I added a Bluesky comment section to my blog

Published on **January 24, 2026**

You can now view replies to this blog post made on Bluesky directly on this website. Check it out here!

I've always wanted to host a comment section on my site, but it's difficult because the content is statically generated and hosted on a CDN. I could host comments on a separate VPS or cloud service. But maintaining a dynamic web service like this can be expensive and time-consuming – in general, I'm not interested in being an unpaid, part-time DevOps engineer.

Recently, however, I read a blog post by Cory Zue about how he embedded a comment section from Bluesky on his blog. I immediately understood to benefits of this approach. With this approach, Bluesky could handle all of the difficult work involved in managing a social media like account verification, hosting, storage, spam, and moderation. Meanwhile because Bluesky is an open platform with a public API, it's easy to directly embed comments on my own site.

There are other services that could be used for this purpose instead. Notably, I could embed replies from the social media formerly known as Twitter. Or I could use a platform like Disqus or even giscus, which hosts comments on GitHub Discussions. But I see Bluesky as a clearly superior choice among these options. For one, Bluesky is built on top of an open social media platform in AT Proto, meaning it can't easily be taken over by an authoritarian billionaire creep. Moreover, Bluesky is a full-fledged social media platform, which naturally makes it a better option for hosting a conversation than GitHub.

Zue published a standalone package called bluesky-comments that allows embedding comments in a React component as he did. But I decided to build this feature myself instead. Mainly this is because I wanted to make a few styling changes anyway to match the rest of my site. But I also wanted to leave the option open to adding more features in the future, which would be easier to do if I wrote

the code myself. The entire implementation is small regardless, amounting to only ~200 LOC between the UI components and API functions.

Initially, I planned to allow people to directly post on Bluesky via my site. This would work by providing an OAuth flow that gives my site permission to post on Bluesky on behalf of the user. I actually did get the auth flow working, but building out a UI for posting and replying to existing comments is difficult to do well. Going down this path quickly leads to building what is essentially a custom Bluesky client, which I didn't have the time or interest in doing right now. Moreover, because the user needs to go through the auth flow and sign-in to their Bluesky account, the process is not really much easier than posting directly on a linked Bluesky post.

Without the requirement of allowing others to directly post on my site, the implementation became much simpler. Essentially, my task was to specify a Bluesky post that corresponds to the article in the site's metadata. Then, when the page loads I fetch the replies to that post from Bluesky, parse the response, and display the results in a simple comment section UI.

As explained in my last post, this site is built using React Server Components and Parcel. The content of my articles are written using MDX, an extension to Markdown that allows directly embedding JavaScript and JSX. In each post, I export a `metadata` object that I validate using a Zod schema. For instance, the metadata for this post looks like this:

```
export const metadata = {
    title: "I added a Bluesky comment section to my blog",
    description:
        "How I embedded Bluesky replies directly on my site",
    date: "2026-01-24",
    bskyPostId: <post-id>,
    tags: ["web-dev"],
};
```

The value of `bskyPostId` references the Bluesky post from which I'll pull replies to display in the comment section. Because my project is built in TypeScript, it was easy to integrate with the Bluesky TypeScript SDK (`@bluesky/api` on NPM). Reading the Bluesky API documentation and Zue's implementation led me to the

`getPostThread` endpoint. Given an AT Protocol URI, this endpoint returns an object with data on the given post and its replies.

I could have interacted directly with the Bluesky API from my React component using `fetch` and `useEffect`. However, it can be a bit tricky to correctly handle loading and a error states, even for a simple feature like this. Because of this, I decided to use the Tanstack `react-query` package to manage the API request/response cycle. This library takes care of the messy work of handling errors, retries, and loading states while I simply provide it a function to fetch the post data.

Once I obtain the Bluesky response, the next task is parsing out the content and metadata for the replies. Bluesky supports a rich content structure in its posts for representing markup, references, and attachments. Building out a UI that fully respects this rich content would be difficult. Instead, I decided to keep it simple by just pulling out the text content from each reply.

Even so, building a UI that properly displays threaded comments, particularly one that is formatted well on small mobile devices, can be tricky. For now, my approach was to again keep it simple. I indented each reply and added a left border to make it easier to follow reply threads. Otherwise, I mostly copied design elements for layout of the profile picture and post date from Bluesky.

Lastly, I added a UI component linking to the parent post on Bluesky, and encouraging people to add to the conversation there. With this, the read-only comment section implementation was complete. If there's interest, I could publish my version of Bluesky comments as a standalone package. But several of the choices I made were relatively specific to my own site. Moreover, the implementation is simple enough that others could probably build their own version from reading the source code, just as I did using Zue's version.

Let me know what you think by replying on Bluesky. Hopefully this can help increase engagement with my blog posts, but then again, my last article generated no replies, so maybe not 😭.

Comments

Join the conversation by replying on Bluesky... →

Loading comments...