Practical No. 1

Create an application to demonstrate Node.js Modules.

AIM:-1(A) Core Module/Built in Modules

DISCRIPTION:-

Node.js modules are reusable pieces of JavaScript code that can be imported and exported between files. They allow developers to break applications into smaller, manageable chunks. Node.js provides built-in modules such as fs (file system), http (server creation), and path (file paths).

CODE:- 1 .OS

```
var os = require('os');

console.log(os.EOL);

console.log(os.arch());

console.log(os.hostname());

console.log(os.totalmem());

console.log(os.freemem()) ;

console.log(os.platform());

console.log(os.type());

console.log(os.userInfo())  ;
```

OUTPUT:-

```
PS C:\Users\Rohit\Desktop\Web Technology\CoreModule> node os.js


x64
DESKTOP-1F86KRB
4175806464
772153344
win32
Windows_NT
{
  uid: -1,
  gid: -1,
  username: 'Rohit',
  homedir: 'C:\\Users\\Rohit',
  shell: null
}
```

CODE:- 2. URL

```
var url =require('url');

console.log(url.parse('https://google.com'));

console.log(url.domainToASCII('google.com'));

console.log(url.domainToUnicode('xn--espaol-zwa.com'));
```

OUTPUT:-

```
PS C:\Users\Rohit\Desktop\Web Technology\CoreModule> node URL.js
Url {
  protocol: 'https:',
  slashes: true,
  auth: null,
  host: 'google.com',
  port: null,
  hostname: 'google.com',
  hash: null,
  search: null,
  query: null,
  pathname: '/',
  path: '/',
  href: 'https://google.com/'
}
google.com
español.com
```

CODE:- 3. PATH

```
var path = require('path');

console.log(path.dirname('C:/Users/Rohit/Desktop/Web Technology/CoreModule/path.js'));
console.log(path.basename('C:/Users/Rohit/Desktop/Web Technology/CoreModule/path.js'));
console.log(path.extname('C:/Users/Rohit/Desktop/Web Technology/CoreModule/path.js'));
console.log(path.join('/sys','/rohit','/CoreModule'));
console.log(path.parse('C:/Users/Rohit/Desktop/Web Technology/CoreModule/path.js'));
var mypath=(path.parse('C:/Users/Rohit/Desktop/Web Technology/CoreModule/path.js'));
console.log(mypath.name);
console.log(mypath.root);
console.log(mypath.ext);
```

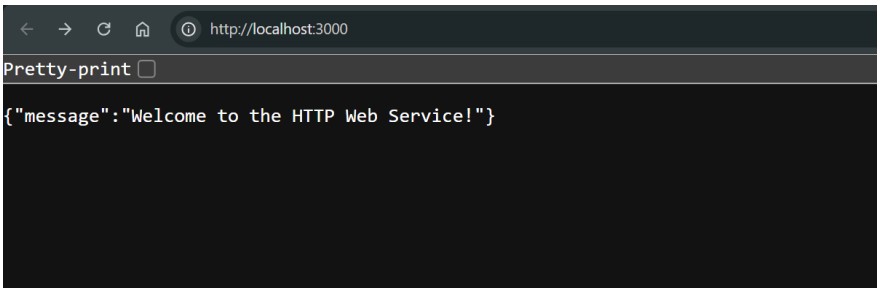
OUTPUT:-

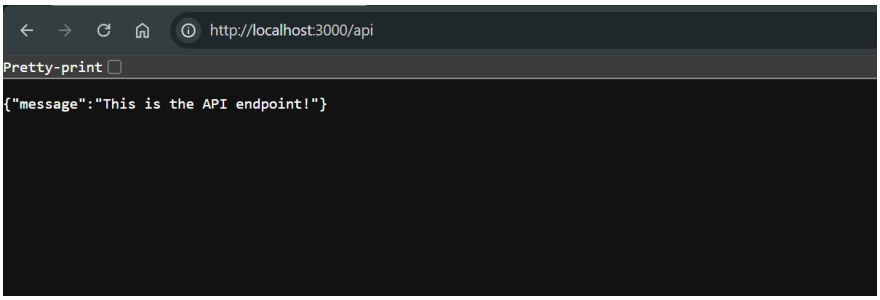
```
PS C:\Users\Rohit\Desktop\Web Technology\CoreModule> node path.js
C:/Users/Rohit/Desktop/Web Technology/CoreModule
path.js
.js
\sys\rohit\CoreModule
{
  root: 'C:/',
  dir: 'C:/Users/Rohit/Desktop/Web Technology/CoreModule',
  base: 'path.js',
  ext: '.js',
  name: 'path'
}
path
C:/
.js
```

CODE:- 4. HTTP

```
const http = require('http');

// Create an HTTP server
const server = http.createServer((req, res) => {
  // Set the response header
  res.writeHead(200, { 'Content-Type': 'application/json' });

  // Define a simple route
  if (req.url === '/' && req.method === 'GET') {
    res.end(JSON.stringify({ message: 'Welcome to the HTTP Web Service!' }));
  } else if (req.url === '/api' && req.method === 'GET') {
    res.end(JSON.stringify({ message: 'This is the API endpoint!' }));
  } else {
    // Handle 404
    res.writeHead(404, { 'Content-Type': 'application/json' });
    res.end(JSON.stringify({ error: 'Resource not found' }));
  }
});

// Start the server
const PORT = 3000;
server.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});
```

OUTPUT:-





AIM:-1(B) Customizable Module/local Modules

DISCRIPTION:-

Additionally, custom modules can be created using the exports or module.exports keyword. To include a module, the require() function is used. Modules encourage code reusability, separation of concerns, and easy maintenance. For example, a module containing database operations can be created and reused in multiple files.

