Thadomal Shahani Engineering College

Bandra (W.), Mumbai- 400 050.

CERTIFICATE

Certify that Mr./~~Miss~~ Prasad Sunil Arote

of IT Department, Semester V with

Roll No. 06 has completed a course of the necessary

experiments in the subject Internet Programming under my

supervision in the **Thadomal Shahani Engineering College**

Laboratory in the year 2023 - 2024

Teacher In-Charge

Head of the Department

Date 13/10/23

Principal

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

CONTENTS

SR. NO.	EXPERIMENTS	PAGE NO. Date	DATE Pg No	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	2/8/23		Samoked 13/10/23
4	(a) WAP in JS to study loops and functions	9/8/23		
	(b) WAP on inheritence, iterators & generator	9/8/23		
5	(a) WAP to study arrow function DOM and CSS manipulation	21/8/23		
	(b) Write a JS program to implement promise, fetch & asynchronous JS	21/8/23		
6.	(a) WAP to implement prop & state	23/8/23		Samoked 13/10/23
	(b) WAP to implement forms & events	23/8/23		
7)	(a) WAP to implement React JS router and animation	6/9/23		
	(b) WAP to implement React Hooks	13/9/23		
8)	REPL (command line)	27/9/23		
9)	WAP to implement React Ref	13/10/23		
10)	WAP in NodeJS to create a file, read and write data, rename the file and delete	27/9/23		
11)	Create, a web application that perform CRUD operations (data base connectivity)	13/10/23		Samoked 13/10/23
12)	Theory Assignment 1	10/8/23		
13)	Theory Assignment 2	23/9/23		

Prasad Arote T11 06

Assignment 01

Aim: Design a web application by using HTML Tags, Elements, Attributes, head, Body, Hyperlinks, Formatting Images, Tables, Lists, Frames, Forms, and Multimedia elements should be used.

LO1: To orient students to HTML for making webpages.

Theory:

Introduction to HTML: HTML, which stands for Hypertext Markup Language, is the standard markup language used to create and structure the content of websites.

1. **Basic Structure:** HTML documents consist of elements represented by tags, which are enclosed in angle brackets ("<" and ">"). The basic structure includes the <html>, <head>, and <body> tags.

2**. Document Type Declaration:** The <!DOCTYPE> declaration at the beginning of an HTML document defines its version and type, ensuring proper rendering in web browsers.

3. **Head Section:** The <head> section contains meta-information about the HTML document, such as the title, character encoding, and links to external resources like CSS and JavaScript files. <title></title>: This element sets the title of the web page, which is displayed on the browser's title bar or tab.

4. **Body:** <body> This element contains the visible content of the web page.

5. **Text Formatting:** HTML provides various tags for text formatting, including headings (<h1> to <h6>), paragraphs (<p>), emphasis (and), line breaks (
), and horizontal rules (<hr>).mark>: This tag highlights the text with a background color, usually yellow, to indicate that it has been marked or selected.

<del>: Similar to the <s> tag, <del> is used to indicate deleted or removed text, which is usually displayed with a strike-through.

<ins>: This tag is used to indicate inserted or added text, which is usually underlined.

6**. Hyperlinks**: Hyperlinks are essential in HTML, allowing users to navigate between pages and resources on the internet. They are created using the <a> (anchor) tag, with the "href" attribute specifying the URL.

7. **Lists:** HTML supports ordered lists (<ol>), unordered lists (<ul>), and definition lists (<dl>) to organize content in a structured manner.

8**. Table:** <table>: Defines a table for organizing tabular data, and rows are represented by <tr> tags.

<tr>: Represents a table row, and table cells (data or header) are represented by <td> or <th> tags, respectively.

<th>: Represents a table header cell.

<td>: Represents a table data cell.

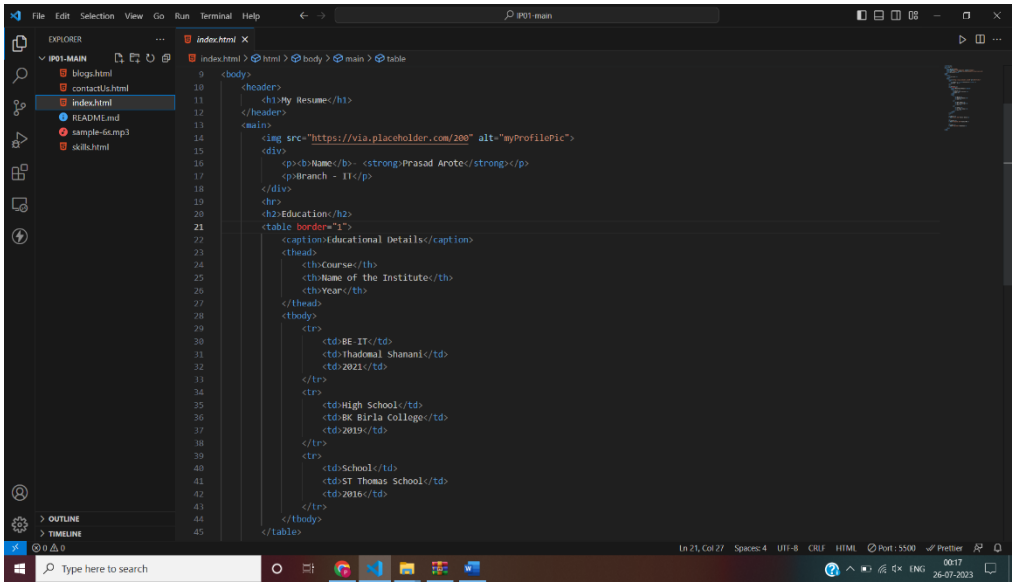
<caption>: Provides a caption or title for a table.

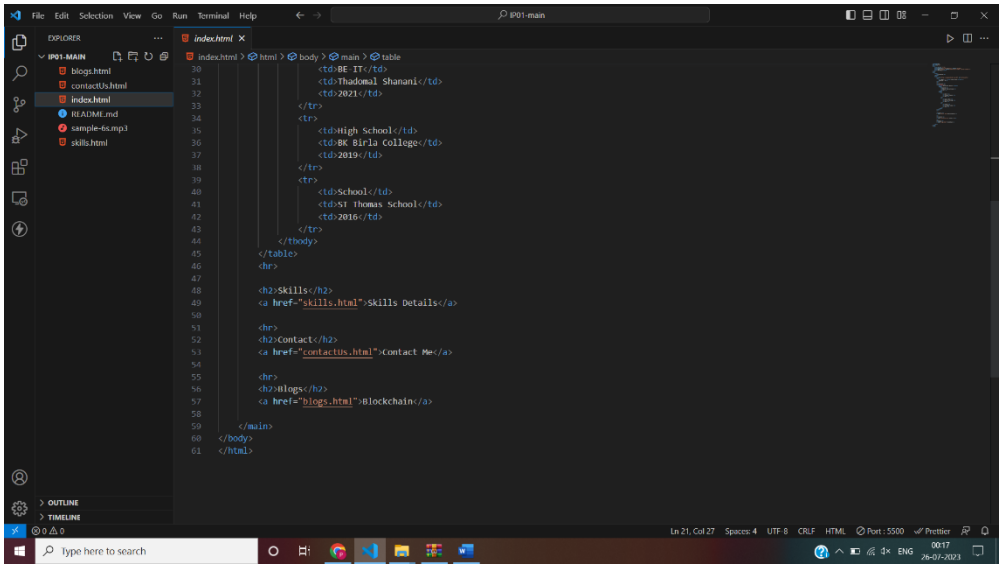
9. **Semantic Tags**: <header>, <main>, <footer>, <nav>, <section>, <article>: Semantic elements that help structure the web page and provide meaning to different sections. <article>: Represents a self-contained piece of content that could be distributed independently and potentially reused in other contexts. <footer>: Represents the footer section of the web page, typically containing copyright information, contact details, or links to related resources.

10. **Multimedia Tags:** <iframe>: Embeds an inline frame to display another web page within the current page.

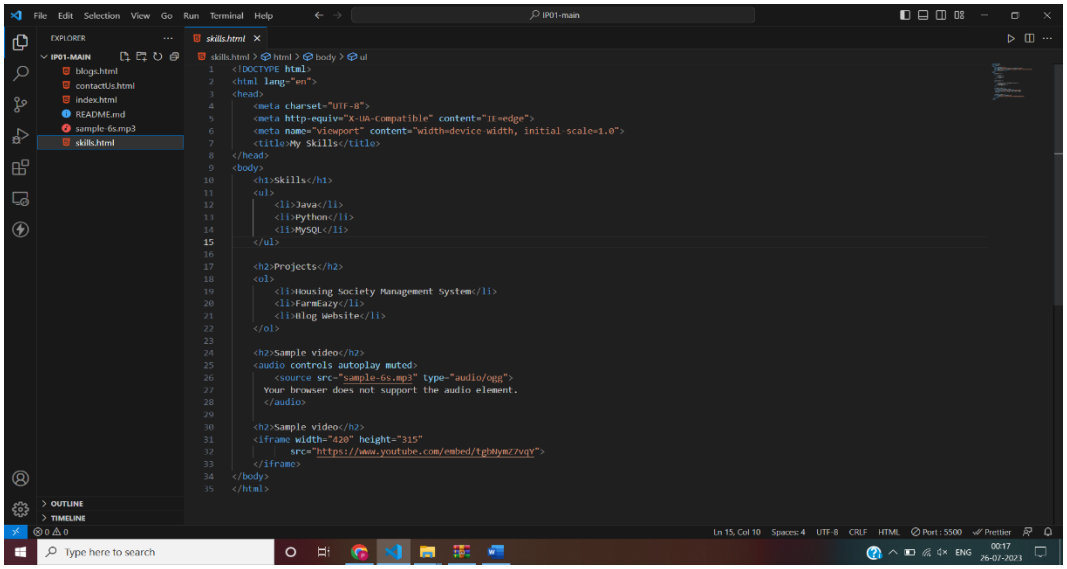
<audio>: Used to embed audio files into a web page. It supports various audio formats and can include controls to play, pause, and adjust the audio.

<video>: Used to embed video files into a web page. It supports various video formats and can include controls to play, pause, and adjust the video.

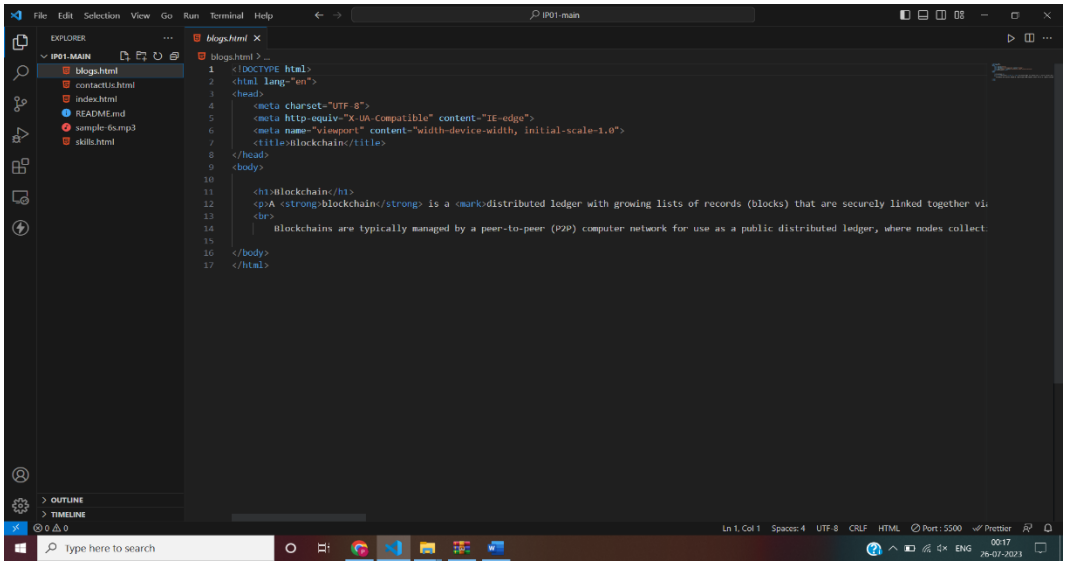




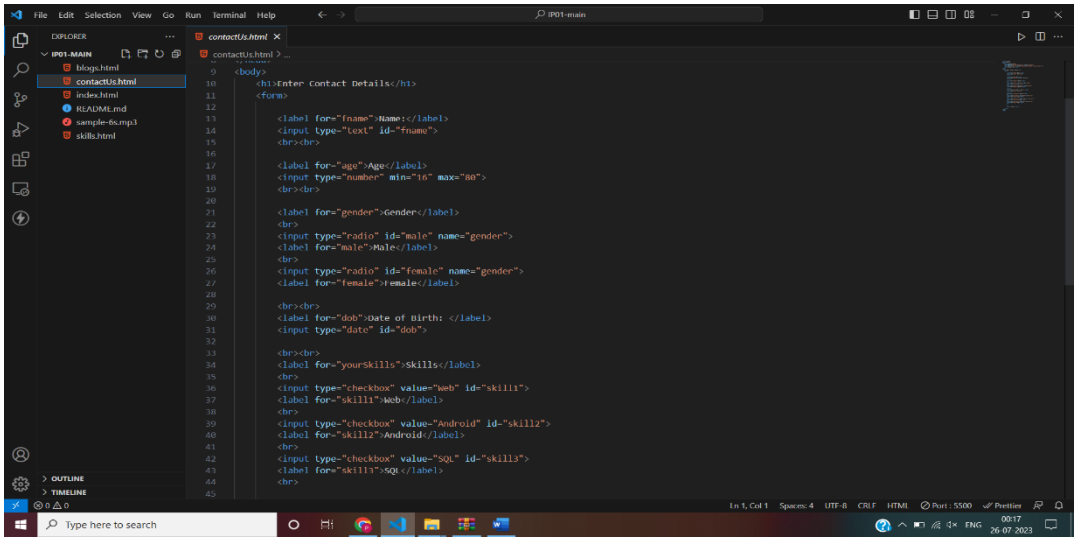
```
<td>BE-IT</td>
<td>Thadomal Shanani</td>
<td>2021</td>
</tr>
<tr>
<td>High School</td>
<td>BK Birla College</td>
<td>2019</td>
</tr>
<tr>
<td>School</td>
<td>St Thomas School</td>
<td>2016</td>
</tr>
</tbody>
</table>
<hr>

<h2>Skills</h2>
<a href="skills.html">Skills Details</a>

<hr>
<h2>Contact</h2>
<a href="contactUs.html">Contact Me</a>

<hr>
<h2>Blogs</h2>
<a href="blogs.html">Blockchain</a>

</main>
</body>
</html>
```



```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>My Skills</title>
</head>
<body>
<h1>Skills</h1>
<ul>
<li>Java</li>
<li>Python</li>
<li>MySQL</li>
</ul>

<h2>Projects</h2>
<ol>
<li>Housing Society Management System</li>
<li>FarmEazy</li>
<li>Blog Website</li>
</ol>

<h2>Sample video</h2>
<audio controls autoplay muted>
<source src="sample-6s.mp3" type="audio/ogg">
Your browser does not support the audio element.
</audio>

<h2>Sample video</h2>
<iframe width="420" height="315"
src="https://www.youtube.com/embed/tpbNymZ7vqY">
</iframe>
</body>
</html>
```



```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Blockchain</title>
</head>
<body>

<h1>Blockchain</h1>
<p>A <strong>blockchain</strong> is a <mark>distributed ledger with growing lists of records (blocks) that are securely linked together vi
<br>
Blockchains are typically managed by a peer-to-peer (P2P) computer network for use as a public distributed ledger, where nodes collect

</body>
</html>
```


File Edit Selection View Go Run Terminal Help
IP01-main
EXPLORER
IP01-MAIN
blogs.html
contactUs.html
index.html
README.md
sample-6s.mp3
skills.html
OUTLINE
TIMELINE
contactUs.html
<body>
<h1>Enter Contact Details</h1>
<form>
<label for="fname">Name:</label>
<input type="text" id="fname">

<label for="age">Age</label>
<input type="number" min="16" max="80">

<label for="gender">Gender</label>

<input type="radio" id="male" name="gender">
<label for="male">Male</label>

<input type="radio" id="female" name="gender">
<label for="female">Female</label>

