Assignment 03 - Laying the foundation

1. What is JSX? Super power of JSX?

- JSX stands for **JavaScript Syntax eXtension**.
- JSX allows us to write **HTML-like** elements in JavaScript and place them in the DOM **without any createElement()** and/or **appendChild()** methods.
- JSX converts HTML tags into react elements.
- It is not mandatory to use JSX, but JSX makes it easier to write React applications.

Advantages:

- If you are familiar with HTML, it is quite easy to use JSX in React.
- JSX makes the code simpler and readable.
- JSX allows React to show more useful errors and warning messages.
  JSX is faster than normal JavaScript, as it performs optimization when translating to regular JavaScript.

2. The role of the “type” attribute in Script tag

- The type attribute specifies the type of the script.
- The type attribute identifies the content between <script> and </script> tags
- Ex:
  - <script type="application/javascript">
    document.getElementById("demo").innerHTML = "Hello JavaScript!";
    </script>
  - <script type="module" src="/App.js" ></script>

- <script type="application/json">
  {
  “name”:”suganya”,
  “age”:28
  }
  </script>

3. {TitleComponent}vs{<TitleComponent/>}vs{<TitleComponent><TitleComponent/>} in JSX.

- {TitleComponent} —> **React variable.**
- {<TitleComponent/>}, {<TitleComponent><TitleComponent/>} —> **React Components.**

4. What is Component Composition in React?

- Component composition is the name for passing **components as props to other components,** thus creating new components with other components.
- **Example:**

```
Const Parent = () =>{
return(
    <h1>Hello React!</h1>
    <Child/>                    // component composition
    )
}
```

5. What are the roles of Babel and Parcel in JSX?

- **Bable in JSX:**
  - JSX should not be implemented directly by browsers, but instead requires **a compiler to transform it into ECMAScript.** This is where Babel comes in.
  - **Babel acts as this compiler** allowing us to leverage all the benefits of JSX while building React components.
  - JS Engine does not understand JSX. That only understands ES6 EcmaScript. The code will be **transpiled** before it reaches the JS Engine by **Babel**.
  - And also Bable is responsible for converting,

```
React.createElement(“”h1, {id=”headId”}, “Hello World!”)
                        Into
<h1 id=”headId”>Hello World!</h1>
```

- **Parcel in JSX:**
  - Parcel supports JSX automatically when it detects we are using React. If we’re using React 17 or later, it also automatically enables the modern JSX transform, which means we **don't even need to import React for JSX to work**, as we can see in the below example.
  - **App.js**

```
export const App= () =>{
return(
        <h1>Hello React!</h1>
        )
}
```

5. What are Components in Reactjs?

- A component is an independent **reusable bit of code which divides the UI into smaller pieces.**
- They are like the javascript functions, which **return the HTML elements.**
- There are two types of components:

1. **Class Components**

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

2. **Functional Components**

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

6. Difference between React Elements and Components.

- **React Elements:**
  - ➢ It is a simple **object** that describes a DOM node and its attributes or properties you can say. It is an immutable description object and you can not apply any methods on it.

    Ex: <button class="blue">Cancel</button>

- **React Components:**
  - ➢ It is a function or class that accepts an input and returns a React element. It has to keep references to its DOM nodes and to the instances of the child components.

    Ex: const SignIn = () => (

    Sign In
    <button>Continue</button>
    <button color='blue'>Cancel</button>

    );

7. Can we have multiple roots and multiple rendering?

- **YES**. It is possible to have multiple roots and multiple rendering in the same file in React.

```
App.js
// react element
const headTag = (
<h1 className="headClass">Hello from JSX Element</h1>
```

```
)
// react component
const Title = () => (
    <h1 className="headClass">Vanakkam React</h1>
)
// variable
const text="Functional component";
// functional component
const Head = () => {
    return(
        <div>
            <h1>Hello from Head component</h1>
        </div>
    )
}
const root=ReactDOM.createRoot(document.getElementById('root'))
const root1=ReactDOM.createRoot(document.getElementById('root1'))
const root2=ReactDOM.createRoot(document.getElementById('root2'))
root.render(headTag)
root1.render(<Title/>)
root2.render(text)
```

Index.html

```
<body>

    <div id="root">
```

```
        <h1>Not Rendered</h1>
    </div>
    <div id="root1">
        <h1>Not Rendered 1</h1>
    </div>
    <div id="root2">
        <h1>Not Rendered 2</h1>
    </div>
    <script type="module" src="./App.js"></script>
</body>
```

8. What are Cross-Site Scripting (XSS) attacks?

- **Cross-Site Scripting (XSS)** attacks are a type of injection, in which **malicious scripts are injected** into otherwise benign and trusted websites.
- XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user.
- When the DOM is mutated, it is easy to inject the malicious code to the user's browser.
- In React the JSX prevents the site from XSS attacks. Because in JSX the data will be wrapped in between {}, so it will validate any data before entering into the functionality.

9. Multiple ways to call the components:

EX:

```
Const Title = () => (
    <h1>Hello World!</h1>
)

Const Head = () =>(
    <Title />
    <Title></Title>
    {Title()}
)
```

10. Describe fragments in React.

- In React all the elements should be wrapped within the **single parent** element.
- Ex:

```
Const Head = () =>{
    return(
        <h1>Hello World!</h1>
        <h1>Hello World!</h1>                    // not allowed
    )
}

Const Head = () =>{
    return(
        <div>
            <h1>Hello World!</h1>
            <h1>Hello World!</h1>
```

```
            </div>                                  // allowed
        )
    }
```

❖ But in order to avoid the extra unnecessary Div tag we will use the React fragments as follows

```
    Const Head = () =>{
        return(
            <React.Fragment>
                <h1>Hello World!</h1>
                <h1>Hello World!</h1>
            </React.Fragment>                       // allowed
        )
    }

    Const Head = () =>{
        return(
            <>
                <h1>Hello World!</h1>
                <h1>Hello World!</h1>
            </>                                     // allowed
        )
    }
```