ENSF 381
Full Stack Web Development

Lecture 24: Storage and Styling

Slides: Ahmad Abdellatif, PhD

Instructor: Novarun Deb, PhD

UNIVERSITY OF CALGARY

Outline

- Local Storage
- React CSS Styling

Client-Side data storage

Sometimes we need to store data on the client-side:

- Storing user preferences and settings to provide a personalized experience.

- Persisting authentication tokens to keep users logged in across page reloads.

Local Storage

- Allows web applications to store data persistently on a user's device.

- A simple key-value storage mechanism and is designed to retain data even when the user closes the browser or navigates away from the page.

Local Storage - Syntax

- Set an item in Local Storage:

```
localStorage.setItem('key', 'value');
```

- Get an item from Local Storage:

```
const value = localStorage.getItem('key');
```

Local Storage - Syntax

- Remove an item from Local Storage:

```
localStorage.removeItem('key');
```

- Clear all items from Local Storage:

```
localStorage.clear();
```

Example on storing and retrieving user preferences

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

function App() {
const [theme, setTheme] = useState('light'); // State to track the current theme

// Function to toggle between light and dark themes
function toggleTheme() {
  const newTheme = theme === 'light' ? 'dark' : 'light';
  setTheme(newTheme);
  localStorage.setItem('theme', newTheme); // Save the theme (case-sensitive) preference to local storage
};

// Effect to retrieve the theme preference from local storage on component render
useEffect(() => {
  const savedTheme = localStorage.getItem('theme'); // Retrieve the theme preference from local storage
  setTheme(savedTheme ? savedTheme : 'light'); // Set the theme based on the stored preference or default to 'light'
}, []);

return (
  <div>
    <h1>Current Theme: {theme}</h1>
    <button onClick={toggleTheme}>Toggle Theme</button>
  </div>
);
};
export default App;
```

Example on storing and retrieving user preferences

Example on clearing the local storage

```
// Set an item in local storage
localStorage.setItem('username', 'john_doe');
```

```
// Get an item from local storage
let username = localStorage.getItem('username');
console.log('Username:', username);
```

```
// Remove an item from local storage
localStorage.removeItem('username');
```

```
// Clear all items from local storage
localStorage.clear();
```

```
username = localStorage.getItem('username');
```

```
console.log('username:', username); //Output: username: null
```

Question….

What are some specific scenarios where developers need to clear the local storage?

- **User Logout**: clear sensitive information when a user logs out of the application.

- **User Reset**: when a user explicitly resets their preferences.

React CSS styling

There are many ways to style React with CSS. The most common methods include:

- **CSS stylesheet**: a CSS stylesheet is an external file containing styles written in Cascading Style Sheets (CSS).

CSS stylesheet – Example (App.css)