<label for="dob">Date of Birth: </label>
<input type="date" id="dob">

<label for="yourSkills">Skills</label>

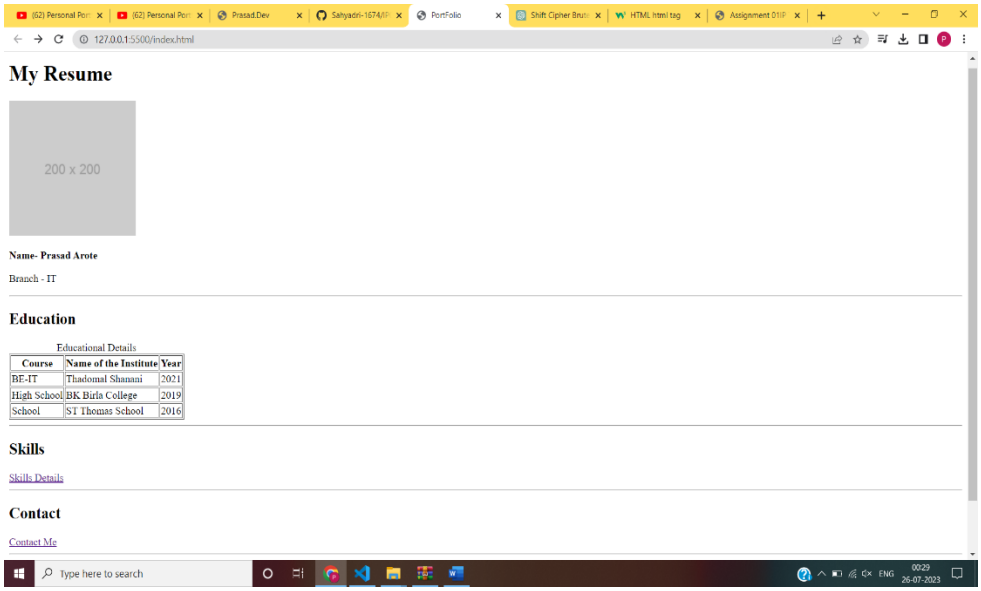
<input type="checkbox" value="Web" id="skill1">
<label for="skill1">Web</label>

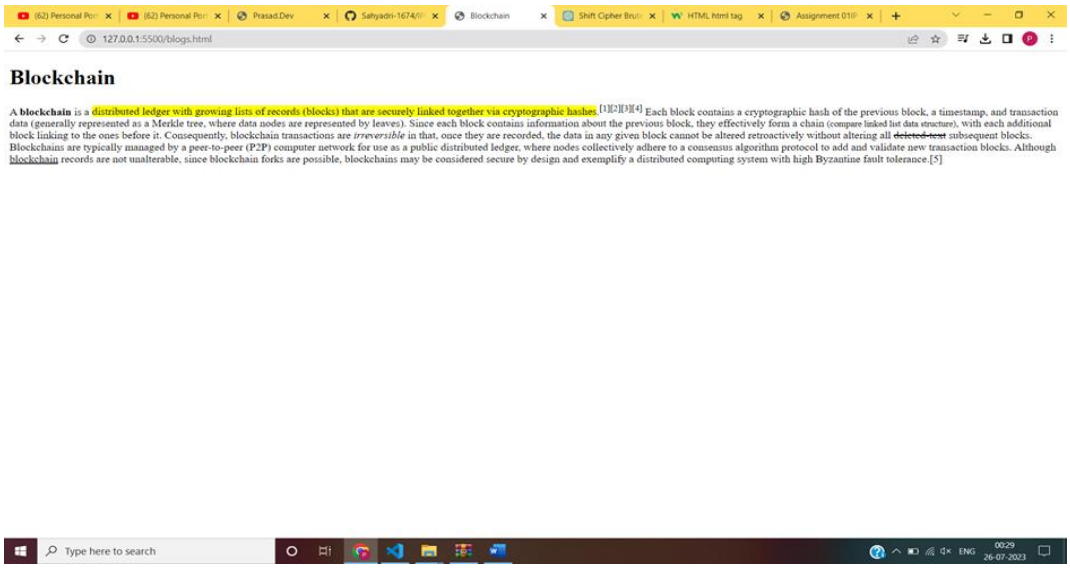
<input type="checkbox" value="Android" id="skill2">
<label for="skill2">Android</label>

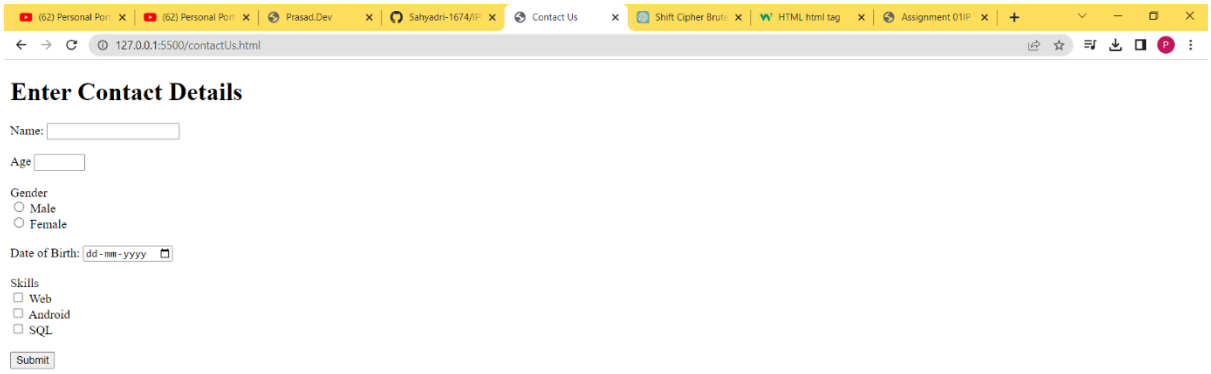
<input type="checkbox" value="SQL" id="skill3">
<label for="skill3">SQL</label>

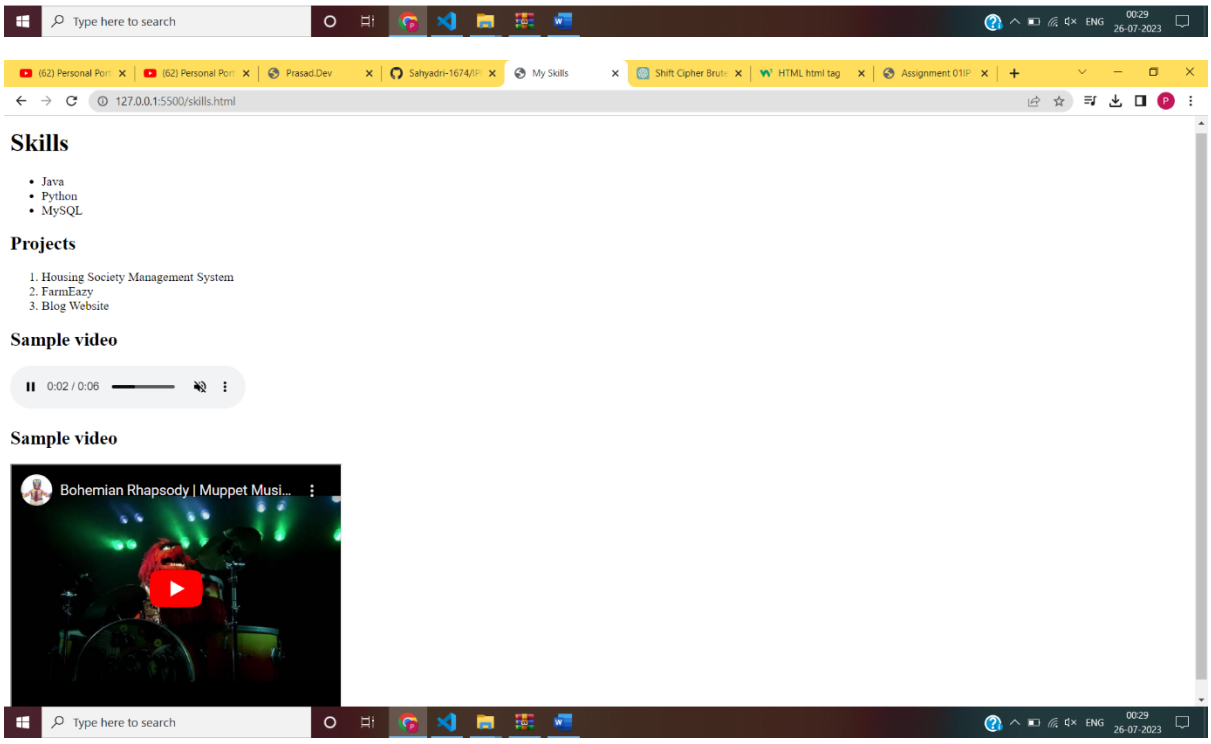
Ln 1, Col 1 Spaces: 4 UTF-8 CRLF HTML Port : 5500 Prettier
Type here to search
ENG
00:17
26-07-2023

Output:


127.0.0.1:5500/index.html
My Resume
200 x 200
Name- Prasad Arote
Branch - IT
Education
Educational Details
Course | Name of the Institute | Year
BE-IT | Thadomal Shanani | 2021
High School | BK Birla College | 2019
School | ST Thomas School | 2016
Skills
Skills Details
Contact
Contact Me
Type here to search
ENG
00:29
26-07-2023


127.0.0.1:5500/blogs.html
Blockchain
A blockchain is a distributed ledger with growing lists of records (blocks) that are securely linked together via cryptographic hashes.[1][2][3][4] Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data (generally represented as a Merkle tree, where data nodes are represented by leaves). Since each block contains information about the previous block, they effectively form a chain (compare linked list data structure), with each additional block linking to the ones before it. Consequently, blockchain transactions are irreversible in that, once they are recorded, the data in any given block cannot be altered retroactively without altering all ~~deleted text~~ subsequent blocks. Blockchains are typically managed by a peer-to-peer (P2P) computer network for use as a public distributed ledger, where nodes collectively adhere to a consensus algorithm protocol to add and validate new transaction blocks. Although blockchain records are not unalterable, since blockchain forks are possible, blockchains may be considered secure by design and exemplify a distributed computing system with high Byzantine fault tolerance.[5]
Type here to search
ENG
00:29
26-07-2023





CONCLUSION:

When designing web pages using only HTML, several challenges arise. HTML's limited styling options hinder achieving complex visual layouts and responsiveness. Interactivity suffers as JavaScript is absent, leaving out dynamic elements and real-time updates. Code maintenance becomes cumbersome due to possible duplication, while cross browser compatibility issues may arise. Functionality is constrained, as complex tasks like user authentication or data processing require additional technologies. Accessibility and SEO efforts can also be hampered. Overall, HTML alone may lead to suboptimal user experiences, limited data persistence, and difficulty in creating modern, engaging web applications. Combining HTML with CSS and JavaScript addresses these challenges, enabling developers to create more feature-rich and user-friendly websites.

Assignment 02

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.

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

Theory:

1. Three Ways to Insert CSS
    a) Inline: Inline CSS is added directly to an HTML element using the style attribute. It affects only that specific element.
    b) Internal: Internal CSS is placed within the <style> tags in the <head> section of an HTML document. It affects elements throughout the document.
    c) External: External CSS is placed in a separate .css file and linked to the HTML document using the <link> tag in the <head> section. It allows for better separation of concerns and reusability of styles across multiple pages.

2. CSS Font Properties
    a) font-family: Specifies the font(s) to be used for text. It can be a list of font names or a generic font family like "sans-serif", "serif", or "monospace".
    b) font-size: Sets the size of the text. It can be defined in pixels, em units, percentages, etc.
    c) font-weight: Sets the thickness or boldness of the text. It can have values like "normal", "bold", "lighter", or numeric values like 100, 200, ..., 900.
    d) font-style: Specifies the style of the font, like "normal", "italic", or "oblique".
    e) color: Sets the color of the text.

3. CSS Text Properties
    a) text-align: Aligns the text horizontally within its container.
    b) text-decoration: Adds visual effects to text, such as underline, overline, or line-through.
    c) text-transform: Changes the capitalization of the text.
    d) line-height: Defines the spacing between lines of text.
4. Table Properties
    a) border-collapse: Specifies whether table borders should be collapsed into a single border or separated.
    b) border: Sets the border properties for table elements, including width, style, and color.
    c) padding: Sets the space between the content and the border inside table cells.
    d) text-align: Aligns the content of table cells horizontally.
    e) vertical-align: Aligns the content of table cells vertically.
    f) width: Sets the width of the table or table cells.

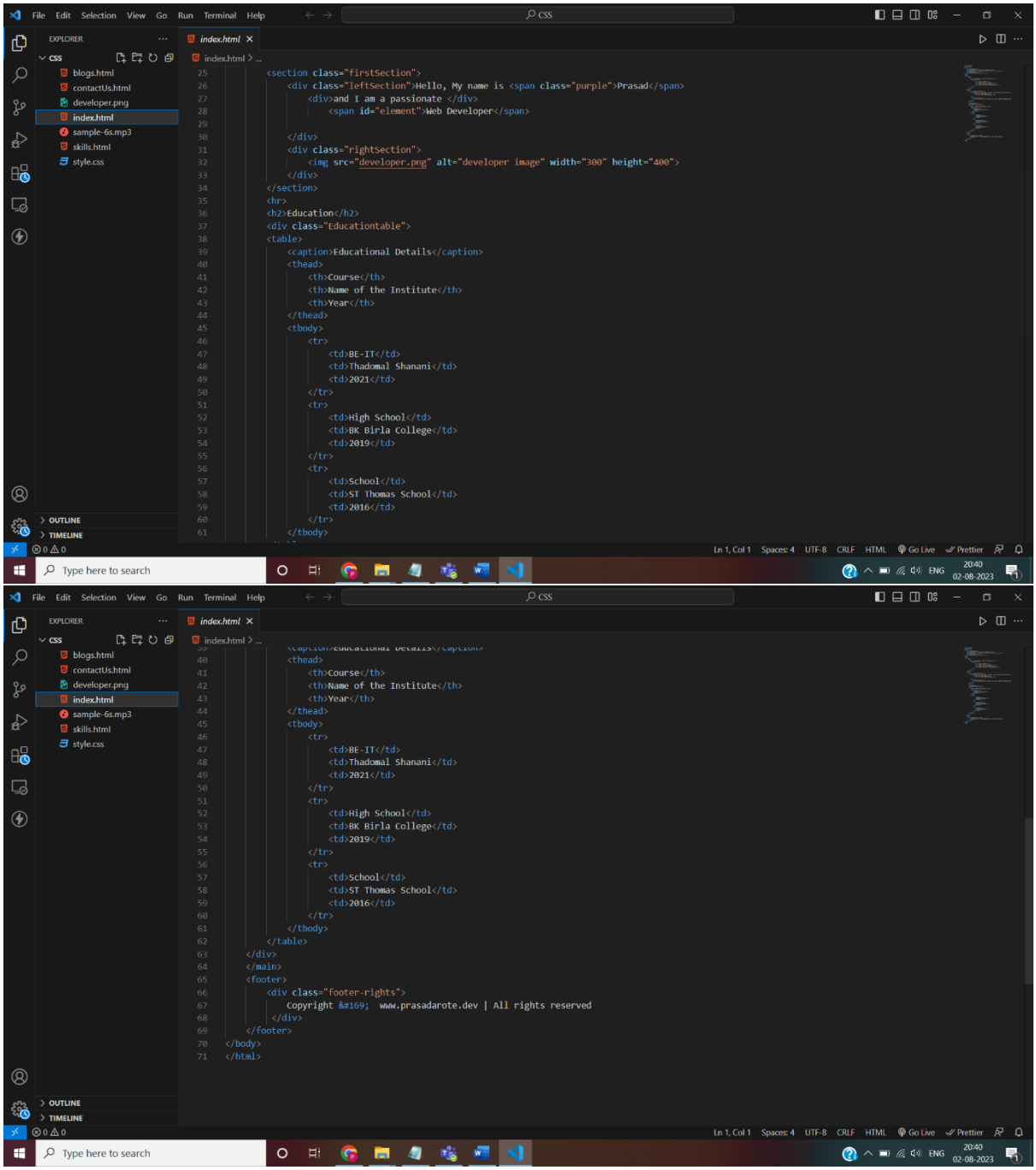
5. List Properties
    a) list-style-type: Specifies the style of the list item marker (bullet, number, etc.).
    b) list-style-image: Sets an image as the list item marker.
    c) list-style-position: Determines whether the list item marker appears inside or outside the content flow.
    d) list-style: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

