

IP-PRACTICAL NO 8

Date of Performance: -10-2022

Software Requirement: Visual Studio Code, Notepad, Notepad++, NodeJS, React.

Aim: To use ReactJS to implement function and class component, props and states and life cycle methods.

Objectives: The aim of this experiment is that the students will be able to:

- To install and configure React and learn its basics.
- To develop front end applications using React.

Outcomes: After study of this experiment, the students will be able to:

- To be able to create components to create static versions of applications.
- To develop interactive UI/front end applications using React.

Prerequisite: Basic knowledge of HTML, CSS, JavaScript & React required.

Theory:

Functional Component

Functional components are a way to write components that only contain a render method and don't have their own state. The Functional Component is also known as a stateless component because they do not hold or manage state.

Example: function Welcome (props) { return

```
<h1>Hello, {props.name}</h1>;
```

```
}
```

Class Component

Class components are more complex than functional components. It requires you to extend from React. Component and create a render function which returns a React element. The Class Component is also known as a stateful component because they can hold or manage local state.

Example: class Welcome extends

```
React.Component {
```

```
  render() {
```

```
    return <h1>Hello, {this.props.name}</h1>
```

```
  }
```

```
}
```

Props

Props stand for "Properties." They are read-only components. It is an object which stores the value of attributes of a tag and work similar to the HTML attributes. Props are passed to the component in the same way as arguments passed in a function. Props are immutable so we cannot modify the props from inside the component. These attributes are available in the component as `this.props` and can be used to render dynamic data in our render method.

State

The state is an updatable structure that is used to contain data or information about the component. The state in a component can change over time. They are also responsible for making a component dynamic and interactive. State can be set by using the `setState()` method and calling `setState()` method triggers UI updates. To define a state, you have to first declare a default set of values for defining the component's initial state. To do this, add a class constructor which assigns an initial state using `this.state`. The `'this.state'` property can be rendered inside `render()` method.

React Lifecycle

Each component in React has a lifecycle which you can monitor and manipulate during its three main phases. The three phases are: Mounting, Updating, and Unmounting.

Mounting

Mounting means putting elements into the DOM. React has four built-in methods that gets called, in this order, when mounting a component:

- `constructor()`
- `getDerivedStateFromProps()`
- `render()`
- `componentDidMount()`

Updating

The next phase in the lifecycle is when a component is updated. A component is updated whenever there is a change in the component's state or props. React has five built-in methods that gets called, in this order, when a component is updated:

- `getDerivedStateFromProps()`
- `shouldComponentUpdate()`
- `render()`
- `getSnapshotBeforeUpdate()`
- `componentDidUpdate()`

Unmounting

The next phase in the lifecycle is when a component is removed from the DOM, or unmounting as React likes to call it. React has only one built-in method that gets called when a component is unmounted:

- `componentWillUnmount()`

Problem Statement:

Implement Functional & Class component, props & states & Life cycle methods in ReactJS.

Steps to create React Application

Step 1: Open command prompt and go to the directory where you want to create your first react js application using the command 'cd <directoryName>'

Step 2: Create react project with the command 'npx create-react-app helloreact'. 'helloreact' is the name of project/folder. It will install all dependencies as well.

OR

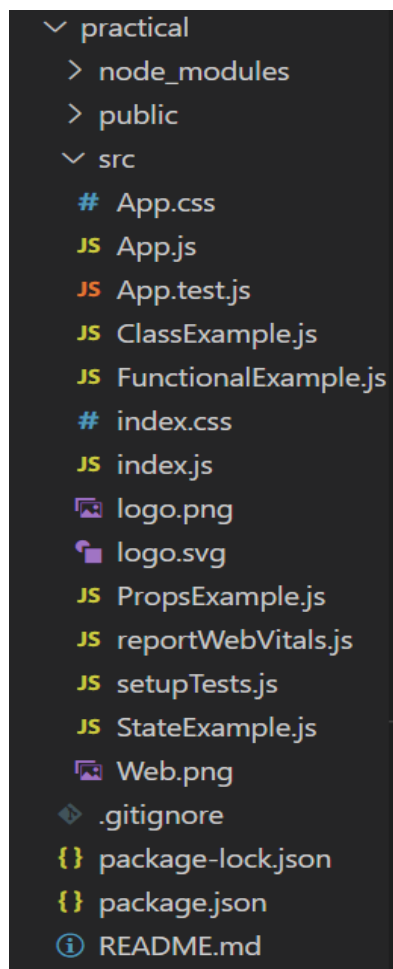
You can first install react using 'npm install -g create-react-app' and then 'npm create-react-app helloreact'

