

# Thadomal Shahani Engineering College

Bandra (W.), Mumbai - 400 050.

## ❧ CERTIFICATE ❧

Certify that Mr./Miss Soumil Vivek Salvi  
of IT Department, Semester V with  
Roll No. 104 has completed a course of the necessary  
experiments in the subject Internet Programming under my  
supervision in the **Thadomal Shahani Engineering College**  
Laboratory in the year 2023 - 2024












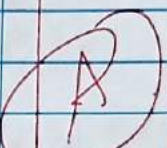
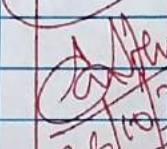
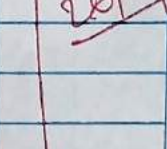
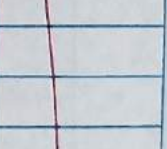
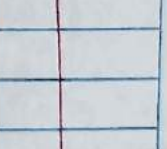
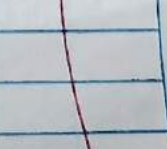
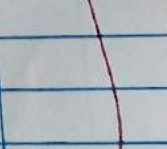
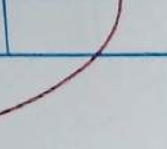

Teacher In- Charge

Head of the Department

Date 26/10/23

Principal

## CONTENTS

SR. NO.	EXPERIMENTS	PAGE NO.	DATE	TEACHERS SIGN.
1)	Develop a web application by using HTML Tags		25/7/23	          
2)	Using CSS and CSS3 enhance the web application developed in 1		28/7/23	
3)	Develop a webpage using the Bootstrap framework.		4/8/23	
4)	a) WAP in JS to study conditional statements, Loops and Function.		11/8/23	
	b) WAP on inheritance, Iterators & Generators		18/8/23	
5)	a) WAP to study arrow functions, DOM Manipulation & CSS Manipulations		18/8/23	
	b) Write a JavaScript program			
	i) Implement the concept of promise			
	ii) Fetch			
	iii) Asynchronous			
6)	a) WAP to implement the concept of props & state.		25/8/23	          
	b) WAP to implement the concept of forms and events.			
7)	a) WAP to implement ReactJS Router & Animation.		6/10/23	
	b) WAP to implement the concept of React Hooks.			
8)	Assignment on REPL		1/9/23	
9)	Write a react program to implement the concept of Asynchronous programming using promises.		20/10/23	
10)	Write a program in Node JS to		1/9/23	
	a) Create a file			
	b) Read the data from file.			
	c) Write the data to a file			



[illegible]

# ASSIGNMENT 1

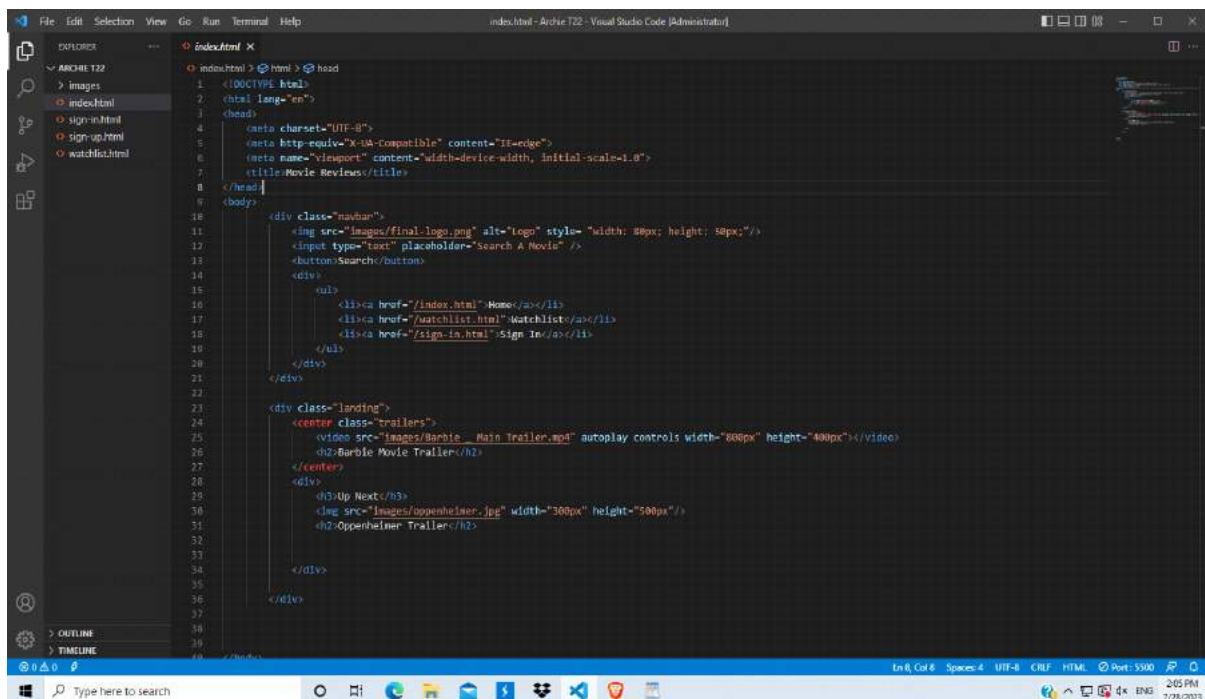
**AIM :** To design web page using html tags

**LO MAPPED :** LO 1- Identify and apply appropriate HTML tags to develop a webpage.

**THEORY:**

Code-

1. Home Page:-



```
index.html - Archive T22 - Visual Studio Code [Administrator]
File Edit Selection View Go Run Terminal Help
index.html X
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>Movie Reviews</title>
8 </head>
9 <body>
10   <div class="navbar">
11     
12     <input type="text" placeholder="Search A Movie" />
13     <button>Search</button>
14     <div>
15       <ul>
16         <li><a href="/index.html" >Home</a></li>
17         <li><a href="/watchlist.html" >Watchlist</a></li>
18         <li><a href="/sign-in.html" >Sign In</a></li>
19       </ul>
20     </div>
21   </div>
22
23   <div class="landing">
24     <center class="trailers">
25       <video src="images/Barbie_Main Trailer.mp4" autoplay controls width="800px" height="400px"></video>
26       <h2>Barbie Movie Trailer</h2>
27     </center>
28     <div>
29       <h3>Up Next</h3>
30       
31       <h2>Oppenheimer Trailer</h2>
32     </div>
33   </div>
34
35 </div>
36
37
38
39
40 </body>
```

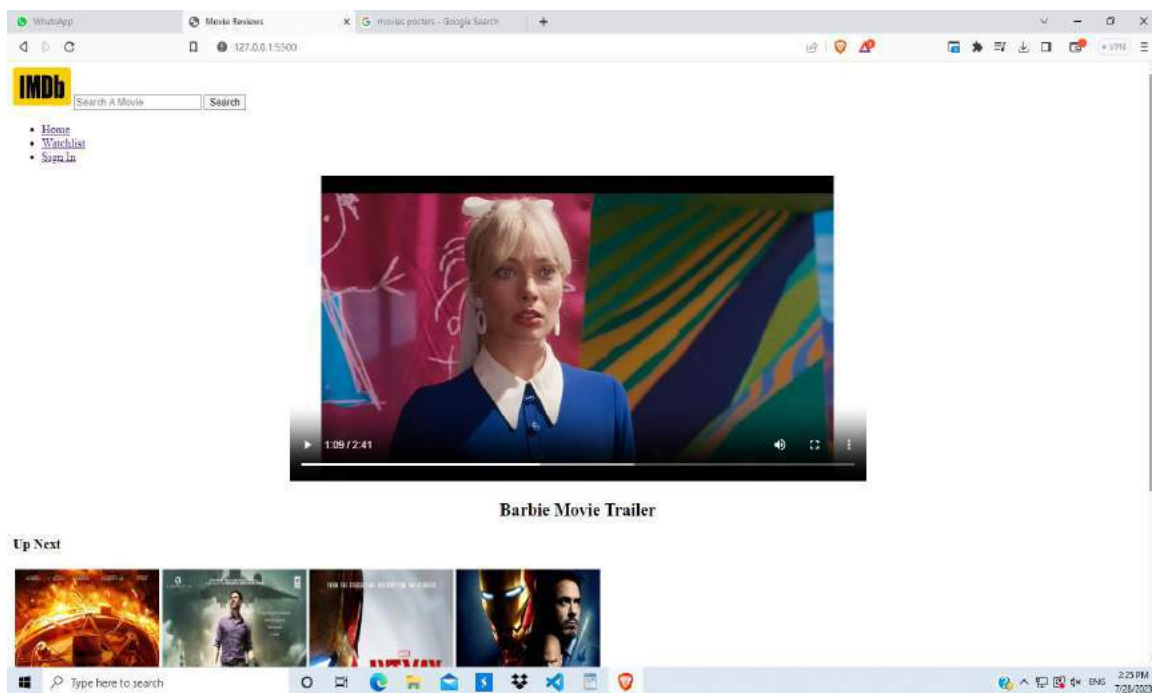
```
File Edit Selection View Go Run Terminal Help
sign-in.html - Apache T22 - Visual Studio Code (Administrator)

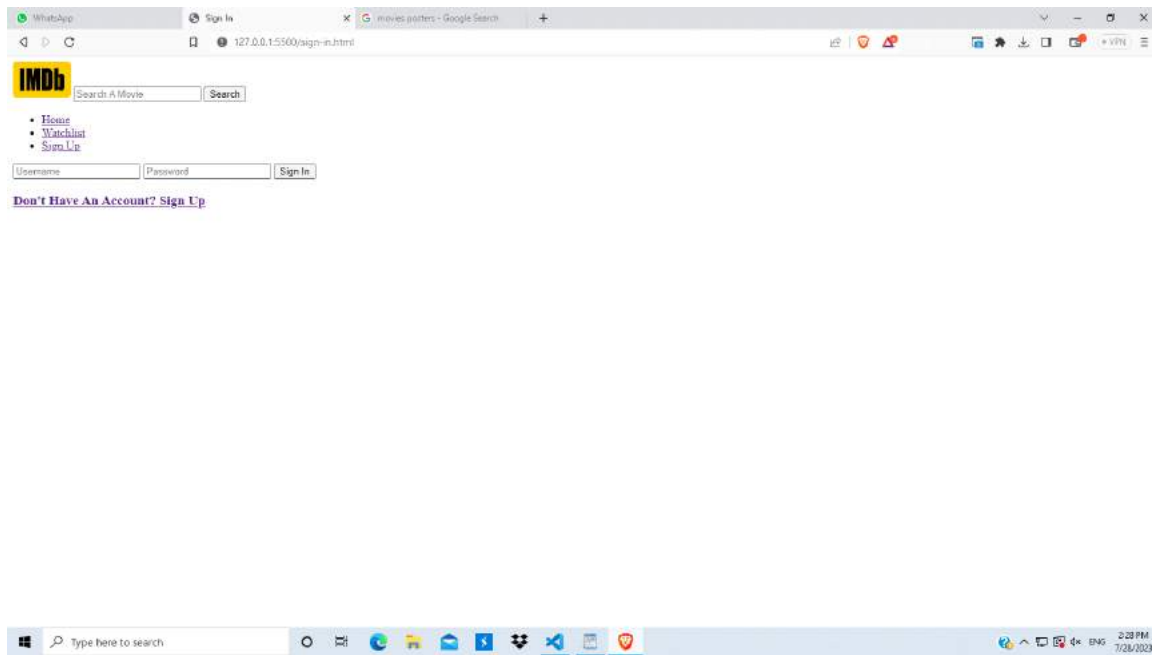
EXPLORER
ARCHIVE...
  images
  index.html
  sign-in.html
  sign-up.html
  watchlist.html

OUTLINE
TIMELINE

sign-in.html X
  1 <!DOCTYPE html>
  2 <html lang="en">
  3 <head>
  4   <meta charset="UTF-8">
  5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
  6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
  7   <title>Sign In</title>
  8 </head>
  9 <body>
 10   <div class="navbar">
 11     
 12     <input type="text" placeholder="Search A Movie" />
 13     <button>Search</button>
 14   </div>
 15   <ul>
 16     <li><a href="/index.html">Home</a></li>
 17     <li><a href="/watchlist.html">Watchlist</a></li>
 18     <li><a href="/sign-up.html">Sign Up</a></li>
 19   </ul>
 20 </div>
 21
 22   <div class="form">
 23     <input type="text" placeholder="Username" />
 24     <input type="password" placeholder="Password" />
 25     <button>Sign In</button>
 26     <h3><a href="/sign-up.html">Don't Have An Account? Sign Up</a></h3>
 27   </div>
 28 </body>
 29 </html>
```

## Web Page:-





## CONCLUSION:

The above web page is created by using the following HTML tags:-

1. <nav>
2. <ul> , <li>
3. <img /> , <video>
4. <input/> , <button> , <form>
5. <center>
6. <table> , <tr> , <td>
7. <h2> , <h3>
8. <a >



## ASSIGNMENT 2

**AIM :** To enhance web page using CSS tags

**LO MAPPED :** LO 2- Identify and apply appropriate CSS tags to format data on webpage.

**THEORY:**

Code-

1. Home Page:-

```
C:\Users\Vivek Salvi\Desktop\IP Lab> index.html > html > head
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <link rel="stylesheet" type="text/css" href="C:\Users\Vivek Salvi\Desktop\IP Lab\style
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Apple</title>
8  </head>
9  <body>
10     <div class="topnav">
11
12         <a class="logo">Home</a>
15         <a href="C:\Users\Vivek Salvi\Desktop\IP Lab\account.html">Account</a>
16         <a href="C:\Users\Vivek Salvi\Desktop\IP Lab\iphone.html">Buy</a>
17         <input type="text" placeholder="Search..">
18     </div>
19
20
```

```
22     <hr><hr>
23     <h1 style="background-color: #b7b5b5;">
24         <span style="color: #0a0a0a">Store.</span>
25         <span style="color: #5d07e7">The best way to buy the products you love.</span>
26     </h1>
27     <br><br>
28     <div class="elements">
29         <div class="elements" id="item1">
30             Macbook</p>
32         </div>
33         <div class="elements" id="item2">
34             <a href="C:\Users\Vivek Salvi\Desktop\IP Lab\iphone.html">
38             <img src="C:\Users\Vivek Salvi\Desktop\IP Lab\479-4795158_ipad-pro-3rd-generation-hd-pi
39             <p>Ipad</p>
40         </div>
41     <br>
```

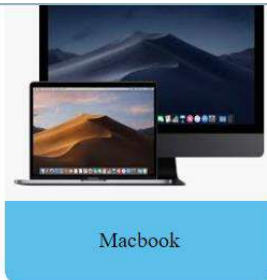
Web Page:-

[Home](#)[Account](#)[Buy](#)

**Store. The best way to buy the products you love.**



**The Latest. Take a look at what's new right now.**




**The Latest. Take a look at what's new right now.**



More ways to shop: [Find an Apple Store](#) or [other retailer](#) near you. Or call 1-800-MY-APPLE.  
Copyright © 2023 Apple Inc. All rights reserved.