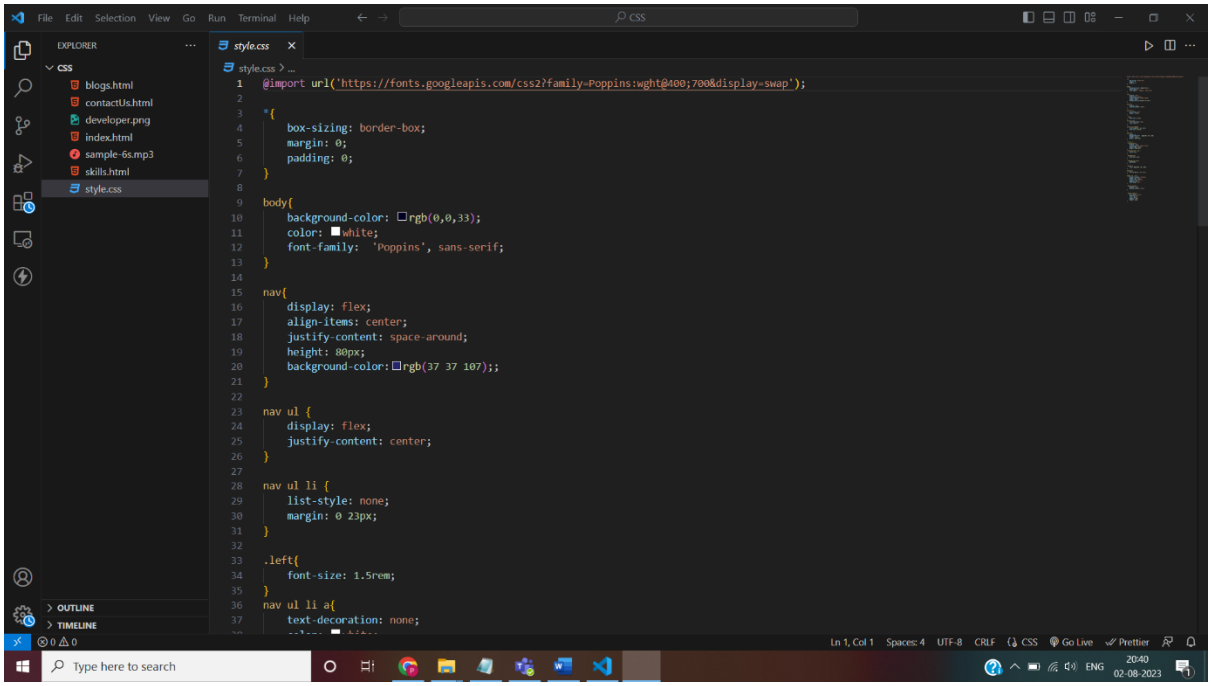
6. Background
    a) background-color: Sets the background color of an element.
    b) background-image: Sets an image as the background of an element.
    c) background-repeat: Specifies how the background image should repeat (repeat, repeat-x, repeat-y, no-repeat).

7. CSS Selectors
    a) Universal Selector (*): Targets all elements on the page.
    b) Type Selector (Element Selector): Targets elements based on their HTML tag names, e.g., p, h1, div.
    c) Class Selector (.classname): Targets elements with a specific class attribute, e.g., .container, .btn.
    d) ID Selector (#idname): Targets a single element with a specific ID attribute, e.g., #header, #section1.
    e) Descendant Selector (Whitespace): Targets elements that are descendants of another element, e.g., ul li, div p.
    f) Child Selector (>): Targets direct children of an element, e.g., ul > li, div > p.
    g) Adjacent Sibling Selector (+): Targets an element that immediately follows another element, e.g., h2 + p.
    h) General Sibling Selector (~): Targets elements that are siblings of another element, e.g., h2 ~ p.
    i) Attribute Selector ([attr=value]): Targets elements with a specific attribute and value, e.g., [type="submit"].
8. Pseudo Classes
    a) :hover: Applies styles when an element is being hovered over by the mouse pointer.
    b) :active: Applies styles when an element is being clicked or activated.
    c) :focus: Applies styles when an element gains focus (e.g., when clicked or tabbed into).
    d) :visited: Applies styles to visited links. (Note: Limited for security reasons)
    e) :nth-child(n): Selects elements based on their position within a parent container.
    f) :first-child: Selects the first child element of its parent.
    g) :last-child: Selects the last child element of its parent.
    h) :not(selector): Negates the selection of elements that match the specified selector.
9. Pseudo Elements
    a) ::before: Inserts content before the selected element (requires content property).
    b) ::after: Inserts content after the selected element (requires content property).
    c) ::first-line: Selects the first line of text within an element.
    d) ::first-letter: Selects the first letter of text within an element.

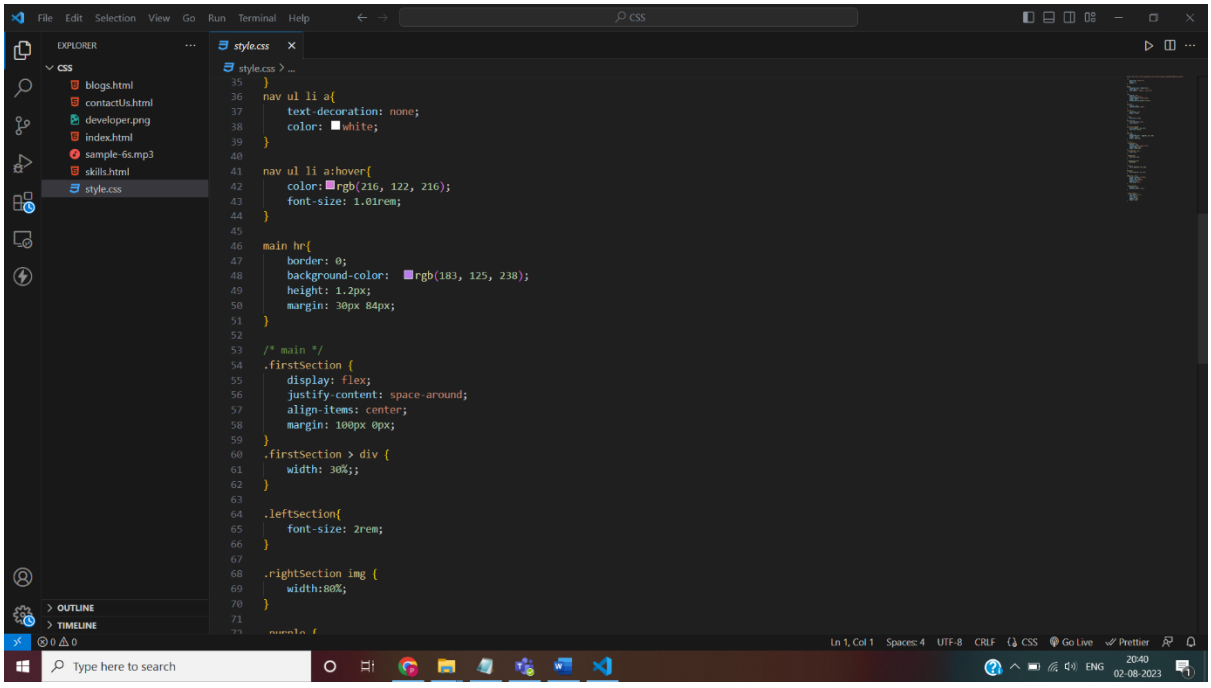
e) ::selection: Applies styles to the portion of text selected by the user.
f) ::placeholder: Targets input fields' placeholder text.

CODE

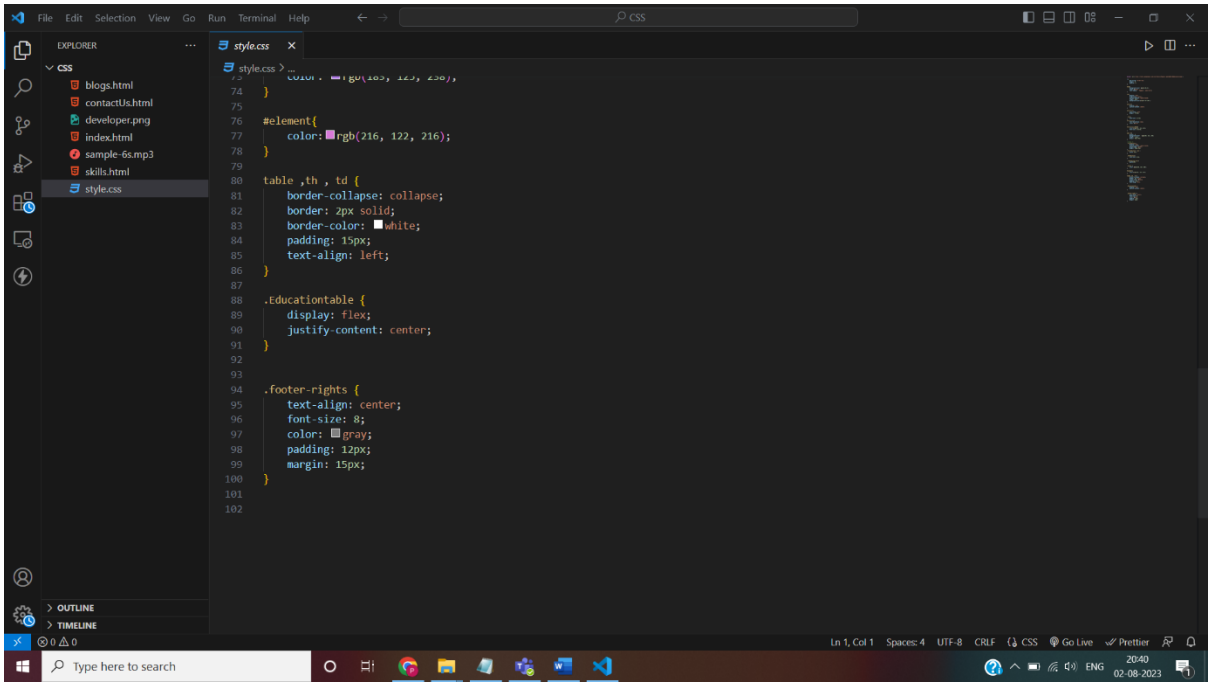




```
@import url('https://fonts.googleapis.com/css2?family=Poppins:wght@400;700&display=swap');

*{
    box-sizing: border-box;
    margin: 0;
    padding: 0;
}

body{
    background-color: rgb(0,0,33);
    color: white;
    font-family:  'Poppins', sans-serif;
}

nav{
    display: flex;
    align-items: center;
    justify-content: space-around;
    height: 80px;
    background-color:rgb(37 37 107);;
}

nav ul {
    display: flex;
    justify-content: center;
}

nav ul li {
    list-style: none;
    margin: 0 23px;
}

.left{
    font-size: 1.5rem;
}
nav ul li a{
    text-decoration: none;
```

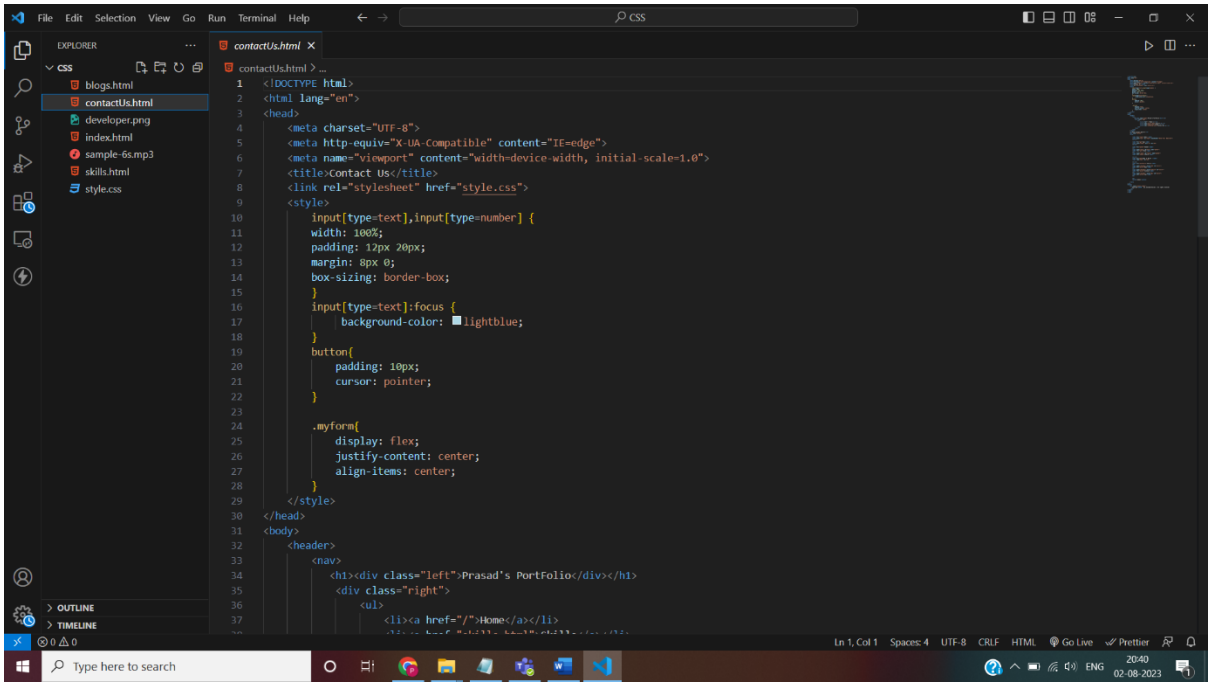


```
}
nav ul li a{
    text-decoration: none;
    color: white;
}

nav ul li a:hover{
    color:rgb(216, 122, 216);
    font-size: 1.01rem;
}

main hr{
    border: 0;
    background-color:  rgb(183, 125, 238);
    height: 1.2px;
    margin: 30px 84px;
}

/* main */
.firstSection {
    display: flex;
    justify-content: space-around;
    align-items: center;
    margin: 100px 0px;
}
.firstSection > div {
    width: 30%;;
}

.leftSection{
    font-size: 2rem;
}

.rightSection img {
    width:80%;
}
```