Step 3: Type in the command prompt

cd helloreact

npm start

folder structure of 'practical' folder



Source Code and Output:

Functional Component, Class Component, Props & State:

App.js

```
import logo from './Web.png';
import './App.css';
import ClassExample from './ClassExample';
import FunctionalExample from './FunctionalExample';
import PropsExample from './PropsExample';
import StateExample from './StateExample';
function App() {
  return (
    <><div className="App">
      <FunctionalExample />
      <br></br>
      <img src={logo} className="IP tech pic" alt="logo" />
    </div>
    <div className='body'>
      <ClassExample />
      <br></br>
      <StateExample />
      <header className="App-header">
        <a
          className="App-link"
          href="https://developer.mozilla.org/en-US/docs/Learn"
          target="_blank"
          rel="noreferrer"
        >Learn Web development </a>
      </header>
      <h2>Some technologies in IP -</h2>
      <ul>
        <li><PropsExample name="HTML"/></li>
        <li><PropsExample name="CSS"/></li>
        <li><PropsExample name="JavaScript"/></li>
        <li><PropsExample name="Bootstrap"/></li>
        <li><PropsExample name="React"/></li>
        <li><PropsExample name="Angular"/></li>
        <li><PropsExample name="NodeJS"/></li>
        <li><PropsExample name="Express"/></li>
      </ul>
    </div></>
  );
}
export default App;
```

App.css

```
* {  
  font-family: Arial, Helvetica, sans-serif;  
  background-color:lavender;  
}  
  
.App {  
  text-align: center;  
  color:darkviolet;  
}  
  
.body {  
  color:dimgray;  
}  
  
.App-logo {  
  height: 60px;  
  width: 60px;  
  display: block;  
  margin-left: auto;  
  margin-right: auto;  
}  
  
.App-header {  
  display: flex;  
  flex-direction: column;  
  align-items: center;  
  justify-content: center;  
}  
  
.App-link {  
  color: crimson;  
  font-size: 150%;  
}
```

FunctionalExample.js

```
import React from 'react'  
function FunctionalExample() {  
  return <h1>Internet Programming</h1>  
}  
export default FunctionalExample
```

ClassExample.js

```
import React, { Component } from 'react'
class ClassExample extends Component {
  render() {
    return <h3><br></br>Internet programming refers to the writing, markup
and coding involved in Web development
    , which includes Web content, Web client and server scripting and
network security.
    Web programming is different from just programming, which requires
interdisciplinary knowledge
    on the application area, client and server scripting, and database
technology.</h3>
  }
}
export default ClassExample
```

PropsExample.js

```
import React from 'react'
const PropsExample = props => {
  console.log(props)
  return <h4>IP - {props.name}</h4>
}
export default PropsExample
```

StateExample.js

```
import React, { Component } from 'react'
class StateExample extends Component {
  constructor() {
    super()
    this.state = {message: 'Accessibility, Easy to use, Security, Flexibility,
Low Costt.'}
  }
  render() {
    return <h4><br></br>Internet programming is used on a very large scale
today.
    The important features of this technology are as following -
    {this.state.message}</h4>
  }
}
export default StateExample
```

Output:

Internet Programming



Internet programming refers to the writing, markup and coding involved in Web development , which includes Web content, Web client and server scripting and network security. Web programming is different from just programming, which requires Interdisciplinary knowledge on the application area, client and server scripting, and database technology.