Node.js also supports third-party modules via npm (Node Package Manager), such as express for web applications or axios for HTTP requests.

CODE:-

calc.js

```
exports.add = function (x, y) {
    return x + y;
}
exports.sub = function (x, y) {
    return x  -   y;
}
exports.mul = function (x, y) {
    return x * y;
}
exports.div = function (x, y) {
    return x / y;
}
```

usingmodule .js

```
var cal = require('./calc');   var x = 50;   var y = 100;
console.log("Adddition=", cal.add(x, y));
console.log("Subtraction=", cal.sub(x, y));
console.log("Multiplication=", cal.mul(x, y));
console.log("Divison=", cal.div(x, y));
```

OUTPUT:-

```
PS C:\Users\Rohit\Desktop\Web Technology\Customize_Local_Module> node usingmodule.js
Adddition= 150
Subtraction= -50
Multiplication= 5000
Divison= 0.5
```

Practical No. 2

AIM:- Create an application to demonstrate various Node.js Events

DISCRIPTION:-

Node.js events are a core feature of its asynchronous programming model. It uses the `events` module to create and manage events. The `EventEmitter` class allows binding functions (listeners) to events, which are triggered upon specific occurrences. For instance, an HTTP server triggers a `request` event when it receives a request. Custom events can also be created using `emit()` and `on()`. This event-driven model enhances Node.js's performance, making it ideal for I/O-heavy and real-time applications like chat apps and live notifications.

CODE:-

```
var events = require('events');
var emitter = new events.EventEmitter();

//listening to event
emitter.on('eventname', () => {
    console.log('event get fired');
})

//event fired
emitter.emit('eventname');
```

OUTPUT:-

```
PS C:\Users\Rohit\Desktop\Web Technology\Event> node events.js
event get fired
```

Practical No. 3

AIM:- Create an application to demonstrate Node.js Functions

DISCRIPTION:-

Functions in Node.js are essential for structuring code. They can be standard JavaScript functions or asynchronous ones using `async/await` or callbacks. Functions enable modularity and reusability of code. For example, a function to connect to a database or fetch an API response can be reused multiple times. Node.js supports higher-order functions, closures, and anonymous functions. Asynchronous functions help manage non-blocking operations, critical for handling multiple user requests efficiently.

CODE:-

function.js

```
//function in nodejs
var div = function (a, b) {
return (a / b);
}

console.log("Division=", div(10, 5));
//arrow function format in node js
var add = (a, b) => (a + b);
console.log("Addition=", add(12, 13));
```

OUTPUT:-

```
PS C:\Users\Rohit\Desktop\Web Technology> node function.js
Division= 2
Addition= 25
```

CODE:-

anonymousvalue.js

```
var middlename = "ADHIK"

let myname = {     firstname: "ROHIT",

                   lastname: "SURYAWANSHI"

};

(function () {

 console.log(myname.firstname, middlename, myname.lastname);

  console.log("This is Hiray College");

})(myname);

//console.log(middlename) setTimeout(function ()

{     console.log("it Get Execute After 5

Seconds")

}, 5000);
```

OUTPUT:-

```
PS C:\Users\Rohit\Desktop\Web Technology\Anonymous Function> node anonymousvalue.js
ROHIT ADHIK SURYAWANSHI
This is Hiray College
it Get Execute After 5 Seconds
```

Practical No. 4

AIM:- Using File Handling demonstrate all basic file operations (Create, write, read, delete)

DISCRIPTION:-

Node.js supports file handling through the `fs` module, enabling developers to perform file operations such as creating, reading, writing, and deleting files. The `fs.writeFile()` function creates and writes to a file, while `fs.readFile()` reads file content. To delete files, the `fs.unlink()` method is used. Both synchronous and asynchronous methods are available. Asynchronous methods are preferred for non-blocking execution. File handling is crucial for tasks such as logging, data storage, and file-based communication.

CODE:-

Read.js

```
var fs = require('fs');
fs.readFile('C:\\Users\\Rohit\\Desktop\\Web Technology\\FileSystem\\data.txt', function
(err, data) {
    if (err) throw err;

    console.log(data.toString());

});
```

OUTPUT:-

```
PS C:\Users\Rohit\Desktop\Web Technology\FileSystem> node read.js
Hi! Rohit
```

CODE:-

async.js

```
var fs = require('fs');
console.log('here we are begin');
console.log('this is start point of application');
var content = fs.readFile('C:\\Users\\Rohit\\Desktop\\Web
Technology\\FileSystem\\data.txt', function(err, data) {
    if (err) throw err;

    console.log(data.toString());
})
console.log('all actions completed');
```

OUTPUT:-

```
PS C:\Users\Rohit\Desktop\Web Technology\FileSystem> node async.js
here we are begin
this is start point of application
all actions completed
Hi! Rohit
```

Practical No. 5

AIM:- Create an HTTP Server and perform operations on it

DISCRIPTION:-

Node.js provides the `http` module to create servers that handle client requests. A basic HTTP server can be created using the `createServer()` function, which listens for requests on a specified port. Developers can handle HTTP methods such as GET and POST, manage responses, and route requests. For example, an HTTP server can serve static files or APIs. The asynchronous nature of Node.js ensures high performance for concurrent user requests.