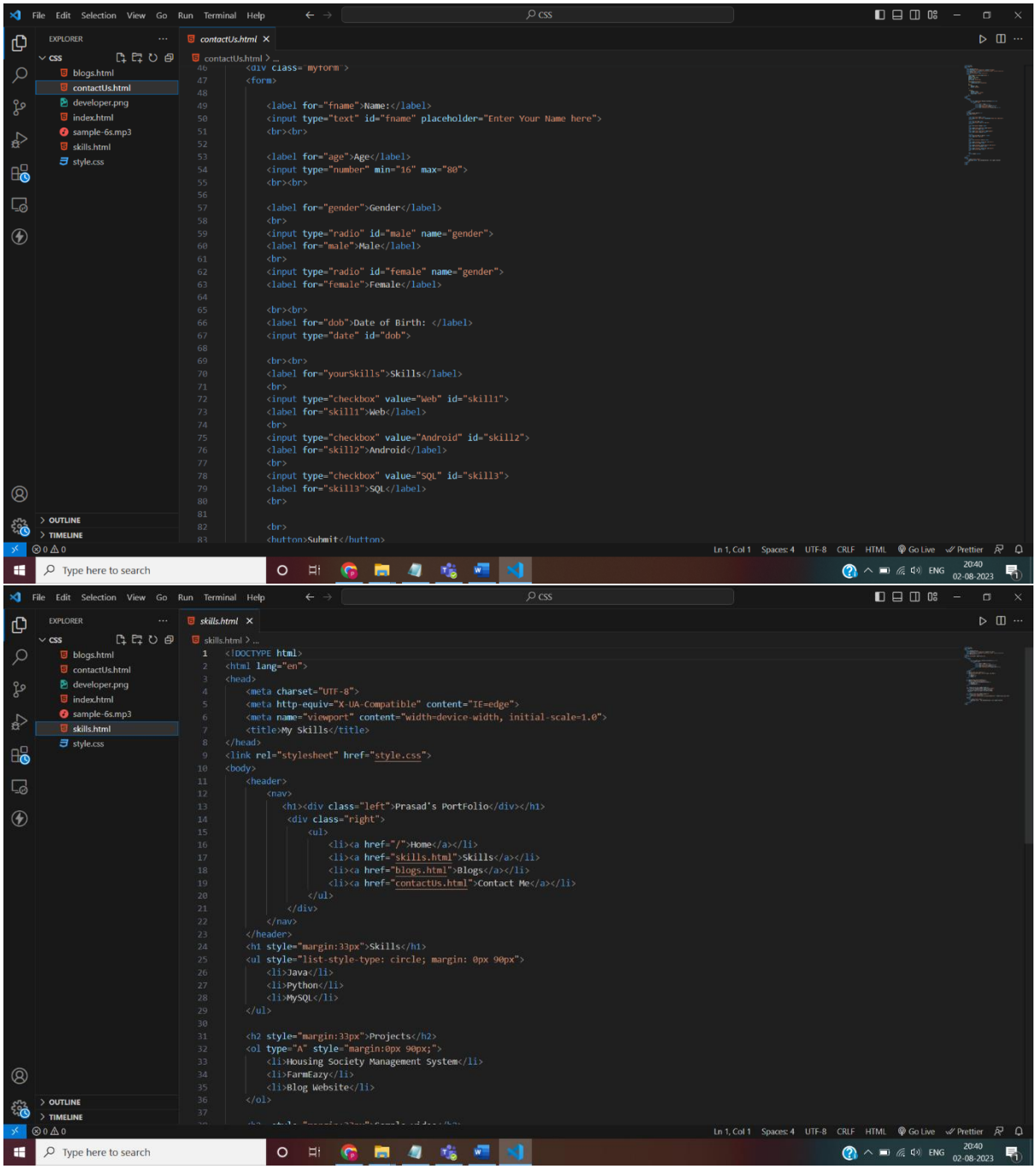


```
#element{
    color:rgb(216, 122, 216);
}

table ,th , td {
    border-collapse: collapse;
    border: 2px solid;
    border-color: white;
    padding: 15px;
    text-align: left;
}

.Educationtable {
    display: flex;
    justify-content: center;
}


.footer-rights {
    text-align: center;
    font-size: 8;
    color: gray;
    padding: 12px;
    margin: 15px;
}
```

\



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Contact Us</title>
    <link rel="stylesheet" href="style.css">
    <style>
        input[type=text],input[type=number] {
        width: 100%;
        padding: 12px 20px;
        margin: 8px 0;
        box-sizing: border-box;
        }
        input[type=text]:focus {
            background-color: lightblue;
        }
        button{
            padding: 10px;
            cursor: pointer;
        }

        .myform{
            display: flex;
            justify-content: center;
            align-items: center;
        }
    </style>
</head>
<body>
    <header>
        <nav>
          <h1><div class="left">Prasad's PortFolio</div></h1>
           <div class="right">
               <ul>
                   <li><a href="/">Home</a></li>
```


File Edit Selection View Go Run Terminal Help
EXPLORER
CSS
blogs.html
contactUs.html
developer.png
index.html
sample-6s.mp3
skills.html
style.css
contactUs.html

<form>
<label for="fname">Name:</label>
<input type="text" id="fname" placeholder="Enter Your Name here">

<label for="age">Age</label>
<input type="number" min="16" max="80">

<label for="gender">Gender</label>

<input type="radio" id="male" name="gender">
<label for="male">Male</label>

<input type="radio" id="female" name="gender">
<label for="female">Female</label>

<label for="dob">Date of Birth: </label>
<input type="date" id="dob">

<label for="yourSkills">Skills</label>

<input type="checkbox" value="Web" id="skill1">
<label for="skill1">Web</label>

<input type="checkbox" value="Android" id="skill2">
<label for="skill2">Android</label>

<input type="checkbox" value="SQL" id="skill3">
<label for="skill3">SQL</label>

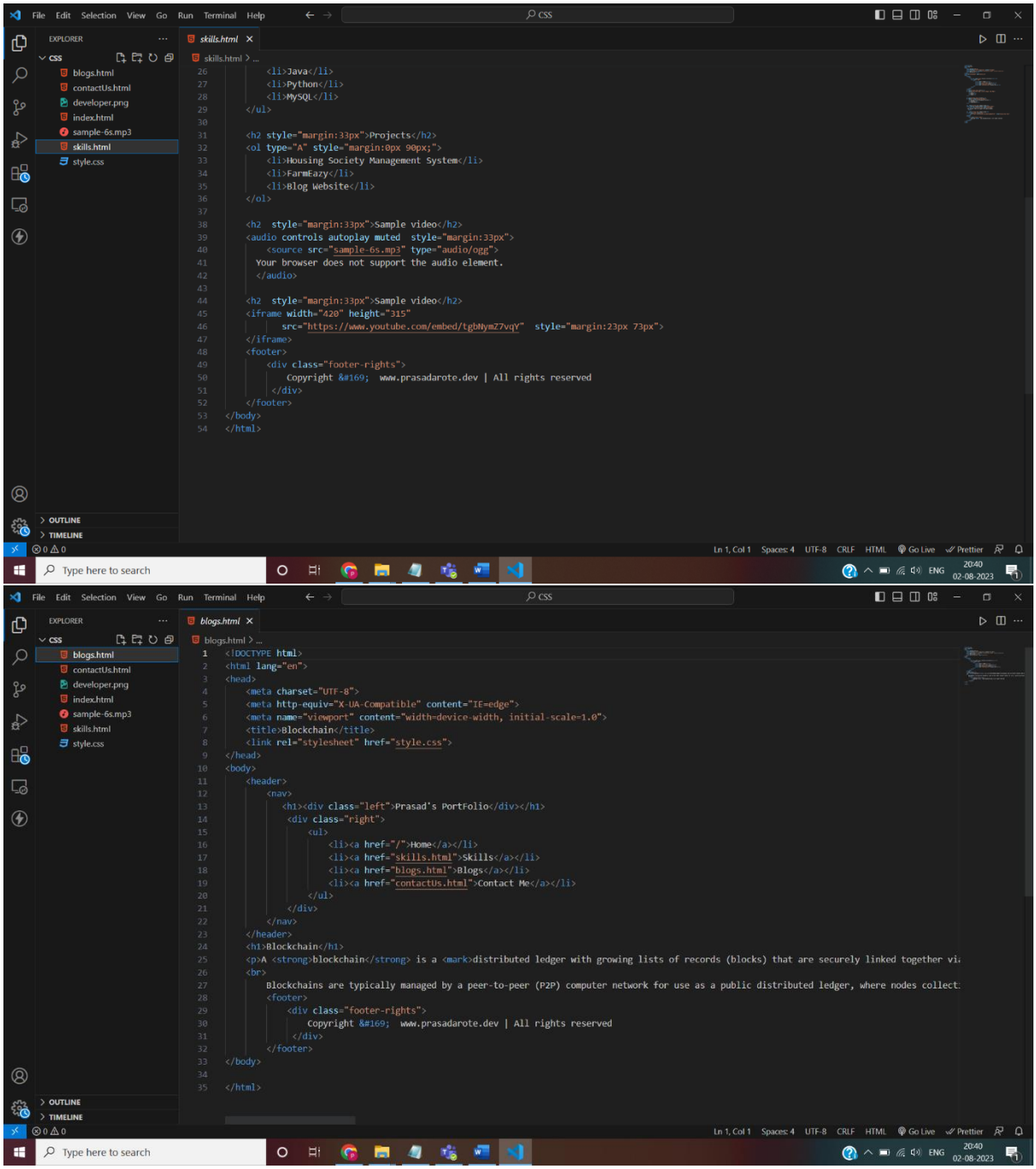
<button>Submit</button>
OUTLINE
TIMELINE
Ln 1, Col 1 Spaces: 4 UTF-8 CRLF HTML Go Live Prettier
Type here to search
ENG
20:40
02-08-2023
File Edit Selection View Go Run Terminal Help
EXPLORER
CSS
blogs.html
contactUs.html
developer.png
index.html
sample-6s.mp3
skills.html
style.css
skills.html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>My Skills</title>
</head>
<link rel="stylesheet" href="style.css">
<body>
<header>
<nav>
<h1> Prasad's PortFolio </h1>

Home
Skills
Blogs
Contact Me

</nav>
</header>
<h1 style="margin:33px">Skills</h1>
<ul style="list-style-type: circle; margin: 0px 90px">
Java
Python
MySQL

<h2 style="margin:33px">Projects</h2>
<ol type="A" style="margin:0px 90px;">
Housing Society Management System
FarmEazy
Blog Website

OUTLINE
TIMELINE
Ln 1, Col 1 Spaces: 4 UTF-8 CRLF HTML Go Live Prettier
Type here to search
ENG
20:40
02-08-2023

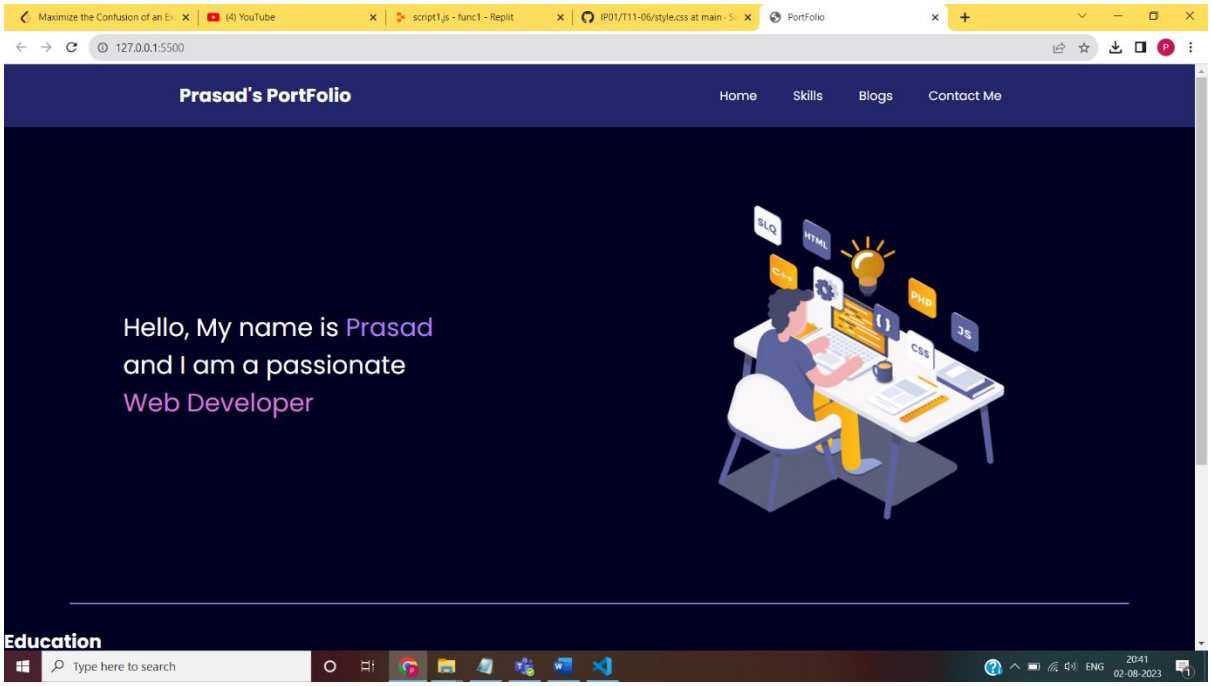


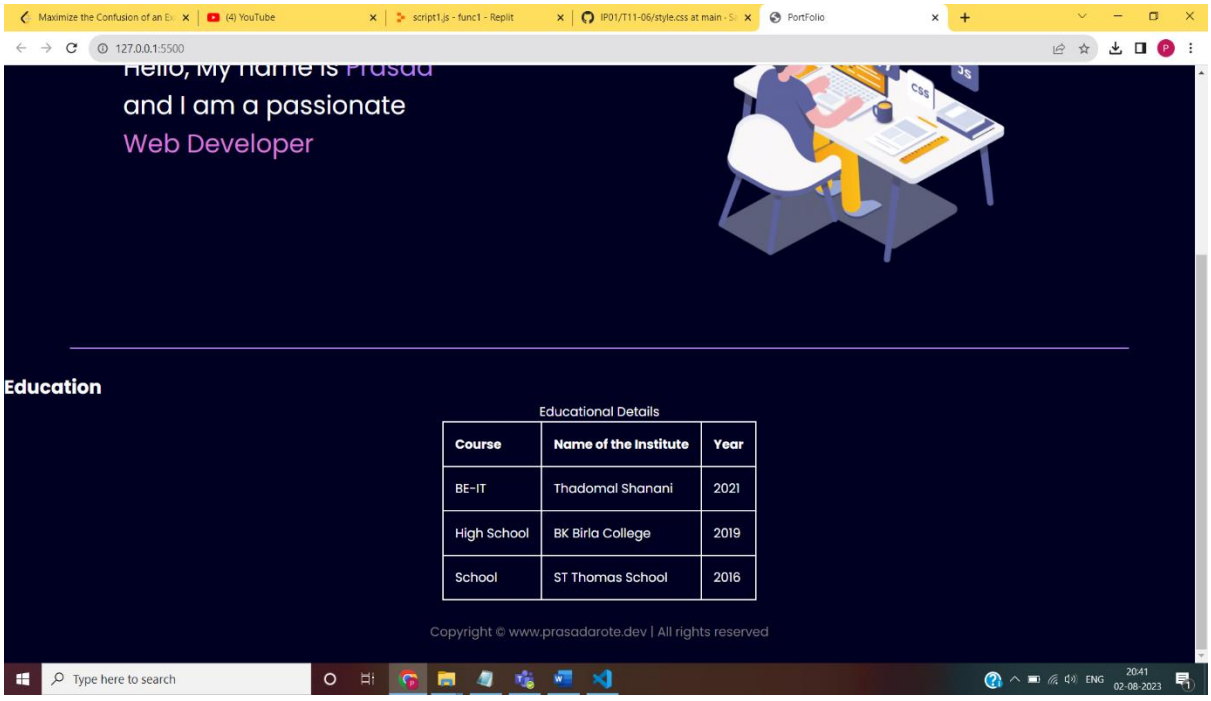
```
skills.html
        <li>Java</li>
        <li>Python</li>
        <li>MySQL</li>
    </ul>

    <h2 style="margin:33px">Projects</h2>
    <ol type="A" style="margin:0px 90px;">
        <li>Housing Society Management System</li>
        <li>FarmEazy</li>
        <li>Blog Website</li>
    </ol>

    <h2  style="margin:33px">Sample video</h2>
    <audio controls autoplay muted  style="margin:33px">
        <source src="sample-6s.mp3" type="audio/ogg">
      Your browser does not support the audio element.
      </audio>