Internet programming is used on a very large scale today. The important features of this technology are as following -Accessibility, Easy to use, Security, Flexibility, Low Cost.

[Learn Web development](#)

[Learn Web development](#)

Some technologies in IP -

- IP - HTML
- IP - CSS
- IP - JavaScript
- IP - Bootstrap
- IP - React
- IP - Angular
- IP - NodeJS
- IP - Express

Folder Structure of 'lifecycledemo' folder:

```
└─ lifecycledemo
  └─ > node_modules
  └─ > public
  └─ > src
    └─ # App.css
    └─ JS App.js
    └─ JS App.test.js
    └─ # index.css
    └─ JS index.js
    └─ JS Lifecycle1.js
    └─ JS Lifecycle2.js
    └─ JS LifecycleUnmount.js
    └─ logo.svg
    └─ JS reportWebVitals.js
    └─ JS setupTests.js
    └─ .gitignore
    └─ {} package-lock.json
    └─ {} package.json
    └─ i README.md
```

Source Code:**React Lifecycle - Mounting*****App.js:***

```
import React, {Component} from 'react'
import './App.css';
import Lifecycle1 from './components/Lifecycle1';
class App extends Component {
  render() {
    return (
      <div className="App">
        <Lifecycle1></Lifecycle1>
      </div>
    )
  }
}
export default App;
```

App.css:

```
*{
  text-align: center;
  font-size: 120%;
  color:darkviolet;
  background-color: blanchedalmond;
}
```

Lifecycle1.js:

```
import React, { Component } from 'react'
import Lifecycle2 from './Lifecycle2'
class Lifecycle1 extends Component {
  constructor(props) {
    super(props)
    this.state = {
      name:'Husna'
    }
    console.log('Lifecycle1')
  }
  static getDerivedStateFromProps(props, state)
  {
    console.log('Lifecycle1 getDerivedStateFromProps')
    return null
  }
  componentDidMount() {
    console.log('Lifecycle1 componentDidMount')
  }
  render() {
    console.log('Lifecycle1 render')
    return(<div>
```

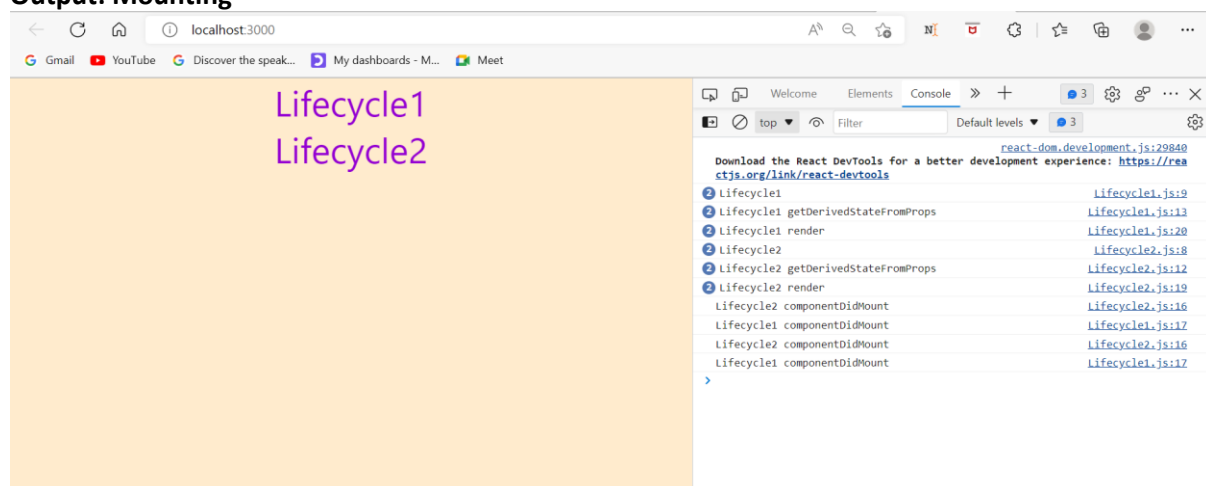


```
    <div>Lifecycle1</div>
    <Lifecycle2></Lifecycle2>
  </div>
)

}
}
export default Lifecycle1
```

