WEB TECHNOLOGIES

JSX-Rendering Of Elements

Prof. Pavan A C

Department of Computer Science and Engineering

Acknowledgement:

Teaching Assistants(Harini B and Chandana MS)

JSX - Rendering Of Elements

What is JSX

JSX stands for JavaScript XML, and it is a special syntax used in React to simplify building user interfaces.

- JSX allows you to write HTML-like code directly inside JavaScript, enabling you to create UI components more efficiently.
- Although JSX looks like regular HTML, it’s actually a syntax extension for JavaScript.

Although there are other ways to write components, most React developers prefer the conciseness of JSX, and most codebases use it.

Example :

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

JSX - Rendering Of Elements

How JSX Works

- JSX code is compiled/transformed into React.createElement() calls by tools like Babel.
- Compiled JSX creates plain JavaScript objects called React elements.

```
const element = <h1>Hello, world!</h1>;
// tools like Babel Compiles to:
const element = React.createElement('h1', null, 'Hello, world!');
```

The JSX code <h1>{message}</h1> will be transformed into JavaScript by Babel. Then react will then create a virtual DOM element for the <h1> tag with the text inside. and this virtual DOM is then used to update the actual browser DOM, displaying "Hello, world!" on the screen.

JSX - Rendering Of Elements

Converting HTML to JSX

- React components use JSX to define UI markup inside JavaScript.
- HTML looks similar but JSX has stricter syntax rules.
- Direct copy-pasting HTML into JSX often causes errors

Rules for converting HTML to JSX

- **Rule 1 – Single Root Element**
- **Rule 2 – Close All Tags**
- **Rule 3 – Use camelCase for Attributes**

JSX compiles to JavaScript objects; attribute names become object keys.

JavaScript has naming restrictions, hence camelCase.

Single root element needed for consistent element representation in React.

JSX - Rendering Of Elements

Converting HTML to JSX

Single Root Element

BEFORE

```
<h2>Tasks</h2>
<ul>
  <li>Debugging code</li>
  <li>Writing documentation</li>
  <li>Presenting talks</li>
</ul>
```

AFTER

```
return (
 <>
  <h2>Tasks</h2>
   <ul>
    <li>Debugging code</li>
    <li>Writing documentation</li>
    <li>Presenting talks</li>
   </ul>
  </>
);
```

JSX must return a single parent element.

Wrap siblings inside a <div>, <section>, or React Fragment <>...</>.

JSX - Rendering Of Elements

Converting HTML to JSX

Attribute Names

BEFORE

```
    <img     src="image.jpg"     alt="Alt"
class="portrait" />
<ul>
  <li>Presenting talks</li>
</ul>
```

AFTER

```
return (
 <>
<img     src="image.jpg"     alt="Alt"
className="portrait" />
   <ul>
    <li>Presenting talks</li>
   </ul>
  </>
);
```

JSX attributes are **camelCase**, not HTML attribute names.

- Use className not class
- Use htmlFor not for
- Use tabIndex instead of tabindex

JSX - Rendering Of Elements

Embedding Expressions

In the example below, we declare a variable called name and then use it inside JSX by wrapping it in curly braces:

```
const name = 'Josh Perez';
const element = <h1>Hello, {name}</h1>;
```

You can put any valid JavaScript expression inside the curly braces in JSX. After compilation, JSX expressions become regular JavaScript function calls and evaluate to JavaScript objects.

```
function getGreeting(user) {
  if (user) {
    return <h1>Hello, {formatName(user)} !</h1>;
  }
  return <h1>Hello, Stranger.</h1>;
}
```

This means that you can use JSX inside of if statements and for loops, assign it to variables, accept it as arguments, and return it from functions:

JSX - Rendering Of Elements

Embedding Expressions

```
const user = { name: 'John' };
const element = <h1>Welcome, {user.name.toUpperCase()}!</h1>;
```

```
const isLoggedIn = true;
const message = <h1>{isLoggedIn ? 'Logout' : 'Login'}</h1>;
```

```
const numbers = [1, 2, 3];
const listItems = numbers.map(num => <li key={num.toString()}>{num}</li>);
const element = <ul>{listItems}</ul>;
```