    <h2  style="margin:33px">Sample video</h2>
    <iframe width="420" height="315"
        src="https://www.youtube.com/embed/tgbNymZ7vqY"  style="margin:23px 73px">
    </iframe>
    <footer>
        <div class="footer-rights">
            Copyright &#169;  www.prasadarote.dev | All rights reserved
        </div>
    </footer>
</body>
</html>
```

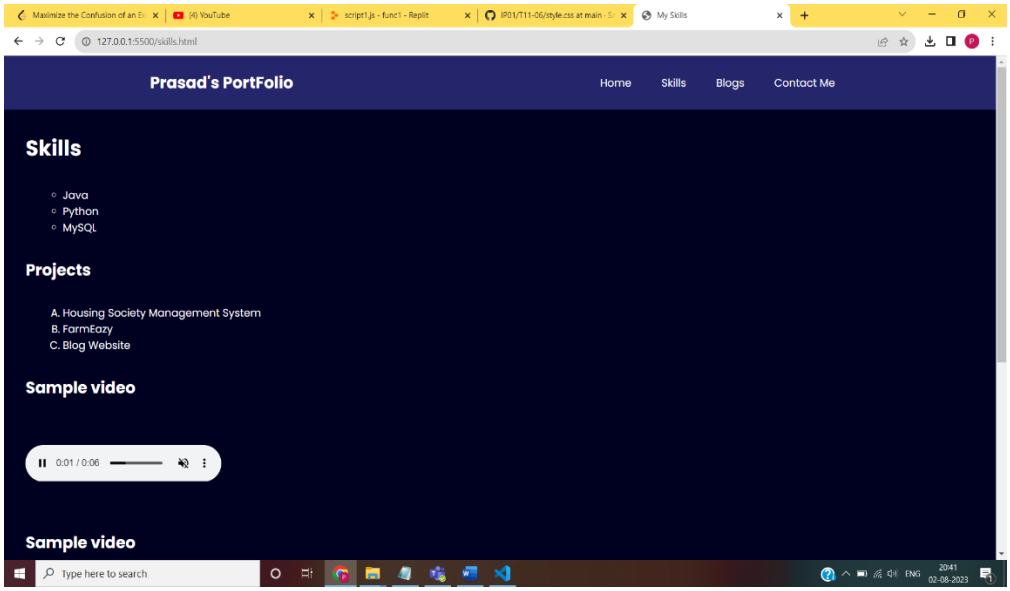


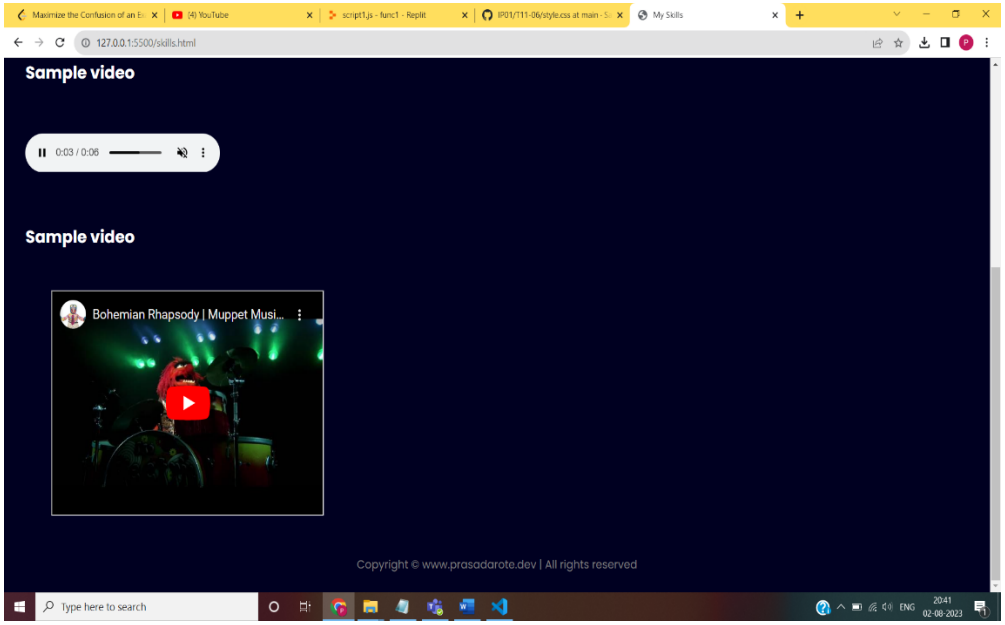
```
blogs.html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Blockchain</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <header>
        <nav>
            <h1><div class="left">Prasad's PortFolio</div></h1>
            <div class="right">
                <ul>
                    <li><a href="/">Home</a></li>
                    <li><a href="skills.html">Skills</a></li>
                    <li><a href="blogs.html">Blogs</a></li>
                    <li><a href="contactUs.html">Contact Me</a></li>
                </ul>
            </div>
        </nav>
    </header>
    <h1>Blockchain</h1>
    <p>A <strong>blockchain</strong> is a <mark>distributed ledger with growing lists of records (blocks) that are securely linked together vi
    <br>
        Blockchains are typically managed by a peer-to-peer (P2P) computer network for use as a public distributed ledger, where nodes collect
        <footer>
            <div class="footer-rights">
                Copyright &#169;  www.prasadarote.dev | All rights reserved
            </div>
        </footer>
</body>

