

Ha, classic! Glad it's working.

For step 3 we're hard-coding a single fake item in the drawer so we can style how real items will eventually look — name, quantity, price, and a remove button. Nothing functional yet.

Share your `BagDrawer.jsx` and `BagDrawer.css` as they currently stand in your codebase and I'll add the mock item into what you already have rather than starting fresh.

```
/* Overlay */
.bag-overlay {
  position: fixed;
  inset: 0;
  background: var(--overlay);
  opacity: 0;
  pointer-events: none;
  transition: opacity 0.3s ease;
```

Show more

Here's the updated drawer with a hard-coded mock item and the CSS to style it:

Ran 2 commands, read a file >

**What to check when you open the drawer:**

- Item name on the left, total price (`$11.50`) in gold on the right
- Quantity controls (`-` `2` `+`) in a pill on the left, trash icon on the right
- Nothing is wired up — buttons do nothing yet, that's intentional

The price shown multiplies `price × quantity` so you can test different values in `mockItem` to make sure the layout holds up with longer names or larger numbers. Once you're happy with how it looks we can move to step 4.



Bagdrawer  
Code · JSX

[Download](#)



Bagdrawer  
Code · CSS

[Download](#)

[Download all](#)

review your code for errors and bugs. What would you change?

Viewed 2 files >

Four things I'd change:

### 1. Bug — z-index mismatch between files

Your working `BagDrawer.css` in your codebase has `z-index: 1001` and `1002` (which you set to fix the navbar issue), but the file I just generated reverted to `200` and `201`. Make sure your actual file still has the higher values — if you copied my CSS output it would break the drawer again.

## 2. Bug — `pointer-events: all` is still in the file (line 14)

We caught this before but it crept back in. `all` is not a valid value — change it to `auto`:

css

```
.bag-overlay.active {  
  pointer-events: auto;  
}
```

## 3. Minor — floating point can produce bad prices

`mockItem.price * mockItem.quantity` uses floating point multiplication which is fine with `.toFixed(2)`, but once real prices come in from a menu, values like `2.30 * 3` can produce `6.8999999...`. The safer pattern is to store prices in cents as integers and only format at display time:

js

```
const mockItem = {  
  price: 575, // cents  
  quantity: 2,  
};  
  
// display as:  
${(mockItem.price / 100 * mockItem.quantity).toFixed(2)}
```

Worth deciding now before real data comes in, since changing it later means updating every place prices are stored and displayed.

#### 4. Minor — touch target sizes on `qty-btn` are too small

The `-` and `+` buttons have `padding: 0`, which makes them very small tap targets on mobile — likely around 18×18px. The recommended minimum is 44×44px. Easy fix:

css

```
.qty-btn {  
  padding: 8px;  
  min-width: 32px;  
  min-height: 32px;  
  justify-content: center;  
}
```

Items 1 and 2 are real bugs worth fixing now. Item 3 is a design decision worth making before step 6 when real prices come in. Item 4 is a polish fix you can do anytime. Want me to send updated files with all of these applied?