CODE:-

```
const express = require('express');
const app = express();

// Middleware to parse JSON requests
app.use(express.json());

// Define routes
app.get('/', (req, res) => {
  res.json({ message: 'Welcome to the HTTP Web Service!' });
});

app.get('/api', (req, res) => {
  res.json({ message: 'This is the API endpoint!' });
});

// Handle 404
app.use((req, res) => {
  res.status(404).json({ error: 'Resource not found' });
});

// Start the server
const PORT = 3000;
app.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});
```

OUTPUT:-

```
PROBLEMS    OUTPUT    PORTS    TERMINAL

PS C:\Users\A\Downloads\ds project> node .\http.js
Server is running on http://localhost:3000
```

http://localhost:3000

Pretty-print

{"message":"Welcome to the HTTP Web Service!"}

http://localhost:3000/api

Pretty-print

{"message":"This is the API endpoint!"}

http://localhost:3000/api2

Pretty-print

{"error":"Resource not found"}

Practical No. 6

AIM:- Create an application to establish a connection with the MySQL database and perform basic database operations on it

DISCRIPTION:-

To connect to a MySQL database, the `mysql` or `mysql2` package is used in Node.js. A connection is established using a configuration object containing the host, user, password, and database name. Operations such as INSERT, SELECT, UPDATE, and DELETE can be performed using SQL queries. Asynchronous methods like callbacks or promises ensure smooth data interaction without blocking the event loop. This is essential for applications requiring persistent data storage, such as user management systems.

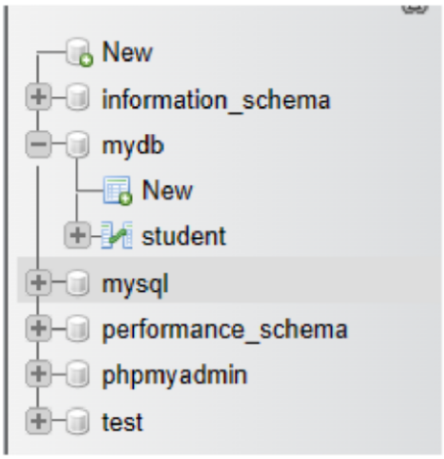
CODE:-

Createtable.js

```
var mysql = require('mysql');

var con = mysql.createConnection({
    host: "localhost",
    user: "root",
    password: "",
    database: 'mydb'
});

con.connect(function (err) {
    if (err) throw err;
    console.log("Connected");


    con.query("CREATE TABLE STUDENT(STUDENT_ID INT(10), STUDENT_NAME
VARCHAR(20),STUDENT_EMAILID VARCHAR(100),STUDENT_MOBNO INT(15))",
        function (err, result) {
            if (err) throw err;
            console.log("TABLE CREATED");
        });
});
```

OUTPUT:-

```
PS C:\Users\Rohit\Desktop\Web Technology\Database> node CREATETABLE.js
Connected
TABLE CREATED
```



Practical No. 7

AIM:- : Create an application in ReactJS to implement component life cycle

DISCRIPTION:-

constructor():

