

Thadomal Shahani Engineering College

Bandra (W.), Mumbai- 400 050.

© CERTIFICATE ©

Certify that Mr./Miss Altaf Alam
of Information Technology Department, Semester V with
Roll No. 02 has completed a course of the necessary
experiments in the subject IP Lab under my
supervision in the **Thadomal Shahani Engineering College**
Laboratory in the year 2023 - 2024

~~Teacher In-Charge~~

Head of the Department

Date 13/10/23

Principal

CONTENTS

SR. NO.	EXPERIMENTS	PAGE NO.	DATE	TEACHERS SIGN.
1.	Develop a web application using HTML Tags		19/7/23	
2.	Using CSS and CSS3 enhance the web app of Assignment 1		26/7/23	
3.	Develop a webpage using the Bootstrap framework.		21/8/23	
4.	a) WAP in JS to study loops and functions b) WAP on inheritance, iterators, generators	9/8/23	9/8/23	Sanobek 13/10/23
5.	a) WAP to study arrow functions DOM and CSS manipulation. b) Write a JS program to implement promises, fetch & async JS	21/8/23	21/8/23	
6.	a) WAP to implement props & state b) WAP to implement forms & events	23/8/23	23/8/23	Sanobek 13/10/23
7.	a) WAP to implement React JS router and animation b) WAP to implement React Hooks	6/9/23	13/9/23	
8.	REPL (command line)		27/9/23	
9.	WAP to implement React Refs		13/10/23	
10.	WAP in NodeJS to create a file, read, write and delete and rename the file.		27/9/23	
11.	Create a web application that performs CRUD operations (database connectivity)		13/10/23	Sanobek 13/10/23
12.	Theory Assignment 1		10/8/23	
13.	Theory Assignment 2		23/9/23	

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 18/07/23

Assignment No 1

Aim : Develop a web app using HTML tags.

Lab Outcome : LO1: To orient students to HTML for making webpages.

Theory :

What is HTML?

HTML (Hypertext Markup Language) is the standard language used to create and design websites. It provides a structured way to define the content and layout of web pages. It is the backbone of web development. It allows developers to create documents that are structured and easily readable by web browsers. HTML documents consist of a series of elements or tags, each serving a specific purpose. Tags are enclosed within angle brackets and instruct the browser how to display the content. These tags are the building blocks of web pages, and by combining them in different ways, developers can create visually appealing and interactive websites.

Description of Tags used in the example:

1. `<!DOCTYPE html>` : This is the document type declaration that tells the browser that the document is an HTML5 document. It must be the first line of an HTML document.
2. `<html>` : The root element of an HTML document, enclosing all other elements.
3. `<head>` : Contains meta-information about the document, such as the document title, character set, and links to external resources like stylesheets or scripts.
4. `<meta>` : Defines metadata about the document, such as the character encoding and viewport settings.
5. `<link>` : Used to link external resources like stylesheets to the HTML document.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 18/07/23

6. `<title>`: Sets the title of the document, which appears in the browser's title bar or tab.
7. `<body>`: Contains the visible content of the web page, including text, images, links, and other elements.
8. `<nav>`: Represents a section of the page that contains navigation links.
9. ``: Embeds an image in the document. The `src`, `alt`, `height`, and `width` attributes define the image source, alternative text, height, and width, respectively.
10. `` and ``: Create an unordered list and list items, respectively. Used for creating lists of items without specific numbering.
11. `<h1>`, `<h2>`, ..., `<h6>`: Heading tags define headings of different levels, with `<h1>` being the highest level and `<h6>` being the lowest.
12. `<hr>`: Creates a horizontal rule or divider on the page.
13. `<video>`: Embeds video content in the document. The `controls`, `height`, and `width` attributes define video controls and dimensions.
14. `<source>`: Specifies the video source for the `<video>` element.
15. `<audio>`: Embeds audio content in the document. The `controls` attribute adds audio controls to the element.
16. `<fieldset>` and `<legend>`: Used together to group form elements and provide a legend or caption for the fieldset.
17. `<textarea>`: Creates a multi-line text input field for forms.
18. `<form>`: Defines an HTML form for user input. The `action` attribute specifies the URL to send the form data.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 18/07/23

19. '`<input>`': Creates various types of input fields within a form, such as text, email, and submit buttons.

20. `<table>`, `<thead>`, `<tbody>`, `<tr>`, `<th>`, and `<td>`: Used to create tables with table headers, table body, table rows, table headers, and table data cells, respectively.

21. '**'': Renders text in bold font.**

22. '``': Renders text in italic font.

23. '<a>': Creates hyperlinks, linking to other web pages or resources. The 'href' attribute specifies the link destination.

24. '`<pre>`': Renders preformatted text, preserving spaces and line breaks.

25. `<footer>`: Defines the footer section of the web page, usually containing copyright information and social media links.

Source Code :

Index.html

```
<html>
<head>
    <meta name="viewport" content="width=device-width">
    <title>Personal Information</title>
    </head>
    <body>
        <div>
            <img alt="Logo" href="index.html" height="50px" width="150px"/>
            <ul class="nav-menu">
                <li><a href="index.html">Home</a></li>
                <li><a href="contactme.html">Contact</a></li>
                <li><a href="OthersSkills.html">Know More</a></li>
            </ul>
        </div>
        <div>
            <h1>Altaf Alam</h1>
            <p>+919875421218</p>
            <img alt="Profile Photo" href="index.html" height="100px" width="200px"/>
            <div>IT & ENGINEERING (MCA) (15CG)</div>
            <div>TECH GEEK || Web Developer</div>
        </div>
        <div>
            <h2>Education</h2>
            <table border="1px solid black">
                <thead>
                    <tr>
                        <th>Year</th>
                        <th>College</th>
                        <th>Description</th>
                    </tr>
                </thead>
                <tbody>
                    <tr>
                        <td>2025(exp)</td>
                        <td>Thosmeh Shahani Engineering College</td>
                        <td>Maintained 9.32 CGPA(lIII 3rd sem).</td>
                    </tr>
                    <tr>
                        <td>2021(exp)</td>
                        <td>Guru Nanak Khalsa College</td>
                        <td>Secured 97.98th percentile in MHCIET STATE Level Exam.</td>
                    </tr>
                    <tr>
                        <td>2021</td>
                        <td>Guru Nanak Khalsa College</td>
                        <td>Secured 99.83% in HSC exam.</td>
                    </tr>
                </tbody>
            </table>
        </div>
        <div>
            <h2>Skills</h2>
            <ul>
                <li>Full Stack Web Developer</li>
                <li>Python</li>
                <li>C/C++</li>
                <li>Java</li>
                <li>Leadership & Teamwork</li>
            </ul>
        </div>
        <div>
            <img alt="Social Media Icons" href="index.html" height="50px" width="150px"/>
            <div>Altaf Alam, All Rights Reserved</div>
        </div>
    </body>

```

Contactme.html

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 18/07/23

File Edit Selection View Go Run Terminal Help contactme.html - 90's Portfolio - Visual Studio Code

index.html contactme.html OtherSkills.html ... contactme.html

You, 21 minutes ago [2 authors (You and others)]

```
1 <!DOCTYPE html>
2 <html>
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     <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css" />
8   <title>Contact Me</title>
9   </head>
10 <body>
11   <nav>
12     <img alt="Logo" class="logo" height="80px" width="150px" />
13     <ul class="nav-menu">
14       <li class="nav-items"><a href="index.html">Home</a></li>
15       <li class="nav-items"><a href="contactme.html">Contact</a></li>
16       <li class="nav-items"><a href="OtherSkills.html">Know More</a></li>
17     </ul>
18   </nav>
19   <div>
20     <h2>Contact Me</h2>
21     <form action="mailto:altaf526687@gmail.com" method="post">
22       <fieldset>
23         <legend>Your Info:</legend>
24         <label>Your Name:</label>
25         <input type="text" /> <br />
26         <label>Your Email:</label>
27         <input type="email" value="altaf526687@gmail.com" /> <br />
28       </fieldset>
29     </div>
30   <div>
31     <p>Mobile No: 9143143143</p>
32     <p>Address: 541, Noor Mahal Building, Adenwala Road, King Circle, Matunga, Mumbai - 400019</p>
33     <p>Email: altaf526687@gmail.com</p>
34   </div>
35   <div>
36     <a href="https://www.instagram.com/altaf_alam_687?igshid=YmMyMTA2M2Y=" class="fa fa-facebook fa-2x" />
37     <a href="https://twitter.com/Altaf_Alam_003?ct=tB1Q2yUflcEJB5bs5Zt2w8s-09" class="fa fa-twitter fa-2x" />
38     <a href="https://www.linkedin.com/in/altaf-alam-432849234/" class="fa fa-github fa-2x" />
39     <a href="https://www.instagram.com/altaf_alam_687?igshid=YmMyMTA2M2Y=" class="fa fa-instagram fa-2x" />
40   </div>
41   <div>
42     <h3>© Altaf Alam, All Rights Reserved</h3>
43   </div>
44 </body>
45 </html>
```

OtherSkills.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css" />
    <title>Additional Skills</title>
</head>
<body>
    <nav>
        
        <ul class="nav-menu">
            <li class="nav-item"><a href="index.html">Home</a></li>
            <li class="nav-item"><a href="contactme.html">Contact</a></li>
            <li class="nav-item"><a href="OtherSkills.html">Know More</a></li>
        </ul>
    </nav>
    <section>
        <h2>Blogs</h2>
        <div>
            <ul>
                <li><a href="https://www.geeksforgeeks.org/complete-roadmap-to-learn-data-structures-in-7-days/">Roadmap to DSA in 7 days</a></li>
                <li><a href="https://www.geeksforgeeks.org/complete-roadmap-to-learn-algorithms-in-7-days/">5 steps to learn DSA from scratch learn at least one Programming Language Learn about Complexities Learn Data Structure and Algorithms</a></li>
                <li><a href="https://www.geeksforgeeks.org/complete-roadmap-to-learn-dsa-in-7-days/">Divide and Conquer Algorithm</a></li>
                <li><a href="https://www.geeksforgeeks.org/complete-roadmap-to-learn-data-structures-in-7-days/">Sorting Algorithm</a></li>
                <li><a href="https://www.geeksforgeeks.org/complete-roadmap-to-learn-algorithms-in-7-days/">Searching Algorithm</a></li>
                <li><a href="https://www.geeksforgeeks.org/complete-roadmap-to-learn-data-structures-in-7-days/">Linked List</a></li>
            </ul>
        </div>
    </section>
</body>
```

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 18/07/23

```
<html>
<head>
    <title>90's Portfolio - Visual Studio Code</title>
</head>
<body>
    <div>
        <h1>Data Structures</h1>
        <ul>
            <li><a href="https://www.geeksforgeeks.org/complete-roadmap-to-learn-dsa-from-scratch/">Stack</a></li>
            <li><a href="https://www.geeksforgeeks.org/complete-roadmap-to-learn-dsa-from-scratch/">Queue</a></li>
            <li><a href="https://www.geeksforgeeks.org/complete-roadmap-to-learn-dsa-from-scratch/">Tree Data Structure</a></li>
            <li><a href="https://www.geeksforgeeks.org/complete-roadmap-to-learn-dsa-from-scratch/">Graph Data Structure</a></li>
            <li><a href="https://www.geeksforgeeks.org/complete-roadmap-to-learn-dsa-from-scratch/">Greedy Methodology</a></li>
        </ul>
        <p>Total 20 minutes ago * Uncommitted changes</p>
        <h1>Recursion</h1>
        <ul>
            <li><a href="https://www.geeksforgeeks.org/complete-roadmap-to-learn-dsa-from-scratch/">Backtracking Algorithm</a></li>
            <li><a href="https://www.geeksforgeeks.org/complete-roadmap-to-learn-dsa-from-scratch/">Dynamic Programming</a></li>
        </ul>
        <div>Practice, practice, practice and become a pro!</div>
    </div>
    <div>
        <h2>Maths</h2>
        <ul>
            <li><a href="#">Maths Lecture 1 : LPM</a></li>
            <li><a href="#">Maths Lecture 2 : Probability</a></li>
        </ul>
        <div>
            <h3>My experience at Digidemia</h3>
            <audio controls>
                <source src="assets/audio1.mp3" type="audio/mpeg" />
            </audio>
            <h3>My experience at 3ptogen</h3>
            <audio controls>
                <source src="assets/audio2.mp3" type="audio/mpeg" />
            </audio>
        </div>
        <div>
            <h3>Mastery</h3>
            <ul>
                <li><a href="https://instagram.com/altaf_alam_6871igshidarwylm1AZH0V/">Instagram</a></li>
                <li><a href="#">Facebook Fa-2x</a></li>
                <li><a href="https://twitter.com/altaf00327t+0lQywmrlcJ3B0ns5zt2wls-098">Twitter</a></li>
                <li><a href="https://www.linkedin.com/in/altaf-alam-432849234">LinkedIn</a></li>
                <li><a href="#">GitHub Fa-2x</a></li>
                <li><a href="https://instagram.com/altaf_alam_6871igshidarwylm1AZH0V/">Instagram</a></li>
            </ul>
        </div>
    </div>
    <div>Copyright © Altaf Alam, All Rights Reserved</div>
</body>
</html>
```

Output :

Altaf Alam , 02 , T11

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 18/07/23

The screenshot shows a web browser window with multiple tabs open at the top. The main content area displays a resume for Altaf Alam. At the top left is a green logo with the name 'Altaf'. Below it is a navigation menu with links to 'Home', 'Contact', and 'Know More'. The main heading is 'Altaf Alam' with contact information '+919876543210' and 'altaf526687@gmail.com'. There is a small profile picture of Altaf. The section 'TECH GEEK || Web Developer.' follows. Below this is a 'Education' section containing a table with four rows of data. The table has columns for 'Year', 'College', and 'Description'. The last row is for 'SIWS High School' where it states 'Secured 76.8% overall with 90+ in maths in SSC exam.'. The browser's taskbar at the bottom shows various pinned icons.

This screenshot is identical to the one above, showing the same resume content for Altaf Alam. It includes the green 'Altaf' logo, the navigation menu, the main heading with contact info, the profile picture, the 'TECH GEEK || Web Developer.' section, and the 'Education' table. The browser taskbar at the bottom is also visible.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 18/07/23



Altaf

- [Home](#)
- [Contact](#)
- [Know More](#)

Contact Me!!!

Your Info:

Your Name:

Your Email: @gmail.com

Purpose for filling the form:

Mobile No: 9143143143

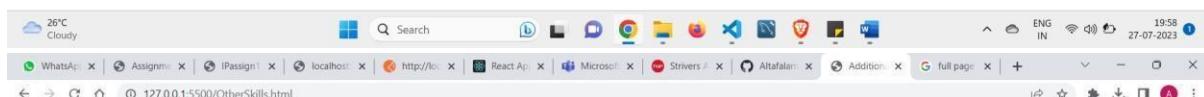
541, Noor Mahal Building, Adenwala Road,

King Circle, Fortungo, Mumbai - 400019

altaf526687@gmail.com



© Altaf Alam, All Rights Reserved

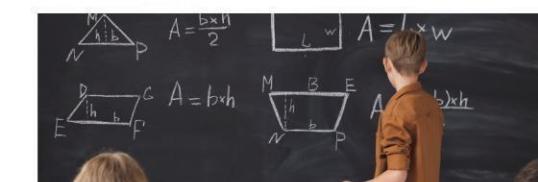


Video lectures

Maths Lecture 1 : LPP



Maths Lecture 2 : Probability



Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 18/07/23

The screenshot shows a web browser window with a video player displaying a chalkboard with geometric diagrams and formulas. Below the video are two podcast player cards. The first card is for 'My experience at DigiDunia' and the second for 'My experience at JpMorgan'. At the bottom is a Windows taskbar showing various icons and system status.

Conclusion :

When designing web pages using only HTML without using CSS can result in a lack of styling and visual appeal, leading to a plain and unattractive user experience. Additionally, the absence of CSS can make it challenging to achieve consistent formatting and responsive design across different devices and screen sizes. Furthermore, without CSS, maintaining and updating the page's layout and presentation becomes cumbersome and time-consuming.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 26/07/23

Assignment No 2

Aim : Using CSS and CSS3 enhance the web application developed in Assignment #1
Color, Background, Font, Table, List, CSS3 selectors, Pseudo classes, and Pseudo elements properties should be used to enhance the web pages.

Lab Outcome :

LO2: To expose students to CSS for formatting web pages.

Theory :

What is CSS?

CSS (Cascading Style Sheets) is a powerful stylesheet language used to control the presentation and layout of HTML documents. It allows web developers to apply various styles to HTML elements, making websites visually appealing and interactive. CSS follows a cascading approach, where styles are applied in a specific order, allowing for easy and efficient control of styles across an entire document.

Types of CSS:

Inline CSS:

Inline CSS is a way of applying styles directly to individual HTML elements using the style attribute. When you use inline CSS, the styles are defined right within the HTML tag of the element.

Pros:

Quick and easy to apply styles directly to specific elements.
Useful for small, one-off styling changes.

Cons:

Lack of separation of concerns, making it difficult to maintain and manage styles.
Not suitable for applying styles to multiple elements as it can lead to code duplication.

Internal CSS:

Internal CSS is defined within the <style> element, which is placed in the <head> section of the HTML document. It allows you to define styles that apply to multiple elements within the same HTML file.

Pros:

Separates the presentation (styles) from the content (HTML), leading to better code organization.

Allows for reusability of styles across multiple elements within the same HTML file.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 26/07/23

Cons:

Still lacks full separation of concerns compared to external CSS. May become hard to manage when stylesheets grow larger.

External CSS:

External CSS is a powerful and widely used way of applying styles to HTML documents. In this approach, the CSS styles are written in a separate .css file and then linked to the HTML document using the <link> tag in the <head> section.

Pros:

Best practice for large-scale projects as it promotes maintainability and reusability of styles. Allows for easy management of styles in a separate file.

Cons:

Requires an additional HTTP request to fetch the external CSS file (minimal impact on performance). May involve caching issues if the file is frequently updated.

2. CSS Font Properties

- a) font-family: Specify font(s) for text, using font names or generic families like "sans-serif," "serif," or "monospace."
- b) font-size: Set text size in pixels, em units, percentages, etc.
- c) font-weight: Control text thickness or boldness using values like "normal," "bold," or numeric values (100 to 900).
- d) font-style: Define font style, such as "normal," "italic," or "oblique."
- e) color: Change text color.

3. CSS Text Properties

- a) text-align: Align text horizontally within its container.
- b) text-decoration: Add visual effects to text, like underline, overline, or line-through.
- c) text-transform: Modify text capitalization.
- d) line-height: Set the spacing between lines of text.

4. Table Properties

- a) border-collapse: Control whether table borders are collapsed into a single border or separated.
- b) border: Define border properties for table elements, including width, style, and color.
- c) padding: Set space between content and table cell borders.
- d) text-align: Align table cell content horizontally.
- e) vertical-align: Align table cell content vertically.
- f) width: Set table or table cell width.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 26/07/23

5. List Properties

- a) list-style-type: Specify the style of the list item marker (bullet, number, etc.).
- b) list-style-image: Set an image as the list item marker.
- c) list-style-position: Determine whether the list item marker appears inside or outside the content flow.
- d) list-style: Shorthand for specifying all list-related properties in one declaration.

6. Background

- a) background-color: Set the background color of an element.
- b) background-image: Set an image as the background of an element.
- c) background-repeat: Control how the background image should repeat (repeat, repeat-x, repeat-y, no-repeat).
- d) background-size: Set the size of the background image (contain, cover).

7. CSS Selectors

- a) Universal Selector (*): Target all elements on the page.
- b) Type Selector (Element Selector): Target elements based on their HTML tag names, e.g., p, h1, div.
- c) Class Selector (.classname): Target elements with a specific class attribute, e.g., .container, .btn.
- d) ID Selector (#idname): Target a single element with a specific ID attribute, e.g., #header, #section1.
- e) Descendant Selector (Whitespace): Target elements that are descendants of another element, e.g., ul li, div p.
- f) Child Selector (>): Target direct children of an element, e.g., ul > li, div > p.
- g) Adjacent Sibling Selector (+): Target an element immediately following another element, e.g., h2 + p.
- h) General Sibling Selector (~): Target elements that are siblings of another element, e.g., h2 ~ p.
- i) Attribute Selector ([attr=value]): Target elements with a specific attribute and value, e.g., [type="submit"].

8. Pseudo Classes

- a) :hover: Apply styles when an element is being hovered over by the mouse pointer.
- b) :active: Apply styles when an element is being clicked or activated.
- c) :focus: Apply styles when an element gains focus (e.g., when clicked or tabbed into).
- d) :visited: Apply styles to visited links (Note: Limited for security reasons).
- e) :nth-child(n): Select elements based on their position within a parent container.
- f) :first-child: Select the first child element of its parent.
- g) :last-child: Select the last child element of its parent.
- h) :not(selector): Negate the selection of elements that match the specified selector.

9. Pseudo Elements

- a) ::before: Insert content before the selected element (requires the content property).
- b) ::after: Insert content after the selected element (requires the content property).
- c) ::first-line: Select the first line of text within an element.
- d) ::first-letter: Select the first letter of text within an element.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 26/07/23

e) ::selection: Apply styles to the portion of text selected by the user.

f) ::placeholder: Target input fields' placeholder text.

Source Code :

Style.css

The screenshot shows the Visual Studio Code interface with the 'style.css' file open. The code defines various CSS rules for a portfolio website, including body styling, navigation bar rules, article styling, and a table rule. A status bar at the bottom indicates the file is 90's Portfolio - css, has 55 lines, and is last updated by Altaf Alam on 02-08-2023.

```
body {
    margin: 0;
    padding: 0;
    font-family: Arial, sans-serif;
    background: linear-gradient(45deg, #4f72c2, #ebeded);
}

nav {
    display: flex;
    flex-direction: row;
    gap: 40px;
    text-align: center;
    background-color: #333;
    padding: 10px;
}

nav ul {
    list-style: none;
    padding: 0;
}

nav ul li {
    display: inline-block;
    margin-right: 20px;
    padding: 10px;
}

nav ul li::after {
    content: " Altaf Alam, last week added files via upload..";
    position: absolute;
    right: -10px;
    top: 0;
    transform: rotate(-90deg);
    color: red;
    font-size: small;
}

article {
    text-align: center;
    margin: 20px 0;
    animation: fadeInUp 2.5s;
    -webkit-animation: fadeInUp 2.5s;
}

img {
    border-radius: 50px;
}

hr {
    margin: 20px 0;
}

table {
    width: 100%;
    border-collapse: collapse;
}
```

The screenshot shows the Visual Studio Code interface with the 'style.css' file open. The code includes additional styling for a skill rating system, such as hovering effects and table row styling. A status bar at the bottom indicates the file is 90's Portfolio - css, has 183 lines, and is last updated by Altaf Alam on 02-08-2023.

```
table th,
table td {
    padding: 10px;
    border: 1px solid #333;
}

article ul {
    list-style: none;
    text-align: left;
}

article ul li {
    margin-bottom: 10px;
    display: flex;
    align-items: center;
    justify-content: space-between;
}

.skill-rating {
    color: gold;
}

.skill-rating:hover {
    color: orange;
}

article table tbody tr:nth-child(even) {
    background-color: #f2f2f2;
}

article table tbody tr:hover {
    background-color: #d9d9d9;
}

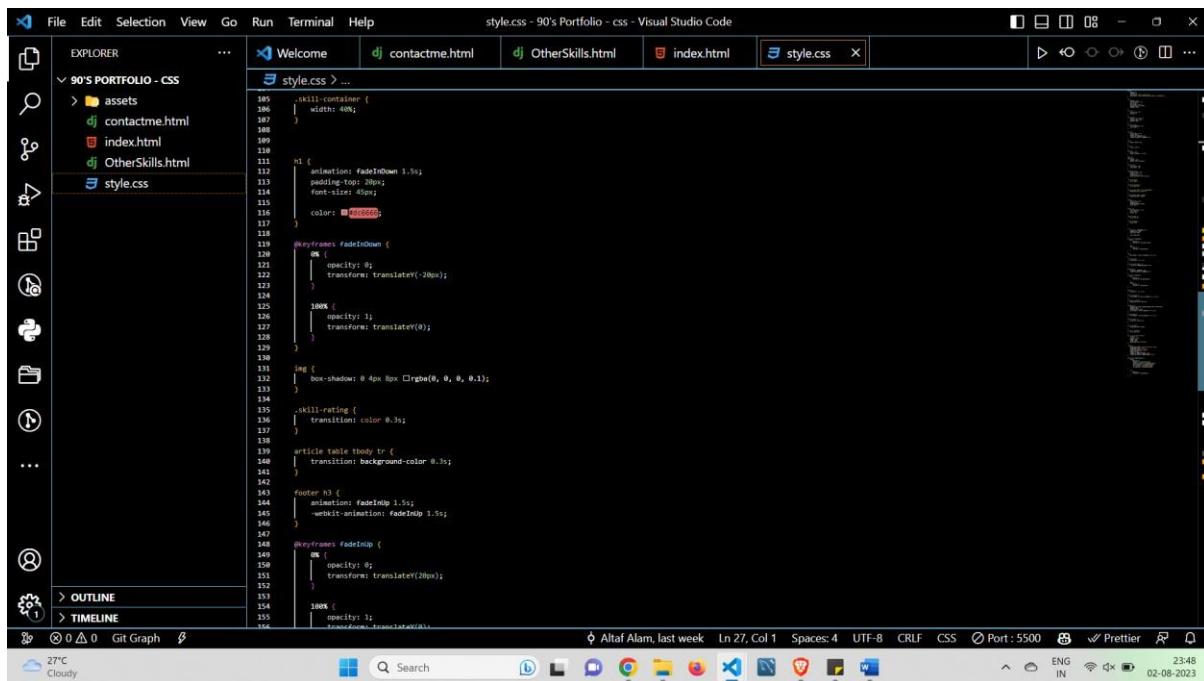
footer {
    text-align: center;
    padding: 10px;
    background-color: #333;
    color: #fff;
}

footer a {
    text-decoration: none;
    color: #fff;
    margin: 5px;
}

footer a:hover {
    color: #ccc;
}
```

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 26/07/23



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a folder named "90'S PORTFOLIO - CSS" containing "assets", "contactme.html", "index.html", "Otherskills.html", and "style.css".
- Editor:** The "style.css" file is open, displaying CSS code with line numbers. The code includes styles for ".skill-container", "h1", ".myProgress .progressBar", ".skill-rating", and ".footer h3". It uses transitions and keyframes for fading effects.
- Bottom Status Bar:** Shows "Altaf Alam, last week", "Ln 27, Col 1", "Spaces: 4", "UTF-8", "CRLF", "CSS", "Port : 5500", and system icons for weather, battery, and network.
- Bottom Taskbar:** Includes icons for Git Graph, Search, and various browser tabs.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 26/07/23

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a folder named "90'S PORTFOLIO - CSS" containing "assets", "contactme.html", "index.html", "OthersSkills.html", and "style.css".
- Editor:** The "style.css" file is open, displaying CSS code for styling a portfolio website. The code includes styles for footer links, article cards, navigation bars, and an "about" section.
- Bottom Status Bar:** Shows "Altaf Alam, last week", "Ln 27, Col 1", "Spaces: 4", "UTF-8", "CSS", "Port : 5500", and "Prettier".
- System Tray:** Shows weather (27°C, Cloudy), system icons, and a date/time stamp (02-08-2023).

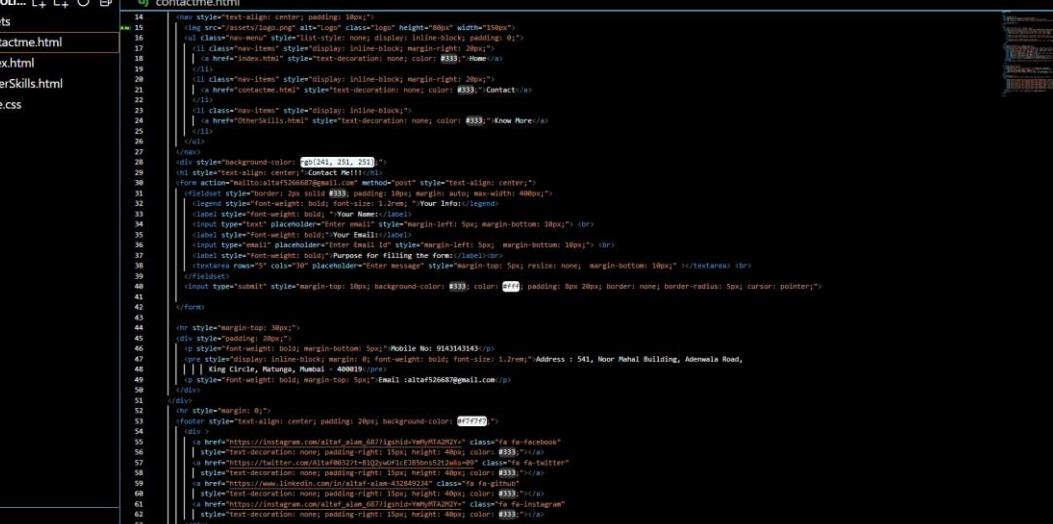
The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a folder named "90'S PORTFOLIO - CSS" containing "assets", "contactme.html", "index.html", "OthersSkills.html", and "style.css".
- Editor:** The "style.css" file is open, displaying CSS code for styling a portfolio website. The code includes styles for active navigation items and an "about" section.
- Bottom Status Bar:** Shows "Altaf Alam, last week", "Ln 27, Col 1", "Spaces: 4", "UTF-8", "CSS", "Port : 5500", and "Prettier".
- System Tray:** Shows weather (27°C, Cloudy), system icons, and a date/time stamp (02-08-2023).

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 26/07/23

Contactme.html



```
<html>
  <head>
    <title>contactme.html</title>
    <link href="style.css" type="text/css" rel="stylesheet"/>
  </head>
  <body>
    <div style="background-color: #f0f0f0; padding: 10px; border-radius: 5px; margin-bottom: 20px;">
      <div style="display: flex; justify-content: space-between; align-items: center; margin-bottom: 10px;">
        <img alt="Logo" style="height: 40px; width: 150px; border-radius: 5px; border: 1px solid black;"/>
        <div style="flex-grow: 1; text-align: center; padding: 0 10px;">
          <ul style="list-style-type: none; padding-left: 0; margin: 0; font-size: 0.9em; color: #333; font-weight: bold;">
            <li class="nav-item" style="display: inline-block; margin-right: 20px;">
              <a href="index.html" style="text-decoration: none; color: #333; text-decoration: none; color: #333; font-weight: bold;">Home</a>
            </li>
            <li class="nav-item" style="display: inline-block; margin-right: 20px;">
              <a href="contactme.html" style="text-decoration: none; color: #333; font-weight: bold;">Contact</a>
            </li>
            <li class="nav-item" style="display: inline-block;">
              <a href="OthersSkills.html" style="text-decoration: none; color: #333; font-weight: bold;">Know More</a>
            </li>
          </ul>
        </div>
      </div>
      <div style="background-color: #e0e0e0; padding: 10px; border-radius: 5px; border: 1px solid black; margin-bottom: 10px;">
        <div style="text-align: center; font-size: 0.9em; color: #333; font-weight: bold; margin-bottom: 10px;">
          <form action="mailto:altaf5266@gmail.com" method="post" style="text-align: center;">
            <div style="border: 1px solid #ccc; padding: 5px; width: 100%; max-width: 400px;">
              <label style="font-weight: bold; font-size: 1.2em; color: #333; margin-bottom: 5px;">Name</label>
              <input type="text" placeholder="Your Name" style="margin-left: 5px; margin-bottom: 10px;"/>
              <label style="font-weight: bold; font-size: 1.2em; color: #333; margin-bottom: 5px;">Email</label>
              <input type="email" placeholder="Enter email" style="margin-left: 5px; margin-bottom: 10px;"/>
              <label style="font-weight: bold; font-size: 1.2em; color: #333; margin-bottom: 5px;">Purpose</label>
              <input type="text" placeholder="Enter message" style="margin-top: 5px; resize: none; margin-bottom: 10px;"/>
            </div>
            <input type="submit" style="margin-top: 10px; background-color: #333; color: #fff; padding: 8px 20px; border: none; border-radius: 5px; cursor: pointer;"/>
          </form>
        </div>
      </div>
      <div style="margin-top: 30px;">
        <div style="padding: 10px; border: 1px solid #ccc; border-radius: 5px; margin-bottom: 5px;">Mobile No: 9163132343</div>
        <div style="display: flex; justify-content: space-between; align-items: center; margin-bottom: 5px;">
          <div style="display: flex; align-items: center; gap: 10px;">
            <div style="font-weight: bold; font-size: 1.2em; color: #333; margin-right: 5px;">Address :</div>
            <div style="font-size: 0.9em; color: #333; margin-right: 10px;">541, Noor Mahal Building, Adenwala Road, King Circle, Ratnagiri, Mumbai - 400019</div>
            <div style="font-weight: bold; margin-top: 5px;">Email :altaf5266@gmail.com</div>
          </div>
          <div style="margin-top: 8px;">
            <div style="text-align: center; padding: 20px; background-color: #333; color: #fff; border-radius: 5px; margin-bottom: 10px;">
              <a href="https://www.instagram.com/altaf5266/" style="color: #fff; text-decoration: none; font-size: 1.2em; margin-right: 10px;">Facebook</a>
              <a href="https://twitter.com/altaf5266" style="color: #fff; text-decoration: none; font-size: 1.2em; margin-right: 10px;">Twitter</a>
              <a href="https://www.linkedin.com/in/altaf5266/" style="color: #fff; text-decoration: none; font-size: 1.2em; margin-right: 10px;">GitHub</a>
              <a href="https://www.linkedin.com/in/altaf5266/" style="color: #fff; text-decoration: none; font-size: 1.2em; margin-right: 10px;">Instagram</a>
              <a href="https://www.linkedin.com/in/altaf5266/" style="color: #fff; text-decoration: none; font-size: 1.2em; margin-right: 10px;">YouTube</a>
            </div>
          </div>
        </div>
        <div style="text-align: center; font-size: 0.9em; color: #333; margin-top: 10px;">
          <h3><span style="color: #333; font-weight: bold;">© Altaf Alam, All Rights Reserved</span></h3>
        </div>
      </div>
    </div>
  </body>
</html>
```

OtherSkills.html

```
File Edit Selection View Go Run Terminal Help OtherSkills.html - 90's Portfolio - css - Visual Studio Code

EXPLORER ... Welcome dj contactme.html dj OthersSkills.html X index.html style.css

