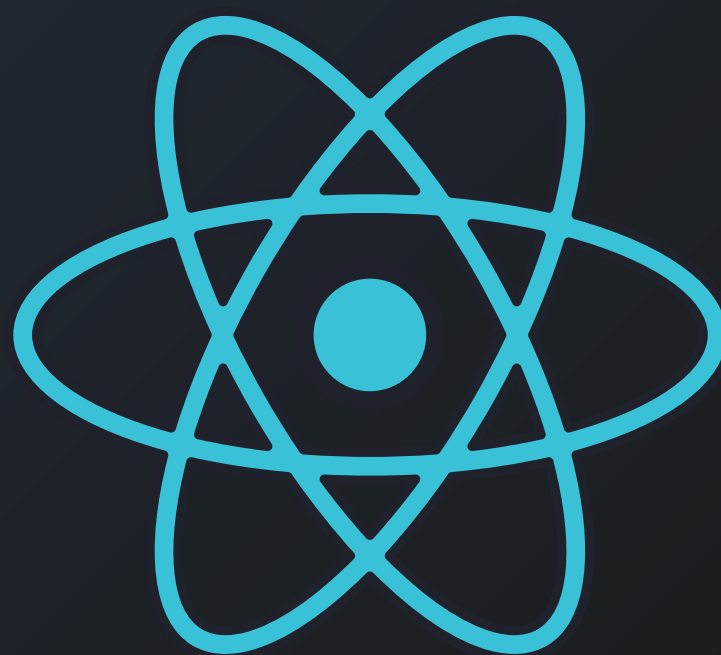




Boas práticas para React JS

Boas práticas de React JS que você
deveria saber



@patrycksilva



Propriedades booleanas redundantes

Se você está definindo o valor da propriedade como "true", não é necessário passar um valor de "true".



```
export const App = () => {  
  return (  
    <div>  
      <Button isPrimary={true} />  
    </div>  
  )  
}
```



```
export const App = () => {  
  return (  
    <div>  
      <Button isPrimary />  
    </div>  
  )  
}
```



Propriedades de String Simples

Se você está definindo uma propriedade como uma string simples, não há necessidade de usar chaves, elas só são necessárias se estiver passando valores em template string.



```
export const App = () => {  
  return (  
    <div>  
      <Button tooltip={'primary'} />  
    </div>  
  )  
}
```



```
export const App = () => {  
  return (  
    <div>  
      <Button tooltip="primary" />  
    </div>  
  )  
}
```



Nome de componentes em Pascal case

Usar Pascal case para qualquer componente React é a convenção recomendada pela indústria.

```
export const primaryButton = () => {  
  return (  
    <button>  
      <span>Click me</span>  
    </button>  
  );  
};
```



```
export const PrimaryButton = () => {  
  return (  
    <button>  
      <span>Click me</span>  
    </button>  
  );  
};
```



Isso nos ajuda a diferenciar facilmente entre HTML normal e componentes React.



Renderização Condicional com &&

Não use o operador ternário desnecessariamente, você pode simplesmente usar && e o React cuidará do resto para você.

```
export const App = ({ showButton }) => {  
  return (  
    <div>  
      <span>Component</span>  
      {showButton ? <Button /> : null}  
    </div>  
  )  
}
```



```
export const App = ({ showButton }) => {  
  return (  
    <div>  
      <span>Component</span>  
      {showButton && <Button />}  
    </div>  
  )  
}
```





Evite Ternários Complexos

Ternários com muitas condições podem ficar realmente complexos, então eu prefiro sempre armazená-los em sua própria variável separada.

```
const Component = ({ isLoggedIn, purchases }) => {  
  return (  
    <div>  
      {isLoggedIn && purchases.length ≥ 100  
        ? <VipProducts />  
        : <Products />}  
    </div>  
  )  
}
```




```
const Component = ({ isLoggedIn, purchases }) => {  
  const minPurchasesToVip = 100  
  
  const isVip = isLoggedIn && purchases.length ≥ minPurchasesToVip  
  
  return (  
    <div>  
      {isVip  
        ? <VipProducts />  
        : <Products />}  
    </div>  
  )  
}
```






Passagem Redundante de Eventos

Se você está apenas passando o evento HTML de um elemento para uma função, não há necessidade de passar implicitamente o evento.



```
const Input = () => {
  const handleChange = e => {
    setName(e.target.value)
  }
  return (
    <input type="text" onChange={e => handleChange(e)} />
  )
}
```



```
const Input = () => {
  const handleChange = e => {
    setName(e.target.value)
  }
  return (
    <input type="text" onChange={handleChange} />
  )
}
```

Gostou do post?
Deixe uma reação!



@patrycksilva