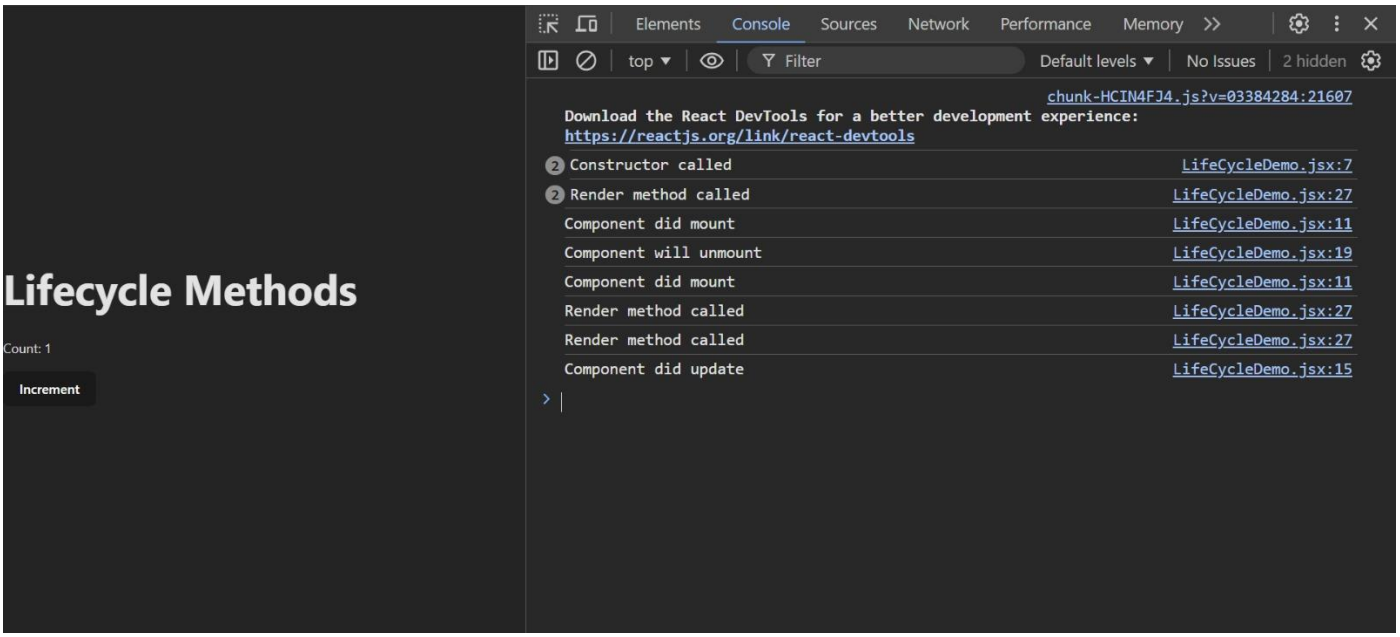
React components have a lifecycle with distinct phases: mounting, updating, and unmounting. Lifecycle methods, such as `componentDidMount`, `componentDidUpdate`, and `componentWillUnmount`, allow developers to perform specific actions during these phases. For instance, `componentDidMount` is used to fetch initial data. React's lifecycle methods help manage state, API calls, and cleanups, ensuring smooth application performance.

CODE:-

```
// 1. Component Life Cycle (Class Component)
import React, { Component } from 'react';

class LifeCycleDemo extends Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
    console.log('Constructor called');
  }

  componentDidMount() {
    console.log('Component did mount');
  }

  componentDidUpdate() {
    console.log('Component did update');
  }

  componentWillUnmount() {
    console.log('Component will unmount');
  }

  incrementCount = () => {
    this.setState({ count: this.state.count + 1 });
  };

  render() {
    console.log('Render method called');
```

```
      return (
        <div>
          <h1>Lifecycle Methods</h1>
          <p>Count: {this.state.count}</p>
          <button onClick={this.incrementCount}>Increment</button>
        </div>
      );
    }
  }

  export default LifeCycleDemo;
```

OUTPUT:-



Practical No. 8

AIM:- : Create an application to implement class and functional component in ReactJS

DISCRIPTION:-

Class components in React use `class` syntax and can manage state using `this.state` and lifecycle methods. Functional components, on the other hand, are simpler and use functions to render UI. With the introduction of React Hooks, functional components can also manage state (`useState`) and lifecycle (`useEffect`). Functional components are lightweight, easier to test, and encourage cleaner code. Both are used based on application requirements.

CODE:-

ClassComponent.jsx

```
import { Component } from "react";
class ClassComponent extends Component {
  render() {
    return <h1>This is a Class Component</h1>;
  }
}
export default ClassComponent;
```

Functional

```
import ClassComponent from "./ClassComponent.jsx";

const Functional = () => {
  return (
    <div>
      <ClassComponent />
      <h2>This is Functional Component</h2>
    </div>
  );
};
export default Functional;
```

main

```
import { createRoot } from "react-dom/client";
import Functional from "./Functional";

createRoot(document.getElementById("root")).render(<Functional />);
```

OUTPUT:-

```
This is a Class Component
This is Functional Component
```

Practical No. 9

AIM:- : Create an application in ReactJSto import and export the files (components)

DISCRIPTION:-

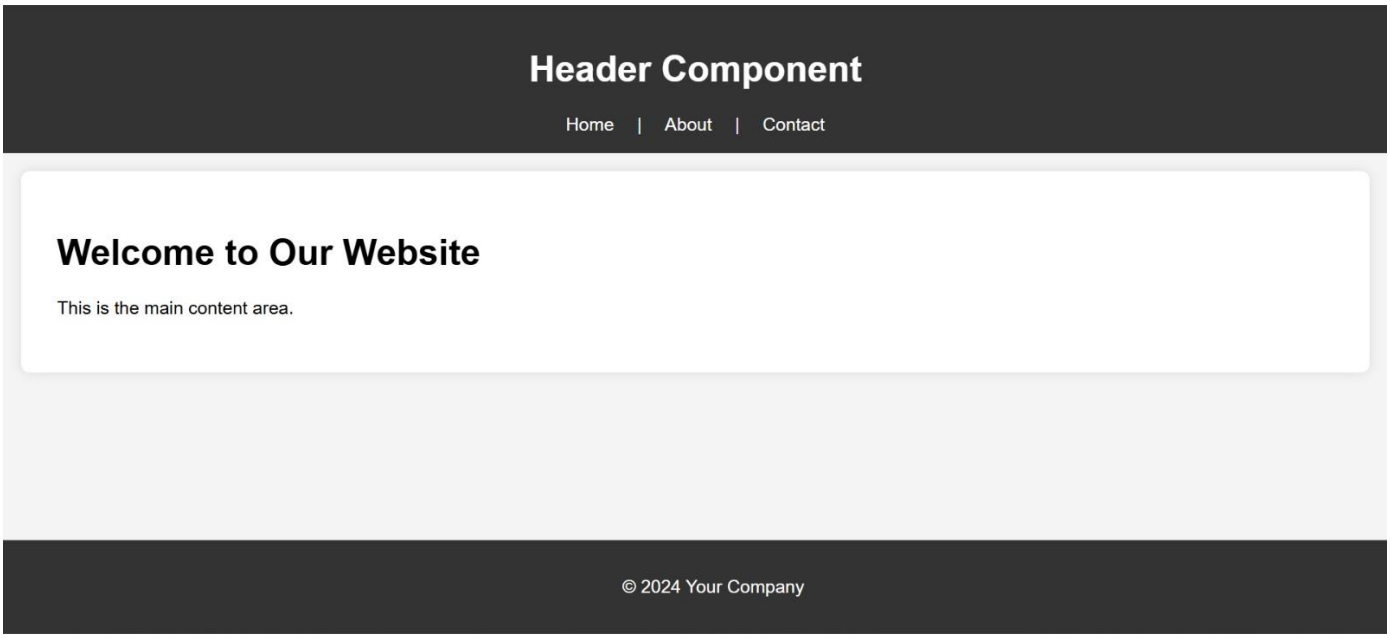
In ReactJS, `import` and `export` enable modular code by separating components into different files. Components, functions, or variables can be exported using `export` or `export default`. These can then be imported into other files using `import`. This separation improves maintainability and reusability. For example, a reusable button component can be created and imported into multiple pages.