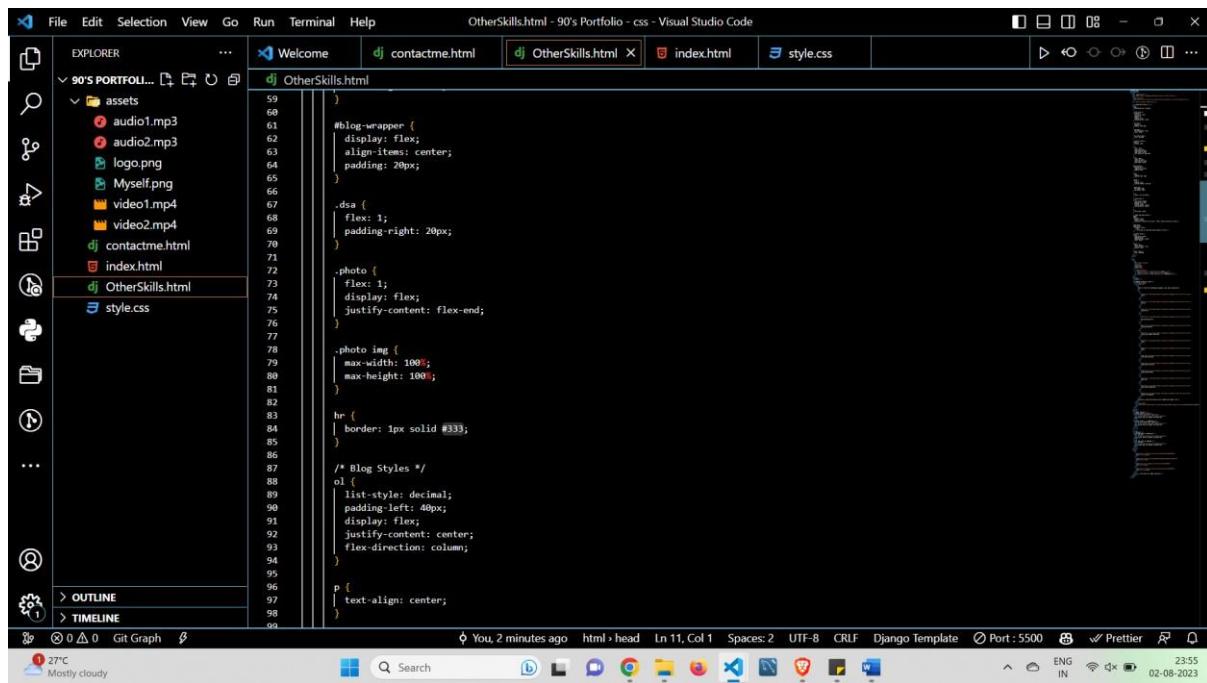
13 <html>
14   | body {
15   |   | background-color: #f2f2f2;
16   |
17   }
18
19   /* Nav Styles */
20   .nav-menu {
21     list-style: none;
22     padding: 0;
23     margin: 0;
24     display: flex;
25     background-color: #333;
26   }
27
28   .nav-items {
29     margin: 0;
30     padding: 10px 20px;
31   }
32
33   .nav-items a {
34     text-decoration: none;
35     color: white;
36   }
37
38   .nav-items a:hover {
39     color: #ffcc00;
40   }
41
42   /* Section Styles */
43   section {
44     padding: 20px;
45   }
46
47   h1 {
48     color: #333;
49     font-size: 32px;
50     margin-bottom: 10px;
51     text-align: center;
52     text-transform: uppercase;
53   }

You, 2 minutes ago html > head In 11, Col 1 Spaces: 2 UTF-8 CRLF Django Template Port: 5500 Prettier

27°C Mostly cloudy ENG IN 23:55 02-08-2023
```

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 26/07/23

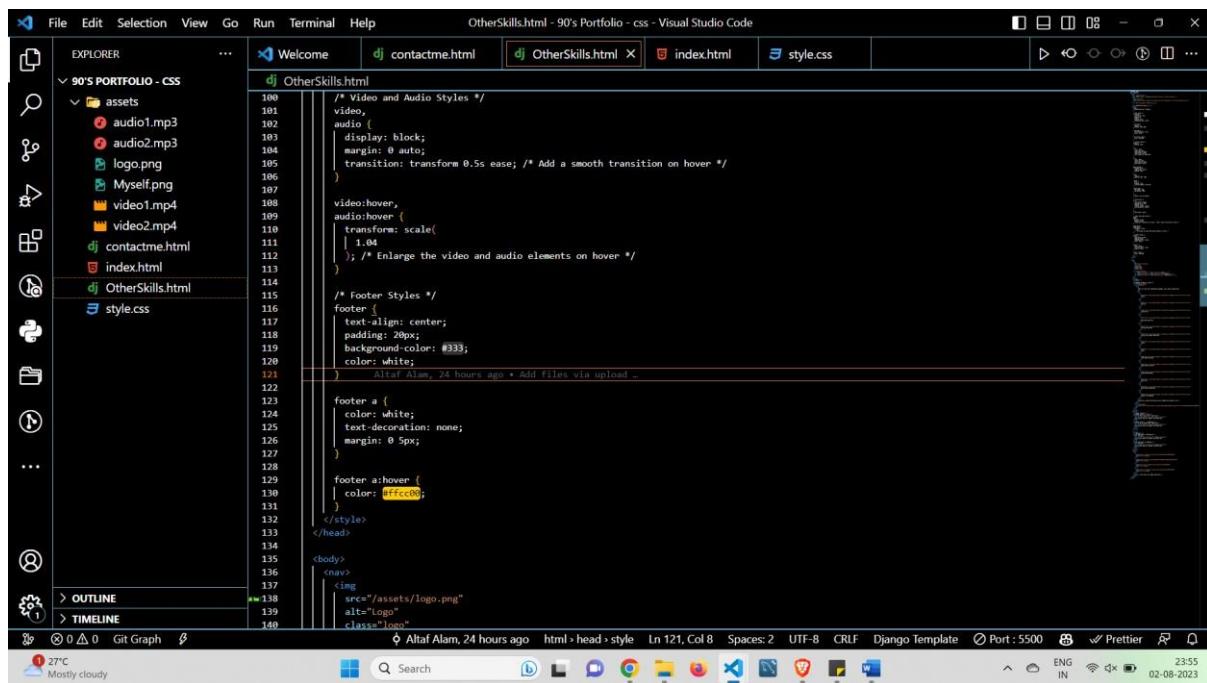


```
File Edit Selection View Go Run Terminal Help OtherSkills.html - 90's Portfolio - css - Visual Studio Code

EXPLORER ... Welcome dj contactme.html dj OtherSkills.html X index.html style.css

59 }
60 #blog-wrapper {
61   display: flex;
62   align-items: center;
63   padding: 20px;
64 }
65
66 .dsa {
67   flex: 1;
68   padding-right: 20px;
69 }
70
71 .photo {
72   flex: 1;
73   display: flex;
74   justify-content: flex-end;
75 }
76
77 .photo img {
78   max-width: 100%;
79   max-height: 100%;
80 }
81
82 hr {
83   border: 1px solid #333;
84 }
85
86 /* Blog Styles */
87 ol {
88   list-style: decimal;
89   padding-left: 40px;
90   display: flex;
91   justify-content: center;
92   flex-direction: column;
93 }
94
95 p {
96   text-align: center;
97 }
98
99
100 /* Video and Audio Styles */
101 video,
102 audio {
103   display: block;
104   margin: 0 auto;
105   transition: transform 0.5s ease; /* Add a smooth transition on hover */
106 }
107
108 video:hover,
109 audio:hover {
110   transform: scale(1.04);
111   /* Enlarge the video and audio elements on hover */
112 }
113
114 /* Footer Styles */
115 footer {
116   text-align: center;
117   padding: 20px;
118   background-color: #333;
119   color: white;
120 }
121 Altaf Alam, 24 hours ago • Add files via upload ...
122
123 footer a {
124   color: white;
125   text-decoration: none;
126   margin: 0 5px;
127 }
128
129 footer a:hover {
130   color: #ffccbc;
131 }
132
133 </style>
134 </head>
135
136 <body>
137   <nav>
138     
```

You, 2 minutes ago html > head Ln 11, Col 1 Spaces: 2 UTF-8 CRLF Django Template Port : 5500 Prettier 2355 02-08-2023



```
File Edit Selection View Go Run Terminal Help OtherSkills.html - 90's Portfolio - css - Visual Studio Code

EXPLORER ... Welcome dj contactme.html dj OtherSkills.html X index.html style.css

100 /* Video and Audio Styles */
101 video,
102 audio {
103   display: block;
104   margin: 0 auto;
105   transition: transform 0.5s ease; /* Add a smooth transition on hover */
106 }
107
108 video:hover,
109 audio:hover {
110   transform: scale(1.04);
111   /* Enlarge the video and audio elements on hover */
112 }
113
114 /* Footer Styles */
115 footer {
116   text-align: center;
117   padding: 20px;
118   background-color: #333;
119   color: white;
120 }
121 Altaf Alam, 24 hours ago • Add files via upload ...
122
123 footer a {
124   color: white;
125   text-decoration: none;
126   margin: 0 5px;
127 }
128
129 footer a:hover {
130   color: #ffccbc;
131 }
132
133 </style>
134 </head>
135
136 <body>
137   <nav>
138     
```

Altaf Alam, 24 hours ago html > head > style Ln 121, Col 8 Spaces: 2 UTF-8 CRLF Django Template Port : 5500 Prettier 2355 02-08-2023

Output :

Altaf Alam , 02 , T11

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 26/07/23

The screenshot shows a web browser window with the URL localhost:5500/OtherSkills.html. The page has a header with 'Home', 'Contact', and 'Know More' links. A logo for 'Altaf' is on the left. The main content area has a section titled 'BLOGS' and a sub-section titled 'Roadmap to DSA in 7 days'. Below this, there is a heading 'Learn at least one Programming Language, Learn about Complexities' followed by a numbered list of 14 items. At the bottom, a footer says 'Practice, practice and practice more Compete and become a pro'. To the right of the text, there is a cartoon illustration of a man with a beard looking at a laptop screen displaying some code.

Learn at least one Programming Language, Learn about Complexities

1. [Array](#)
2. [String](#)
3. [Linked List](#)
4. [Searching Algorithm](#)
5. [Sorting Algorithm](#)
6. [Divide and Conquer Algorithm](#)
7. [Stack](#)
8. [Queue](#)
9. [Tree Data Structure](#)
10. [Graph Data Structure](#)
11. [Greedy Methodology](#)
12. [Recursion](#)
13. [Backtracking Algorithm](#)
14. [Dynamic Programming](#)

Practice, practice and practice more Compete and become a pro

The screenshot shows a web browser window with the URL localhost:5500/OtherSkills.html. The video player is playing a clip titled 'Maths Lecture 2 : Probability'. The video frame shows a teacher writing formulas on a chalkboard. The formulas include:
$$A = \frac{b \times h}{2}$$
$$A = l \times w$$
$$A = b \times h$$
$$A = (l + b) \times h$$
$$A = l \times w$$

The chalkboard also has some geometric diagrams of triangles and rectangles labeled with letters A through P.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 26/07/23

The screenshot shows a Microsoft Edge browser window. At the top, there are tabs for 'Additional Skills' and 'Microsoft Teams'. The main content area displays a video player with a thumbnail of two people, a progress bar at 0:00 / 0:13, and a play button. Below the video player, there is a section titled 'PODCASTS' with two entries: 'My experience at Digiidunia' and 'My experience at JpMorgan', each with a play button and a progress bar.

The screenshot shows a Microsoft Edge browser window with a tab for 'Personal Portfolio'. The main content area displays a personal portfolio website for Altaf Alam. The header features a logo with the name 'Altaf' and a photo of Altaf. The page includes contact information (+919876543210, altaf526687@gmail.com), a title 'TE IT ENGINEER(TSEC).', and a subtitle 'TECH GEEK || Web Developer.'. Below this, there is a section titled 'Education' with a table:

Year	College	Description
2025(exp)	Thadomal Shahani Engineering College	Maintained 9.12 CGPA(till 3rd sem).
2021	Guru Nanak Khalsa College	Secured 97.98% percentile in MHTCET STATE Level Exam.
2021	Guru Nanak Khalsa College	Secured 90.83% in HSC exam.
2019	SIWS High School	Secured 76.8% overall with 90+ in maths in SSC exam.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 26/07/23

The screenshot shows a web browser window with the URL localhost:5500/index.html. The page displays a table of education history and a skills section.

Year	College	Description
2025(exp)	Thadomal Shahani Engineering College	Maintained 9.12 CGPA(till 3rd sem).
2021	Guru Nanak Khalsa College	Secured 97.98% percentile in MHTCET STATE Level Exam.
2021	Guru Nanak Khalsa College	Secured 90.83% in HSC exam.
2019	SIWS High School	Secured 76.8% overall with 90+ in maths in SSC exam.

Skills

- FULL STACK WEB DEVELOPER ★★★★★
- Python ★★★★★
- Cpp & C ★★★★☆
- Java ★★★★☆
- LEADERSHIP & TEAMWORK ★★★★★

At the bottom of the page, there are social media sharing icons (Facebook, Twitter, LinkedIn, etc.) and a copyright notice: © Altaf Alam, All Rights Reserved. The browser status bar shows the date 02-08-2023 and time 23:56.

The screenshot shows a web browser window with the URL localhost:5500/contactme.html. The page features a large red header "Contact Me!!!". Below it is a form titled "Your Info:" with fields for "Your Name" and "Your Email", and a larger text area for "Purpose for filling the form" with placeholder text "Enter message". A "Submit" button is at the bottom of the form.

Mobile No: 9143143143
Address : 541, Noor Mahal Building, Adenwala Road,
King Circle, Matunga, Mumbai - 400019
Email : altaf526687@gmail.com

At the bottom of the page, there are social media sharing icons (Facebook, Twitter, LinkedIn, etc.) and a copyright notice: © Altaf Alam, All Rights Reserved. The browser status bar shows the date 02-08-2023 and time 23:57.

Conclusion :

The web application developed in Assignment 1 has been greatly enhanced using CSS and CSS3 features. The strategic implementation of CSS properties such as color, background, font, table, and list styles has not only improved the visual appeal but also enhanced the overall user experience. The careful and creative use of these CSS features has contributed to

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 26/07/23

creating a compelling and attractive web application that is not only visually appealing but also userfriendly and efficient.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

Assignment No 3

Aim : Develop a web page using the Bootstrap framework with following atleast these classes used: Grid system, Forms, Button, Navbar, Breadcrumb, Jumbotron.

Lab Outcome :

LO3: To expose students to developing responsive layouts.

Theory :

What is CSS?

Bootstrap is a popular open-source front-end framework developed by Twitter. It provides a set of pre-designed CSS and JavaScript components that help developers build responsive and mobile-friendly websites and web applications. Bootstrap makes it easier to create consistent and visually appealing user interfaces across different devices and screen sizes.

Uses of Bootstrap:

1. Responsive Layout: Bootstrap's grid system allows developers to create responsive layouts that adapt to various screen sizes, ensuring a consistent user experience on different devices.
2. UI Components: Bootstrap offers a wide range of UI components like buttons, forms, navigation bars, carousels, and more, which can be easily customized and integrated into web projects.
3. Mobile-First Approach: Bootstrap follows a mobile-first approach, meaning the design and development start from small devices and scale up for larger ones, improving performance on mobile devices.
4. Customization: Developers can customize the Bootstrap framework by using its SASS variables and mixins to match their specific project requirements.
5. Cross-Browser Compatibility: Bootstrap is designed to work efficiently on all modern web browsers, ensuring that websites look consistent for users across different browsers.

Bootstrap Classes in Detail:

1. Grid System:

Sure! Here's a more detailed explanation of each Bootstrap class and component:

1. Grid System:

The Bootstrap grid system is based on a responsive, 12-column layout. It provides a way to create responsive designs by dividing the page horizontally into rows and columns. The `container` class creates a fixed-width container for the content, while `row` defines a horizontal row to hold columns. Each column is defined using classes like `col-*`, where `*`

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

represents the number of columns the element should span. For example, ` .col-md-6` means the element will span six columns on medium-sized devices and larger.

2. Forms:

Bootstrap's form classes allow developers to style form elements in a consistent and visually appealing way. The ` .form-group` class is used to group form elements together, such as labels and input fields. The ` .form-control` class styles input elements like text fields, select dropdowns, and text areas. Additionally, the ` .btn` class can be applied to buttons within the form for consistent styling.

3. Buttons:

Bootstrap provides various button styles using the ` .btn` class. Buttons can be styled as primary, secondary, success, danger, warning, info, and light. Each style has its own color scheme. For example, ` .btn-primary` applies the primary color theme to the button.

4. Navbar:

The Bootstrap navbar class (` navbar`) helps create a navigation bar at the top of the web page. It provides a responsive and collapsible menu for smaller screens. The navigation links are defined inside a ` .navbar-nav` list, and the ` .navbar-brand` class is used for the website logo. The ` .navbar-toggler` class creates a button to toggle the navigation menu on smaller screens.

5. Breadcrumbs:

Breadcrumbs provide a hierarchical navigation path to help users understand their current location within the website. The ` .breadcrumb` class creates a horizontal list of breadcrumb items, each represented by a list item with the ` .breadcrumb-item` class. The last item should have the ` .active` class to indicate the current page.

6. Jumbotron:

The Jumbotron class (` jumbotron`) creates a large, eye-catching container for showcasing important content. It is typically used for headings, subheadings, and calls-to-action. The Jumbotron provides a clean and prominent way to display key messages on a web page.

7. Carousel:

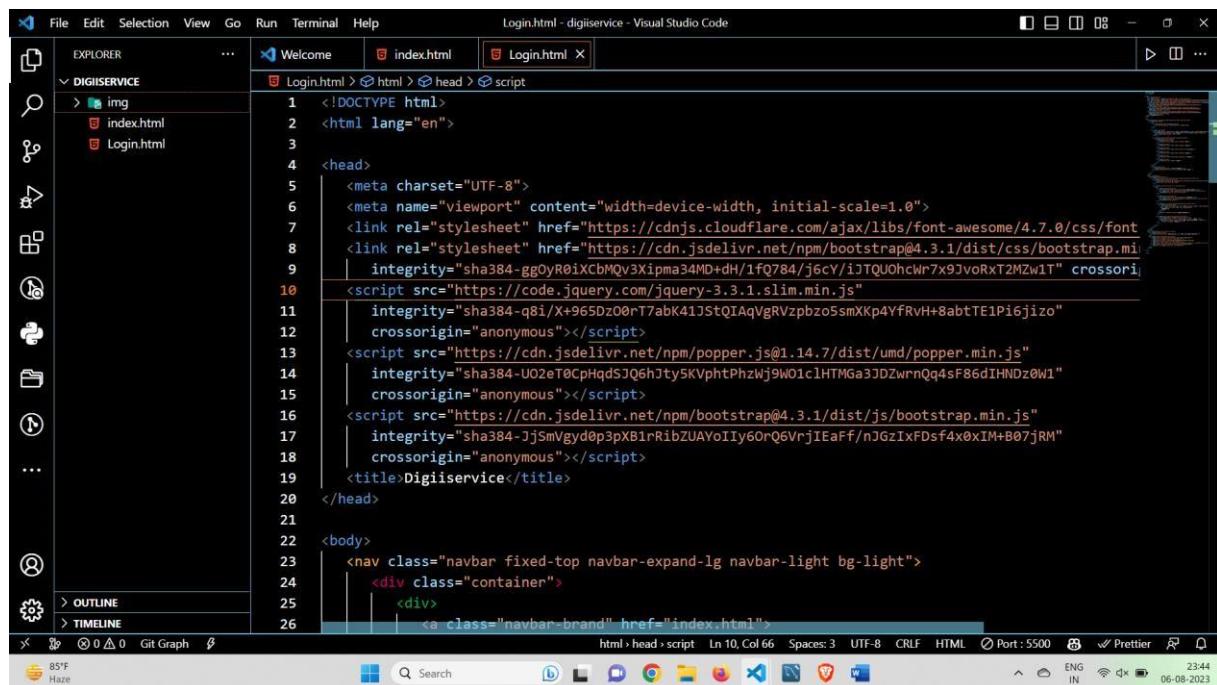
Bootstrap's carousel component allows you to create an image slider or a carousel of content. The carousel is based on a series of ` .carousel-item` elements, each containing the content to be displayed. The ` .carousel-indicators` class provides navigation dots, and the ` .carousel-control-prev` and ` .carousel-control-next` classes create navigation arrows.

These Bootstrap classes and components simplify the process of building responsive, visually appealing, and user-friendly websites and web applications. By utilizing these elements, developers can create consistent and professional-looking interfaces with less effort and code.

Name : Altaf Alam

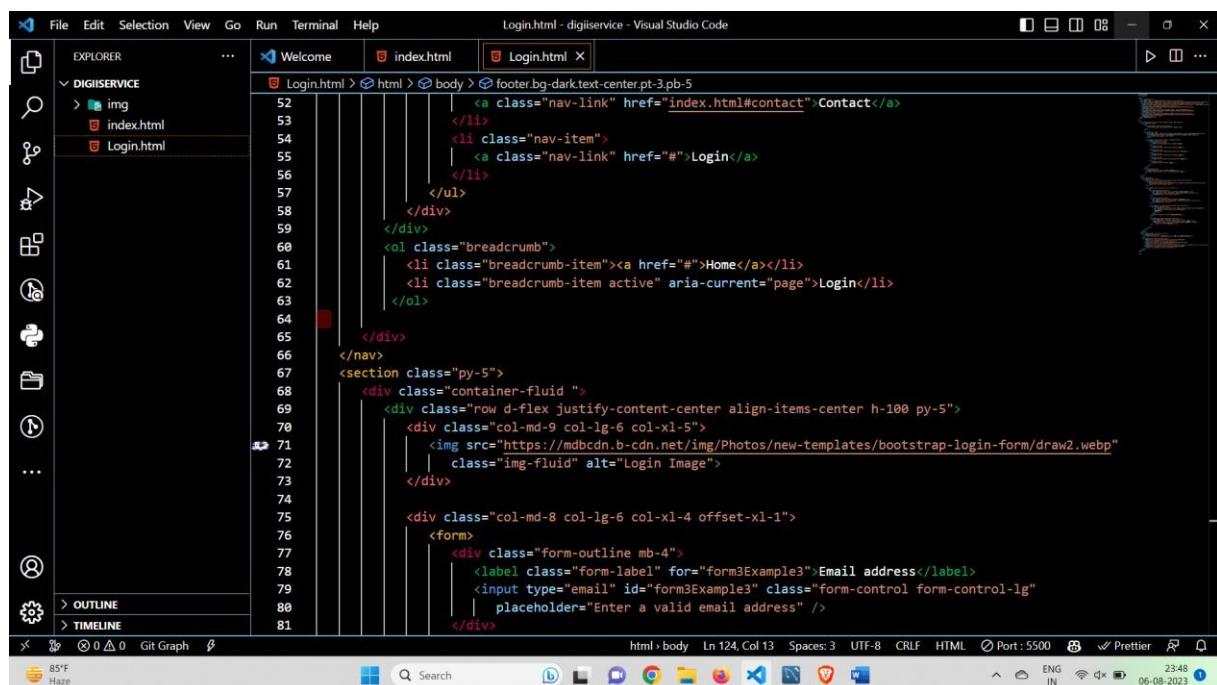
Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

Source Code : Index.html



The screenshot shows the Visual Studio Code interface with the 'index.html' file open in the editor. The code is a basic HTML page with a header section including meta tags, links to CSS and JS files, and a title. The browser tab at the top is 'Login.html - digiservice - Visual Studio Code'. The status bar at the bottom shows the file is an 'html > head > script' file, line 10, column 66, with other details like port 5500 and Prettier status.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css" integrity="sha384-gg0yR0iXcbMqV3Xipma34MD+dH/1fQ784/j6cY/iJTQUhcWr7x9JvoRxT2Mzw1T" crossorigin="anonymous">
    <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-gg0yR0iXcbMqV3Xipma34MD+dH/1fQ784/j6cY/iJTQUhcWr7x9JvoRxT2Mzw1T" crossorigin="anonymous"></script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.7/dist/umd/popper.min.js" integrity="sha384-UO2eT0CphdsJQ6hJty5KvphPhzWj9W0ic1HTMGa3JDZwnQq4sF86dIHNDz0W1" crossorigin="anonymous"></script>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.3.1/dist/js/bootstrap.min.js" integrity="sha384-JJSmVgdy0p3pXB1rRibZUAYoIIy60rQ6VrjIEaFF/nJGzIxFDsf4x0xIM+B07jRM" crossorigin="anonymous"></script>
<title>Digiiservice</title>
</head>
<body>
    <nav class="navbar fixed-top navbar-expand-lg navbar-light bg-light">
        <div class="container">
            <div>
                <a class="navbar-brand" href="index.html">
```



The screenshot shows the Visual Studio Code interface with the 'Login.html' file open in the editor. The code includes a navigation bar with links to 'Contact' and 'Login', and a breadcrumb trail. Below it is a section with a form containing an email input field. The browser tab at the top is 'Login.html - digiservice - Visual Studio Code'. The status bar at the bottom shows the file is an 'html > body > footer' file, line 124, column 13, with other details like port 5500 and Prettier status.

```
<a class="nav-link" href="#contact">Contact</a>
<li class="nav-item">
    <a class="nav-link" href="#">Login</a>
</li>
</ul>
</div>
</div>
<div class="breadcrumb">
    <ol class="list-group list-group-flush">
        <li class="breadcrumb-item"><a href="#">Home</a></li>
        <li class="breadcrumb-item active" aria-current="page">Login</li>
    </ol>
</div>
</nav>
<section class="py-5">
    <div class="container-fluid">
        <div class="row d-flex justify-content-center align-items-center h-100 py-5">
            <div class="col-md-9 col-lg-6 col-xl-5">
                
            </div>
            <div class="col-md-8 col-lg-6 col-xl-4 offset-xl-1">
                <form>
                    <div class="form-outline mb-4">
                        <label class="form-label" for="form3Example3">Email address</label>
                        <input type="email" id="form3Example3" class="form-control form-control-lg" placeholder="Enter a valid email address" />
                    </div>
                </form>
            </div>
        </div>
    </div>
</section>
```

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

The screenshot shows the Visual Studio Code interface with the following details:

- Title Bar:** Login.html - digiservice - Visual Studio Code
- File Explorer (Left):** Shows a folder named "DIGISERVICE" containing "index.html" and "Login.html".
- Editor Area (Center):** Displays the content of the "Login.html" file. The code includes HTML for a login form, a footer with social media links, and a copyright notice.
- Bottom Status Bar:** Shows file paths (html > head > script), line numbers (Ln 10, Col 66), spaces (Spaces: 3), encoding (UTF-8), and port information (Port: 5500). It also includes icons for Git Graph, Prettier, and a terminal.
- System Tray (Bottom Right):** Shows battery level (85%), Haze icon, and system status (ENG IN).

```
<div class="container-fluid">
  <div class="row d-flex justify-content-center align-items-center h-100 py-5">
    <div class="col-md-9 col-lg-6 col-xl-5">
      
    </div>
    <div class="col-md-8 col-lg-6 col-xl-4 offset-xl-1">
      <form>
        <div class="form-outline mb-4">
          <label class="form-label" for="formExampleEmail">Email address</label>
          <input type="email" id="formExampleEmail" class="form-control form-control-lg"
                 placeholder="Enter a valid email address" />
        </div>

        <div class="form-outline mb-3">
          <label class="form-label" for="formExamplePassword">Password</label>
          <input type="password" id="formExamplePassword" class="form-control form-control-lg"
                 placeholder="Enter password" />
        </div>

        <div class="d-flex justify-content-between align-items-center">
          <div class="form-check mb-0">
            <input checked="" type="checkbox" name="checkbox" value="" id="form2Example33" />
            <label class="form-check-label" for="form2Example33">
              Remember me
            </label>
          </div>
          <a href="#" class="text-body">Forgot password?</a>
        </div>

        <div class="text-center text-lg-start mt-4 pt-2">
          <button type="button" class="btn btn-primary btn-lg"
                 style="padding-left: 2.5rem; padding-right: 2.5rem;">Login</button>
          <p class="small fw-bold mt-2 pt-1 mb-0">Don't have an account? <a href="#">Register</a></p>
        </div>
      </form>
    </div>
  </div>
</section>
<footer class="bg-dark text-center pt-3 pb-5">
  <div class="container pb-4">
    <div class="text-white mb-3">Digiiservice &copy; 2023, All Rights Reserved</div>
    <div class="d-flex justify-content-center">
      <a href="https://instagram.com/altaf_alam_687?igshid=YmMyMTA2M2Y="
         class="fa fa-facebook fa-2x mx-2 text-white"></a>
      <a href="https://twitter.com/Altaf00327tBQ2zywUfcCEJ85bns52t2w&s=09"
         class="fa fa-twitter fa-2x mx-2 text-white"></a>
      <a href="https://www.linkedin.com/in/altaf-alam-432849234" class="fa fa-github fa-2x mx-2 text-white">
        <a href="https://instagram.com/altaf_alam_687?igshid=YmMyMTA2M2Y="
           class="fa fa-instagram fa-2x mx-2 text-white"></a>
    </div>
  </div>
</footer>
</body>
</html>
```

The screenshot shows the Visual Studio Code interface with the following details:

- Title Bar:** Login.html - digiservice - Visual Studio Code
- File Explorer (Left):** Shows a folder named "DIGISERVICE" containing "index.html" and "Login.html".
- Editor Area (Center):** Displays the content of the "Login.html" file. The code includes HTML for a login form, a footer with social media links, and a copyright notice. There are several red highlights and underlines in the code, indicating errors or warnings.
- Bottom Status Bar:** Shows file paths (html > head > script), line numbers (Ln 10, Col 66), spaces (Spaces: 3), encoding (UTF-8), and port information (Port: 5500). It also includes icons for Git Graph, Prettier, and a terminal.
- System Tray (Bottom Right):** Shows battery level (85%), Haze icon, and system status (ENG IN).

```
<div class="container-fluid">
  <div class="row d-flex justify-content-center align-items-center h-100 py-5">
    <div class="col-md-9 col-lg-6 col-xl-5">
      
    </div>
    <div class="col-md-8 col-lg-6 col-xl-4 offset-xl-1">
      <form>
        <div class="form-outline mb-4">
          <label class="form-label" for="formExampleEmail">Email address</label>
          <input type="email" id="formExampleEmail" class="form-control form-control-lg"
                 placeholder="Enter a valid email address" />
        </div>

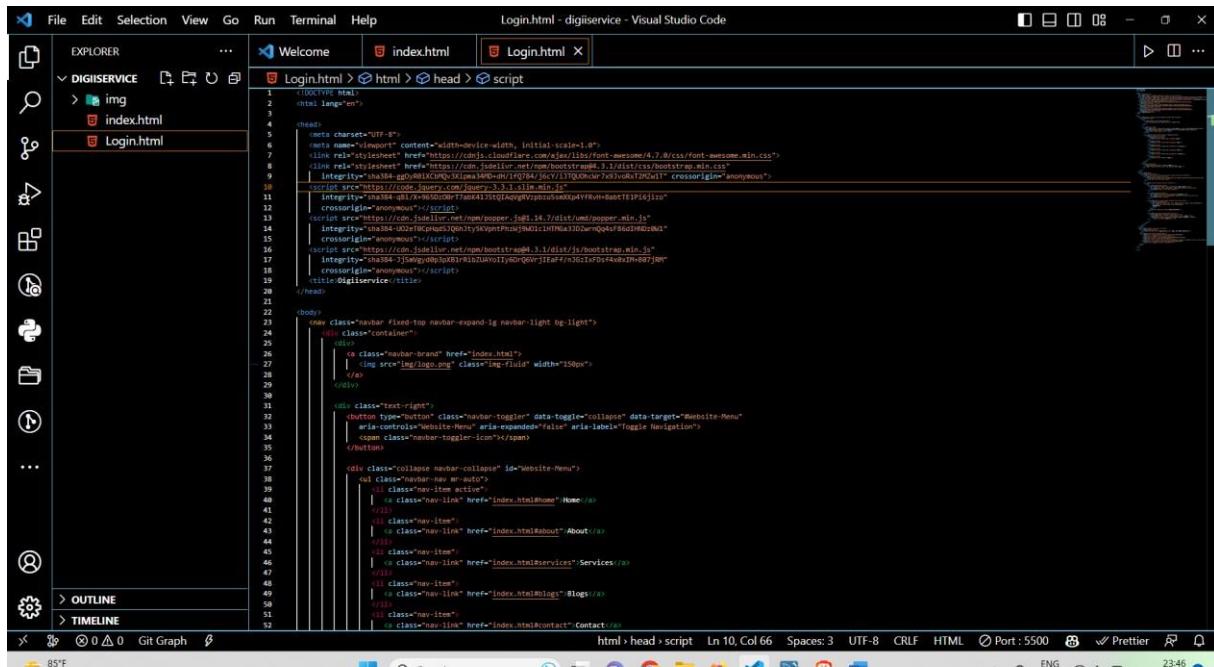
        <div class="form-outline mb-3">
          <label class="form-label" for="formExamplePassword">Password</label>
          <input type="password" id="formExamplePassword" class="form-control form-control-lg"
                 placeholder="Enter password" />
        </div>

        <div class="d-flex justify-content-between align-items-center">
          <div class="form-check mb-0">
            <input checked="" type="checkbox" name="checkbox" value="" id="form2Example33" />
            <label class="form-check-label" for="form2Example33">
              Remember me
            </label>
          </div>
          <a href="#" class="text-body">Forgot password?</a>
        </div>

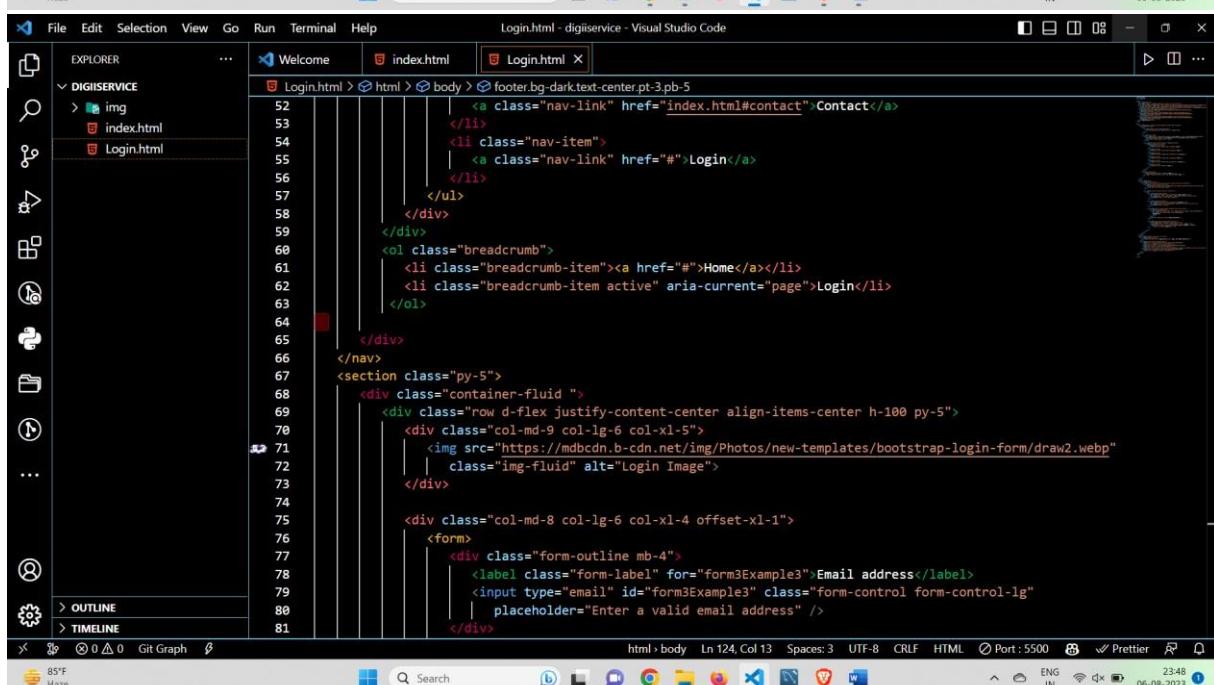
        <div class="text-center text-lg-start mt-4 pt-2">
          <button type="button" class="btn btn-primary btn-lg"
                 style="padding-left: 2.5rem; padding-right: 2.5rem;">Login</button>
          <p class="small fw-bold mt-2 pt-1 mb-0">Don't have an account? <a href="#">Register</a></p>
        </div>
      </form>
    </div>
  </div>
</section>
<footer class="bg-dark text-center pt-3 pb-5">
  <div class="container pb-4">
    <div class="text-white mb-3">Digiiservice &copy; 2023, All Rights Reserved</div>
    <div class="d-flex justify-content-center">
      <a href="https://instagram.com/altaf_alam_687?igshid=YmMyMTA2M2Y="
         class="fa fa-facebook fa-2x mx-2 text-white"></a>
      <a href="https://twitter.com/Altaf00327tBQ2zywUfcCEJ85bns52t2w&s=09"
         class="fa fa-twitter fa-2x mx-2 text-white"></a>
      <a href="https://www.linkedin.com/in/altaf-alam-432849234" class="fa fa-github fa-2x mx-2 text-white">
        <a href="https://instagram.com/altaf_alam_687?igshid=YmMyMTA2M2Y="
           class="fa fa-instagram fa-2x mx-2 text-white"></a>
    </div>
  </div>
</footer>
</body>
</html>
```

