ENSF 381
Full Stack Web Development

Lecture 23: useEffect and Fetch

Slides: Ahmad Abdellatif, PhD

Instructor: Novarun Deb, PhD

UNIVERSITY OF CALGARY

Outline

- useEffect.
- Fetch.

useEffect

- A hook that allows components to perform side effects in a React application.

- Side effects in this context refer to operations that are not directly related to rendering the user interface.

- Side effects can include things like:
  - Fetching data from an API.
  - Subscribing to external data sources.
  - Any other asynchronous or imperative operations.

useEffect - Syntax

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

function MyComponent() {
  useEffect(() => {
  // Side effect logic goes here
 // This will run after the initial render and after every re-render

  }, [dependencies]);

  // Rest of the component code
}
```

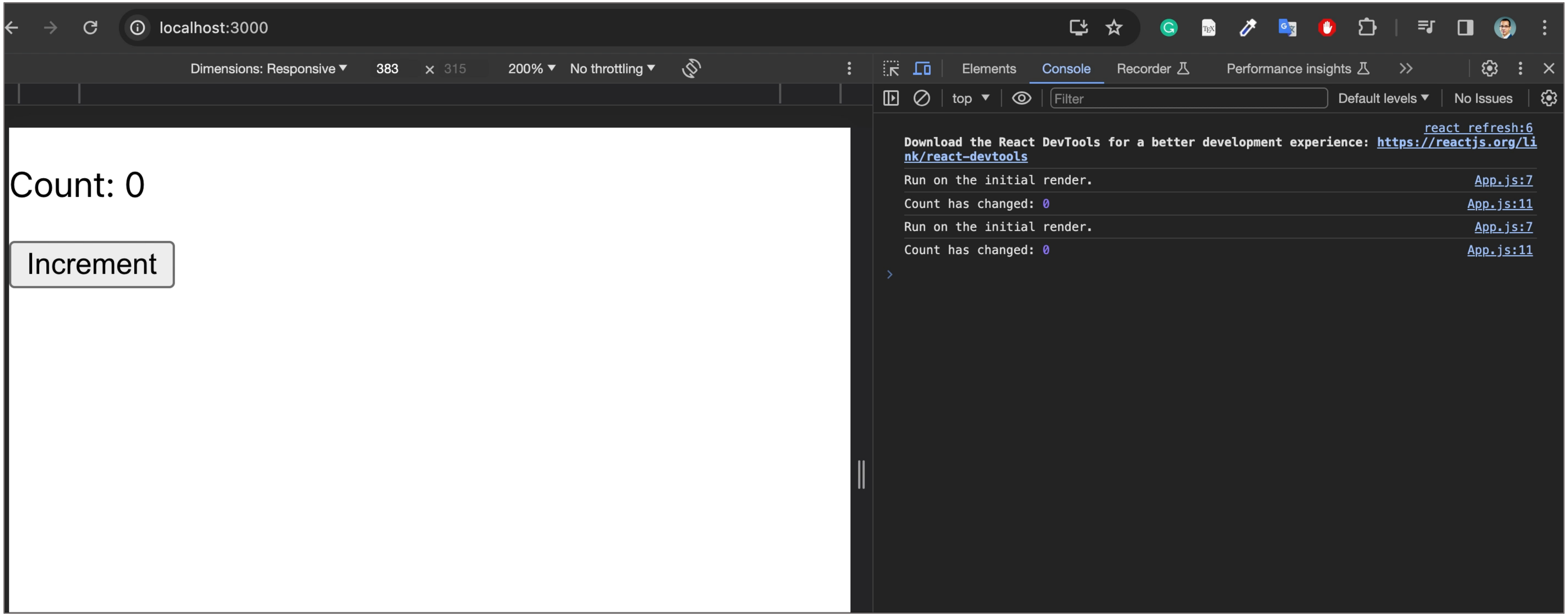
useEffect - Syntax

- useEffect accepts two arguments.
- Effect function: a function that contains the code we want to run.
- Dependencies array: an array of dependencies. It determines when the effect function should run.
  - If the array is empty, the effect runs only once after the initial render.
  - If you provide dependencies, the effect will run whenever any of the dependencies change.

useEffect - Example

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

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

useEffect(() => { // Effect for running code on the initial render
  console.log('Run on the initial render.');
}, []);

useEffect(() => {// Effect for running code on the first render when the 'count' state changes
  console.log('Count has changed:', count);
}, [count]);

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

useEffect - Example