CODE:-

```
// File: Header.js
export const Header = () => {
  return (
    <header>
      <h1>Header Component</h1>
      <nav>
        <a href="/home">Home</a> | <a href="/about">About</a> | <a
href="/contact">Contact</a>
      </nav>
    </header>
  );
};

// File: Footer.js
export const Footer = () => {
  return (
    <footer>
      <p>© 2024 Your Company</p>
    </footer>
  );
};

// File: App.js
import React from 'react';
import { Header } from './Header';
import { Footer } from './Footer';
```

```
function App() {
  return (
    <div>
      <Header />
      <main>
        <h1>Welcome to Our Website</h1>
        <p>This is the main content area.</p>
      </main>
      <Footer />
    </div>
  );
}

export default App;
```

OUTPUT:-



Practical No. 10

AIM:- : Create an application to implement state and props

DISCRIPTION:-

State and props are the two primary data-handling methods in React. State represents the dynamic data of a component and is managed internally. Props are read-only inputs passed from parent to child components. Together, they allow components to be dynamic and interactive. For example, a parent component can pass a user's name as a prop to a child component to display it.

CODE:-

//**ParentComponent.jsx**

```
import  { useState } from "react";
import ChildComponent from "./ChildComponent";

const ParentComponent = () => {
  const [userName, setUserName] = useState("Vitthal");

  const changeUserName = () => {
    setUserName("Korvan");
  };

  return (
    <div>
      <h1>Welcome, {userName}!</h1>
      <ChildComponent name={userName} />
      <button onClick={changeUserName}>Change User Name</button>
    </div>
  );
};
```

```
export default ParentComponent;
```

