ENSF 381
Full Stack Web Development

Lecture 22: React Hooks

Slides: Ahmad Abdellatif, PhD

Instructor: Novarun Deb, PhD

UNIVERSITY OF CALGARY

Outline

- Hooks
- useState
- useContext

React Hooks

- Functions that allow components to manage state, side effects, and other features.
- Allow developers to write more concise and readable code.
- Encourage the separation of concerns and the creation of reusable logic.

React Hooks

Here are some of the commonly used React Hooks:

- useState
- useContext
- useEffect
- UseRef
- UseCallback

useState

- One of the fundamental React Hooks used to manage state in components.
- It allows developers to add state to the components, making them more powerful and versatile.
- Track of data that may change over time, causing the component to re-render.
- Hooks can **only be called inside components**.
- The useState hook returns an array with two elements:
  - The current state value.
  - A function that allows to update the state.

UseState - Syntax

```
import React, { useState } from 'react';

const [state, setState] = useState(initialState);
```

The current state value.

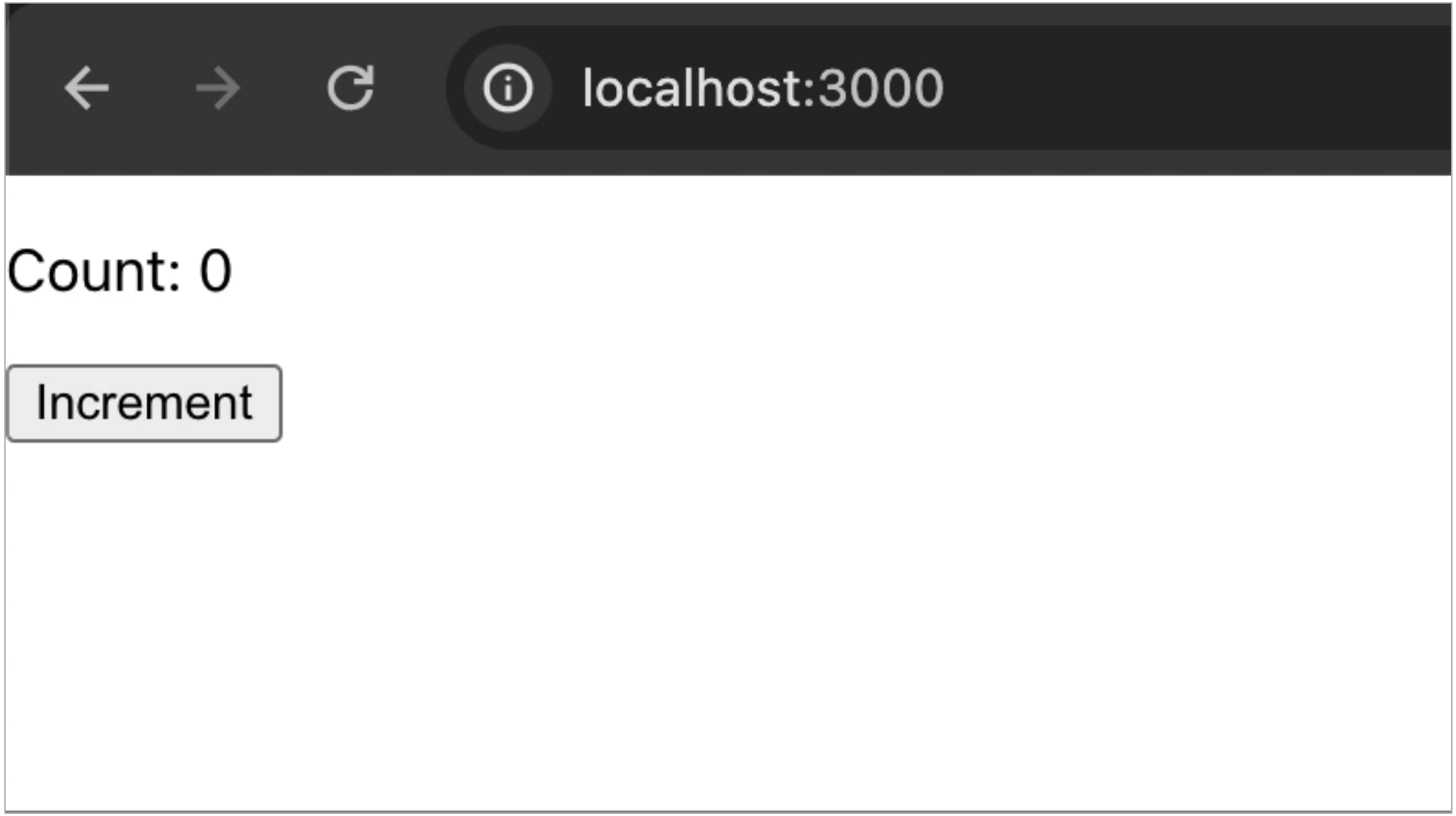
A function that can be used to update the state. It takes a new state value as an argument.