```
                                                                  react_refresh:6
Download the React DevTools for a better development experience: https://reactjs.org/link/react-devtools
Run on the initial render.                                        App.js:7
Count has changed: 0                                              App.js:11
Run on the initial render.                                        App.js:7
Count has changed: 0                                              App.js:11
```

**The app rendered twice due to the use of StrictMode. In other words, StrictMode causes the app to render twice during development (but not in production).

Recap: What is asynchronous programming?

- A programming paradigm in which the execution of code does not occur in a sequential and blocking manner.
- Asynchronous programming allows certain operations to be initiated and continue executing without waiting for their completion.
- Asynchronous tasks in JavaScript do not impede the main thread.

Recap: Asynchronous programming is crucial in web development

There are tasks often have to wait for some work to finish before they can be completed:

- Fetch data from external servers.
- Access a database.
- Stream video or audio content.
- Handling user input.
- Performing time-consuming operations can introduce delays that would negatively impact the user experience.

Recap: Async/Await syntax

```
async function fetchData() {
  try {
    //make an asynchronous request to the specified URL
    const response = await fetch('https://jsonplaceholder.typicode.com/todos/1');

    //Parse the response body as JSON; wait for this asynchronous operation to complete
    const data = await response.json();

    // Process the fetched data
    console.log('Fetched Data:', data);
  } catch (error) {
    // Handle errors
    console.error('Error fetching data:', error);
  }}
```

`async:` declares a function as asynchronous that returns a Promise.

`fetch(URL):` A function used to make network requests, typically to retrieve data from a server.

`await:` within an asynchronous function, you can use the await keyword to pause the execution of the function until the Promise is resolved.

Recap: Async/Await - example

```html
<!DOCTYPE html>
<html><head>
  <title>Fetch Data Example</title>
</head>
<body>
<button onclick="fetchData()">Click here to fetch data!</button>
<script>
// Function to fetch data asynchronously using the Fetch API
async function fetchData() {
  try {
    const response = await fetch('https://jsonplaceholder.typicode.com/todos/1');
    const data = await response.json();
    // Process the fetched data
    console.log('Fetched Data:', data);
  } catch (error) {
    // Handle errors
    console.error('Error fetching data:', error);
  }
}
</script>
</body>
</html>
```

Recap: Async/Await - example

Fetch in React

- The fetch function is not specific to React.
- It is a part of the JavaScript language and is used for making HTTP requests.
- Fetch in React often involves integrating it with React's state management, component lifecycle, and JSX rendering to create dynamic and responsive user interfaces.

Fetch in React - Example

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

function App() {
const [email, setEmail] = useState(null);
const [cellphone, setCellphone] = useState(null);
const [isLoading, setIsLoading] = useState(false);
```