 [Home](#) [Account](#) [Buy](#)

---

## Create an Apple Id

Academic Information

Degree

Master of Business Administration ▾

Student ID

Personal Details

First Name:

Last Name:

Gender:

☐ Male ☐ Female

Email Id:

Password:

Send Form

Clear Form

### What is Apple ID?

An Apple ID is the personal account you use to access Apple services like the App Store, iCloud, Messages, the

## CONCLUSION:

The above web page is created by using the following HTML tags:-

1. <nav>
2. color
3. <img />, <video>
- 4, <input/> , <button> , <form>
5. <div>
6. <table>, <tr> , <td>
7. <h2> , <h3>
8. <a >

## ASSIGNMENT 3

**AIM :** Develop a web page using the Bootstrap framework

**LO MAPPED :** LO 3- Construct responsive websites using Bootstrap

**THEORY:**

Code-

1. Home Page:-

```
index.html X style.css account.html iphone.html
C:\Users\Vivek Salvi\Desktop> IP Lab > index.html > html > body > div.topnav > a.topnav.btn-secondary.dropdown-toggle
12 <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.min.js" integrity="sha384-364-364-364-364-364-364-364-364-364-364-364-364" crossorigin="anonymous"></script>
13 <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
14 </head>
15 <body>
16 <div class="topnav">
17
18 <a class="logo"></a>
19 <a class="active" href="#home">Home</a>
20 <a href="C:\Users\Vivek Salvi\Desktop\IP Lab\account.html">Account</a>
21 <a href="C:\Users\Vivek Salvi\Desktop\IP Lab\iphone.html">Buy</a>
22 <a class="topnav btn-secondary dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown">
23   Dropdown link
24 </a>
25
26 <ul class="dropdown-menu">
27 <li><a class="dropdown-item" href="#">Visit Store</a></li>
28 <li><a class="dropdown-item" href="#">Set Location</a></li>
29 <li><a class="dropdown-item" href="#">Consult Expert</a></li>
30 </ul>
31
32 </div>
33 </body>
34 </html>
```

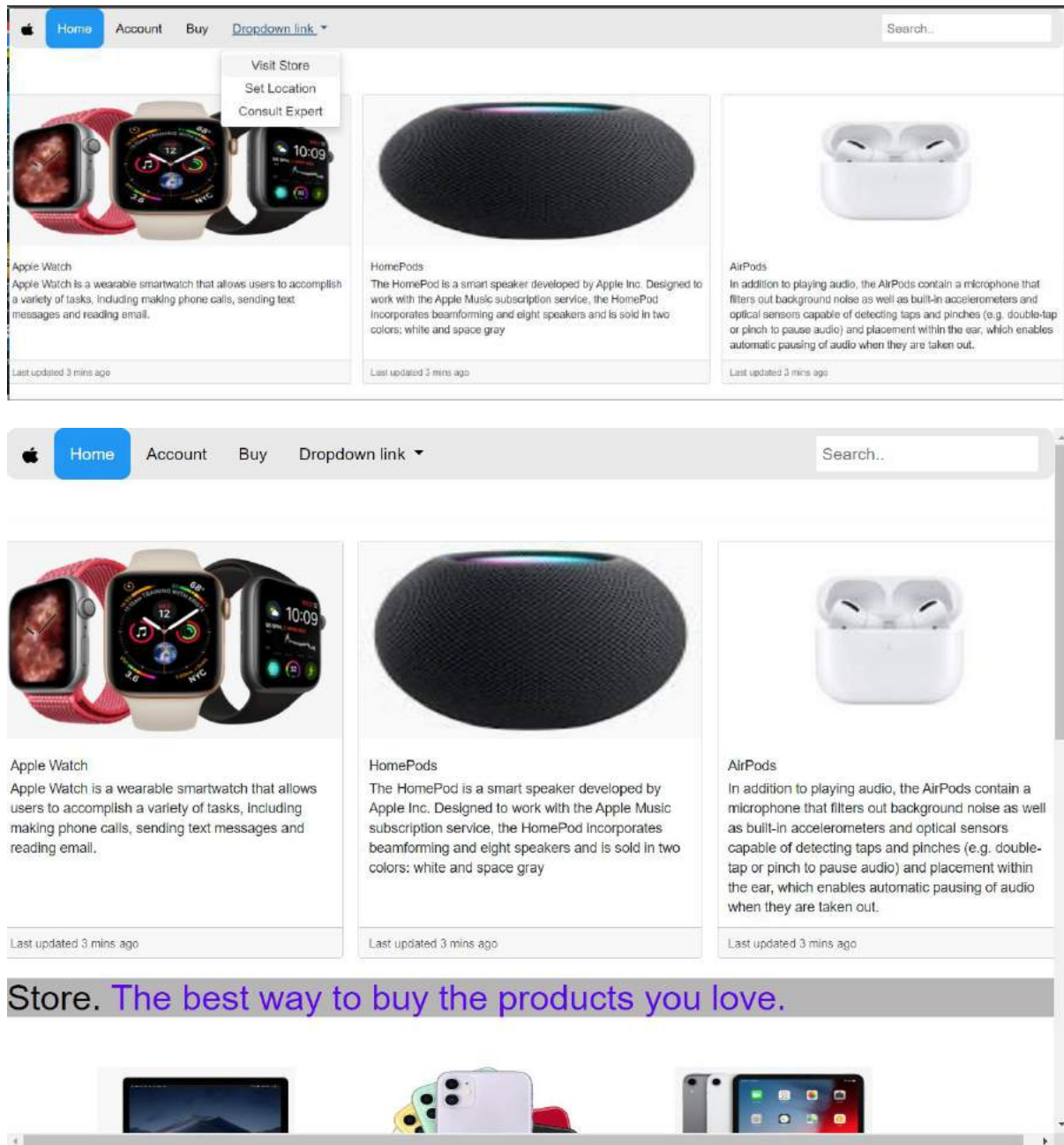
```
index.html X style.css account.html iphone.html
C:\Users\Vivek Salvi\Desktop> IP Lab > index.html > html > body > div.topnav > ul.dropdown-menu > li > a.dropdown-item
45 <div class="row row-cols-1 row-cols-md-3 g-4">
46 <div class="col">
47 <div class="card h-100">
48 
49 <div class="card-body">
50 <h5 class="card-title">Apple Watch</h5>
51 <p class="card-text">Apple Watch is a wearable smartwatch that allows users to make calls, send messages, and track their health.
52 </p>
53 <div class="card-footer">
54 <small class="text-body-secondary">Last updated 3 mins ago</small>
55 </div>
56 </div>
57 </div>
58 <div class="col">
59 <div class="card h-100">
60 
61 <div class="card-body">
62 <h5 class="card-title">HomePods</h5>
63 <p class="card-text">The HomePod is a smart speaker developed by Apple Inc. It features a high-quality audio system and is compatible with
64 </p>
65 </div>
66 </div>
67 </div>
68 </div>
69 </div>
70 </div>
71 </div>
72 </div>
73 </div>
74 </div>
75 </div>
76 </div>
77 </div>
78 </div>
79 </div>
80 </div>
81 </div>
82 </div>
83 </div>
84 </div>
85 </div>
86 </div>
87 </div>
88 </div>
89 </div>
90 </div>
91 </div>
92 </div>
93 </div>
94 </div>
95 </div>
96 </div>
97 </div>
98 </div>
99 </div>
100 </div>
```

```
index.html X # style.css account.html iphone.html
C:\Users\Vivek Salvi\Desktop> IP Lab > index.html > html > body > div.topnav > ul.dropdown-menu > li > a.dropdown-item
63     <p class="card-text">The HomePod is a smart speaker developed by Apple Inc. I
64     </div>
65     <div class="card-footer">
66     <small class="text-body-secondary">Last updated 3 mins ago</small>
67     </div>
68     </div>
69 </div>
70 <div class="col">
71     <div class="card h-100">
72     
74     <h5 class="card-title">AirPods</h5>
75     <p class="card-text">In addition to playing audio, the AirPods contain a mic
76     </div>
77     <div class="card-footer">
78     <small class="text-body-secondary">Last updated 3 mins ago</small>
79     </div>
80     </div>
81 </div>
82 </div>
```

```
index.html X # style.css account.html iphone.html
C:\Users\Vivek Salvi\Desktop> IP Lab > index.html > html > body > div.row.row-cols-1.row-cols-md-3.g-4 > div.col > div.card.h-100.
129 <div id="myCarousel" class="carousel slide" data-ride="carousel">
130 <!-- Indicators -->
131 <ol class="carousel-indicators">
132 <li data-target="#myCarousel" data-slide-to="0" class="active"></li>
133 <li data-target="#myCarousel" data-slide-to="1"></li>
134 <li data-target="#myCarousel" data-slide-to="2"></li>
135 </ol>
136
137 <!-- Wrapper for slides -->
138 <div class="carousel-inner">
139 <div class="item active">
140 
142
143 <div class="item">
144 
146
147 <div class="item">
148 <img src="C:\Users\Vivek Salvi\Desktop\IP Lab\479-4795158 ipad-pro-3rd-generat
```



Web Page:-





## CONCLUSION:

The above web page is created by using the following Bootstrap Framework:-

1. Grid System
2. Forms
3. Button
4. Navbar
5. Breadcrumb
6. Jumbotron

## ASSIGNMENT - 4a

**AIM :** WAP in JS to study conditional statements , Loops and Functions

**LO MAPPED :** LO 4 – Use JavaScript to develop interactive web pages

**THEORY:**

Code :

```
JS js-1.js JS // Conditional Statements Untitled-1 JS js-2.js Workspace Trust
1 // Conditional Statements
2 let num = 10;
3
4 if (num > 0) {
5     console.log("Number is positive");
6 } else if (num < 0) {
7     console.log("Number is negative");
8 } else {
9     console.log("Number is zero");
10 }
11
12 // Loops
13 console.log("Counting from 1 to 5 using a for loop:");
14 for (let i = 1; i <= 5; i++) {
15     console.log(i);
16 }
17
18 console.log("Counting from 5 to 1 using a while loop:");
19 let counter = 5;
20 while (counter >= 1) {
21     console.log(counter);
22     counter--;
23 }
24
25 // Functions
26 function add(a, b) {
27     return a + b;
28 }
29
30 function multiply(a, b) {
31     return a * b;
32 }
33
34 console.log("Sum of 3 and 4:", add(3, 4));
```



```

21     console.log(counter);
22     counter--;
23 }
24
25 // Functions
26 function add(a, b) {
27     return a + b;
28 }
29
30 function multiply(a, b) {
31     return a * b;
32 }
33
34 console.log("Sum of 3 and 4:", add(3, 4));
35 console.log("Product of 3 and 4:", multiply(3, 4));
36

```

Output :

```

Number is positive
Counting from 1 to 5 using a for loop:
1
2
3
4
5
Counting from 5 to 1 using a while loop:
5
4
3
2
1
Sum of 3 and 4: 7
Product of 3 and 4: 12

```

## CONCLUSION:

In this assignment we learnt about conditional statements , Loops and Functions in JavaScript.