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

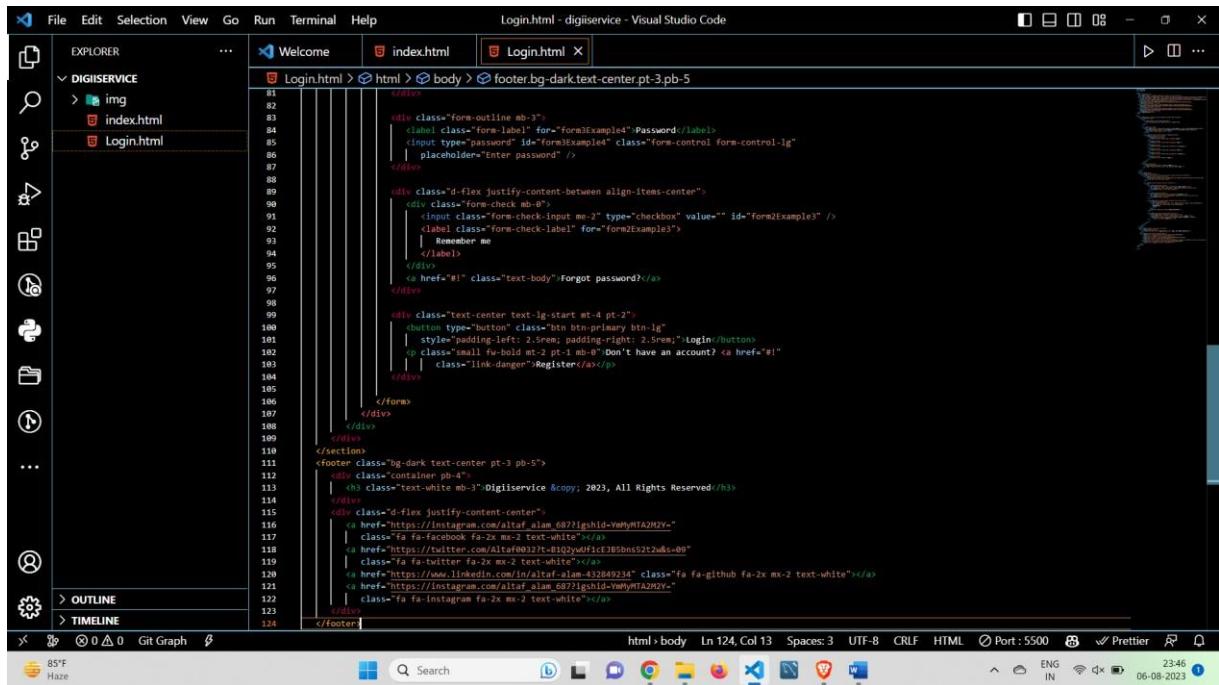


```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@3.3.1/dist/css/bootstrap.min.css" integrity="sha384-gO8XzJxPuQgDyRlCQVjwPmQWdHr7Q0hW73o0kxM01" crossorigin="anonymous">
    <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8kXuPq+QVZtYXuJNcK2Z4Ntqf+jdFpWZI4ZqEJwGgOgHdXpHvZlWZLJU" crossorigin="anonymous">
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.12.9/dist/umd/popper.min.js" integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScFdeoEjEd1u4eLCwTS7RD4uJ3" crossorigin="anonymous">
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.3.1/dist/js/bootstrap.min.js" integrity="sha384-J3tSC1SupdJ4/lnXo2p0k0Q9KuO8XWZJQmH01" crossorigin="anonymous">
    <script src="https://digiservice.com/">
  </head>
  <body>
    <nav class="navbar fixed-top navbar-expand-lg navbar-light bg-light">
      <div class="container">
        <div class="header-brand" href="index.html">
          | 
        </div>
        <div class="text-right">
          <button type="button" class="navbar-toggler" data-toggle="collapse" data-target="#Website-Menu" aria-controls="Website-Menu" aria-expanded="false" aria-label="Toggle Navigation">
            <span class="navbar-toggler-icon"></span>
          </button>
          <div class="collapse navbar-collapse" id="Website-Menu">
            <ul class="nav-item">
              | <a class="nav-link active" href="index.html#home">Home</a>
              | <a class="nav-link" href="index.html#about">About</a>
              | <a class="nav-item">
                | <a class="nav-link" href="index.html#services">Services</a>
                | <a class="nav-item">
                  | <a class="nav-link" href="index.html#blogs">Blogs</a>
                  | <a class="nav-item">
                    | <a class="nav-link" href="index.html#contact">Contact</a>
                </a>
              </a>
            </ul>
          </div>
        </div>
        <ol class="breadcrumb">
          <li class="breadcrumb-item"><a href="#">Home</a></li>
          <li class="breadcrumb-item active" aria-current="page">Login</li>
        </ol>
      </div>
    </nav>
    <section class="py-5">
      <div class="container-fluid ">
        <div class="row d-flex justify-content-center align-items-center h-100 py-5">
          <div class="col-md-9 col-lg-6 col-xl-5">
            
          </div>
          <div class="col-md-8 col-lg-6 col-xl-4 offset-xl-1">
            <form>
              <div class="form-outline mb-4">
                <label class="form-label" for="form3Example3">Email address</label>
                <input type="email" id="form3Example3" class="form-control form-control-lg" placeholder="Enter a valid email address" />
              </div>
            </form>
          </div>
        </div>
      </div>
    </section>
  </body>
</html>
```



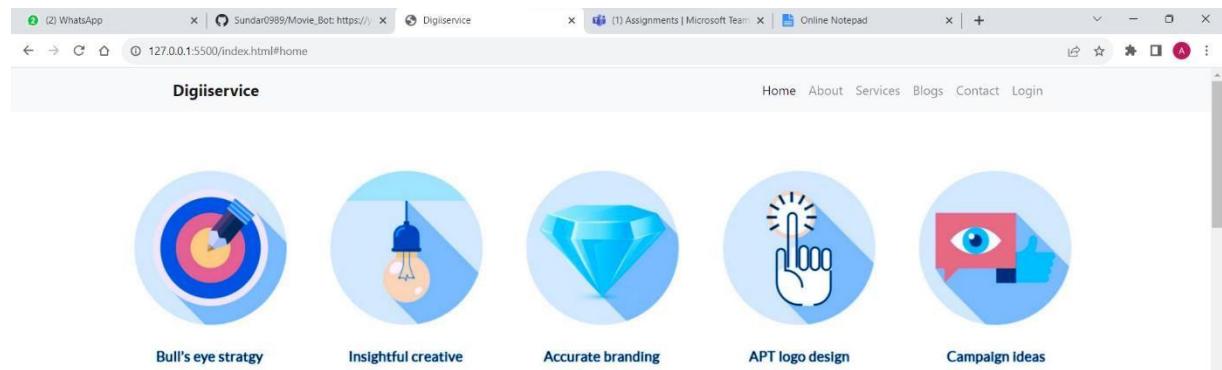
Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

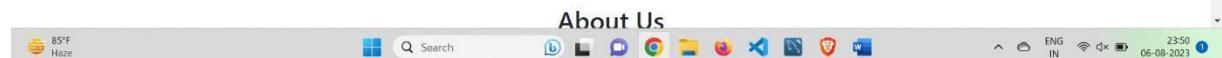


```
<div>
  <div>
    <div><img alt="Digiiservice logo" /></div>
    <div>Welcome</div>
    <div>index.html</div>
    <div>Login.html</div>
  </div>
  <div>
    <div><form>
      <div><input type="text" id="formExample1" placeholder="Enter email" /></div>
      <div><input type="password" id="formExample4" placeholder="Enter password" /></div>
      <div><div class="d-flex justify-content-between align-items-center">
        <div><input checked="" type="checkbox" value="2" id="formExample3" />
          <label>Remember me</label>
        </div>
        <a href="#" class="text-body">Forgot password?
      </div></div>
      <div><button type="button" class="btn btn-primary w-100" style="padding-left: 2.5rem; padding-right: 2.5rem;">Login</button>
        <p class="small fw-bold ms-2 pt-2 mb-0">Don't have an account? <a href="#">Register</a></p>
      </div>
    </form>
  </div>
  <div><div><small>© Digiiservice &copy; 2023, All Rights Reserved</small>
    <div><small><a href="https://www.instagram.com/altaf_alam_43284923/">fa-instagram fa-2x mx-2 text-white</a>
      <a href="https://twitter.com/Altaf00327810Qywd1c1E3B50ns52t2w&s=09" class="fa fa-twitter fa-2x mx-2 text-white"></a>
      <a href="https://www.linkedin.com/in/altaf-alam-6871ghsh1-yMyHtAZJH2Y/" class="fa fa-github fa-2x mx-2 text-white"></a>
      <a href="https://www.instagram.com/altaf_alam_6871ghsh1-yMyHtAZJH2Y/" class="fa fa-instagram fa-2x mx-2 text-white"></a>
    </div>
  </div>
</div>
</div>
```

Output :



Value added Branding & Advertising



Name : Altaf Alam

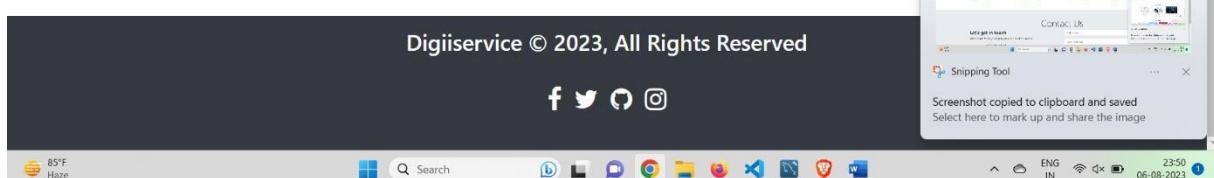
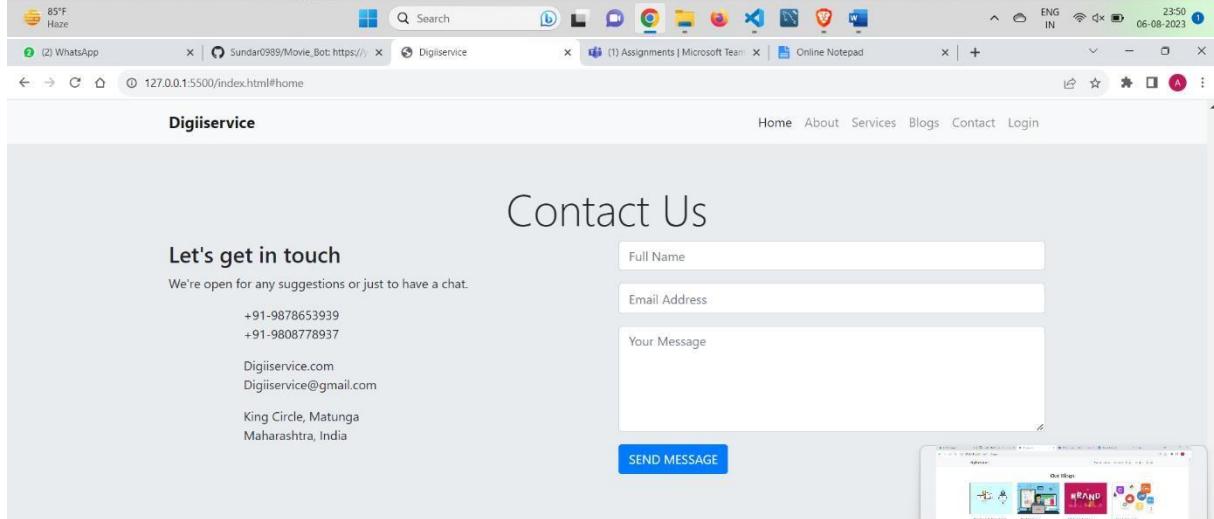
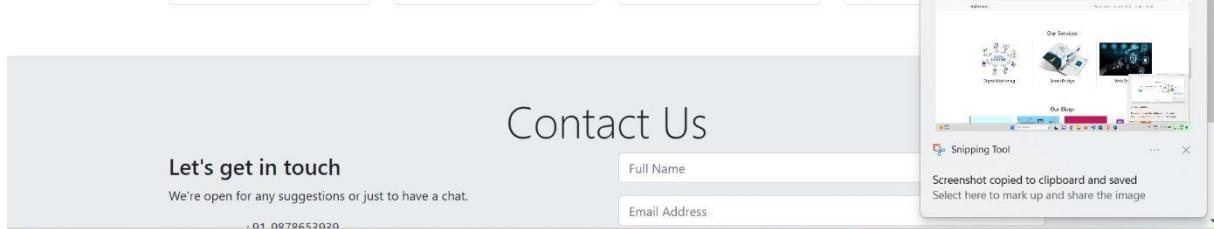
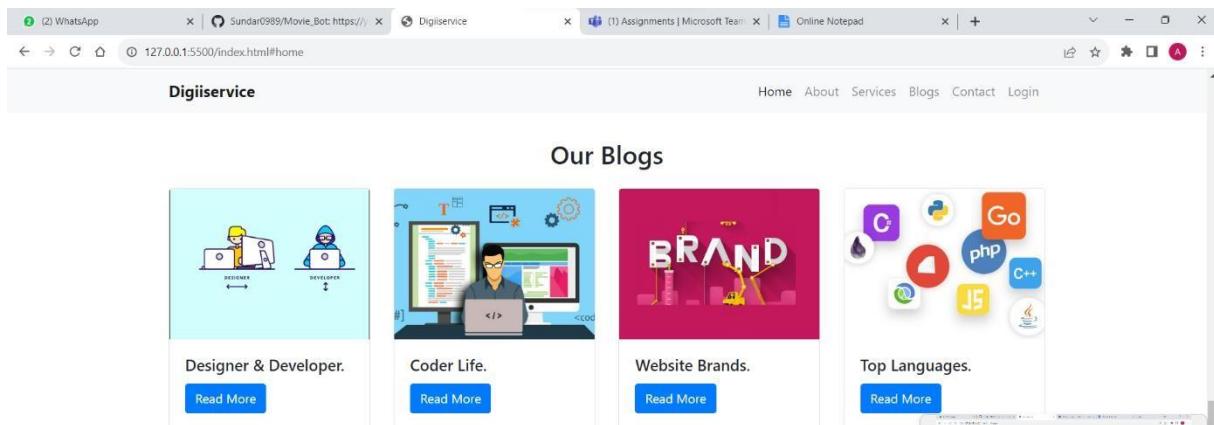
Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

The screenshot shows a web browser window with multiple tabs open. The active tab displays the 'About Us' page for 'Digiiservice'. The page features a header with the company logo and navigation links for Home, About, Services, Blogs, Contact, and Login. Below the header, there is a section titled 'Digiiservice in Town' which contains a brief description of the company's mission and a 'Read More' button. To the right of this text is a graphic illustrating a network of interconnected people icons. A callout bubble from a 'Value added Branding & Advertising' icon on a tablet screen indicates that a screenshot has been copied to the clipboard.

The screenshot shows a web browser window displaying the 'Our Services' page for 'Digiiservice'. The page includes a header with the company logo and navigation links. Below the header, there is a section titled 'Our Services' featuring three main service offerings: 'Digital Marketing', 'Brand Design', and 'Web Dev'. Each service is represented by a thumbnail image and a brief description. A callout bubble from a 'Value added Branding & Advertising' icon on a tablet screen indicates that a screenshot has been copied to the clipboard.

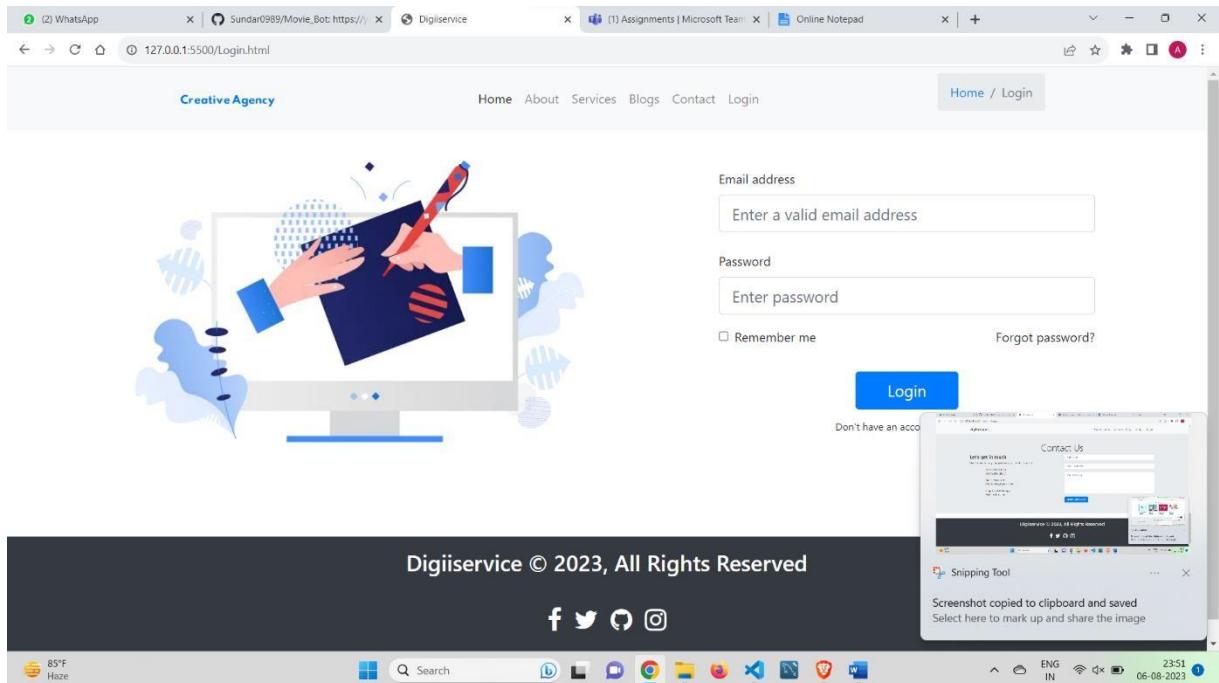
Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23



Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23



Conclusion :

Bootstrap assignment demonstrated the versatility and efficiency of the Bootstrap framework in web development. By leveraging the responsive grid system, we were able to create fluid and adaptive layouts that seamlessly adapt to different screen sizes and devices. Utilizing Bootstrap's pre-built classes for forms, buttons, and navigation components like the Navbar and Breadcrumbs, we achieved consistent and aesthetically pleasing user interfaces.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP , Date : 16/08/23

Assignment No 4 (a)

Aim : To implement conditionals, loops and functions in javascript.

Lab Outcome :

LO1: To expose students to javascript to make webpages interactive.

Theory :

Conditionals :

JavaScript offers various types of conditionals that allow developers to make decisions and control the flow of their code based on different conditions. These conditionals are crucial for creating dynamic and responsive applications. Here's an overview of the main types of conditionals in JavaScript:

1. if Statement:

The 'if' statement is the fundamental conditional construct. It evaluates a specified condition and executes a block of code if the condition is true. It can also be extended with optional 'else if' and 'else' clauses to handle multiple possible outcomes.

2. switch Statement:

The 'switch' statement allows developers to compare a single value against multiple possible case values. It provides an alternative to using multiple 'if' statements for such scenarios. Each case contains code to be executed if the value matches the case value. A 'default' case can be used to handle cases where none of the specific cases match.

3. ternary Operator (?):

The ternary operator is a concise way to write simple conditional expressions. It evaluates a condition and returns one of two values based on whether the condition is true or false. It's commonly used for assigning values or displaying content based on a condition.

4. if...else Statement:

The 'if...else' statement extends the basic 'if' construct by allowing an alternative block of code to be executed when the condition is false. This is useful when there are two distinct outcomes based on the condition's evaluation.

5. if...else if...else Statement:

This extended form of the 'if' statement is used when there are multiple possible conditions to consider. It allows developers to test multiple conditions in sequence and execute the corresponding block of code for the first true condition.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP , Date : 16/08/23

6. Nested Conditionals:

Developers can nest conditionals within other conditionals to handle complex decisionmaking scenarios. This involves placing one conditional statement inside another's code block. However, excessive nesting can lead to code that is hard to read and maintain.

These conditionals provide JavaScript developers with powerful tools to create logic that responds to varying situations. Choosing the appropriate conditional type depends on the complexity of the logic and the desired outcomes. By mastering these conditional constructs, developers can create more sophisticated and flexible applications.

Loops :

In JavaScript, loops are essential programming constructs that enable developers to execute a block of code repeatedly. They help automate repetitive tasks, iterate over collections, and control the flow of a program. JavaScript offers several types of loops, each designed to cater to different scenarios and requirements.

1. for Loop:

The 'for' loop is the most common type of loop in JavaScript. It consists of three parts: initialization, condition, and iteration. It executes a block of code as long as the condition is true, updating the iteration variable at each step. This loop is ideal for iterating over arrays and other collections.

2. while Loop:

The 'while' loop repeatedly executes a block of code as long as a given condition is true. Unlike the 'for' loop, it doesn't require explicit initialization and iteration steps. It's often used when the exact number of iterations is unknown or depends on runtime conditions.

3. do...while Loop:

Similar to the 'while' loop, the 'do...while' loop also executes a block of code based on a condition. However, it guarantees that the code block runs at least once, as the condition is evaluated after the block. This is useful when you want to execute a certain action before checking the condition.

4. for...in Loop:

The 'for...in' loop iterates over the properties of an object. It's particularly useful when you need to access the keys or property names of an object. However, it's not recommended for iterating over arrays, as it may not behave as expected due to JavaScript's object-like nature of arrays.

5. for...of Loop:

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP , Date : 16/08/23

Introduced in ECMAScript 2015 (ES6), the `for...of` loop is designed specifically for iterating over iterable objects like arrays, strings, maps, sets, etc. It provides a cleaner syntax compared to other loops and is preferred when you don't need to access the index or the properties.

Loops are indispensable tools for any programmer, enabling efficient and concise repetition of code. However, excessive or inefficient use of loops can impact performance, so it's important to choose the appropriate loop type for each situation. Developers should also consider modern practices and techniques, such as the use of higher-order functions like `forEach`, `map`, `filter`, and `reduce` for working with arrays. By mastering these loop types and leveraging modern methods, JavaScript developers can create more effective, readable, and maintainable code.

Functions:

In JavaScript, functions are the building blocks of code, enabling modularization, reusability, and the organization of logic. There are several types of functions, each serving different purposes and offering unique features that cater to various programming scenarios.

1. Regular Functions:

These are the most common type of functions in JavaScript. They are defined using the `function` keyword and can take parameters and return values. Regular functions encapsulate a block of code and can be invoked whenever needed.

2. Arrow Functions:

Introduced in ES6, arrow functions provide a more concise syntax for writing functions. They automatically inherit the `this` value from the surrounding context, making them particularly useful for callbacks and short anonymous functions.

3. Anonymous Functions:

These are functions without a defined name. They are often used as arguments to other functions or in scenarios where the function's identity isn't critical.

4. IIFE (Immediately Invoked Function Expression):

An IIFE is a function that is defined and executed immediately after its creation. It's enclosed in parentheses to ensure it's treated as an expression rather than a declaration.

5. Generator Functions:

Generator functions use the `function*` syntax. They allow for pausing and resuming the execution of a function using the `yield` keyword. This is useful for creating iterators and working with asynchronous code.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP , Date : 16/08/23

6. Callback Functions:

Callbacks are functions passed as arguments to other functions, often used in asynchronous programming. They are executed after a specific task or event completes.

7. Higher-Order Functions:

Higher-order functions are functions that take one or more functions as arguments or return a function. They enable functional programming paradigms and are common in modern JavaScript development.

8. Constructor Functions:

These functions are used to create and initialize objects. They are invoked using the `new` keyword and set up the object's properties and methods.

9. Method Functions:

Method functions are functions defined as properties of objects. They are often used to encapsulate behavior relevant to a specific object.

10. Recursive Functions:

Recursive functions call themselves within their own body. They are useful for solving problems that can be broken down into simpler instances of the same problem.

11. Async Functions:

Introduced in ES8, async functions make asynchronous code easier to write and understand. They return Promises and allow the use of the `await` keyword to pause execution until a Promise resolves.

12. Named Functions:

Named functions have a specified name and can be called by that name. They are often used for clarity, especially in debugging or stack traces.

13. Rest Parameters and Spread Operator:

Functions can use the rest parameter syntax to accept an indefinite number of arguments as an array. Conversely, the spread operator can spread an array into individual arguments when calling a function.

Understanding these different types of functions in JavaScript equips developers with the tools to create well-structured, efficient, and maintainable code. Each type has its own advantages and use cases, contributing to the flexibility and expressiveness of the language.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP , Date : 16/08/23

Source Code & Output :

index.js

The screenshot shows a browser window with two tabs open. The left tab contains the source code for `index.js`, and the right tab shows the console output.

index.js Content:

```
1 let i = 1;
2 let counter = 0;
3 while (i <= 10) {
4     counter += i;
5     i++;
6 }
7 console.log("sum of 1 to 10 is:", counter);
8
9 // -----
10 const cars = ["BMW", "Volvo", "Ford", "Audi"];
11
12 let allCars = "All cars are: ";
13 for (let i = 0; i < cars.length; i++) {
14     allCars += cars[i] + " ";
15 }
16
17 console.log(allCars);
18
19 // -----
20
21 const person = {fname:"Altaf", lname:"Alam", age:20};
22
23 let personDetail = "";
24 let personKey = "";
25
26 for (let key in person) {
27     personDetail += person[key] + " ";
28     personKey += key + ", ";
29 }
30
31 console.log("details:", personDetail);
32 console.log("keys:", personKey);
33
34 // -----
35
36 const marks = [39, 40, 78, 22, 78, 19];
37 let passcount = 0;
38 let failcount = 0;
39
40 for (let mark of marks){
41     if (mark > 35){
42         console.log("passed:", mark);
43         passcount++;
44     }
45     else {
46         console.log("failed:", mark);
47         failcount++;
48     }
49 }
50
51 console.log('pass count is: ${passcount}, fail count is: ${failcount}');
52
53 // -----
54
```

Console Output:

```
sum of 1 to 10 is: 55
All cars are: BMW Volvo Ford Audi
details: Altaf Alam 20
keys: fname, lname, age,
passed: 39
passed: 40
passed: 78
failed: 22
passed: 78
failed: 19
pass count is: 4, fail count is: 2
sum of all positive numbers in array is: 245
Hello Altaf
Hello Sarah
Hello Kulsum
9
Hint: hit control+c anytime to enter REPL.
>
```

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP , Date : 16/08/23

```
index.js
54
55 let sum = 0;
56
57 function myFunction(item) {
58     sum += (item >= 0) ? item : 0;
59 }
60
61 const numbers = [65, 44, -12, 4, 98, -5, 34];
62 numbers.forEach(myFunction);
63 console.log("sum of all positive numbers in array is:", sum);
64
65 // -----
66
67 let greet = "Hello ";
68 i = 0;
69 const users = ["Altaf", "Sarah", "Kulsum"];
70
71 const greetFunction = (a, b) => {
72     console.log(a + b);
73 }
74
75 do {
76     greetFunction(greet, users[i]);
77     i++;
78 }
79 while (i < 3);
80
81 // -----
```

```
sum of 1 to 10 is: 55
All cars are: BMW Volvo Ford Audi
details: Altaf Alam 20
keys: fname, lname, age,
passed: 39
passed: 46
passed: 78
failed: 22
passed: 78
failed: 19
pass count is: 4, fail count is: 2
sum of all positive numbers in array is: 245
Hello Altaf
Hello Sarah
Hello Kulsum
9
Hint: hit control+c anytime to enter REPL.
```



```
index.js
80
81 // -----
82
83 let sumFunc = function(a, b) {
84     console.log(a + b);
85 }
86
87 sumFunc(4, 5);
```

```
sum of 1 to 10 is: 55
All cars are: BMW Volvo Ford Audi
details: Altaf Alam 20
keys: fname, lname, age,
passed: 39
passed: 46
passed: 78
failed: 22
passed: 78
failed: 19
pass count is: 4, fail count is: 2
sum of all positive numbers in array is: 245
Hello Altaf
Hello Sarah
Hello Kulsum
9
Hint: hit control+c anytime to enter REPL.
```

Conclusion :

I successfully implemented conditionals, functions and loops and learn about the program flow using conditionals and different types of loops for handling repeated tasks easily.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP , Date : 16/08/23

Assignment No 4 (b)

Aim : To implement Javascript code to change the background color of the web page automatically after every 5 second.

Lab Outcome :

LO4: To expose students to javascript to make webpages interactive.

Theory :

1. HTML Structure:

The document adheres to HTML5 standards, incorporating the `<html>`, `<head>`, and `<body>` elements to define the webpage's structure. Meta tags inside the `<head>` section ensure proper character encoding and responsive rendering.

2. Script Inclusion:

The `<script>` element facilitates the inclusion of JavaScript code within the HTML document. This segment houses the JavaScript logic responsible for modifying the webpage's appearance.

3. getRandomColors() Function:

This JavaScript function generates random hexadecimal color codes. By assembling a color code using a loop that selects random characters from a string of hexadecimal values, it creates dynamic color variations.

4. changebg() Function:

The `changebg()` function invokes the `getRandomColors()` function to acquire a random color code. Subsequently, it applies this color as the new background color for the webpage's `<body>` element.

5. setInterval() Method:

The `setInterval()` method establishes a recurring execution pattern for the `changebg()` function. In this instance, the method triggers the function's execution every 5000 milliseconds (5 seconds), leading to a periodic alteration of the background color.

6. Webpage Content:

Within the `<body>` section, a single `<h1>` element with the text "Hello Pikachu" constitutes the webpage's visible content. This content remains unaffected by the dynamic background color changes.

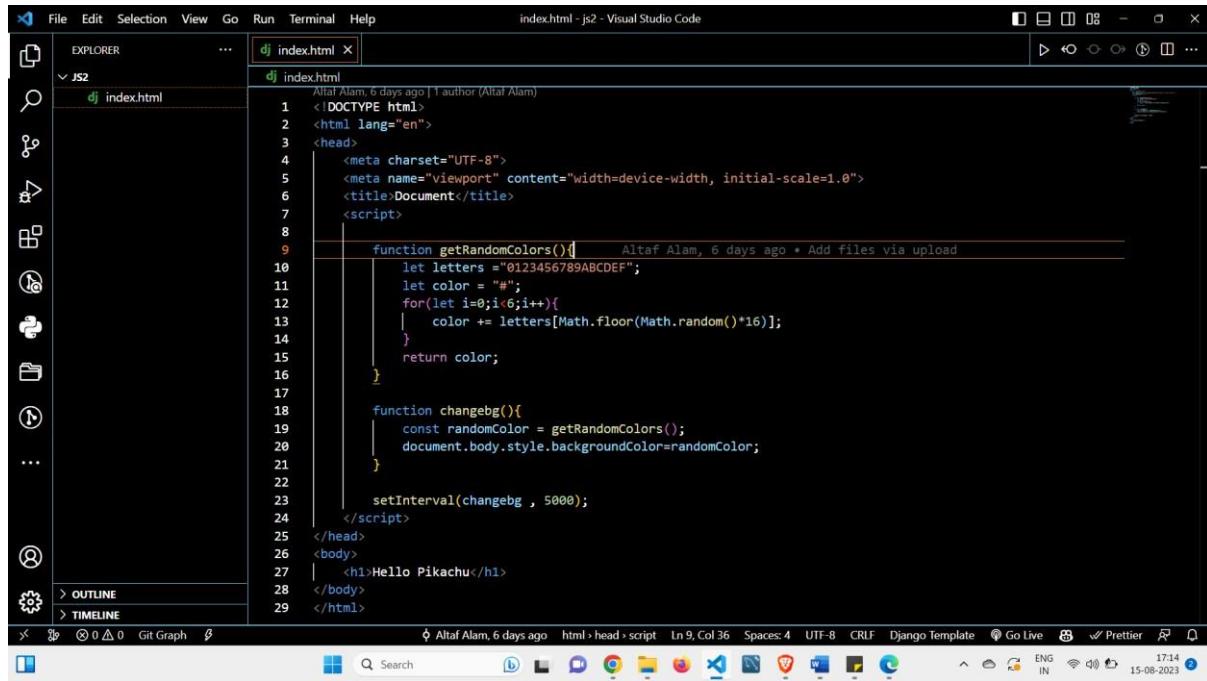
Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP , Date : 16/08/23

This HTML document showcases the integration of JavaScript to enhance user experience through dynamic visuals. By leveraging random color generation and timed execution, the background color of the webpage transitions periodically, introducing an engaging and interactive element to the webpage's design.

