Batch: A1	**Roll No.: 16010123012**
Experiment / assignment / tutorial No.: 03	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

Title: Implementation of React Fundamentals.

AIM: To Implement the Basic operations of React js

Problem Definition:

-Demonstrate the

- React Fundamentals
- Function Component
- Styling / Bootstrap
- React JSX
- Expressions in JSX
- React Props

Resources used: https://react.dev/

Expected OUTCOME of Experiment:
CO1: Build full stack applications in JavaScript using the MERN technologies.
CO5: Deploy MERN applications to cloud platforms like Heroku or AWS and collaborate using GitHub version control.

Books/ Journals/ Websites referred:

1. Shelly Powers Learning Node O' Reilly 2nd Edition, 2016.

Pre Lab/ Prior Concepts:

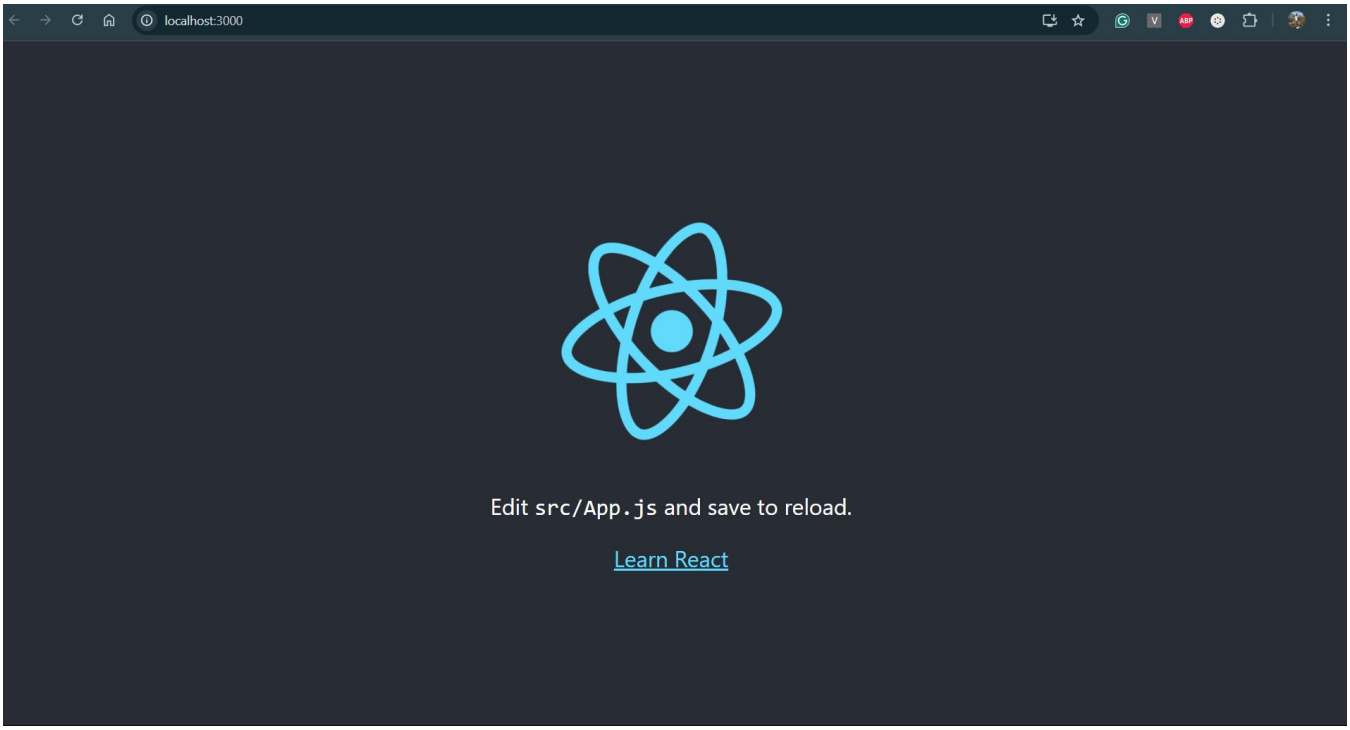
1. JavaScript

React JS is a JavaScript library developed for building fast, scalable, and interactive user interfaces, primarily for single-page applications. It focuses on creating reusable components that manage their own state and efficiently update the UI when data changes. React uses a virtual DOM to optimize rendering performance, ensuring that only the necessary parts of the UI are updated rather than reloading the entire page. Its component-based architecture promotes code reusability, maintainability, and scalability,

making it a popular choice for modern web development.

