Class 2: Class-based vs. Function-based Components

Session Overview

By the end of this session, students will be able to:

- Understand the differences between class-based and function-based components.
- Learn to manage state in class components using this.state and this.setState().
- Utilize the useState hook for state management in functional components.
- Explore the use of props to pass data between components in both styles.
- Implement event-handling techniques for user interactions in components.
- Write and use JSX to create and render UI elements effectively.
- Set up and create a new React project using Create React App.
- Apply conditional rendering to control component output based on logic.
- Organize code by efficiently importing and exporting components.
- Build a simple application that integrates both class and functional components.

IRCTC PROJECT DEVELOPMENT

Step-1 Implement Header as a Class-Based Component

1.1 Create Header.js

- Navigate to the components directory and create a new file named Header.js.
- Write the following code in Header.js :

```
import React from "react";
import "./Header.css";

class Header extends React.Component {
  render() {
    return (
      <nav className="header">
        <h1>IRCTC</h1>
        <ul>
          <li>
            <a href="/">Home</a>
          </li>
          <li>
            <a href="/Login">Login</a>
          </li>
          <li>
            <a href="/register">Register</a>
          </li>
        </ul>
      </nav>
    );
  }
}

export default Header;
```

Explanation:

- **Import Statements:** The code begins by importing React, necessary for building the component
- **Class-Based Component:** Header is defined as a class component, which allows for state and lifecycle methods in the future, even if not used here.
- **Render Method:** The render() method returns the JSX to be displayed. It includes navigation links wrapped in a <nav> element.
- **Links:** Link components are used instead of anchor tags (<a>) for internal navigation, which prevents full page reloads.

Step-2 Create a CSS File for the Header

2.1 Create Header.css

1. In the same components directory, create a new file named Header.css.
2. Add the following styles to Header.css:

```
.header {
   display: flex;
   justify-content: space-between;
   align-items: center;
   background-color: #ffcc00; /* IRCTC yellow */
   padding: 10px 20px;
   color: #343a40; /* Dark text for contrast */
}

.header h1 {
   margin: 0;
   font-family: 'Arial', sans-serif; /* Simple, clean font */
}

.header ul {
   list-style-type: none;
   margin: 0;
   padding: 0;
   display: flex;
}

.header li {
   margin: 0 15px;
}

.header a {
   color: #343a40; /* Dark text for links */
   text-decoration: none;
   font-size: 16px;
}

.header a:hover {
   text-decoration: underline;
}
```

- Import Header.css in Header.js

Step 3: Update HomePage to Use the Header Component

3.1 Modify `HomePage.js`

1. **Open `HomePage.js`** in the `components` directory.
2. **Update the code to include the `Header` component:**

```
import React, { useState } from 'react';
import Header from './Header';
import './HomePage.css'; // Import the CSS file for styling

const HomePage = () => {

  const [formData, setFormData] = useState({ from: '', to: '', date: '' });

    const handleChange = (event) => {
      const { id, value } = event.target;
      setFormData({ ...formData, [id]: value });
    };

    const handleSubmit = (event) => {
      event.preventDefault();
      // Here you can handle form data submission
      console.log(formData);
    };

    return (
      <div className="homepage">
        <Header /> {/* Include the Header here */}
        <h2>Book Your Train Ticket</h2>
        <form onSubmit={handleSubmit}>
          <div className="form-group">
            <label htmlFor="from">From</label>
            <input type="text" id="from" value={formData.from} onChange={handleChange}
required />
          </div>
          <div className="form-group">
            <label htmlFor="to">To</label>
            <input type="text" id="to" value={formData.to} onChange={handleChange} required
/>
          </div>
          <div className="form-group">
            <label htmlFor="date">Date</label>
            <input type="date" id="date" value={formData.date} onChange={handleChange}
required />
          </div>
          <button type="submit">Search Trains</button>
        </form>
      </div>
  );
};

export default HomePage;
```

Explanation:

- **Importing Header**: The `Header` component is imported and used in the `HomePage` component, which centralizes the navigation.
- **State Management**: The `useState` hook manages the form data state. This enables the application to store user inputs dynamically.
- **Form Submission**: The `handleSubmit` function prevents the default form submission action and logs the form data to the console for testing.
-

Step 4: Manage State for Form Inputs

4.1 Implement State Management

1. Ensure `useState` is used for managing input fields in `HomePage.js`.
2. The form fields should bind their values to state:

```
<input type="text" id="from" value={formData.from} onChange={handleChange} required />
<input type="text" id="to" value={formData.to} onChange={handleChange} required />
<input type="date" id="date" value={formData.date} onChange={handleChange} required />
```

Explanation:

- **Controlled Components**: By binding the input fields to the component's state, they become "controlled components." This means their values are always in sync with the state, making it easier to manage user input and validation.