```css
.my-component {
background-color: #f0f0f0;
padding: 20px;
border-radius: 8px;
box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}
.title {
color: red;
font-size: 24px;
}
.description {
color: #666;
font-size: 16px;
margin-top: 10px;
}
```

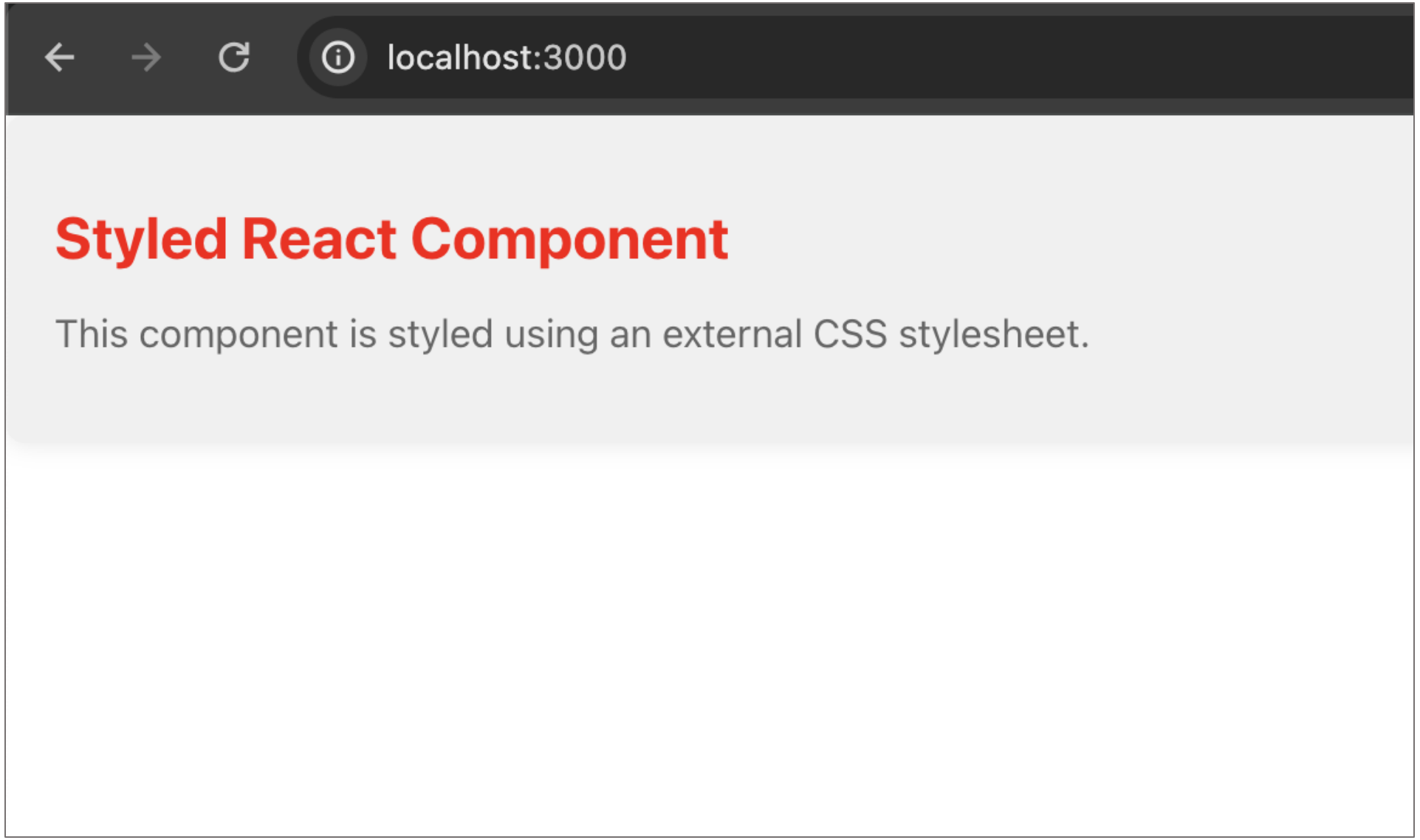
CSS stylesheet – Example (StyledComponent)

```
import React from 'react';
import './App.css';  // Import the CSS stylesheet

function StyledComponent() {
return (
  <div className="my-component">
    <h1 className="title">Styled React Component</h1>
    <p className="description">This component is styled using an
external CSS stylesheet.</p>
  </div>
);
};

export default StyledComponent;
```

CSS stylesheet – Example (App)

```
import React from 'react';
import StyledComponent from './StyledComponent';

function App() {
return (
  <div>
    <StyledComponent />
  </div>
);
};

export default App;
```

CSS stylesheet – Example (App)



But…

What is a potential issue of modifying the App.css?

- The styles you add or modify will likely affect the entire application.

- Solution: create separate CSS files for each component or module.

CSS stylesheet – Example (my-component.css)

