**ReactJS Part-2 - Lab Assignment**

**1. Creating and Using Class Components with Constructors**

**Concepts Covered**: Class Components, Constructors, State Initialization

**Task**:

* + Create a **class component** with a constructor that initializes state.
  + Display a **welcome message** with the user's name stored in the state.
  + Example:



import React, { Component } from "react";

class Welcome extends Component {

constructor(props) {

super(props);

this.state = { name: "Alice" };

}

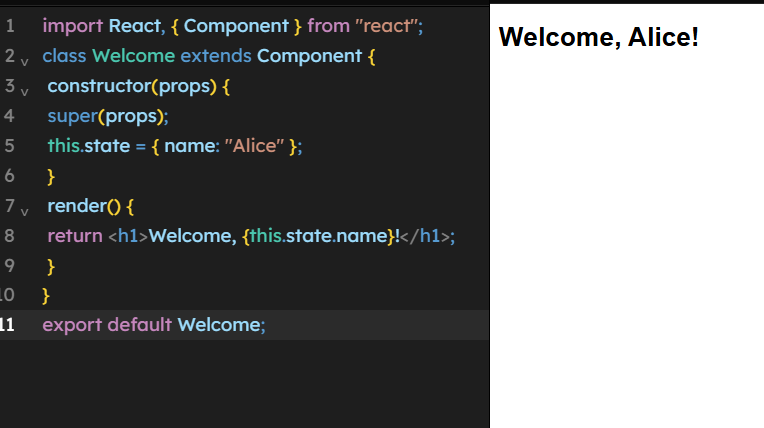
render() {

return <h1>Welcome, {this.state.name}!</h1>;

}

}

export default Welcome;





**2. Implementing Component Life Cycle Methods**

**Concepts**



**Covered**: componentDidMount, componentDidUpdate, componentWillUnmount **Task**:



* Create a class component that **fetches data from an API** in componentDidMount.
* Update the state when the user clicks a button (componentDidUpdate).
* Cleanup when the component is unmounted (componentWillUnmount).
* Example:

import React, { Component } from "react";

class DataFetcher extends Component {

constructor() {

super();

this.state = { data: "Loading..." };

}

componentDidMount() {

setTimeout(() => {

this.setState({ data: "API Data Loaded!" }); }, 2000);

}

componentDidUpdate() { console.log("Component Updated!");

}

componentWillUnmount() { console.log("Component Will Unmount");

