**FULLSTACK DEVELOPMENT**

**Week-2 Tutorial – 2**

**React useState & Props: Interactive Activity**

This activity will guide you through understanding props using both class and functional components, explore the differences between them, and then dive into managing component state with useState. We'll also look at the powerful React Dev Tools.

**Prerequisites:**

Node.js and npm (or yarn) installed.

VS Code installed.

Basic understanding of creating React components.

**Part 1: Setting Up Your React Project (5 minutes)**

If you already have a React project set up from the previous class, you can skip to Part 2.

**Open your terminal or command prompt.**

**Navigate to your desired directory** where you want to create your project.

cd path/to/your/projects

**Create a new React project** using Create React App.

npx create-react-app my-react-app

(Replace my-react-app with your preferred project name.)

**Navigate into your new project directory:**

cd my-react-app

**Start the development server:**

npm start

This will open your React app in your browser (usually http://localhost:3000). Keep this running!

**Open the project in VS Code:**

code .

**Part 2: Understanding and Using Props with Class Components (10 minutes)**

Props (short for properties) are how you pass data from a parent component to a child component. We'll start by seeing how this works with **class components**.

**Create a new file** named GreetingCardClass.js inside the src folder.

**Add the following code to GreetingCardClass.js:**

// src/GreetingCardClass.js

import React from 'react';

// This is a class component that receives 'props'.

class GreetingCardClass extends React.Component {

render() {

// In a class component, you access props using 'this.props'.

// For example, if you pass <GreetingCardClass name="David" />,

// you would access it as this.props.name.

return (

<div>

<h2>Hello from Class Component, {this.props.name}!</h2>

<p>{this.props.message}</p>

</div>

);

}

}

export default GreetingCardClass;

**Open src/App.js** (your main application component). We will transform App.js into a class component that uses GreetingCardClass.

**Replace the existing code in src/App.js with this:**

// src/App.js

import React from 'react';

import GreetingCardClass from './GreetingCardClass'; // Import the class component

// This is our main application component, now a class component.

class App extends React.Component {

// You can define variables directly within the class or in state (covered later).

// For now, we'll just use them directly in render.

render() {

// Variables to pass as props

const user1Name = "David";

const user1Message = "Learning with class components!";

const user2Name = "Eve";

const user2Message = "Props work here too!";

return (

<div>

<h1>My React App with Class Components & Props</h1>

{/\* Using GreetingCardClass and passing variables as props \*/}

<GreetingCardClass name={user1Name} message={user1Message} />

<GreetingCardClass name={user2Name} message={user2Message} />

</div>

);

}

}

export default App;

**Observe your browser:** You should now see two greeting cards, each displaying different data passed from the App class component as props. Notice how this.props.name and this.props.message are used in GreetingCardClass.

**Part 3: Introducing Functional Components and Props (5 minutes)**

Functional components are a simpler way to write components in React, especially with the introduction of Hooks.

**Create a new file** named GreetingCardFunctional.js inside the src folder.

**Add the following code to GreetingCardFunctional.js:**

// src/GreetingCardFunctional.js

import React from 'react';

// This is a functional component. It receives 'props' directly as an argument.

const GreetingCardFunctional = (props) => {

// Or, you can destructure props directly in the function signature:

// const GreetingCardFunctional = ({ name, message }) => {

return (

<div>

{/\* We access the 'name' property directly from the props argument \*/}

<h2>Hello from Functional Component, {props.name}!</h2>

{/\* We access the 'message' property directly from the props argument \*/}

<p>{props.message}</p>

</div>

);

};

export default GreetingCardFunctional;

**Temporarily modify src/App.js** to use GreetingCardFunctional instead of GreetingCardClass to see it in action.

// src/App.js (Temporary modification to see functional component)

import React from 'react';

import GreetingCardFunctional from './GreetingCardFunctional'; // Import the functional component

class App extends React.Component {

render() {

const user3Name = "Frank";

const user3Message = "Functional components are neat!";

return (

<div>

<h1>My React App with Class Components & Functional Props</h1>

{/\* Using GreetingCardFunctional and passing variables as props \*/}

<GreetingCardFunctional name={user3Name} message={user3Message} />

</div>

);

}

}

export default App;

**Observe your browser:** You should now see the greeting from the functional component. Notice the simpler syntax for defining the component and accessing props.