</html>
```

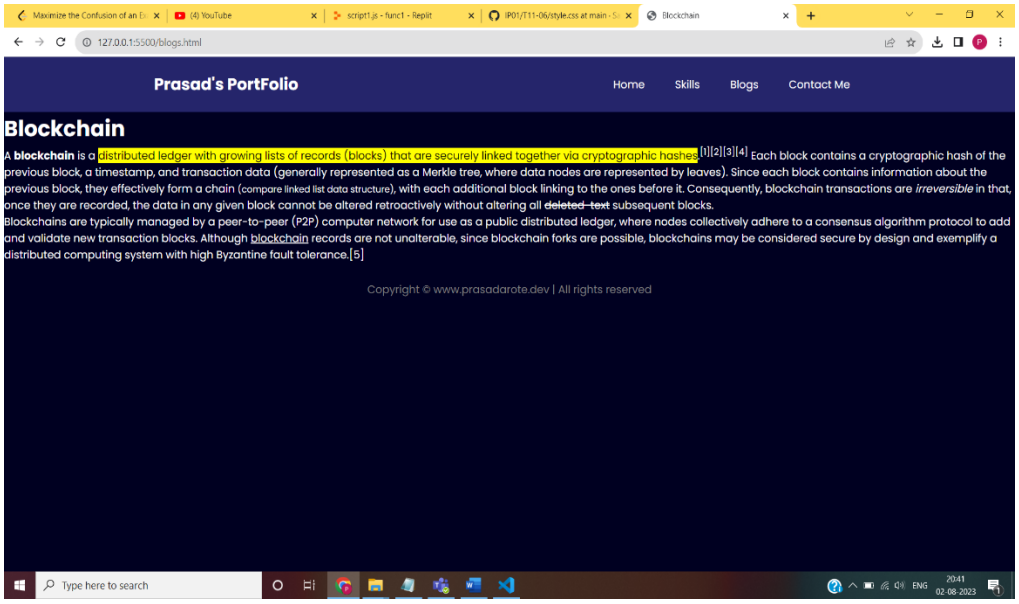
OUTPUT:

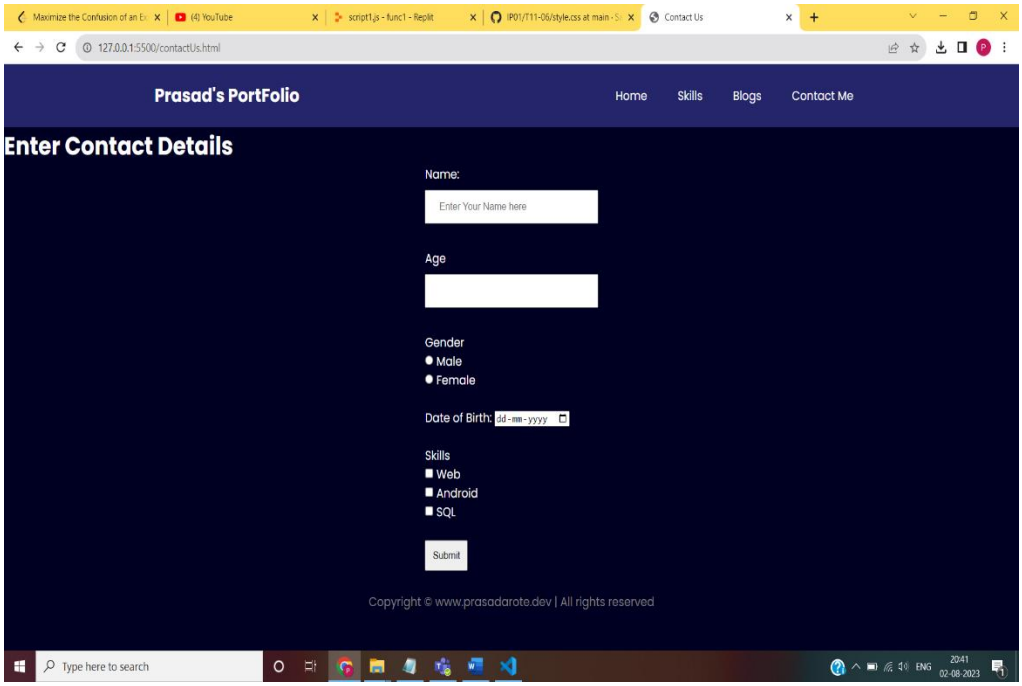

Prasad's PortFolio
Home
Skills
Blogs
Contact Me
Hello, My name is Prasad
and I am a passionate
Web Developer
Education


and I am a passionate
Web Developer
Education
Educational Details
Course
Name of the Institute
Year
BE-IT
Thadomal Shanani
2021
High School
BK Birla College
2019
School
ST Thomas School
2016
Copyright © www.prasadarote.dev | All rights reserved


Prasad's PortFolio
Home
Skills
Blogs
Contact Me
Skills
Java
Python
MySQL
Projects
A. Housing Society Management System
B. FarmEazy
C. Blog Website
Sample video
0:01 / 0:06
Sample video


Sample video
0:03 / 0:06
Sample video
Bohemian Rhapsody | Muppet Musi...
Copyright © www.prasadarote.dev | All rights reserved


Prasad's PortFolio
Home
Skills
Blogs
Contact Me
Blockchain
A blockchain is a distributed ledger with growing lists of records (blocks) that are securely linked together via cryptographic hashes.[1][2][3][4] Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data (generally represented as a Merkle tree, where data nodes are represented by leaves). Since each block contains information about the previous block, they effectively form a chain (compare linked list data structure), with each additional block linking to the ones before it. Consequently, blockchain transactions are irreversible in that, once they are recorded, the data in any given block cannot be altered retroactively without altering all deleted text subsequent blocks.
Blockchains are typically managed by a peer-to-peer (P2P) computer network for use as a public distributed ledger, where nodes collectively adhere to a consensus algorithm protocol to add and validate new transaction blocks. Although blockchain records are not unalterable, since blockchain forks are possible, blockchains may be considered secure by design and exemplify a distributed computing system with high Byzantine fault tolerance.[5]
Copyright © www.prasadarote.dev | All rights reserved



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

Assignment No 3

Aim: Develop a web page using the Bootstrap framework. Grid system, Forms, Button, Navbar, Breadcrumb, Jumbotron should be used.

LO3: To expose students to developing responsive layout.

Theory:

Bootstrap is a popular front-end framework created by Twitter and now maintained by the open-source community. Bootstrap provides a set of pre-designed HTML, CSS, and JavaScript components that developers can use to build responsive and visually appealing websites and web applications.

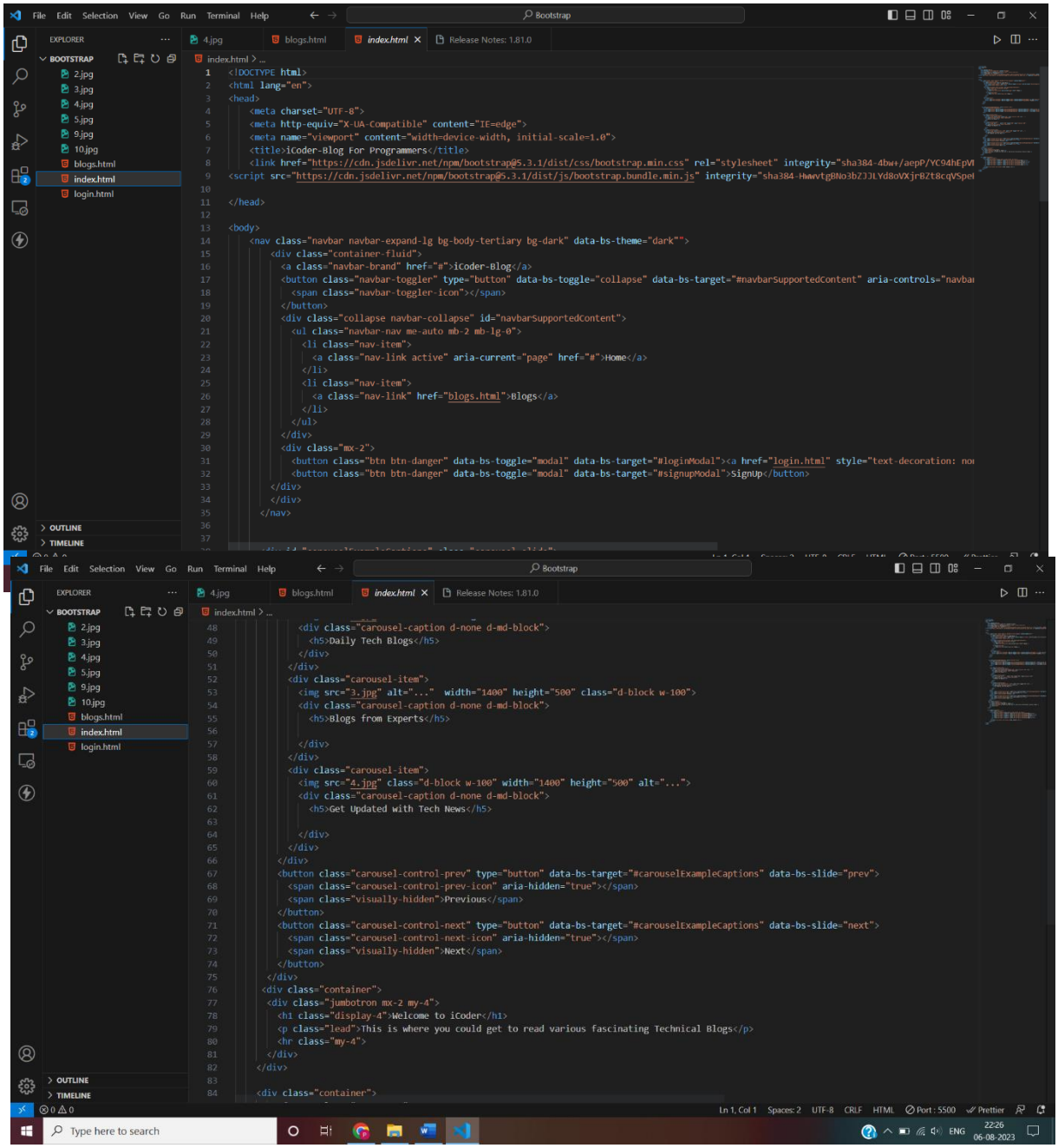
The Bootstrap framework is designed to simplify the process of creating consistent, mobile-first, and user-friendly web interfaces. It includes a wide range of ready-to-use UI components, such as buttons, forms, navigation bars, modals, carousels, and more.

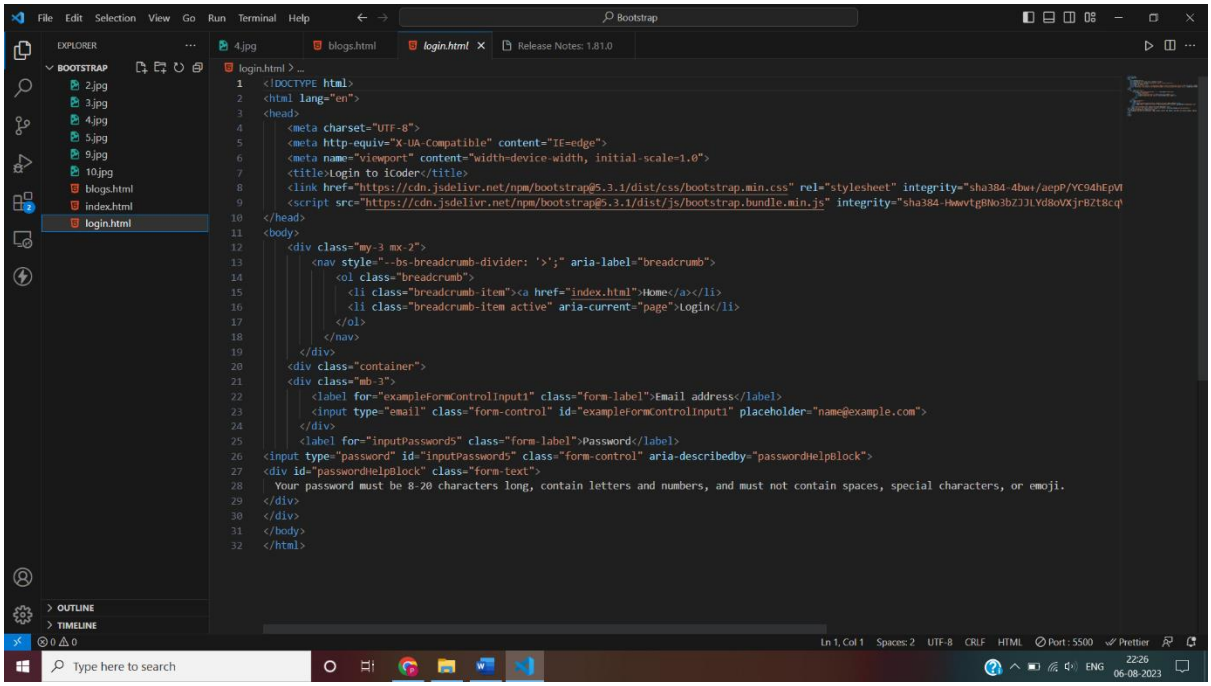
Key features and components of the Bootstrap framework include:

1. Grid System: Bootstrap's grid system is a flexible layout mechanism that allows developers to create responsive, column-based layouts. It helps in organizing content into rows and columns, which automatically adjust based on the screen size, ensuring the content looks great on various devices, from large desktops to mobile phones.
2. Components: Bootstrap offers a wide range of UI components, such as buttons, forms, dropdowns, navigation bars, cards, and alerts, which can be easily customized and integrated into web projects.
3. JavaScript Plugins: Bootstrap includes JavaScript plugins that add interactivity and functionality to the UI components. For example, there are plugins for sliders, modals, tooltips, and more.
4. Forms: Bootstrap provides styles and classes to create attractive and functional forms easily. Forms are essential for collecting user input and submitting data. Bootstrap form components include text inputs, checkboxes, radio buttons, select dropdowns, and more. These components come with predefined styles, making it simple to create consistent and visually appealing forms.
5. Buttons in Bootstrap can be customized and styled easily using various classes. The framework offers different button styles, such as primary, secondary, success, danger, warning, info, and more. Developers can also create various button sizes and styles, including outline buttons, block-level buttons, and buttons with icons.
6. Navbar: The Navbar (short for Navigation Bar) is a horizontal component that provides site navigation links and other content. In Bootstrap, the Navbar is responsive and collapses into a "hamburger" menu on smaller screens to improve mobile usability. It is a crucial component for creating a user-friendly and accessible navigation experience on web applications and websites.

7. Breadcrumb: Breadcrumbs are a navigational aid that helps users understand their current location within a website's hierarchy. Bootstrap offers breadcrumb styles and classes to create breadcrumb trails easily. It helps users navigate backward through a series of links, improving the overall user experience.
8. Jumbotron: The Jumbotron is a large, prominent container that is often used to showcase important content or messages on a web page. It's typically used as a header or hero section to grab the user's attention. With Bootstrap, developers can quickly create visually appealing Jumbotrons with customizable background images or colors and large, bold headings.

CODE :




File Edit Selection View Go Run Terminal Help
Bootstrap
EXPLORER
BOOTSTRAP
2.jpg
3.jpg
4.jpg
5.jpg
9.jpg
10.jpg
blogs.html
index.html
login.html
4.jpg
blogs.html
login.html
Release Notes: 1.81.0
login.html > ...
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Login to iCoder</title>
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-4bw+/aepP/YC94hEpVN
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.bundle.min.js" integrity="sha384-HwwvtgBNo3bZJJLYd8oVXjrBZt8cqV
</head>
<body>

<nav style="--bs-breadcrumb-divider: '>';" aria-label="breadcrumb">
<ol class="breadcrumb">
<li class="breadcrumb-item">Home
<li class="breadcrumb-item active" aria-current="page">Login

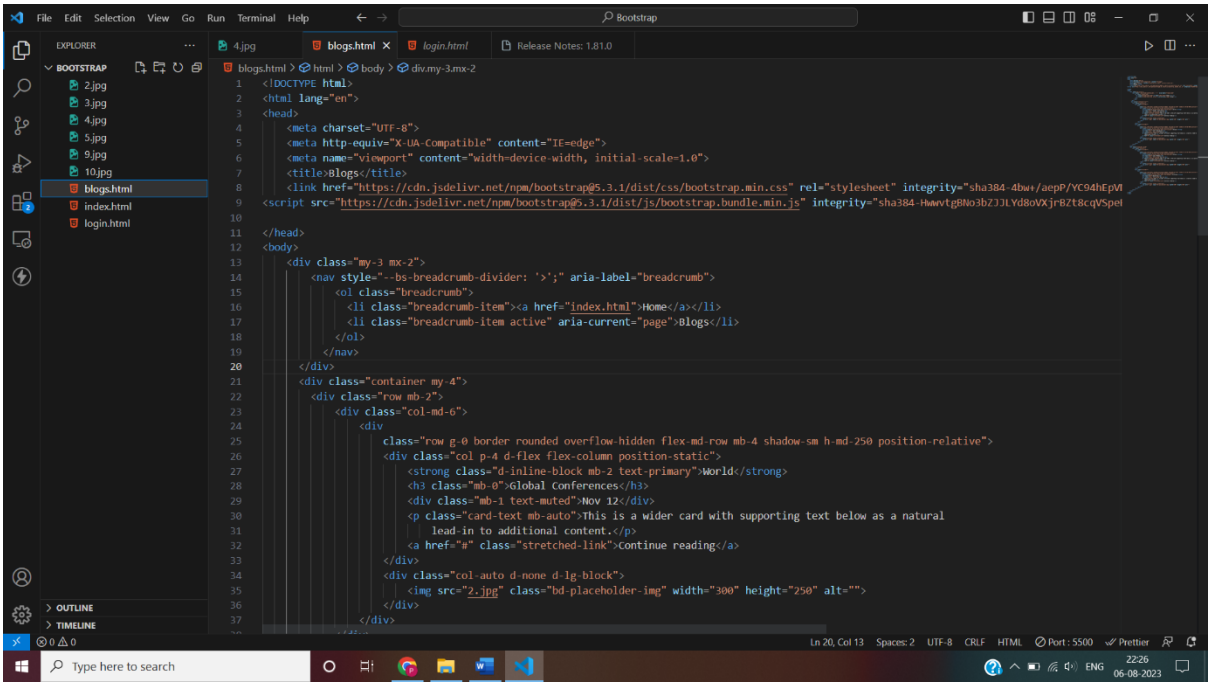
</nav>

<label for="exampleFormControlInput1" class="form-label">Email address</label>
<input type="email" class="form-control" id="exampleFormControlInput1" placeholder="name@example.com">

<label for="inputPassword5" class="form-label">Password</label>
<input type="password" id="inputPassword5" class="form-control" aria-describedby="passwordHelpBlock">

Your password must be 8-20 characters long, contain letters and numbers, and must not contain spaces, special characters, or emoji.

</body>
</html>
OUTLINE
TIMELINE
Ln 1, Col 1 Spaces: 2 UTF-8 CRLF HTML Port: 5500 Prettier
Type here to search
ENG
22:26
06-08-2023


File Edit Selection View Go Run Terminal Help
Bootstrap
EXPLORER
BOOTSTRAP
2.jpg
3.jpg
4.jpg
5.jpg
9.jpg
10.jpg
blogs.html
index.html
login.html
4.jpg
blogs.html
login.html
Release Notes: 1.81.0
blogs.html > html > body > div.my-3.mx-2
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Blogs</title>
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-4bw+/aepP/YC94hEpVN
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.bundle.min.js" integrity="sha384-HwwvtgBNo3bZJJLYd8oVXjrBZt8cqVSpeI
</head>
<body>

<nav style="--bs-breadcrumb-divider: '>';" aria-label="breadcrumb">
<ol class="breadcrumb">
<li class="breadcrumb-item">Home
<li class="breadcrumb-item active" aria-current="page">Blogs

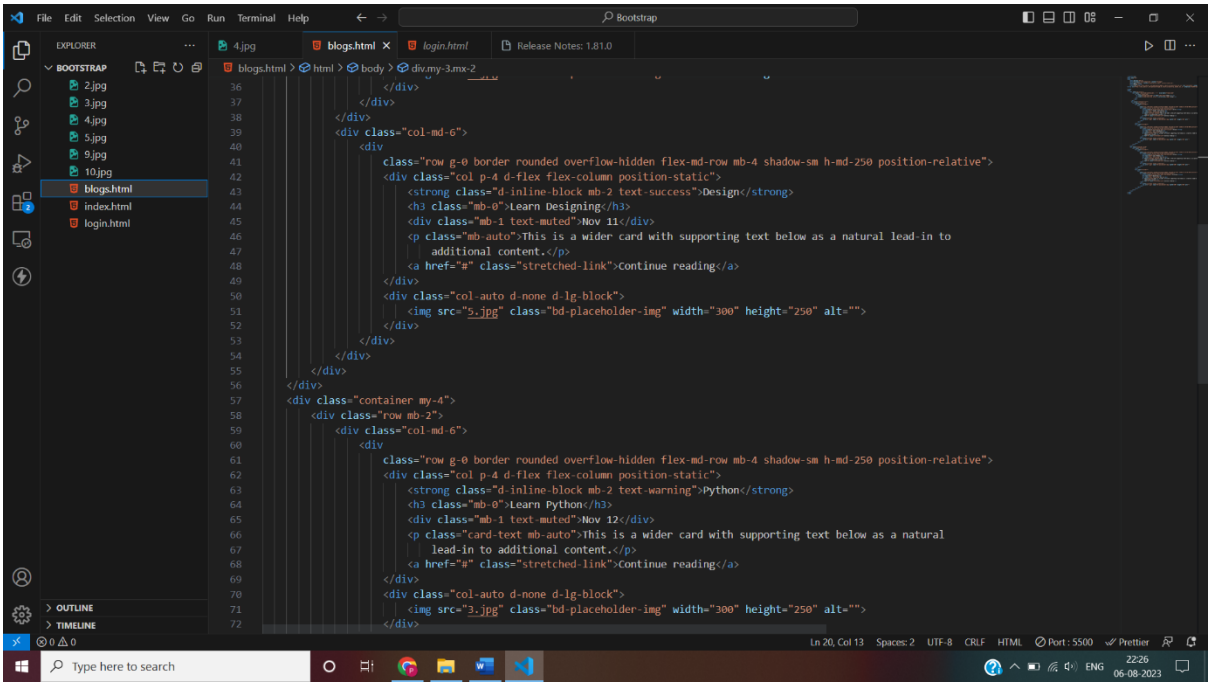
</nav>

<div
class="row g-0 border rounded overflow-hidden flex-md-row mb-4 shadow-sm h-md-250 position-relative">

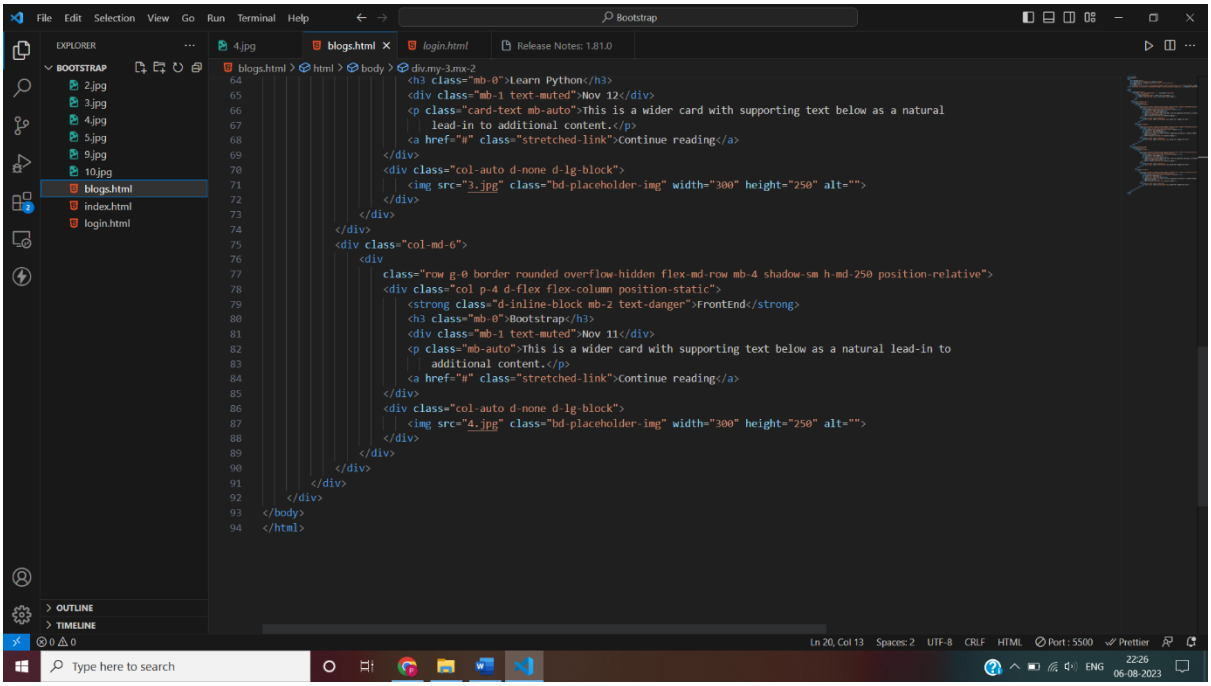
World
<h3 class="mb-0">Global Conferences</h3>
Nov 12
This is a wider card with supporting text below as a natural
lead-in to additional content.
Continue reading

<img src="2.jpg" class="bd-placeholder-img" width="300" height="250" alt="">

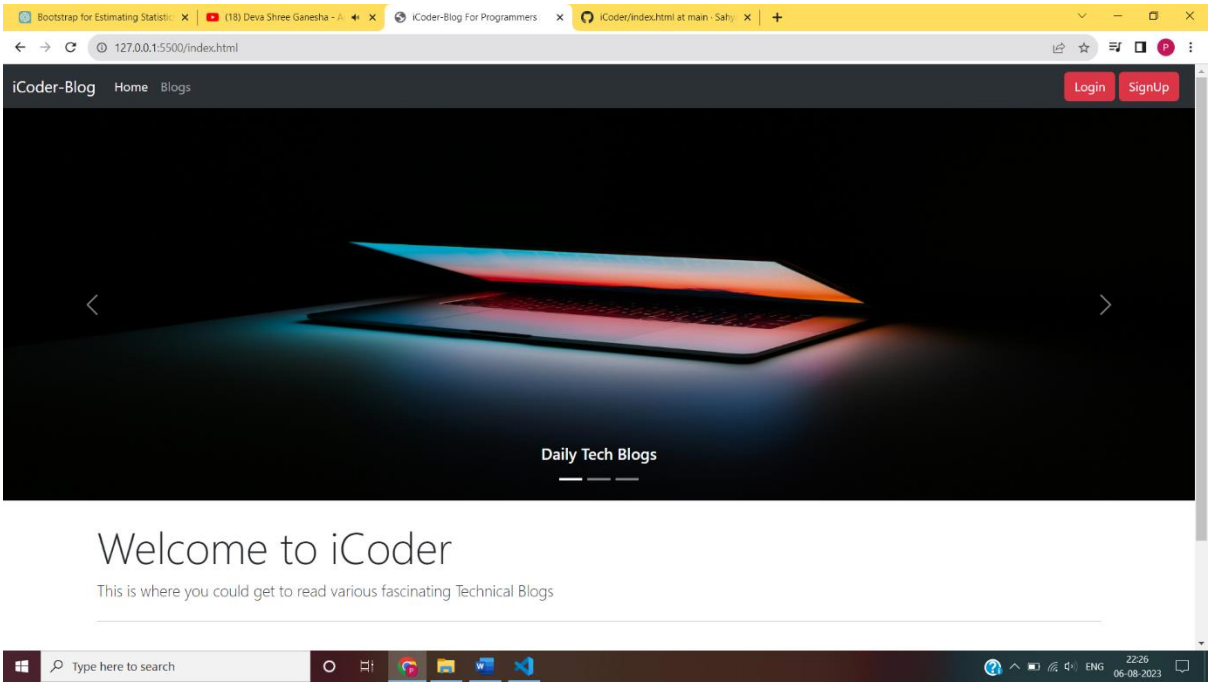
OUTLINE
TIMELINE
Ln 20, Col 13 Spaces: 2 UTF-8 CRLF HTML Port: 5500 Prettier
Type here to search
ENG
22:26
06-08-2023

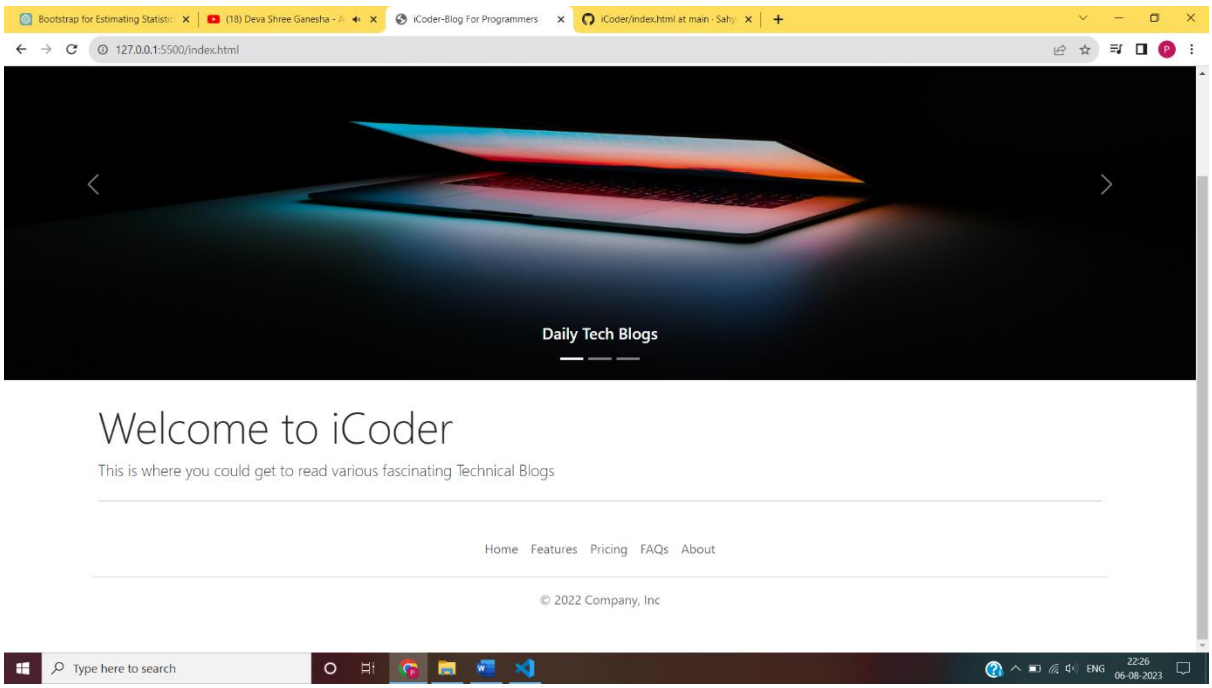


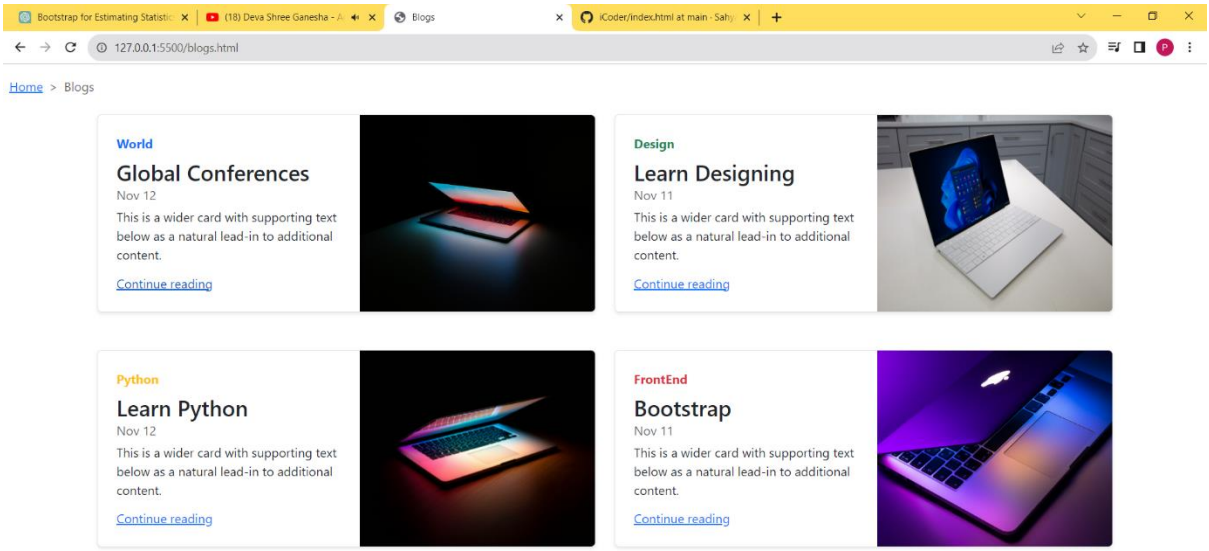
```
                        </div>
                    </div>
                </div>
                <div class="col-md-6">
                    <div
                        class="row g-0 border rounded overflow-hidden flex-md-row mb-4 shadow-sm h-md-250 position-relative">
                        <div class="col p-4 d-flex flex-column position-static">
                            <strong class="d-inline-block mb-2 text-success">Design</strong>
                            <h3 class="mb-0">Learn Designing</h3>
                            <div class="mb-1 text-muted">Nov 11</div>
                            <p class="mb-auto">This is a wider card with supporting text below as a natural lead-in to
                                additional content.</p>
                            <a href="#" class="stretched-link">Continue reading</a>
                        </div>
                        <div class="col-auto d-none d-lg-block">
                            <img src="5.jpg" class="bd-placeholder-img" width="300" height="250" alt="">
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
    <div class="container my-4">
        <div class="row mb-2">
            <div class="col-md-6">
                <div
                    class="row g-0 border rounded overflow-hidden flex-md-row mb-4 shadow-sm h-md-250 position-relative">
                    <div class="col p-4 d-flex flex-column position-static">
                        <strong class="d-inline-block mb-2 text-warning">Python</strong>
                        <h3 class="mb-0">Learn Python</h3>
                        <div class="mb-1 text-muted">Nov 12</div>
                        <p class="card-text mb-auto">This is a wider card with supporting text below as a natural
                            lead-in to additional content.</p>
                        <a href="#" class="stretched-link">Continue reading</a>
                    </div>
                    <div class="col-auto d-none d-lg-block">
                        <img src="3.jpg" class="bd-placeholder-img" width="300" height="250" alt="">
                    </div>
```

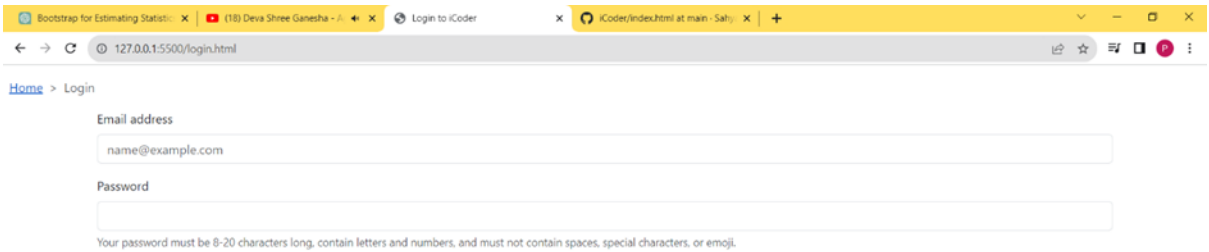