}

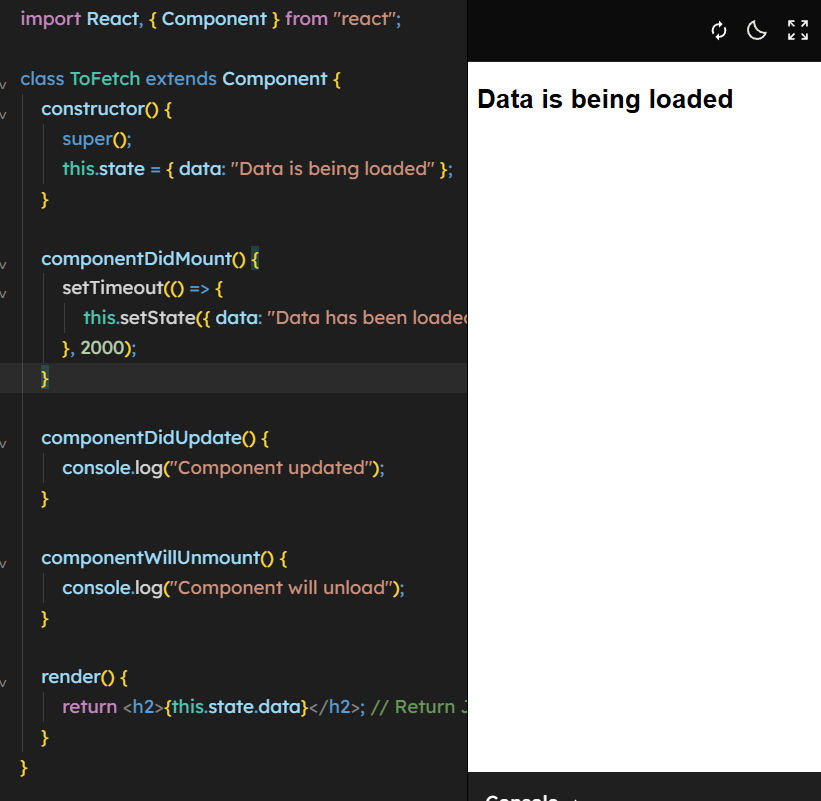
render() {

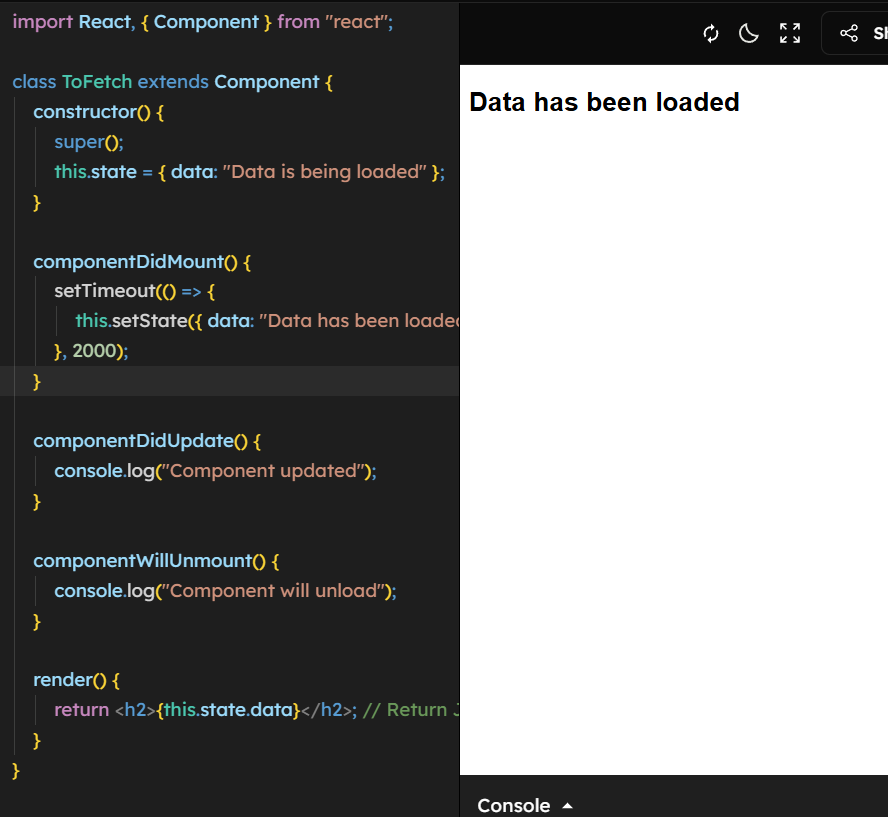
return <h2>{this.state.data}</h2>;

}

}

export default DataFetcher;







**3. Using React Component API: forceUpdate and shouldComponentUpdate**

**Concepts Covered**: forceUpdate, shouldComponentUpdate



**Task**:

* Create a class component that **prevents unnecessary updates** using shouldComponentUpdate.
* Use a button to **force update** the component.
* Example:

import React, { Component } from "react";

