Tutorial No. 08

Title: Study of React page setup, React Elements, React components

Batch: A1 **Roll No.: 16010422013** **Tutorial No:9**

Aim: To showcase the functionality of react components , hooks and props.

Resources needed: VS Code, Node.js ,Web Browser , npm

Theory:

React JS

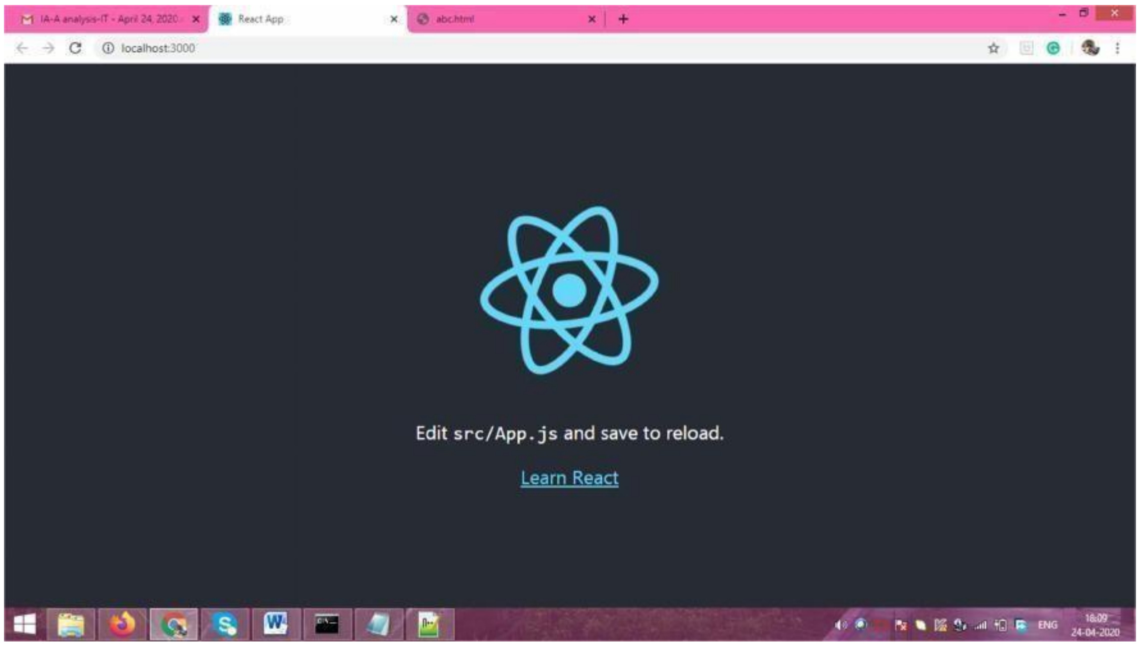
React is a declarative, efficient, and flexible JavaScript library for building user interfaces.
It lets you compose complex UIs from small and isolated pieces of code called components”. We use components to tell React what we want to see on the screen. When our data changes, React will efficiently update and re-render our components.
React does not manipulate the browser's DOM directly. Instead, React creates a virtual DOM in memory, where it does all the necessary manipulating, before making the changes in the browser DOM.

Setting up a React Environment

First of all you need to download NodeJs for your operating system version.
https://nodejs.org/en/download/
npm(node packet manager once you download NodeJs
Then follow the following commands:

1) run the command “C:\Users\Your Name>npm install -g create-react-app”
2) run the command to create an application name helloworld
“C:\Users\Your Name>npx create-react-app Helloworld”
3) run this command to get to the current directory “C:\Users\Your Name>cd helloworld”
4) run this command to start the react application “C:\Users\Your Name\helloworld >npm start”

A new browser window will pop up with your newly created React App! If not, open your browser and type localhost:3000 in the address bar.



Edit src/App.js and save to reload.