```
                        <h3 class="mb-0">Learn Python</h3>
                        <div class="mb-1 text-muted">Nov 12</div>
                        <p class="card-text mb-auto">This is a wider card with supporting text below as a natural
                            lead-in to additional content.</p>
                        <a href="#" class="stretched-link">Continue reading</a>
                    </div>
                    <div class="col-auto d-none d-lg-block">
                        <img src="3.jpg" class="bd-placeholder-img" width="300" height="250" alt="">
                    </div>
                </div>
            </div>
            <div class="col-md-6">
                <div
                    class="row g-0 border rounded overflow-hidden flex-md-row mb-4 shadow-sm h-md-250 position-relative">
                    <div class="col p-4 d-flex flex-column position-static">
                        <strong class="d-inline-block mb-2 text-danger">FrontEnd</strong>
                        <h3 class="mb-0">Bootstrap</h3>
                        <div class="mb-1 text-muted">Nov 11</div>
                        <p class="mb-auto">This is a wider card with supporting text below as a natural lead-in to
                            additional content.</p>
                        <a href="#" class="stretched-link">Continue reading</a>
                    </div>
                    <div class="col-auto d-none d-lg-block">
                        <img src="4.jpg" class="bd-placeholder-img" width="300" height="250" alt="">
                    </div>
                </div>
            </div>
        </div>
    </div>
</body>
</html>
```


iCoder-Blog
Home
Blogs
Login
SignUp
Daily Tech Blogs
Welcome to iCoder
This is where you could get to read various fascinating Technical Blogs
Type here to search


127.0.0.1:5500/index.html
Daily Tech Blogs
Welcome to iCoder
This is where you could get to read various fascinating Technical Blogs
Home
Features
Pricing
FAQs
About
© 2022 Company, Inc
Type here to search





Conclusion:

Successfully incorporated various Bootstrap components, including Grid, Forms, Buttons, Navbar, Breadcrumb, and Jumbotron, to design a responsive and visually appealing web interface. Utilizing these components allowed me to streamline the development process, enhance user experience, and maintain consistency across different devices and screen sizes.

Assignment No 4

AIM: WAP in JS to study conditional Statements, Loops and Functions. Write a JavaScript code to change the background color of the web page automatically after every 5 second.

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

THEORY:

Conditional Statements:

1) if Statement:

The if statement is used to execute a block of code only if a specified condition is true. It has an optional else clause to provide an alternative block of code to execute when the condition is false.

```
if (condition) {
  // Code to execute if the condition is true
} else {
  // Code to execute if the condition is false
}
```

2) else if Statement:

The else if statement allows you to check multiple conditions in sequence. It's used when you have multiple alternatives.

```
if (condition1) {
  // Code to execute if condition1 is true
} else if (condition2) {
  // Code to execute if condition2 is true
} else {
  // Code to execute if no condition is true
}
```

3) switch Statement:

The switch statement is used to select one of many code blocks to be executed. It compares an expression against different cases and executes the code block associated with the first matching case.

```
switch (expression) {
  case value1:
    // Code to execute if expression matches value1
    break;
  case value2:
```

```
    // Code to execute if expression matches value2
    break;
  default:
    // Code to execute if no case matches
}
```

Loops:

1) for Loop:

The for loop is used to repeatedly execute a block of code as long as a specified condition is true. It typically includes an initialization, a condition, and an increment or decrement.

```
for (initialization; condition; increment) {
  // Code to execute repeatedly
}
```

2) while Loop:

The while loop executes a block of code while a specified condition is true.

```
while (condition) {
  // Code to execute repeatedly
}
```

3) do...while Loop:

The do...while loop is similar to the while loop, but it ensures that the code block is executed at least once before checking the condition.

```
do {
  // Code to execute at least once
} while (condition);
```

Functions:

A function in JavaScript is a block of code that can be defined and reused. Functions can have parameters and return values.

```
function functionName(parameters) {
  // Code to execute
  return value; // Optional return statement
}
```

Function Definition: Declares a function with a name and a set of parameters.

Function Call: Invokes the function with specific arguments.

1) Named Functions:

A named function is defined with a name and can be called by that name. It's one of the most common ways to define functions.

```
function greet(name) {
  return "Hello, " + name + "!";
}
```

2) Anonymous Functions:

An anonymous function is a function without a name. It's often used in situations where a function is used as an argument to another function, like in event handlers or higher-order functions.

```
const sum = function(x, y) {
  return x + y;
};
```

3) Arrow Functions:

Arrow functions provide a more concise syntax for writing functions, especially for small inline functions. They have a more compact syntax and automatically inherit the this value from the surrounding code.

```
const multiply = (a, b) => a * b;
```

4) Function Expressions:

A function expression is any valid JavaScript expression that defines a function. It can be named or anonymous.

```
const divide = function(x, y) {
  return x / y;
};
```

5) Function Declarations vs. Function Expressions:

Function declarations are hoisted to the top of their scope, which means they can be called before they're defined in the code. Function expressions are not hoisted in the same way and should be defined before they are called.

6) Callback Functions:

Callback functions are functions passed as arguments to another function and are executed later, often in response to an event or an asynchronous operation.

function fetchData(url, callback) {

// Fetch data from URL

callback(data);

}

CODE:

index.js

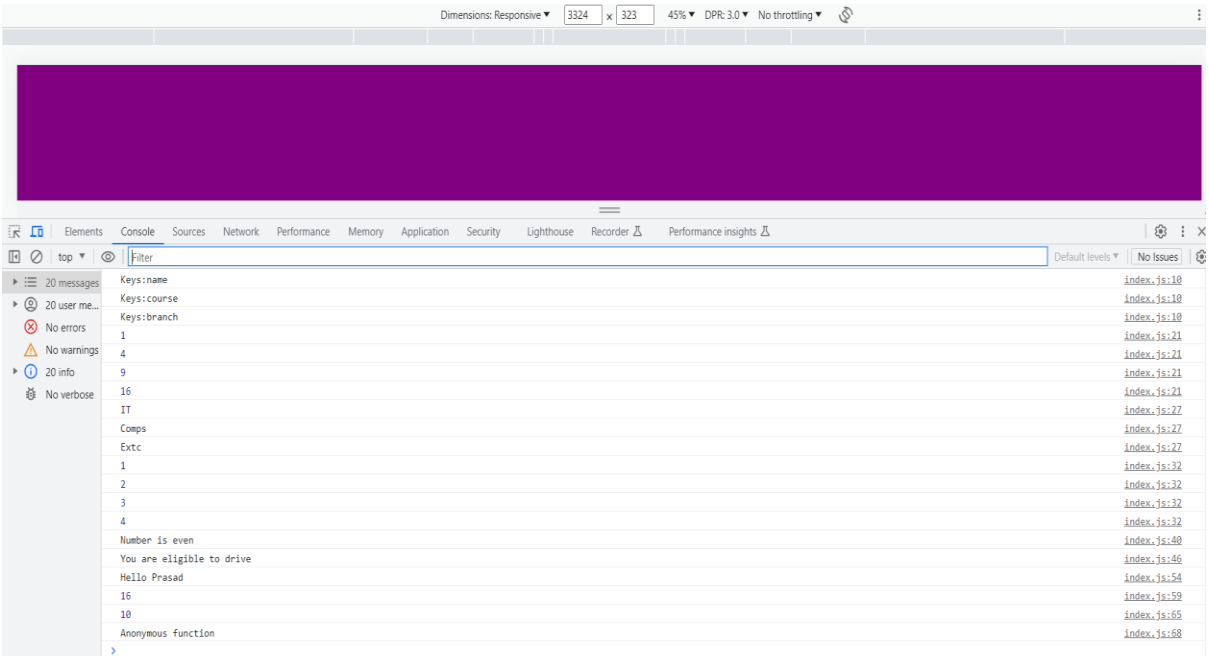
```
//loops

let student = {
    name:"Prasad",
    course:"BE",
    branch:"IT"
}

for(let k in student){
    console.log("Keys:"+k);
}

let arr = [1,2,3,4]
let sqr = []
function func(value){
    sqr.push(value*value);
}
arr.forEach(func)

for(let v of sqr){
    console.log(v);
}

let dept = ["IT","Comps","Extc"]

for(let i =0;i<dept.length;i++){
    console.log(dept[i]);
}

let j = 1;
while(j<5){
    console.log(j)
    j++;
}
```

index.js

```
const colours = ['red','yellow','purple','blue','green']

const changeColor = () => {
    // var randomColor = Math.floor(Math.random()*16777215).toString(16);
    let color = colours.shift();
    document.body.style.backgroundColor = color;
    colours.push(color)
}
setInterval(changeColor,5000)
```

