Write Better React JS

Ternary Conditional Rendering

If you are conditionally rendering two different components then use the ternary operator over &&.

✗

```
const Wrapper = ({ showLogin }) =>
  <>
    {showLogin && <Login />}
    {!showLogin && <Signup />}
  </>
```

✓

```
const Wrapper = ({ showLogin }) =>
  showLogin ? <Login /> : <Signup />
```

I'm a bit undecided one this one as both look pretty clean to me 👌

Pascal Component Names

Using Pascal case for any React component is the industry recommended convention.

✗

```
import coolButton from "components"

const wrapper = () =>
  <div>
    <coolButton />
  </div>
```

✓

```
import CoolButton from "components"

const Wrapper = () =>
  <div>
    <CoolButton />
  </div>
```

It helps us easily differentiate between normal HTML and React components

Avoid Inline Event Handlers

Inline events can get real messy once they are doing multiple things. Creating a series of handle functions is cleaner.

```
const Textbox = () =>
  <input onChange={e =>
      setState(e.target.value)
      logChange(e.target.value)
    }
  />
```

```
const Textbox = () => {
  const handleChange = e => {
    setState(e.target.value)
    logChange(e.target.value)
  }
  return <input onChange={e => handleChange(e)} />
}
```

Avoid Complex Ternary

Ternary operators with many conditions can get real complex, so I like to always store them in their own separate variable.

```
const Wrapper = ({ items, purchases }) =>
  items?.length > 10 &&
  purchases?.length > 100
    ? <VIPOffer />
    : <Offer />
```

```
const Wrapper = ({ items, purchases }) => {
  const isVIP =
    items?.length > 10 && purchases?.length > 100

  return isVIP ? <VIPOffer /> : <Offer />
}
```

Redundant Event Passing

If you are only passing the HTML event of an element to a function there is no need to implicitly pass the event.

```
const Textbox = () => {
  const handleChange = e =>
    setState(e.target.value)

  return <input onChange={e => handleChange(e)} />
}
```

```
const Textbox = () => {
  const handleChange = e =>
    setState(e.target.value)

  return <input onChange={handleChange} />
}
```

Redundant Boolean Props

If you are setting the property value to “true” it is not required to pass a value of “true”.

```
import CoolButton from "components"

const PrimaryButton = () =>
  <CoolButton isPrimary={true} />
```

```
import CoolButton from "components"

const PrimaryButton = () =>
  <CoolButton isPrimary />
```

&& Conditional Rendering

Don’t use the ternary operator unnecessarily you can simply use && and React will handle the rest for you.

❌

```
const Wrapper = ({ showButton }) =>
  <>
    <CoolTextbox />
    {showButton ? <CoolButton /> : null}
  </>
```

✓

```
const Wrapper = ({ showButton }) =>
  <>
    <CoolTextbox />
    {showButton && <CoolButton />}
  </>
```

Simple String Props

If you are setting a prop to a simple string there is no need to use curly brackets, they are only required if passing backtick values.

✗

```
import CoolButton from "components"

const PrimaryButton = () =>
  <CoolButton isPrimary tooltip={"Primary"} />
```

✓

```
import CoolButton from "components"

const PrimaryButton = () =>
  <CoolButton isPrimary tooltip="Primary" />
```

FOLLOW