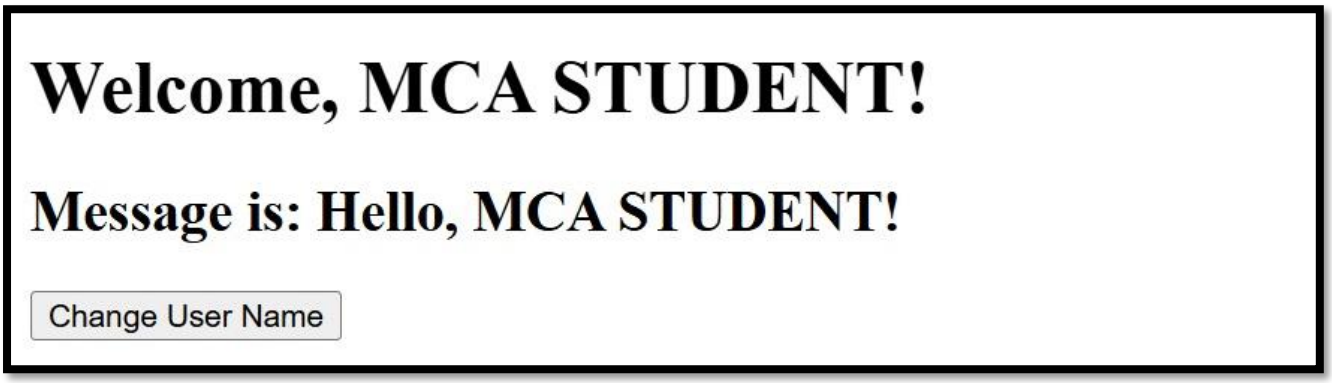
//**ChildComponent**

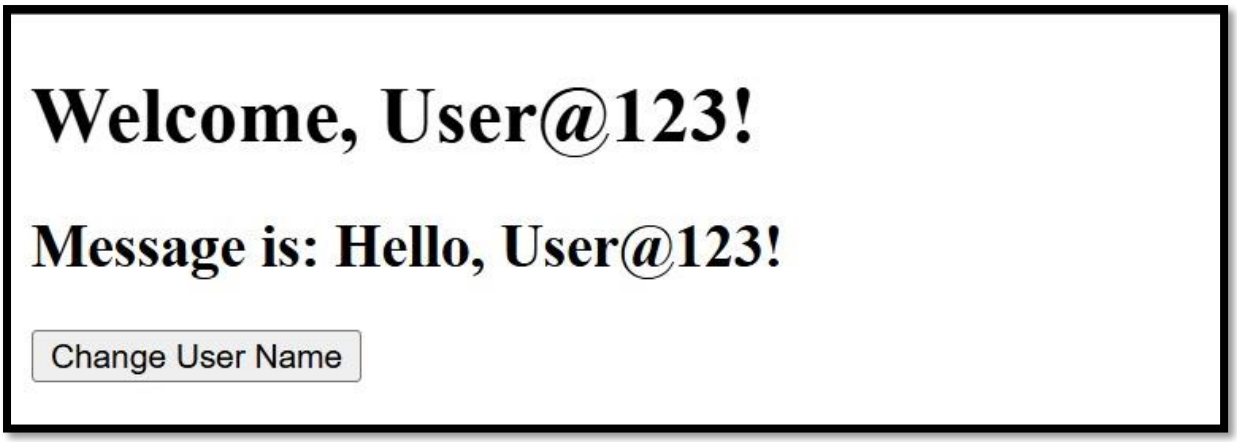
```
const ChildComponent = ( { name } ) => {
  return <h2>Message is: Hello, {name}!</h2>;
};

export default ChildComponent;
```

//Main.jsx

```
import { createRoot } from "react-dom/client";
import ParentComponent from "./ParentComponent";

createRoot(document.getElementById("root")).render(<ParentComponent />
```

OUTPUT:-





Practical No. 11

AIM:- : Create an application in ReactJSto use DOM events

DISCRIPTION:-

React uses synthetic events, which are wrappers around browser-native DOM events. Events such as `onClick`, `onChange`, and `onSubmit` are used to handle user interactions. For example, a button click can trigger a function to update state or send an API request. React's event system ensures cross-browser compatibility and optimizes performance by delegating events.

CODE:-

//App.jsx

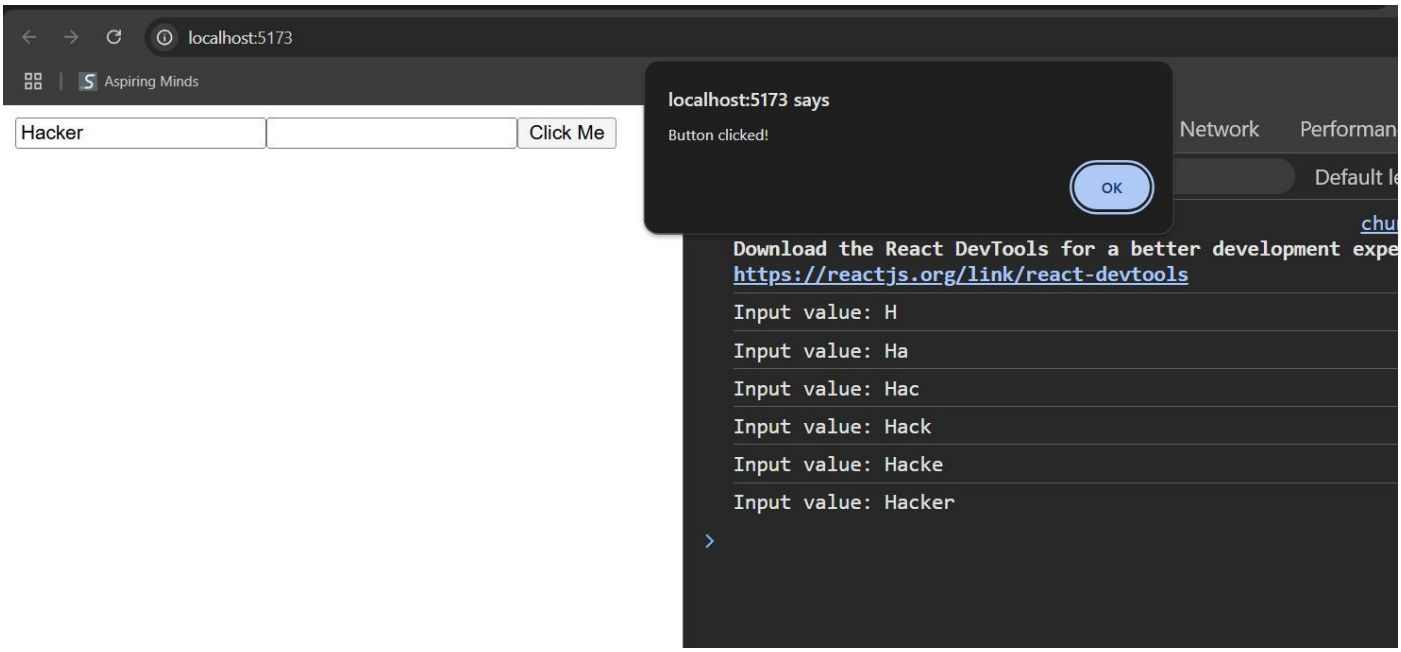
```
import React from "react";

const App = () => {
  function handleClick() {
    alert("Button clicked!");
  }

  function handleInputChange(event) {
    console.log("Input value:", event.target.value);
  }

  function handleFocus() {
    console.log("Input focused");
  }

  return (
    <>
      <input type="text" onChange={handleInputChange} />
      <input type="text" onFocus={handleFocus} />
```

```
    <button onClick={handleClick}>Click Me</button>
   </>
  );
};

export default App;
```