## ASSIGNMENT – 4b

**AIM :** WAP on Inheritance , Iterators and Generators

**LO MAPPED :** LO 4 – Use JavaScript to develop interactive web pages

**THEORY:**

Code : Inheritance

```
js js-1.js js-2.js X Workspace Trust
C:\Users\> Vivek Salvi > AppData > Local > Temp > Temp1_soumil_IP_104.zip > soumil_IP_104 > js-2.js > Animal

1 class Animal {
2   constructor(name) {
3     this.name = name;
4   }
5
6   speak() {
7     console.log(`${this.name} makes a sound.`);
8   }
9 }
10
11
12 class Dog extends Animal {
13   constructor(name, breed) {
14     super(name);
15     this.breed = breed;
16   }
17 }
```

```
js js-1.js js-2.js X Workspace Trust
C:\Users\> Vivek Salvi > AppData > Local > Temp > Temp1_soumil_IP_104.zip > soumil_IP_104 > js-2.js > ...

23
24 const animal = new Animal('Generic Animal');
25 const dog = new Dog('Buddy', 'Golden Retriever');
26
27
28 function* iterateNames(collection) {
29   for (let item of collection) {
30     yield item.name;
31   }
32 }
33
34 const animals = [animal, dog];
35 const namesIterator = iterateNames(animals);
36
37 for (let name of namesIterator) {
38   console.log(name);
39 }
```

Output :

Generic Animal  
Buddy

Iterators :

Code :

```
JS js-1.js JS js-2.js JS Iterator.js • Workspace Trust
C: > Users > Vivek Salvi > AppData > Local > Temp > Temp1_soumil_IP_104.zip > soumil_IP_104 > JS Iterator.js > makeRangeIterator > rangeIterator > next
1 function makeRangeIterator(start = 0, end = Infinity, step = 1) {
2     let nextIndex = start;
3     let iterationCount = 0;
4
5     const rangeIterator = {
6         next() {
7             let result;
8             if (nextIndex < end) {
9                 result = { value: nextIndex, done: false };
10                nextIndex += step;
11                iterationCount++;
12                return result;
13            }
14            return { value: iterationCount, done: true };
15        },
16    };
17    return rangeIterator;
18 }
```

```
17     return rangeIterator;
18 }
19 const it = makeRangeIterator(1, 10, 2);
20
21 let result = it.next();
22 while (!result.done) {
23     console.log(result.value);
24     result = it.next();
25 }
26
27 console.log("Iterated over sequence of size:", result.value);
28
29
```

Output :



```
1
3
5
7
9
Iterated over sequence of size: 5
```

Generators :

Code :

```
function* generatorFunc() {
  let x = yield 'hello';
  console.log(x);
  console.log('Hii');
  yield 5;
}
const generator = generatorFunc();
console.log(generator.next());
console.log(generator.next(6));
console.log(generator.next());
```

Output :

```
{ value: 'hello', done: false }
6
Hii
{ value: 5, done: false }
{ value: undefined, done: true }
```

## CONCLUSION:

In this assignment we learnt about Inheritance , Iterators and Generators in JavaScript.

## ASSIGNMENT 5-a

**AIM :** Write a Javascript Program to study arrow functions , DOM Manipulation and CSS Manipulations .

**LO MAPPED : LO4**

**THEORY :**

**ARROW FUNCTIONS :**

Arrow function `{()=>>}` is concise way of writing Javascript functions in shorter way. Arrow functions were introduced in the ES6 version. They make our code more structured and readable.

Arrow functions are anonymous functions i.e. functions without a name and are not bound by an identifier. Arrow functions do not return any value and can be declared without the function keyword. They are also called Lambda Functions.

Syntax:

```
const gfg = () => {  
    console.log( "Hi Geek!" );  
}
```

**DOM MANIPULATION :**

The DOM in dom manipulation in javascript stands for Document Object Model. The Document Object Model (DOM) is a tree-like structure illustrating the hierarchical relationship between various HTML elements. It can be easily explained as a tree of nodes generated by the browser. Each node has unique properties and methods that can be changed using JavaScript.

DOM manipulation in javascript is an important factor while creating a web application using HTML and JavaScript. It is the process of interacting with the DOM API to change or modify an HTML document that will be displayed in a web browser. This HTML document can be changed to add or remove elements, update existing elements, rearrange existing elements, etc.

By manipulating the DOM, we can create web applications that update the data in a web page without refreshing the page and can change its layout

without doing a refresh. Throughout the document, items can be deleted, moved, or rearranged.

## CSS MANIPULATIONS :

The common manipulation of classes includes actions like adding a class or removing a class from the HTML tags.

The following classes are used for the manipulations.

`addClass()`

`removeClass()`

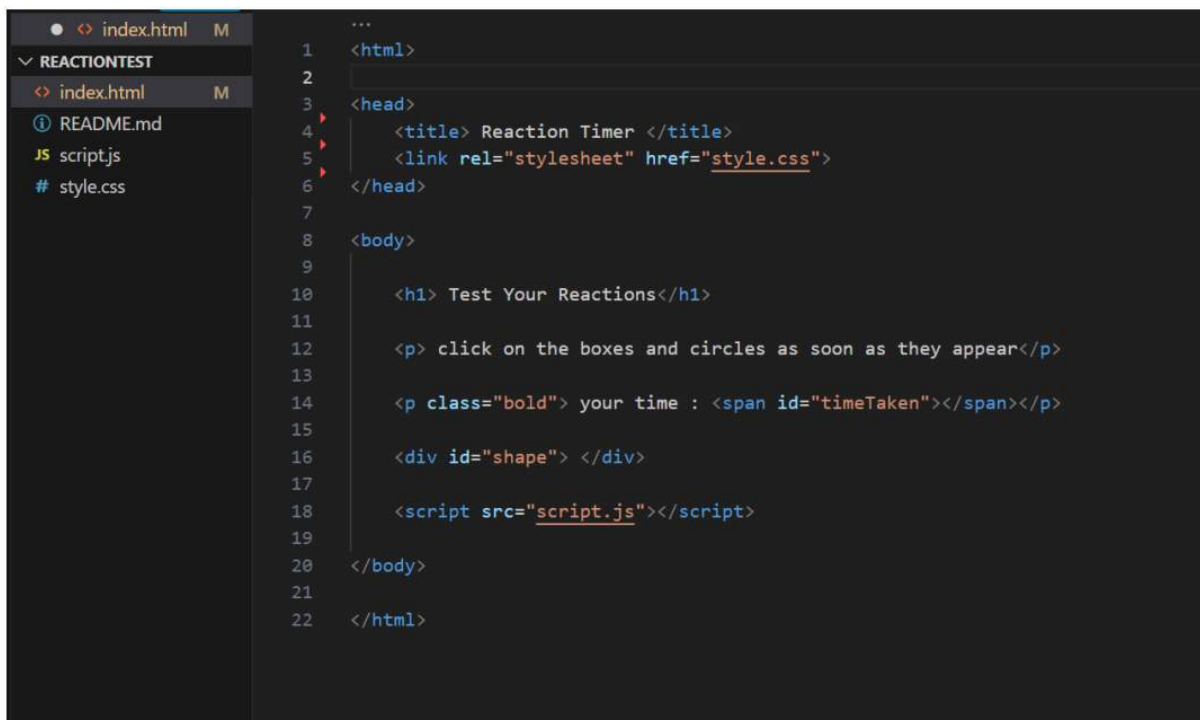
`toggleClass()`

`addClass()` method: The purpose of the `addClass()` function is to add one or more classes to the specified element or tag.

Syntax:

```
$('.selector expression').addClass('class name');
```

## CODE :



```
1  <html>
2
3  <head>
4    <title> Reaction Timer </title>
5    <link rel="stylesheet" href="style.css">
6  </head>
7
8  <body>
9
10   <h1> Test Your Reactions</h1>
11
12   <p> click on the boxes and circles as soon as they appear</p>
13
14   <p class="bold"> your time : <span id="timeTaken"></span></p>
15
16   <div id="shape"> </div>
17
18   <script src="script.js"></script>
19
20 </body>
21
22 </html>
```



```
× JS script.js
▼ REACTIONTEST
  <> index.html M
  ⓘ README.md
  JS script.js
  # style.css

4   var letters = '0123456789ABCDEF';
5   var color = '#';
6   for (var i = 0; i < 6; i++) {
7     color += letters[Math.floor(Math.random() * 16)];
8   }
9   return color;
10  }
11
12  function makeShapeAppear() {
13
14    var top = Math.random() * 400;
15
16    var left = Math.random() * 400;
17
18    var width = (Math.random() * 200) + 100;
19
20    if (Math.random() > 0.5) {
21
22      document.getElementById("shape").style.borderRadius = "50%";
23
24    } else {
25
26      document.getElementById("shape").style.borderRadius = "0"
27    }
28
29    document.getElementById("shape").style.backgroundColor = getRandomColor();
30
31    document.getElementById("shape").style.width = width + "px";
32  }
```

```
...
1  body {
2
3    font-family: sans-serif;
4
5  }
6
7  #shape {
8
9    height: 200px;
10   width: 200px;
11   background-color: red;
12   display: none;
13   position: relative;
14 }
15
16 .bold {
17
18   font-weight: bold;
19 }
```

OUTPUT :

## Test Your Reactions

click on the boxes and circles as soon as they appear

your time : 1.196s



---

## Test Your Reactions

click on the boxes and circles as soon as they appear

your time : 31.069s



**CONCLUSION :** We understood and Implemented the concept of arrow functions , DOM Manipulation and CSS Manipulations and executed it successfully.

## ASSIGNMENT 5-b

**AIM :** Write a Javascript Program

- Implement the concept of Promises
- Fetch( Client Server Communication )
- Asynchronous Javascript

**LO MAPPED : LO4**

**THEORY :**

**PROMISES :**

Promises simplify deferred and asynchronous computations. A promise represents an operation that hasn't completed yet.

A Promise is in one of these states:

- pending: initial state, neither fulfilled nor rejected.
- fulfilled: meaning that the operation was completed successfully.
- rejected: meaning that the operation failed.

**FETCH :**

The `fetch()` method in JavaScript is used to request data from a server. The request can be of any type of API that returns the data in JSON or XML. The `fetch()` method requires one parameter, the URL to request, and returns a promise.

Syntax:

```
fetch('url')      //api for the get request
  .then(response => response.json())
  .then(data => console.log(data));
```

**ASYNCHRONOUS JAVASCRIPT :**

The `async` function declaration creates a binding of a new `async` function to a given name. The `await` keyword is permitted within the function body, enabling

asynchronous, promise-based behaviour to be written in a cleaner style and avoiding the need to explicitly configure promise chains.

You can also define async functions using the async function expression.

Async simply allows us to write promises-based code as if it was synchronous and it checks that we are not breaking the execution thread. It operates asynchronously via the event loop. Async functions will always return a value. It makes sure that a promise is returned and if it is not returned then JavaScript automatically wraps it in a promise which is resolved with its value.

CODE :

```
1  <!DOCTYPE html>
2  <html Lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <meta name="viewport" content="width=device-width, initial-scale=1.0">
6    <title>Async JavaScript Example</title>
7  </head>
8  <body>
9    <h1>Async JavaScript Example</h1>
10   <p id="output"></p>
11   <div id="fetchData"></div>
12
13   <script>
14     // Create a Promise
15     function fetchData() {
16       return new Promise((resolve, reject) => {
17         setTimeout(() => {
18
19           const data = { message: "Data fetched successfully!" };
20           resolve(data);
21
22           | reject("Error: Unable to fetch data");
23         }, 2000);
24       });
25     }
26
27     // Fetch data asynchronously
28     async function fetchDataAsync() {
29       try {
30         const response = await fetchData();
31         document.getElementById('output').textContent = response.message;
32       } catch (error) {
33         document.getElementById('output').textContent = `Error: ${error}`;
34       }
35     }
36   </script>
37 </body>
38 </html>
```



```

1  <!DOCTYPE html>
2  <html Lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Async JavaScript Example</title>
7  </head>
8  <body>
9      <h1>Async JavaScript Example</h1>
10     <p id="output"></p>
11     <div id="fetchedData"></div>
12
13     <script>
14         // Create a Promise
15         function fetchData() {
16             return new Promise((resolve, reject) => {
17                 setTimeout(() => {
18
19                     const data = { message: "Data fetched successfully!" };
20                     resolve(data);
21
22                     reject("Error: Unable to fetch data");
23                 }, 2000);
24             });
25         }
26
27         // Fetch data asynchronously
28         async function fetchDataAsync() {
29             try {
30                 const response = await fetchData();
31                 document.getElementById('output').textContent = response.message;
32             } catch (error) {
33                 document.getElementById('output').textContent = `Error: ${error}`;
34             }
35         }
36     </script>
37 </body>
38 </html>

```

```

C:\Users\Lab1006\Desktop\soumil_IP_104>python -m http.server
Serving HTTP on :: port 8000 (http://[::]:8000/) ...
:::1 - - [01/Sep/2023 15:34:38] "GET /assignment-5b%20mydata.json HTTP/1.1" 200 -
:::1 - - [01/Sep/2023 15:35:57] "GET /assignment-5b%20mydata.json HTTP/1.1" 200 -
:::1 - - [01/Sep/2023 15:37:36] "GET /assignment-5b%20mydata.json HTTP/1.1" 200 -
:::1 - - [01/Sep/2023 15:38:00] "GET /assignment-5b%20mydata.json HTTP/1.1" 200 -
:::1 - - [01/Sep/2023 15:42:47] "GET /assignment-5b-mydata.json HTTP/1.1" 200 -
:::1 - - [01/Sep/2023 15:51:38] "GET /assignment-5b-mydata.json HTTP/1.1" 200 -
:::1 - - [01/Sep/2023 15:52:16] "GET /assignment-5b-mydata.json HTTP/1.1" 200 -
:::1 - - [01/Sep/2023 16:00:13] "GET /assignment-5b-mydata.json HTTP/1.1" 200 -

```

OUTPUT :

```

← → ↺ ⓘ File | c%3A\Users\Lab1006\Desktop\soumil_IP_104\assignmet%20-%205b.html

```

## Async JavaScript Example

Data fetched successfully!

**CONCLUSION :** We understood and Implemented the concept of promises , fetch and async javascript into one program and executed it successfully.

NAME : SOUMIL SALVI

ROLL NO : 104

## ASSIGNMENT 6-a

**AIM :** WAP to implement the concept of props and state

**LO MAPPED :** LO5

**THEORY :**

### 1. PROPS

In React, "props" is a commonly used abbreviation for "properties," and it refers to the mechanism by which data is passed from a parent component to a child component. Props allow you to make your React components reusable and dynamic by providing a way to pass data and configuration to them.

Here's a more detailed explanation of props in React:

- **Passing Data:** Props are a way to pass data from a parent component to a child component. This data can be of any JavaScript data type, such as strings, numbers, objects, or functions.
- **Immutable:** Props are read-only, which means that child components cannot modify their props. They are meant to be a one-way communication mechanism from parent to child.
- **Component Configuration:** Props are often used to configure how a child component should render. For example, you might pass a "title" prop to a child component to specify what text should be displayed.
- **Dynamic Behavior:** Props allow you to create dynamic components. By changing the props passed to a component, you can change its behavior and appearance without changing the component's code.