```css
.my-component {
background-color: #f0f0f0;
padding: 20px;
border-radius: 8px;
box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}
.title {
color: red;
font-size: 24px;
}
.description {
color: #666;
font-size: 16px;
margin-top: 10px;
}
```

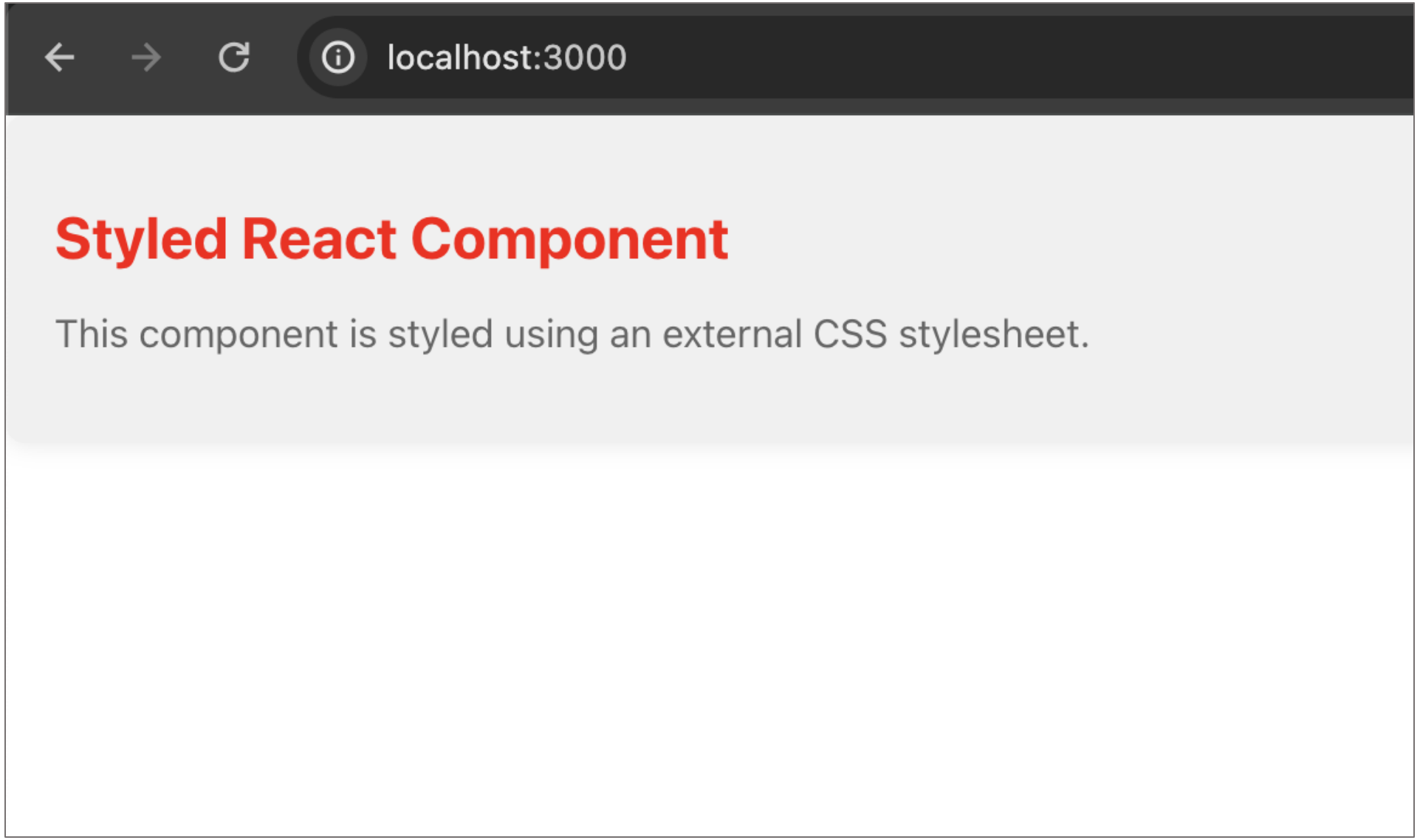
CSS stylesheet – Example (StyledComponent)