//Main.jsx

```
import { StrictMode } from "react";
import { createRoot } from "react-dom/client";

import App from "./App.jsx";

createRoot(document.getElementById("root")).render(
  <StrictMode>
    <App />
  </StrictMode>
);
```

OUTPUT:-



Practical No. 12

AIM:- : Create an application in ReactJSform and add client and server side validation

DISCRIPTION:-

React forms handle user input through controlled components, where form elements' values are bound to state. Validation can be added using functions that check for specific conditions (e.g., email format, required fields). Server-side validation is also performed by sending the form data to a backend server. Libraries like Formik and React Hook Form simplify form handling and validation.

CODE:-

//FormValidation.jsx

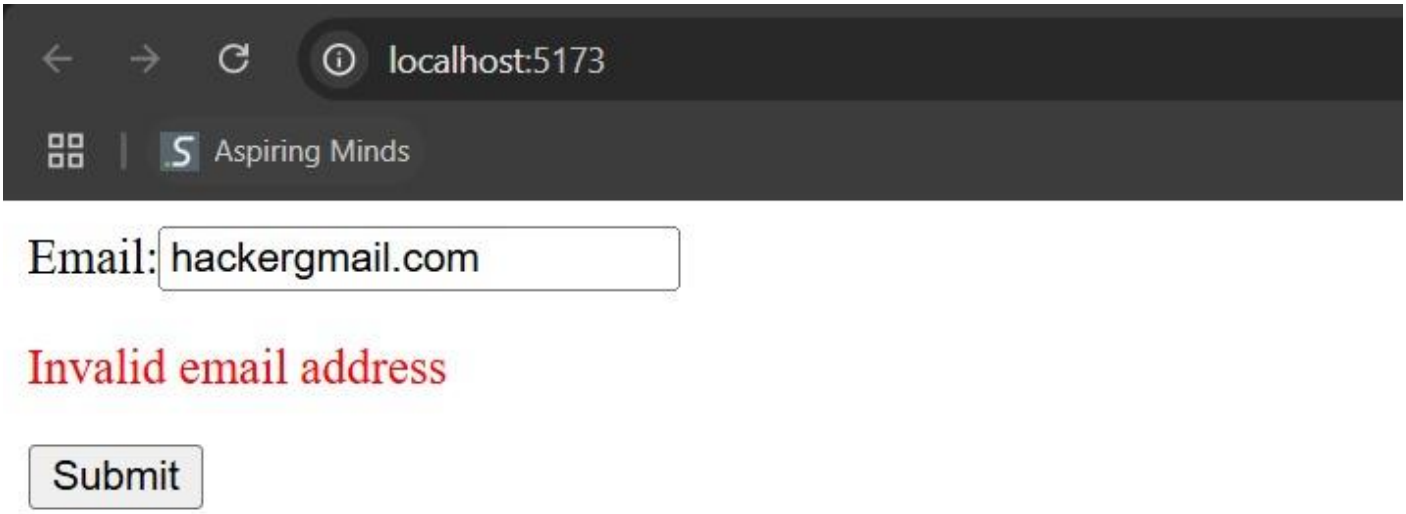
```
import { useState } from "react";

const FormValidation = () => {
  const [email, setEmail] = useState("");
  const [error, setError] = useState("");

  const validateEmail = (email) => {
    const regex = /^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$/g;
    return regex.test(email);
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    if (!validateEmail(email)) {
      setError("Invalid email address");
    } else {
      setError("");
      alert("Form submitted successfully");
    }
  };

  return (
    <form onSubmit={handleSubmit}>
      <label>Email:</label>
      <input
```

```
      type="text"
      value={email}
      onChange={(e) => setEmail(e.target.value)}
     />
     {error && <p style={{ color: "red" }}>{error}</p>}
     <button type="submit">Submit</button>
   </form>
  );
};

export default FormValidation;
```

