Module No: 3

Subject: React

Week Number: 7

Table of Contents

Sl.no	Topic	Page No
1	jsx	03
2	Components	06
3	Brain teaser	12
4	Reference	16

JSX and Components

JSX

JSX (JavaScript XML) is a syntax used in React to describe the structure of the user interface (UI). It allows you to write HTML-like elements directly within JavaScript code, providing a more intuitive way to design React components. JSX gets transpiled to standard JavaScript using tools like Babel.

1.Key Concepts

A.Basic Features of JSX

1.HTML-like Syntax in JavaScript JSX resembles HTML but is embedded in JavaScript files.
Example:

```
const element = <h1>Hello, world!</h1>;
```

2.Expression Embedding Use curly braces {} to embed JavaScript expressions inside JSX.
Example:

```
const name = "Alice";
const greeting = <h1>Hello, {name}!</h1>;
```

3.Components JSX can render React components, both functional and class-based.
Example:

```
function Welcome() {
  return <h1>Welcome to React</h1>;
}

const element = <Welcome />;
```

B.JSX Syntax Rules

1.Single Parent Element JSX must return a single parent element. Use a `<div>` or

```
<React.Fragment> if needed.
jsx
Copy code
return (
  <div>
    <h1>Title</h1>
    <p>Description</p>
  </div>
);
```

Or use a **fragment**:

```
return (
  <>
    <h1>Title</h1>
    <p>Description</p>
  </>
);
```

2.HTML Attributes Use camelCase for attributes like `className`, `onClick`, etc.

```
const element = <button className="btn"
onClick={handleClick}>Click me</button>;
```

3.**Self-Closing Tags** All elements must be self-closed if they don't have children.

```
const element = <img src="image.png" alt="Description" />;
```

4.**JavaScript Expressions** Use `{}` to include dynamic content.

```
const items = ['React', 'JSX'];
return <ul>{items.map(item => <li key={item}>{item}</li>)}</ul>;
```

5.Inline Styles Style properties are written as objects using camelCase keys.

```
const style = { color: 'blue', fontSize: '20px' };
return <p style={style}>Styled Text</p>;
```

C.JSX Under the Hood

JSX is not plain JavaScript. It is converted into `React.createElement` calls behind the scenes.

JSX:

```
const element = <h1>Hello, world!</h1>;
```

Transpiles to:

```
const element = React.createElement('h1', null, 'Hello, world!');
```

D.JSX with React Components

Functional Component Example:

```
function Greeting() {
  const userName = "John";
  return <h1>Hello, {userName}!</h1>;
}
```

Class Component Example:

```
import React from "react";

class Greeting extends React.Component {
  render() {
    const userName = "John";
    return <h1>Hello, {userName}!</h1>;
  }
}
```

E.Advantages of JSX

1. **Readability**: JSX makes the structure of UI components easy to visualize.
2. **Integration**: You can directly include dynamic data within your UI.
3. **Debugging**: Errors often point directly to the offending JSX line.
4. **Efficiency**: JSX optimizes for performance during the compilation phase.

Components in React

In React, components are the building blocks of a user interface. They allow you to break a complex UI into smaller, reusable, and independent parts. React components can be categorized broadly into functional components and class components.

1.Key Concepts

A.Types of Components in React

1. Functional Components

A simpler way to create components.

Written as JavaScript functions.

Can use React Hooks for state management and lifecycle methods.

Example:

```
function Greeting() {
  return <h1>Hello, world!</h1>;
}

export default Greeting;
```

With Props:

```
function Greeting(props) {
  return <h1>Hello, {props.name}!</h1>;
}

export default Greeting;
```

- **Use:**

```
import Greeting from './Greeting';
<Greeting name="Alice" />;
```

2. Class Components

- Created using ES6 classes.
- Use a `render` method to return JSX.
- Were traditionally used when state and lifecycle methods were needed, but now often replaced by functional components with hooks.

Example:

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

class Greeting extends Component {
  render() {
    return <h1>Hello, {this.props.name}!</h1>;
  }
}

export default Greeting;
```

- **Use:**

```
import Greeting from './Greeting';
<Greeting name="Bob" />;
```

B.Core Concepts of Components

1. Props (Properties)

Props are the mechanism for passing data from a parent to a child component. They are read-only.

Example:

```
function Profile(props) {
  return <h1>{props.name}'s Profile</h1>;
}

<Profile name="Alice" />;
```

2. State

State is data that is managed locally by a component and can change over time.

Example with Functional Component:

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

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

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

Example with Class Component:

```
class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
  }

  increment = () => {
    this.setState({ count: this.state.count + 1 });
  };

  render() {
    return (
      <div>
        <p>Count: {this.state.count}</p>
        <button onClick={this.increment}>Increment</button>
      </div>
    );
  }
}
```

3. Lifecycle Methods (Class Components)

Lifecycle methods allow components to hook into key moments in the rendering process:

componentDidMount: Invoked after the component is added to the DOM.

componentDidUpdate: Invoked after an update.

componentWillUnmount: Invoked before unmounting.

Example:

```
class Timer extends React.Component {
  componentDidMount() {
    this.timerID = setInterval(() => this.tick(), 1000);
  }

  componentWillUnmount() {
    clearInterval(this.timerID);
  }

  render() {
    return <h1>Timer</h1>;
  }
}
```

4. Hooks (Functional Components Only)

React Hooks allow you to use state and lifecycle features in functional components:

useState: Manage state.

useEffect: Side effects like data fetching, timers, etc.

useContext: Access context.

Example: useEffect

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

function Timer() {
  const [seconds, setSeconds] = useState(0);

  useEffect(() => {
```

```
    const interval = setInterval(() => setSeconds(seconds + 1), 1000);
    return () => clearInterval(interval);
  }, [seconds]);

  return <p>Elapsed time: {seconds}s</p>;
}
```

Splitting UI into Components

When designing a React application, split the UI into smaller components:

Example Structure:

```
App
├── Header
├── Content
│   ├── Sidebar
│   ├── MainContent
├── Footer
```

Code Example:

```
function Header() {
  return <header>Header Section</header>;
}

function Footer() {
  return <footer>Footer Section</footer>;
}

function Sidebar() {
  return <aside>Sidebar Content</aside>;
}

function MainContent() {
  return <main>Main Content Area</main>;
}

function App() {
  return (
```

```
    <div>
      <Header />
      <div style={{ display: 'flex' }}>
        <Sidebar />
        <MainContent />
      </div>
      <Footer />
    </div>
  );
}
```

Component Communication

Parent to Child (via Props):

```
function Child({ message }) {
  return <p>{message}</p>;
}

function Parent() {
  return <Child message="Hello from Parent!" />;
}
```

Child to Parent (via Callback Props):

```
function Child({ onClick }) {
  return <button onClick={onClick}>Click Me</button>;
}

function Parent() {
  const handleClick = () => alert('Button clicked!');
  return <Child onClick={handleClick} />;
}
```

Sibling Communication (via State Lifting):

```
function Sibling1({ updateText }) {
  return <button onClick={() => updateText("Hello from
Sibling1")}>Send</button>;
```

```
}

function Sibling2({ text }) {
  return <p>{text}</p>;
}

function Parent() {
  const [text, setText] = useState('');
  return (
    <div>
      <Sibling1 updateText={setText} />
      <Sibling2 text={text} />
    </div>
  );
}
```

Conclusion

Best Practices

- Keep components small and focused on a single task.
- Use functional components for simplicity and hooks for state/lifecycle.
- Use meaningful names for components and props.
- Write reusable components to avoid repetition.

2.Brain teaser :

1. Which of the following is NOT true about JSX?

a) JSX must have one parent element.
b) JSX allows embedding JavaScript expressions inside {}.
c) JSX is a valid JavaScript syntax and can run in browsers without transformation.
d) JSX attributes like class are replaced with className.

---

2. What will the following component render?

```
function Greeting() {
  const user = { name: "Alice" };
  return <h1>Hello, {user}!</h1>;
}
```

a) Hello, Alice!
b) Hello, [object Object]!
c) Syntax Error
d) Hello, {user}!

---

3. What happens if you write JSX with unclosed elements?

```
const element = <div><p>Hello</p><span>World</span></div>;
```

a) The code works fine.
b) SyntaxError: Unclosed element.
c) React automatically closes elements.
d) Browser ignores unclosed elements.

---

4. What is the output of the following code?

```
function App() {
  return (
    <div>
      <h1>JSX</h1>
      <p>The value is: {2 + 2}</p>
    </div>
  );
}
```

a) Error: Expressions not allowed inside JSX.
b) The value is: 2 + 2
c) The value is: 4
d) Undefined behavior

---

5. What will happen if you forget to include `export` in a component file?

a) The application will still work.
b) Importing that component will throw an error.
c) The file won't compile.
d) The file will be treated as an unnamed component.

---

6. What is the default behavior of a React component function with no `return` statement?

a) Returns undefined.
b) Renders nothing but causes a warning.
c) Results in a runtime error.
d) Automatically returns null.

---

7. Given the following code, which will cause an error?

```
const Greeting = () => {
  return <div>Hello</div>;
};
export default Greeting;
```

a) <Greeting />
b) <Greeting>
c) {Greeting}
d) <Greeting></Greeting>

---

8. Which of the following lifecycle methods exists only in class components?

a) useEffect
b) componentDidMount
c) useState
d) render

---

9. In React, what does the `key` prop help with?

a) Managing state inside child components.
b) Allowing components to communicate with each other.
c) Ensuring efficient updates to lists in the virtual DOM.
d) Passing unique CSS classes to elements.

---

10. What will this component render?

```
function App() {
  const element = <h1></h1>;
  return <div>{element}</div>;
}
```

a) A blank .
b) <h1></h1> .
c) Error: Empty tags not allowed.
d) Undefined behavior.

References

Official Documentation:

- React Components and Props:
  - https://legacy.reactjs.org/docs/components-and-props.html
- JSX:
  - https://legacy.reactjs.org/docs/introducing-jsx.html

Interactive Tutorials:

2. **freeCodeCamp - React challenges**
   An excellent hands-on course with exercises to learn React.
3. **W3Schools - React Tutorial**
   A simple and easy-to-understand intrReact with examples.

Online Coding Platforms:

Coding platforms

i.https://www.hackerrank.com/domains/react

ii.https://www.freecodecamp.org/news/react-for-beginners-handbook/