Setup of First Application using React js Steps:

1. npx create-react-app app-name
2. cd app-name
3. npm start



Importance of all Files considering the Folder Structure of First React Application:

When you create your first React application, the folder structure and each file in it has a specific role. Importance of all files in the default structure:

1. node_modules/ - Stores all the packages and dependencies that your project needs.

2. public/ - Contains static files that are served directly without going through Webpack.
   - index.html - Main HTML template where your React app is injected.
   - favicon.ico - Browser tab icon for your app.
   - manifest.json - Describes how the app behaves if installed on a device.
   - robots.txt - Tells search engines which pages they can or cannot crawl.

3. src/ - Where all your React code lives. This is the heart of the project.
   - index.js / index.jsx - Entry point of the React app.
   - App.js - Main component that holds your application's structure.
   - App.css - Styles specific to App.js.
   - index.css - Global styles for the entire app.
   - App.test.js - Contains default test cases.

- logo.svg - Example logo image imported into App.js.
- reportWebVitals.js - Tracks performance metrics.
- setupTests.js - Configuration for running tests.

4. .gitignore - Lists files and folders that Git should ignore.
5. package.json - Describes your project and dependencies.
6. package-lock.json - Locks dependency versions for consistent installs.
7. README.md - Contains documentation for your project.

Component in React: A component is a reusable and self-contained unit of a user interface that can be combined with other components to build complex applications. It acts like a building block, encapsulating its own structure, styling, and behavior, making it easier to manage and reuse across different parts of an application. Components can be broadly classified into two types: functional components, which are simple JavaScript functions that return JSX and can use hooks for managing state and side effects, and class components, which are ES6 classes that extend React.Component and use a render() method to return JSX. Each component can receive props (short for properties) from its parent, which are read-only inputs used to pass data, and can also manage its own state, which holds dynamic data that can change over time and trigger UI updates.

React JSX (JavaScript XML) is a syntax extension for JavaScript used in React to describe the UI structure in a way that closely resembles HTML. It allows developers to write HTML-like code directly inside JavaScript, making the UI easier to visualize and maintain. JSX is not understood by browsers directly, so it is compiled into standard JavaScript by tools like Babel before execution. It supports embedding JavaScript expressions within curly braces {}, enabling dynamic rendering of data. JSX also enforces a component-based structure and ensures that elements are returned as a single root node. Together, React JS provides the library and logic for building UIs, while JSX offers a readable and declarative way to define those UIs within JavaScript code.

Implementation Details:
Components/Greeting.js

```
import React from 'react';

const Greeting = (props) => {
  return (
    <div >
      Hello, {props.name} !
    </div>
  );
}

export default Greeting;
```