**Part 4: Class Components vs. Functional Components (2 minutes)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Class Components** | **Functional Components (before Hooks)** | **Functional Components (with Hooks)** |
| **Declaration** | class MyComponent extends React.Component | const MyComponent = (props) => { ... } | const MyComponent = (props) => { ... } |
| **Props Access** | this.props | props (as argument) | props (as argument) |
| **State Management** | this.state, this.setState() | No built-in state (were "stateless") | useState Hook |
| **Lifecycle** | Lifecycle methods (componentDidMount, etc.) | No lifecycle methods | useEffect Hook (for side effects/lifecycle) |
| **Syntax** | More verbose, uses this | Simpler, more concise | Concise, but requires understanding Hooks |
| **Modern Usage** | Less common for new components, still supported | Primarily for simple, presentational components | **Recommended for new components** |

**Part 5: Managing Component State with useState (5 minutes)**

useState is a React Hook that lets you add state to **functional components**. State is data that can change over time and cause the component to re-render.

**Revert src/App.js to a functional component** and incorporate the useState Hook for a counter. This will be the main App.js for the rest of the activity.

// src/App.js (Main App - now functional with useState)

import React, { useState } from 'react'; // Import useState

import GreetingCardFunctional from './GreetingCardFunctional'; // Use the functional GreetingCard

function App() {

// Declare a state variable 'count' and a function 'setCount' to update it.

// Initial value of count is 0.

const [count, setCount] = useState(0);

// Function to increment the count

const incrementCount = () => {

setCount(count + 1); // Update the state

};

// Function to decrement the count

const decrementCount = () => {

setCount(count - 1); // Update the state

};

return (

<div>

<h1>My React App with Functional Components, Props & State</h1>

{/\* Counter Section \*/}

<div>

<h2>Simple Counter</h2>

<p>Current Count: {count}</p>

<div>

<button onClick={decrementCount}>

Decrement

</button>

<button onClick={incrementCount}>

Increment

</button>

</div>

</div>

{/\* Greeting Cards Section using functional component \*/}

<div>

<GreetingCardFunctional name="Alice" message="Welcome to React!" />

<GreetingCardFunctional name="Bob" message="Hope you enjoy learning!" />

<GreetingCardFunctional name="Charlie" message="Let's build something great!" />

</div>

</div>

);

}

export default App;

**Observe your browser:** You should now see a counter with "Decrement" and "Increment" buttons. Click them and watch the number change! This demonstrates useState in action within a functional component.

**Part 6: Exploring React Dev Tools (3 minutes)**

The React Developer Tools are essential for debugging and understanding your React applications.

**Install the React Developer Tools extension** for your browser (Chrome or Firefox).

**Chrome:** Search "React Developer Tools" in the Chrome Web Store.

**Firefox:** Search "React Developer Tools" in Firefox Add-ons.

**Open your browser's developer tools** (usually by pressing F12 or right-clicking and selecting "Inspect").

**Look for the "Components" tab** (or "⚛️ Components" tab). Click on it.

**Explore the component tree:**

You should see your App component and its children (GreetingCardFunctional).

Click on the App component. In the right panel, you'll see its "Hooks" section, showing State: 0 (or whatever the current count is). Click the + or - buttons in your app and watch the state update live in the Dev Tools!

Click on a GreetingCardFunctional component. In the right panel, you'll see its "Props" section, showing the name and message props that were passed to it.

**Look for the "Profiler" tab.** This is for performance analysis, which you'll learn about later, but it's good to know it's there.

**Part 7: Further Exploration & Resources (Optional, but highly recommended)**

**Sample React SPA Code:** The code you've written in App.js and GreetingCardFunctional.js is a simple example of a Single Page Application (SPA) structure. Notice how the page doesn't reload when you interact with the counter – only parts of the UI update.

**Official React Documentation:** This is your best friend for learning React!

**React Docs (New Beta):** <https://react.dev/> (Highly recommended for modern React)

**React Docs (Old Stable):** <https://reactjs.org/>

Specifically, look up "State: A Component's Memory" and "Passing Props to a Component" in the official docs.

**Activity Wrap-up:**

You've successfully created React components using both class and functional approaches.

You've learned how to pass and access data using props in both types of components.

You've implemented state management using the useState Hook in a functional component.

You've started using the React Dev Tools to inspect component state and props.