- Default Values: You can specify default values for props in case a parent component doesn't provide a value for a particular prop. This can be done using default props.

CODE :

```
JS reactjs > ...
1  import React from 'react';
2
3
4  function DateTimeDisplay(props) {
5    return (
6      <div>
7        <h1>Current Date and Time:</h1>
8        <p> {props.currentDateTime}</p>
9      </div>
10   );
11 }
12
13 function App() {
14   const currentDateTime = new Date().toLocaleString();
15
16   return (
17     <div className="App">
18       <DateTimeDisplay currentDateTime={currentDateTime} />
19     </div>
20   );
21 }
22
23
24 export default App;
25 |
```

OUTPUT :

# Current Date and Time:

8/30/2023, 8:12:57 PM

## 2. STATES :

In React, "state" refers to a fundamental concept that allows you to manage and store data that can change over time within a component. State is used to represent the dynamic aspects of your application, such as user interactions, input data, and other variables that can change and affect how your user interface behaves. Understanding and using state effectively is crucial when building React applications.

Here's a detailed explanation of states in React:

- i. **Component State:**
  - Each React component can have its own state, which is a JavaScript object containing data that should be saved and managed by that component.
  - State is initialized in the constructor of a class component or using the `useState` hook in a functional component.
  - State should only be used for data that will change over time and affects the component's rendering.
- ii. **State Initialization:**
  - In class components, you initialize state in the constructor using `this.state`.
  - In functional components, you use the `useState` hook to initialize state:
- iii. **Updating State:**
  - To update state, you should never modify it directly. Instead, use the `setState` method in class components or the function returned by `useState` in functional components to update state.
- iv. **Re-rendering:**
  - When you update the state using the appropriate methods, React will automatically re-render the component. React efficiently compares the new state with the previous state and updates the DOM only as needed to reflect the changes.

v. Local Component State:

- State is local to the component that owns it. It can't be accessed or modified directly by other components.
- If you need to share state between components, you can lift the state up to a common ancestor component and pass it down as props.

vi. Asynchronous State Updates:

- State updates may be asynchronous in React to optimize performance. React may batch multiple state updates into a single update for better performance. This means you shouldn't rely on the current state immediately after calling `setState`. Instead, use the callback function provided by `setState` or use the functional update pattern.

CODE :

```
1  import React, { Component } from 'react';
2
3  class ColorChanger extends Component {
4    constructor(props) {
5      super(props);
6      this.state = {
7        selectedColor: 'white', // Default color
8      };
9    }
10
11    handleColorChange = (event) => {
12      this.setState({
13        selectedColor: event.target.value,
14      });
15    };
16
17    render() {
18      const { selectedColor } = this.state;
19
20      const containerStyle = {
21        display: 'flex',
22        justifyContent: 'center',
23        alignItems: 'center',
24        minHeight: '100vh',
25        backgroundColor: selectedColor,
26        transition: 'background-color 0.3s ease',
27      };
28
29      return (
30        <div style={containerStyle}>
31          <div>
32            <h2>Color Changer</h2>
33            <label>
34              <input
35                type="radio"
36                value="red"
37                checked={selectedColor === 'red'}

```



```

37   checked={selectedColor === 'red'}
38   onChange={this.handleColorChange}
39   />
40   Red
41 </label>
42 <label>
43   <input
44     type="radio"
45     value="green"
46     checked={selectedColor === 'green'}
47     onChange={this.handleColorChange}
48   />
49   Green
50 </label>
51 <label>
52   <input
53     type="radio"
54     value="blue"
55     checked={selectedColor === 'blue'}
56     onChange={this.handleColorChange}
57   />
58   Blue
59 </label>
60 <label>
61   <input
62     type="radio"
63     value="yellow"
64     checked={selectedColor === 'yellow'}
65     onChange={this.handleColorChange}
66   />
67   Yellow
68 </label>
69 </div>
70 </div>
71 );
72 }
73 }

```

OUTPUT :



**CONCLUSION :** In this assignment, we learned how to use props to pass data from a parent component to a child component and the State to modify the state of a component.

NAME : SOUMIL SALVI

ROLL NO : 104

## **ASSIGNMENT 6-b**

**AIM :** WAP to implement the concept of forms and events

**LO MAPPED :** LO5

**THEORY :**

### **a) FORMS**

In React, forms are a fundamental part of building interactive and user-friendly web applications. Forms allow users to input data, submit it to the server, and interact with the application. React simplifies form handling by providing a component-driven approach to manage form elements and their state. Here's a theoretical overview of forms in React:

#### **1. Form Elements as Components:**

In React, form elements like input fields, checkboxes, radio buttons, and select dropdowns are treated as components. Each form element has its state, and you can create controlled components to manage and manipulate their values.

#### **2. Controlled vs. Uncontrolled Components:**

React supports both controlled and uncontrolled components. Controlled components have their value controlled by React's state, making it easier to track and manipulate the form data. Uncontrolled components, on the other hand, store their state in the DOM itself.

#### **3. State Management:**

To create a controlled component, you maintain its value in the state of a parent component and pass the value and an event handler as props to the child component. When the user interacts with the form element, React updates the state and re-renders the component.

#### 4. Event Handling:

React provides a consistent way to handle form events using functions like `onChange`, `onSubmit`, `onClick`, etc. These event handlers are used to respond to user input and update the component's state accordingly.

#### 5. Form Submission:

When a user submits a form, you can prevent the default behavior of the browser using `event.preventDefault()` in the `onSubmit` handler. This allows you to handle form submission within your React application, validate the data, and send it to the server using AJAX requests or other methods.

#### 6. Validation:

React allows you to implement client-side validation easily by checking the data in the state before submitting it. You can provide feedback to users by displaying error messages or applying CSS classes based on validation results.

#### 7. Form Libraries and Tools:

There are several third-party libraries and tools available in the React ecosystem for form handling, such as `Formik`, `Redux Form`, and `react-hook-form`. These libraries provide additional abstractions and utilities to simplify form management and validation.

#### 8. Conditional Rendering:

React makes it straightforward to conditionally render form elements based on the application's state. You can show or hide fields, sections, or entire forms based on user actions or the data entered.

## 9. Accessibility:

When building forms in React, it's essential to consider accessibility. Ensure that form elements have appropriate labels, use ARIA attributes, and provide keyboard navigation and focus management for users with disabilities.

## 10. Testing:

React's component-based structure makes it easier to write unit tests for your forms. You can use testing libraries like Jest and testing utilities provided by React to ensure that your forms work as expected.

```
13
14     <script type="text/babel">
15         // Define a React component
16         class FormComponent extends React.Component {
17             constructor(props) {
18                 super(props);
19                 this.state = {
20                     name: '',
21                     email: '',
22                     submitted: false,
23                 };
24             }
25
26             handleInputChange = (event) => {
27                 const { name, value } = event.target;
28                 this.setState({
29                     [name]: value,
30                 });
31             }
32
```

```

39
40 render() {
41   const { name, email, submitted } = this.state;
42   return (
43     <div>
44       <h1>React Form Submission</h1>
45       <form onSubmit={this.handleSubmit}>
46         <label>
47           Name:
48           <input type="text" name="name" value={name} onChange={this.handleChange} />
49         </label>
50         <br />
51         <br />
52         <label>
53           Email:
54           <input type="email" name="email" value={email} onChange={this.handleChange} />
55         </label>
56         <br />
57         <br />
58         <button type="submit">Submit</button>
59       </div>
60     );
61   }

```

OUTPUT :

React Form Submission

Name:

Email:

## b) EVENTS

In React, events are a fundamental part of building interactive user interfaces. They allow you to capture and respond to user interactions such as clicks, keyboard input, and mouse movements. React uses a synthetic event system that wraps and normalizes native browser events, making it easier to work with events in a cross-browser and consistent manner.

Here's how events work in React:



### i. Event Handling:

To handle events in React, you define event handlers as functions in your components. These event handlers are responsible for executing a specific action when a particular event occurs. Event handlers are typically defined as methods on a component class.

For example, you can define a click event handler like this:

```
class MyComponent extends React.Component {  
  handleClick() {  
    // Code to handle the click event  
  }  
  render() {  
    return <button onClick={this.handleClick}>Click me</button>;  
  }  
}
```

In this example, the handleClick method will be called when the button is clicked.

### ii. Event Binding:

When you use event handlers in React, you need to be careful about how you bind this to the handler function. There are a few ways to ensure that this refers to the correct context when the event handler is executed:

#### a. Binding in the constructor:

You can explicitly bind the event handler to the component's instance in the constructor:

```
class MyComponent extends React.Component {  
  constructor(props) {  
    super(props);
```

```

    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    // Code to handle the click event
  }

  render() {
    return <button onClick={this.handleClick}>Click me</button>;
  }
}

```

#### b. Arrow function syntax:

You can use arrow function syntax to define your event handlers. Arrow functions automatically bind this to the current component instance:

```

class MyComponent extends React.Component {
  handleClick = () => {
    // Code to handle the click event
  }

  render() {
    return <button onClick={this.handleClick}>Click me</button>;
  }
}

```

#### iii. Event Object:

When an event occurs, React passes a synthetic event object to your event handler. This event object contains information about the event, such as the target element, type of event, and any event-specific data.

```
handleClick(event) {  
  console.log(event.target); // Access the target element that triggered the  
  event  
}
```

#### iv. Preventing Default Behavior:

You can use the `event.preventDefault()` method within an event handler to prevent the default browser behavior for certain events. For example, you can prevent a form submission when a user clicks a submit button.

```
handleSubmit(event) {  
  event.preventDefault();  
  // Code to handle form submission  
}
```

CODE :

```

import React, { Component } from 'react';
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      companyName: ""
    };
  }
  changeText(event) {
    this.setState({
      companyName: event.target.value
    });
  }
  render() {
    return (
      <div>
        <h2>Simple Event Example</h2>
        <label htmlFor="name">Enter company name: </label>
        <input type="text" id="companyName" onChange={this.changeText.bind(this)} />
        <h4>You entered: { this.state.companyName } </h4>
      </div>
    );
  }
}
export default App;

```

OUTPUT :



Simple Event Example

Enter company name:

You entered:



Simple Event Example

Enter company name:

You entered: www.javatpoint.com

**CONCLUSION :** In this assignment we understood and implemented the concept of forms and events.

NAME : SOUMIL SALVI

ROLL NO : 104

## **ASSIGNMENT – 7a**

**AIM :** WAP to implement ReactJS Router and Animation.

**LO MAPPED :** LO5

### **THEORY :**

React Router:

React Router is a popular library for adding routing and navigation to React applications. It allows you to create single-page applications (SPAs) by managing the rendering of components based on the URL, enabling seamless transitions between different views or pages within your application. Here's the theory behind React Router:

- **Routing:** In a React application, routing refers to the process of determining which component or view should be displayed based on the current URL. React Router provides a way to map URLs to specific components.
- **Components:** React Router introduces special components like `BrowserRouter`, `Route`, `Link`, and `Switch` to facilitate routing. The `BrowserRouter` component establishes a context for routing, and `Route` components define which components should render for specific URL paths.
- **Route Configuration:** You define the routing configuration by specifying which components should render for different routes. Routes can include dynamic segments, query parameters, and optional route parameters.
- **Navigation:** React Router also provides components like `Link` or `NavLink` to create links that navigate to different routes. These components generate anchor tags (`<a>`) with the appropriate `href` attributes, ensuring that the URL changes when navigating.
- **Nested Routes:** React Router supports nested routing, allowing you to render components within other components. This is useful for building complex layouts with multiple nested views.



- **Programmatic Navigation:** You can programmatically navigate to different routes using the history object provided by React Router or by using the useHistory hook (for functional components).
- **Route Parameters:** You can extract route parameters from the URL using React Router, which allows you to create dynamic routes. For example, /users/:id could match both /users/1 and /users/2.
- **Route Guards:** React Router enables you to implement route guards or authentication checks before rendering specific routes. This helps control access to certain parts of your application.

### Animation in React:

Adding animations to a React application can enhance user experience and make your app more engaging. React provides various ways to implement animations. Here's the theory behind adding animations in React:

- **CSS Transitions and Animations:** React can leverage CSS for animations using CSS transitions and animations. You can define CSS classes with @keyframes animations or CSS transitions to apply animations to elements when certain conditions are met, like adding or removing CSS classes.
- **React Transition Group:** The react-transition-group library is a commonly used tool for adding animations to React components. It provides a set of components that manage the entrance and exit animations of elements when they are added or removed from the DOM. Examples include CSSTransition and TransitionGroup.
- **React Spring:** React Spring is another popular library for creating animations in React. It provides a more declarative way to define animations using JavaScript objects. With React Spring, you can animate not only CSS properties but also any numeric value, making it highly flexible.
- **React Motion:** React Motion is a physics-based animation library for React. It uses spring physics to create smooth animations and is particularly useful for complex, natural-looking animations.

- **SVG and Canvas:** For more advanced animations or custom graphics, you can use SVG (Scalable Vector Graphics) or the HTML5 Canvas API within React components to create interactive and dynamic visuals.
- **Third-Party Libraries:** Many third-party animation libraries, such as GreenSock Animation Platform (GSAP), can be used with React to create sophisticated animations and interactions.
- **Hooks and Effects:** React's `useEffect` and `useState` hooks can be used to trigger animations in response to component updates or user interactions. You can set up animations based on component state changes.
- **Performance:** When working with animations in React, it's essential to consider performance. Avoid excessive re-renders and use techniques like `shouldComponentUpdate`, `React.memo`, or `useMemo` to optimize performance when necessary.

In summary, React Router is essential for handling navigation and routing in React applications, allowing you to create SPAs with multiple views. Meanwhile, animation in React can be achieved through CSS transitions, libraries like `react-transition-group`, `react-spring`, and more, enhancing the visual appeal and interactivity of your application. Careful consideration of performance is crucial when implementing animations to ensure a smooth user experience.