Source Code & Output :

index.html



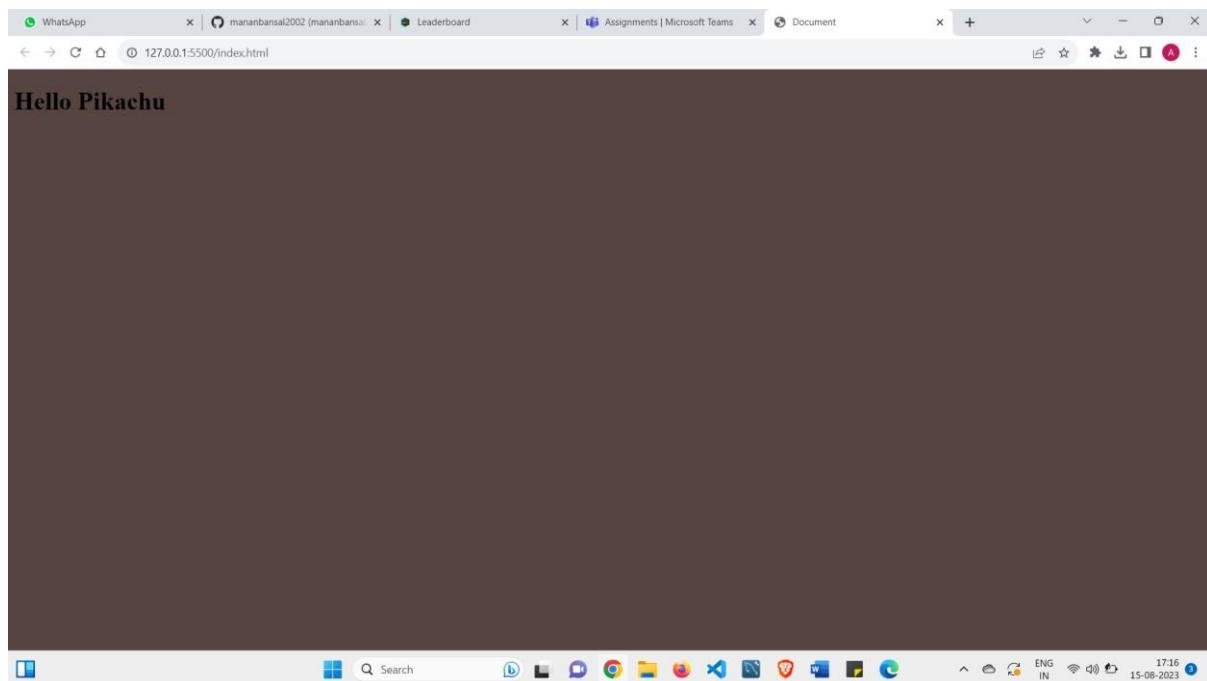
The screenshot shows the Visual Studio Code interface with the file 'index.html' open. The code editor displays the following HTML and JavaScript:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script>

        function getRandomColors(){
            let letters = "0123456789ABCDEF";
            let color = "#";
            for(let i=0;i<6;i++){
                color += letters[Math.floor(Math.random()*16)];
            }
            return color;
        }

        function changebg(){
            const randomColor = getRandomColors();
            document.body.style.backgroundColor=randomColor;
        }

        setInterval(changebg , 5000);
    </script>
</head>
<body>
    <h1>Hello Pikachu</h1>
</body>
</html>
```



Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP , Date : 16/08/23

Conclusion :

I successfully used conditionals, functions and loops and implemented this.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

Assignment No 4b

Aim : Write a Program on inheritance, Iterators and Generators.

Lab Outcome :

LO4: To expose students to javascript to make web pages interactive.

Theory :

1. Inheritance:

Inheritance is a fundamental concept in object-oriented programming (OOP) that allows you to create new classes (subclasses) based on existing classes (superclasses). It promotes code reuse and the creation of hierarchies of related objects. In JavaScript, inheritance is achieved through prototype-based inheritance.

- Superclass (Parent Class): The superclass is the original class that defines the properties and methods that can be inherited.
- Subclass (Child Class): The subclass is a new class that inherits properties and methods from a superclass. It can also add its own properties and methods or override the inherited ones.
- `extends` Keyword: In modern JavaScript, you can use the `extends` keyword to create a subclass that inherits from a superclass.

1. Single Inheritance:

- In Single inheritance, classes are organized in a hierarchical structure.
- Each class can inherit properties and methods from a single parent class.

2. Multiple Inheritance:

- In multiple inheritance, a class can inherit properties and methods from multiple parent classes.
- While powerful, it can lead to complex inheritance hierarchies and the "diamond problem," which makes it challenging to resolve conflicts when two parent classes define the same method or property.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

3. Multilevel Inheritance:

- In multilevel inheritance, classes are organized in a linear hierarchy, with each class inheriting from its immediate parent.
- It doesn't involve multiple inheritance, and it's easier to manage than multiple inheritance.

4. Hierarchical Inheritance:

- In hierarchical inheritance, multiple child classes inherit from a single parent class.
- Each child class can add its own unique properties and methods.

Note: JavaScript does not support multiple inheritance, or explicit class-based inheritance as found in some other programming languages like Java or C++. Instead, JavaScript relies on prototype-based inheritance, where objects inherit directly from other objects. This is a unique feature of the language and can be both powerful and flexible when used effectively.

2. Iterators:

Iterators are objects in JavaScript that provide a way to iterate (loop) over a collection of items, such as an array, set, or custom data structure. They ensure that you can access each element of the collection sequentially.

- **Iterable:** An iterable is an object that has an associated iterator. Arrays, strings, maps, sets, and custom iterable objects are examples of iterables.
- **Iterator:** An iterator is an object that provides a `next()` method. When called, the `next()` method returns the next item in the collection along with a `done` flag to indicate if the iteration is complete.
- **`Symbol.iterator`:** In JavaScript, objects that are iterable must implement the `Symbol.iterator` method, which returns an iterator.

3. Generators:

Generators are a type of function in JavaScript that allow you to control the flow of execution explicitly. They can be paused and resumed, enabling you to write more readable and maintainable asynchronous code.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

- `function*` Syntax: You define a generator function using the `function*` syntax. Inside a generator function, you can use the `yield` keyword to pause the function's execution and yield a value.
- `yield`: The `yield` keyword is used within a generator function to produce a value and temporarily pause the function. When the generator is resumed, it continues execution from where it left off.
- Lazy Evaluation: Generators enable lazy evaluation, meaning that they produce values on demand rather than all at once. This is useful for efficiently handling large datasets or asynchronous operations.
- Iterating with Generators: Generators are often used to create custom iterators, providing a more concise and readable way to define iterators for custom data structures.

Source Code :

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

The screenshot shows two instances of Visual Studio Code side-by-side, both displaying the same JavaScript code related to inheritance.

The code defines four classes:

- Vehicle**: A base class with a constructor taking a name and a present() method returning a string indicating the vehicle's name.
- Car**: An extended class that inherits from Vehicle. It has its own constructor taking name and company, and a show() method that returns a string combining the vehicle's present() output and the company name.
- Model**: An extended class that inherits from Car. It has its own constructor taking name, company, and speed, and a show2() method that returns a string combining the car's show() output and the speed.
- Bike**: An extended class that inherits from Vehicle. It has its own constructor taking name and company, and a show() method that returns a string combining the vehicle's present() output and the company name.

Both instances of VS Code have identical file structures in the Explorer sidebar, showing files named generators.js, inheritances.js, and iterators.js. The status bar at the bottom of each window shows the date as 06-09-2023 and the time as 12:18. The system tray indicates a temperature of 31°C and a weather condition of Haze.

```
// single inheritance
class Vehicle {
  constructor(name) {
    | this.Vehiclename = name;
  }
  present() {
    | return `My vehicle's name is ${this.Vehiclename}`;
  }
}

// multi level inheritance
class Car extends Vehicle {
  constructor(name, company) {
    super(name);
    | this.company = company;
  }
  show() {
    | return this.present() + ". It is a car of company " + this.company;
  }
}

let myCar = new Car("Audi T2", "Audi");
console.log(myCar.show());

// hierarchical inheritance
class Model extends Car {
  constructor(name, company, speed) {
    super(name, company);
    | this.speed = speed;
  }
  show2() {
    | return this.show() + ", it has a speed " + this.speed;
  }
}

let myModel = new Model("BMW X5", "BMW", "520");
console.log(myModel.show2());

// hierarchical inheritance
class Bike extends Vehicle {
  constructor(name, company) {
    super(name);
    | this.company = company;
  }
  show() {
    | return this.present() + ". It is a bike of company " + this.company;
  }
}

let myBike = new Bike("Splendor Xtech", "Splendor");
console.log(myBike.show());
```

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

The screenshot shows two instances of Visual Studio Code side-by-side. Both instances have the same interface with a dark theme. The top instance is titled 'iterators.js - 5th - Visual Studio Code' and contains the following code:

```
//Iterators: Altaf Alam, 2 weeks ago * Add files via upload ...
const names = [ 'altaf', 'sarah', 'aman', 'prasad' ];
const obj = names[Symbol.iterator]();

while(true){
  let curr = obj.next();
  if(curr.done==false){
    | console.log(curr);
  }
  else{
    | console.log(curr, "Finished");
    break;
  }
}
// console.log(obj.next());
// console.log(obj.next());
// console.log(obj.next());
// console.log(obj.next());
// console.log(obj.next());
```

The bottom instance is titled 'generators.js - 5th - Visual Studio Code' and contains the following code:

```
You, 12 minutes ago | 2 authors (Altaf Alam and others)
function* generator() {
  console.log("g1")
  yield 'altaf';
  console.log("g2")
  return 'result';
  yield 'aman';
}

You, 12 minutes ago * Uncommitted changes
let it = generator();
console.log("paused1")
console.log(it.next());

console.log("paused2")
console.log(it.next());
```

Output :

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

The screenshot shows two instances of Visual Studio Code side-by-side. Both instances have the same file structure and terminal output.

File Structure:

- Both instances show a folder named "5th" containing three files: "generators.js", "inheritances.js", and "iterators.js".
- The "inheritances.js" file is open in the editor.
- The "iterators.js" file is open in the terminal tab.

Terminal Output (from inheritance.js):

```
PS D:\SEMS-LAB\IP\5th> node "d:\SEMS-LAB\IP\5th\inheritances.js"
My vehicle's name is Audi T2. It is a car of company Audi
My vehicle's name is BMW X5. It is a car of company BMW, it has a speed 520
My vehicle's name is Splendor Xttech. It is a bike of company Splendor
PS D:\SEMS-LAB\IP\5th>
```

Terminal Output (from iterators.js):

```
Altaf Alam, 2 weeks ago | author (Altaf Alam)
1 //Iterators: Altaf Alam, 2 weeks ago * Add files via upload
2 const names = ['altaf', 'sarah', 'aman', 'prasad'];

PS D:\SEMS-LAB\IP\5th> node "d:\SEMS-LAB\IP\5th\inheritances.js"
My vehicle's name is Audi T2. It is a car of company Audi
My vehicle's name is BMW X5. It is a car of company BMW, it has a speed 520
My vehicle's name is Splendor Xttech. It is a bike of company Splendor
PS D:\SEMS-LAB\IP\5th> node "d:\SEMS-LAB\IP\5th\iterators.js"
{
  value: 'altaf', done: false
}
{
  value: 'sarah', done: false
}
{
  value: 'aman', done: false
}
{
  value: 'prasad', done: false
}
{
  value: undefined, done: true } Finished
PS D:\SEMS-LAB\IP\5th>
```

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

The screenshot shows a Visual Studio Code interface. The terminal window displays the execution of three JavaScript files: generators.js, inheritances.js, and iterators.js. The output of each file is shown below:

```
PS D:\SEMS-LAB\IP\5th> node "d:\SEMS-LAB\IP\5th\inherities.js"
My vehicle's name is Audi T2. It is a car of company Audi.
My vehicle's name is BMW X5. It is a car of company BMW, it has a speed 520
My vehicle's name is Splendor Xtech. It is a bike of company Splendor

PS D:\SEMS-LAB\IP\5th> node "d:\SEMS-LAB\IP\5th\iterators.js"
{ value: 'altaf', done: false }
{ value: 'sarah', done: false }
{ value: 'aman', done: false }
{ value: 'prasad', done: false }
{ value: undefined, done: true } Finished

PS D:\SEMS-LAB\IP\5th> node "d:\SEMS-LAB\IP\5th\generators.js"
paused1
g1
{ value: 'altaf', done: false }
paused2
g2
{ value: 'result', done: true }

PS D:\SEMS-LAB\IP\5th>
```

Conclusion :

In conclusion, JavaScript primarily supports prototype-based inheritance, allowing objects to inherit directly from other objects. While it doesn't support classical inheritance or multiple inheritance, this unique inheritance model provides flexibility and simplicity in managing object relationships.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

Assignment No 5a

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

Lab Outcome :

LO4: To expose students to javascript to make web pages interactive.

Theory :

1. Arrow Functions:

Arrow functions are a feature introduced in ECMAScript 6 (ES6) JavaScript, and they provide a concise and more readable way to write functions. Here's a deeper look at arrow functions:

- Syntax:

Arrow functions have a shorter syntax compared to traditional function declarations. The basic syntax is as follows:

```
```javascript
(parameters) => expression
```

```

- No `this` Binding:

Arrow functions do not have their own `this` context. Instead, they inherit the `this` value from their enclosing (parent) function or scope. This behavior can be beneficial in certain situations, especially when working with callbacks and event handlers.

```
// Arrow function with a single argument
const square = x => x * x;

// Arrow function in a higher-order function
const numbers = [1, 2, 3, 4, 5];
const squaredNumbers = numbers.map(x => x * x);
```

```

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

## 2. DOM Manipulation (Document Object Model):

The Document Object Model (DOM) is a programming interface for HTML and XML documents. It represents the structure of a web page as a tree-like structure of objects that can be manipulated using JavaScript. Here's a deeper look at DOM manipulation:

### - Accessing Elements:

JavaScript can be used to access and manipulate HTML elements in the DOM. Common methods for accessing elements include `getElementById`, `querySelector`, and `querySelectorAll`. For example:

```
```javascript
// Get an element by its ID
const element = document.getElementById('myElement');

// Select elements using CSS-like selectors
const elements = document.querySelectorAll('.myClass');
````
```

### - Modifying Content:

JavaScript can change the content of HTML elements, such as text, attributes, and HTML structure. You can use properties like `innerHTML`, `textContent`, and `setAttribute` to modify elements.

### - Adding and Removing Elements:

JavaScript allows you to dynamically add or remove elements from the DOM. Methods like `appendChild`, `removeChild`, and `createElement` are commonly used for this purpose.

```
```javascript
// Create a new element
const newElement = document.createElement('div');

// Append the new element to an existing one
parentElement.appendChild(newElement);
````
```

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

```
// Remove an element
parentElement.removeChild(childElement);
...
...
```

#### - Event Handling:

You can attach event listeners to elements to respond to user interactions like clicks, key presses, and form submissions. Event listeners are essential for creating interactive web applications.

```
```javascript  
element.addEventListener('click', function() {  
    // Handle the click event  
});  
...  
...
```

3. CSS Manipulation:

CSS manipulation involves changing the styles and appearance of HTML elements using JavaScript. Here's a deeper look at CSS manipulation:

- Changing Styles:

JavaScript can change the CSS styles of elements by accessing their `style` property. You can modify properties like `backgroundColor`, `fontSize`, and `display`.

```
...  
// Modify font size  
element.style.fontSize = '16px';  
...  
...
```

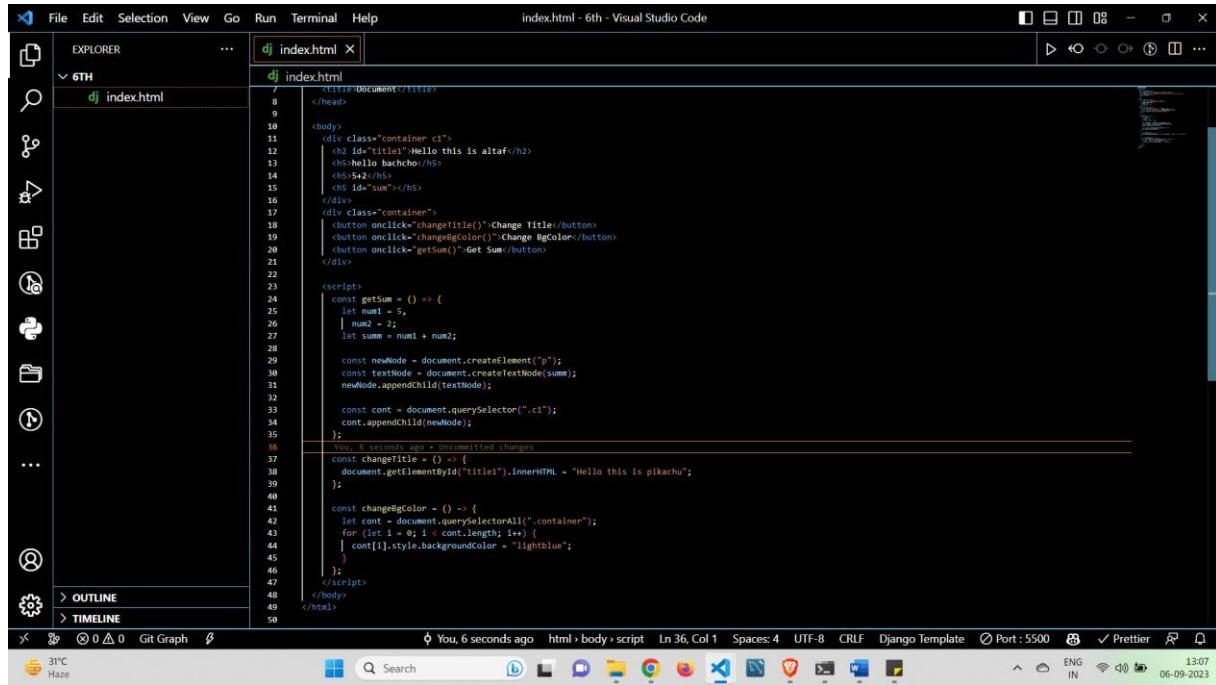
- Adding and Removing Classes:

```
element.classList.add('active');  
  
// Remove a CSS class  
element.classList.remove('inactive');
```

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

Code :



The screenshot shows the Visual Studio Code interface with the file 'index.html' open. The code is a simple web page with a title, some text, and three buttons for interacting with the DOM.

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<div class="container c1">
<h2 id="title1">Hello this is altaf</h2>
<h3>Hello batch</h3>
<h5 id="sum"></h5>
</div>
<div class="container">
<button onclick="changeTitle()">Change Title</button>
<button onclick="changeBgColor()">Change BgColor</button>
<button onclick="getSum()">Get Sum</button>
</div>
<script>
const getSum = () => {
let num1 = 5,
| num2 = 2;
let sum = num1 + num2;

const newNode = document.createElement("p");
const textNode = document.createTextNode(sum);
newNode.appendChild(textNode);

const cont = document.querySelectorAll(".c1");
cont.appendChild(newNode);
};

const changeTitle = () => {
document.getElementById("title1").innerHTML = "Hello this is pikachu";
};

const changeBgColor = () => {
let cont = document.querySelectorAll(".container");
for (let i = 0; i < cont.length; i++) {
| cont[i].style.backgroundColor = "lightblue";
}
}
</script>
</body>
</html>
```

You, 6 seconds ago + Uncommitted changes

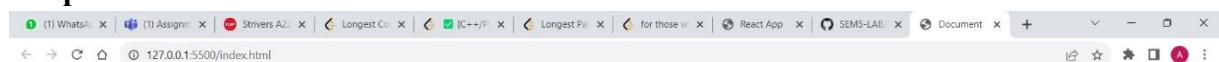
File Explorer sidebar icons: File, Edit, Selection, View, Go, Run, Terminal, Help.

Bottom status bar: You, 6 seconds ago, html > body > script, Ln 36, Col 1, Spaces: 4, UTF-8, CRLF, Django Template, Port: 5500, Prettier, ENG IN, 13:07, 06-09-2023.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

Output :



Hello this is altaf

hello bachcho

5+2



Hello this is pikachu

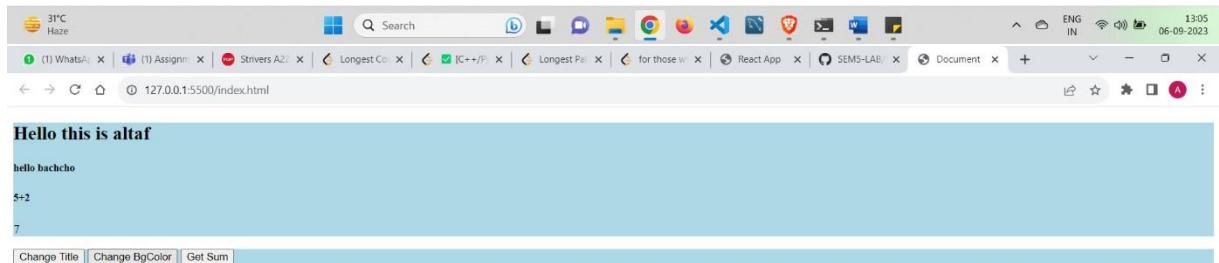
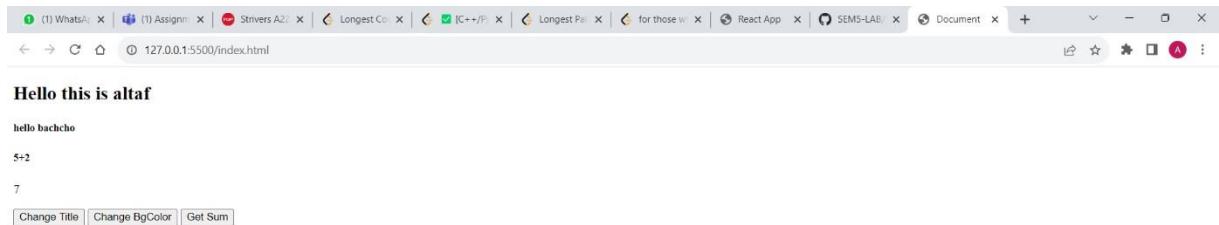
hello bachcho

5+2



Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23



Conclusion :

In conclusion, We learn about the arrow functions and implemented different ways to change add or delete the HTML , CSS using javascript dynamically on situation.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

Assignment No 5b

Aim : Write a Javascript program to implement the concept of Promise.

Access the data from any API using Promise and either resolve or reject the request by taking appropriate action.

Lab Outcome :

LO4: To expose students to javascript to make web pages interactive.

Theory :

Promises in JavaScript are a powerful asynchronous programming concept used for managing and coordinating asynchronous operations. They provide a way to handle asynchronous tasks more elegantly and avoid callback hell (also known as "Pyramid of Doom"). The concept of promises can be summarized as follows:

1. Promise States:

- Promises have three states: pending, fulfilled, and rejected.
- Initially, a promise is in the pending state, meaning the asynchronous operation it represents has not completed yet.
- When the operation succeeds, the promise transitions to the fulfilled state, and if it fails, it transitions to the rejected state.

2. Asynchronous Operations:

- Promises are typically used for asynchronous operations, such as fetching data from a server, reading a file, or making network requests.
- The result of an asynchronous operation is eventually resolved or rejected, and promises help you handle these outcomes.

3. Chaining Promises:

- Promises can be chained together, allowing you to sequence asynchronous operations in a more readable and maintainable manner.
- You can use the ` `.then()` method to attach success and error handlers to a promise, which will be invoked when the promise is fulfilled or rejected, respectively.

4. Error Handling:

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

- Promises provide a central mechanism for error handling through the `catch()` method. Errors that occur during promise execution can be caught and handled in a single place.
 - This helps improve code maintainability by separating error handling from the logic of asynchronous operations.

5. Promise.all and Promise.race:

- `Promise.all()` takes an array of promises and returns a new promise that fulfills when all the input promises have fulfilled or rejects when any one of them rejects.
 - `Promise.race()` returns a promise that fulfills or rejects as soon as one of the input promises fulfills or rejects. It's useful for tasks where you're interested in the first result.

Promises are a foundational concept in modern JavaScript for handling asynchronous code flows and have become even more prominent with the introduction of `async/await` in ES2017, which simplifies working with promises even further.

Output :

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

Conclusion :

In conclusion, Promises in JavaScript provide a structured and efficient way to handle asynchronous operations like api callings, improving code readability and error handling.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

Assignment No 6a

Aim : WAP to implement the concept of props and state. Create a functional component and pass current date as props and with class component , on button click display both date and time with change font and color.

Lab Outcome :

LO5: To orient students to React for developing front end applications.

Theory :

In React, 'props' and 'state' are fundamental concepts that allow you to manage and pass data within a component-based application. Additionally, 'useState' and 'useRef', etc. are React hooks that help you work with state and references in functional components.

Props:

Props (short for properties) are a mechanism for passing data from parent components to child components. They are read-only and are used to share data and behavior between components in a unidirectional flow. Key points about props:

1. Immutable: Props are immutable, meaning they cannot be modified within the child component. They are passed down from parent components and are read-only.
2. Functional Components: Props are commonly used in functional components to receive data and render it. You define props as parameters in the component's function signature.
3. Data Flow: Props facilitate the flow of data from a parent component to a child component. This allows you to create reusable and composable components.
4. Usage: To access props in a functional component, you simply refer to them as properties of the 'props' object (e.g., 'props.name').
5. Example:

```
```jsx
function ParentComponent() {
 const greeting =
 "Hello, React!";
 return <ChildComponent
 message={greeting} />;
}
```

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

```
function ChildComponent(props) {
return <p>{props.message}</p>; }
...
```
```

State:

State is used to manage and store data that can change over time within a component. Unlike props, which are read-only, state is mutable and allows components to re-render when the state changes. Key points about state:

1. Class Components: In class components, you manage state using the `this.state` object and the `this.setState()` method. State can be modified using `setState()` to trigger re-renders.
2. Functional Components with Hooks: With the introduction of hooks in React, functional components can also manage state using the `useState` hook. It returns an array with the current state value and a function to update it.
3. Local to a Component: State is local to a component and cannot be directly accessed or modified by other components. It encapsulates a component's internal data.
4. Reactive UI: When state changes, React re-renders the component, ensuring that the UI reflects the latest data.
5. Example (Functional Component with `useState`):

```
```jsx  
import React, { useState } from 'react';

function Counter() {
 const [count, setCount] = useState(0);

 const increment = () => {
 setCount(count + 1);
 };

 return (
 <div>
 <p>Count: {count}</p>
 <button onClick={increment}>Increment</button>
 </div>
);
}
```

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

...

### useRef:

'useRef' is a React hook that provides a way to create mutable object references that persist across renders. It is commonly used for accessing and interacting with DOM elements, managing focus, and persisting values between renders. Key points about 'useRef':

1. Mutable Refs: 'useRef' creates a mutable object with a '.current' property. This property can be assigned a value, and it retains its value between renders without causing re-renders.
2. Accessing DOM Elements: One common use of 'useRef' is to access and manipulate DOM elements directly. You can attach a 'ref' attribute to a JSX element and assign it to a 'useRef' variable.
3. Preserving Values: 'useRef' is suitable for preserving values or state between renders without causing re-renders. This makes it different from 'useState'.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

## Code :

The screenshot shows two instances of Visual Studio Code side-by-side. Both instances have the title bar "File Edit Selection View Go Run Terminal Help" and the status bar at the bottom showing system information like battery level, network, and date.

**Top Window (App.js - react1 - Visual Studio Code):**

- Explorer:** Shows the project structure with files like node\_modules, public, src, components, App.css, App.js, App.test.js, index.css, index.jsx, logo.svg, reportWebVitals.js, setupTests.js, package-lock.json, package.json, and README.md.
- Editor:** The code for App.js is displayed:

```
1 import './App.css';
2 import ConditionalRendering from './components/ConditionalRendering';
3 import Counter from './components/Counter';
4 import MyDate from './components/MyDate';

5
6 function App() {
7 const date = new Date();

8 let day = date.getDate();
9 let month = date.getMonth() + 1;
10 let year = date.getFullYear();

11 let currentDate = `${day}-${month}-${year}`;
12
13 return (
14 <div className="App">
15 <h2>Date</h2>
16 <MyDate currDate={currentDate} /> <hr/>
17 <h2>Counter</h2>
18 <Counter /> <hr/>
19 <h2>Conditional Rendering Examples</h2>
20 <ConditionalRendering />
21 </div>
22);
23
24}
25
26 export default App;
27
```
- Status Bar:** Shows "You, 16 minutes ago" and "Uncommitted changes".

**Bottom Window (MyDate.jsx - react1 - Visual Studio Code):**

- Explorer:** Shows the project structure with files like node\_modules, public, src, components, MyDate.jsx, App.css, App.js, App.test.js, index.css, index.jsx, logo.svg, reportWebVitals.js, setupTests.js, package-lock.json, package.json, and README.md.
- Editor:** The code for MyDate.jsx is displayed:

```
1 const MyDate = (props) => {
2 return (
3 <div>
4 <h5>{props.CurrDate}</h5>
5 </div>
6);
7 }
8 export default MyDate;
```
- Status Bar:** Shows "You, 18 minutes ago" and "Uncommitted changes".

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

The image shows two instances of Visual Studio Code side-by-side. Both instances have the title bar "Counterjsx - react1 - Visual Studio Code".

**File 1: Counter.jsx**

```
You, 1 second ago | 2 authors (Altaf Alam and others)
1 import { useState } from "react"; Altaf Alam, 2 weeks ago * Add files via upload
2
3 const Counter = () => {
4 const [counter, setCounter] = useState(0);
5
6 const handleCounter = () => setCounter(counter + 1);
7 const handleTitle = () => {
8 document.title = "Pikachu izz here"
9 }
10
11 return (
12 <div>
13 <h5>You have clicked {counter} times.</h5>
14 <button onClick={handleCounter}>Increment Counter</button>
15 <button onClick={handleTitle}>Change Title</button>
16 </div>
17);
18
19 }
20 export default Counter;
```

**File 2: ConditionalRendering.jsx**

