How Can You Debug Your React Components Using the React DevTools?

freecodecamp.org/learn/front-end-development-libraries-v9/lecture-react-strategies-and-debugging/how-can-you-debug-your-react-components-using-react-devtools

The browser has built-in developer tools you can use to debug HTML, CSS, and JavaScript.

However, they're not great for finding and fixing bugs in React apps. So the React team developed a tool called "React Developer Tools" (AKA React DevTools) so you can inspect, debug, and profile React apps.

React DevTools is available as a browser extension for Chrome, Edge, and Firefox. If you're on Chrome or Edge, head over to the Chrome web store, search for "React Developer Tools", and add it to your browser.

And if you use Firefox, head over to the Firefox Add-ons page, search for the tool, and add it to your browser.

If you use Safari, you can install React DevTools from npm by running `npm install -g react-devtools` or `yarn global add react-devtools`.

After installing and enabling React DevTools, if you open a React app in your browser, then open your browser's developer tools, you should see two extra tabs: Components and Profiler.

The Components tab displays each component for you in a tree view format. With it, you can:

- View the app's component hierarchy.

- Check and modify props, states, and context values in real time.
- Check the source code for each selected component.
- Log the component data to the console.
- Inspect the DOM elements for the component.

On the other hand, the Profiler tab helps you record and analyze component performance so you can identify unnecessary re-renders, view commit durations, and things you can optimize.

Here's a simple app to show you how you can inspect components and any props and state they have. This is similar to the code we used in a previous lesson on prop drilling:

```
import { useState } from "react";

export default function App() {
  const greeting = "Hello, Prop Drilling!";
  const response = "I'm not here to play!";

  return <Parent greeting={greeting} response={response} />;
}

const Parent = ({ greeting, response }) => {
  return <Child greeting={greeting} response={response} />;
};

const Child = ({ greeting, response }) => {
  return <Grandchild greeting={greeting} response={response} />;
};

const Grandchild = ({ greeting, response }) => {
  const [count, setCount] = useState(0);

  return (
    <>
      <h1>{greeting}</h1>
      <h2>{response}</h2>

      <button onClick={() => setCount(count + 1)}>Increase Count</button>
      <h2>Count: {count}</h2>
      <button onClick={() => setCount(count - 1)}>Decrease Count</button>
    </>
  );
};
```

If you look in the Components tab in React DevTools, you can see the tree view of the components. The `App` component is at the top, followed by the `Parent`, `Child`, and `Grandchild` components.

If you select any of these components, you can see the props and state in them. If you select the `Parent` component, you can see the `greeting` and `response` props, which are `Hello, Prop Drilling!` and `I'm not here to play!`, respectively.

You can see the props and update state in real-time, and change them if necessary. For instance, you can select the `Grandchild` component and change the `greeting` prop from `Hello, Prop Drilling!` to `Hello, Welcome to Prop Drilling!`, and see it reflected on the page immediately.

To log data in a component to the console, inspect the matching DOM elements and view the source code of the component. The icons in the top right corner let you do that. If you select the `Grandchild` component and click the `Log the component data to the console` button, it will log the props, state, hooks, nodes, and other data in the console.

A common bug you might encounter in React is called props mismatch.

For example, say that for the `Child` component, you mistakenly pass in `reply` as the prop instead of `response`:

```
const Child = ({ greeting, response }) => {
  return <Grandchild greeting={greeting} reply={response} />;
};
```

Remember that `Grandchild` expects a `response` prop. Because the component receives a different prop, it can't display that text on the page, and just adds and empty `h2` to the DOM. Instead, you'll just see the `h1` element with the text `Hello, Prop Drilling!`, along with the other buttons and text already on the page. The empty `h2` element is still there, but because it's empty, you can't see it without inspecting the DOM.

To fix this, you can inspect the prop progression from the `Parent` component down to the `Child` and edit the prop name directly. If you go to the Components tab, select the `Child` component, and change the `reply` prop to `response`, you'll see the `h2` element on the page with the text `I'm not here to play!`.

Questions

How can you install React DevTools on Chrome or Edge?

Download it from the official React website.

Install it via `npm` in your project.

Add it as a browser extension.

Correct!

It comes pre-installed with React.

After installing React DevTools, which two extra tabs appear in the browser developer tools for debugging React?

Components and Profiler.

Correct!

Network and Performance.

Elements and Console.

Sources and Memory.

What does the Profiler tab in React DevTools help you analyze?

Network requests in a React app.

JavaScript memory leaks.

CSS styling issues.

Component performance and re-renders.

Correct!