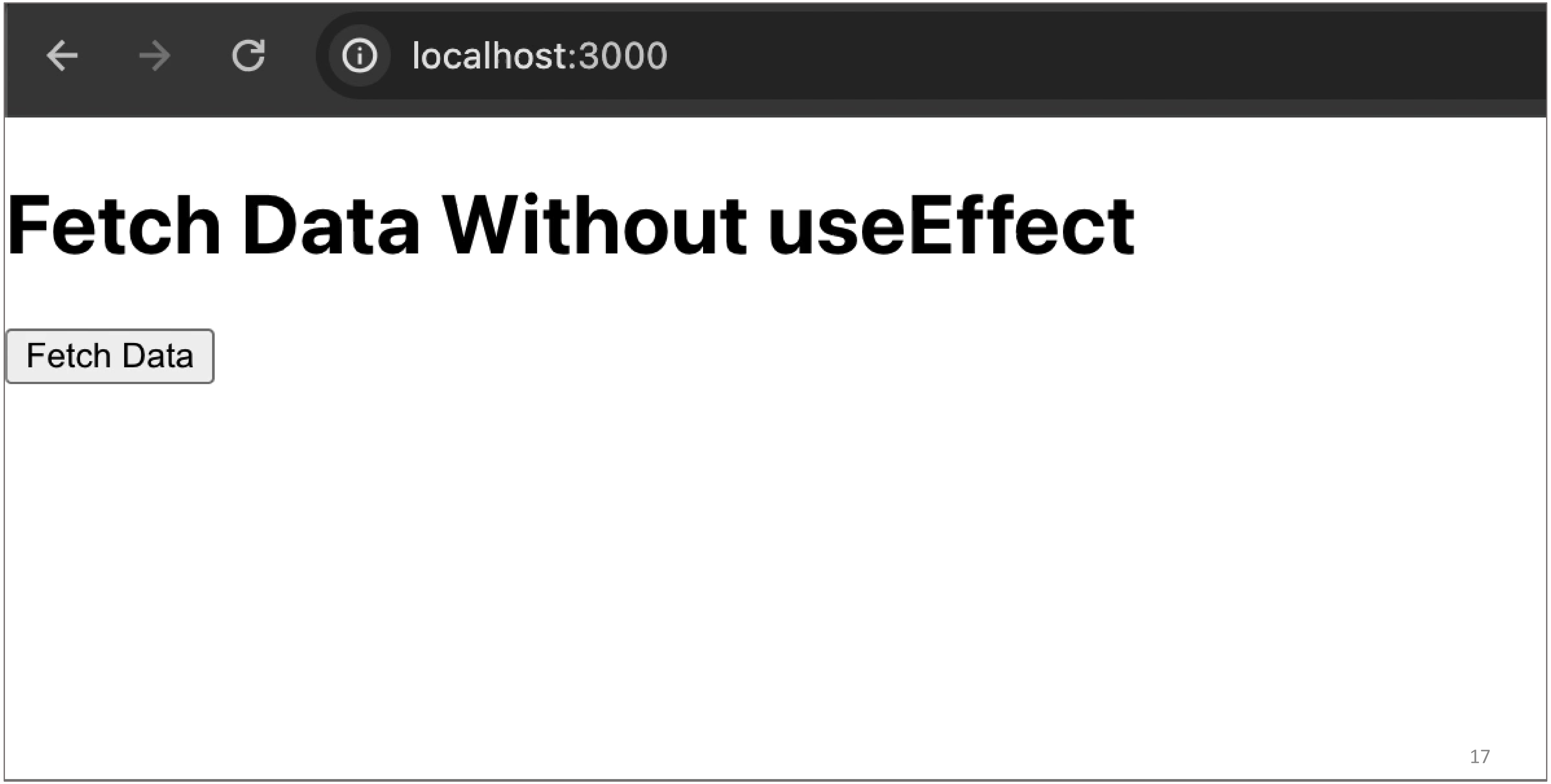
Fetch in React - Example

```
async function fetchData() {
  try {
    // Set loading to true while data is being fetched
    setIsLoading(true);

    // Fetch data from an API
    const response = await fetch('https://api.randomuser.me/?nat=US&results=1');

    if (response.ok) {
      // Check if the request was successful
      let { results } = await response.json();

      // Parse the JSON data from the response
      let { email, cell } = results[0];

      // Set the fetched data to the state
      setEmail(email);
      setCellphone(cell);
    } else {
      // Handle error if the request was not successful
      console.error('Failed to fetch data:', response.statusText);
    }
  } catch (error) {
     // Handle network errors or other exceptions
     console.error('Error during data fetching:', error);
  } finally {
    setIsLoading(false); // Set loading to false once data fetching is complete
  }}
```

Fetch in React - Example

```
return (
  <div>
    <h1>Fetch Data Without useEffect</h1>
    {isLoading ? (
      <p>Loading...</p>
    ) : (
      <div>
        <button onClick={fetchData}>Fetch Data</button>
        {email && (
          <div>
            <h2>Fetched Data:</h2>
            <pre>{email}</pre>
            <pre>{cellphone}</pre>
          </div>
        )}
      </div>
    )}
  </div>
);
}
export default App;
```

Fetch in React - Example



localhost:3000

Fetch Data Without useEffect

Fetching data using .then()

- Promises are a way to handle asynchronous operations in JavaScript.
- .then() executes the code after a Promise is successfully resolved.
- Multiple .then() methods can be chained to handle sequential steps in asynchronous operations.
- Enhances readability and makes it easier to handle success and error scenarios.

