SD545-Web Application Development 1: Assignment-2

1. **How does JSX differ from HTML?**
   Key Differences Between JSX and HTML

   JavaScript Integration:

   JSX: Allows embedding JavaScript expressions within curly braces {} directly in the markup.
   HTML: Typically, JavaScript is separate from HTML and is added using <script> tags.
   Component-Based vs. Tag-Based:

   JSX: Encourages a component-based structure where UI elements are created as reusable components.
   HTML: Utilizes a tag-based structure where elements are defined using HTML tags.
   Event Handling:

   JSX: Event handling is defined using camelCase, like onClick for a click event.
   HTML: Event attributes are typically in lowercase, like onclick for a click event.
   Conditional Rendering:

   JSX: Supports conditional rendering directly within the JSX code using JavaScript constructs like if statements or ternary operators.
   HTML: Conditional rendering often involves using JavaScript to manipulate the DOM.
   Styling:

   JSX: Often uses JavaScript-based styling solutions like CSS-in-JS or CSS modules for scoped styling.
   HTML: Styling is commonly done using CSS in separate files or within <style> tags in HTML.
   Browser Compatibility:

   JSX: Requires transpilation using tools like Babel to convert JSX code into JavaScript that browsers can understand.
   HTML: HTML is natively supported by all browsers.
   Accessibility and SEO:

   JSX: Developers need to pay extra attention to accessibility features and SEO optimization when using JSX in web applications.
   HTML: HTML has established practices for accessibility and SEO, making it easier to follow best practices.
   Tooling and Libraries:

   JSX: Often used in conjunction with JavaScript libraries and frameworks like React, Vue.js, or Angular.
   HTML: Can be used in traditional web development without relying on specific libraries or frameworks.
   Use Cases

Use Cases for JSX:

React and Other JavaScript Frameworks
Single-Page Applications (SPAs)
Complex User Interfaces
Conditional Rendering
Integration with APIs
Use Cases for HTML:

Static Web Pages
SEO and Accessibility
Email Templates
Embedding in Documents
Legacy Systems
Conclusion

Understanding the differences between JSX and HTML is crucial for modern web developers. While JSX offers dynamic rendering and tight integration with JavaScript, HTML remains essential for static content and compatibility.

The choice between JSX and HTML depends on project requirements and the technology stack. Both have strengths and use cases, and developers should make informed choices based on project needs to create effective and user-friendly web experiences.

2. **Why is JSX used in React?**
JSX is used in React.js because it aligns with React's component-based architecture, making UI development more declarative and visually intuitive. Benefits include improved readability, performance, and safety against injection attacks. JSX has revolutionized React development by abstracting away complex JavaScript logic, making it more intuitive and efficient for developers.

3. Can you embed JavaScript expressions in JSX? If so, how?

Yes, when you need to include a JavaScript expression inside your JSX code, you need to wrap it in curly braces {}. This can be used for anything from displaying dynamic data to conditionally rendering components.

4. How do you write comments in JSX?

Comments are enclosed within curly braces {}.

They are written using the {/* */} syntax, similar to block comments in JavaScript.

Comments can be placed anywhere within the JSX code to explain specific parts of the UI or logic.

Comments are helpful for documenting code and are not rendered in the final output, so they won't affect the appearance of the web page.

The provided example demonstrates these points well by embedding comments within JSX elements and explaining their purpose. Overall, it provides a clear and concise understanding of how to use comments in JSX.

Example:

```
import React from 'react';


function App() {
  return (
    <div>
      {/* This is a comment in JSX */}
      <h1>Hello, World!</h1>
      {/* Comments can be written anywhere within JSX */}
      <p>This is a paragraph.</p>
      {/* They help explain the code and are not rendered in the output */}
    </div>
  );
}


export default App;
```

5. **Explain the significance of curly braces {} in JSX.**

In summary, curly braces {} play a crucial role in JSX by enabling the embedding of JavaScript expressions, defining dynamic attribute values, and executing JavaScript statements within JSX code blocks, thereby facilitating dynamic content rendering and logic execution in React components.

6. Can JSX be directly rendered to the DOM?

No, JSX cannot be directly rendered to the DOM. JSX is a syntax extension for JavaScript, used with libraries like React to describe what the UI should look like.

JSX needs to be transpiled into regular JavaScript before it can be rendered to the DOM. Typically, a build process using tools like Babel is used to convert JSX into JavaScript that the browser can understand. This transpiled JavaScript code contains React.createElement calls, which create React elements representing the structure of the UI, and these elements are then rendered to the DOM using ReactDOM or similar libraries.

In summary, JSX is a convenient syntax for writing UI components in React, but it needs to be transpiled into regular JavaScript before it can be rendered to the DOM.

7.**What is the purpose of Babel in relation to JSX?**

Babel transpiles JSX code into plain JavaScript so that it can be understood by browsers. It converts JSX syntax, like Hello , into JavaScript function calls, such as React.createElement('div', null, 'Hello'), which represent the structure of the UI. This enables developers to write JSX for React components while ensuring browser compatibility.

8. Practice JSX syntax listed on the slide with code examples

1. Don't use quote when define virtual DOM.

```
'<h1>React introduction</h1>';//it is considered as a string it rule one violated
```

2. JavaScript Expressions: To insert dynamic data, variables, or expressions into the rendered content, curly braces {} inside JSX elements. You cannot use statements.

```
//return<h1> {titile}</h1>
//return<h1> {getName()}</h1>;
  //return<h1> {const i=20 }</h1>;//do not use statements
```

3.Class Name: Use the className attribute to set the CSS class of an element.

```
return <h1 className ="title">{getName()}</h1>;//correct

return <h1 class="title">{getName()}</h1>;// not class it should be className otherwise it is an Error: invalid DOM class property
```

4.Inline Style: Use the style attribute to set inline style with syntax {{key:value}}.

```
const yellow='blue'
  return <h1 className ="title" style={{color:yellow,fontSize:'50px'}}>{getName()}</h1>;//for the value use quotes otherwise it will be an error
```

5. Only One Top-Level Element: all elements within a JSX expression must be wrapped in a single parent element.

```
return <div>first div</div><div>second div</div> // It should be one parent element otherwise it is an error
```

```
return <div><div>first div</div><div>second div</div></div>//
correct.
return <><div>first div</div><div>second div</div></> //
correct. (void element)
```

6. Must have closing tag. Be aware of Self-Closing Tags.

```
<div>
        <input type='text'>// Error needed closing tag
        <input type='text'/> // correct has closing tag
    </div>
```

7. The first letter of a tag

1) If it's lower case, it'll be translated to HTML element with the same tag name. If couldn't find in HTML, throw error but still display.

```
<foo></foo>  //foo is not unrecognized
```

2) If it's upper case, react will render the component, if no component, throw error.

```
<Foo></Foo>//Foo is undefined
If we define a function: function Foo(){
    return <h1>Foo Component</h1>
}
It works!
```

8. Comments: use syntax below, must inside the top-level element.

```
<div>First Div</div>
    {/*this is a comment */}
    <div>Second Div</div>
```