey: PRIVATE_KEY,
        payload: DUMMY_USER,
      });
      setSignedToken(token);
    };

    handleSignJwt();
  }, []);

  return (
    <ReplykeProvider projectId={PROJECT_ID} signedToken={signedToken}>
      <SocialCommentSection foreignId={DUMMY_POST.id} />
    </ReplykeProvider>
  );
}
```

Component Props

`entity`, `entityId`, `foreignId`, `shortId`

Required

Used to identify the entity (e.g., a post or item) for which comments are rendered. None are mandatory individually, but at least one must be provided:

- `entity`: A complete `Entity` object.
- `entityId`: The internal Replyke ID for the entity.
- `foreignId`: An external ID if integrating with your own dataset.
- `shortId`: A short, human-readable identifier.

`callbacks`

Optional

Object of interaction handlers conforming to `SocialStyleCallbacks`. These may include login prompts, navigation to user profiles, mention behavior, and more.

See full reference in the Callbacks section.

`styleConfig`

Optional

A `PartialSocialStyleConfig` object used to control layout and visual styling. You can override only the parts you wish to change, offering flexibility while relying on defaults where needed.

See Styling for more details.

`isVisible`

Default: `true`

Indicates whether the comment section is currently visible. This is useful when rendering the section inside a drawer or another conditional layout, ensuring that certain listeners and behaviors are only active when visible.

`sortOptions`

Default: `["top", "new", "old"]`

An array of comment sorting modes to display. Use `null` to disable the sort controls entirely.

Each value must be one of:

- `"top"`
- `"new"`
- `"old"`

`header`

Optional

A React node rendered above the comment feed and to the left of the sort buttons.

`withEmojis`

Optional

When set to `true`, enables emoji picker functionality in the comment input.

Notes

- This component is internally powered by `useSocialComments` and renders:
  - `CommentSectionProvider`
  - `CommentsFeed`
  - `SortByButton` (if enabled)
  - `NewCommentForm` (if `isVisible`)
- If you need greater control over layout and behavior, you may use the Comment Section Hook directly.

When to Use

Use `SocialCommentSection` if you want to:

- Integrate comments quickly with minimal setup
- Maintain flexibility with styling and user interaction
- Rely on a standard layout that covers most use cases

For more flexibility in customizing the layout of the comment section beyond the options provided, consider using the hook and provider. For advanced use cases, such as building custom comment sections, consider the lower-level hooks & API.

Last updated on May 19, 2025