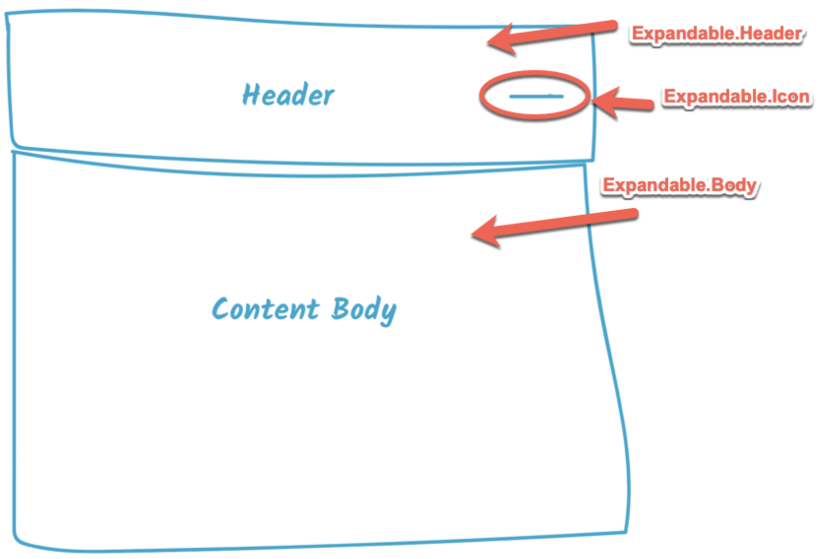
Building the Compound Child Components

Now, let’s work on the Expandable component's child components and actually see some output!

WE'LL COVER THE FOLLOWING

Child Components of `Expandable`

There are three child components for the `Expandable` component.



These child components need to consume values from the context object created in `Expandable.js`.

To make this possible, we’ll do a little refactoring as shown below:

```
import React, { createContext, useState, useCallback, useRef, useEffect, useMemo } from 'reac

export const ExpandableContext = createContext()
const { Provider } = ExpandableContext

const Expandable = ({ children, onExpand }) => {
  const [expanded, setExpanded] = useState(false)
  const toggle = useCallback(
    () => setExpanded(prevExpanded => !prevExpanded),
    []
  )
  const componentJustMounted = useRef(true)
  useEffect(
    () => {
    if (!componentJustMounted.current) {
        onExpand(expanded)
    }
     componentJustMounted.current = false
    },
    [expanded]
  )
  const value = useMemo(
   () => ({ expanded, toggle }),
   [expanded, toggle]
  )
  return (
    <Provider value={value}>
        {children}
    </Provider>
  )
}

export default Expandable
```

We export the context object, `ExpandableContext`, from `Expandable.js`.

Now, we may use the `useContext` hook to consume the values from the `Provider`.

The `Header` Child

Below is the `Header` child component fully implemented.

Header.js

Expandable.js

```
//Header.js
import React, { useContext } from 'react'
import { ExpandableContext } from './Expandable'

const Header = ({children}) => {
  const { toggle } = useContext(ExpandableContext)
  return <div onClick={toggle}>{children}</div>
}
export default Header
```

Simple, right?

It renders a `div` whose `onClick` callback is the `toggle` function for toggling the `expanded` state within the `Expandable` parent component.

The `Body` Child

Here’s the implementation for the `Body` child component:

Body.js

Header.js

Expandable.js

```
// Body.js
import { useContext } from 'react'
import { ExpandableContext } from './Expandable'

const Body = ({ children }) => {
  const { expanded } = useContext(ExpandableContext)
  return expanded ? children : null
}
export default Body
```

Pretty simple as well.

The `expanded` value is retrieved from the context object and used within the rendered markup. **Line** 7 reads like this: expanded, render `children`, otherwise, render nothing.

The `Icon` Child

The `Icon` component is just as simple.

Icon.js

Body.js

Header.js

Expandable.js

```
// Icon.js
import { useContext } from 'react'
import { ExpandableContext } from './Expandable'

const Icon = () => {
  const { expanded } = useContext(ExpandableContext)
  return expanded ? '-' : '+'
}
export default Icon
```

It renders either `+` or `-` depending on the value of `expanded` retrieved from the context object.

With all child components built, we can set them as `Expandable` properties. See below:

```
import Header from './Header'
import Icon from './Icon'
import Body from './Body'
...

const Expandable = ({ children, onExpand }) => {
        ...
}
// Remember this is just a personal reference. It's not mandatory
Expandable.Header = Header
Expandable.Body = Body
Expandable.Icon = Icon
```

Using the `Expandable` Component

Now, we can go ahead and use the `Expandable` component as designed:

```
<Expandable>
    <Expandable.Header>React hooks</Expandable.Header>
    <Expandable.Icon />
    <Expandable.Body>Hooks are awesome</Expandable.Body>
</Expandable>
```

Current Look

Here is the `Expandable` component so far!

```
// Body.js
import { useContext } from 'react'
import { ExpandableContext } from './Expandable'

const Body = ({ children }) => {
  const { expanded } = useContext(ExpandableContext)
  return expanded ? children : null
}
export default Body
```

Quick Quiz!

1

Select all that apply for this question.

Why do we export the `ExpandableContext` from `Expandable.js`?

This works but it has to be the ugliest component I’ve ever seen. We can do better. Let’s try in the next lesson!