Step 5: Implement Event Throttling for Station Fields

5.1 Throttling the Input Handling

1. **Add throttling to the handleChange function in HomePage.js:**

```
import { useCallback } from 'react';

const handleChange = useCallback((event) => {
   const { id, value } = event.target;
   setFormData(prevData => ({ ...prevData, [id]: value }));
}, []);
```

Explanation:

- **Using `useCallback`:** This hook prevents the creation of a new function on every render, which helps with performance, especially if the function is passed as a prop to child components.
- **Debounce Logic:** For more advanced throttling, consider using a library like lodash's debounce to limit how often `handleChange` is called.
-

Step 6: Create Login and Register Pages

6.1 Create `Login.js`

1. In the `components` directory, create a new file named `Login.js`.
2. Add the following code:

```
import React from "react";
import "./Login.css";

const Login = () => {
  return (
    <div className="login-container">
      <div className="login-box">
        <h2>Login</h2>
        <form>
          <div className="form-group">
            <label htmlFor="username">Username</label>
            <input type="text" id="username" required />
          </div>
          <div className="form-group">
            <label htmlFor="password">Password</label>
            <input type="password" id="password" required />
          </div>
          <button type="submit">Login</button>
        </form>
      </div>
    </div>
  );
};

export default Login;
```

6.2 Create Login.css

```
/* Outer container to center the form on the page */
.login-container {
  display: flex;
  justify-content: center; /* Center horizontally */
  align-items: center; /* Center vertically */
  min-height: 100vh; /* Full viewport height */
  background-color: #f5f5f5; /* Light gray background */
}

/* Form box layout similar to IRCTC */
.login-box {
  background-color: #ffffff; /* White background for form */
  padding: 30px;
  border-radius: 10px; /* Rounded corners */
  box-shadow: 0 4px 12px rgba(0, 0, 0, 0.1); /* Soft shadow */
```

```css
  width: 100%; /* Full width of container */
  max-width: 450px; /* Max width to limit form size */
}

/* Form heading */
.login-box h2 {
  text-align: center;
  color: #007bff; /* IRCTC blue color for heading */
  margin-bottom: 20px;
}

/* Form group styling */
.form-group {
  margin-bottom: 20px;
}

/* Input field styling */
.form-group label {
  display: block;
  font-size: 14px;
  font-weight: bold;
  margin-bottom: 5px;
  color: #343a40;
}

input[type="text"],
input[type="password"] {
  width: 100%;
  padding: 12px;
  border: 1px solid #ced4da;
  border-radius: 4px;
  font-size: 14px;
  background-color: #ffffff;
  color: #495057;
}

input:focus {
  border-color: #007bff;
  outline: none;
  box-shadow: 0 0 5px rgba(0, 123, 255, 0.5);
}

/* Login button */
button {
  width: 100%;
  padding: 12px;
  background-color: #007bff; /* IRCTC blue color */
  color: white;
  font-size: 16px;
  font-weight: bold;
  border: none;
  border-radius: 4px;
  cursor: pointer;
  transition: background-color 0.3s ease;
}

button:hover {
  background-color: #0056b3;
}

button:active {
  background-color: #004085;
}
```

Step-7 Create Register.js

```
import React from "react";
import "./Register.css";

const Register = () => {
  return (
    <div className="register-container">
      <div className="register-box">
        <h2>Register</h2>
        <form>
          <div className="form-group">
            <label htmlFor="username">Username</label>
            <input type="text" id="username" required />
          </div>
          <div className="form-group">
            <label htmlFor="email">Email</label>
            <input type="email" id="email" required />
          </div>
          <div className="form-group">
            <label htmlFor="password">Password</label>
            <input type="password" id="password" required />
          </div>
          <button type="submit">Register</button>
        </form>
      </div>
    </div>
  );
};
```

```
export default Register;
```

7.2 Create Register.css and Import it in the Register.js

```
/* Outer container to center the form on the page */
.register-container {
  display: flex;
  justify-content: center; /* Center horizontally */
  align-items: center; /* Center vertically */
  min-height: 100vh; /* Full viewport height */
  background-color: #f5f5f5; /* Light gray background */
}

/* Form box layout similar to IRCTC */
.register-box {
  background-color: #ffffff; /* White background for form */
  padding: 30px;
  border-radius: 10px; /* Rounded corners */
  box-shadow: 0 4px 12px rgba(0, 0, 0, 0.1); /* Soft shadow */
  width: 100%; /* Full width of container */
  max-width: 450px; /* Max width to limit form size */
}

/* Form heading */
.register-box h2 {
  text-align: center;
  color: #007bff; /* IRCTC blue color for heading */
  margin-bottom: 20px;
}

/* Form group styling */
.form-group {
  margin-bottom: 20px;
}

/* Input field styling */
.form-group label {
  display: block;
  font-size: 14px;
  font-weight: bold;
  margin-bottom: 5px;
  color: #343a40;
}

input[type="text"],
input[type="email"],
input[type="password"] {
  width: 100%;
  padding: 12px;
  border: 1px solid #ced4da;
  border-radius: 4px;
  font-size: 14px;
  background-color: #ffffff;
  color: #495057;
}

input:focus {
  border-color: #007bff;
  outline: none;
  box-shadow: 0 0 5px rgba(0, 123, 255, 0.5);
}

/* Register button */
button {
  width: 100%;
  padding: 12px;
  background-color: #007bff; /* IRCTC blue color */
  color: white;
  font-size: 16px;
  font-weight: bold;
  border: none;
  border-radius: 4px;
  cursor: pointer;
  transition: background-color 0.3s ease;
}

button:hover {
  background-color: #0056b3;
}

button:active {
  background-color: #004085;
}
```