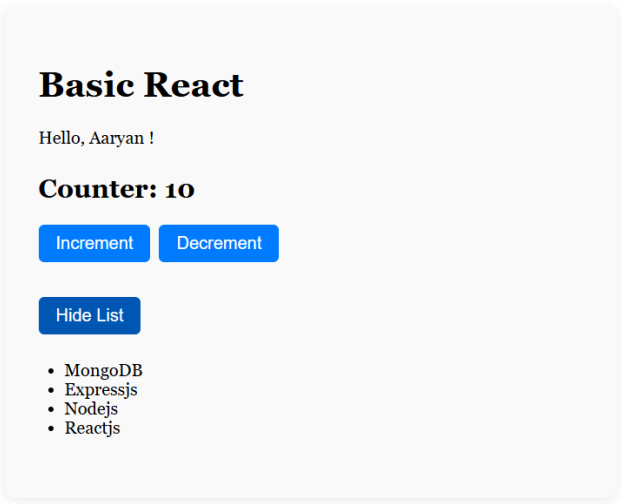
App.js

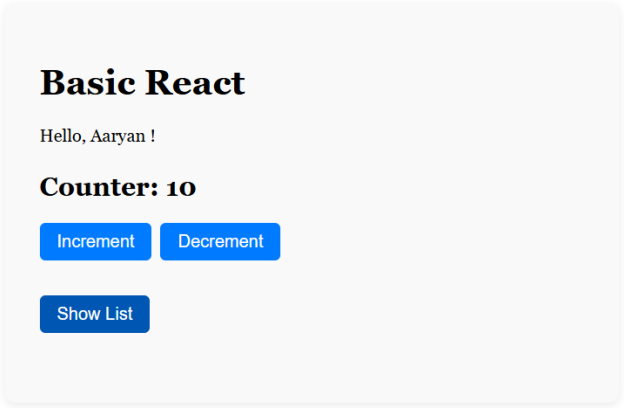
```
import React, { useState } from 'react';
import './App.css'
import Greeting from './components/Greeting';

function App() {
  const [counter, setCounter] = useState(0);
  const [showList, setShowList] = useState(true);
  const user = 'Aaryan';

  const items = ['MongoDB', 'Expressjs', 'Nodejs', 'Reactjs'];

  return (
    <div className="App">
      <h1>Basic React</h1>
      <Greeting name={user} />

      <div className="counter-section">
        <h2>Counter: {counter}</h2>
        <button onClick={() => setCounter(counter + 1)}>Increment</button>
        <button onClick={() => setCounter(counter - 1)}>Decrement</button>
      </div>

      <div className="toggle-list">
        <button onClick={() => setShowList((v) => !v)}>
          {showList ? 'Hide' : 'Show'} List
        </button>
        {showList && (
          <ul>
            {items.map((item, idx) => (
              <li key={idx}>{item}</li>
            ))}
          </ul>
        )}
      </div>
    </div>
  );
}

export default App;
```

App.css

```css
.App {
  max-width: 500px;
  margin: 2rem auto;
  padding: 2rem;
  background: #f9f9f9;
  border-radius: 10px;
  box-shadow: 0 2px 8px rgba(0,0,0,0.08);
  font-family: Georgia, 'Times New Roman', Times, serif;
}

.counter-section {
  margin: 1.5rem 0;
}

button {
  margin: 0 0.5rem 0.5rem 0;
  padding: 0.5rem 1rem;
  border: none;
  border-radius: 5px;
  background: #007bff;
  color: #fff;
  cursor: pointer;
  font-size: 1rem;
  transition: background 0.2s;
}
button:hover {
  background: #0056b3;
}

.toggle-list {
  margin: 1.5rem 0;
}

ul {
  padding-left: 1.5rem;
}
```

Output:





Conclusion:
I have successfully completed this experiment. I learned and implemented the fundamental concepts of React, including setting up the first React application, understanding the importance of each file in the folder structure, working with components, and using React JSX for building UI. I explored how props enable data flow between components, making them reusable and dynamic, and compared functional and class components to understand their syntax, state management, and lifecycle handling.

Postlab questions:

1. **Explain the Concept of SPA.**

   A Single Page Application (SPA) is a type of web application that loads a single HTML page and dynamically updates its content as the user interacts with the app, without requiring a full page reload. In an SPA, navigation between different sections is handled using JavaScript and frameworks/libraries like React, which update only the required parts of the page via the Virtual DOM or AJAX calls. This results in faster performance, a smoother user experience, and reduced server load compared to traditional multi-page applications.

2. **Describe Props**

   Props (properties) are read-only inputs passed from a parent component to a child component in React. They allow data to flow in a unidirectional manner, making components reusable and dynamic. Props cannot be modified by the receiving component, they are immutable within that component. For example:

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

function App() {
  return <Greeting name="Aaryan" />;
}
```

Here, "Aaryan" is passed as a prop to Greeting, which displays it dynamically.

3. Comparison with Functional and Class Component with example.

Functional components and class components in React differ in their syntax, state management, and lifecycle handling. Functional components are simple JavaScript functions that return JSX and can use React Hooks like useState and useEffect to manage state and side effects. They are generally shorter, easier to read, and perform slightly better because they have less overhead. Class components, on the other hand, are ES6 classes that extend React.Component and use a render() method to return JSX. They manage state using this.state and update it with setState(), and they rely on predefined lifecycle methods such as componentDidMount, componentDidUpdate, and componentWillUnmount. While functional components promote cleaner and more concise code, class components offer a more traditional object-oriented approach and were the primary way to manage state and lifecycle before Hooks were introduced in React 16.8. Today, functional components are preferred for most use cases, but class components are still supported and found in many legacy projects.