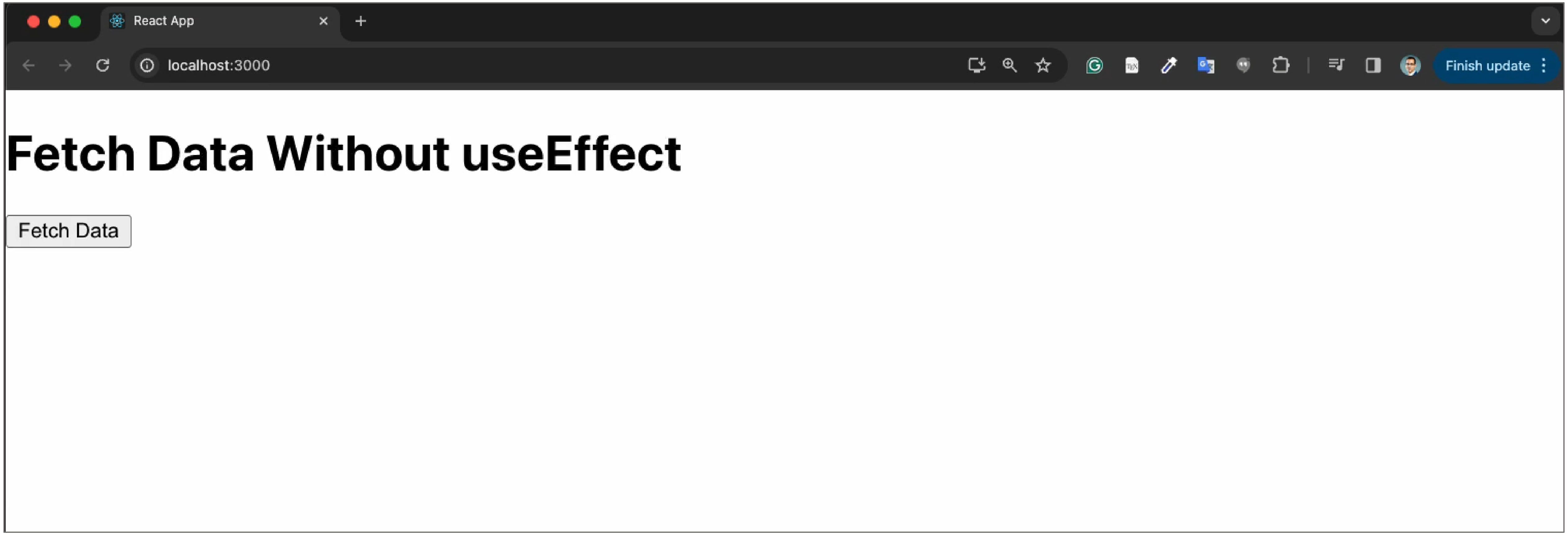
Fetching data using .then() - syntax

```
fetch('ENDPOINT URL')
  .then((response) => {
    // Handle the response (can be checking for errors, parsing JSON)
    return response.json(); // Parse the JSON data from the response
  })
  .then((data) => {
    // Handle the parsed data
    console.log('Data:', data);
  })
  .catch((error) => {
    // Handle any errors that occurred during the fetch
    console.error('Fetch error:', error);
  });
```

Fetching data using .then() - Example

```
function fetchData() {
    // Set loading to true while data is being fetched
    setIsLoading(true);

    // Fetch data from an API using .then()
    fetch('https://api.randomuser.me/?nat=US&results=1')
      .then((response) => response.json())
      .then((data) => {
        // Parse the JSON data from the response
        let { email, cell } = data.results[0];

        // Set the fetched data to the state
        setEmail(email);
        setCellphone(cell);
      })
      .catch((error) => {
        // Handle error if the request was not successful
        console.error('Failed to fetch data:', error.message);
      })
      .finally(() => {
        // Set loading to false once data fetching is complete
        setIsLoading(false);
      }); }
```

Fetching data using .then() - Example



Fetch Data Without useEffect

Importance of useEffect with fetch

- Fetching data directly in a component may lead to unintended behaviors.
- Without useEffect, **fetch requests might be triggered on every render**.
- This generates unnecessary network requests.
- useEffect ensures that the effect runs only when necessary, preventing unnecessary requests.