class ForceUpdateExample extends Component { shouldComponentUpdate() {

return false; // Prevent updates

}

render() {

return (

<div>

<h1>Current Time: {new

Date().toLocaleTimeString()}</h1>

<button onClick={() => this.forceUpdate()}>Update

Time</button>

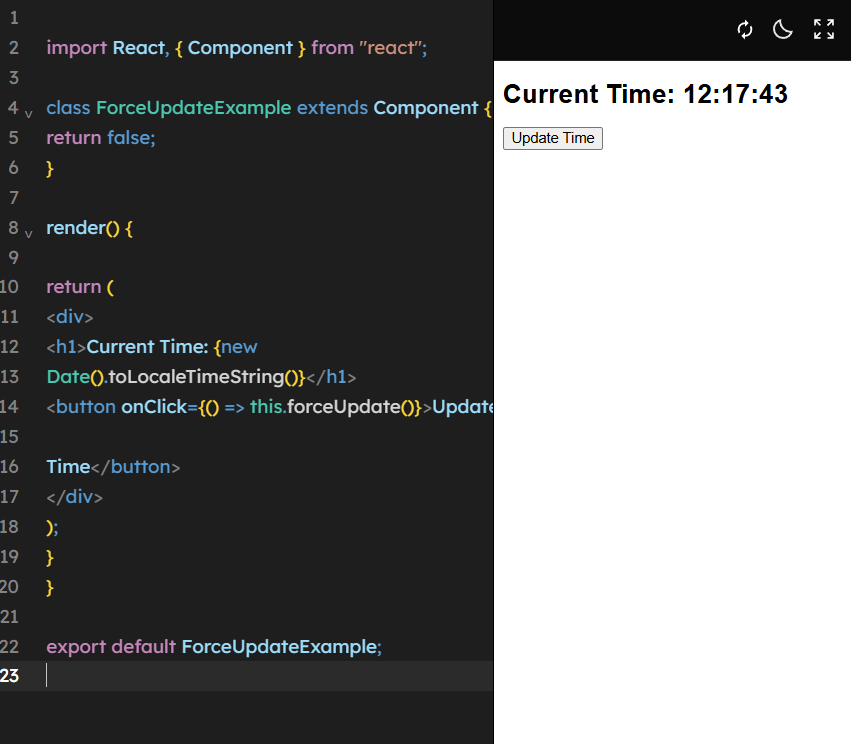
</div>

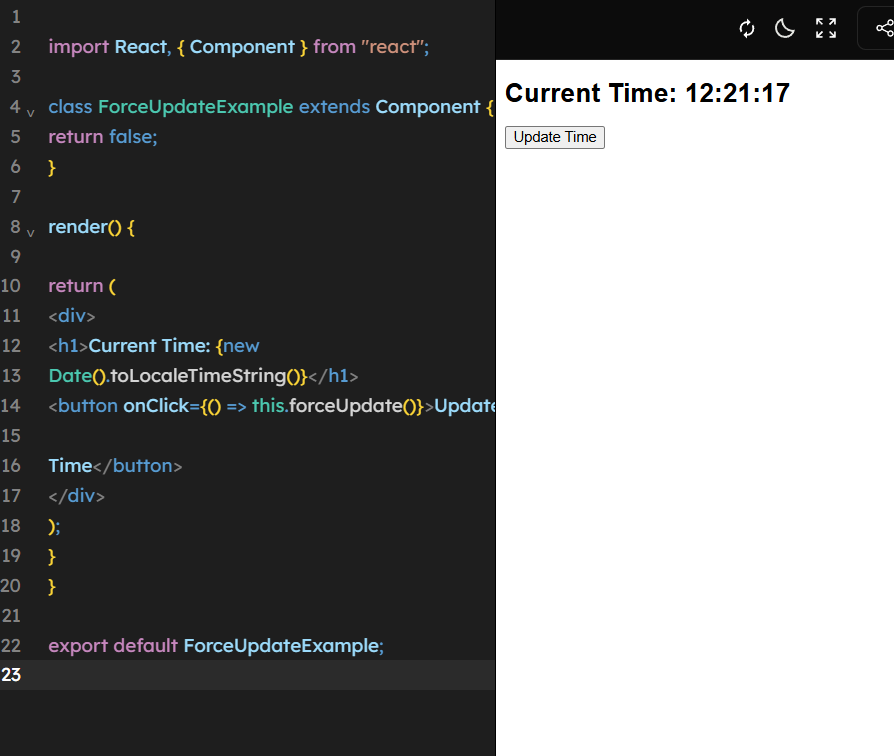
);

}

}

export default ForceUpdateExample;







**4. Debugging with React Developer Tools**

**Concepts Covered**: React Dev Tools **Task**:

* + Install **React Developer Tools**.
  + Open your React app in the browser.
  + Inspect components, modify state using DevTools, and analyze **re-renders**.
  + **Deliverable**: Screenshot of state modification via React Dev Tools.





**5. Comparing React Native and ReactJS**

**Concepts Covered**: Differences Between ReactJS and React Native

**Task**:

* + Write a **table comparison** between ReactJS and React Native.
  + Create a **React component** that displays this comparison.
  + Example:



function ComparisonTable() {

return (

<table border="1">

<thead>

<tr>

<th>Feature</th>

<th>ReactJS</th>

<th>React Native</th>

</tr>

</thead>

<tbody>

<tr>

<td>Platform</td>

<td>Web Applications</td>

<td>Mobile Applications</td>

</tr>

<tr>

<td>Rendering</td>

<td>Uses Virtual DOM</td>

<td>Uses Native Components</td>

</tr>

<tr>

<td>Styling</td>

<td>CSS</td>

<td>React Native Stylesheets</td>

</tr>

</tbody>

</table>

);

}

export default ComparisonTable;



**6. Creating a Parent-Child Component Structure**

**Concepts Covered**: Props, Parent-Child Communication

**Task**:

* + Create a **Parent Component** that passes **data** to a **Child Component** via props.
  + Example:



function ChildComponent(props) {

return <h2>Child Received: {props.message}</h2>;

}

function ParentComponent() {

return <ChildComponent message="Hello from Parent!" />;

}

export default ParentComponent;





**7. Managing State and Lifecycle with Hooks (useEffect)**

**Concepts Covered**: React Hooks, useEffect Lifecycle

**Task**:

* + Convert a class component with lifecycle methods into a **functional component using hooks**.
  + Example:



import { useState, useEffect } from "react";

function Timer() {

const [time, setTime] = useState(new Date().toLocaleTimeString());

useEffect(() => {

const interval = setInterval(() => { setTime(new Date().toLocaleTimeString());

