Problem Statement

Context: Developing a web application using Svelte 4 and leveraging current generation browser APIs

You are tasked with building a web application using Svelte 4 that displays content from an API in two separate feeds on the same page. Both feeds pull data from the same endpoint but use different filters. The application needs to handle user interactions efficiently and maintain consistency across multiple browser tabs and sessions, taking full advantage of modern browser capabilities.

Requirements

- Display posts from a single API endpoint in two distinct feeds on the same page.
- Implement a "favorite" feature that allows users to mark posts as favorites.
- Ensure that when a user favorites a post, the UI updates immediately in both feeds if the post is present in both.
- Propagate these changes across all open tabs of the application in real-time.
- Persist the feed data and user interactions (e.g., favorited posts) during browser restarts.
- Display the persisted data immediately upon application restart, before new API calls are completed. Describe your approach to structuring the data and implementing the required functionality in the application.

Focus on

- Efficient data structures for managing the two feeds
- Strategies for propagating user interactions across feeds and tabs
- Methods for persisting and retrieving data across browser instances
- Utilization of Svelte features and current generation browser APIs to achieve the desired functionality

Outline the key components of your solution, including any relevant design patterns, browser APIs, or libraries you would consider using to achieve a robust and scalable implementation. Be sure to highlight how you would leverage Svelte's reactivity system and any specific browser APIs that could enhance the application's performance and user experience.

Solution Design

The codebase uses a mock data array to represent all posts, with each individual post following this shape:

```
{
    // unique ID for post
    id: "aTJJoHsqVWlK",
    // post title
    title: "Accusantium sustineo unus cariosus speciosus alius.",
    // userId array to keep track of users who liked the post
    liked_users: [],
    // feed filter keywords
```

```
    feed: ["following", "global"],
}
```

Display posts from a single API endpoint in two distinct feeds on the same page

A mock service worker which intercepts API requests and serves mock responses was set up using the msw package. The handler for fetching the feeds:

```
followingPosts = allPosts.filter((post) => post.feed.includes("following"));
globalPosts = allPosts.filter((post) => post.feed.includes("global"));
if (feed === "user") {
    // return HttpResponse.error();
    return HttpResponse.json(followingPosts);
} else if (feed === "global") {
    return HttpResponse.json(globalPosts);
}
```

Using Svelte's onMount hook, which fires when the root component (App.svelte) is mounted, the feeds are fetched in parallel using Promise.all. Loading states of the two fetches are true on mount, and will be set to false once the response is available for render.

```
[followingPosts, globalPosts] = await Promise.all([
    fetch("http://localhost:5173/posts?feed=user&userId=1").then((res) =>
res.json()),
    fetch("http://localhost:5173/posts?feed=global").then((res) => res.json()),
]);
```

The posts data for a feed is rendered using Svelte's if-else blocks:

```
{#if isFollowingPostsPending}
    <div class="px-4 h-[calc(100vh-4rem)] grid place-items-center">Loading your
posts...</div>
{:else if isFollowingPostsError}
    <div class="px-4 h-[calc(100vh-4rem)] grid place-items-center">
    <div class="px-3 py-2 bg-red-200 border-1 border-red-700 rounded-md text-red-
700 m-2">
        Could not load your posts!
    </div>
    </div>
{:else}
    <Posts posts={followingPostsForMutation} feedtitle="Your Posts"
{handleToggleFav} />
{/if}
```

Implement a "favorite" feature that allows users to mark posts as favorites.

A handleToggleFav function which is defined in the parent component is passed as prop to Posts.svelte component. This function takes a postId as input to identify the post to be marked as favourite, and is attached to a 'fav.svg' icon in each individual post.

```
const handleToggleFav = (postId) => {
    followingPostsForMutation = updatePost(followingPostsForMutation, postId,
CURRENT_USER_ID);
    globalPostsForMutation = updatePost(globalPostsForMutation, postId,
CURRENT_USER_ID);
    broadcastChannel.postMessage({ type: "toggle-fav", postId: postId });
    fetch(`http://localhost:5173/post/${postId}`, {
        method: "POST",
        body: JSON.stringify({
        userId: CURRENT_USER_ID,
        }),
    })
        .then((res) => res.json())
        .then((data) => {
            window.localStorage.setItem(
                "posts",
                JSON.stringify([data.followingPosts, data.globalPosts]),
            );
        })
        .catch(() => {
            followingPostsForMutation = updatePost(followingPostsForMutation,
postId, CURRENT_USER_ID);
            globalPostsForMutation = updatePost(globalPostsForMutation, postId,
CURRENT_USER_ID);
        });
};
```

Ensure that when a user favorites a post, the UI updates immediately in both feeds if the post is present in both.

The updatePost function updates the post optimistically while a POST call is initiated to modify the data stored remotely. If the call fails, the posts are reverted to the older state, and the user will have to initiate the fav/unfav action again.

Propagate these changes across all open tabs of the application in real-time.

The root component (App.svelte) instantiates a broadcast channel, and registers an onmessage event listener on component mount.

```
const broadcastChannel = new BroadcastChannel("fav");
```

```
//inside onMount hook
broadcastChannel.onmessage = (event) => {
    if (event.data.type === "toggle-fav") {
        const postId = event.data.postId;
        followingPostsForMutation = updatePost(followingPostsForMutation, postId,
CURRENT_USER_ID);
        globalPostsForMutation = updatePost(globalPostsForMutation, postId,
CURRENT_USER_ID);
    }
};
```

PostID is emitted to the channel on user action, and updatePost function updates the posts feed on reception of the event.

Persist the feed data and user interactions (e.g., favorited posts) during browser restarts.

After the data for the feeds is fetched on initial mount, the data is stored in localStorage for persistence during browser restarts.

```
// after promise.all completes
window.localStorage.setItem("posts", JSON.stringify([followingPosts,
globalPosts]));
```

Display the persisted data immediately upon application restart, before new API calls are completed.

On component mount, before the API calls to fetch the feeds data are initiated, a check is done to see if we have persisted data in localStorage. If found, the feed data is filled using the data from localStorage, while parallel fetches are initiated for the latest feed data.

```
let persistedPosts = window.localStorage.getItem("posts");
if (persistedPosts) {
    [followingPostsForMutation, globalPostsForMutation] =
JSON.parse(persistedPosts);
    if (!(followingPostsForMutation instanceof Array) ||
!followingPostsForMutation) {
        isFollowingPostsError = true;
        window.localStorage.removeItem("posts");
    }
    if (!(globalPostsForMutation instanceof Array) || !globalPostsForMutation) {
        isGlobalPostsError = true;
        window.localStorage.removeItem("posts");
    }
    isFollowingPostsPending = false;
```

```
        isGlobalPostsPending = false;
    }
```

Codebase

https://github.com/borrowedlens/posts-feed