Example: Fetching Data with useEffect

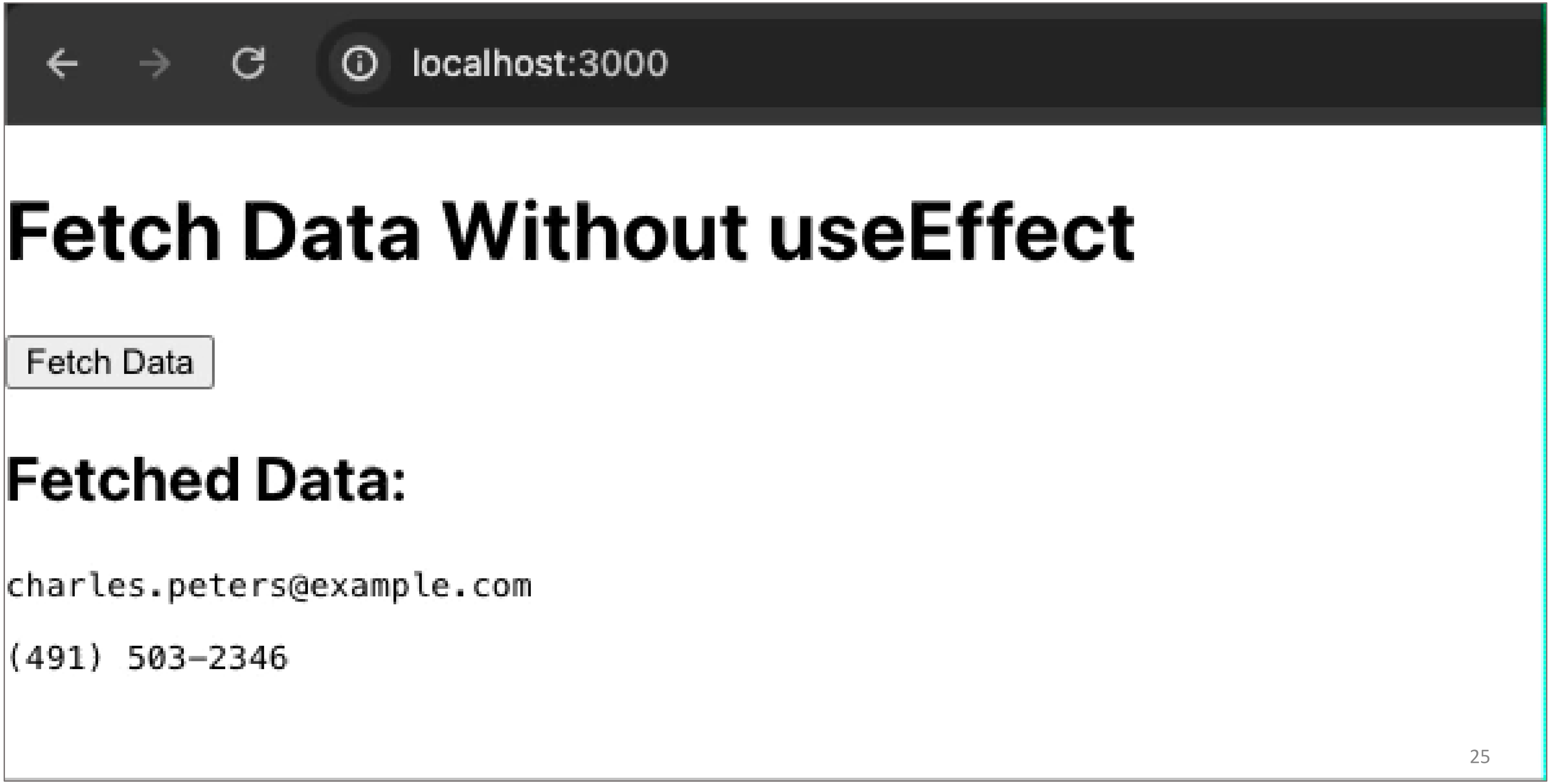
```
import React, { useState, useEffect } from 'react';

function App() {
const [email, setEmail] = useState(null);
const [cellphone, setCellphone] = useState(null);
const [isLoading, setIsLoading] = useState(false);
```

Example: Fetching Data with useEffect

```
function fetchData() {
  // Set loading to true while data is being fetched
  setIsLoading(true);
  // Fetch data from an API using .then()
  fetch('https://api.randomuser.me/?nat=US&results=1')
    .then((response) => response.json())
    .then((data) => {
      // Parse the JSON data from the response
      let { email, cell } = data.results[0];

      // Set the fetched data to the state
      setEmail(email);
      setCellphone(cell);
    })
    .catch((error) => {
      // Handle error if the request was not successful
      console.error('Failed to fetch data:', error.message);
    })
    .finally(() => {
      // Set loading to false once data fetching is complete
      setIsLoading(false);
    });}
```

```
useEffect(() => {
  fetchData();
},[]);
```