```
You, 15 minutes ago | 2 authors (You and others)
1 import React, { useState } from "react";
2
3 const ConditionalRendering = () => {
4 const [name, setName] = useState("Altaf"); 'setName' is assigned a value but never used.
5 const [role, setRole] = useState("admin"); 'setRole' is assigned a value but never used.
6 const [roles, setRoles] = useState(["admin", "user", "guest"]); 'setRoles' is assigned a value but never used.
7
8 return (
9 <div>
10 <p>(name === "Altaf" && <h3>`${name} is logged in`</h3>)</p>
11 <{name === "Altaf" && <h3>`${name} is logged in`</h3>}>
12
13 <h3>{name === "Altaf" ? `${name} is logged in` : "You are a scammer"}</h3>
14
15 <{name === "Altaf" && <h3>`${name} is logged in`</h3>}>
16 <{name !== "Altaf" && <h3>You are a scammer</h3>}>
17
18 <{role === "admin" ? (
19 <h3>You are an admin</h3>
20) : role === "user" ? (
21 <h3>You are a user</h3>
22) : (
23 <h3>You, 15 minutes ago * Uncommitted changes</h3>
24)
25
26 <{roles.map((role) => (
27 <h3 key={role}>You have role: {role}</h3>
28))}>
29
30 </div>
31);
32
33 export default ConditionalRendering;
34 }
```

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

## Output :

**Date**  
6-9-2023

---

**Counter**  
You have clicked 15 times.  
[Increment Counter](#) [Change Title](#)

---

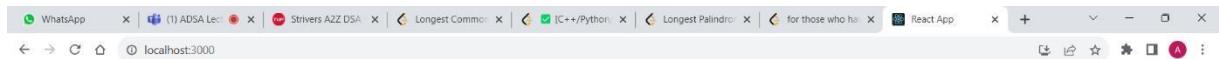
**Conditional Rendering Examples**

- Altaf is logged in
- You are an admin
- You have role: admin
- You have role: user
- You have role: guest



Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23



**Date**  
6-9-2023

---

**Counter**

You have clicked 0 times.

[Increment Counter](#) [Change Title](#)

---

**Conditional Rendering Examples**

Altaf is logged in  
Altaf is logged in  
Altaf is logged in  
Altaf is logged in  
You are an admin  
You have role: admin  
You have role: user  
You have role: guest



**Date**  
6-9-2023

---

**Counter**

You have clicked 15 times.

[Increment Counter](#) [Change Title](#)

---

**Conditional Rendering Examples**

Altaf is logged in  
Altaf is logged in  
Altaf is logged in  
Altaf is logged in  
You are an admin  
You have role: admin  
You have role: user  
You have role: guest



## Conclusion :

In conclusion, `props` and `state` are essential concepts in React that enable you to manage and share data within components. React hooks provide a more declarative and functional approach to managing component state and interacting with the DOM.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

## **Assignment No 6b**

**Aim :** Write a Program to implement forms and events.

**Lab Outcome :**

**LO5:** To orient students to React for developing front end applications.

**Theory :**

**Forms:**

1. What is a Form?

A form is an essential HTML element used to collect and submit data from users on a web page. Forms typically contain various types of input fields, such as text fields, checkboxes, radio buttons, dropdown lists, and buttons. Users can enter information into these fields, and when they submit the form, the data is sent to a server for processing.

2. Common Form Elements:

- Input Fields: These allow users to enter text, numbers, or other data. Common input types include text, password, email, and more.

- Textarea: Used for multi-line text input, suitable for longer messages or comments.

- Checkboxes and Radio Buttons: Used for binary choices (checkboxes) or selecting one option from a group (radio buttons).

- Select Lists: Provide a dropdown menu of options for users to choose from.

- Buttons: Buttons within a form can trigger actions like submission or reset.

3. Form Attributes:

- `action`: Specifies the URL where the form data will be sent upon submission.

- `method`: Defines the HTTP method to be used for the request (e.g., GET or POST).

- `name`: Provides a name for the form, often used for identification in JavaScript.

- `enctype`: Specifies how form data should be encoded for submission (e.g., `application/x-www-form-urlencoded` for simple text data).

- `target`: Specifies where the response should be displayed (e.g., in the same window or a new tab).

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

#### 4. Form Submission:

When a user clicks the submit button within a form, the browser collects the data from the form fields and sends it to the URL specified in the `action` attribute using the HTTP method specified in the `method` attribute (usually POST or GET). The server processes the data and sends back a response.

#### Events:

##### 1. What are Events?

Events are interactions or occurrences that happen in the browser, such as user actions (e.g., clicks, keypresses, mouse movements) or changes in the document (e.g., the page finishes loading). JavaScript allows you to listen for and respond to these events, enabling interactivity and dynamic behavior in web applications.

##### 2. Common Types of Events:

- Mouse Events: Events triggered by mouse actions, such as `click`, `mousedown`, `mouseup`, `mousemove`, and `mouseenter`.
- Keyboard Events: Events related to keyboard input, such as `keydown`, `keyup`, and `keypress`.
- Form Events: Events related to form interactions, like `submit`, `reset`, `change`, and `input`.
- Document and Window Events: Events related to the document and window, including `load`, `unload`, `resize`, and `scroll`.
- Custom Events: Developers can create custom events to handle specific interactions within their applications.

##### 3. Event Handling:

Event handling in JavaScript involves the following steps:

- Event Listener: You attach an event listener to an HTML element, specifying the type of event you want to listen for (e.g., `click`) and the function (event handler) to be executed when the event occurs.

Name : Altaf Alam

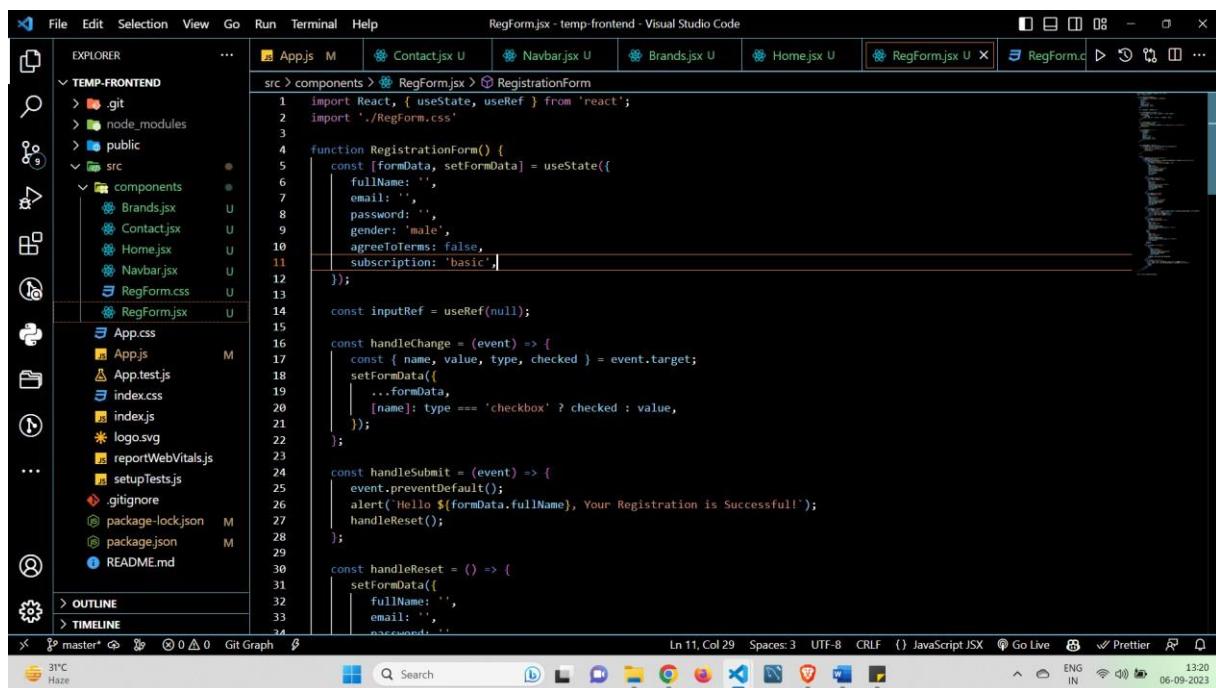
Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

- Event Handler: The event handler is a JavaScript function that defines what should happen when the event occurs. It can perform actions like updating the DOM, making AJAX requests, or triggering animations.
- Event Object: When an event occurs, an event object is automatically created and passed to the event handler. This object contains information about the event, such as the target element and event-specific data.

#### 4. Event Propagation:

Events can propagate through the DOM tree in two phases: capturing phase and bubbling phase. Event propagation determines the order in which event listeners are invoked. You can control event propagation using the `stopPropagation` method.

#### Code :



```
src > components > RegForm.jsx > RegistrationForm
1 import React, { useState, useRef } from 'react';
2 import './RegForm.css'
3
4 function RegistrationForm() {
5 const [formData, setFormData] = useState({
6 fullName: '',
7 email: '',
8 password: '',
9 gender: 'male',
10 agreeToTerms: false,
11 subscription: 'basic'
12 });
13
14 const inputRef = useRef(null);
15
16 const handleChange = (event) => {
17 const { name, value, type, checked } = event.target;
18 setFormData({
19 ...formData,
20 [name]: type === 'checkbox' ? checked : value,
21 });
22 };
23
24 const handleSubmit = (event) => {
25 event.preventDefault();
26 alert(`Hello ${formData.fullName}, Your Registration is Successful!`);
27 handleReset();
28 };
29
30 const handleReset = () => {
31 setFormData({
32 fullName: '',
33 email: '',
34 password: ''
35 });
36 };
37
38 const handleSubscribe = (event) => {
39 event.preventDefault();
40 alert(`Subscribed to ${subscription}`);
41 };
42
43 const handleLogout = (event) => {
44 event.preventDefault();
45 alert('Logout successful');
46 };
47
48 const handleReset = () => {
49 setFormData({
50 fullName: '',
51 email: '',
52 password: ''
53 });
54 };
55
56 const handleSubscribe = (event) => {
57 event.preventDefault();
58 alert(`Subscribed to ${subscription}`);
59 };
60
61 const handleLogout = (event) => {
62 event.preventDefault();
63 alert('Logout successful');
64 };
65
66 const handleReset = () => {
67 setFormData({
68 fullName: '',
69 email: '',
70 password: ''
71 });
72 };
73
74 const handleSubscribe = (event) => {
75 event.preventDefault();
76 alert(`Subscribed to ${subscription}`);
77 };
78
79 const handleLogout = (event) => {
80 event.preventDefault();
81 alert('Logout successful');
82 };
83
84 const handleReset = () => {
85 setFormData({
86 fullName: '',
87 email: '',
88 password: ''
89 });
90 };
91
92 const handleSubscribe = (event) => {
93 event.preventDefault();
94 alert(`Subscribed to ${subscription}`);
95 };
96
97 const handleLogout = (event) => {
98 event.preventDefault();
99 alert('Logout successful');
100 };
101
102 const handleReset = () => {
103 setFormData({
104 fullName: '',
105 email: '',
106 password: ''
107 });
108 };
109
110 const handleSubscribe = (event) => {
111 event.preventDefault();
112 alert(`Subscribed to ${subscription}`);
113 };
114
115 const handleLogout = (event) => {
116 event.preventDefault();
117 alert('Logout successful');
118 };
119
120 const handleReset = () => {
121 setFormData({
122 fullName: '',
123 email: '',
124 password: ''
125 });
126 };
127
128 const handleSubscribe = (event) => {
129 event.preventDefault();
130 alert(`Subscribed to ${subscription}`);
131 };
132
133 const handleLogout = (event) => {
134 event.preventDefault();
135 alert('Logout successful');
136 };
137
138 const handleReset = () => {
139 setFormData({
140 fullName: '',
141 email: '',
142 password: ''
143 });
144 };
145
146 const handleSubscribe = (event) => {
147 event.preventDefault();
148 alert(`Subscribed to ${subscription}`);
149 };
150
151 const handleLogout = (event) => {
152 event.preventDefault();
153 alert('Logout successful');
154 };
155
156 const handleReset = () => {
157 setFormData({
158 fullName: '',
159 email: '',
160 password: ''
161 });
162 };
163
164 const handleSubscribe = (event) => {
165 event.preventDefault();
166 alert(`Subscribed to ${subscription}`);
167 };
168
169 const handleLogout = (event) => {
170 event.preventDefault();
171 alert('Logout successful');
172 };
173
174 const handleReset = () => {
175 setFormData({
176 fullName: '',
177 email: '',
178 password: ''
179 });
180 };
181
182 const handleSubscribe = (event) => {
183 event.preventDefault();
184 alert(`Subscribed to ${subscription}`);
185 };
186
187 const handleLogout = (event) => {
188 event.preventDefault();
189 alert('Logout successful');
190 };
191
192 const handleReset = () => {
193 setFormData({
194 fullName: '',
195 email: '',
196 password: ''
197 });
198 };
199
200 const handleSubscribe = (event) => {
201 event.preventDefault();
202 alert(`Subscribed to ${subscription}`);
203 };
204
205 const handleLogout = (event) => {
206 event.preventDefault();
207 alert('Logout successful');
208 };
209
210 const handleReset = () => {
211 setFormData({
212 fullName: '',
213 email: '',
214 password: ''
215 });
216 };
217
218 const handleSubscribe = (event) => {
219 event.preventDefault();
220 alert(`Subscribed to ${subscription}`);
221 };
222
223 const handleLogout = (event) => {
224 event.preventDefault();
225 alert('Logout successful');
226 };
227
228 const handleReset = () => {
229 setFormData({
230 fullName: '',
231 email: '',
232 password: ''
233 });
234 };
235
236 const handleSubscribe = (event) => {
237 event.preventDefault();
238 alert(`Subscribed to ${subscription}`);
239 };
240
241 const handleLogout = (event) => {
242 event.preventDefault();
243 alert('Logout successful');
244 };
245
246 const handleReset = () => {
247 setFormData({
248 fullName: '',
249 email: '',
250 password: ''
251 });
252 };
253
254 const handleSubscribe = (event) => {
255 event.preventDefault();
256 alert(`Subscribed to ${subscription}`);
257 };
258
259 const handleLogout = (event) => {
260 event.preventDefault();
261 alert('Logout successful');
262 };
263
264 const handleReset = () => {
265 setFormData({
266 fullName: '',
267 email: '',
268 password: ''
269 });
270 };
271
272 const handleSubscribe = (event) => {
273 event.preventDefault();
274 alert(`Subscribed to ${subscription}`);
275 };
276
277 const handleLogout = (event) => {
278 event.preventDefault();
279 alert('Logout successful');
280 };
281
282 const handleReset = () => {
283 setFormData({
284 fullName: '',
285 email: '',
286 password: ''
287 });
288 };
289
290 const handleSubscribe = (event) => {
291 event.preventDefault();
292 alert(`Subscribed to ${subscription}`);
293 };
294
295 const handleLogout = (event) => {
296 event.preventDefault();
297 alert('Logout successful');
298 };
299
300 const handleReset = () => {
301 setFormData({
302 fullName: '',
303 email: '',
304 password: ''
305 });
306 };
307
308 const handleSubscribe = (event) => {
309 event.preventDefault();
310 alert(`Subscribed to ${subscription}`);
311 };
312
313 const handleLogout = (event) => {
314 event.preventDefault();
315 alert('Logout successful');
316 };
317
318 const handleReset = () => {
319 setFormData({
320 fullName: '',
321 email: '',
322 password: ''
323 });
324 };
325
326 const handleSubscribe = (event) => {
327 event.preventDefault();
328 alert(`Subscribed to ${subscription}`);
329 };
330
331 const handleLogout = (event) => {
332 event.preventDefault();
333 alert('Logout successful');
334 };
335
336 const handleReset = () => {
337 setFormData({
338 fullName: '',
339 email: '',
340 password: ''
341 });
342 };
343
344 const handleSubscribe = (event) => {
345 event.preventDefault();
346 alert(`Subscribed to ${subscription}`);
347 };
348
349 const handleLogout = (event) => {
350 event.preventDefault();
351 alert('Logout successful');
352 };
353
354 const handleReset = () => {
355 setFormData({
356 fullName: '',
357 email: '',
358 password: ''
359 });
360 };
361
362 const handleSubscribe = (event) => {
363 event.preventDefault();
364 alert(`Subscribed to ${subscription}`);
365 };
366
367 const handleLogout = (event) => {
368 event.preventDefault();
369 alert('Logout successful');
370 };
371
372 const handleReset = () => {
373 setFormData({
374 fullName: '',
375 email: '',
376 password: ''
377 });
378 };
379
380 const handleSubscribe = (event) => {
381 event.preventDefault();
382 alert(`Subscribed to ${subscription}`);
383 };
384
385 const handleLogout = (event) => {
386 event.preventDefault();
387 alert('Logout successful');
388 };
389
390 const handleReset = () => {
391 setFormData({
392 fullName: '',
393 email: '',
394 password: ''
395 });
396 };
397
398 const handleSubscribe = (event) => {
399 event.preventDefault();
400 alert(`Subscribed to ${subscription}`);
401 };
402
403 const handleLogout = (event) => {
404 event.preventDefault();
405 alert('Logout successful');
406 };
407
408 const handleReset = () => {
409 setFormData({
410 fullName: '',
411 email: '',
412 password: ''
413 });
414 };
415
416 const handleSubscribe = (event) => {
417 event.preventDefault();
418 alert(`Subscribed to ${subscription}`);
419 };
420
421 const handleLogout = (event) => {
422 event.preventDefault();
423 alert('Logout successful');
424 };
425
426 const handleReset = () => {
427 setFormData({
428 fullName: '',
429 email: '',
430 password: ''
431 });
432 };
433
434 const handleSubscribe = (event) => {
435 event.preventDefault();
436 alert(`Subscribed to ${subscription}`);
437 };
438
439 const handleLogout = (event) => {
440 event.preventDefault();
441 alert('Logout successful');
442 };
443
444 const handleReset = () => {
445 setFormData({
446 fullName: '',
447 email: '',
448 password: ''
449 });
450 };
451
452 const handleSubscribe = (event) => {
453 event.preventDefault();
454 alert(`Subscribed to ${subscription}`);
455 };
456
457 const handleLogout = (event) => {
458 event.preventDefault();
459 alert('Logout successful');
460 };
461
462 const handleReset = () => {
463 setFormData({
464 fullName: '',
465 email: '',
466 password: ''
467 });
468 };
469
470 const handleSubscribe = (event) => {
471 event.preventDefault();
472 alert(`Subscribed to ${subscription}`);
473 };
474
475 const handleLogout = (event) => {
476 event.preventDefault();
477 alert('Logout successful');
478 };
479
480 const handleReset = () => {
481 setFormData({
482 fullName: '',
483 email: '',
484 password: ''
485 });
486 };
487
488 const handleSubscribe = (event) => {
489 event.preventDefault();
490 alert(`Subscribed to ${subscription}`);
491 };
492
493 const handleLogout = (event) => {
494 event.preventDefault();
495 alert('Logout successful');
496 };
497
498 const handleReset = () => {
499 setFormData({
500 fullName: '',
501 email: '',
502 password: ''
503 });
504 };
505
506 const handleSubscribe = (event) => {
507 event.preventDefault();
508 alert(`Subscribed to ${subscription}`);
509 };
510
511 const handleLogout = (event) => {
512 event.preventDefault();
513 alert('Logout successful');
514 };
515
516 const handleReset = () => {
517 setFormData({
518 fullName: '',
519 email: '',
520 password: ''
521 });
522 };
523
524 const handleSubscribe = (event) => {
525 event.preventDefault();
526 alert(`Subscribed to ${subscription}`);
527 };
528
529 const handleLogout = (event) => {
530 event.preventDefault();
531 alert('Logout successful');
532 };
533
534 const handleReset = () => {
535 setFormData({
536 fullName: '',
537 email: '',
538 password: ''
539 });
540 };
541
542 const handleSubscribe = (event) => {
543 event.preventDefault();
544 alert(`Subscribed to ${subscription}`);
545 };
546
547 const handleLogout = (event) => {
548 event.preventDefault();
549 alert('Logout successful');
550 };
551
552 const handleReset = () => {
553 setFormData({
554 fullName: '',
555 email: '',
556 password: ''
557 });
558 };
559
560 const handleSubscribe = (event) => {
561 event.preventDefault();
562 alert(`Subscribed to ${subscription}`);
563 };
564
565 const handleLogout = (event) => {
566 event.preventDefault();
567 alert('Logout successful');
568 };
569
570 const handleReset = () => {
571 setFormData({
572 fullName: '',
573 email: '',
574 password: ''
575 });
576 };
577
578 const handleSubscribe = (event) => {
579 event.preventDefault();
580 alert(`Subscribed to ${subscription}`);
581 };
582
583 const handleLogout = (event) => {
584 event.preventDefault();
585 alert('Logout successful');
586 };
587
588 const handleReset = () => {
589 setFormData({
590 fullName: '',
591 email: '',
592 password: ''
593 });
594 };
595
596 const handleSubscribe = (event) => {
597 event.preventDefault();
598 alert(`Subscribed to ${subscription}`);
599 };
599
600 const handleLogout = (event) => {
601 event.preventDefault();
602 alert('Logout successful');
603 };
604
605 const handleReset = () => {
606 setFormData({
607 fullName: '',
608 email: '',
609 password: ''
610 });
611 };
612
613 const handleSubscribe = (event) => {
614 event.preventDefault();
615 alert(`Subscribed to ${subscription}`);
616 };
617
618 const handleLogout = (event) => {
619 event.preventDefault();
620 alert('Logout successful');
621 };
622
623 const handleReset = () => {
624 setFormData({
625 fullName: '',
626 email: '',
627 password: ''
628 });
629 };
630
631 const handleSubscribe = (event) => {
632 event.preventDefault();
633 alert(`Subscribed to ${subscription}`);
634 };
635
636 const handleLogout = (event) => {
637 event.preventDefault();
638 alert('Logout successful');
639 };
640
641 const handleReset = () => {
642 setFormData({
643 fullName: '',
644 email: '',
645 password: ''
646 });
647 };
648
649 const handleSubscribe = (event) => {
650 event.preventDefault();
651 alert(`Subscribed to ${subscription}`);
652 };
653
654 const handleLogout = (event) => {
655 event.preventDefault();
656 alert('Logout successful');
657 };
658
659 const handleReset = () => {
660 setFormData({
661 fullName: '',
662 email: '',
663 password: ''
664 });
665 };
666
667 const handleSubscribe = (event) => {
668 event.preventDefault();
669 alert(`Subscribed to ${subscription}`);
670 };
671
672 const handleLogout = (event) => {
673 event.preventDefault();
674 alert('Logout successful');
675 };
676
677 const handleReset = () => {
678 setFormData({
679 fullName: '',
680 email: '',
681 password: ''
682 });
683 };
684
685 const handleSubscribe = (event) => {
686 event.preventDefault();
687 alert(`Subscribed to ${subscription}`);
688 };
689
690 const handleLogout = (event) => {
691 event.preventDefault();
692 alert('Logout successful');
693 };
694
695 const handleReset = () => {
696 setFormData({
697 fullName: '',
698 email: '',
699 password: ''
700 });
701 };
702
703 const handleSubscribe = (event) => {
704 event.preventDefault();
705 alert(`Subscribed to ${subscription}`);
706 };
707
708 const handleLogout = (event) => {
709 event.preventDefault();
710 alert('Logout successful');
711 };
712
713 const handleReset = () => {
714 setFormData({
715 fullName: '',
716 email: '',
717 password: ''
718 });
719 };
720
721 const handleSubscribe = (event) => {
722 event.preventDefault();
723 alert(`Subscribed to ${subscription}`);
724 };
725
726 const handleLogout = (event) => {
727 event.preventDefault();
728 alert('Logout successful');
729 };
730
731 const handleReset = () => {
732 setFormData({
733 fullName: '',
734 email: '',
735 password: ''
736 });
737 };
738
739 const handleSubscribe = (event) => {
740 event.preventDefault();
741 alert(`Subscribed to ${subscription}`);
742 };
743
744 const handleLogout = (event) => {
745 event.preventDefault();
746 alert('Logout successful');
747 };
748
749 const handleReset = () => {
750 setFormData({
751 fullName: '',
752 email: '',
753 password: ''
754 });
755 };
756
757 const handleSubscribe = (event) => {
758 event.preventDefault();
759 alert(`Subscribed to ${subscription}`);
760 };
761
762 const handleLogout = (event) => {
763 event.preventDefault();
764 alert('Logout successful');
765 };
766
767 const handleReset = () => {
768 setFormData({
769 fullName: '',
770 email: '',
771 password: ''
772 });
773 };
774
775 const handleSubscribe = (event) => {
776 event.preventDefault();
777 alert(`Subscribed to ${subscription}`);
778 };
779
780 const handleLogout = (event) => {
781 event.preventDefault();
782 alert('Logout successful');
783 };
784
785 const handleReset = () => {
786 setFormData({
787 fullName: '',
788 email: '',
789 password: ''
790 });
791 };
792
793 const handleSubscribe = (event) => {
794 event.preventDefault();
795 alert(`Subscribed to ${subscription}`);
796 };
797
798 const handleLogout = (event) => {
799 event.preventDefault();
800 alert('Logout successful');
801 };
802
803 const handleReset = () => {
804 setFormData({
805 fullName: '',
806 email: '',
807 password: ''
808 });
809 };
810
811 const handleSubscribe = (event) => {
812 event.preventDefault();
813 alert(`Subscribed to ${subscription}`);
814 };
815
816 const handleLogout = (event) => {
817 event.preventDefault();
818 alert('Logout successful');
819 };
820
821 const handleReset = () => {
822 setFormData({
823 fullName: '',
824 email: '',
825 password: ''
826 });
827 };
828
829 const handleSubscribe = (event) => {
830 event.preventDefault();
831 alert(`Subscribed to ${subscription}`);
832 };
833
834 const handleLogout = (event) => {
835 event.preventDefault();
836 alert('Logout successful');
837 };
838
839 const handleReset = () => {
840 setFormData({
841 fullName: '',
842 email: '',
843 password: ''
844 });
845 };
846
847 const handleSubscribe = (event) => {
848 event.preventDefault();
849 alert(`Subscribed to ${subscription}`);
850 };
851
852 const handleLogout = (event) => {
853 event.preventDefault();
854 alert('Logout successful');
855 };
856
857 const handleReset = () => {
858 setFormData({
859 fullName: '',
860 email: '',
861 password: ''
862 });
863 };
864
865 const handleSubscribe = (event) => {
866 event.preventDefault();
867 alert(`Subscribed to ${subscription}`);
868 };
869
870 const handleLogout = (event) => {
871 event.preventDefault();
872 alert('Logout successful');
873 };
874
875 const handleReset = () => {
876 setFormData({
877 fullName: '',
878 email: '',
879 password: ''
880 });
881 };
882
883 const handleSubscribe = (event) => {
884 event.preventDefault();
885 alert(`Subscribed to ${subscription}`);
886 };
887
888 const handleLogout = (event) => {
889 event.preventDefault();
890 alert('Logout successful');
891 };
892
893 const handleReset = () => {
894 setFormData({
895 fullName: '',
896 email: '',
897 password: ''
898 });
899 };
900
901 const handleSubscribe = (event) => {
902 event.preventDefault();
903 alert(`Subscribed to ${subscription}`);
904 };
905
906 const handleLogout = (event) => {
907 event.preventDefault();
908 alert('Logout successful');
909 };
910
911 const handleReset = () => {
912 setFormData({
913 fullName: '',
914 email: '',
915 password: ''
916 });
917 };
918
919 const handleSubscribe = (event) => {
920 event.preventDefault();
921 alert(`Subscribed to ${subscription}`);
922 };
923
924 const handleLogout = (event) => {
925 event.preventDefault();
926 alert('Logout successful');
927 };
928
929 const handleReset = () => {
930 setFormData({
931 fullName: '',
932 email: '',
933 password: ''
934 });
935 };
936
937 const handleSubscribe = (event) => {
938 event.preventDefault();
939 alert(`Subscribed to ${subscription}`);
940 };
941
942 const handleLogout = (event) => {
943 event.preventDefault();
944 alert('Logout successful');
945 };
946
947 const handleReset = () => {
948 setFormData({
949 fullName: '',
950 email: '',
951 password: ''
952 });
953 };
954
955 const handleSubscribe = (event) => {
956 event.preventDefault();
957 alert(`Subscribed to ${subscription}`);
958 };
959
960 const handleLogout = (event) => {
961 event.preventDefault();
962 alert('Logout successful');
963 };
964
965 const handleReset = () => {
966 setFormData({
967 fullName: '',
968 email: '',
969 password: ''
970 });
971 };
972
973 const handleSubscribe = (event) => {
974 event.preventDefault();
975 alert(`Subscribed to ${subscription}`);
976 };
977
978 const handleLogout = (event) => {
979 event.preventDefault();
980 alert('Logout successful');
981 };
982
983 const handleReset = () => {
984 setFormData({
985 fullName: '',
986 email: '',
987 password: ''
988 });
989 };
990
991 const handleSubscribe = (event) => {
992 event.preventDefault();
993 alert(`Subscribed to ${subscription}`);
994 };
995
996 const handleLogout = (event) => {
997 event.preventDefault();
998 alert('Logout successful');
999 };
999
1000 const handleReset = () => {
1001 setFormData({
1002 fullName: '',
1003 email: '',
1004 password: ''
1005 });
1006 };
1007
1008 const handleSubscribe = (event) => {
1009 event.preventDefault();
1010 alert(`Subscribed to ${subscription}`);
1011 };
1012
1013 const handleLogout = (event) => {
1014 event.preventDefault();
1015 alert('Logout successful');
1016 };
1017
1018 const handleReset = () => {
1019 setFormData({
1020 fullName: '',
1021 email: '',
1022 password: ''
1023 });
1024 };
1025
1026 const handleSubscribe = (event) => {
1027 event.preventDefault();
1028 alert(`Subscribed to ${subscription}`);
1029 };
1030
1031 const handleLogout = (event) => {
1032 event.preventDefault();
1033 alert('Logout successful');
1034 };
1035
1036 const handleReset = () => {
1037 setFormData({
1038 fullName: '',
1039 email: '',
1040 password: ''
1041 });
1042 };
1043
1044 const handleSubscribe = (event) => {
1045 event.preventDefault();
1046 alert(`Subscribed to ${subscription}`);
1047 };
1048
1049 const handleLogout = (event) => {
1050 event.preventDefault();
1051 alert('Logout successful');
1052 };
1053
1054 const handleReset = () => {
1055 setFormData({
1056 fullName: '',
1057 email: '',
1058 password: ''
1059 });
1060 };
1061
1062 const handleSubscribe = (event) => {
1063 event.preventDefault();
1064 alert(`Subscribed to ${subscription}`);
1065 };
1066
1067 const handleLogout = (event) => {
1068 event.preventDefault();
1069 alert('Logout successful');
1070 };
1071
1072 const handleReset = () => {
1073 setFormData({
1074 fullName: '',
1075 email: '',
1076 password: ''
1077 });
1078 };
1079
1080 const handleSubscribe = (event) => {
1081 event.preventDefault();
1082 alert(`Subscribed to ${subscription}`);
1083 };
1084
1085 const handleLogout = (event) => {
1086 event.preventDefault();
1087 alert('Logout successful');
1088 };
1089
1090 const handleReset = () => {
1091 setFormData({
1092 fullName: '',
1093 email: '',
1094 password: ''
1095 });
1096 };
1097
1098 const handleSubscribe = (event) => {
1099 event.preventDefault();
1100 alert(`Subscribed to ${subscription}`);
1101 };
1102
1103 const handleLogout = (event) => {
1104 event.preventDefault();
1105 alert('Logout successful');
1106 };
1107
1108 const handleReset = () => {
1109 setFormData({
1110 fullName: '',
1111 email: '',
1112 password: ''
1113 });
1114 };
1115
1116 const handleSubscribe = (event) => {
1117 event.preventDefault();
1118 alert(`Subscribed to ${subscription}`);
1119 };
1120
1121 const handleLogout = (event) => {
1122 event.preventDefault();
1123 alert('Logout successful');
1124 };
1125
1126 const handleReset = () => {
1127 setFormData({
1128 fullName: '',
1129 email: '',
1130 password: ''
1131 });
1132 };
1133
1134 const handleSubscribe = (event) => {
1135 event.preventDefault();
1136 alert(`Subscribed to ${subscription}`);
1137 };
1138
1139 const handleLogout = (event) => {
1140 event.preventDefault();
1141 alert('Logout successful');
1142 };
1143
1144 const handleReset = () => {
1145 setFormData({
1146 fullName: '',
1147 email: '',
1148 password: ''
1149 });
1150 };
1151
1152 const handleSubscribe = (event) => {
1153 event.preventDefault();
1154 alert(`Subscribed to ${subscription}`);
1155 };
1156
1157 const handleLogout = (event) => {
1158 event.preventDefault();
1159 alert('Logout successful');
1160 };
1161
1162 const handleReset = () => {
1163 setFormData({
1164 fullName: '',
1165 email: '',
1166 password: ''
1167 });
1168 };
1169
1170 const handleSubscribe = (event) => {
1171 event.preventDefault();
1172 alert(`Subscribed to ${subscription}`);
1173 };
1174
1175 const handleLogout = (event) => {
1176 event.preventDefault();
1177 alert('Logout successful');
1178 };
1179
1180 const handleReset = () => {
1181 setFormData({
1182 fullName: '',
1183 email: '',
1184 password: ''
1185 });
1186 };
1187
1188 const handleSubscribe = (event) => {
1189 event.preventDefault();
1190 alert(`Subscribed to ${subscription}`);
1191 };
1192
1193 const handleLogout = (event) => {
1194 event.preventDefault();
1195 alert('Logout successful');
1196 };
1197
1198 const handleReset = () => {
1199 setFormData({
1200 fullName: '',
1201 email: '',
1202 password: ''
1203 });
1204 };
1205
1206 const handleSubscribe = (event) => {
1207 event.preventDefault();
1208 alert(`Subscribed to ${subscription}`);
1209 };
1210
1211 const handleLogout = (event) => {
1212 event.preventDefault();
1213 alert('Logout successful');
1214 };
1215
1216 const handleReset = () => {
1217 setFormData({
1218 fullName: '',
1219 email: '',
1220 password: ''
1221 });
1222 };
1223
1224 const handleSubscribe = (event) => {
1225 event.preventDefault();
1226 alert(`Subscribed to ${subscription}`);
1227 };
1228
1229 const handleLogout = (event) => {
1230 event.preventDefault();
1231 alert('Logout successful');
1232 };
1233
1234 const handleReset = () => {
1235 setFormData({
1236 fullName: '',
1237 email: '',
1238 password: ''
1239 });
1240 };
1241
1242 const handleSubscribe = (event) => {
1243 event.preventDefault();
1244 alert(`Subscribed to ${subscription}`);
1245 };
1246
1247 const handleLogout = (event) => {
1248 event.preventDefault();
1249 alert('Logout successful');
1250 };
1251
1252 const handleReset = () => {
1253 setFormData({
1254 fullName: '',
1255 email: '',
1256 password: ''
1257 });
1258 };
1259
1260 const handleSubscribe = (event) => {
1261 event.preventDefault();
1262 alert(`Subscribed to ${subscription}`);
1263 };
1264
1265 const handleLogout = (event) => {
1266 event.preventDefault();
1267 alert('Logout successful');
1268 };
1269
1270 const handleReset = () => {
1271 setFormData({
1272 fullName: '',
1273 email: '',
1274 password: ''
1275 });
1276 };
1277
1278 const handleSubscribe = (event) => {
1279 event.preventDefault();
1280 alert(`Subscribed to ${subscription}`);
1281 };
1282
1283 const handleLogout = (event) => {
1284 event.preventDefault();
1285 alert('Logout successful');
1286 };
1287
1288 const handleReset = () => {
1289 setFormData({
1290 fullName: '',
1291 email: '',
1292 password: ''
1293 });
1294 };
1295
1296 const handleSubscribe = (event) => {
1297 event.preventDefault();
1298 alert(`Subscribed to ${subscription}`);
1299 };
1300
1301 const handleLogout = (event) => {
1302 event.preventDefault();
1303 alert('Logout successful');
1304 };
1305
1306 const handleReset = () => {
1307 setFormData({
1308 fullName: '',
1309 email: '',
1310 password: ''
1311 });
1312 };
1313
1314 const handleSubscribe = (event) => {
1315 event.preventDefault();
1316 alert(`Subscribed to ${subscription}`);
1317 };
1318
1319 const handleLogout = (event) => {
1320 event.preventDefault();
1321 alert('Logout successful');
1322 };
1323
1324 const handleReset = () => {
1325 setFormData({
1326 fullName: '',
1327 email: '',
1328 password: ''
1329 });
1330 };
1331
1332 const handleSubscribe = (event) => {
1333 event.preventDefault();
1334 alert(`Subscribed to ${subscription}`);
1335 };
1336
1337 const handleLogout = (event) => {
1338 event.preventDefault();
1339 alert('Logout successful');
1340 };
1341
1342 const handleReset = () => {
1343 setFormData({
1344 fullName: '',
1345 email: '',
1346 password: ''
1347 });
1348 };
1349
1350 const handleSubscribe = (event) => {
1351 event.preventDefault();
1352 alert(`Subscribed to ${subscription}`);
1353 };
1354
1355 const handleLogout = (event) => {

```

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows the project structure:
  - TEMP-FRONTEND**:
    - .git
    - node\_modules
    - public
    - src
      - components (selected):
        - Brands.jsx
        - Contact.jsx
        - Home.jsx
        - Navbar.jsx
        - RegForm.jsx (selected)
      - RegForm.css
  - App.js
  - App.test.js
  - index.css
  - index.jsx
  - logo.svg
  - reportWebVitals.js
  - setupTests.js
  - .gitignore
  - package-lock.json
  - package.json
  - README.md
- Code Editor (Right):** Displays the code for `RegForm.jsx`. The code defines a form group with three input fields: email, password, and gender.
- Bottom Status Bar:** Shows file status (master), Git Graph, and system icons for battery, temperature (31°C), and date (06-09-2023).
- Bottom Taskbar:** Includes a search bar, browser icons (Edge, Chrome, Firefox, Opera), and system icons for battery, signal, and date.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

The screenshot displays two instances of the Visual Studio Code editor, both showing the same file: `RegForm.jsx`. The file is part of a project structure under `src > components > RegForm.jsx`.

**Top Instance (Initial State):**