Lifecycle2.js:

```
import React, { Component } from 'react'
class Lifecycle2 extends Component {
  constructor(props) {
    super(props)
    this.state = {
      name: 'Husna'
    }
    console.log('Lifecycle2')
  }
  static getDerivedStateFromProps(props, state) {
    {
      console.log('Lifecycle2 getDerivedStateFromProps')
      return null
    }
  }
  componentDidMount() {
    console.log('Lifecycle2 componentDidMount')
  }
  render() {
    console.log('Lifecycle2 render')
    return <div>Lifecycle2</div>
  }
}
export default Lifecycle2
```

Output: Mounting

The screenshot shows a web browser at localhost:3000 displaying the text "Lifecycle1" and "Lifecycle2" in purple on an orange background. The React DevTools interface is open on the right, showing the Console tab with the following log messages:

- react-dom.development.js:29849 Download the React DevTools for a better development experience: <https://reactjs.org/link/react-devtools>
- Lifecycle1 Lifecycle1.js:9
- Lifecycle1 getDerivedStateFromProps Lifecycle1.js:13
- Lifecycle1 render Lifecycle1.js:20
- Lifecycle2 Lifecycle2.js:8
- Lifecycle2 getDerivedStateFromProps Lifecycle2.js:12
- Lifecycle2 render Lifecycle2.js:19
- Lifecycle2 componentDidMount Lifecycle2.js:16
- Lifecycle1 componentDidMount Lifecycle1.js:17
- Lifecycle2 componentDidMount Lifecycle2.js:16
- Lifecycle1 componentDidMount Lifecycle1.js:17

Source Code:**React Lifecycle - Updating*****App.js:***

```
import React, { Component } from 'react'
import './App.css';
import Lifecycle1 from './Lifecycle1';
class App extends Component {
  render() {
    return (
      <div className="App">
        <Lifecycle1 />
      </div>
    )
  }
}
export default App;
```

App.css:

```
*{
  text-align: center;
  font-size: 110%;
  color:mediumblue;
}
```

Lifecycle1.js:

```
import React, { Component } from 'react'
import Lifecycle2 from './Lifecycle2'

class Lifecycle1 extends Component {
  constructor(props) {
    super(props)
    this.state = {
      name : 'Husna'
    }
    console.log('Lifecycle1 constructor')
  }
  static getDerivedStateFromProps(props, state) {
    console.log('Lifecycle1 getDerivedStateFromProps')
    return null
  }
  componentDidMount() {
    console.log('Lifecycle1 componentDidMount')
  }
  shouldComponentUpdate() {
    console.log('Lifecycle1 shouldComponentUpdate');
    return true
  }
  getSnapshotBeforeUpdate(prevProps, prevState) {
```

```
    console.log('Lifecycle1 getSnapshotBeforeUpdate');
    return null
  }
  componentDidUpdate() {
    console.log('Lifecycle1 componentDidUpdate')
  }
  changeState = () => {
    this.setState({
      name: 'Sarah'
    })
  }
  render() {
    console.log('Lifecycle1 render')
    return (
      <div>
        <div>Lifecycle1</div>
        <br></br>
        <button onClick={this.changeState}>Change State</button>
        <br></br>
        <br></br>
        <Lifecycle2 />
      </div>
    )
  }
}
export default Lifecycle1;
```