Modify the React Application

Look in the helloworld directory, and you will find a src folder. Inside the src folder there is a file called App.js, open it and make changes to any HTML part.
You will be able to see the change on the newly opened browser

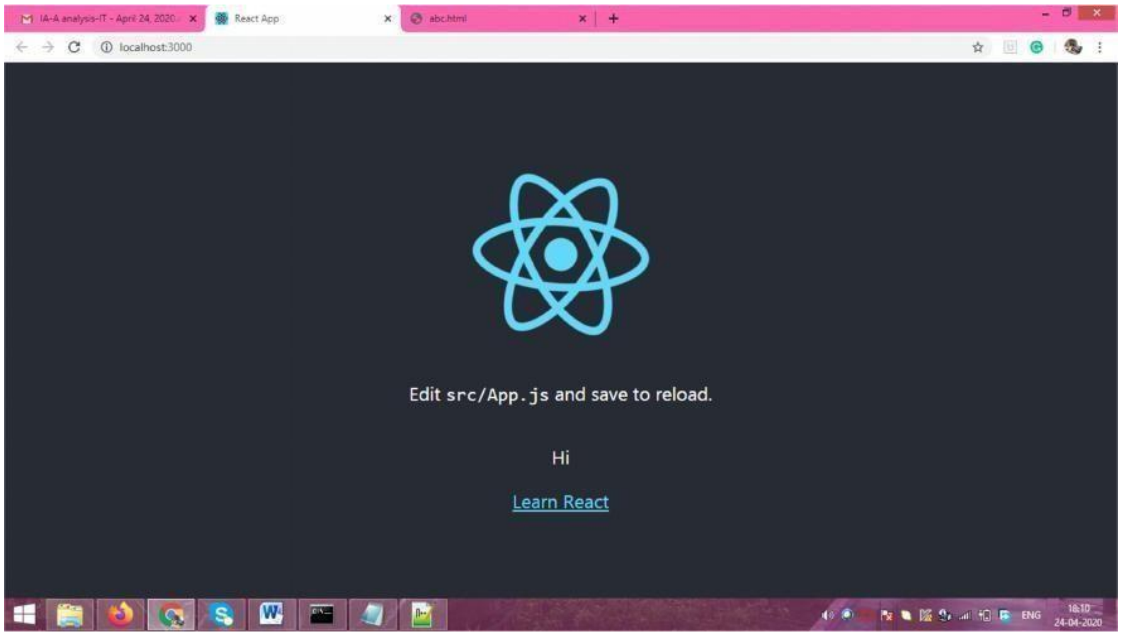
App.js File

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';

class App extends Component {
  render() {
    return (
      <div className="App">
        <header className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <p>
            Edit <code>src/App.js</code> and save to reload.
          </p>
<p> Hi </p>
          <a
            className="App-link"
            href="https://reactjs.org"
            target="_blank"
            rel="noopener noreferrer"
          >
            Learn React
          </a>
        </header>
      </div>
    );
  }
}
```

export default App;

in the file “Hi” is added inside paragraph tag.

The result browser screen is as follows



Edit src/App.js and save to reload.

Hi

Learn React

Activity:

- Explain React Components and how it is created and used.
- Explain Hooks and show an example for the same .

Results:

React components are the building blocks of React applications, encapsulating both UI elements and logic.
They can be created as either functional components or class components.

1. Functional Components: Created as JavaScript functions that return JSX (JavaScript XML) to define the UI structure. Here's an example:

```
import React from 'react';
const MyComponent = () =>
{ return ( <div> <h1>Hello, World!</h1> </div> );
};
export default MyComponent;
```

2. Class Components: Created as ES6 classes that extend React.Component and define a render() method to return JSX.

```
 import React, { Component } from 'react';
class MyComponent extends Component {
render() {
    return (
      <div>
        <h1>Hello, World!</h1>
      </div>
    );
  }
}

export default MyComponent;
```

Once a component is created, it can be used by importing it into other components or files and including it within JSX. For example:

```
import React from 'react';
import MyComponent from './MyComponent';
const App = () => {
  return (
    <div>
      <h1>My React App</h1>
      <MyComponent />
    </div>
  );
};
export default App;
```

In this way, React components provide a modular and reusable approach to building user interfaces, allowing developers to compose complex UIs from smaller, self-contained pieces.

Hooks are functions provided by React that allow functional components to use state and other React features without writing a class. They enable developers to reuse stateful logic across components. Here's an example of using the useState hook to add state to a functional component:

```
import React, { useState } from 'react';
const Counter = () => {
  // Define state using the useState hook
  const [count, setCount] = useState(0);

  // Event handler to increment count
  const increment = () => {
  setCount(count + 1);
  };

  return (
    <div>
      <h2>Counter: {count}</h2>
      <button onClick={increment}>Increment</button>
    </div>
  );
};

export default Counter;
```

In this example:

- We import the useState hook from React.
- Inside the Counter component, we call useState(0) to initialize a state variable named count with an initial value of 0. The useState hook returns an array with two elements: the current state value (count) and a function (setCount) to update that state value.
- We use destructuring assignment to assign these values to count and setCount.
- When the button is clicked, the increment function is called, which updates the count state using setCount.
- The component re-renders with the updated state value, and the new count is displayed.

Hooks like useState allow functional components to manage state and lifecycle events, making them more powerful and flexible compared to class components.

Code –

App.jsx

