Modal Dialog III Solution

React

Yangshun Tay
Ex-Meta Staff Engineer

Medium 20mins 262 done

Solution

We'll build on top of **Modal Dialog II's solution**. Since we're adding new interactions to the modal dialog component, most of the styling and structure can be reused without much modifications.

There are two new interactions to support for closing the dialog: (1) Hitting the `Escape` key, and (2) Clicking outside the dialog contents.

To close the dialog, we simply have to call the `onClose` callback, so it's a matter of when to trigger that callback.

1. Hitting the `Escape` key

We create a generic hook called `useOnKeyDown` that takes in a string key representing the keyboard key to respond to, and a callback to trigger when the key is pressed.

The hook adds an event listener for the `keydown` event to `document`. Within the event listener callback, check if `event.key` corresponds to the `key` argument, and trigger the callback argument if so.

To use the hook, call it with the following argument: `useOnKeyDown('Escape', onClose)`.

2. Clicking outside the dialog contents

The tricky part of implementing this logic is to determine whether the click happened inside or outside the dialog contents body. Thankfully, the **`Node.contains()` method** is a convenient method that can be used to determine if a node is a descendant of a given node, or the node itself. When a `mousedown` or `touchstart` event occurs, check if the `event.target` is a descendant of the modal dialog contents, e.g. `dialogEl.contains(event.target)`.

You may have tried using a `click` event as opposed to `mousedown`, but that does not work and the modal dialog cannot be opened. An explanation as to why can be found below.

It is necessary to obtain a reference to the modal dialog body. In React, this is achieved using the `useRef()` hook.

This functionality can be encapsulated as a `useOnClickOutside` hook that takes in the element ref and the callback function. "On click outside" is in fact a common utility hook and a more robust implementation of the hook can be found on `usehooks-ts` (despite its name, the default event is actually `mousedown`). To use the hook, call it with the following arguments: `useOnClickOutside(dialogRef, onClose)`.

Why using a `click` event does not work

Events bubble up the DOM and event handlers on DOM elements are fired during the bubbling phase of the event by default. When the button is clicked:

1. When users click on a button, a click event is created and travels down from the document root to the element (capturing phase) that the user clicked on, in this case, the button
2. When it reaches the button, the event bubbles back up to the document root
3. The button which has a `click` event handler, responds to the event by mounting the dialog
4. The dialog is rendered and adds `click` event handlers on the `document` to close the dialog when an element outside it is clicked
5. The `click` event travels up to the `document` and fires the event handler added by the dialog. The button is indeed outside the dialog, so the dialog is closed

The `mousedown` event works for clicking-outside interaction because `click` events are fired after `mousedown` and `mouseup` events have completed. The `mousedown` event caused by the click on the button would have completed even before the dialog is rendered.

Can we still use `click` events?

The alternative is to add a `click` event handler that only responds during the capturing phase:

```
document.addEventListener('click', onClickOutside, true); // Called during the capturing phase instead of bubblin
```

Although it works, the `mousedown` event is still preferred as it does not require the mouse button to be lifted before firing and provides a more snappy experience.

Test cases

1. Click outside to close:
   - Open the modal dialog
   - Click on the overlay area outside the modal content
   - Verify that the modal closes
2. Click inside does not close:
   - Open the modal dialog

    - Click on modal content
    - Verify that the modal does not close
3. Escape to close:
    - Open the modal dialog
    - Press the `Escape` key
    - Verify that the modal closes

Accessibility

We're not totally done with accessibility yet, there's still focus management to add, which will be covered in Modal Dialog IV.

Resources

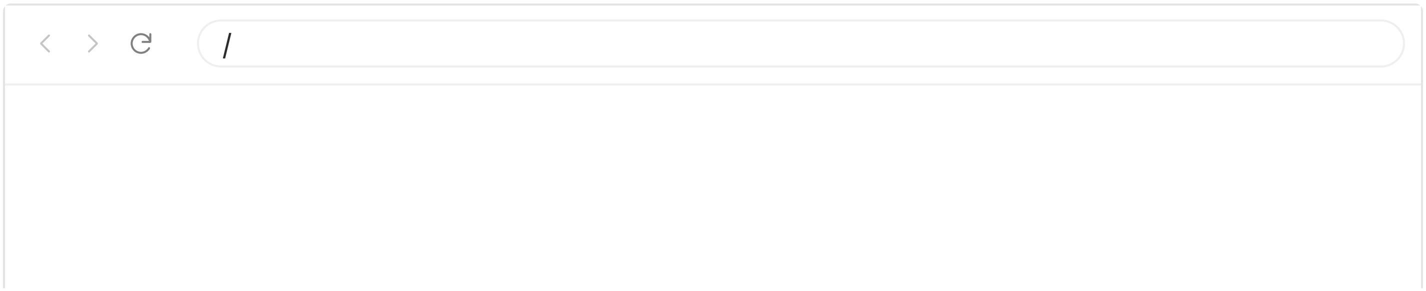
- Dialog (Modal) | ARIA Authoring Practices Guide
- Dialog – Radix Primitives
- Dialog | Reach UI
- Dialog - Headless UI

Follow up

A possible problematic scenario is having multiple dialogs open at once / stacked because the open dialog's contents allow opening of another dialog.

A limitation of our current implementation is surfaced – since most of the event listeners are added on the global level, all the open dialogs might respond to a `Escape` event and close at the same time, when the expected behavior is that only the topmost dialog closes.

Fixing this will require code to track the stack of open dialogs and only close the topmost dialog. However, implementing this is non-trivial and usually beyond the scope of interviews. That said, mentioning this scenario to the interviewer will probably give you brownie points!



Now in Beta
Build real world projects to learn and showcase
Premium
Nightmare
+5k rep
Chat AI
Develop a responsive web application to interact with Large Language Models (LLMs) like ChatGPT,
JAVASCRIPT
REACT
Go to project
62 completed
Hacker News Client
Sign in / Sign up

Sponsor: GreatFrontEnd Projects

The fastest way to learn Front End – by building actual projects

Want to grow in your front end skills? Learn them the right way — build apps step-by-step while referencing best practices from Senior engineers.

- 80+ real world projects
- Professional specs and designs by top-tier product managers and designers
- Guides and reference solutions from Senior Big Tech engineers