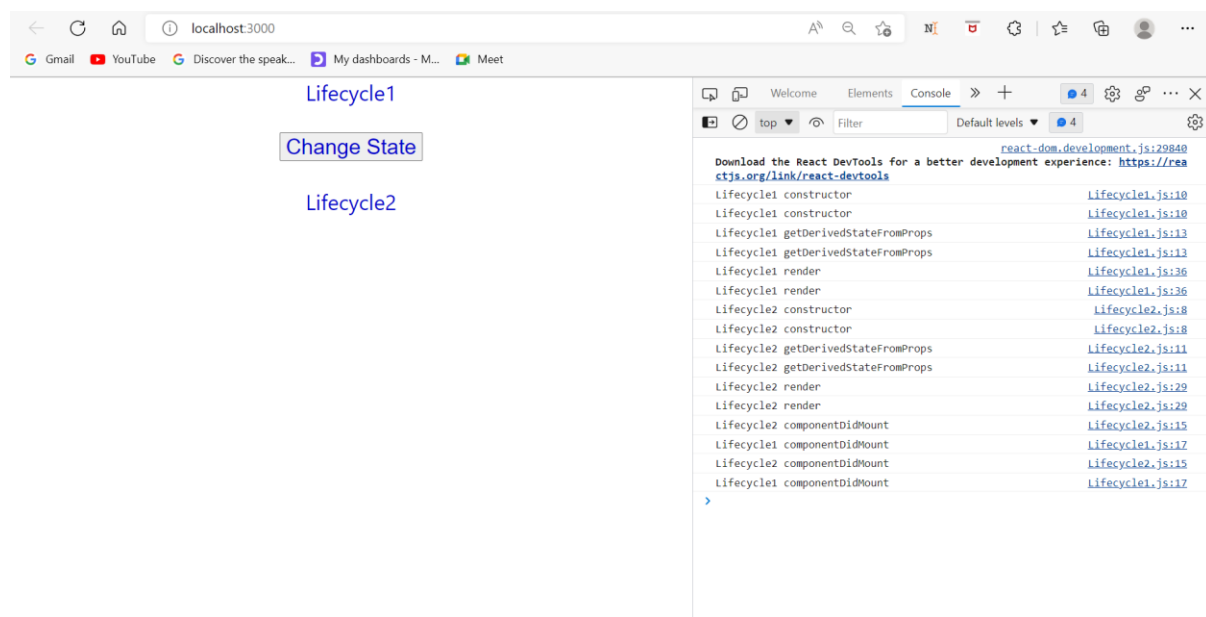
Lifecycle2.js:

```
import React, { Component } from 'react'
class Lifecycle2 extends Component {
  constructor(props) {
    super(props)
    this.state = {
      name : 'Husna'
    }
    console.log('Lifecycle2 constructor')
  }
  static getDerivedStateFromProps(props, state) {
    console.log('Lifecycle2 getDerivedStateFromProps')
    return null
  }
  componentDidMount() {
    console.log('Lifecycle2 componentDidMount')
  }
  shouldComponentUpdate() {
    console.log('Lifecycle2 shouldComponentUpdate');
    return true
  }
}
```

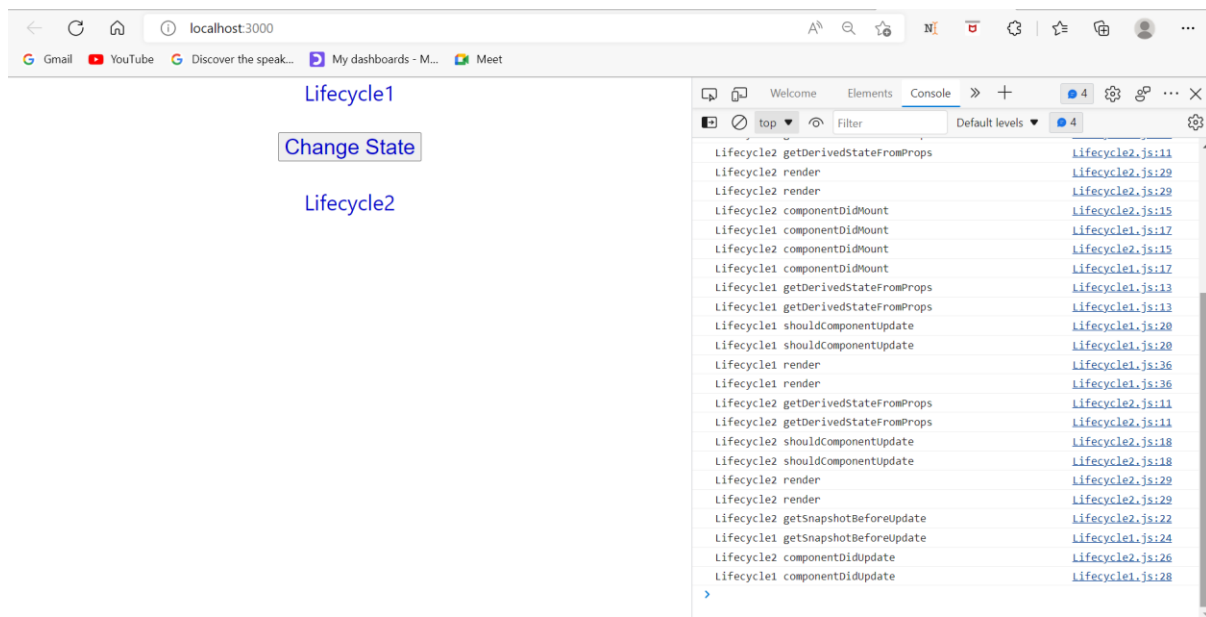
```
getSnapshotBeforeUpdate(prevProps, prevState) {
  console.log('Lifecycle2 getSnapshotBeforeUpdate');
  return null
}
componentDidUpdate() {
  console.log('Lifecycle2 componentDidUpdate')
}
render() {
  console.log('Lifecycle2 render')
  return <div>Lifecycle2</div>
}
}
export default Lifecycle2;
```