```
src > components > RegForm.jsx > RegistrationForm
98 | </select>
99 | </div>
100 | <div className="form-group">
101 | <label>Subscription Plan:</label>
102 | <input
103 | type="radio"
104 | name="subscription"
105 | value="basic"
106 | onChange={handleChange}
107 | checked={formData.subscription === 'basic'}
108 | ref={inputRef}
109 | />{' '}
110 | Basic
111 | <input
112 | type="radio"
113 | name="subscription"
114 | value="premium"
115 | onChange={handleChange}
116 | checked={formData.subscription === 'premium'}
117 | ref={inputRef}
118 | />{' '}
119 | Premium
120 | </div>
121 | <div className='form-checkbox'>
122 | <label>
123 | <input
124 | type="checkbox"
125 | name="agreeToTerms"
126 | checked={formData.agreeToTerms}
127 | onChange={handleChange}
128 | ref={inputRef}
129 | />{' '}
130 | I agree to the terms and conditions
131 | </label>
```

**Bottom Instance (Modified State):**

```
src > components > RegForm.jsx > RegistrationForm
122 | <label>
123 | <input
124 | type="checkbox"
125 | name="agreeToTerms"
126 | checked={formData.agreeToTerms}
127 | onChange={handleChange}
128 | ref={inputRef}
129 | />{' '}
130 | I agree to the terms and conditions
131 | </label>
132 | </div>
133 | <div className="form-buttons">
134 | <button type="submit" className="submit-button">Submit</button>
135 | <button type="reset" onClick={handleReset} className="reset-button">
136 | Reset
137 | </button>
138 | </div>
139 | </form>
140 | </div>
141 |
142 |
143 |
144 export default RegistrationForm;
145 }
```

The main difference between the two instances is the removal of the comment `I agree to the terms and conditions` at line 130 in the bottom instance.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

### Output :

Registration Form

**Full Name:**  
Enter full name

**Email Address:**  
Enter email address

**Password:**  
Enter password

**Gender:**  
Male

**Subscription Plan:**  
Basic Premium

I agree to the terms and conditions

**Submit** **Reset**

localhost:3000 says  
Form closed with Esc key.

**Email Address:**  
alt@gmail.com

**Password:**  
.....

**Gender:**  
Male

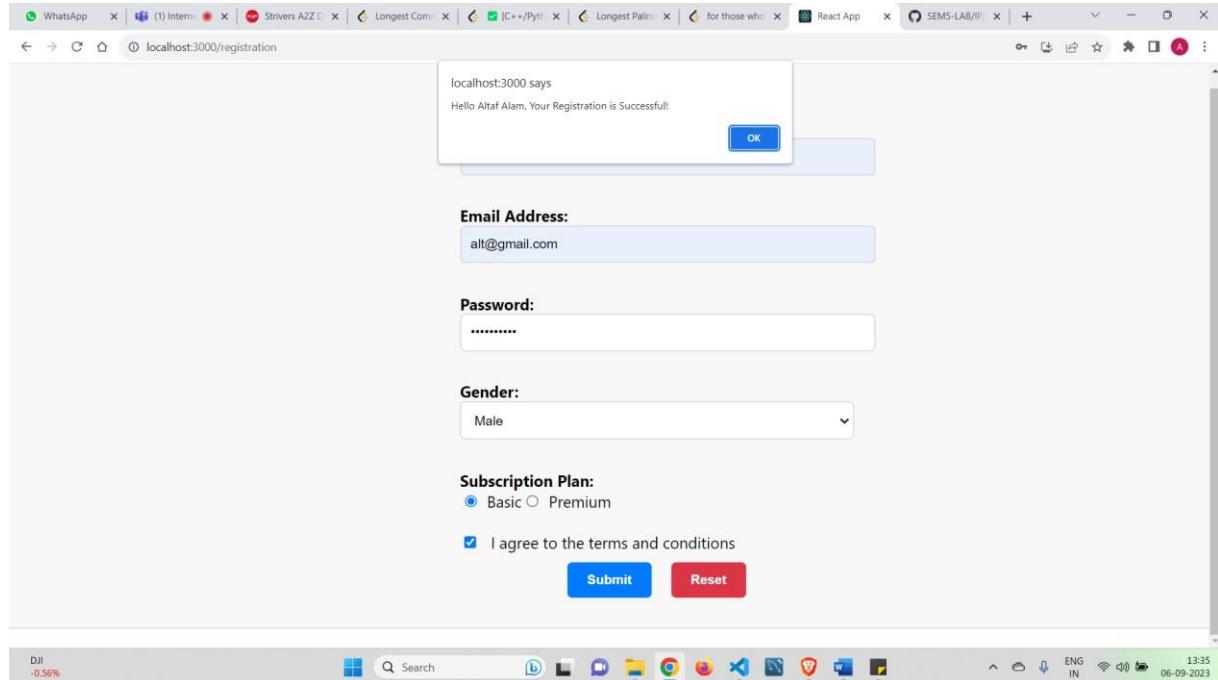
**Subscription Plan:**  
Basic Premium

I agree to the terms and conditions

**Submit** **Reset**

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23



## Conclusion :

In conclusion, forms and events are core components of interactive web development. Forms enable data collection and submission, while events allow you to respond to user actions and create dynamic, interactive web applications.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

## **Assignment No 7a**

**Aim :** WAP to implement ReactJS Router. Create more than two class or functional components and implement program for Routing using browserrouter.

### **Lab Outcome :**

**LO5:** To orient students to React for developing front end applications.

### **Theory :**

React Router is a powerful library for handling routing in React applications. It allows you to create single-page applications (SPAs) where different components are displayed based on the URL, enabling navigation without refreshing the entire page. React Router uses the 'BrowserRouter' component as one of its routing mechanisms. Let's explore how to use 'BrowserRouter' in a React application:

#### 1. Installation:

First, make sure you have React and React Router installed. You can install React Router using npm or yarn:

...

```
npm install react-router-dom
```

...

#### 2. Importing BrowserRouter:

In your React application, import 'BrowserRouter' from 'react-router-dom':

```
```javascript
import { BrowserRouter } from
'react-router-dom';
````
```

#### 3. Wrapping the App Component:

Wrap your entire application or the root component (usually 'App.js') with the 'BrowserRouter' component. This should be done in your main entry file :

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

Here's how `BrowserRouter` works in this context:

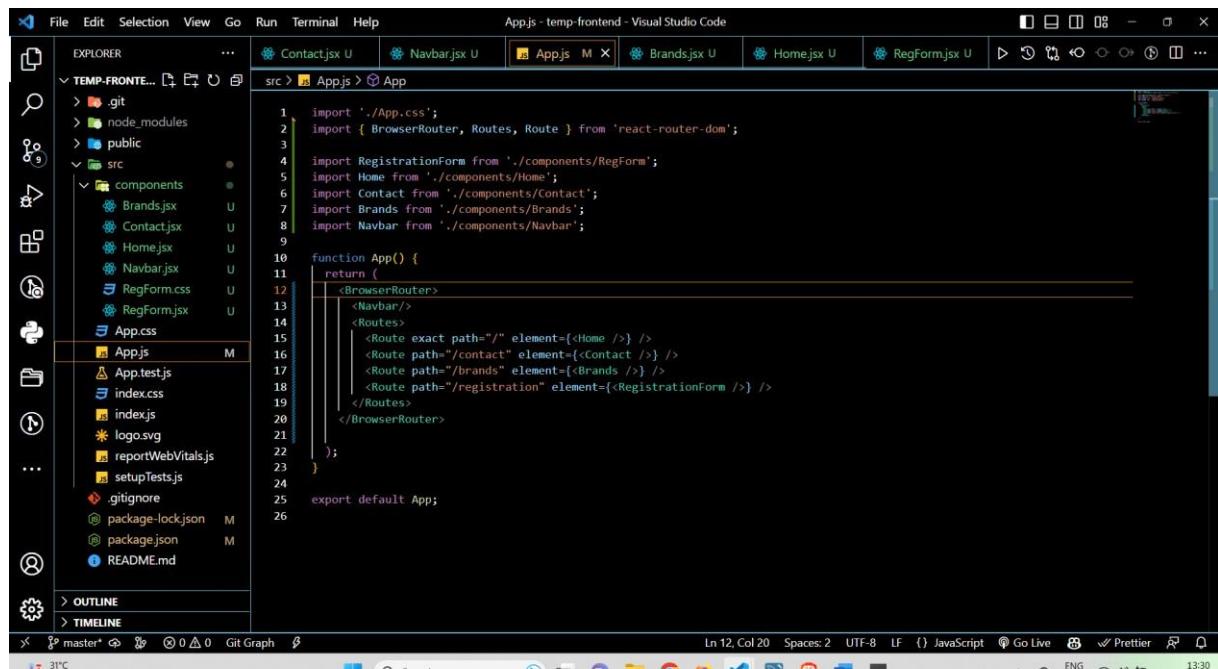
1. `BrowserRouter` Component: The `` component wraps the entire application. It provides the context for routing and is responsible for synchronizing the application's UI with the URL.
2. `Navbar` Component: This component is rendered outside of the `` component, which means it will appear on every page of your application. It likely contains navigation links (e.g., Home, Contact, Brands, Registration), allowing users to navigate to different parts of your app.
3. `Routes` Component: Inside the ``, the `` component is used to define the route configuration for your application. It acts as a container for individual `` components.
4. `Route` Components: Each `` component specifies how a particular URL should map to a React component. For example:
  - The `` with `path="/" element={<Home />}` indicates that when the URL is exactly "/", the `Home` component should be rendered.
  - The `` with `path="/contact" element={<Contact />}` maps the "/contact" URL to the `Contact` component, and so on.

By using `BrowserRouter`, you enable client-side routing. When a user clicks a link or enters a URL, React Router will determine which component to render based on the URL's path. This approach provides a seamless user experience without full page reloads.

**Code :**

Name : Altaf Alam

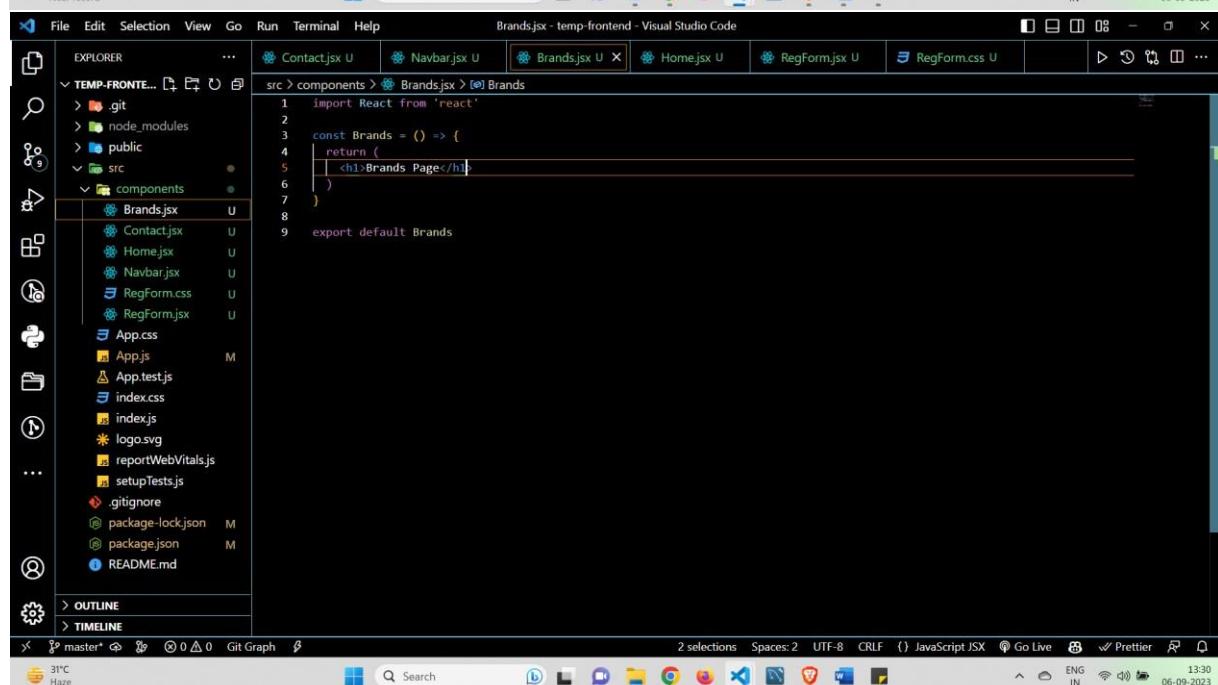
Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23



The screenshot shows the Visual Studio Code interface with the title "App.js - temp-frontend - Visual Studio Code". The code editor displays the following content:

```
1 import './App.css';
2 import { BrowserRouter, Routes, Route } from 'react-router-dom';
3
4 import RegistrationForm from './components/RegForm';
5 import Home from './components/Home';
6 import Contact from './components/Contact';
7 import Brands from './components/Brands';
8 import Navbar from './components/Navbar';
9
10 function App() {
11 return (
12 <BrowserRouter>
13 <Navbar/>
14 <Routes>
15 <Route exact path="/" element={<Home />} />
16 <Route path="/contact" element={<Contact />} />
17 <Route path="/brands" element={<Brands />} />
18 <Route path="/registration" element={<RegistrationForm />} />
19 </Routes>
20 </BrowserRouter>
21);
22 }
23
24 export default App;
```

The Explorer sidebar on the left shows the project structure with files like .git, node\_modules, public, and src containing components like Brands.jsx, Contact.jsx, Home.jsx, Navbar.jsx, RegForm.css, RegForm.jsx, and App.css.

The screenshot shows the Visual Studio Code interface with the title "Brands.jsx - temp-frontend - Visual Studio Code". The code editor displays the following content:

```
1 import React from 'react';
2
3 const Brands = () => {
4 return (
5 <h1>Brands Page</h1>
6)
7 }
8
9 export default Brands;
```

The Explorer sidebar on the left shows the project structure with files like .git, node\_modules, public, and src containing components like Brands.jsx, Contact.jsx, Home.jsx, Navbar.jsx, RegForm.css, RegForm.jsx, and App.css.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

The image shows two instances of Visual Studio Code side-by-side. Both instances have the title bar "Contact.jsx - temp-frontend - Visual Studio Code".

**Left Instance (Contact.jsx):**

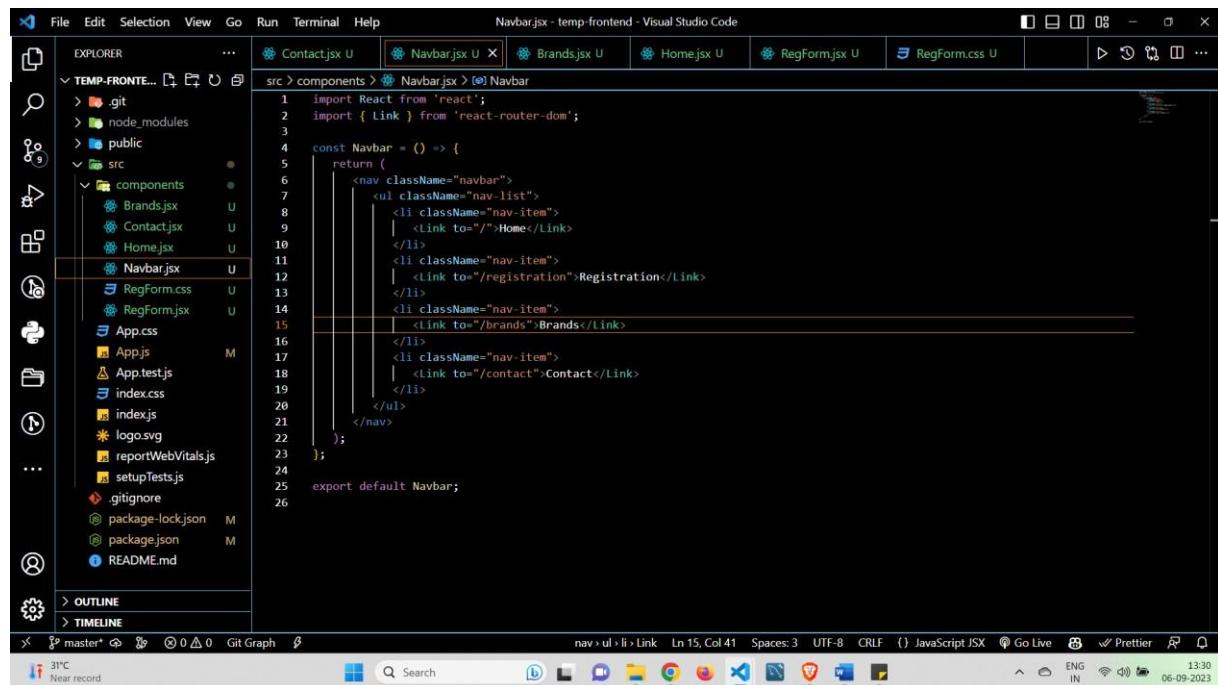
- Explorer: Shows the project structure with files like .git, node\_modules, public, and src. In the src folder, Contact.jsx is selected.
- Editor: Displays the code for Contact.jsx. It includes imports from 'react', a const Contact function returning a JSX structure with a contact container, address, and email sections, and an export default Contact statement.
- Bottom Status Bar: Shows 31°C, Near record, Git Graph, 0 △ 0, Go Live, Prettier, and system icons.

**Right Instance (Home.jsx):**

- Explorer: Shows the project structure with files like .git, node\_modules, public, and src. In the src folder, Home.jsx is selected.
- Editor: Displays the code for Home.jsx. It includes imports from 'react', a const Home function returning a single h1 element with the text "Welcome to the Homepage", and an export default Home statement.
- Bottom Status Bar: Shows 2 selections, Spaces: 2, UTF-8, CRLF, Go Live, Prettier, and system icons.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure with files like .git, node\_modules, public, and src containing components like Brands.jsx, Contact.jsx, Home.jsx, and Navbar.jsx.
- Code Editor:** The active file is Navbar.jsx, displaying the following code:

```
1 import React from 'react';
2 import { Link } from 'react-router-dom';
3
4 const Navbar = () => {
5 return (
6 <nav className="navbar">
7 <ul className="nav-list">
8 <li className="nav-item">
9 | <Link to="/">Home</Link>
10
11 <li className="nav-item">
12 | <Link to="/registration">Registration</Link>
13
14 <li className="nav-item">
15 | <Link to="/brands">Brands</Link>
16
17 <li className="nav-item">
18 | <Link to="/contact">Contact</Link>
19
20
21);
22 };
23
24 export default Navbar;
25
```

The code defines a React component named Navbar that returns a navigation bar with four items: Home, Registration, Brands, and Contact.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

The screenshot displays two instances of the Visual Studio Code editor, both showing the same file: `RegForm.jsx`. The file is part of a React application structure, with components like `App.js`, `Contact.jsx`, `Navbar.jsx`, `Brands.jsx`, `Home.jsx`, and `RegForm.jsx`.

**Top Instance (Initial State):**

```
1 import React, { useState, useRef } from 'react';
2 import './RegForm.css'
3
4 function RegistrationForm() {
5 const [formData, setFormData] = useState({
6 fullName: '',
7 email: '',
8 password: '',
9 gender: 'male',
10 agreeToTerms: false,
11 subscription: 'basic'
12 });
13
14 const inputRef = useRef(null);
15
16 const handleChange = (event) => {
17 const { name, value, type, checked } = event.target;
18 setFormData({
19 ...formData,
20 [name]: type === 'checkbox' ? checked : value,
21 });
22 };
23
24 const handleSubmit = (event) => {
25 event.preventDefault();
26 alert(`Hello ${formData.fullName}, Your Registration is Successful!`);
27 handleReset();
28 };
29
30 const handleReset = () => {
31 setFormData({
32 fullName: '',
33 email: '',
34 password: ''
35 });
36 };
37
38 return (
39 <div className="registration-container">
40 <h2>Registration Form</h2>
41 <form onSubmit={handleSubmit} onKeyUp={handleKeyUp} className="registration-form">
42 <div className="form-group">
43 <label htmlFor="fullName">Full Name:</label>
44 <input
45 type="text"
46 name="fullName"
47 placeholder="Enter full name"
48 value={formData.fullName}
49 onChange={handleChange}
50 id="fullName"
51 ref={inputRef}
52 required
53 />
54 </div>
55 </form>
56 </div>
57);
58}
```

**Bottom Instance (After Adding handleKeyup):**

```
33 email: '',
34 password: '',
35 gender: 'male',
36 agreeToTerms: false,
37 subscription: 'basic',
38);
39
40 const handleKeyUp = (event) => {
41 if (!inputRef.current.contains(document.activeElement)) {
42 if (event.key === 'Escape') {
43 alert('Form closed with Esc key.');
44 handleReset();
45 }
46 }
47 };
48
49 return (
50 <div className="registration-container">
51 <h2>Registration Form</h2>
52 <form onSubmit={handleSubmit} onKeyUp={handleKeyUp} className="registration-form">
53 <div className="form-group">
54 <label htmlFor="fullName">Full Name:</label>
55 <input
56 type="text"
57 name="fullName"
58 placeholder="Enter full name"
59 value={formData.fullName}
60 onChange={handleChange}
61 id="fullName"
62 ref={inputRef}
63 required
64 />
65 </div>
66 </form>
67 </div>
68);
```

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

File Edit Selection View Go Run Terminal Help

RegForm.jsx - temp-frontend - Visual Studio Code

EXPLORER

TEMP-FRONTEND

- > .git
- > node\_modules
- > public
- src
  - components
    - Brands.jsx
    - Contact.jsx
    - Home.jsx
    - Navbar.jsx
    - RegForm.css
    - RegForm.jsx
  - App.css
  - App.js M
  - App.test.js
  - index.css
  - index.js
  - \* logo.svg
  - reportWebVitals.js
  - setupTests.js
  - .gitignore
  - package-lock.json M
  - package.json M
  - README.md

OUTLINE

TIMELINE

App.js M Contact.jsx U Navbar.jsx U Brands.jsx U Home.jsx U RegForm.jsx U RegForm.c

src > components > RegForm.jsx > RegistrationForm

```
66 </div>
67 </div>
68 <div className="form-group">
69 <label>Email Address:</label>
70 <input
71 type="email"
72 name="email"
73 placeholder="Enter email address"
74 value={formData.email}
75 onChange={handleChange}
76 ref={inputRef}
77 required
78 />
79 </div>
80 <div className="form-group">
81 <label>Password:</label>
82 <input
83 type="password"
84 name="password"
85 placeholder="Enter password"
86 value={formData.password}
87 onChange={handleChange}
88 ref={inputRef}
89 required
90 />
91 </div>
92 <div className="form-group">
93 <label>Gender:</label>
94 <select name="gender" value={formData.gender} onChange={handleChange} ref={inputRef}>
95 <option value="male">Male</option>
96 <option value="female">Female</option>
97 <option value="other">Other</option>
98 </select>
```

In 11 Col 29 Spaces: 3 UTF-8 CRLF {} JavaScript JSX Go Live ⚡ Prettier

31°C Haze ENG IN 06-09-2023

```
File Edit Selection View Go Run Terminal Help RegForm.jsx - temp-frontend - Visual Studio Code

EXPLORER ... App.js M Contact.jsx U Navbar.jsx U Brands.jsx U Home.jsx U RegForm.jsx X RegForm.c ...
TEMP-FRONTEND
> .git
> node_modules
> public
src
 > components
 Brands.jsx U
 Contact.jsx U
 Home.jsx U
 Navbar.jsx U
 RegForm.jsx U
 RegForm.css U
App.css M
App.js
App.test.js
index.css
index.jsx
logo.svg
reportWebVitals.js
setupTests.js
.gitignore
package-lock.json M
package.json M
README.md
> OUTLINE
> TIMELINE

src > components > RegForm.jsx > RegistrationForm
 98 | </select>
 99 | </div>
 100 <div className="form-group">
 101 <label>Subscription Plan:</label>
 102 <input
 103 type="radio"
 104 name="subscription"
 105 value="basic"
 106 onChange={handleChange}
 107 checked={formData.subscription === 'basic'}
 108 ref={inputRef}
 109 />(' ')
 110 Basic
 111 <input
 112 type="radio"
 113 name="subscription"
 114 value="premium"
 115 onChange={handleChange}
 116 checked={formData.subscription === 'premium'}
 117 ref={inputRef}
 118 />(' ')
 119 Premium
 120 </div>
 121 <div className="form-checkbox">
 122 <label>
 123 <input
 124 type="checkbox"
 125 name="agreeToTerms"
 126 checked={formData.agreeToTerms}
 127 onChange={handleChange}
 128 ref={inputRef}
 129 />(' ')
 130 I agree to the terms and conditions
 131 </label>

```

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

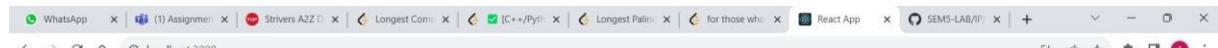
```
122 <label>
123 <input
124 type="checkbox"
125 name="agreeToTerms"
126 checked={formData.agreeToTerms}
127 onChange={handleChange}
128 ref={inputRef}
129 />('
130 I agree to the terms and conditions
131 </label>
132 </div>
133 <div className="form-buttons">
134 <button type="submit" className="submit-button">Submit</button>
135 <button type="reset" onClick={handleReset} className="reset-button">
136 Reset
137 </button>
138 </div>
139 </form>
140 </div>
141 >;
142 >
143 > export default RegistrationForm;
144 >
```

**Output :**

Altaf Alam , 02 , T11

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23



## Welcome to the Homepage

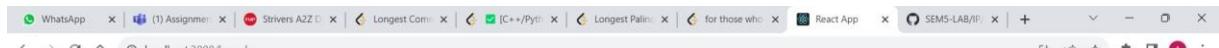
The screenshot shows a registration form titled "Registration Form". The form consists of several input fields:

- Full Name:** A text input field with placeholder text "Enter full name".
- Email Address:** A text input field with placeholder text "Enter email address".
- Password:** A text input field with placeholder text "Enter password".
- Gender:** A dropdown menu currently set to "Male".
- Subscription Plan:** A radio button group where the "Basic" option is selected.
- Terms and Conditions:** A checkbox labeled "I agree to the terms and conditions".

At the bottom of the form are two buttons: "Submit" (blue) and "Reset" (red).

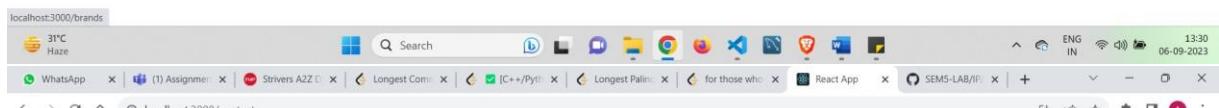
Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23



- [Home](#)
- [Registration](#)
- [Brands](#)
- [Contact](#)

## Brands Page



- [Home](#)
- [Registration](#)
- [Brands](#)
- [Contact](#)

## Contact Us

### Address

123 Main Street  
City, State 12345

### Email

Email: contact@example.com



## Conclusion :

In conclusion, BrowserRouter is a fundamental component in React Router that manages the routing of your application. It allows you to define routes using '`<Route>`' components, making it easy to map URLs to specific components and create a dynamic and responsive user interface.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 20/09/23

## **Assignment No 7b**

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

**Lab Outcome :**

**LO5:** To orient students to React for developing front end applications.

**Theory :**

React Hooks is a feature introduced in React 16.8 that revolutionized how state and sideeffects are managed in functional components. Prior to Hooks, state management and lifecycle methods were primarily handled in class components. Hooks enable functional components to manage local component state, handle side-effects, and access React features, allowing developers to write more readable, reusable, and maintainable code.

The core idea behind React Hooks is to provide a way to "hook into" React state and lifecycle features within functional components, allowing developers to use these features without the need for class components. Here are some key concepts:

1. State Management

2. Effect Handling

3. Context

4. Custom Hooks

5. Rules of Hooks

React Hooks offer several advantages:

- Readability: Functional components with Hooks are often more concise and easier to read than equivalent class components. They reduce boilerplate code.
- Reuse: Hooks promote logic reuse. Custom Hooks can be shared across different components and projects, enhancing code maintainability.
- Separation of Concerns: Hooks separate concerns by allowing logic related to state, effects, and context to be declared separately within a component.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 20/09/23

- Simplified Testing: Testing functional components with Hooks is straightforward because logic can be tested in isolation without the need for React component instances. In React, `useState` and `useRef`, etc. are React hooks that help you work with state and references in functional components.

### **useState:**

useState is a hook in React, a popular JavaScript library for building user interfaces. It allows you to manage and update state within functional components.

Initialization: You use useState to declare a piece of state in your component. It takes an initial value as an argument and returns an array with two elements: the current state value and a function to update it.

Updating State: When you want to change the state, you call the update function with a new value. React will then re-render the component, updating the part of the UI that depends on that state.

### **useRef:**

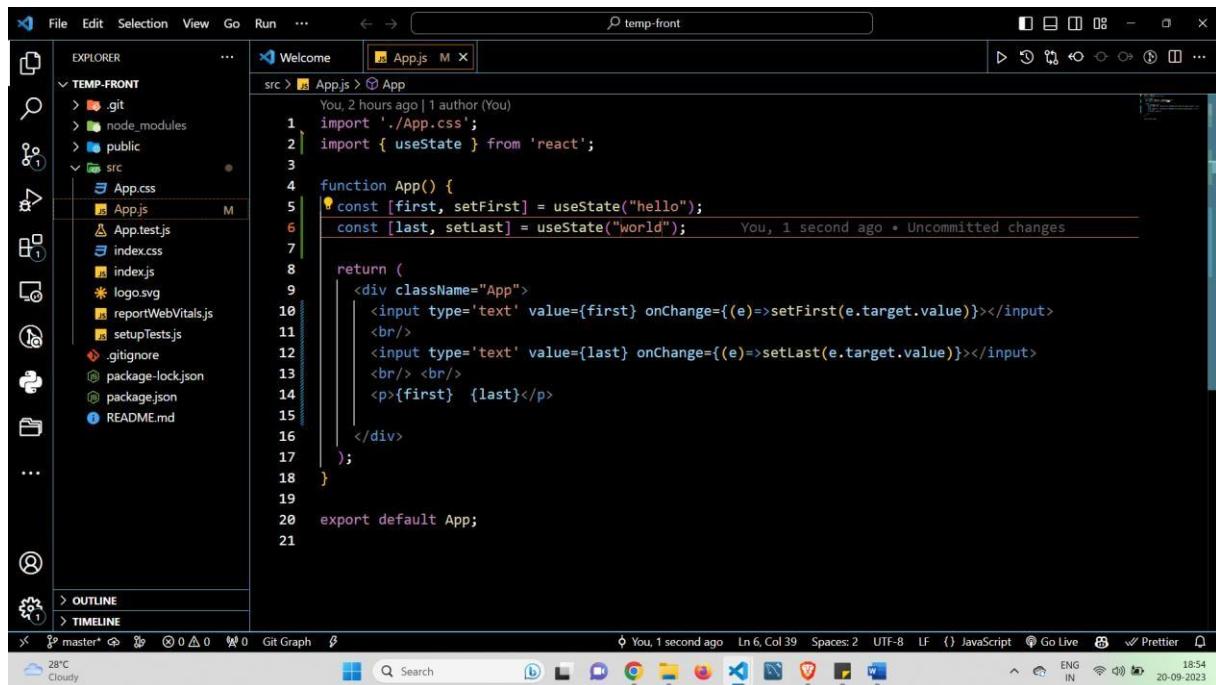
`useRef` is a React hook that provides a way to create mutable object references that persist across renders. It is commonly used for accessing and interacting with DOM elements, managing focus, and persisting values between renders. Key points about `useRef` :

1. Mutable Refs: `useRef` creates a mutable object with a `current` property. This property can be assigned a value, and it retains its value between renders without causing re-renders.
2. Accessing DOM Elements: One common use of `useRef` is to access and manipulate DOM elements directly. You can attach a `ref` attribute to a JSX element and assign it to a `useRef` variable.
3. Preserving Values: `useRef` is suitable for preserving values or state between renders without causing re-renders. This makes it different from `useState` .

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 20/09/23

## Code :



The screenshot shows the Visual Studio Code interface. On the left is the Explorer sidebar with project files like .git, node\_modules, public, and src containing App.css, App.js, App.test.js, index.css, logo.svg, reportWebVitals.js, setupTests.js, .gitignore, package-lock.json, package.json, and README.md. The center is the Editor tab showing App.js with the following code:

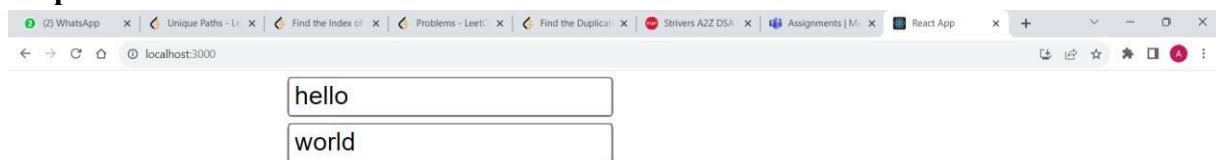
```
1 import './App.css';
2 import { useState } from 'react';
3
4 function App() {
5 const [first, setFirst] = useState("hello");
6 const [last, setLast] = useState("world");
7
8 return (
9 <div className="App">
10 <input type='text' value={first} onChange={(e)=>setFirst(e.target.value)}></input>
11

12 <input type='text' value={last} onChange={(e)=>setLast(e.target.value)}></input>
13