The initial value of the state.

Example on incrementing the count when the button is clicked using UseState

```
import React, { useState } from 'react';

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

// Define a function to handle incrementing the count
function handleIncrement() {
  setCount(count + 1);
};

return (
  <div>
    <p>Count: {count}</p>
    <button onClick={handleIncrement}>
      Increment
    </button>
  </div>
);
}
export default Counter;
```

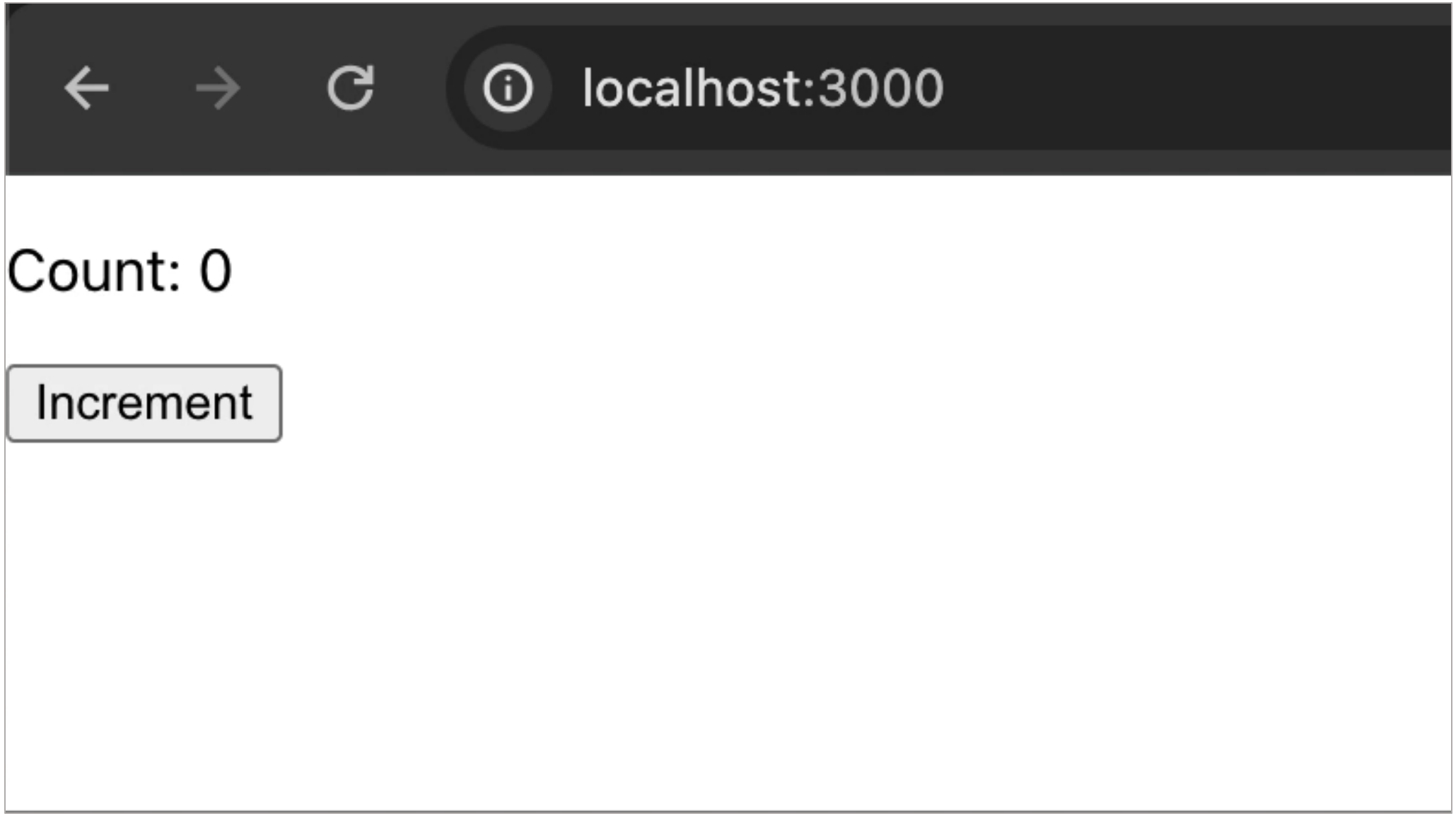
Example on incrementing the count when the button is clicked using UseState



