why did you do it like this?

Which part are you asking about? There are a few decisions in there that could each use explaining.

Q: Which decision are you questioning? (Select all that apply)
A: Why use Context instead of just props, Why put BagDrawer in App.jsx instead of Navbar

Why Context instead of props

The bag state needs to be shared between at least three components — `Navbar` (to open it and show the count), `BagDrawer` (to read and close it), and eventually your `Menu` page (to add items). If you used props, `App.jsx` would own the state and have to pass `openBag`, `closeBag`, `bagCount`, and `addItem` down to each of those separately. As you add more features that list grows, and any new component that needs bag access has to have props threaded to it. Context means any component just calls `useBag()` and gets what it needs directly, with no wiring in between.

Why BagDrawer in App.jsx instead of Navbar

This is the direct fix to your X button problem. CSS stacking contexts are scoped to the element that creates them — when a `sticky` element has a `z-index`, it creates its own stacking context, and nothing inside it can paint outside of it. Since `BagDrawer` was a child of `<nav>`, the drawer was trapped inside the navbar's stacking context no matter how high you set its z-index. Moving it to `App.jsx` puts it at the root level where it's in the top-level stacking context, so its `z-index` works against the whole page rather than just the navbar.

is there another fix to the x button that might better than putting bagdrawer in app.jsx?

Yes — a **React Portal**.

`ReactDOM.createPortal` lets you render a component outside of its parent's DOM hierarchy while keeping it in the same React component tree. You'd keep `<BagDrawer>` inside `Navbar.jsx` in your React code, but under the hood it renders directly onto `document.body`, completely outside the navbar's stacking context.

It would look like this in `BagDrawer.jsx`:

jsx

```jsx
import { createPortal } from 'react-dom';

function BagDrawer() {
  const { bagOpen, closeBag } = useBag();

  return createPortal(
    <>
      <div className={`bag-overlay ${bagOpen ? 'active' : ''}`} onClick={closeBag} /
      <div className={`bag-drawer ${bagOpen ? 'active' : ''}`} role="dialog" aria-mo
        ...
      </div>
    </>,
    document.body  // renders here, outside the navbar DOM entirely
  );
}
```

That's it — one import, wrap the JSX in `createPortal`. No changes to `Navbar.jsx` or `App.jsx`.

So which approach is better?

The Context + App.jsx approach I gave you is still the right call for your project. Here's why — portals solve the DOM stacking problem but you'd still eventually need to lift the bag state up anyway once your `Menu` page needs to call `addItem`. You'd end up doing both: a portal *and* Context. Since you need Context regardless, putting `BagDrawer` in `App.jsx` solves both problems in one move.

Portals are the better choice when you're adding a drawer to an existing app and want minimal disruption to the structure. For a fresh build like yours, the App.jsx approach is cleaner overall.