CODE :

```
src > App.js > App
1  import React from 'react';
2  import ReactDOM from 'react-dom';
3  import { BrowserRouter as Router, Route, Link, Routes } from 'react-router-dom';
4  import { CSSTransition } from 'react-transition-group';
5  import './App.css';
6  import './transition.css';
7  import Home from './components/Home';
8  import About from './components/About';
9
10 const App = () => {
11   return (
12     <Router>
13       <div className="App">
14         <nav>
15           <ul>
16             <li>
17               <Link to="/">Home</Link>
18             </li>
19             <li>
20               <Link to="/about">About</Link>
21             </li>
22           </ul>
23         </nav>
24         <Routes>
25           <Route
```

```

src > JS App.js > App
19
20     <Link to="/about">About</Link>
21   </li>
22 </ul>
23 </nav>
24 <Routes>
25   <Route
26     path="/" element={<Home/>}
27   />
28   <Route
29     path="/about"
30     element={<About/>}
31   />
32 </Routes>
33 </div>
34 </Router>
35
36 );
37 }
38 ReactDOM.render(<App/>,document.getElementById("root"));
39 export default App;
40

```

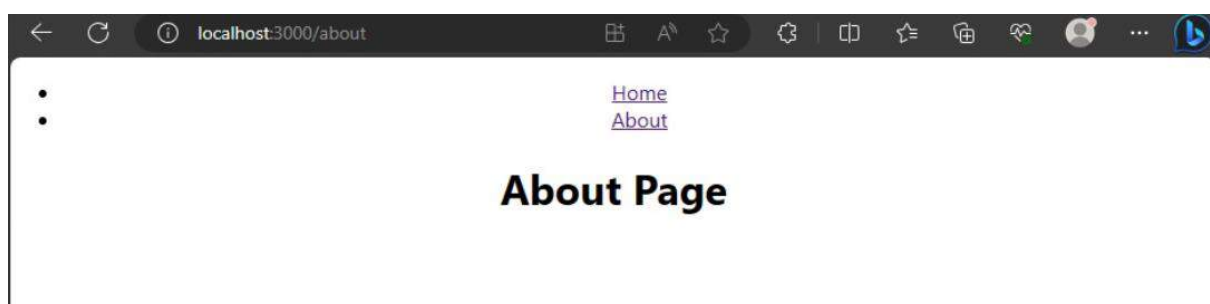
```

src > components > JS Home.js > Home
1  import React from 'react';
2
3
4  const Home = () => {
5    return (
6      <div>
7        <h1>Home Page</h1>
8      </div>
9    );
10  }
11
12  export default Home;

```

```
src > components > JS About.js > ...
1  import React from 'react';
2
3  const About = () => {
4    return (
5      <div>
6        <h1>About Page</h1>
7      </div>
8    );
9  }
10
11  export default About;
```

OUTPUT :



**CONCLUSION :** In this assignment, we learned that React Router is essential for managing routing in React applications, while animations can be added for enhanced user engagement and interactivity, utilizing various techniques and libraries to create dynamic user experiences.

NAME : SOUMIL SALVI

ROLL NO : 104

## ASSIGNMENT – 7b

**AIM :** WAP to implement the concept of React Hooks.

**LO MAPPED :** LO5

### **THEORY :**

React Hooks are a powerful addition to the React library that allows developers to manage state, side effects, and other React features in functional components. To understand the theory behind implementing React Hooks, let's break down the concept and key aspects.

#### 1. Definition :

React Hooks are functions that allow you to use state and other React features in functional components. They were introduced in React 16.8 to address the limitations of class components in managing state and side effects. Hooks provide a more concise and intuitive way to work with React's core concepts.

#### 2. Core Hooks:

React provides several core hooks, each serving a specific purpose:

- **useState:** Allows functional components to manage local state.
- **useEffect:** Enables side effects in functional components, such as data fetching or DOM manipulation.
- **useContext:** Provides access to the React context system, allowing components to consume context values.
- **useReducer:** A more advanced state management hook that can be used as an alternative to **useState** for complex state logic.
- **useRef:** Allows access to a mutable ref object, useful for accessing and manipulating DOM elements or persisting values between renders.
- **useMemo** and **useCallback:** Optimize performance by memoizing expensive calculations or callbacks.



- `useLayoutEffect`: Similar to `useEffect`, but fires synchronously after all DOM mutations, which can be important for measuring DOM elements.
- `useImperativeHandle`: Customizes the instance value that is exposed when using `React.forwardRef`.
- `useDebugValue`: Provides additional information for custom hooks in the React DevTools.

### 3. Rules of Hooks:

To use hooks correctly, it's essential to follow the "Rules of Hooks," which are enforced by React:

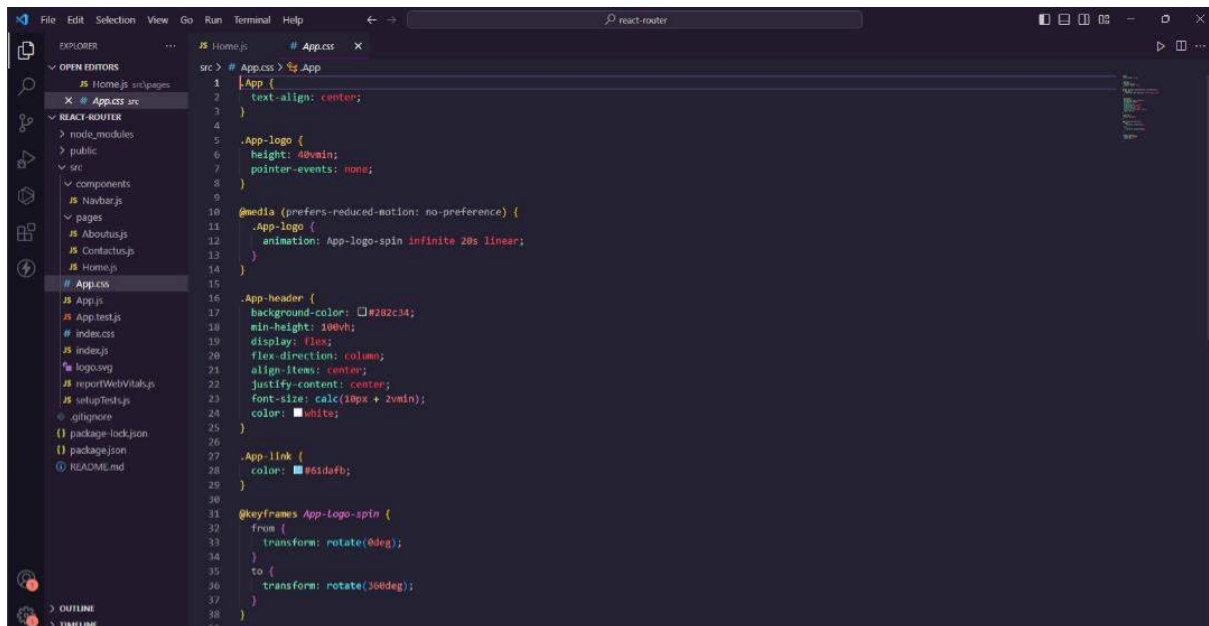
- Hooks must be called at the top level of a functional component or custom hook. They should not be called conditionally or within loops, as this could lead to unexpected behavior.
- Hooks should have the same order when called on every render. This ensures consistency between renders and allows React to maintain the component's state correctly.
- Custom hooks should always start with the word "use" to be recognized as hooks by the ESLint plugin for React.

CODE :

```

Counter.js x
1  import React, { Component, useState, useEffect } from 'react';
2  function Counter() {
3    const [count, setCount] = useState(0);
4    const incrementCount = () => setCount(count + 1);
5
6    useEffect(() => {
7      document.title = `You clicked ${count} times`
8    });
9
10   return (
11     <div>
12       <p>You clicked {count} times</p>
13       <button onClick={incrementCount}>Click me</button>
14     </div>
15   )
16 }
17
18 export default Counter;
19
20
21
22
23

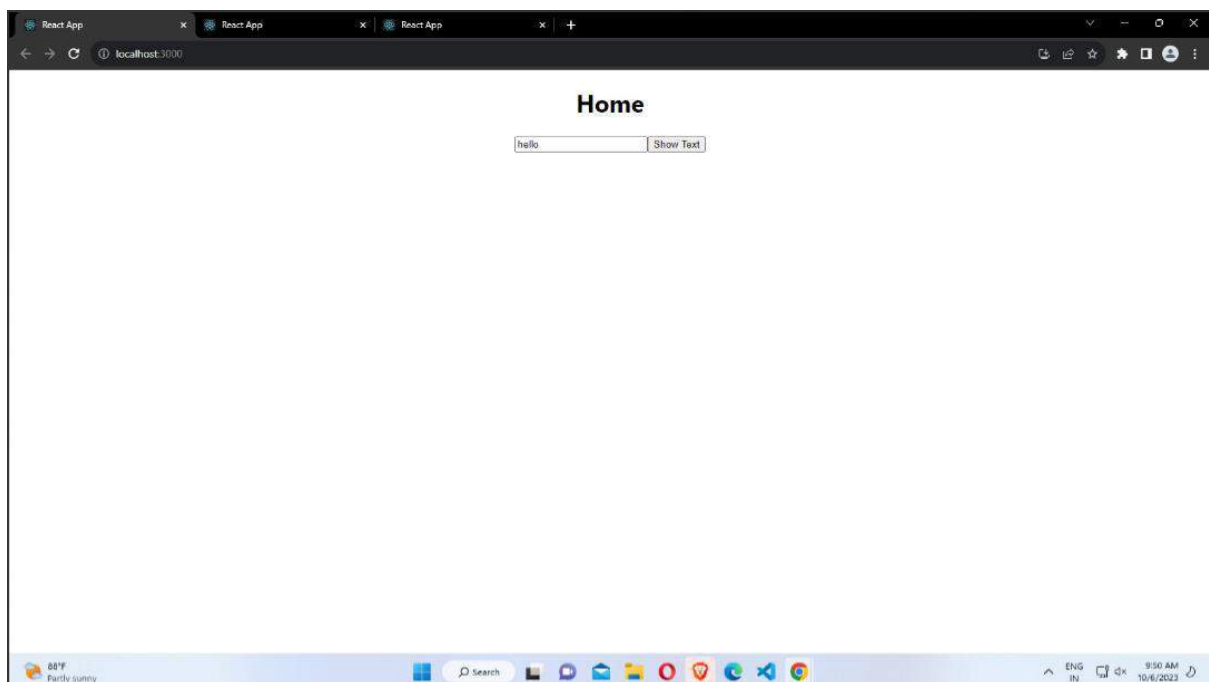
```



The image shows a Visual Studio Code editor window with a dark theme. The Explorer sidebar on the left shows a file tree with folders like 'src', 'components', 'pages', and 'App.css'. The main editor area displays the content of 'App.css'. The code includes a root 'App' selector with text-align, a media query for reduced motion, a keyframe animation 'App-Logo-spin', and several utility classes like '.App-logo', '.App-header', and '.App-link' with various styling properties.

```
1  App {
2    text-align: center;
3  }
4
5  .App-logo {
6    height: 40vmin;
7    pointer-events: none;
8  }
9
10 @media (prefers-reduced-motion: no-preference) {
11   .App-logo {
12     animation: App-Logo-spin infinite 20s linear;
13   }
14 }
15
16 .App-header {
17   background-color: #282c34;
18   min-height: 100vh;
19   display: flex;
20   flex-direction: column;
21   align-items: center;
22   justify-content: center;
23   font-size: calc(10px + 2vmin);
24   color: white;
25 }
26
27 .App-link {
28   color: #61dafb;
29 }
30
31 @keyframes App-Logo-spin {
32   from {
33     transform: rotate(0deg);
34   }
35   to {
36     transform: rotate(360deg);
37   }
38 }
39
```

OUTPUT :



**CONCLUSION :** In conclusion, React Hooks offer a streamlined and intuitive approach to state management and side effects in functional components, enhancing the development of more readable and maintainable React applications. Embracing this modern React paradigm empowers developers to build robust user interfaces with greater ease and efficiency.

NAME : SOUMIL SALVI

ROLL NO : 104

## ASSIGNMENT 8

**AIM :** Assignment on REPL (Commandline) and NodeJs File System

**LO MAPPED :** LO6

### **THEORY :**

The node:repl module exports the repl. REPL Server class. While running, instances of repl. REPL Server will accept individual lines of user input, evaluate those according to a user-defined evaluation function, then output the result. Input and output may be from stdin and stdout, respectively, or may be connected to any Node.js stream.

Instances of repl. REPL Server support automatic completion of inputs, completion preview, simplistic Emacs-style line editing, multi-line inputs, ZSH-like reverse-i-search, ZSH-like substring-based history search, ANSI-styled output, saving and restoring current REPL session state, error recovery, and customizable evaluation functions. Terminals that do not support ANSI styles and Emacs-style line editing automatically fall back to a limited feature set.

Commands and special keys

The following special commands are supported by all REPL instances:

- `.break`: When in the process of inputting a multi-line expression, enter the `. break` command (or press Ctrl + C) to abort further input or processing of that expression.
- `.clear`: Resets the REPL context to an empty object and clears any multiline expression being input.
- `.exit`: Close the I/O stream, causing the REPL to exit. `.help`: Show this list of special commands.
- `.save`: Save the current REPL session to a file: `> .save ./file/to/save.js`
- `.load`: Load a file into the current REPL session. `> .load ./file/to/load.js` .  
editor: Enter editor mode (Ctrl + D to finish, Ctrl + C to cancel).