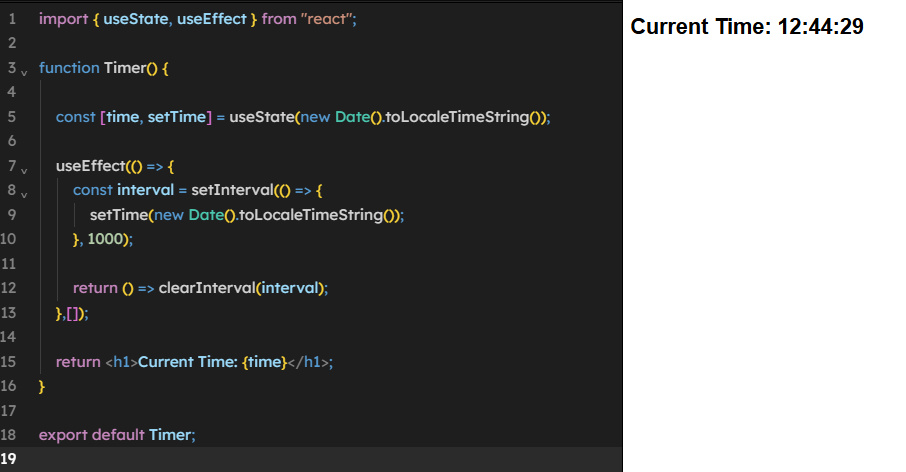
}, 1000);

return () => clearInterval(interval); // Cleanup on unmount }, []);

return <h1>Current Time: {time}</h1>;

}

export default Timer;





**8. Implementing Component Composition with Multiple Components**

**Concepts Covered**: Component Reusability, Composition

**Task**:

* + Create **Header, Content, and Footer** components.
  + Render them inside an **App component**.
  + Example:



function Header() {

return <h1>My Website</h1>;

}

function Content() {

return <p>This is the main content.</p>;

}

function Footer() {

return <p>© 2025 My Website</p>;

}

function App() {

return (

<div>

<Header />

<Content />

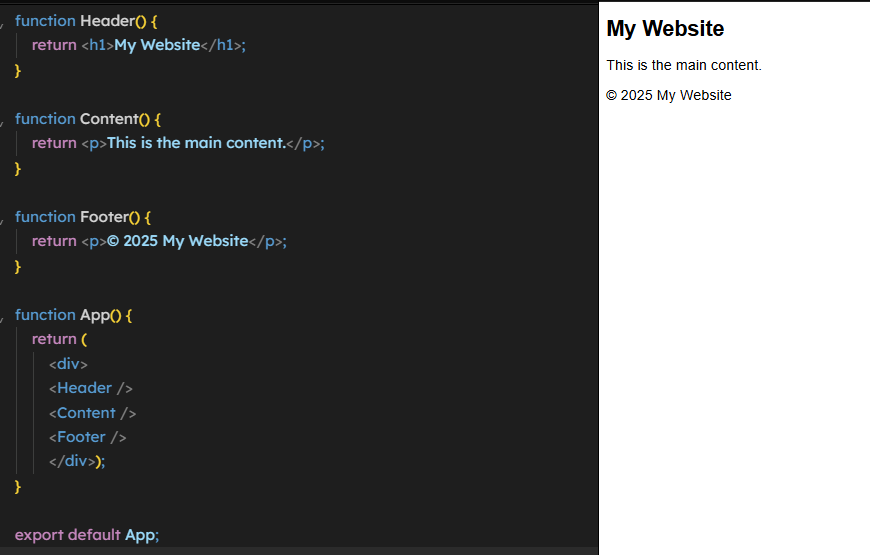
<Footer />

</div>

);

}

export default App;





**9. Simulating an API Call and Displaying Data**

**Concepts Covered**: Fetching Data in React

**Task**:

* + Use fetch() or axios to get data from an API.
  + Display the fetched data in a React component.
  + Example:



import { useState, useEffect } from "react";

function UserList() {

const [users, setUsers] = useState([]);

useEffect(() => {

fetch("https://jsonplaceholder.typicode.com/users")

.then((response) => response.json())

.then((data) => setUsers(data));

}, []);

return (

<ul>

{users.map((user) => (

<li key={user.id}>{user.name}</li>

))}

</ul>

);

}

export default UserList;



**10. Creating a Component with Controlled Inputs**

**Concepts Covered**: Handling User Input, State Management

**Task**:

* + Create a form component with an **input field** and a **button**.



* Update state when the user types.
* Example:

import { useState } from "react";

function NameForm() {

const [name, setName] = useState("");

return (

<div>

<input type="text" onChange={(e) => setName(e.target.value)} />

<p>Hello, {name}!</p>

</div>

);

}

export default NameForm;