Example on incrementing the count when the button is clicked using UseState – Arrow function

```
import React, { useState }  from 'react';

function App() {
// Declare a state variable named "count" with an initial value of 0
const [count, setCount] = useState(0);

return (
  <div>
    <p>Count: {count}</p>
    <button onClick={() => setCount(count + 1)}>
      Increment
    </button>
  </div>
);
}
export default App;
```

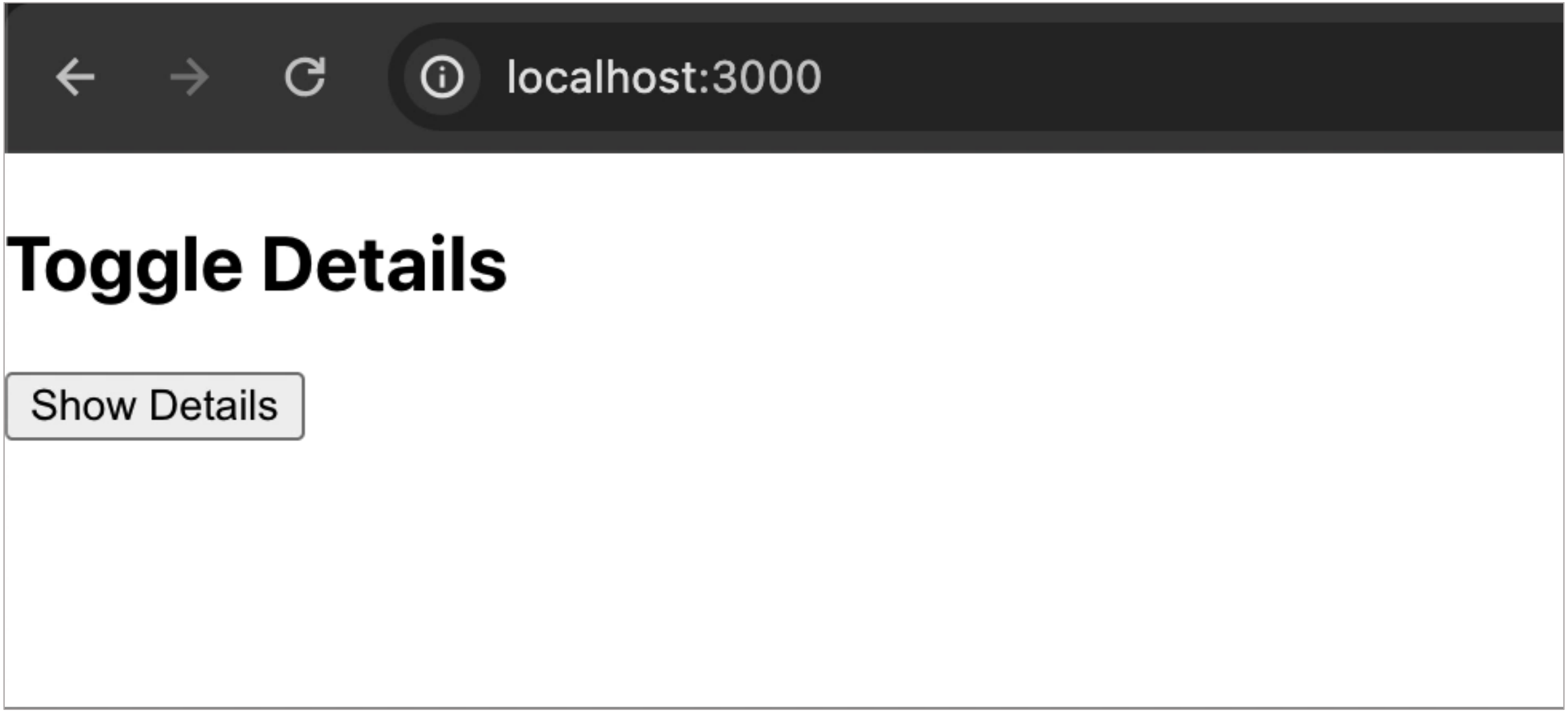
Example on incrementing the count when the button is clicked using UseState - Arrow function