Output: Updating

In the console you will be able to see the lifecycle components in the order they are executing before clicking on 'Change State' Button.



In the console you will be able to see the lifecycle components change their state & in the order they are executing after clicking on 'Change State' Button.



Source Code:

React Lifecycle – Unmounting

App.js:

```
import React, { Component } from 'react'
import './App.css';
import LifecycleUnmount from './LifecycleUnmount';
class App extends Component {
  render() {
    return (
      <div className="App">
        <LifecycleUnmount />
      </div>
    )
  }
}
export default App;
```

App.css:

```
*{
  text-align: center;
  font-size: 110%;
  color:mediumblue;
}
```

LifecycleUnmount.js:

```
import React, { Component } from 'react'
export default class LifecycleUnmount extends Component {
```

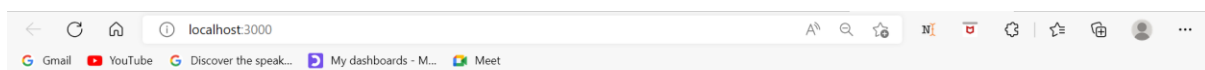
```
constructor(props){
  super(props)
  this.state = {
    show: true,
  }
}

render() {
  return (
    <div>
    <p>{this.state.show ? <Child/> : null}</p>
    <div className="divbtn">
    <button className="btn" onClick={() => {this.setState({show:
!this.state.show})}}> Click here to Unmount </button>
    </div>
    </div>
  )
}
}

export class Child extends Component {
  componentWillUnmount(){
    alert('This will unmount')
  }

  render(){
    return(
    <p>LifecycleUnmount</p>
    )
  }
}
```

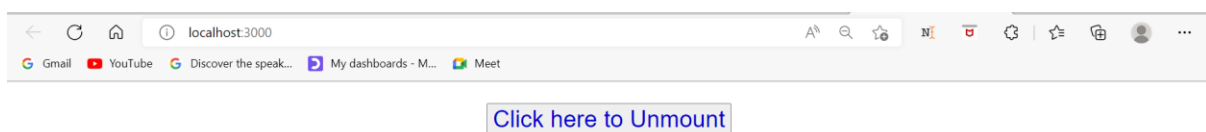
Output: Unmounting



On clicking the button 'Click here to Unmount', an alert pops-up.



After clicking the button, the component is removed.



Conclusion:

From this experiment, we learned about ReactJS which is a JavaScript library, used for creating front-end applications. We implemented various JavaScript components namely – Functional Component, Class Component, Props & State. We learned about the lifecycle of components and implemented them.