Section 3 Summary - React Basics Concepts

Creating component

- Component is Reusable piece of code that used to define certain part of user interface.
- Think of a React component as a small, self-contained piece of code that describes how a part of a website or application should look and behave.
- Here is the basic structure of react component:

```
import React from "react";

const Card = () => {
    return <div>Card Component</div>;
};

export default Card;
```

JSX and Babel

- JSX stands for "JavaScript XML" and it is code in which we can write HTML and JavaScript code together. This code looks very similar to the HTML markup.

```
import React from "react";

const Card = () => {
    const name = "Code Bless You!";
    return <h1>Name: {name}</h1>;  // This is jsx code
};

export default Card;
```

- Now this JSX code, browser can not understand, so we have to covert this code into vanilla JavaScript code and for that React use babel library.
- You can paste this code in to babel.io website and you can see it's vanilla JavaScript code.
- So we can easily write JSX code in our file and babel will convert that code in to JavaScript code that browsers can understand.
- Article - https://reactjs.org/blog/2020/09/22/introducing-the-new-jsx-transform.html

Adding multiple elements

- So for adding multiple elements in react component, we have to wrap them with one parent element. Because we know in babel our code convert in React JSX function and that function only add one element in its first argument.
- There are 2 solutions for that:
    1. We can wrap our elements with any one parent element.

```
import React from "react";
```

```jsx

const Card = () => {
    return (
        <div>
            <h1>This is Card Component</h1>
            <button>Add new task</button>
        </div>
    );
};

export default Card;
```

2. We can use React.Fragement component for wrapping them.

```jsx
import React from "react";

const Card = () => {
    return (
        <React.Fragment>
            <h1>This is Card Component</h1>
            <button>Add new task</button>
        </React.Fragment>
    );
};

export default Card;
```

or we can use this also

```jsx
import React from "react";

const Card = () => {
    return (
        <>
            <h1>This is Card Component</h1>
            <button>Add new task</button>
        </>
    );
};

export default Card;
```

Adding JavaScript expression in to JSX

- Now to add JavaScript expression in to JSX, we just need to add curly brackets and inside this we can add JavaScript expression.

```jsx
import React from "react";

const Card = () => {
    const tasks = 5;
    return (
        <>
            <h1>Total Tasks: {tasks}</h1>
            <button>Add new task</button>
        </>
    );
};

export default Card;
```

Adding attributes and events in JSX

- In JSX, we can add attribute/events same as we add attributes/events in HTML. But some attributes are different.
- For example, for adding class to our element, we have to use className attribute and also for defining inline style, we have to pass style object in style attribute.

State

- When we change our normal variable value, that value change will not happen on our web page. So for reflecting that change on our DOM, we have to make our variable as state variable.
- In simple words, if we define our variable as state variable, then react will watch that variable and if its value change, then react will reflect that change immediately.
- So for defining state variable we have to use useState hook.

What are hooks?

- hooks in React are functions that allow us to add state and other React features to functional components.
- Before hooks were introduced, state and other features could only be used in class components. However, with hooks, you can use these features directly in functional components, making them more powerful and versatile.

useState Hook

- useState is used to create state variables in functional components. So to use the useState hook we need to first import that and use it inside the function component.
- We pass any type of data - Like Boolean, number, text, object, array anything. This useState hook returns array with two items. The first one is its current value and the second one is the function for updating that value.

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

const Card = () => {
    const [count, setCount] = useState(5);
    const handleClick = () => {
        setCount(count + 1);
    };
    return (
        <>
            <h1>Total Counts: {count}</h1>
            <button onClick={handleClick}>Add new task</button>
        </>
    );
};

export default Card;
```

- We can also use useState hook, for handling inputs. Just we have to pass our set function in onChange event.

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

const Card = () => {
    const [input, setInput] = useState("");
    const handleChange = (e) => {
        setInput(e.target.value);
    };
    return (
        <>
            <input type='text' onChange={handleChange} />
            <h1>Input: {input}</h1>
        </>
    );
};

export default Card;
```

Mapping list items

- As we see in Section 2, we will use map method to display the list of items in react. This map method is very useful.
- Always remember to add key attribute for returning element and pass unique value in it. That will help react to quickly identified which element gets changed.

```
import React from "react";

const Card = () => {
    const products = ["Product1", "Product2", "Product3"];
    return (
        <>
            <ul>
                {products.map((product) => (
                    <li key={product}>{product}</li>
                ))}
            </ul>
        </>
    );
};

export default Card;
```