Example on toggling the visibility of content using UseState

```
import React, { useState }  from 'react';

function App() {
// Declare a state variable to track the visibility of details
const [showDetails, setShowDetails] = useState(false);
return (
  <div>
    <h2>Toggle Details</h2>
    <button onClick={() => setShowDetails(!showDetails)}>
      {showDetails ? 'Hide Details' : 'Show Details'}
    </button>
// The content inside the {showDetails && ...} block is conditionally rendered based on the value of
showDetails. If showDetails is true, the additional information is displayed; otherwise, it remains
hidden.
    {showDetails && (
      <div>
        <p>This is additional information that can be toggled.</p>
      </div>
    )}
  </div>
);
}
export default App;
```

Example on toggling the visibility of content using UseState



Question….

What are some specific scenarios or types of applications where useState proves to be particularly useful?

- **Dynamic UI Updates**: facilitate dynamic updates in the UI based on user interactions. State changes **trigger re-renders**, ensuring the UI reflects the latest user input.

- **Form Handling**: each form input (like text fields, checkboxes, etc.) can have its state managed independently.

Recap: Props – Example (App Component)

```
import React from 'react';
import ChildComponent from './ChildComponent.js';

function App() {
  const dataToPass = "Hello from Parent!";

  return (
    <div>
      <ChildComponent message={dataToPass} />
    </div>
  );
}

export default App;
```

Pass the ‘message’ from the parent component (App) to the child component (ChildComponent).

Recap: Props – Example (ChildComponent)

```
import React from 'react';

function ChildComponent (props) {
    return (
        <div>
            <p>{props.message}</p>
        </div>
            );
        };

export default ChildComponent;
```

What if we need to share state between nested components?

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


function Component1() {
  const [user, setUser] = useState("John");
  return (
    <div>
      <h1>{`Hello ${user}!`}</h1>
      <Component2 user={user} />
    </div>
  );
}
function Component2({ user }) {
  return (
    <div>
      <h1>Component 2</h1>
      <Component3 user={user} />
    </div>
  );
}
.
.
function Component10({ user }) {
  return (
    <div>
      <h1>Component 10</h1>
      <h2>{`Hello ${user} from Component 10!`}</h2>
    </div>
  );}
```

useContext

- Provides a way to access the values from a React context directly within a component.

- Making it easier to share data across **different parts of an application without prop drilling**.

- Allows developers to efficiently share and consume context values in a React application.

How useContext works?

1. **Create a Context**: this creates a context object with Provider and Consumer components.