14 <p>{first} {last}</p>
15 </div>
16);
17
18}
19
20 export default App;
```

The bottom status bar shows the file is 1 second ago, has 6 lines, 39 columns, and 2 spaces. It also indicates the code is in JavaScript mode, has 1 uncommitted change, and is using Prettier.

## Output :



hello world



## Conclusion :

In conclusion, React hooks provide a more declarative and functional approach to managing component state and interacting with the DOM.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 20/09/23

## **Assignment No 8**

**Aim** : WAP to implement the calculator on commandline REPL.

**Lab Outcome :**

**LO5:** To orient students to React for developing front end applications.

**Theory :**

Node.js Calculator Using REPL (Command-Line Input)

1. **REPL (Read-Eval-Print Loop):** The program utilizes a REPL environment, which stands for Read-Eval-Print Loop. This is an interactive command-line interface where users can input commands or expressions, and the system immediately evaluates and returns results. It's commonly used for experimentation, testing, and quick code execution.
2. **Importing Modules:** The program begins by importing the 'readline' module, which is a core Node.js module for reading input from the user in the command line. This module provides an interface for user interaction.
3. **Creating the Readline Interface:** An instance of the 'readline' interface is created, specifying that it should read input from 'stdin' (standard input) and display output in 'stdout' (standard output). The 'terminal: false' option is set to disable certain terminal-specific features.
4. **User Input and Prompts:** The program uses the 'rl.question' function to prompt the user for input in a series of steps. It starts by asking for the first number, then the second number, and finally the desired mathematical operation (addition, subtraction, multiplication, or division).
5. **Parsing User Input:** After gathering user input, the program parses the entered values into numeric format (floats) to ensure they are suitable for mathematical operations. This step involves using 'parseFloat' to convert the input.
6. **Mathematical Operations:** Depending on the user's chosen operation, the program performs the corresponding mathematical calculation (e.g., addition, subtraction, multiplication, or division). This is achieved using conditional statements, such as a 'switch' statement, to determine the operation and execute the appropriate calculation.
7. **Result Display:** The calculated result is then displayed on the command line for the user to see. It provides immediate feedback on the outcome of the operation.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 20/09/23

8. Error Handling: The program includes error handling to address potential issues. For instance, it checks for invalid inputs (non-numeric values) and handles division by zero to prevent errors and crashes.

9. Interactive User Experience: Users interact with the program by entering input at each prompt, and the program responds with real-time feedback, making it a user-friendly calculator tool.

**Output :**

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 20/09/23

```
C:\Users\abdul>node
Welcome to Node.js v16.17.0.
Type ".help" for more information.
> .editor
// Entering editor mode (Ctrl+D to finish, Ctrl+C to cancel)
const readline = require('readline');

const rl = readline.createInterface({
 input: process.stdin,
 output: process.stdout,
 terminal: false
});

rl.question('Enter the first number: ', (num1) => {
 rl.question('Enter the second number: ', (num2) => {
 rl.question('Select an operation (add, subtract, multiply, divide): ', (operation) => {
 const number1 = parseFloat(num1);
 const number2 = parseFloat(num2);

 if (!isNaN(number1) && !isNaN(number2)) {
 let result;

 switch (operation.toLowerCase()) {
 case 'add':
 result = number1 + number2;
 break;
 case 'subtract':
 result = number1 - number2;
 break;
 case 'multiply':
 result = number1 * number2;
 break;
 case 'divide':
 if (number2 === 0) {
 console.log('Error: Division by zero');
 rl.close();
 return;
 }
 result = number1 / number2;
 }

 console.log(`Result: ${result}`);
 } else {
 console.log('Invalid input. Please enter valid numbers.');
 }

 rl.close();
 });
 });
});

Enter the first number: undefined
> 5
> Enter the second number: 7
> Select an operation (add, subtract, multiply, divide): add
Uncaught ReferenceError: add is not defined
> Result: 12
C:\Users\abdul>
```

## Conclusion :

In summary, the Node.js calculator program demonstrates the principles of a REPL environment, allowing users to interactively input mathematical operations and receive immediate results.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 28/09/23

## **Assignment No 9**

**Aim :** Write a program to implement a.

- Create React refs
- b. How to access Refs
- c. Forward Refs
- d. Callback Refs.

### **Lab Outcome :**

**LO5:** To orient students to React for developing front end applications.

### **Theory :**

React is a popular JavaScript library for building user interfaces, and it provides various tools and patterns to efficiently manage and manipulate the Document Object Model (DOM) elements. One of these tools is React Refs, which allow developers to access and interact with DOM elements directly. With the introduction of React Hooks, working with refs has become more accessible and intuitive. In this guide, we'll explore how to create and use React refs effectively using functional components and hooks.

#### a. Creating React Refs:

React Refs are objects that provide a way to access and interact with DOM elements directly. In functional components, you can create refs using the `useRef` hook. Here's how to create a ref:

```
```javascript
const myRef = useRef(null);
````
```

The `useRef` hook initializes the `myRef` variable with the `current` property set to `null`. You can attach this ref to a DOM element by assigning it as a `ref` attribute in the JSX.

#### b. How to Access Refs:

Once you've created a ref, you can access and manipulate the corresponding DOM element. To access the DOM element, you can use the `current` property of the ref. For example:

```
```javascript
const element = myRef.current;
if (element) {
  // Access and manipulate the DOM element
}
```

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 28/09/23

```

It's  
imp  
orta  
nt to  
chec  
k if  
the  
elem  
ent  
exist  
s  
befo  
re  
acce  
ssin  
g  
and  
mani  
pulat  
ing  
it  
beca  
use  
the  
'current' property may initially be 'null'.

#### c. Forward Refs:

React allows you to forward refs from a parent component to a child component. This is particularly useful when you want to access a child's DOM element from a parent component. To create a forward ref, use the `React.forwardRef` function:

```
```javascript const ChildComponent =  
React.forwardRef((props, ref) => {  
  // JSX for child component  
  return <input ref={ref} />;  
});  
```
```

In this example, the `ChildComponent` accepts a `ref` parameter and attaches it to the `input` element. You can then use this forwarded ref in a parent component:

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 28/09/23

```
```javascript function ParentComponent() {  
  const childRef = useRef(null);  
  
  // Attach ref to the child component  
  return <ChildComponent ref={childRef} />;  
}  
```
```

Now, `childRef.current` refers to the `input` element inside `ChildComponent`.

#### d. Callback Refs:

Callback refs are another way to work with refs in React. Instead of using the `useRef` hook, you define a function that receives the DOM element as an argument and stores it as a variable. Here's how to create and use a callback ref:

```
```javascript let myRef = null;  
const setMyRef = (element) => {  
  myRef = element;  
};  
```
```

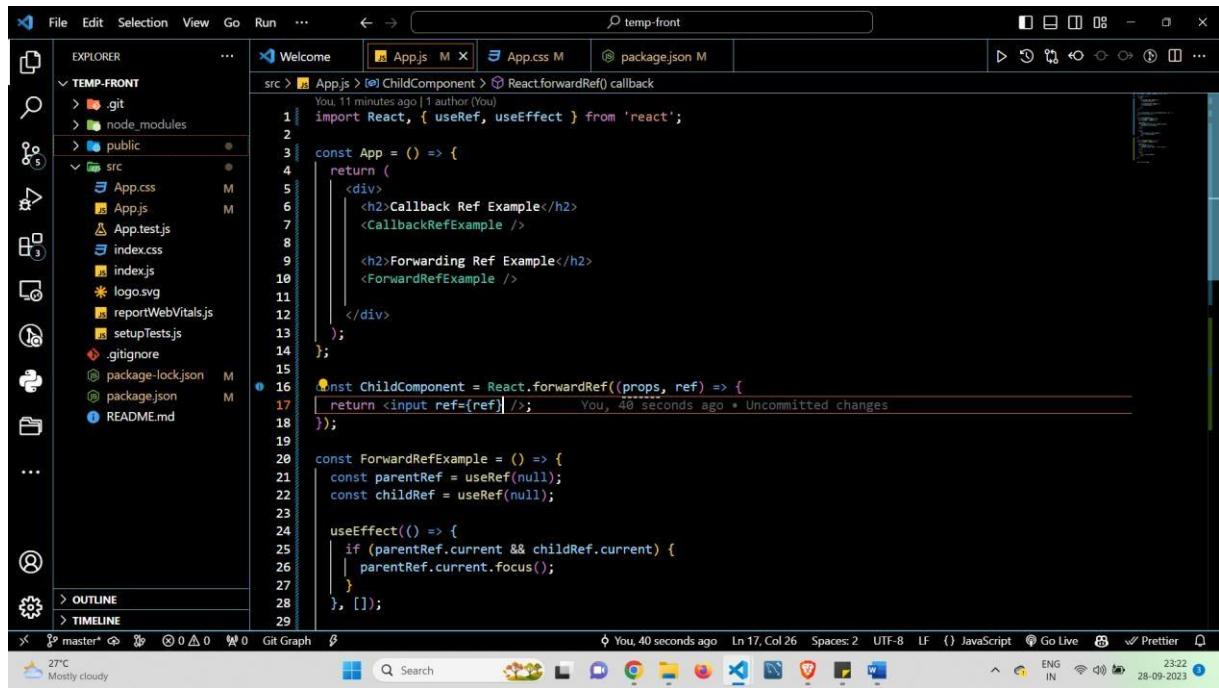
In this example, `setMyRef` is a callback function that receives the DOM element as an argument and assigns it to the `myRef` variable. You can then use `myRef` to access and manipulate the DOM element. Callback refs are typically defined inside the functional component.

React Hooks and functional refs provide a flexible and powerful way to work with DOM elements in your React applications. Whether you're creating refs, accessing DOM elements, forwarding refs to child components, or using callback refs, these techniques empower you to build dynamic and interactive user interfaces with ease.

#### Code :

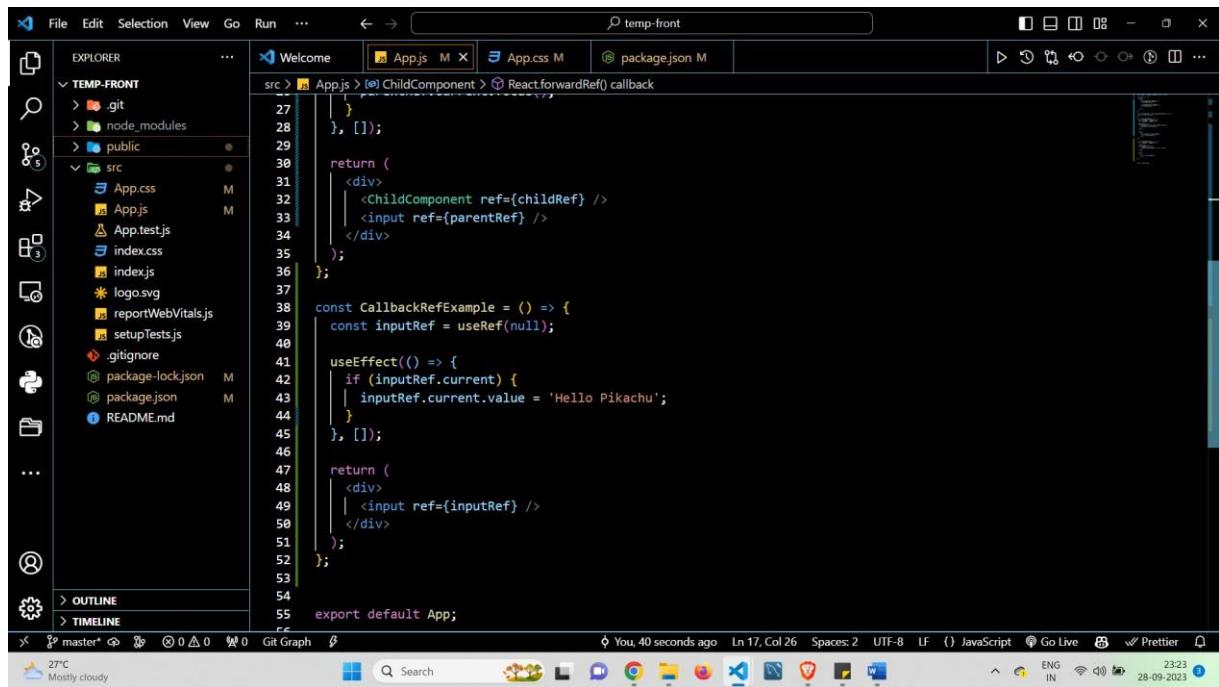
Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 28/09/23



The screenshot shows the Visual Studio Code interface. The left sidebar contains the Explorer, showing a project structure for 'TEMP-FRONT' with files like .git, node\_modules, public, and src. The src folder contains App.css, App.js, App.test.js, index.css, index.js, logo.svg, reportWebVitals.js, setupTests.js, .gitignore, package-lock.json, package.json, and README.md. The main editor area displays a portion of 'App.js' with code related to React.forwardRef(). The status bar at the bottom shows the file is 27°C, mostly cloudy, and the date is 28-09-2023.

```
src > App.js > ChildComponent > React.forwardRef() callback
1 import React, { useRef, useEffect } from 'react';
2
3 const App = () => {
4 return (
5 <div>
6 <h2>Callback Ref Example</h2>
7 <CallbackRefExample />
8
9 <h2>Forwarding Ref Example</h2>
10 <ForwardRefExample />
11 </div>
12);
13}
14
15 const ChildComponent = React.forwardRef((props, ref) => {
16 return <input ref={ref} />; You, 40 seconds ago * Uncommitted changes
17 });
18
19
20 const ForwardRefExample = () => {
21 const parentRef = useRef(null);
22 const childRef = useRef(null);
23
24 useEffect(() => {
25 if (parentRef.current && childRef.current) {
26 | parentRef.current.focus();
27 }
28 }, []);
29}
```



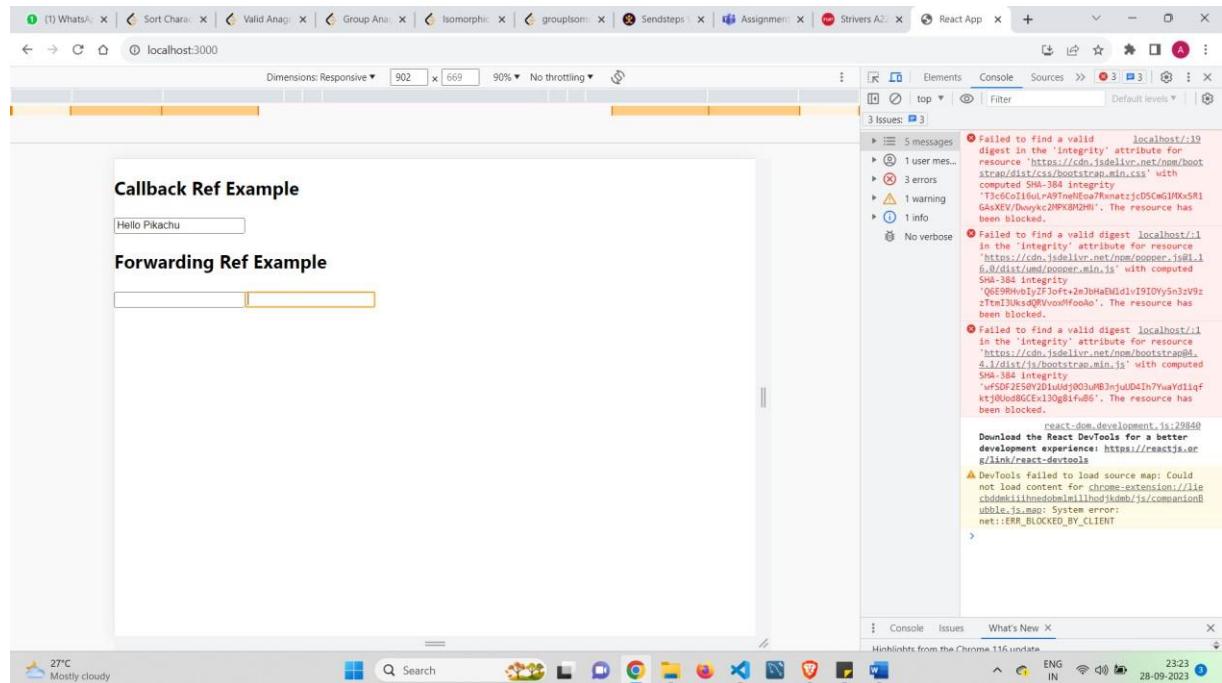
The screenshot shows the Visual Studio Code interface. The left sidebar contains the Explorer, showing a project structure for 'TEMP-FRONT' with files like .git, node\_modules, public, and src. The src folder contains App.css, App.js, App.test.js, index.css, index.js, logo.svg, reportWebVitals.js, setupTests.js, .gitignore, package-lock.json, package.json, and README.md. The main editor area displays a portion of 'App.js' with code related to React.forwardRef(). The status bar at the bottom shows the file is 27°C, mostly cloudy, and the date is 28-09-2023.

```
src > App.js > ChildComponent > React.forwardRef() callback
27 },
28];
29
30 return (
31 <div>
32 <ChildComponent ref={childRef} />
33 <input ref={parentRef} />
34 </div>
35);
36 }
37
38 const CallbackRefExample = () => {
39 const inputRef = useRef(null);
40
41 useEffect(() => {
42 if (inputRef.current) {
43 | inputRef.current.value = 'Hello Pikachu';
44 }
45 }, []);
46
47 return (
48 <div>
49 <input ref={inputRef} />
50 </div>
51);
52 }
53
54 export default App;
```

## Output :

Name : Altaf Alam

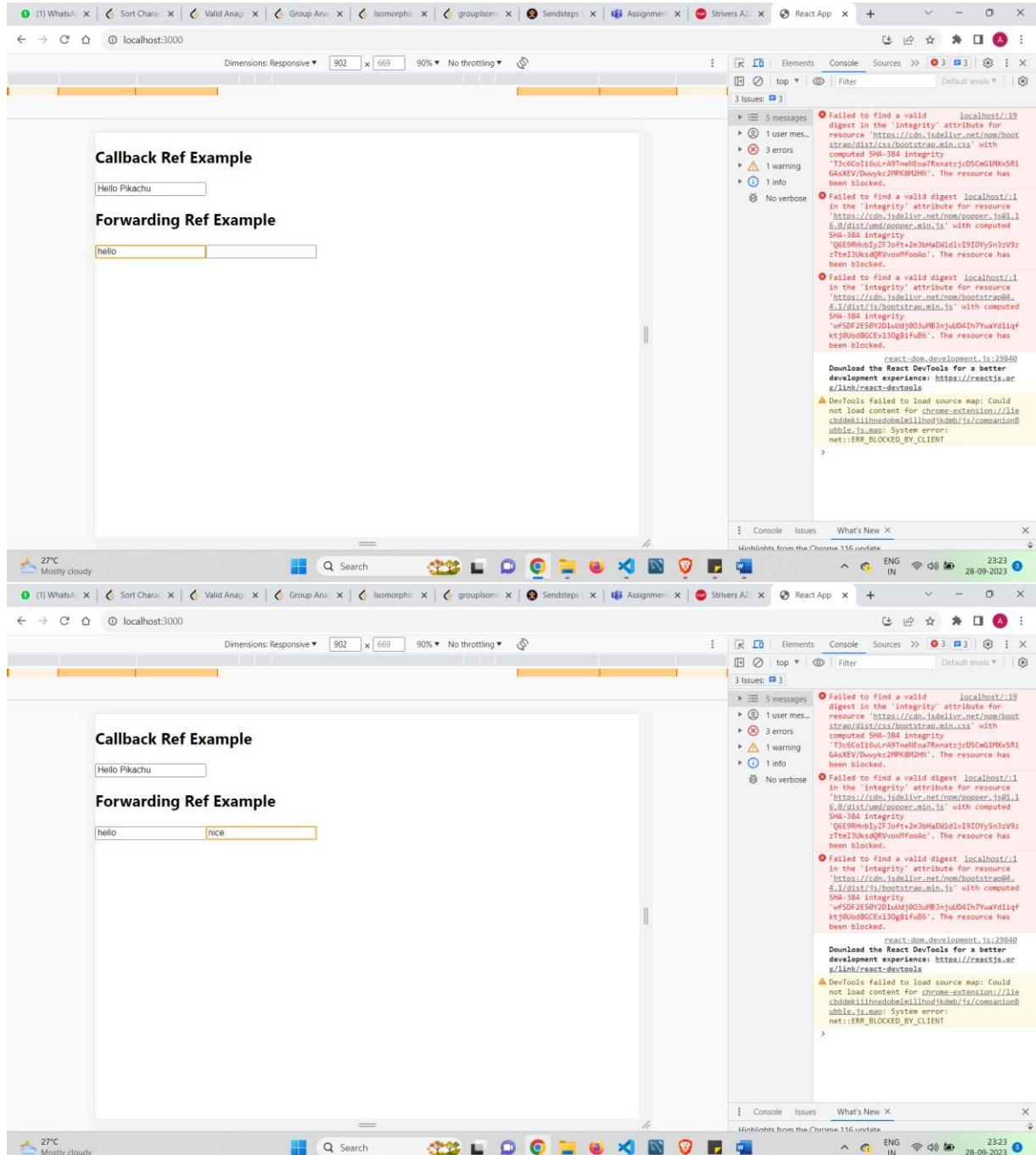
Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 28/09/23



Altaf Alam , 02 , T11

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 28/09/23



## Conclusion :

In conclusion, React Hooks and functional refs offer a streamlined approach to managing DOM elements in React applications, enhancing accessibility and ease of use. These tools empower developers to create dynamic and interactive user interfaces with efficiency and flexibility.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 23/09/23

## **Assignment No 10**

**Aim :** Write a program in Node JS to

- a. Create a file
- b. Read the data from file
- c. Write the data to a file
- d. Rename a file
- e. Append data to a file
- f. Delete a file

### **Lab Outcome :**

**LO6:** To orient students to Nodejs for developing back end applications.

### **Theory :**

Node.js is a powerful runtime environment for executing JavaScript on the server side. It provides built-in modules, including the 'fs' (File System) module, that allow developers to perform various file operations. Here, we will explore how to perform common file operations in Node.js:

#### a. Creating a File

To create a file in Node.js, you can use the 'fs' (File System) module. You typically use the `fs.writeFile()` method, specifying the file name and content. This method will create the file if it doesn't exist or overwrite its content if it does.

#### b. Reading Data from a File

Reading data from a file involves using the 'fs' module's `fs.readFile()` method. You provide the file name and an encoding (e.g., 'utf8' for text files) to read the file's content. The method then asynchronously reads the content and provides it as a callback argument.

#### c. Writing Data to a File

To write data to an existing file, you can use the 'fs' module's `fs.writeFile()` method. Similar to creating a file, you specify the file name and the content you want to write. This method overwrites the file's existing content with the new data.

#### d. Renaming a File

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 23/09/23

To rename a file in Node.js, you use the 'fs' module's `fs.rename()` method. This method accepts two arguments: the current file name and the new file name. It effectively renames the file by changing its name in the file system.

#### e. Appending Data to a File

Appending data to a file without overwriting its existing content can be done using the 'fs' module's `fs.appendFile()` method. You provide the file name and the data to append. This method appends the data to the end of the file, preserving the existing content.

#### f. Deleting a File

To delete a file, you utilize the 'fs' module's `fs.unlink()` method. You specify the file name as the argument, and this method removes the file from the file system.

In Node.js, these file operations are typically performed asynchronously, allowing your application to continue executing other tasks while file operations are in progress. Proper error handling is crucial to catch and manage any errors that may occur during these operations, ensuring the robustness and reliability of your Node.js applications.

#### Code :

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 23/09/23

The image shows two screenshots of a code editor interface, likely Visual Studio Code, demonstrating file operations in JavaScript. Both screenshots show the same code structure but with different execution steps highlighted.

**Top Screenshot:** This screenshot shows the initial state of the script. The code performs the following steps:

- Line 1: `const fs = require('fs');` - Imports the file system module.
- Line 2: `// Write to the file` - A comment indicating the purpose of the next operation.
- Line 3: `fs.writeFile('example.txt', 'Hello, Node.js!', (err) => {` - Writes "Hello, Node.js!" to a file named "example.txt".
- Line 4: `if (err) throw err;` - Checks for errors and throws them if any occur.
- Line 5: `console.log('File created and content written.');// Read the file (1st read)` - Logs a message and then reads the file again.
- Line 6: `fs.readFile('example.txt', 'utf8', (err, data) => {` - Reads the file "example.txt" using UTF-8 encoding.
- Line 7: `if (err) throw err;` - Checks for errors and throws them if any occur.
- Line 8: `console.log('File content (1st read):', data);` - Logs the content of the file after the first read.
- Line 9: `// Append to the file` - A comment indicating the purpose of the next operation.
- Line 10: `fs.appendFile('example.txt', '\nThis is an appended line.', (err) => {` - Appends a new line ("This is an appended line.") to the file.
- Line 11: `if (err) throw err;` - Checks for errors and throws them if any occur.
- Line 12: `console.log('Content appended to the file.');// Read the file after append (2nd read)` - Logs a message and then reads the file again.
- Line 13: `fs.readFile('example.txt', 'utf8', (err, data) => {` - Reads the file "example.txt" using UTF-8 encoding.
- Line 14: `if (err) throw err;` - Checks for errors and throws them if any occur.
- Line 15: `console.log('File content after append (2nd read):', data);` - Logs the content of the file after the second read.
- Line 16: `// Update the file` - A comment indicating the purpose of the next operation.
- Line 17: `fs.readFile('example.txt', 'utf8', (err, data) => {` - Reads the file "example.txt" using UTF-8 encoding.
- Line 18: `if (err) throw err;` - Checks for errors and throws them if any occur.
- Line 19: `const updatedContent = data.replace('Hello', 'Hi');` - Replaces the word "Hello" with "Hi" in the file content.

**Bottom Screenshot:** This screenshot shows the state of the script after the update operation has been completed. The code continues from where it left off:

- Line 20: `fs.writeFile('example.txt', updatedContent, (err) => {` - Writes the updated content back to the file.
- Line 21: `if (err) throw err;` - Checks for errors and throws them if any occur.
- Line 22: `console.log('File updated.');// Read the file after update (3rd read)` - Logs a message and then reads the file again.
- Line 23: `fs.readFile('example.txt', 'utf8', (err, data) => {` - Reads the file "example.txt" using UTF-8 encoding.
- Line 24: `if (err) throw err;` - Checks for errors and throws them if any occur.
- Line 25: `console.log('File content after update (3rd read):', data);` - Logs the content of the file after the third read.
- Line 26: `// Rename the file` - A comment indicating the purpose of the next operation.
- Line 27: `fs.rename('example.txt', 'renamed-example.txt', (err) => {` - Renames the file "example.txt" to "renamed-example.txt".
- Line 28: `if (err) throw err;` - Checks for errors and throws them if any occur.
- Line 29: `console.log('File renamed to renamed-example.txt.');//` - Logs a message indicating the file has been renamed.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 23/09/23

```
File Edit Selection View Go Run ... < > ⌂ temp-back
EXPLORER TEMP-BACK
node_modules package-lock.json package.json renamed-example.txt server.js
Welcome server.js
50
51
52 // Create an empty file
53 const fileName = 'delete-file.txt';
54 fs.open(fileName, 'w', (err, fileDescriptor) => {
55 if (err) throw err;
56
57 console.log(`File '${fileName}' has been created.`);
58
59 // Close the file after creating it
60 fs.close(fileDescriptor, (err) => {
61 if (err) throw err;
62 console.log(`File '${fileName}' has been closed.`);
63
64 // Delete the newly created file
65 fs.unlink('delete-file.txt', (err) => {
66 if (err) throw err;
67 console.log('File deleted delete-file.txt.');
68 });
69 });
70 });
71
Ln 51, Col 1 Spaces: 3 UTF-8 CRLF {} JavaScript Go Live Prettier
26°C Near record Search
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell
Code
23-09-2023 19:54
```

## Output :

```
File Edit Selection View Go Run ... < > ⌂ temp-back
EXPLORER TEMP-BACK
node_modules package-lock.json package.json renamed-example.txt server.js
Welcome server.js
50
51
52
53 PS D:\temp\temp-back> node "d:\temp\temp-back\server.js"
File created and content written.
File 'delete-file.txt' has been created.
File 'delete-file.txt' has been closed.
File content (1st read): Hello, Node.js!
File deleted delete-file.txt.
Content appended to the file.
File content after append (2nd read): Hello, Node.js!
This is an appended line.
File updated.
File content after update (3rd read): Hi, Node.js!
This is an appended line.
File renamed to renamed-example.txt.
o PS D:\temp\temp-back>
```

## Conclusion :

In conclusion, we learnt about the file system in Nodejs and different operations like create, write, read, delete etc using Nodejs.

## Assignment No 11

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 23/09/23

**Aim :** Create a web application that performs CRUD operations (database connectivity).

**Lab Outcome :**

**LO1, LO2, LO3, LO4, LO5, LO6**

**Theory :**

Todo app is a web application built using React for the frontend and Express.js with MongoDB for the backend. It allows users to perform CRUD (Create, Read, Update, Delete) operations on a list of to-do items. Here's a brief explanation:

1. Create: Users can add new to-do items by entering text and clicking the "Add" button.
2. Read: The app displays a list of existing to-do items, and users can see their to-do list.
3. Update: Users can edit existing to-do items by clicking the "Edit" button, making changes, and clicking "Update."
4. Delete: Users can delete to-do items by clicking the "Delete" button.

The app stores the to-do items in a MongoDB database and uses Axios to communicate with the Express.js backend for data retrieval, creation, updating, and deletion. It follows a typical CRUD architecture with separate routes and controllers for each operation.

**Here's a brief overview for the project:**

React Frontend ('App.js', 'ToDo.js'):

1. 'App.js' is the main React component for the frontend.
  2. It uses 'useState' and 'useEffect' hooks to manage the application's state and perform side effects.
  3. 'useState' is used to manage the state of 'ToDo' (an array of to-do items), 'text' (the text for a new to-do item), 'isUpdating' (a boolean flag for update mode), and 'toDoId' (the ID of the to-do item being updated).
  4. In the 'useEffect' hook, it fetches all to-do items when the component mounts using the 'getAllToDo' function.
  5. The 'updateMode' function is called when an item is to be updated. It sets 'isUpdating' to 'true', populates 'text' with the current item's text, and sets 'toDoId' to the item's ID.
  6. The component renders a form to add or update to-do items and lists existing to-do items using the 'ToDo' component.
- 'ToDo.js' is a functional component for rendering individual to-do items.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 23/09/23

- It receives `text`, `updateMode`, and `deleteToDo` as props.
- It renders the to-do text and icons for editing and deleting.

Frontend Functionality (`utils/HandleApi.js`):

1. The `addToDo`, `updateToDo`, and `deleteToDo` functions make HTTP requests to the server using Axios to perform the corresponding CRUD operations.
2. They interact with the backend API to create, update, and delete to-do items.

Express.js Backend (`server.js`, `routes/ToDoRoute.js`, `controllers/ToDoController.js`):

1. `server.js` sets up the Express.js server, establishes a connection to the MongoDB database, and listens on a specified port (in this case, port 5000).
2. `ToDoRoute.js` defines the routes for CRUD operations using Express Router. It specifies the routes for `get`, `post` (create), `put` (update), and `delete` (delete) operations.
3. `ToDoController.js` contains controller functions for handling the CRUD operations.
  - `getToDo` retrieves all to-do items from the database.
  - `saveToDo` creates a new to-do item.
  - `updateToDo` updates an existing to-do item.
  - `deleteToDo` deletes a to-do item.

Database (`models/ToDoModel.js`):

1. `ToDoModel.js` defines the MongoDB schema for to-do items, specifying that each item has a `text` field.

## Code :

Frontend :

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 23/09/23

```
File Edit Selection View Go Run ...
Todo-frontend
EXPLORER ... HandleApi.js App.js App.css index.css index.js package.json tailwind.config.js
TODO-FRONTEND ...
public index.html
src ...
components ToDojs
utils HandleApijs
App.css
App.js
index.css
index.js
reportWebVitals.js
setupTests.js
package-lock.json
package.json
README.md
tailwind.config.js
OUTLINE
TIMELINE
src > App.js > App > useEffect() callback
6 function App() {
7 const [todo, setTodo] = useState([]);
8 const [text, setText] = useState("");
9 const [isUpdating, setIsUpdating] = useState(false);
10 const [todoId, setTodoId] = useState("");
11
12 useEffect(() => {
13 getTodos();
14 }, []);
15
16 const updateMode = (_id, text) => {
17 setIsUpdating(true);
18 setText(text);
19 setTodoId(_id);
20 }
21
22 return (
23 <div className="app">
24 <div className="container">
25 <h1>ToDo App</h1>
26 <div className="top">
27 <input type="text" placeholder="Add Todos..." value={text} onChange={(e) => setText(e.target.value)} />
28 <button onClick={() => addTodo(text, setText, setTodo)}>Add</button>
29 </div>
30 <div className="list">
31
32 {todo.map(item) => (
33 <li key={item._id}>
34 {text}