```
import React from 'react';
import './my-component.css'  // Import the CSS stylesheet

function StyledComponent() {
return (
  <div className="my-component">
    <h1 className="title">Styled React Component</h1>
    <p className="description">This component is styled using an
external CSS stylesheet.</p>
  </div>
);
};

export default StyledComponent;
```

CSS stylesheet – Example (App)

```
import React from 'react';
import StyledComponent from './StyledComponent';

function App() {
return (
  <div>
    <StyledComponent />
  </div>
);
};

export default App;
```

CSS stylesheet – Example (App)



React CSS styling

There are many ways to style React with CSS. The most common methods include:

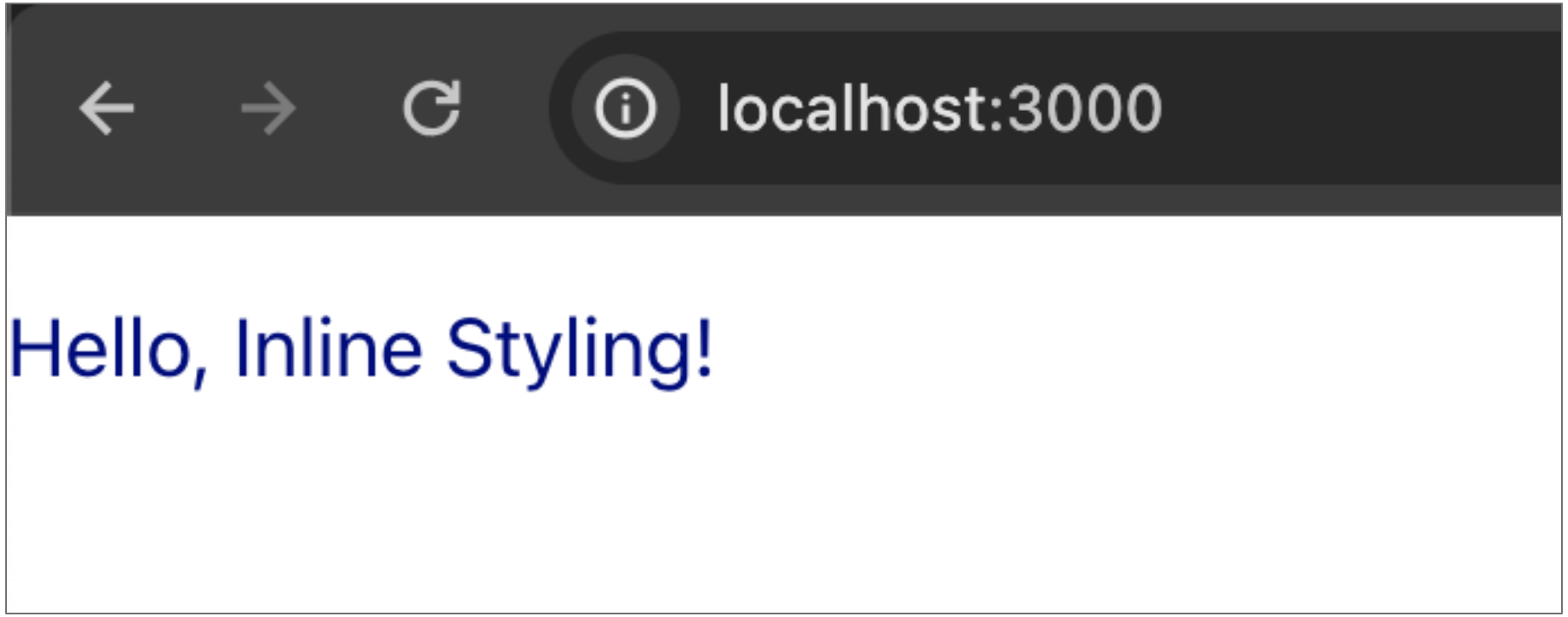
- **CSS stylesheet**: a CSS stylesheet is an external file containing styles written in Cascading Style Sheets (CSS).

- **Inline styling**: involves applying styles directly within HTML elements using the style attribute.

Inline styling - Example