```jsx
import React, { useState } from 'react';
import './App.css'; // Import CSS file for styling

const Counter = () => {
  // Define state using the useState hook
  const [count, setCount] = useState(0);

  // Event handler to increment count
  const increment = () => {
    setCount(count + 1);
  };

  // Event handler to decrement count
  const decrement = () => {
    setCount(count - 1);
  };

  return (
    <div className="counter-container">
      <h2 className="counter-value">Counter: {count}</h2>
      <div className="button-container">
        <button className="increment-button" onClick={increment}>
          Increment
        </button>
        <button className="decrement-button" onClick={decrement}>
          Decrement
        </button>
      </div>
    </div>
  );
};

export default Counter;
```

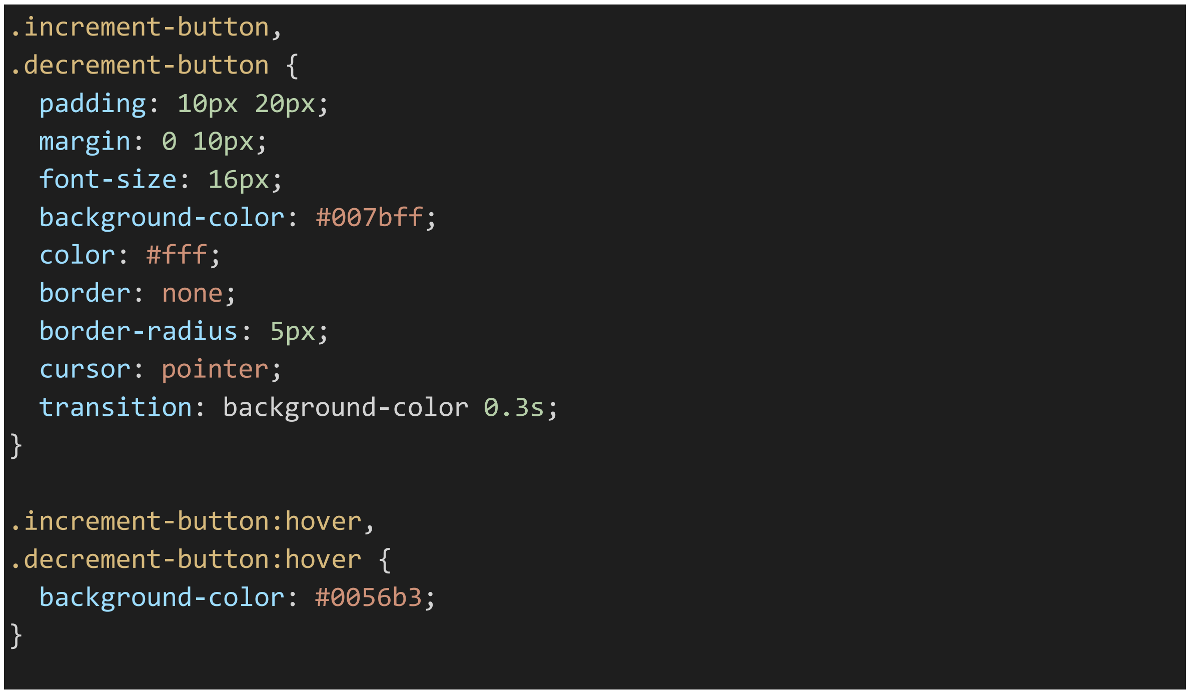
Index.css

```css
:root {
  font-family: Inter, system-ui, Avenir, Helvetica, Arial, sans-serif;
  line-height: 1.5;
  font-weight: 400;
  color-scheme: light dark;
  color: rgb(255, 255, 255);
  background-color: #242424;
  font-synthesis: none;
  text-rendering: optimizeLegibility;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

a {
  font-weight: 500;
  color: #050617;
  text-decoration: inherit;
}
a:hover {
  color: #535bf2;
}

body {
  margin: 0;
  display: flex;
  justify-content: center; /* Center the content horizontally */
  align-items: center; /* Center the content vertically */
  min-width: 320px;
  min-height: 100vh;
  background-color: rgb(219, 139, 197); /* Set background color */
}