By default, all instances of repl. REPL Server use an evaluation function that evaluates JavaScript expressions and provides access to Node.js built-in modules. This default behaviour can be overridden by passing in an alternative evaluation function when the repl. REPL Server instance is created.

## OUTPUT :

```
.load    Load JS from a file into the REPL session
.save    Save all evaluated commands in this REPL session to a file

Press Ctrl+C to abort current expression, Ctrl+D to exit the REPL
> .editor
// Entering editor mode (Ctrl+D to finish, Ctrl+C to cancel)
function calculate(exp){
  try{
    return eval(exp);
  }catch(error){
    return 'Error';
  }
}

function getUserInput(){
  rl.question('Enter a num : ',(userInput) =>{
    if(userInput.toLowerCase() === 'exit'){
      rl.close();
    }
    const readline = require('readline');
    const rl = readline.createInterface({
      input: process.stdin,
      output: process.stdout
    });
    console.log('Simple Calculator (Type "exit" to quit)');

    function calculate(expression) {
      try {
        return eval(expression);
      } catch (error) {
        return 'Error';
      }
    }

    function getUserInput() {
      rl.question('Enter an expression: ', (userInput) => {
        if (userInput.toLowerCase() === 'exit') {
          rl.close();
          return;
        }

        const result = calculate(userInput);
        console.log('Result:', result);
        getUserInput();
      });
    }

    getUserInput()
  });
}
```

```

    }

    rl.question('Enter the second number: ', (num2) => {
      if (num2.toLowerCase() === 'exit') {
        rl.close();
        return;
      }

      rl.question('Select operation (+, -, *, /): ', (operator) => {
        const parsedNum1 = parseFloat(num1);
        const parsedNum2 = parseFloat(num2);

        if (isNaN(parsedNum1) || isNaN(parsedNum2)) {
          console.log('Invalid input. Please enter numeric values. ');
          getUserInput();
        } else {
          const result = calculate(operator, parsedNum1, parsedNum2);
          console.log('Result:', result);
          getUserInput();
        }
      });
    });
  });
});
});

console.log('Simple Command-Line Calculator (Type "exit" to quit)');
getUserInput();

Simple Command-Line Calculator (Type "exit" to quit)
Enter the first number: undefined
Enter the first number: 11
1
>
Enter the second number: 22
2
>
Select operation (+, -, *, /): **
*
>
Uncaught SyntaxError: Unexpected token '**'
>
Result: 2
Enter the first number: _
```

**CONCLUSION :** We learnt that a “REPL” stands for “Read-Eval-Print Loop.” It’s an interactive programming environment that allows you to enter commands or code snippets, which are then read, evaluated, and the results printed back to you. Many programming languages, like Python and JavaScript, have REPL environments that enable developers to interactively work with code.

## **NODEJS FILE SYSTEEM**

### **THEORY :**

File handling in Node.js allows you to interact with the file system, including reading from and writing to files. Node.js provides a built-in module called "fs" (file system) for this purpose.

i. Synchronous vs. Asynchronous File Operations:

Node.js provides both synchronous and asynchronous methods for file handling. Asynchronous methods are generally preferred because they don't block the execution of other code while waiting for file operations to complete. Synchronous methods, on the other hand, can block the event loop, which can lead to performance issues, especially in web applications. Asynchronous methods use callbacks or Promises to handle results when the operation is finished.

ii. Working with Buffers:

When reading or writing files in Node.js, you often encounter Buffers. Buffers are binary data objects used to work with binary files, such as images or binary documents. When you read a file using `fs.readFile()`, the result is a Buffer by default. You can convert Buffers to strings by specifying an encoding (e.g., 'utf8') when reading the file.

iii. Streams for Efficient File Handling:

Node.js provides a powerful feature called streams for handling files efficiently. Streams allow you to read or write data in chunks, which is

useful for large files as it minimizes memory usage. There are readable streams (for reading) and writable streams (for writing).

#### iv. Working with File Paths:

When working with files and directories, it's crucial to handle file paths correctly, especially if your code needs to run on different platforms (Windows, macOS, Linux). Node.js provides the path module to help with path manipulation and ensure cross-platform compatibility. `path.join()` constructs a platform-independent file path using the `__dirname` variable representing the current directory.

OUTPUT :

```
JS app.js > ...
1  const fs = require('fs');
2  const readline = require('readline');
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout
7  });
8
9  function createFile() {
10   rl.question('Enter the file name to create: ', (fileName) => {
11     fs.writeFile(fileName, '', (err) => {
12       if (err) {
13         console.error('Error creating file:', err);
14       } else {
15         console.log('File "${fileName}" created successfully.');
```

```

37     rl.question('Enter the content to write: ', (content) => {
38         fs.writeFile(fileName, content, (err) => {
39             if (err) {
40                 console.error('Error writing to file:', err);
41             } else {
42                 console.log('Content written to "${fileName}" successfully.');

```

JS app.js

harsh

×

harsh

1 |

PROBLEMS 1

OUTPUT

DEBUG CONSOLE

TERMINAL

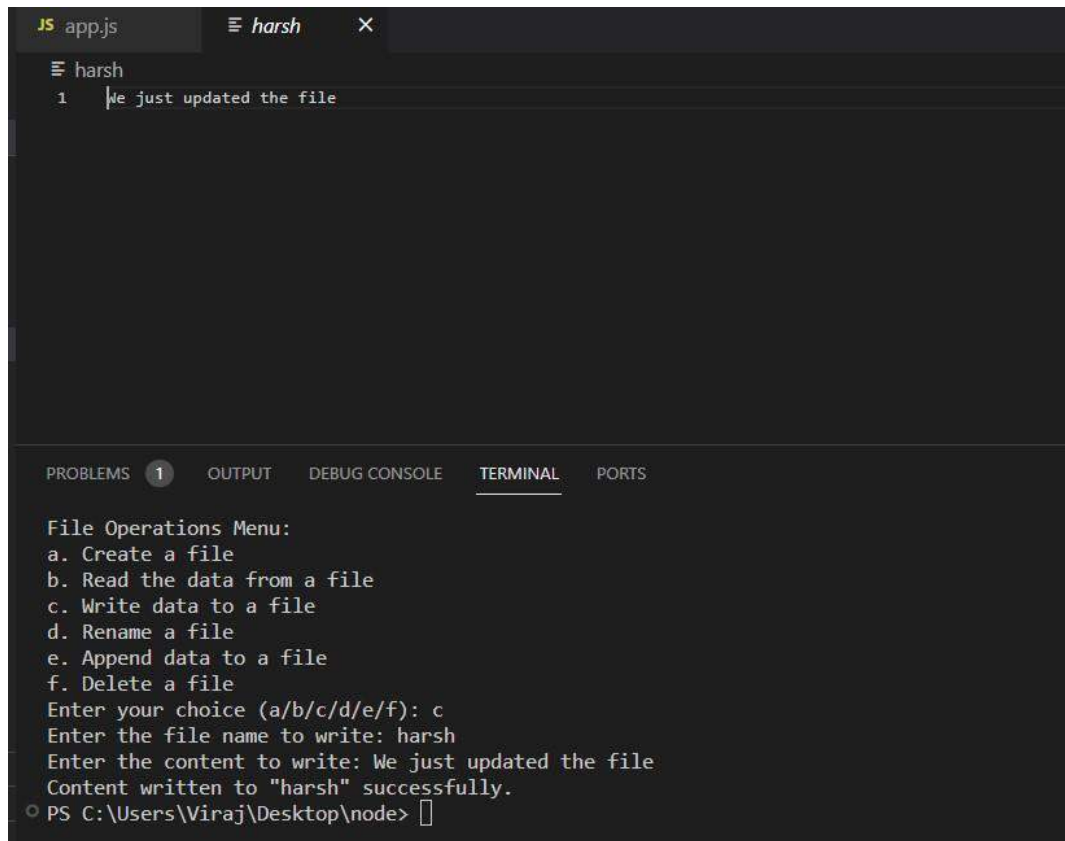
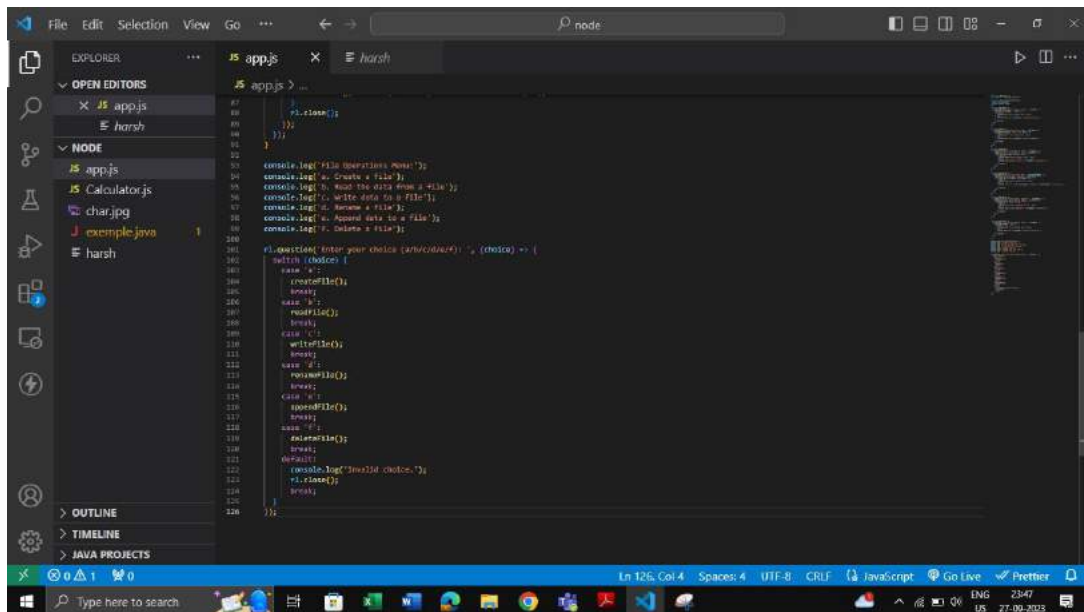
PORTS

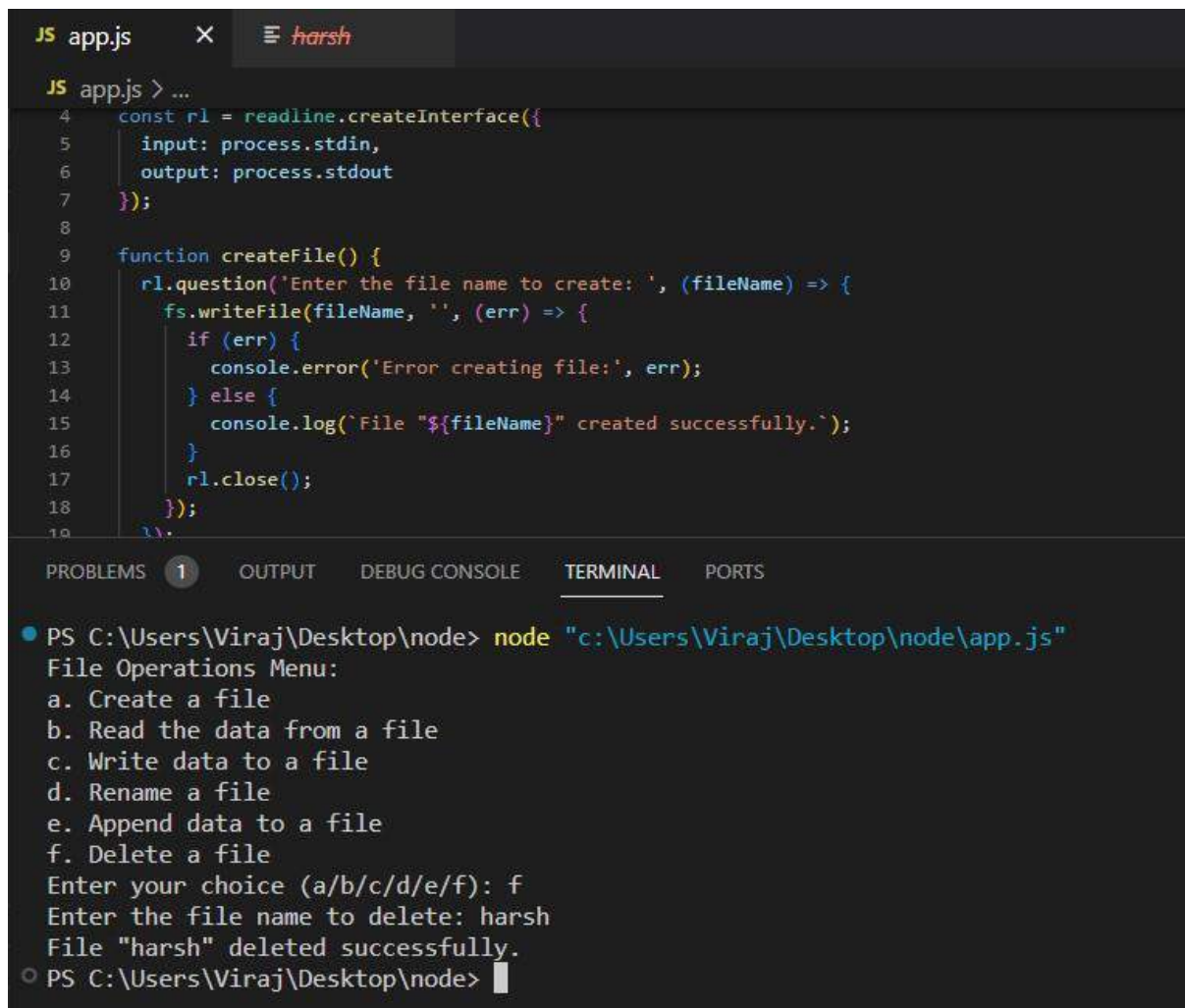
```

PS C:\Users\Viraj\Desktop\node> node "c:\Users\Viraj\Desktop\node\app.js"
• File Operations Menu:
a. Create a file
b. Read the data from a file
c. Write data to a file
d. Rename a file
e. Append data to a file
f. Delete a file
Enter your choice (a/b/c/d/e/f): a
Enter the file name to create: harsh
File "harsh" created successfully.
○ PS C:\Users\Viraj\Desktop\node>

```







The image shows a VS Code editor window with a file named `app.js` and a terminal window below it. The code in `app.js` uses the `readline` module to create a command-line interface. It prompts the user to enter a file name to create, then uses `fs.writeFile` to create the file. If successful, it logs a message; otherwise, it logs an error. The terminal shows the command to run the application and the resulting menu of file operations. The user has chosen option 'f' (Delete a file) and entered 'harsh' as the file name to delete. The application successfully deleted the file 'harsh'.

```
JS app.js x harsh  
JS app.js > ...  
4 const rl = readline.createInterface({  
5   input: process.stdin,  
6   output: process.stdout  
7 });  
8  
9 function createFile() {  
10   rl.question('Enter the file name to create: ', (fileName) => {  
11     fs.writeFile(fileName, '', (err) => {  
12       if (err) {  
13         console.error('Error creating file:', err);  
14       } else {  
15         console.log(`File "${fileName}" created successfully.`);  
16       }  
17       rl.close();  
18     });  
19   });  
20 }
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
● PS C:\Users\Viraj\Desktop\node> node "c:\Users\Viraj\Desktop\node\app.js"  
File Operations Menu:  
a. Create a file  
b. Read the data from a file  
c. Write data to a file  
d. Rename a file  
e. Append data to a file  
f. Delete a file  
Enter your choice (a/b/c/d/e/f): f  
Enter the file name to delete: harsh  
File "harsh" deleted successfully.  
○ PS C:\Users\Viraj\Desktop\node> 
```

**CONCLUSION :** In this assignment, we explored the fundamentals of file handling in Node.js, including reading and writing files asynchronously, appending to existing files, checking file existence, deleting files, and working with directories.