//Main.jsx

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import FormValidation from './FormValidation.jsx'

createRoot(document.getElementById('root')).render(
  <StrictMode>
    <FormValidation />
  </StrictMode>,
)
```

OUTPUT:-



Practical No. 13

AIM:- : Create an application to implement React Hooks

DISCRIPTION:-

React Hooks, introduced in React 16.8, enable state and lifecycle management in functional components. Common hooks include `useState` for managing state, `useEffect` for side effects, and `useContext` for context management. Hooks simplify React development by eliminating the need for class components while offering a cleaner, more concise syntax.

CODE:-

//HooksDemo.jsx

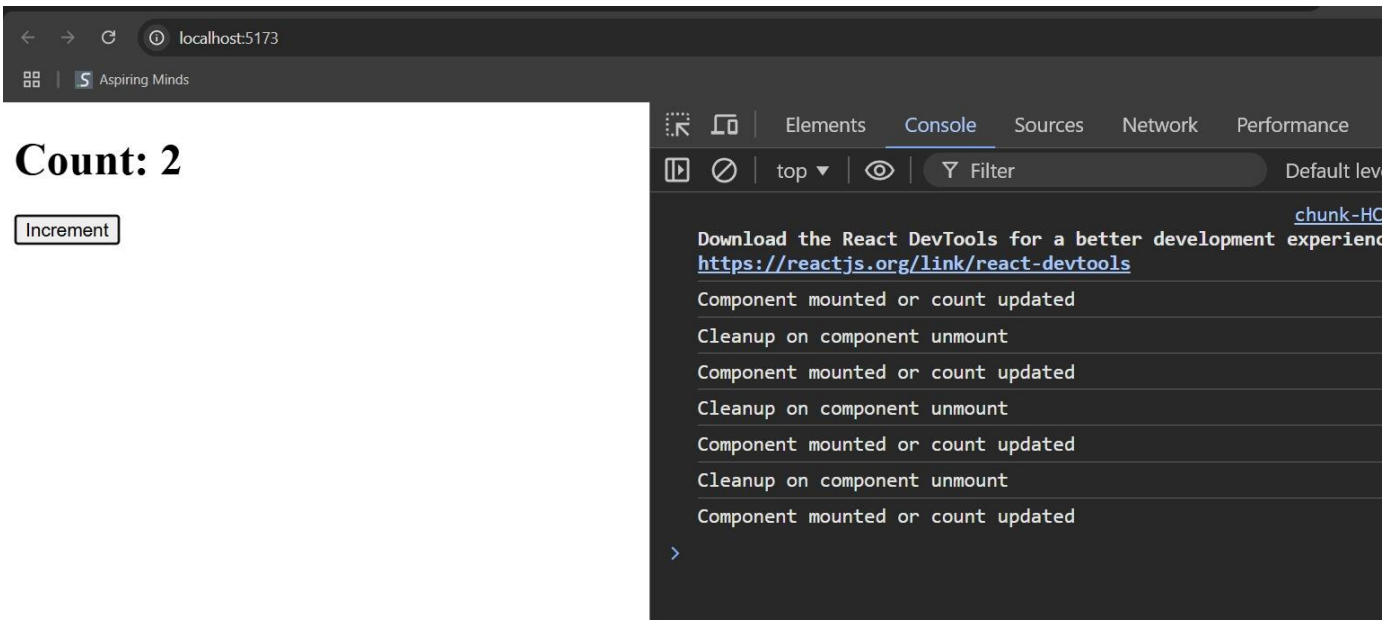
```
import { useState, useEffect } from "react";

const HooksDemo = () => {
  const [count, setCount] = useState(0);

  useEffect(() => {
    console.log("Component mounted or count updated");
    return () => {
      console.log("Cleanup on component unmount");
    };
  }, [count]);

  return (
    <div>
      <h1>Count: {count}</h1>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
};

export default HooksDemo;
```

//Main.jsx

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
```

```
import HooksDemo from './HooksDemo.jsx'

createRoot(document.getElementById('root')).render(
  <StrictMode>
    <HooksDemo />
  </StrictMode>,
)
```

OUTPUT:-



Practical No. 14

AIM:- : Create SPA using React Router

DISCRIPTION:-

Single Page Applications (SPAs) using React Router enable seamless navigation without reloading the page. The `react-router-dom` package provides components like `BrowserRouter`, `Route`, and `Link` to create routes and manage navigation. SPAs improve performance and user experience by dynamically rendering content based on the route without server-side page reloading.

CODE:-

npm install react-router-dom

//Home.jsx

```
const Home = () => {
  return (
    <div>
      <h1>Welcome to Home Page</h1>
      <p>This is the home page of our single-page application.</p>
    </div>
  );
};

export default Home;
```

//About.jsx

```
const About = () => {
  return (
    <div>
      <h1>Welcome to About Page</h1>
      <p>This is the About page of our single-page application.</p>
    </div>
  );
};

export default About;
```

//Contact.jsx

```
const Contact = () => {
  return (
    <div>
      <h1>Contact Us</h1>
      <p>Feel free to contact us via the form below.</p>
    </div>
  );
};

export default Contact;
```

//App.jsx

```
import { BrowserRouter as Router, Routes, Route, Link } from "react-router-dom";
import Home from "./components/Home";
import About from "./components/About";
import Contact from "./components/Contact";

const App = () => {
  return (
    <Router>
      <nav>
        <ul>
          <li>
            <Link to="/">Home</Link>
          </li>
          <li>
            <Link to="/about">About</Link>
          </li>
          <li>
            <Link to="/contact">Contact</Link>
          </li>
        </ul>
      </nav>

      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
        <Route path="/contact" element={<Contact />} />
```

```
    </Routes>
   </Router>
  );
};

export default App;
```

//Main.jsx

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import App from './App.jsx'

createRoot(document.getElementById('root')).render(
  <StrictMode>
    <App />
  </StrictMode>,
)
```

OUTPUT:-