.tic-tac-toe {
  text-align: center;
}

.board {
  display: grid;
  grid-template-columns: repeat(3, 100px);
  grid-template-rows: repeat(3, 100px);
  gap: 5px;
}

.square {
  width: 100%;
  height: 100%;
  font-size: 2rem;
  display: flex;
  align-items: center;
  justify-content: center;
  border: 1px solid #000; /* Set border color to black */
  background-color: #8ae5fb;
}

.x,
.o {
  font-size: 3rem;
```

```css
  line-height: 0.7;
}

.status {
  font-size: 1.5rem;
  margin-bottom: 20px;
  color: #faf8f8; /* Set text color */
}

button {
  border-radius: 8px;
  border: 1px solid transparent;
  padding: 0.6em 1.2em;
  font-size: 1em;
  font-weight: 500;
  font-family: inherit;
  background-color: #fffbfb;
  cursor: pointer;
  transition: border-color 0.25s;
  color: #120101; /* Set text color to white */
}
button:hover {
  border-color: #646cff;
}
button:focus,
button:focus-visible {
  outline: 4px auto -webkit-focus-ring-color;
}
```

App.css

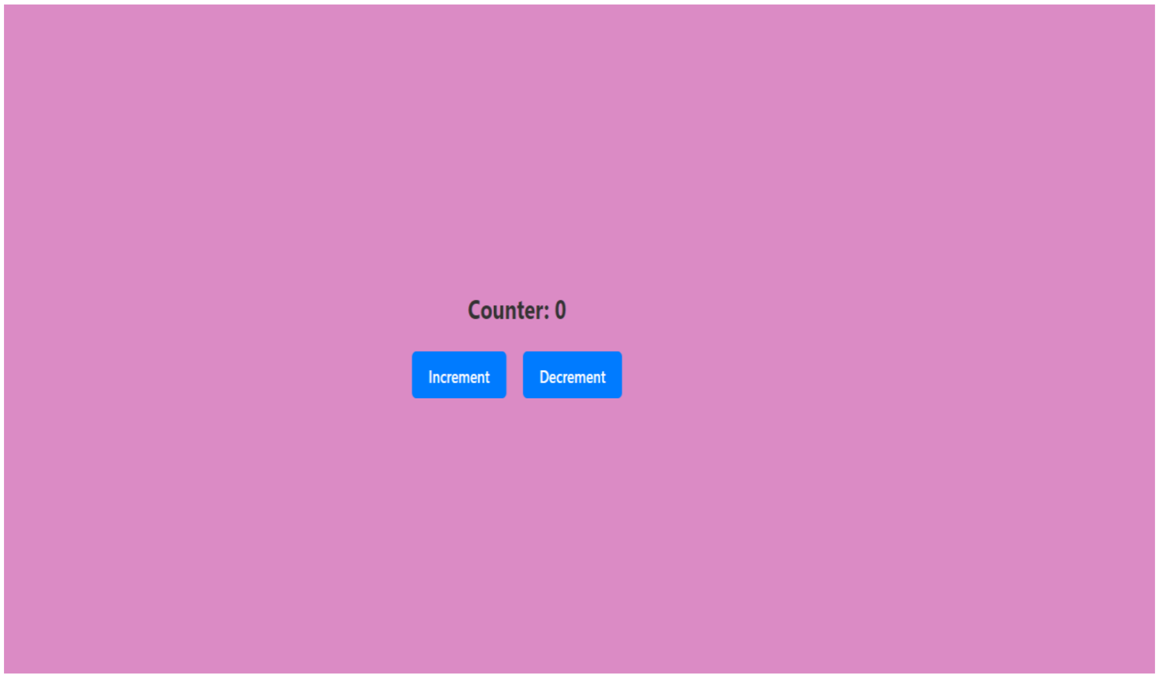
```css
.counter-container {
  text-align: center;
}

