Providing Default UI Elements

We'll now make our open source library more user-friendly and convenient by allowing export of default UI elements,

WE'LL COVER THE FOLLOWING

Exporting Default Elements

If the user is reaching out for an `Expandable` OS library, like the one we made in the last lesson, they'll need the custom logic you've extracted into a reusable hook. They'll also need to render some UI elements.

You could also export some default `Header`, `Body` and `Icon` elements built to make their styling more configurable.

This sounds as if we can reuse the components we built earlier. However, we need to do some refactoring.

The child elements for the compound component required access to some context object. We may get rid of that and solely base the contract on props.

```
// Body.js (before)
const Body = ({ children, className = '', ...otherProps }) => {
  ...
  // look here 👇
  const { expanded } = useContext(ExpandableContext)
  return ...
}
// now (takes in expanded as props)
const Body = ({
        children, className = '', expanded, ...otherProps }) => {
        ...
  return ...
}
```

The same can be done for the `Icon` and `Header` components.

By providing these default UI elements, you allow the consumer of your library the option of not having to bother with UIs as much.

The Final Result

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

It's important to note that you don't always have to do this. You can always share custom functionality by just writing a custom hook.

Quick Quiz!

Test yourself on this lesson!

1 Why are we using our custom UI elements here?

We've done a brilliant job building this imaginary OS library, huh? In the following chapters, we'll incorporate even more advanced patterns that users will find incredibly useful.