Interview and FAQ’s

1. Class-based Components

Overview: Class-based components are ES6 classes that extend from `React.Component`. They are used to create complex components with state and lifecycle methods.

Interview Questions:

- **What are class-based components in React?**
  - **Answer**: Class-based components are components defined using ES6 classes. They can hold and manage state and have access to lifecycle methods.
- **How do you initialize the state in a class-based component?**

Answer: State is initialized in the constructor using `this.state`. For example:

```
constructor(props) {
   super(props);
   this.state = { count: 0 };
}
```

FAQs:

- **What lifecycle methods are available in class-based components?**
  - **Answer**: Common lifecycle methods include `componentDidMount`, `componentDidUpdate`, and `componentWillUnmount`, which allow you to run code at specific points in a component's lifecycle.

---

2. Function-based Components

Overview: Function-based components are simpler, defined as JavaScript functions that return JSX. They can use hooks for state management.

Interview Questions:

- **What is a function-based component?**
  - **Answer**: A function-based component is a JavaScript function that returns JSX. It can accept props and utilize hooks like `useState`.
- **What are hooks in React?**
  - **Answer**: Hooks are functions that allow you to use state and other React features in function components, making them more powerful and flexible.

FAQs:

- **How do you define a simple function-based component?**

Answer: A basic function-based component could look like this:

```
const MyComponent = () => <h1>Hello, World!</h1>;
```

---

3. Using State with Functional Components

Overview: Functional components manage state using the `useState` hook, enabling dynamic updates to the UI.

Interview Questions:

- **How do you manage state in functional components?**

Answer: The state is managed using the `useState` hook, which provides a state variable and a function to update it:

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

- **Can you give an example of using `useState`?**

Answer:

```
const Counter = () => {

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

    return <button onClick={() => setCount(count +
1)}>{count}</button>;

};
```

FAQs:

- **What are the limitations of using state in functional components?**
  - **Answer**: While functional components can manage state, they lack lifecycle methods found in class components, but this is mitigated by using the `useEffect` hook for side effects.

---

4. Using State with Class-based Components

Overview: In class-based components, state is managed through `this.state` and updated via `this.setState()`.

Interview Questions:

- **How do you update state in a class-based component?**

Answer: State is updated using `this.setState()` which schedules an update and triggers a re-render:

```
this.setState({ count: this.state.count + 1 });
```

- **What is the constructor's role in class components?**
  - **Answer**: The constructor is used for initializing state and binding methods. It's called when the component is created.

FAQs:

- **What is the difference between state and props in React?**
  - **Answer**: The state is managed within a component and can change, while props are passed to components and are immutable.

---

5. Using Props with Functional Components

Overview: Props are passed as arguments to function components, allowing data to be shared from parent to child components.

Interview Questions:

- **How do you pass props to a functional component?**

Answer: Props are passed as an argument to the component function:

```
const Greeting = ({ name }) => <h1>Hello, {name}!</h1>;
```

What is prop drilling?

- ○ **Answer**: Prop drilling refers to passing props through multiple layers of components, which can lead to cumbersome code if many components are involved.

FAQs:

- **What happens if you do not provide props to a functional component?**
  - **Answer**: The component will still render, but the values for the props will be `undefined`.

---

6. Using Props with Class-based Components

Overview: Props in class-based components are accessed via `this.props`, allowing components to receive data from their parent.

Interview Questions:

- **How do you access props in a class-based component?**

Answer: Props are accessed using `this.props`. For example:

```
class Greeting extends React.Component {

    render() {

        return <h1>Hello, {this.props.name}!</h1>;
        }

}
```

- **Can you explain the significance of default props?**
    - **Answer**: Default props provide default values for props if none are passed, ensuring the component has a fallback value.

FAQs:

- **What is the purpose of prop types?**
    - **Answer**: Prop types are used to validate the types of props passed to components, providing warnings in development if props are of the wrong type.