.counter-value {
  font-size: 24px;
  margin-bottom: 20px;
  color: #333;
}

.button-container {
  display: flex;
  justify-content: center;
}
```



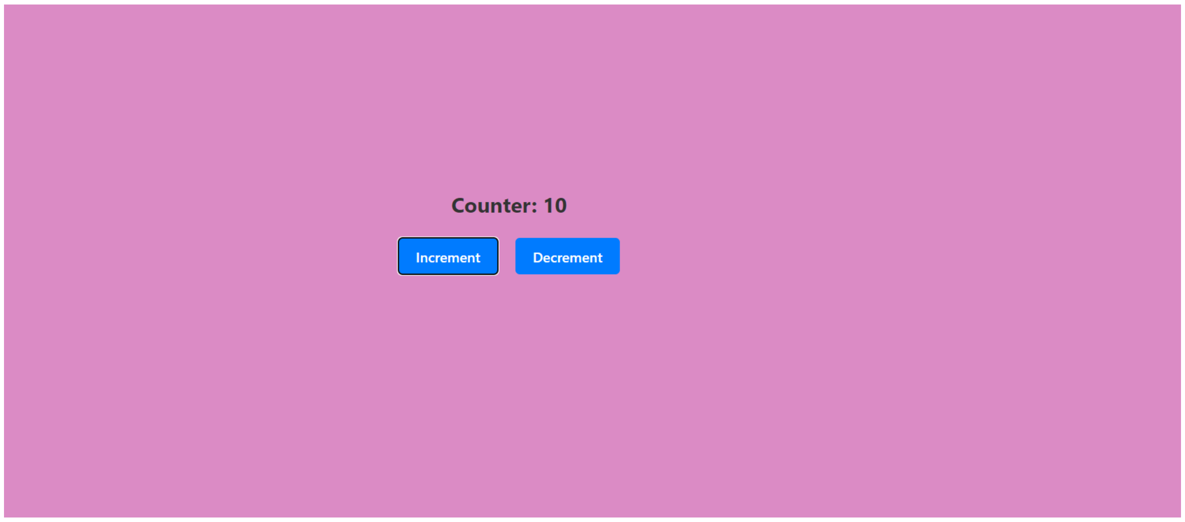
```
.increment-button,
.decrement-button {
  padding: 10px 20px;
  margin: 0 10px;
  font-size: 16px;
  background-color: #007bff;
  color: #fff;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  transition: background-color 0.3s;
}

.increment-button:hover,
.decrement-button:hover {
  background-color: #0056b3;
}
```

Output -

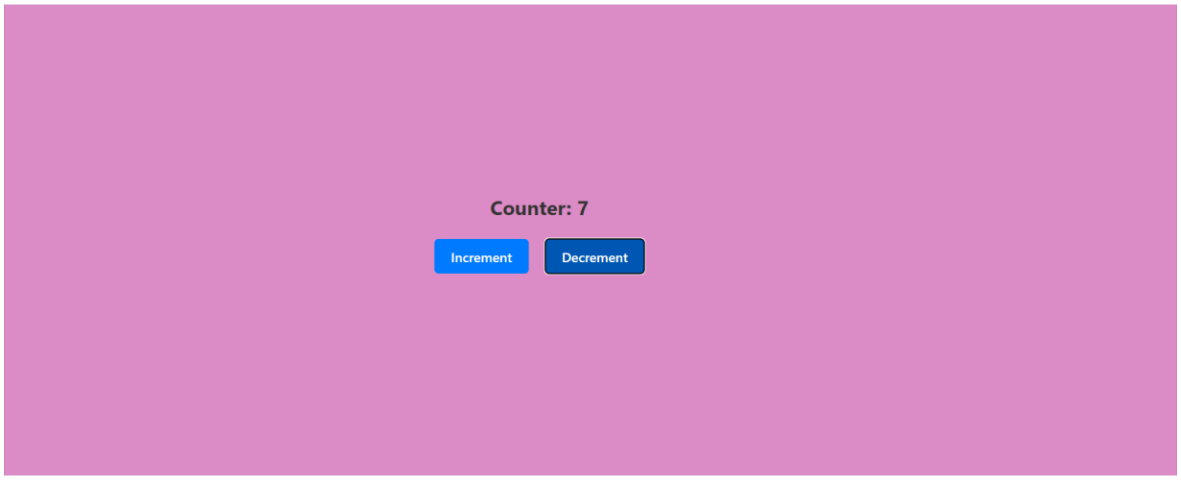
Initial:



Increment :



Decrement:



Outcomes:

CO 4: Implement web application using React JS, JSON and CBOR

Conclusion: (Conclusion to be based on the outcomes achieved)

Through the experiment, we understood that the JSX code for the Counter component showcases the use of useState hook to manage state and event handlers for incrementing and decrementing the count. Additionally, the CSS file enhances the visual presentation with styled buttons and a centered counter display. React Components are the fundamental building blocks encapsulating UI and logic, while Hooks enable functional components to utilize state and other React features efficiently, fostering modular and reusable code.

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of faculty in-charge with date

References:

Books/ Journals/ Websites:

- http://www.w3schools.com
- https://react.dev/learn/thinking-in-react