Save for later

Quick Start Guide to React in 8 Slides

@nacercodes

JSX

```
<>
  <HeroSection />

  <h1>What is React?</h1>
  <p>A JS framework, I mean, library.</p>
</>
```

This weird syntax is called JSX. You will be writing this all the time to describe what the UI should look like.

So... It's like... HTML?

It's stricter than HTML, Jeremy

Components

A React app is made out of components, much like Building Blocks. Below is our `HeroSection` component that we nested in the previous slide.

HeroSection.jsx

```jsx
function HeroSection() {
  return (
    <div className='hero'>
      <h1>React in 8 Slides</h1>
      <p style={{ maxWidth: '512px' }}>
        You won't find this elsewhere!
      </p>
    </div>
  )
}
```

Styling

We styled the `HeroSection` parent element by specifying a CSS class with the `className` attribute.

```
<div className='hero'>...</div>
```

HeroSection.css

```
.hero {
  /* Just regular CSS */
}
```

We can also use inline styles by specifying the `style` attribute with a JS object with camelCased properties.

```
<p style={{ maxWidth: '512px' }}>...</p>
```

Conditional Rendering

Sometimes we want to render a component based on a condition. We are still using JavaScript, right? So...

```
<header>
  {isLoggedIn ? (
    <SignOutButton />
  ) : (
    <SignInButton />
  )}
</header>
```

No need for `else`? We can use the logical `&&` operator.

```
{isLoggedIn && <SignOutButton />}
```

Rendering a List

```
const fruits = [
  { id: 1, name: 'Apple' },
  { id: 2, name: 'Lemon' }
]
```

```
<ul>
  {fruits.map(({ id, name }) => (
    <li key={id}>{name}</li>
  ))}
</ul>
```

For each item in a list, we should pass a unique key or React will complain.

Events Handling

We need our app to respond to events such as clicks, submissions, changes, and more.

```
function SignOutButton() {
  function handleClick() {
    // ... sign out logic ...
  }

  return (
    <button onClick={handleClick}>
      Sign out
    </button>
  )
}
```

Hooks

Functions starting with `use` are called Hooks, and React comes with some built-in ones.

State Hooks

- `useState`
- `useReducer`

Context Hooks

- `useContext`

Resource Hooks

- `use`

Effect Hooks

- `useEffect`
- `useLayoutEffect`
- `useInsertionEffect`

Performance Hooks

- `useMemo`
- `useCallback`
- `useTransition`
- `useDeferredValue`

Ref Hooks

- `useRef`
- `useImperativeHandle`

Other Hooks

- `useDebugValue`
- `useId`
- `useSyncExternalStore`

We can also define our own hooks if we want to 🙊...

Sharing Data

Both `Button` components need to read and update the `count` value, so we need a shared state between the two buttons and pass things via props.

```
function DualCounter() {
  const [count, setCount] = useState(0)

  function increment() {
    setCount((prev) => prev + 1)
  }

  return (
    <div>
      <Button text={count} onClick={increment} />
      <Button text={count} onClick={increment} />
    </div>
  )
}
```

Nacer Codes

@nacercodes

Save it or lose it. 🙂