```
import React from 'react';

function App() {
return (
  <div>

<p style={{ color: 'navy', fontSize: '18px' }}>Hello, Inline Styling!</p>

  </div>
);
};

export default App;
```

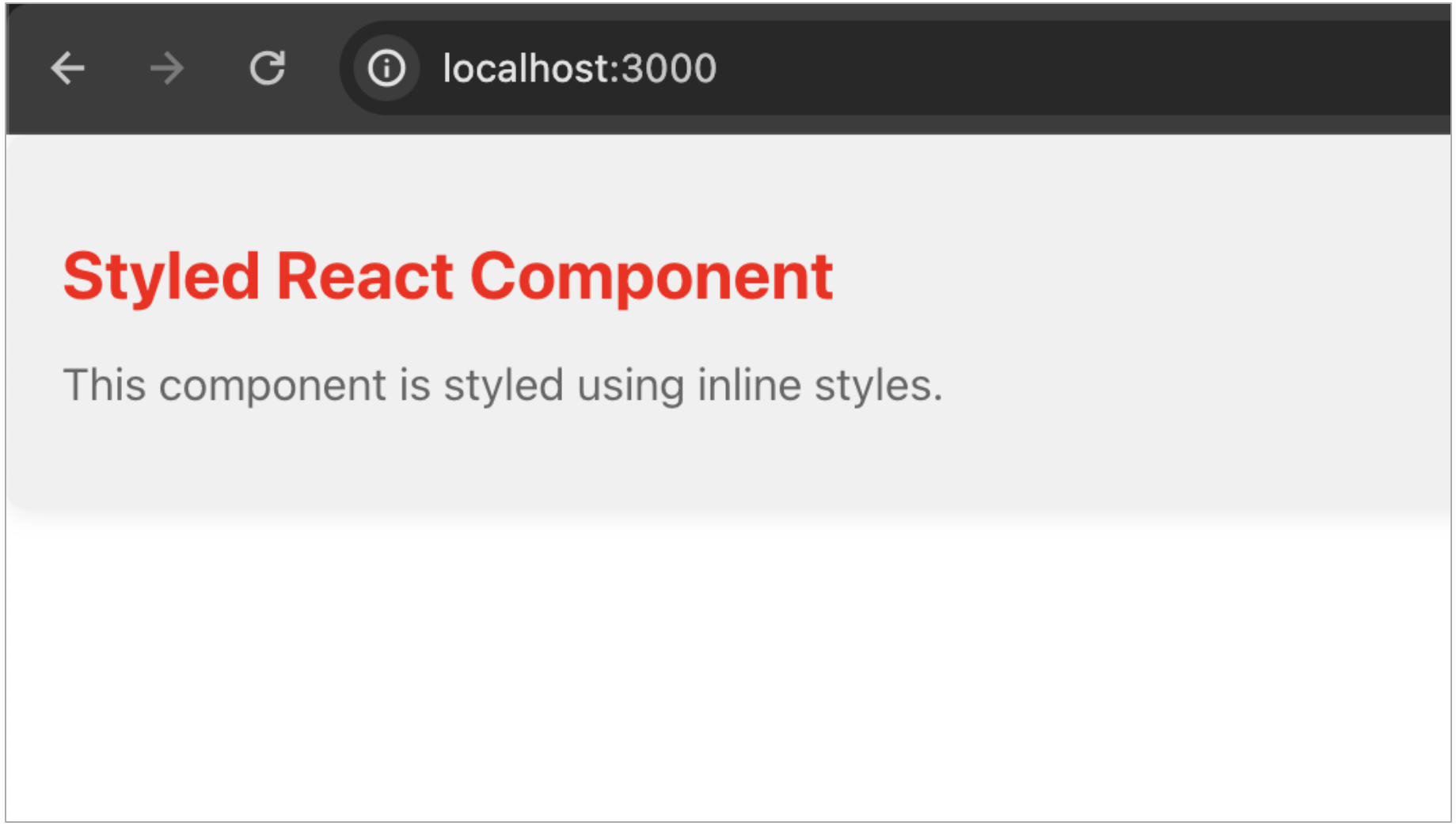
Inline styling - Example



Inline styling – Example 2