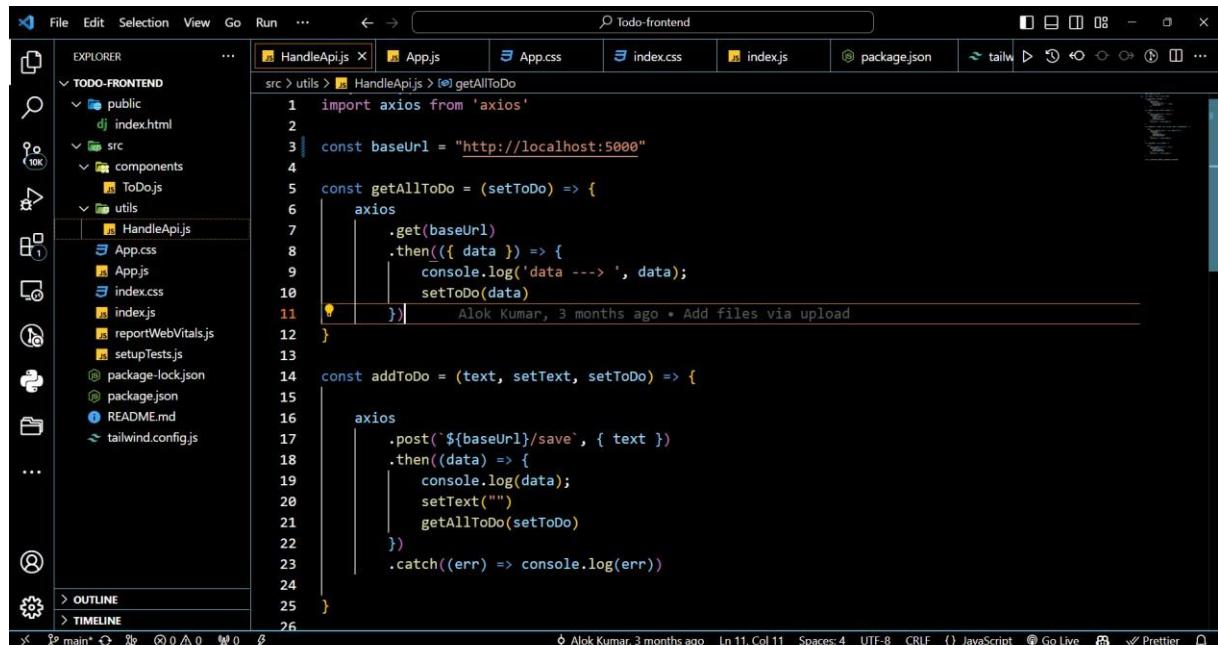
35 <button onClick={() => updateMode(item._id, item.text)}>Update</button>
36 <button onClick={() => deleteTodo(item._id, setTodos)}>Delete</button>
37
38)}
39
40 </div>
41 </div>
42 </div>
43);
44}
45
```

The screenshot shows the VS Code interface with the following details:

- File Explorer (Left):** Shows the project structure under "TODO-FRONTEND". The "components" folder contains "ToDo.js". Other files include "index.css", "index.js", "package.json", "tailwind.config.js", "ToDo.js", "utils", "HandleApi.js", "App.css", "App.js", "index.css", "index.js", "reportWebVitals.js", "setupTests.js", "package-lock.json", "package.json", "README.md", and "tailwind.config.js".
- Search Bar (Top):** Contains the text "Todo-frontend".
- Code Editor (Center):** Displays the content of "ToDo.js". The code defines a "ToDo" component that takes "text", "updateMode", and "deleteToDo" as props. It renders a div with class "todo" containing a text div and an icon div. The icon div contains two icons: "BiEdit" and "AiFillDelete", each with an onClick handler. The code ends with an export default statement.
- Bottom Status Bar:** Shows file paths ("main\*", "index.js", "index.css"), line and column numbers (Ln 18, Col 1), spaces count (Spaces: 2), CRLF indicator, language (JavaScript), Go Live button, Prettier button, and system status (26°C, Cloudy).

Name : Altaf Alam

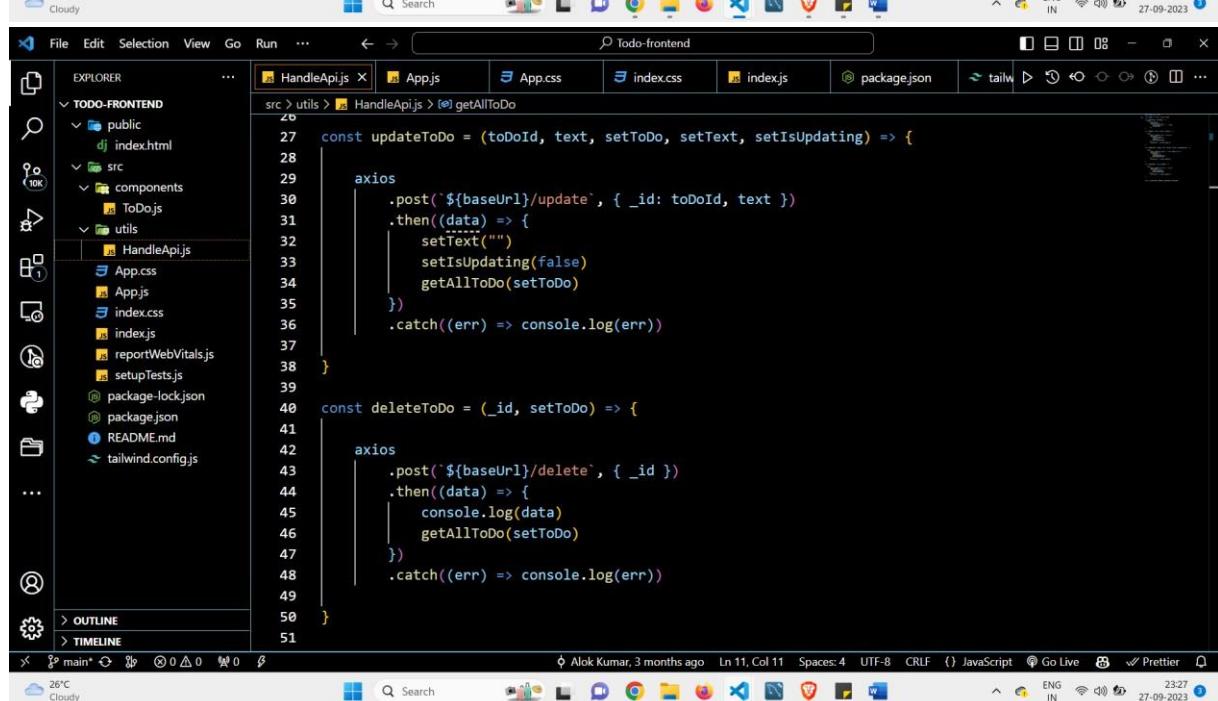
Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 23/09/23



The screenshot shows the VS Code interface with the 'Todo-frontend' project open. The Explorer sidebar on the left shows the project structure, including 'public', 'src' (containing 'components' and 'utils'), and 'utils'. The 'utils' folder contains 'HandleApi.js'. The 'HandleApi.js' file is the active tab, showing the following code:

```
1 import axios from 'axios'
2
3 const baseUrl = "http://localhost:5000"
4
5 const getAllToDo = (setToDo) => {
6 axios
7 .get(baseUrl)
8 .then(({ data }) => {
9 console.log('data -->', data);
10 setToDo(data)
11 })
12 }
13
14 const addToDo = (text, setText, setToDo) => {
15
16 axios
17 .post(`${baseUrl}/save`, { text })
18 .then((data) => {
19 console.log(data);
20 setText("")
21 getAllToDo(setToDo)
22 })
23 .catch((err) => console.log(err))
24 }
25
26 }
```

The screenshot also shows the Windows taskbar at the bottom with various pinned icons.



The second screenshot shows the same VS Code interface with the 'utils' folder expanded. It displays two additional functions: 'updateToDo' and 'deleteToDo'.

**updateToDo Function:**

```
27 const updateToDo = (todoId, text, setToDo, setText, setIsUpdating) => {
28
29 axios
30 .post(`${baseUrl}/update`, { _id: todoId, text })
31 .then(({ data }) => {
32 setText("")
33 setIsUpdating(false)
34 getAllToDo(setToDo)
35 })
36 .catch((err) => console.log(err))
37 }
38
39
40 const deleteToDo = (_id, setToDo) => {
41
42 axios
43 .post(`${baseUrl}/delete`, { _id })
44 .then((data) => {
45 console.log(data)
46 getAllToDo(setToDo)
47 })
48 .catch((err) => console.log(err))
49 }
50
51 }
```

The taskbar at the bottom is identical to the first screenshot.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 23/09/23

Backend :

The screenshot shows a code editor interface with the title bar "Todo-backend". The left sidebar displays a project structure under "TODO-BACKEND" with files: controllers (ToDoController.js), models (ToDoModel.js), routes (ToDoRoute.js), .env, package-lock.json, package.json, README.md, and Server.js. The main editor area shows the content of Server.js:

```
const express = require("express");
const mongoose = require("mongoose");
const cors = require("cors");
const routes = require("./routes/ToDoRoute");
require('dotenv').config();
const app = express();
const PORT = process.env.port || 5000;
app.use(express.json());
app.use(cors());
mongoose
 .connect(process.env.MONGODB_URL, {
 useNewUrlParser: true,
 useUnifiedTopology: true
 })
 .then(() => console.log(`Connected To MongoDB...`))
 .catch(err) => console.log(err);
app.use(routes);
app.listen(PORT, () => {
 console.log(`Listening on: ${PORT}`);
})
```

The code editor includes status bars at the bottom showing "Alok Kumar, 3 months ago", "Ln 18, Col 26", "Spaces: 4", "UTF-8", "CRLF", "JavaScript", "Go Live", "Prettier", and system icons like battery level and network.

The screenshot shows a code editor interface with the title bar "Todo-backend". The left sidebar displays a project structure under "TODO-BACKEND" with files: controllers (ToDoController.js), models (ToDoModel.js), routes (ToDoRoute.js), .env, package-lock.json, package.json, README.md, and Server.js. The main editor area shows the content of ToDoRoute.js:

```
// CURD :-
// GET(read) POST(create) PUT(update) DELETE(delete)
const express = require("express");
const router = express.Router();
const { getToDo, saveToDo, deleteToDo, updateToDo } = require("../controllers/ToDoController");
router.get('/', getToDo);
router.post('/save', saveToDo);
router.post("/update", updateToDo); // Alok Kumar, 3 months ago * Add files via upload
router.post("/delete", deleteToDo);
module.exports = router;
```

The code editor includes status bars at the bottom showing "Alok Kumar, 3 months ago", "Ln 13, Col 29", "Spaces: 3", "UTF-8", "CRLF", "JavaScript", "Go Live", "Prettier", and system icons like battery level and network.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 23/09/23

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows the project structure under "TODO-BACKEND". Files include Server.js, ToDoModel.js, ToDoController.js, README.md, ToDoRoute.js, package-lock.json, package.json, .env, and .gitignore.
- Code Editor:** Displays the contents of ToDoController.js. The code handles HTTP requests for getting, saving, updating, and deleting todos from a MongoDB database using the mongoose library.
- Bottom Status Bar:** Shows file path (Todo-backend), author (Alok Kumar, 3 months ago), line (Ln 14, Col 21), and character (Spaces: 4, UTF-8, CRLF).
- System Tray:** Shows weather (26°C Cloudy), system icons, and a clock (23:29 27-09-2023).

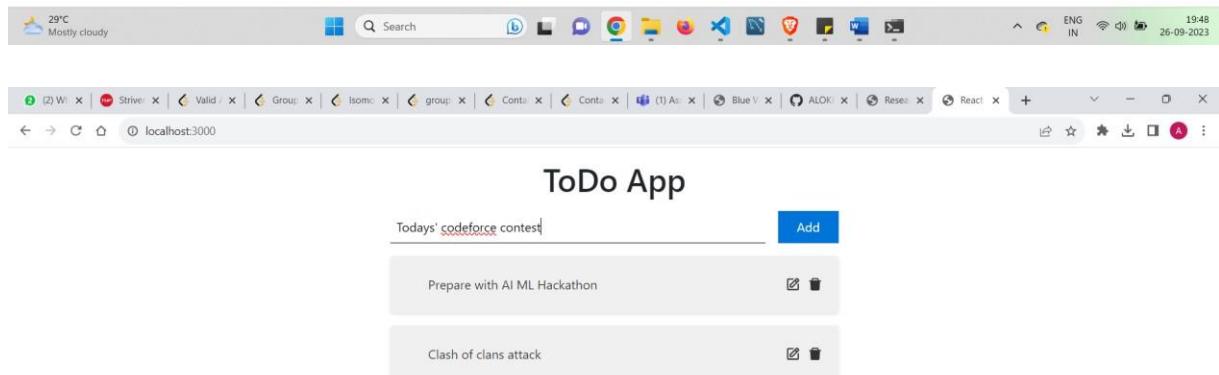
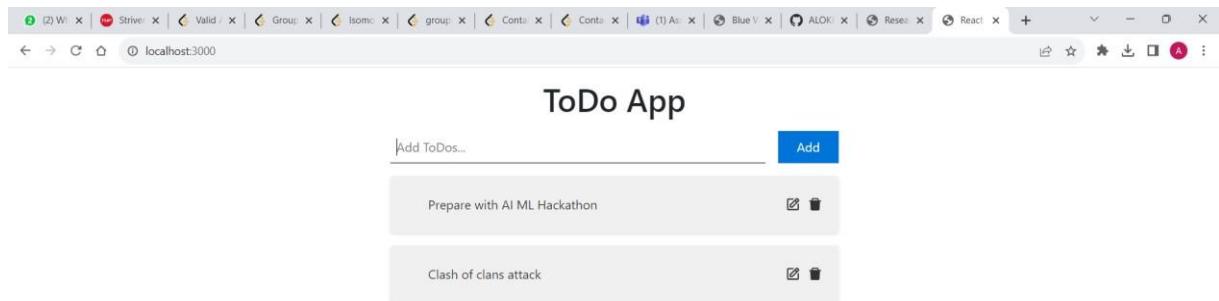
The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows the project structure under "TODO-BACKEND". Files include Server.js, ToDoModel.js, ToDoController.js, README.md, ToDoRoute.js, package-lock.json, package.json, .env, and .gitignore.
- Code Editor:** Displays the contents of ToDoModel.js. It defines a mongoose schema for a "ToDo" model with a "text" field.
- Bottom Status Bar:** Shows file path (Todo-backend), author (Alok Kumar, 3 months ago), line (Ln 10, Col 52), and character (Spaces: 4, UTF-8, CRLF).
- System Tray:** Shows weather (26°C Cloudy), system icons, and a clock (23:29 27-09-2023).

Name : Altaf Alam

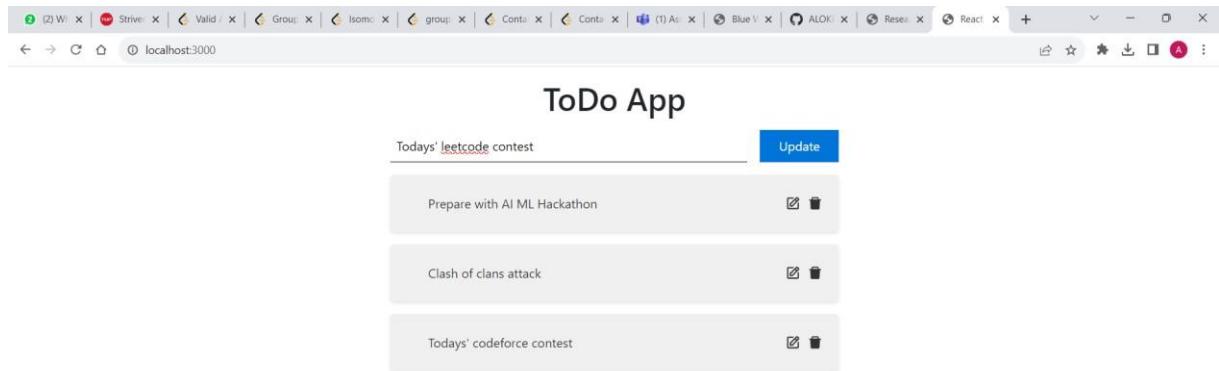
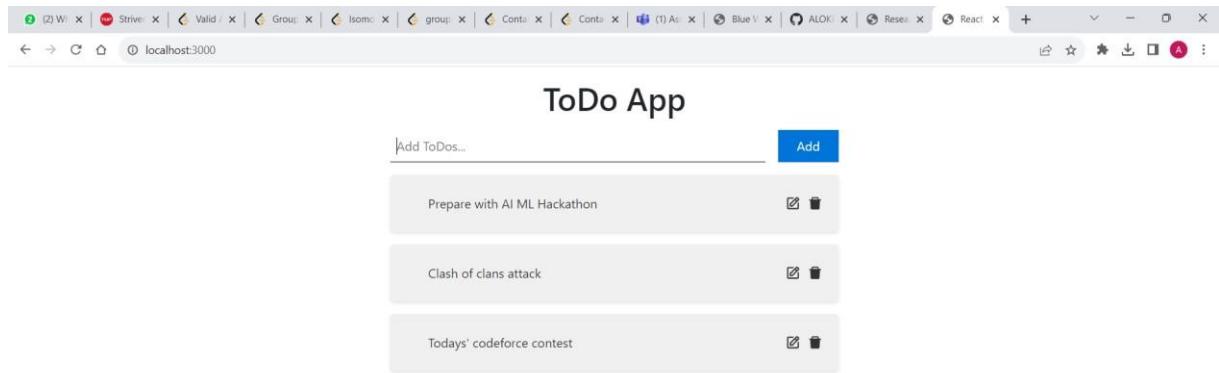
Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 23/09/23

### Output :



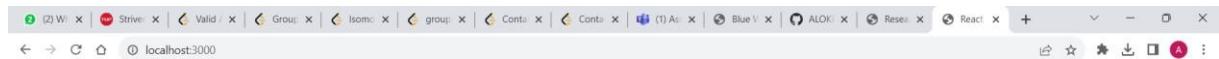
Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 23/09/23



Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 23/09/23



## ToDo App

Add ToDos...

Add

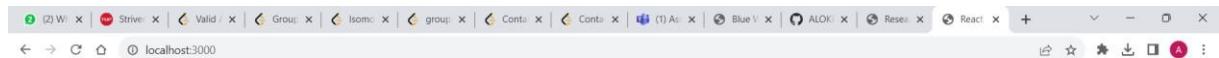
Prepare with AI ML Hackathon



Clash of clans attack



Todays' leetcode contest



## ToDo App

Add ToDos...

Add

Prepare with AI ML Hackathon



Todays' leetcode contest



## Conclusion :

In conclusion, we learnt to make a fullstack application using react in the frontend, backend with expressjs and nodejs and database with mongodb. We learned to integrate the frontend with backend and create various apis and connect to database.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP , Date : 10/08/23

## **Written Assignment No 1**

### **Compare XML and JSON.**

Comparing XML and JSON: Structured Data Representation

XML (eXtensible Markup Language) and JSON (JavaScript Object Notation) are both widely used formats for representing structured data in a human-readable and machine-readable manner. They serve as essential tools for data exchange and storage in various applications, including web services, APIs, configuration files, and more.

Similarities:

1. Human-Readable Format: Both XML and JSON are designed to be human-readable, making them easy to understand and edit by developers and users alike.
2. Hierarchical Structure: Both formats support a hierarchical structure, enabling the representation of complex data relationships and nested elements.
3. Platform-Independence: XML and JSON can be used across various programming languages and platforms, making them versatile choices for data interchange.
4. Support for Arrays/Collections: Both formats allow for the representation of arrays or collections of data elements.

Differences:

1. Syntax:

- XML: Uses a tag-based syntax with opening and closing tags, making it more verbose. Tags provide additional metadata and are enclosed in angle brackets (<>).
- JSON: Employs a lightweight syntax with key-value pairs. Data is represented in objects and arrays using curly braces ({} ) and square brackets ([]).

2. Readability:

- XML: Due to its tag-based structure, XML can be more verbose and less compact than JSON, leading to larger file sizes.
- JSON: JSON's compact syntax makes it more concise, resulting in smaller file sizes and improved network efficiency.

3. Data Types:

- XML: XML doesn't have built-in data types; data is treated as text by default. Additional attributes or schema definitions are required to specify data types.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP , Date : 10/08/23

- JSON: JSON inherently supports various data types, including strings, numbers, booleans, objects, arrays, and null values.

#### 4. Attributes:

- XML: Supports attributes that provide metadata or additional information for an element. Attributes are enclosed within an opening tag.
- JSON: Lacks the concept of attributes; all data is represented as key-value pairs within objects.

#### 5. Comments:

- XML: Supports comments between <!-- and --> markers.
- JSON: Doesn't natively support comments, which can make it challenging to annotate the data.

### Use Cases:

#### 1. XML:

- Used in configuration files for various software and applications.
- Commonly used in web services where a schema definition is required for validation.
- Suited for documents that require extensive metadata, such as legal contracts or scientific research.

#### 2. JSON:

- Preferred for APIs due to its concise syntax and ease of parsing in web applications.
- Widely used in modern web development for AJAX requests and data interchange.
- Ideal for representing complex data structures, making it a popular choice for NoSQL databases and document-oriented databases like MongoDB.

Both XML and JSON offer structured data representation, each with its strengths and weaknesses. XML's verbosity and metadata support make it suitable for certain use cases, while JSON's lightweight syntax and ease of use make it a preferred choice in many modern applications. Choosing between XML and JSON depends on factors like data complexity, use case requirements, and compatibility with existing systems.

### **Explain different types of arrow functions**

Arrow functions are a concise way to write functions in JavaScript. They were introduced in

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP , Date : 10/08/23

ECMAScript 6 (ES6) and offer a shorter syntax compared to traditional function expressions. Arrow functions have various forms, each with its own use cases. Different types of arrow functions with examples are:

### Basic Arrow Function:

The simplest form of an arrow function has a parameter list followed by an arrow ('=>') and an expression to be evaluated and returned. If there's only one parameter, parentheses around it can be omitted.

```

```
const add = (a, b) => a + b; console.log(add(5,  
3));  
// Output: 8  
```
```

### Arrow Function with Multiple Statements:

If you need to perform multiple statements inside an arrow function, you can use curly braces and specify the 'return' keyword explicitly.

```

```
const multiply = (x, y) =>  
{ let result = x * y;  
return result;  
}; console.log(multiply(4,  
6));  
// Output: 24  
```
```

### Arrow Function with No Parameters:

You can create an arrow function with no parameters by providing an empty set of parentheses.

```

```
const greet = () => "Hello, world!";  
console.log(greet()); // Output:  
Hello, world!  
```
```

### Arrow Functions with Object literals:

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP , Date : 10/08/23

When returning an object literal from an arrow function, wrap the object in parentheses to avoid confusion with a function body.

```

```
const createPerson = (name, age) => ({ name, age });
console.log(createPerson("Alice", 30));
// Output: { name: 'Alice', age: 30 } ````
```

Arrow Function with Rest Parameters:

You can use the rest parameter syntax to capture an arbitrary number of arguments into an array.

```

```
const sumAll = (...numbers) => numbers.reduce((acc, val) => acc + val, 0);
console.log(sumAll(1, 2, 3, 4, 5));
// Output: 15
````
```

Arrow Function with Default Parameters: Arrow functions can also use default parameters to provide fallback values.

```

```
const greetUser = (name = "Guest") => `Hello,
${name}!`; console.log(greetUser()); // Output: Hello,
Guest!
console.log(greetUser("Alice")); //
Output: Hello, Alice!
````
```

Arrow Functions and `this` Binding:

One significant difference between arrow functions and traditional functions is the behavior of the `this` keyword. Arrow functions lexically bind `this`, which means they inherit the `this` value from their containing scope.

```

```
function Counter() {
this.count = 0;
setInterval(() => {
this.count++;
console.log(this.count);
```

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP , Date : 10/08/23

```
 }, 1000); }
const counter = new Counter();
...
```

In this example, the arrow function preserves the 'this' context of the 'Counter' object, allowing you to access and modify the 'count' property.

## **What is DNS? Explain working of DNS.**

DNS (Domain Name System): Navigating the Web with Names.

The Domain Name System (DNS) is a critical component of the Internet infrastructure that translates human-readable domain names into machine-readable IP addresses. It acts as a distributed and hierarchical database, serving as the "phone book" of the Internet. DNS plays a fundamental role in enabling users to access websites, send emails, and connect to online services using easy-to-remember domain names instead of numeric IP addresses.

Working of DNS:

### **1. Name Resolution Request:**

When a user enters a domain name (e.g., www.example.com) into their web browser, the browser initiates a DNS query to resolve the domain name to its corresponding IP address. This query is sent to the user's configured DNS resolver, which can be the ISP's DNS server, a public DNS service like Google DNS, or even a local caching DNS server.

### **2. Caching and Recursion:**

The DNS resolver first checks its local cache to see if it already has the IP address for the requested domain. If not, it performs a recursive query. It starts by querying the root DNS servers, which point to the top-level domain (TLD) DNS servers (like .com, .org). The TLD servers then direct the resolver to the authoritative DNS server for the specific domain (e.g., example.com).

### **3. Authoritative DNS Server:**

The authoritative DNS server for the domain stores the domain's DNS records, including information like IP addresses, mail server addresses, and more. The resolver queries this authoritative server for the IP address corresponding to the requested domain.

### **4. Response and Caching:**

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP , Date : 10/08/23

Once the authoritative server responds with the IP address, the resolver caches this information locally for a specified time (Time-To-Live or TTL). This caching reduces the need to repeatedly query authoritative servers for the same domain, improving efficiency.

### 5. Data Transmission:

With the resolved IP address, the browser can initiate a connection to the appropriate web server hosting the requested content. This connection allows users to access the website associated with the domain name.

### Benefits and Importance:

1. User-Friendly Experience: DNS enables users to access websites, services, and resources using easy-to-remember domain names instead of complex IP addresses.
2. Scalability: DNS is a distributed system, ensuring efficient and reliable resolution for the billions of domain names on the Internet.
3. Redundancy and Load Balancing: Multiple DNS servers exist for each domain, providing redundancy. Load balancing can distribute user requests across various servers for improved performance.
4. Dynamic IP Addresses: DNS allows dynamic IP addresses to be associated with a domain name, making it possible to host websites and services on changing IP addresses.
5. Global Accessibility: DNS ensures worldwide access to websites and services, regardless of users' geographic locations.
6. Security Considerations: DNS also plays a role in security by implementing techniques like Domain Name System Security Extensions (DNSSEC) to prevent DNS-related attacks like DNS spoofing.

### Explain Promises in ES6.

#### Promises in ES6: Managing Asynchronous Operations with Confidence

Promises were introduced in ECMAScript 6 (ES6) as a powerful way to handle asynchronous operations in JavaScript. They provide a structured and more readable approach to managing asynchronous tasks, such as fetching data from APIs, reading files, or performing network requests. Promises play a crucial role in addressing the challenges of

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP , Date : 10/08/23

callback hell and improving code maintainability. Let's delve into the concept of promises and how they work.

### Understanding Promises:

A promise is an object that represents a value that might be available now, or in the future, or not at all. It's a placeholder for the result of an asynchronous operation. Promises have three states: 'pending', 'fulfilled', and 'rejected'.

### Creating a Promise:

The 'Promise' constructor takes a single argument, a function (often referred to as the "executor"), which is called asynchronously when the promise is created. This function has two parameters: 'resolve' and 'reject'. Inside the executor, you perform the asynchronous task, and when it's done, you call 'resolve' with the result or 'reject' with an error.

```

```
const fetchData = new Promise((resolve, reject) => {
  fetch('https://api.example.com/data')
    .then(response => response.json())
    .then(data => resolve(data))
    .catch(error => reject(error));
});
```

Chaining Promises:

One of the most significant benefits of promises is the ability to chain multiple asynchronous operations together using '.then()' and handle errors with '.catch()'. This pattern improves code readability and avoids callback hell.

```

```
fetchData
 .then(data => process(data))
 .then(result => display(result))
 .catch(error => handleError(error));
```
```

Handling Multiple Promises:

ES6 introduced the 'Promise.all()' method, which takes an array of promises and returns a new promise that resolves when all the input promises resolve, or rejects when any of them reject.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP , Date : 10/08/23

...

```
const promise1 = fetch(url1); const  
promise2 = fetch(url2);  
  
Promise.all([promise1, promise2])  
.then(results => processResults(results))  
.catch(error => handleErrors(error));  
...
```

Async/Await with Promises:

ES2017 introduced the `async` and `await` keywords, which provide a more synchronous-like syntax for handling promises. The `async` keyword is used to define an asynchronous function, while `await` is used inside that function to pause execution until a promise is resolved.

...

```
async function fetchData() {  
  try {  
    const response = await  
    fetch('https://api.example.com/data');  
    const data = await  
    response.json();  
    return data;  
  } catch (error) {  
    throw  
    error;  
  }  
}
```

Benefits of Promises:

1. Readability: Promises improve code readability by eliminating nested callbacks and providing a more linear flow.
2. Error Handling: Promises provide a structured way to handle errors using `catch()`, making error management more organized.
3. Chaining: Chaining promises simplifies handling sequential asynchronous tasks, making code easier to understand and maintain.
4. Parallelism: `Promise.all()` allows parallel execution of multiple promises, enhancing performance when dealing with independent tasks.

Promises are a fundamental addition to JavaScript that revolutionized how developers handle asynchronous operations. By providing a structured way to manage and chain asynchronous

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP , Date : 10/08/23

tasks, promises enhance code readability, maintainability, and error handling. The introduction of `async` and `await` in ES2017 further improved the syntax, making asynchronous programming in JavaScript more intuitive and manageable.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 22/09/23

Assignment No 2

Aim :

1. What are Refs? When to use Refs and when not to use refs.
2. Write short note on
 - a. NPM
 - b. REPL
3. Explain Routing in ExpressJS along with an example

Theory :

1: Refs in JavaScript

Refs in JavaScript, short for "references," are a mechanism for accessing and interacting with the DOM (Document Object Model) elements in a web page. They are primarily used in React, a popular JavaScript library for building user interfaces, although the concept of refs is not limited to React and can be found in vanilla JavaScript as well.

When to Use Refs

Refs are typically used in the following scenarios:

1. Accessing DOM Elements: When you need to access a specific DOM element directly. This is often necessary for tasks like setting focus on an input field, reading the dimensions of an element, or manually triggering events.
2. Managing Third-party Libraries: Integrating third-party libraries or plugins that don't play well with React's virtual DOM can require the use of refs to access and manipulate their elements.
3. Animations: When you need to animate an element or apply animations using libraries like GreenSock Animation Platform (GSAP), you may need to use refs to target the DOM element to animate.
4. Managing Uncontrolled Components: In some cases, you may need to work with uncontrolled components (e.g., HTML input elements) where React state is not directly linked. Refs allow you to interact with these components without relying on React's state management.

When Not to Use Refs

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 22/09/23

While refs are a powerful tool, they should be used sparingly, and there are situations where they should be avoided:

1. Overusing Refs: Avoid using refs for common tasks that can be managed through React's state and props. Overusing refs can make your code harder to maintain and debug.
2. Breaking Component Isolation: Refs can break the encapsulation and isolation of React components. It's often better to pass data and callbacks as props instead of using refs to directly manipulate child components.
3. Avoiding Functional Components: With the introduction of React Hooks, functional components provide a more modern and concise way to manage component state and side effects. In functional components, you can use the `useRef` hook instead of the older `createRef` method.

2: NPM (Node Package Manager)

NPM, or the Node Package Manager, is a package manager for JavaScript and Node.js. It is used for installing, managing, and sharing JavaScript libraries and tools. NPM is included with Node.js by default, making it an essential tool for Node.js development.

Key Features of NPM

1. Package Installation: NPM allows developers to install packages and libraries from the NPM registry. You can use the `npm install` command followed by the package name to add dependencies to your project.
2. Dependency Management: NPM manages project dependencies by maintaining a `package.json` file, which lists all the packages your project relies on, along with their versions. This makes it easy to share projects and ensure consistent dependencies across development teams.
3. Scripts: NPM provides a powerful scripting system defined in the `package.json` file. You can define custom scripts for tasks like building, testing, or running your application. These scripts can be executed using the `npm run` command.
4. Version Control: NPM allows you to specify version ranges for your project dependencies. This ensures that your project can easily update to the latest compatible versions of packages while maintaining stability.

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 22/09/23

5. Global Packages: Some packages can be installed globally using the '-g' flag. These packages provide command-line utilities that can be used across multiple projects.

Basic NPM Commands

Here are some essential NPM commands:

- `npm init`: Initializes a new Node.js project by creating a `package.json` file.
- `npm install <package-name>`: Installs a package locally in the current project.
- `npm install -g <package-name>`: Installs a package globally.
- `npm uninstall <package-name>`: Uninstalls a package from the current project.
- `npm list`: Lists installed packages.
- `npm update <package-name>`: Updates a package to the latest compatible version.
- `npm search <package-name>`: Searches for packages on the NPM registry.
- `npm run <script-name>`: Executes a custom script defined in `package.json`.

3: REPL (Read-Eval-Print Loop)

REPL stands for Read-Eval-Print Loop and is an interactive programming environment available in various programming languages, including JavaScript. It allows developers to enter code, have it evaluated, and see the results immediately. In the context of Node.js, the Node REPL provides a way to interact with the Node.js runtime.

How to Use the Node.js REPL

To start the Node.js REPL, open your terminal and type `node`. This will launch the REPL, and you can begin entering JavaScript code interactively. Here are some basic commands and features of the Node.js REPL:

- Entering Code: Simply type JavaScript code and press 'Enter' to execute it. The result will be displayed immediately.
- Variables: You can declare and work with variables within the REPL.

```
```javascript>
let x = 10;
undefined
> x + 5
```

15

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 22/09/23

...

- Multiline Input: For multiline statements or code blocks, you can use the `...` prompt to continue entering code.

```
```javascript
> function add(a, b) {
...   return a + b;
...
undefined >
add(3, 4)
7
````
```

- Previous Results: You can access the results of previous expressions using `\_\_\_. For example, `\_\_` will contain the result of the last expression evaluated.

```
```javascript
> 2 + 3
5
> __^ 2
10
````
```

- Special Commands: The REPL provides special commands prefixed with a period (`.`) that can be used for various purposes, such as `help` for help, `exit` to exit the REPL, and `save` to save the REPL session to a file.

- Tab Completion: You can use tab completion to explore available objects, properties, and methods. Pressing `Tab` will suggest completions based on what you've typed.

```
```javascript
> const fs = require('fs');
undefined
> fs.
...
````
```

## 4: Routing in Express.js

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 22/09/23

## Express.js Routing Basics

Express.js is a powerful web application framework for Node.js, simplifying web app development with its robust routing system. Here, we'll cover essential routing concepts and building a basic API.

## Setting Up Express.js

1. Install Express: Use `npm install express` to add Express as a dependency.
  2. Create an Express App:

```
```javascript
const express = require('express');
const app = express();
const port = 3000;

app.listen(port, () => {
  console.log(`Server is
running on port ${port}`);
});

```

```

## Basic Routing

Example of a route for handling a GET request at the root URL ('/'):

```
```javascript    app.get('/',  
(req,      res)  =>  {  
res.send('Hello, World!');  
});  
```
```

## Route Parameters

Define routes with parameters to capture values from the URL:

```
```javascript app.get('/users/:id',  
(req, res) => { const userId =
```

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 22/09/23

```
req.params.id; res.send(`User  
ID: ${userId}`);  
});  
```
```

## Handling POST Requests

Handle POST requests with Express:

```
```javascript  
app.post('/users', (req, res) => {  
  const newUser = req.body; //  
  Handle user creation  
  res.status(201).json(newUser);  
});  
```
```

## Route Middleware

Middleware functions perform actions before or after route handlers. Log requests using middleware:

```
```javascript  
function logRequest(req, res, next) {  
  console.log(`Received ${req.method} request at ${req.url}`);  
  next();  
}  
  
app.use(logRequest);  
```
```

## Error Handling

Define error-handling middleware for error responses:

```
```javascript app.use((err, req, res, next) => {  
  console.error(err.stack);  
  res.status(500).send('Something went wrong!');  
})
```

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 22/09/23

});

...

Building a Simple API

Here's a concise example of an API managing a list of books:

```
```javascript
const express = require('express');
const app = express();
const port = 3000;

app.use(express.json());

const books = [
 { id: 1, title: 'Book 1' },
 { id: 2, title: 'Book 2' },
];

app.get('/books', (req, res) => {
 res.json(books);
});

app.post('/books', (req, res) => {
 const newBook = req.body;
 books.push(newBook);
 res.status(201).json(newBook);
});

app.get('/books/:id', (req, res) => {
 const bookId = parseInt(req.params.id);
 const book = books.find((b) => b.id === bookId);
 book ? res.json(book) : res.status(404).send('Book not found');
});

app.listen(port, () => {
 console.log(`Server is running on port ${port}`);
});
```

Name : Altaf Alam

Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 22/09/23

\*\*\*