Using the Controlled Component

In this lesson, we'll learn how a user would use a controlled component from an app!

WE'LL COVER THE FOLLOWING

Sending a Prop

We reach out to the user and let them know that we have implemented a pattern to cater for their specific use case.

Out of excitement, they get on to implement the change.

Here's how.

Within the user app, they send a `data-index` prop to every `Header` element:

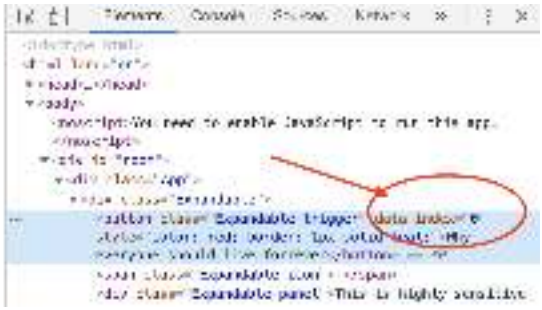
```
// before
<Expandable.Header />
// now
<Expandable.Header data-index={index} />
```

The `index` is retrieved from the iteration index of `information`.

```
function App () {
  ...
                // see index below 👇
 {information.map(({ header, note }, index) => (
   ...
 )}
}
```

How The `data-index` Property Helps

The returned element from the header gets a `data-index` property.



How does this help their cause?

Well, here's the full usage of the controlled component.

```
// user's app
function App () {
  const [activeIndex, setActiveIndex] = useState(null)
  const onExpand = evt => setActiveIndex(evt.target.dataset.index)

  return (
    <div className='App'>
      {information.map(({ header, note }, index) => (
        <Expandable
          shouldExpand={index === +activeIndex}
          onExpand={onExpand}
          key={index}
        >
          <Expandable.Header
            style={{ color: 'red', border: '1px solid teal' }}
            data-index={index}
          >
            {header}
          </Expandable.Header>
          <Expandable.Icon />
          <Expandable.Body>{note}</Expandable.Body>
        </Expandable>
      ))}
    </div>
  )
}
```

Note the `shouldExpand` controlled prop being the result of `index === +activeIndex`.

`activeIndex` is a state variable within the user app. Whenever any header is clicked, the `activeIndex` is set to the `data-index` of the element.

This way the user knows which header was just clicked and should be active at that given time.

This explains why they've defined the control prop `shouldExpand` as shown below:

```
shouldExpand={index === +activeIndex} // 👈 the "+" converts the activeIndex to a number
```

Setting the Active Index

Also, note the user's `onExpand` callback.

```
const onExpand = evt => setActiveIndex(evt.target.dataset.index)
```

This is what's invoked on every click, NOT our default `toggle` function. When the user clicks on a header, the `activeIndex` state variable is set to the `data-index` from the click target.

With this, the user implements their feature to their satisfaction. No two headers are expanded at the same time.

The Final Output

```
.Expandable-panel {
    margin: 0;
    padding: 1em 1.5em;
    border: 1px solid hsl(216, 94%, 94%);;
    min-height: 150px;
  }
```

Great job so far!

Quick Quiz

Let's take a quick quiz before we move on.

1 What does the `===` operator do in JavaScript?

With our controlled props API, we've made it relatively easy for the user to control the state of the `Expanded` component. How convenient! Let's build a custom hook next.