Simple Rendering Example

Render an element to DOM using React 18+ API:

```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<h1>Hello React!</h1>);
```

JSX - Rendering Of Elements

JSX Rendering

JSX Spread Attributes

Spread operator to pass props dynamically:

```
const props = { name: 'Sara', age: 30 };
const element = <User {...props} />;
```

Use function variables:

```
function renderContent(isLoading) {
  if (isLoading) return <LoadingSpinner />;
  return <Content />;
}
```

Handling Events in JSX

React uses camelCase event naming:

```
<button onClick={handleClick}>Click</button>
```

JSX - Rendering Of Elements

JSX Rendering - Example

```
const element = (
  <div>
    <h1>Hello, JSX!</h1>
    <p>2 + 2 = {2 + 2}</p>
  </div>
);

ReactDOM.createRoot(document.getElementById("root")).render(element);
```

JSX - Rendering Of Elements

JSX Rendering - Example

```
const element = (
  <div>
    <h1>My Favorite Fruits</h1>
    <ul>
      <li>Apple</li>
      <li>Mango</li>
      <li>Banana</li>
    </ul>
  </div>
);

ReactDOM.createRoot(document.getElementById("root")).render(element);
```

JSX - Rendering Of Elements

JSX Rendering

JSX and Forms

React handles forms via components and controlled inputs:

```
function MyForm() {
  const [name, setName] = React.useState('');
  return (
    <input type="text" value={name} onChange={e =>
setName(e.target.value)} />
  );
}
```

Comments in JSX are written with {/* */}

JSX - Rendering Of Elements

JSX Rendering

Use spread syntax to pass all props concisely:

```
const props = { multiple: true, disabled: false };
<input type="checkbox" {...props} />;
```

React controls form inputs via state:

```
function MyForm() {
  const [value, setValue] = React.useState('');
  return <input value={value} onChange={e => setValue(e.target.value)}
/>;
}
```

JSX - Rendering Of Elements

JSX Rendering

Events use camelCase: onClick, onChange.
Pass handler functions, not strings:

```
<button onClick={() => alert('Clicked!')}>Click Me</button>
```

- Always provide keys on lists for minimal re-rendering.
- Avoid anonymous functions inline to prevent unnecessary re-creation.
- Break UI into reusable components for maintainability.

JSX - Rendering Of Elements

MCQ

1. What is the correct way to set a CSS class in JSX?

   A) `<div class="container">`

   B) `<div className="container">`

   C) `<div classname="container">`

   D) `<div class-name="container">`

   Answer: B

2. How would you embed the JavaScript expression `2 + 2` inside a JSX element?

   A) `<p>2 + 2</p>`

   B) `<p>{2 + 2}</p>`

   C) `<p>{{2 + 2}}</p>`

   D) `<p>{(2 + 2)}</p>`

   Answer: B (also D is valid)

JSX - Rendering Of Elements

MCQ

Which of the following is a valid self-closing JSX tag?

A) `<img>`

B) `<img />`

C) `<img></img>`

D) Both B and C

Answer: D

How do you embed a JavaScript expression in JSX?

- A) { }
- B) < >
- C) ( )

Answer: A

JSX - Rendering Of Elements

MCQ

What will be the output of the following JSX code?

```
const show = false;
```

```
const element = <div>{show && <p>Hello, World!</p>}</div>;
```

A) `<div><p>Hello, World!</p></div>`
B) `<div></div>`
C) `<div>false</div>`
D) Syntax error

Answer: B) `<div></div>`

Since show is false, the logical AND (&&) expression will short-circuit and not render the <p> element, resulting in an empty <div>.

THANK YOU

Prof. Pavan A C

Department of Computer Science and Engineering