NAME : SOUMIL SALVI

ROLL NO : 104

## ASSIGNMENT 9

**AIM :** WAP to implement Refs in React.

**LO MAPPED :** LO6

### **THEORY :**

In React, refs are a way to access and interact with DOM elements or React components directly. They provide a means to reference a specific element or component within your application, allowing you to read values, trigger actions, or perform other operations that would be difficult to achieve using React's standard data flow and state management mechanisms. Refs can be helpful in scenarios such as handling form input focus, triggering animations, integrating with third-party libraries, or accessing underlying DOM APIs.

Use cases for React refs:

#### 1. Accessing DOM Elements:

Refs are commonly used to interact with DOM elements directly. For instance, you can access input values, set focus, or trigger animations.

#### 2. Managing Third-Party Libraries:

When integrating React with third-party libraries that require direct DOM manipulation, refs are essential. For example, if you're using a charting library or a video player, you can use refs to interact with their APIs.

#### 3. Animating Elements:

You can use refs to trigger animations by changing CSS classes or properties. This is useful when you need precise control over animations.

#### 4. Scrolling and Measuring Elements:

You can use refs to scroll to specific elements on the page or measure their dimensions. This is helpful for building features like smooth scrolling or lazy-loading content.

#### 5. Integration with Non-React Code:

If you need to integrate React with non-React code, such as legacy JavaScript or libraries like D3.js, you can use refs to bridge the gap between React components and external code.

#### 6. Forms and Input Validation:

Refs can be useful for form handling and input validation. You can access input elements directly to check and manipulate their values or apply custom validation logic.

### Creating Refs:

You can create a ref using the **React.createRef()** method (for class components) or the **createRef** hook (for functional components).

### Accessing Refs:

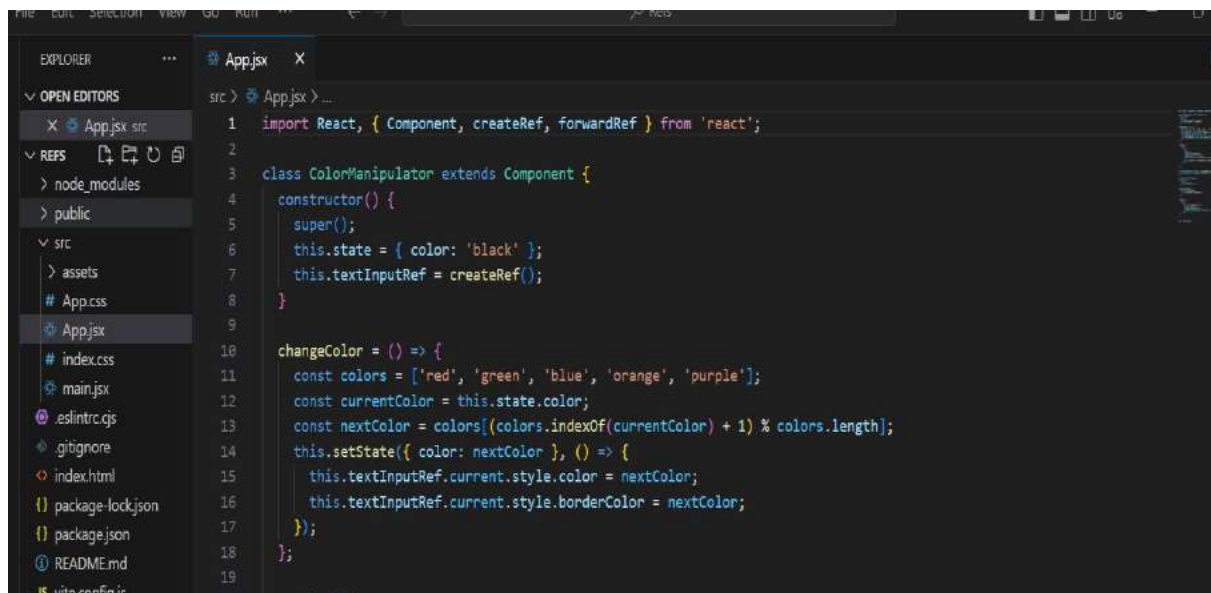
You can access the DOM element or React component associated with a ref by using the **current** property of the ref object. This property holds the reference to the underlying element or component. You typically access the ref inside lifecycle methods (for class components) or within the functional component itself.

## Forwarding Refs:

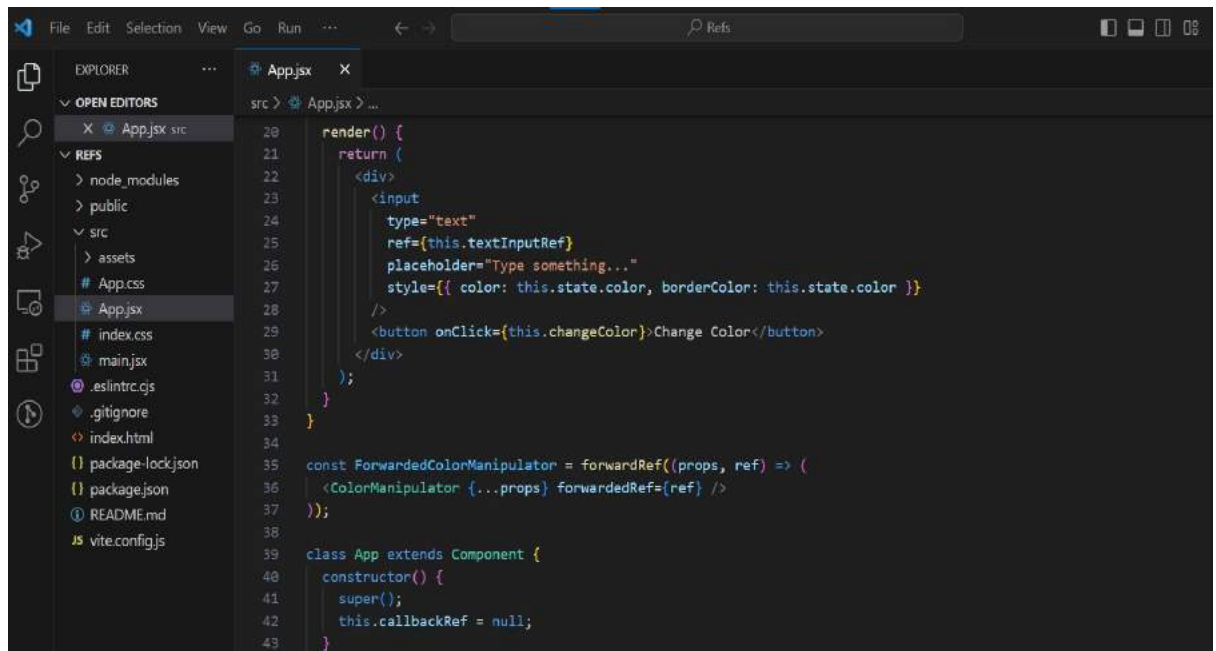
Sometimes, you may need to pass a ref from a parent component to a child component. This can be achieved using the "forwarding refs" technique. React provides the **forwardRef** function for this purpose. It allows you to pass a ref down the component hierarchy and access the child component's ref from a parent component.

## Callback Refs:

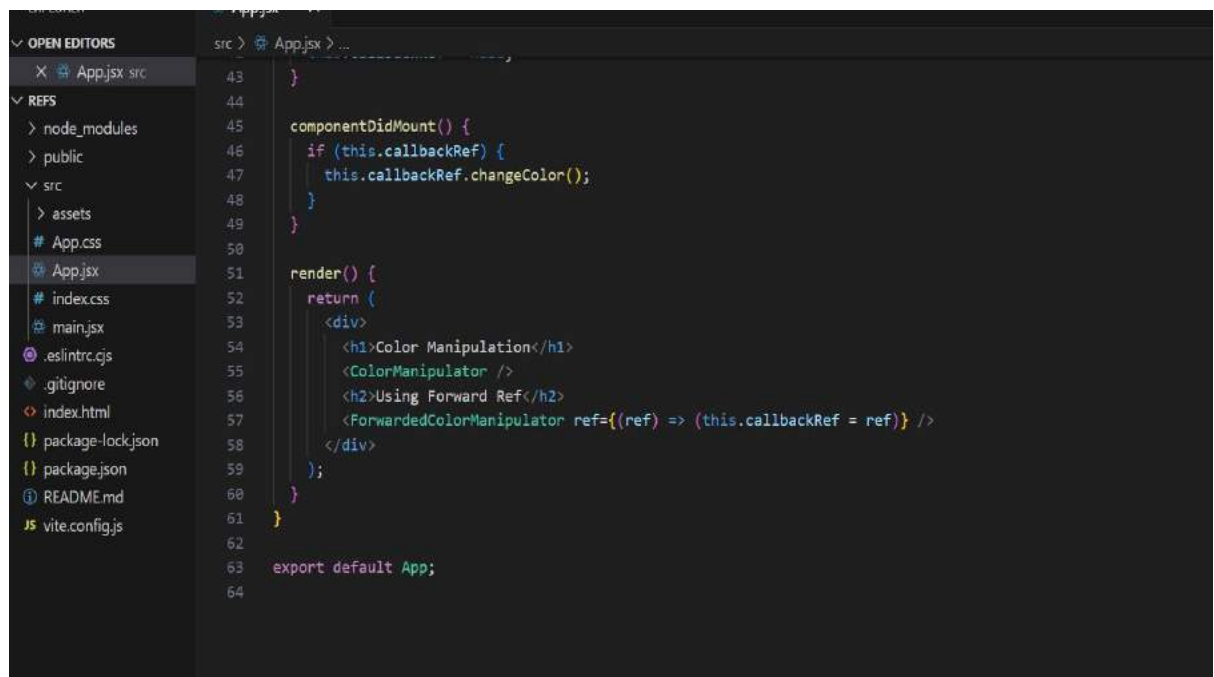
Callback refs are another way to work with refs, especially in cases where you need to perform additional operations when a ref is set. Instead of using the **createRef** or **useRef** functions, you define a callback function that will be called when the ref is attached to an element or component.