```
import React from 'react';
function App() {
// Inline styles as JavaScript objects
const componentStyles = {
  backgroundColor: '#f0f0f0',
  padding: '20px',
  borderRadius: '8px',
  boxShadow: '0 0 10px rgba(0, 0, 0, 0.1)',
};
const titleStyles = {
  color: 'red',
  fontSize: '24px',
};
const descriptionStyles = {
  color: '#666',
  fontSize: '16px',
  marginTop: '10px',
};
return (
  <div style={componentStyles}>
    <h1 style={titleStyles}>Styled React Component</h1>
    <p style={descriptionStyles}>This component is styled using inline styles.</p>
  </div>
);}; export default App;
```

Inline styling – Example 2



React CSS styling

There are many ways to style React with CSS. The most common methods include:

- **CSS stylesheet**: a CSS stylesheet is an external file containing styles written in Cascading Style Sheets (CSS).

- **Inline styling**: involves applying styles directly within HTML elements using the style attribute.

- **CSS modules**: are a CSS file organization technique in React that locally scopes styles to specific components. **Each component imports its own CSS module**, preventing style conflicts and allowing for encapsulation of styles within the component.

CSS modules

- We need to use styled-components library.
- Enable developers to write CSS in JS while building custom components in React.
- To install the library, run the following command:

```
npm install styled-components
```

CSS modules - Syntax

- Creating a Styled Component:

```
import styled from 'styled-components';
```

- Using the Styled Component:

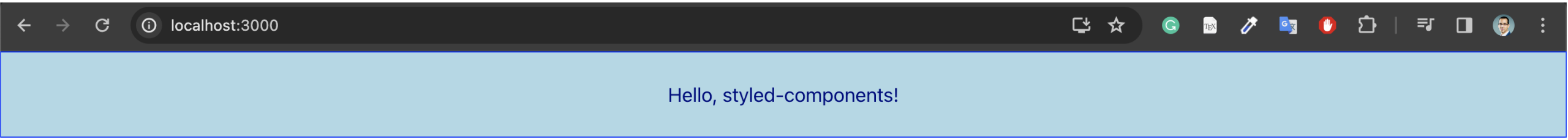
```
function MyComponent() {
  return (
    <StyledDiv>
      <p>This is a styled component.</p>
    </StyledDiv>
  );
};
```

CSS modules - Example

```
import React from 'react';
import styled from 'styled-components';

// Create a styled component using the styled() function
const StyledDiv = styled.div`
background-color: lightblue;
padding: 10px;
border: 1px solid blue;
text-align: center;
`;

const StyledText = styled.p`
color: navy;
font-size: 18px;
`;

function App() {
return (
  <StyledDiv>
    <StyledText>Hello, styled-components!</StyledText>
  </StyledDiv>
);};
export default App;
```

Create a styled version of a specific HTML element, in this case, a div.

CSS modules - Example



Use cases

CSS Stylesheet

- **Reusability**: when styles need to be shared across multiple components or pages.
- **Consistent Styling**: when consistency in styling across the application is a high priority.

Inline Styling:

- **Small Components**: for smaller, self-contained components where encapsulating styles is not a high priority.
- **Dynamic Styles**: when styles need to be computed dynamically based on component state or props.

CSS Modules:

- **Component-Level Styling**: when emphasizing component-level styling and encapsulation.
- **Dynamic Styling**: convenient for dynamic styles using props or theme variables.

Questions