Example: Fetching Data with useEffect



Fetch Data Without useEffect

Fetched Data:

```
charles.peters@example.com

(491) 503-2346
```

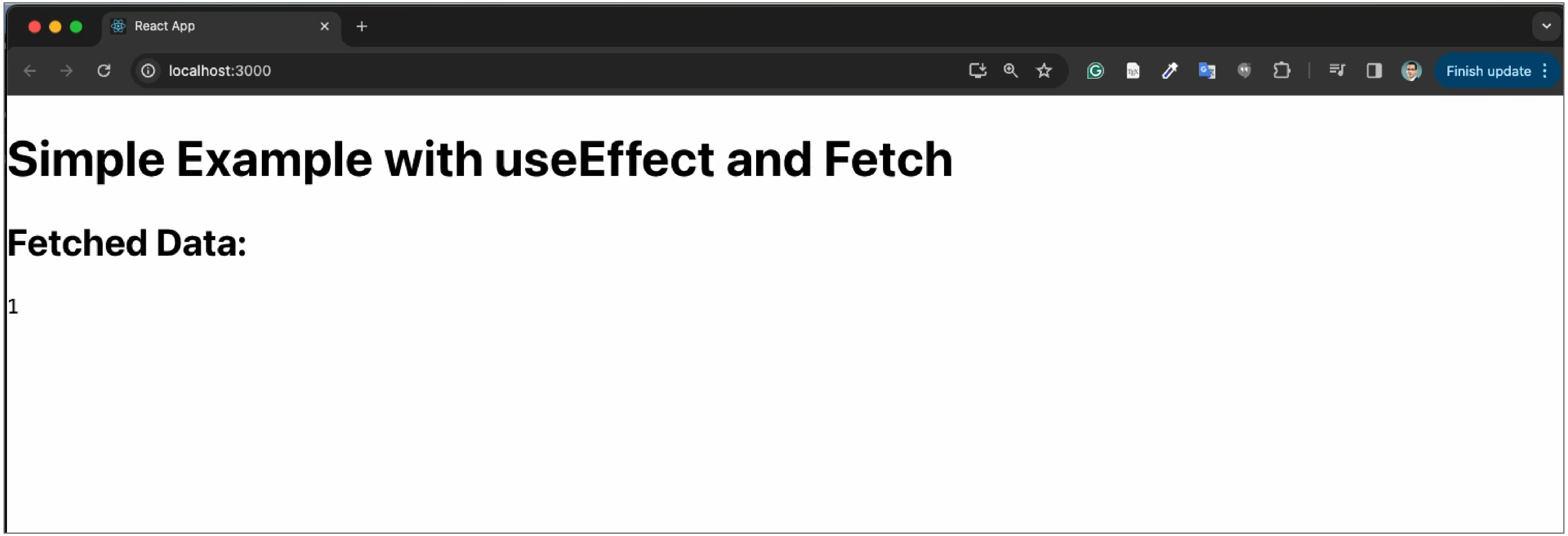
Example 2: Fetching Data with useEffect

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

function App() {
// State to store the fetched data
const [data, setData] = useState(null);
// State to track loading status
const [isLoading, setIsLoading] = useState(true);
// Effect to fetch data when the component mounts
useEffect(() => {
  // Fetch data from an API
  fetch('https://jsonplaceholder.typicode.com/todos/1')
    .then((response) => response.json())
    .then((result) => {
      // Set the fetched data to the state
      setData(result);
      // Set loading to false once data fetching is complete
      setIsLoading(false);
    })
    .catch((error) => {
      // Handle errors
      console.error('Error fetching data:', error);
      // Set loading to false in case of an error
      setIsLoading(false); }); }, []);
```

Example 2: Fetching Data with useEffect

```
return (
  <div>
    <h1>Simple Example with useEffect and Fetch</h1>
    {isLoading ? (
      <p>Loading...</p>
    ) : (
      <div>
        <h2>Fetched Data:</h2>
        <pre>{data.userId}</pre>
      </div>
    )}
  </div>
);
}

export default App;
```

Example 2: Fetching Data with useEffect



```
Simple Example with useEffect and Fetch

Fetched Data:

1
```

Questions