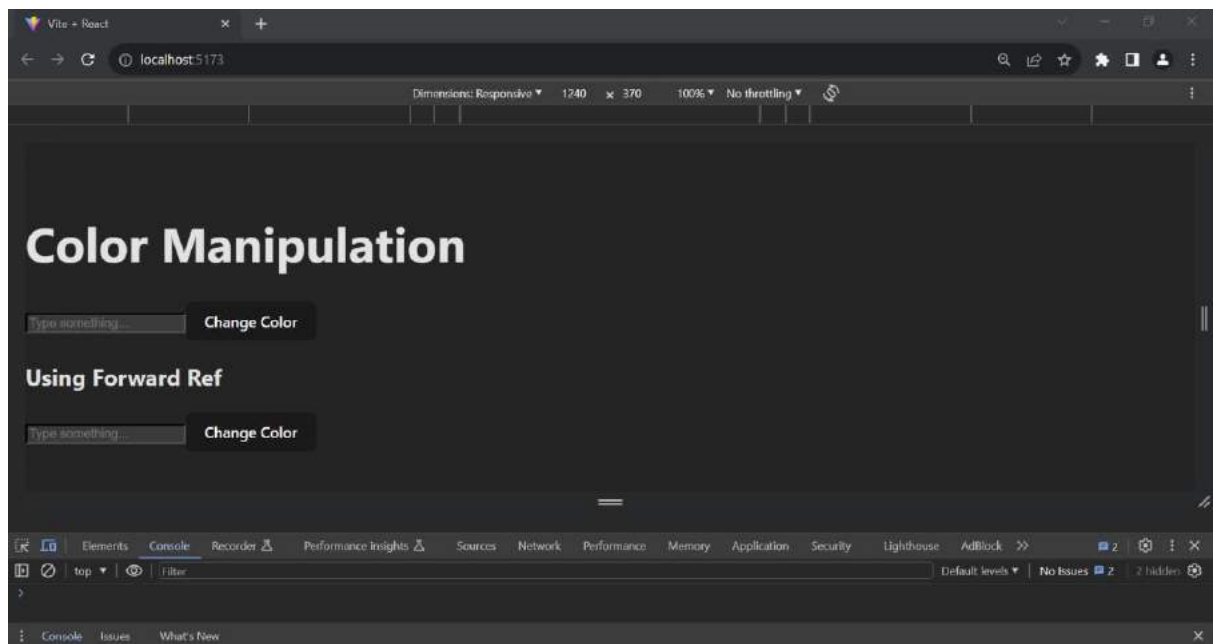
```
1 import React, { Component, createRef, forwardRef } from 'react';
2
3 class ColorManipulator extends Component {
4   constructor() {
5     super();
6     this.state = { color: 'black' };
7     this.textInputRef = createRef();
8   }
9
10  changeColor = () => {
11    const colors = ['red', 'green', 'blue', 'orange', 'purple'];
12    const currentColor = this.state.color;
13    const nextColor = colors[(colors.indexOf(currentColor) + 1) % colors.length];
14    this.setState({ color: nextColor }, () => {
15      this.textInputRef.current.style.color = nextColor;
16      this.textInputRef.current.style.borderColor = nextColor;
17    });
18  };
19
20  render() {
21    return (
22      <div>
23        <input type="text" ref={this.textInputRef}/>
24      </div>
25    );
26  }
27}
```



```
20 render() {
21   return (
22     <div>
23       <input
24         type="text"
25         ref={this.textInputRef}
26         placeholder="Type something..."
27         style={{ color: this.state.color, borderColor: this.state.color }} />
28     </div>
29     <button onClick={this.changeColor}>Change Color</button>
30   </div>
31 );
32 }
33 }
34
35 const ForwardedColorManipulator = forwardRef((props, ref) => (
36   <ColorManipulator {...props} forwardedRef={ref} />
37 ));
38
39 class App extends Component {
40   constructor() {
41     super();
42     this.callbackRef = null;
43   }
```



```
43 }
44
45 componentDidMount() {
46   if (this.callbackRef) {
47     this.callbackRef.changeColor();
48   }
49 }
50
51 render() {
52   return (
53     <div>
54       <h1>Color Manipulation</h1>
55       <ColorManipulator />
56       <h2>Using Forward Ref</h2>
57       <ForwardedColorManipulator ref={(ref) => (this.callbackRef = ref)} />
58     </div>
59   );
60 }
61
62
63 export default App;
64
```





# Color Manipulation

om\_91

Change Color

## Using Forward Ref

om\_t21

Change Color

# Color Manipulation

om\_91

Change Color

## Using Forward Ref

om\_t21

Change Color

## CONCLUSION :

We successfully demonstrated how React refs can be used to create, access, and manipulate DOM elements and components, showcasing their versatility within a React application.

# CRUD Project

**AIM:** Express App CRUD Operation

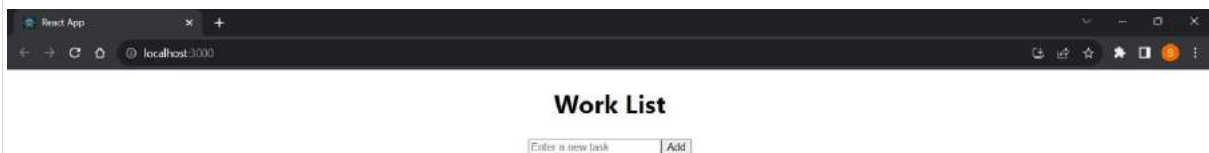
**LO:** Maps to all LOs

**Theory:**

Express.js is a popular web application framework for Node.js that simplifies the process of building robust and scalable web applications. One of the fundamental operations in web development is CRUD, which stands for Create, Read, Update, and Delete. CRUD operations are used to interact with databases, manipulate data, and perform various actions within a web application. In this theoretical discussion, we will explore how Express.js facilitates the implementation of CRUD operations.

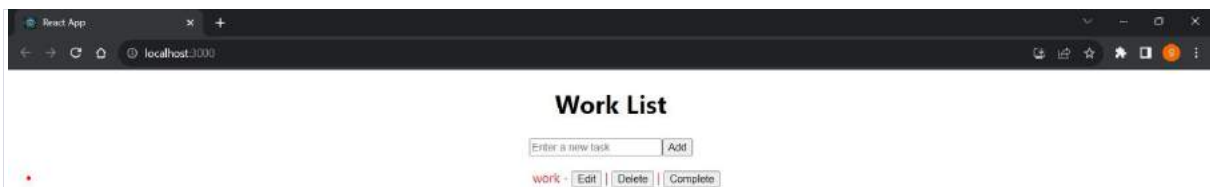
## CRUD Operations (with our project steps)

**Initial:**



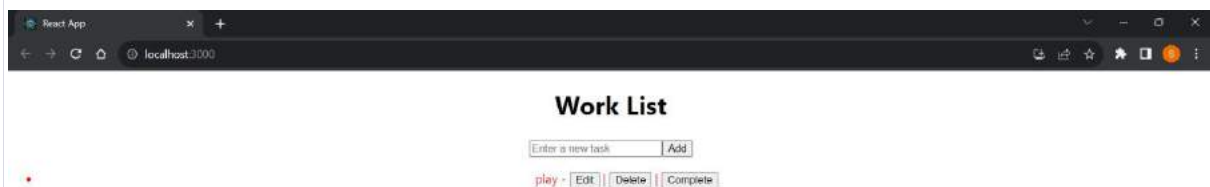
### 1. Create (POST)

The "Create" operation is used to add new data to a database. In Express.js, you typically create a route with the HTTP POST method to handle the creation of new records. When a client sends a request to this route, the server receives the data in the request body, processes it, and stores it in the database. Express simplifies this process by providing middleware like `body-parser` or `express.json()` to parse incoming JSON data.



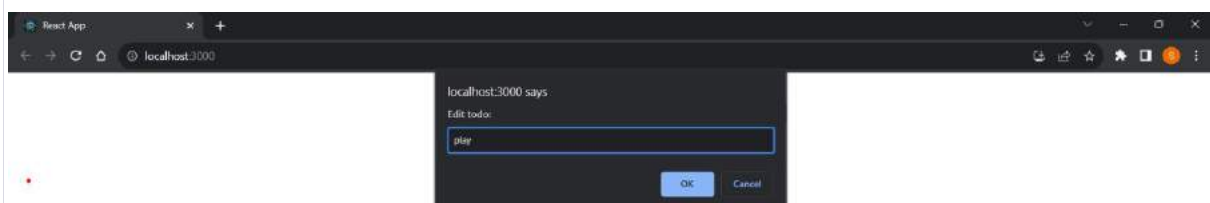
## 2. Read (GET)

The "Read" operation is used to retrieve data from a database. In Express.js, you create routes with HTTP GET methods to fetch data. You can implement routes for fetching all records or a specific record by its unique identifier. This data is then sent back to the client as a response. Express allows you to create flexible routes to filter, sort, and paginate data for efficient retrieval.



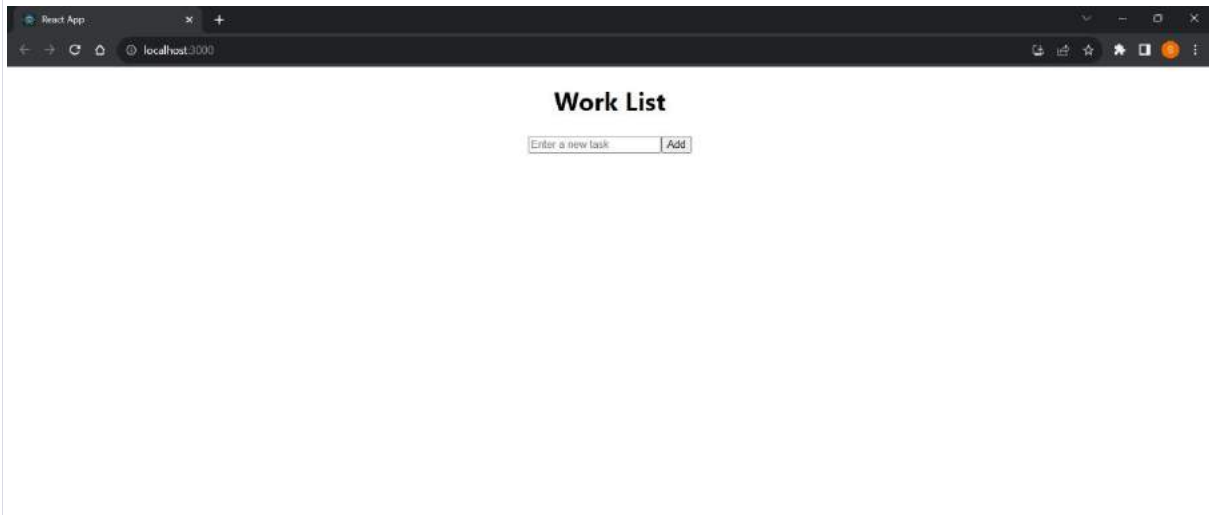
## 3. Update (PUT/PATCH)

The "Update" operation is used to modify existing data in the database. In Express.js, you can use routes with HTTP PUT or PATCH methods to handle updates. PUT typically updates an entire resource, while PATCH is used to update specific fields. The server receives the updated data, identifies the resource to update (often by an ID), and makes the necessary changes in the database. Express simplifies routing and data validation for updates.



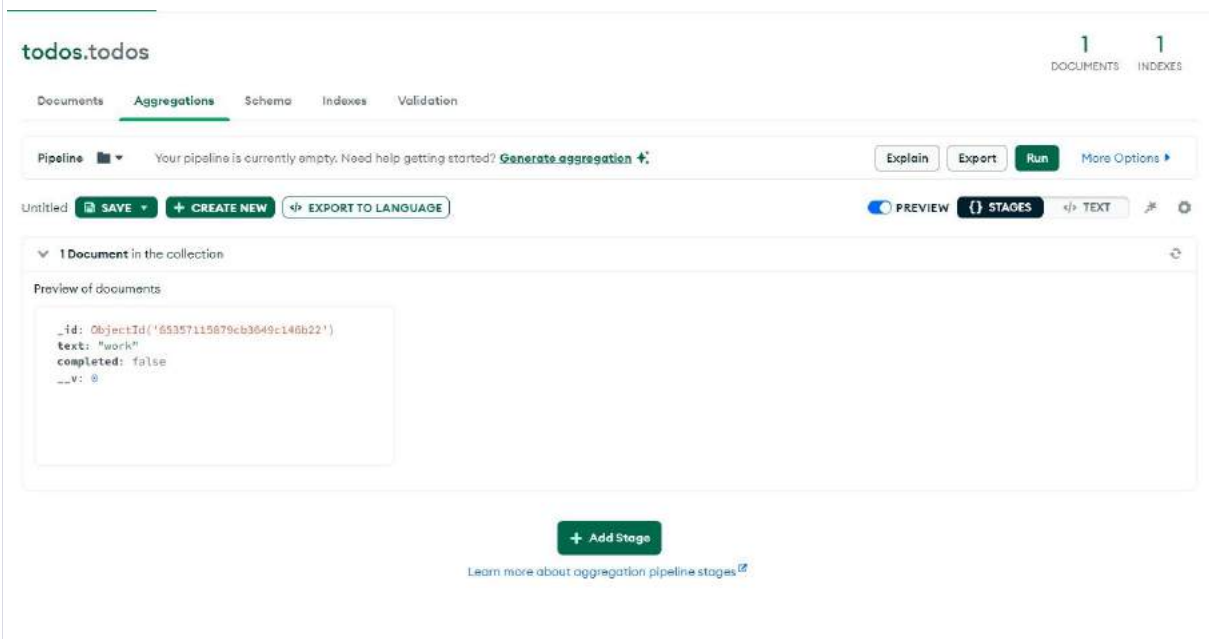
## 4. Delete (DELETE)

The "Delete" operation is used to remove data from the database. In Express.js, you create routes with the HTTP DELETE method to delete records. The server receives the request and identifies the record to delete, then removes it from the database. Express helps streamline this process by providing routing and error handling capabilities.



## CRUD Database Connectivity:

In an Express.js application, connecting to a database for CRUD (Create, Read, Update, Delete) operations is a common requirement. Below, we'll discuss the steps to establish a database connection and perform CRUD operations using the popular MongoDB database and the Mongoose ODM (Object Data Modeling).



# Key Components in Express.js CRUD Operations

## 1. Routing

Express.js allows you to define routes for handling different CRUD operations. You can specify the HTTP method, the route path, and the corresponding controller function to execute when a request matches that route. This separation of concerns makes code modular and maintainable.

## 2. Controllers

Controllers contain the logic for each CRUD operation. They receive the incoming request, process the data, interact with the database using models, and return an appropriate response to the client. Express.js encourages the use of separate controller functions for each CRUD operation.

## 3. Models

Models represent the data structure and schema in your application. In Express.js, you can use libraries like Mongoose (for MongoDB) or Sequelize (for SQL databases) to define the structure of your data and perform database operations. Models help ensure data consistency and validation.

## 4. Middleware

Middleware functions in Express.js play a crucial role in processing incoming requests. Middleware can be used for tasks like authentication, validation, error handling, and data parsing. Middleware enhances the security and reliability of your CRUD operations.

## Conclusion

Express.js provides a robust and efficient framework for implementing CRUD operations in web applications. By creating routes, controllers, and models, and by using middleware, you can easily handle the full spectrum of CRUD actions. The framework's simplicity, flexibility, and extensive community support make it a popular choice for developers when building web applications that require Create, Read, Update, and Delete functionality.