```
import { createContext } from 'react';
const MyContext = createContext();
```

2. **Provide the Context Value**: wrap the component with a Provider component and pass the value you want to share through the context.

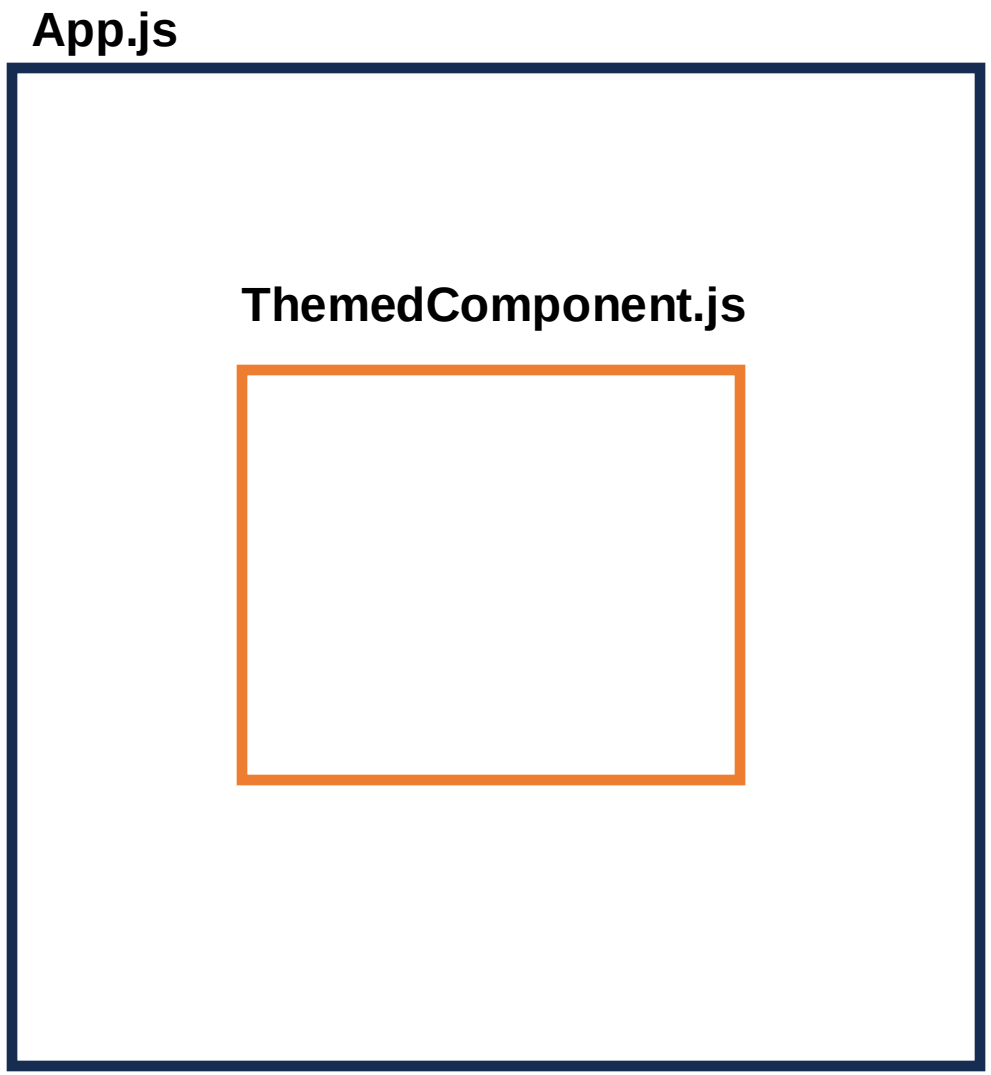
```
<MyContext.Provider value={/* some value */}>
  {/* Your component tree */}
</MyContext.Provider>
```

How useContext works?

3. **Consume the Context Value**: provide an access the context value.

```
import React, { useContext } from 'react';

function MyComponent() {
  const contextValue = useContext(MyContext);

  // Now you can use contextValue in your component
}
```

UseContext - Example



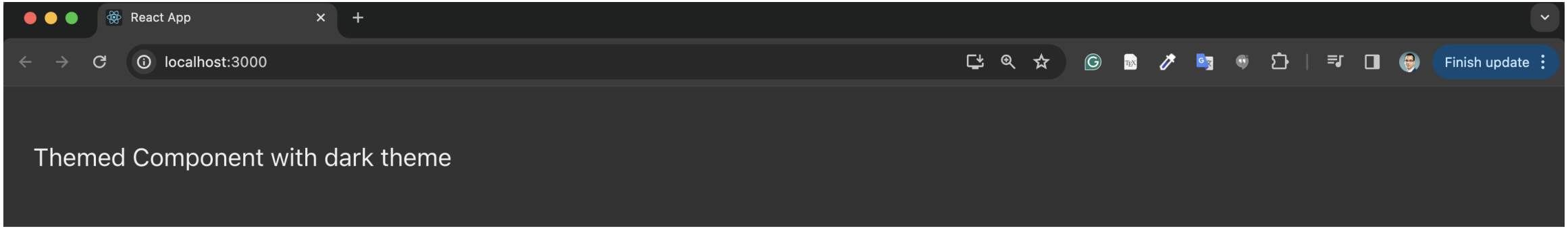
UseContext – Example (App)

```
import {React, createContext} from 'react';
import ThemedComponent from './ThemedComponent';

export const ThemeContext = createContext(null);

function App () {
const theme = 'dark';

return (
  <ThemeContext.Provider value={{theme}}>
    <ThemedComponent />
  </ThemeContext.Provider>
);
};

export default App;
```

UseContext – Example (ThemedComponent)

```
import React from 'react';
import { useContext } from 'react';
import { ThemeContext } from './App';

function ThemedComponent() {
const { theme } = useContext(ThemeContext);

return (
  <div style={{ background: theme === 'light' ? '#f0f0f0' : '#333', padding: '20px'
}}>
    <p style={{ color: theme === 'light' ? '#333' : '#f0f0f0' }}>
      Themed Component with {theme} theme
    </p>
  </div>
);
}

export default ThemedComponent;
```

UseContext - Example



UseContext – Example (App)

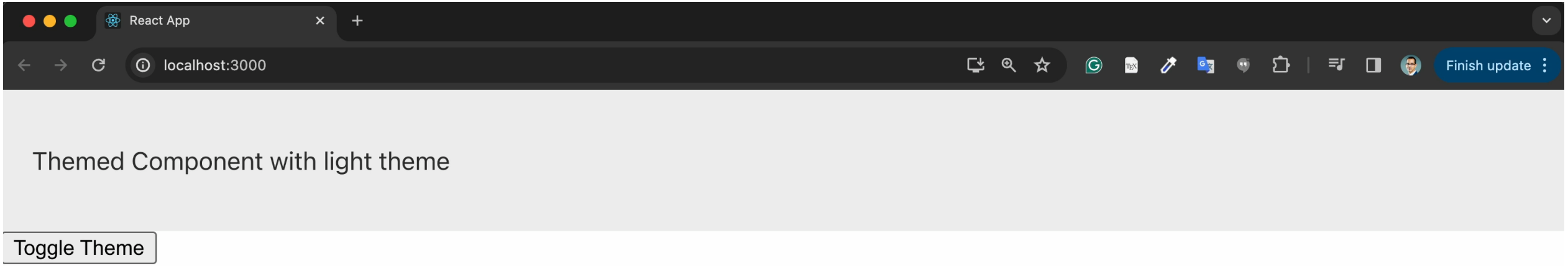
```
import {React, createContext} from 'react';
import ThemedComponent from './ThemedComponent';

export const ThemeContext = createContext(null);

function App () {
const theme = 'light'; // You might get this from state or another source

return (
  <ThemeContext.Provider value={{theme}}>
    <ThemedComponent />
  </ThemeContext.Provider>
);
};

export default App;
```

UseContext - Example

Example on toggling the theme on button click (App)

```
import {React, createContext, useState} from 'react';
import ThemedComponent from './ThemedComponent';

export const ThemeContext = createContext(null);

function App () {

const [theme, setTheme] = useState('light');
function toggleTheme(){
setTheme(theme === 'light' ? 'dark' : 'light');
};

return (
  <div>
    <ThemeContext.Provider value={{theme}}>
      <ThemedComponent />
    </ThemeContext.Provider>
    <button onClick={toggleTheme}>Toggle Theme</button>
  </div>
);
};

export default App;
```

Example on toggling the theme on button click



Questions