```
index.js > ...
36  //Conditional Statements
37  const n1 = 44
38
39  if(n1%2==0){
40      console.log("Number is even")
41  }else{
42      console.log("Number is odd")
43  }
44
45  let age = 20
46  age>18?console.log("You are eligible to drive"):console.log("You are not eligible to drive");
47
48
49
50
51  //Functions
52
53  const greet = (person="Human") => {
54      console.log("Hello "+person)
55  }
56  greet('Prasad')
57
58  const Sqr = n => n*n
59  console.log(Sqr(4))
60
61  function multiply(a, b = 1) {
62      return a * b;
63  }
64
65  console.log(multiply(5, 2));
66
67  let show = function() {
68      console.log('Anonymous function');
69  };
70
71  show();
72
```

OUTPUT:



CONCLUSION:

Here we understood loops, conditionals, and functions in JavaScript.

Assignment 4b

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

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

THEORY:

Inheritance:

Inheritance is a fundamental concept in object-oriented programming (OOP), including JavaScript. It allows one class (or constructor function) to inherit properties and methods from another class, creating a hierarchical relationship between them. In JavaScript, you can achieve inheritance through prototypes and prototype chains.

Types of Inheritance

1) **Prototypal Inheritance:**
   Prototypal inheritance is the most fundamental and widely used form of inheritance in JavaScript. Every object in JavaScript has a prototype, and objects can inherit properties and methods from their prototypes. When you access a property or method on an object, JavaScript first checks if that property or method exists on the object itself. If not, it looks up the prototype chain to find it.

2) **ES6 Class Inheritance:**
   With the introduction of ECMAScript 6 (ES6), JavaScript introduced a more class-like syntax for creating objects and inheritance. Although it's just syntactical sugar over the prototype-based inheritance, it provides a clearer syntax for defining classes and inheritance.

3) **Functional Inheritance:**
   Functional inheritance involves creating objects by calling functions that return an object. This approach is often used when you want to create objects with specific properties and behaviours without using constructors or classes.

Iterators:

Iterators are a way to traverse or loop through a collection of data in a sequential manner. In JavaScript, the Iterable protocol allows objects to define their own iteration behaviour. The most common use case for iterators is with arrays, strings, maps, sets, and custom iterable objects.

Iterables: An iterable is an object that has an iterator, which defines how to iterate over its values. Iterables are denoted by the Symbol.iterator property.

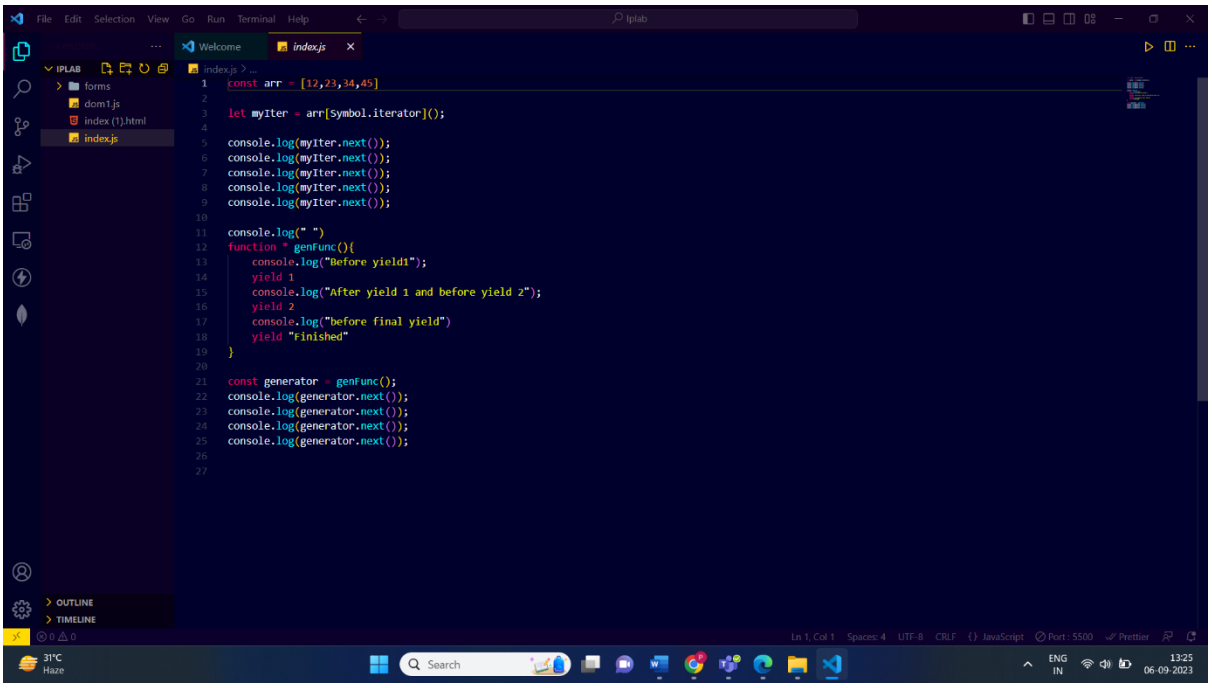
Iterators: An iterator is an object that provides a next() method, which returns the next value in the sequence along with an indicator of whether the sequence has ended.

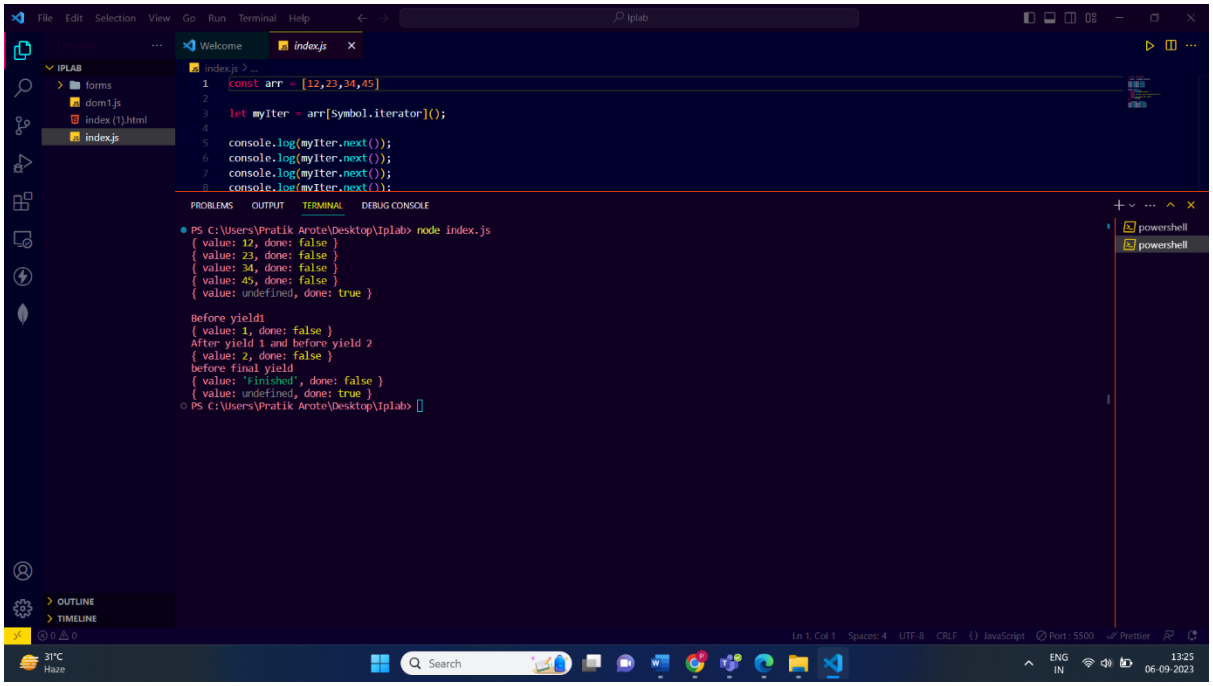
Generators:

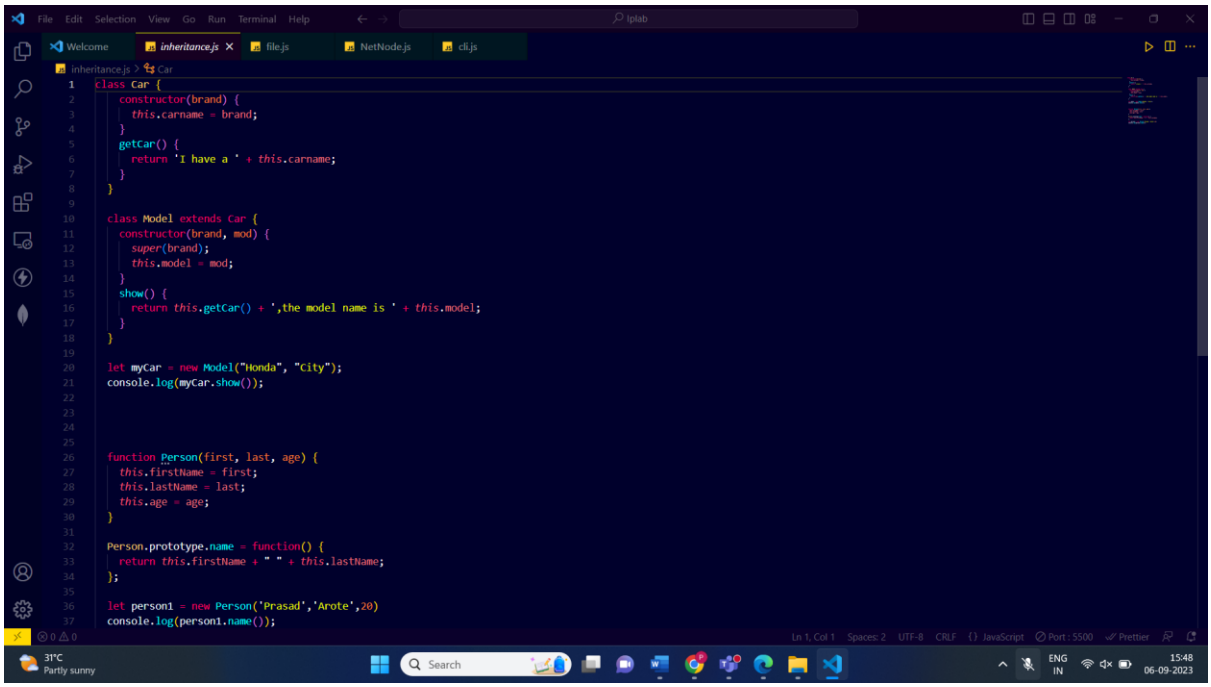
Generators are a special type of function in JavaScript that allow you to pause and resume their execution. They are defined using the function* syntax and use the yield keyword to yield control back to the caller temporarily.

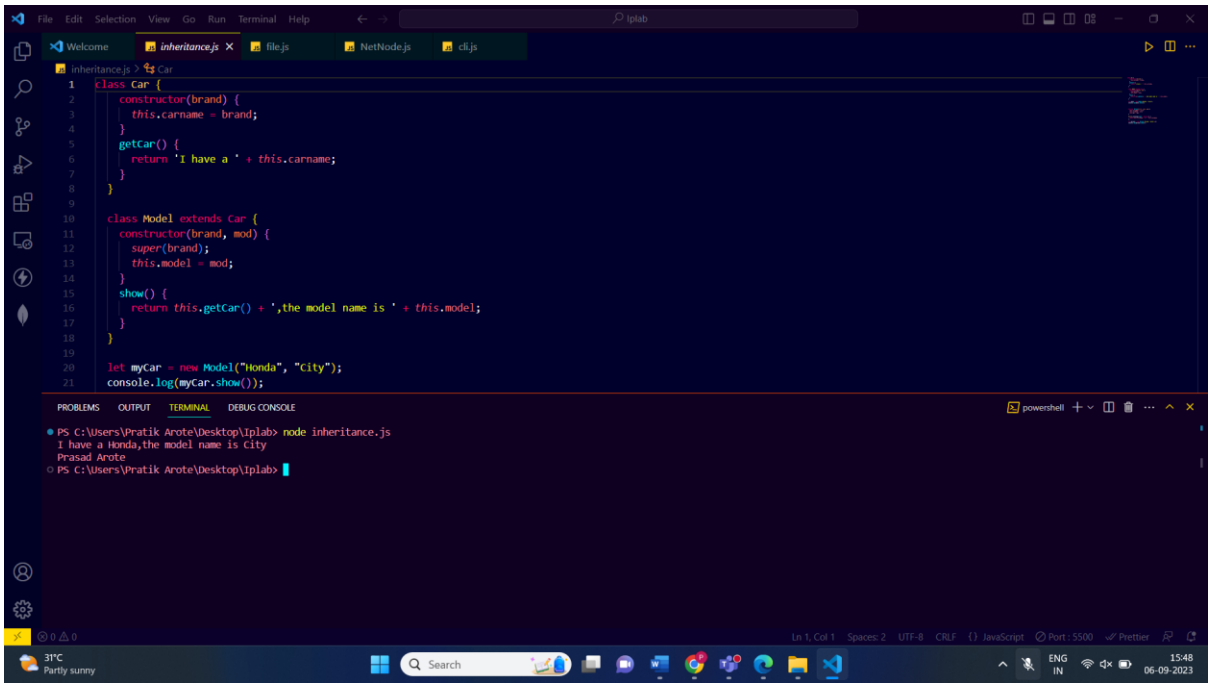
Yield: The yield keyword is used inside a generator function to yield a value and pause the function's execution. When the generator is resumed, it continues from where it left off.

Iterator: Generator functions automatically return an iterator, making them iterable.









CONCLUSION:

Here we studied about iterators, generators and inheritance in JavaScript.

Assignment 5a

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

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

THEORY:

ARROW FUNCTIONS

An arrow function in JavaScript is a concise way to write function expressions. Arrow functions were introduced in ECMAScript 6 (ES6) and provide a more compact syntax compared to traditional function expressions. They are especially useful for writing short, simple functions.

Here's the basic syntax of an arrow function:

```
const functionName = (parameters) => {
  // Function body
};
```

Key characteristics of arrow functions:

No "this" Binding: Arrow functions do not have their own this. Instead, they inherit the this value from their containing (lexical) context. This behaviour is especially useful in callback functions, where the context of this can be different in traditional functions.

Shorter Syntax: Arrow functions offer a shorter and more concise syntax for writing functions, which can be especially beneficial for one-liners.

Implicit Return: If an arrow function consists of only one expression, you can omit the curly braces {} and the return keyword. The result of the expression will be automatically returned.

DOM Manipulation:

What is DOM? The DOM is a representation of the structure of an HTML document as a tree-like structure in memory. Each element in an HTML document is represented as a node in the DOM tree, and you can think of the DOM as an API for interacting with these nodes.

DOM Manipulation involves using JavaScript to access, create, modify, or delete elements and attributes in the DOM tree. It allows you to change the content and structure of a web page dynamically without requiring a full page reload.

Common Tasks:

Accessing DOM elements: You can use methods like getElementById, querySelector, and getElementsByClassName to access elements by their IDs, CSS selectors, or classes.

Modifying content: You can change the text or HTML content of elements using properties like textContent and innerHTML.

Adding and removing elements: You can create new elements with the createElement method and add or remove elements with appendChild, removeChild, or other methods.

Modifying attributes: You can change attributes like src, href, or class using JavaScript.

Event handling: You can attach event listeners to DOM elements to respond to user interactions, such as clicks, keypresses, or form submissions.

CSS Manipulation:

What is CSS? CSS is a style sheet language used for describing the presentation and layout of HTML documents. It defines how elements on a web page should be styled, including properties like colors, fonts, margins, padding, and positioning.

CSS Manipulation involves changing the styles applied to DOM elements dynamically using JavaScript. This can be done to respond to user interactions, create animations, or modify the appearance of elements based on various conditions.

Common Tasks:

Changing styles: You can modify the CSS properties of elements directly using JavaScript. For example, you can change the backgroundColor, fontSize, or display properties.

Adding and removing classes: Instead of modifying individual CSS properties, you can add or remove CSS classes from elements. This is a more organized way to manage styles, especially for complex applications.

Toggle classes: You can toggle the presence of a class on an element to switch between different styles or states.

Creating animations: You can use CSS transitions and animations to create smooth visual effects and transitions.

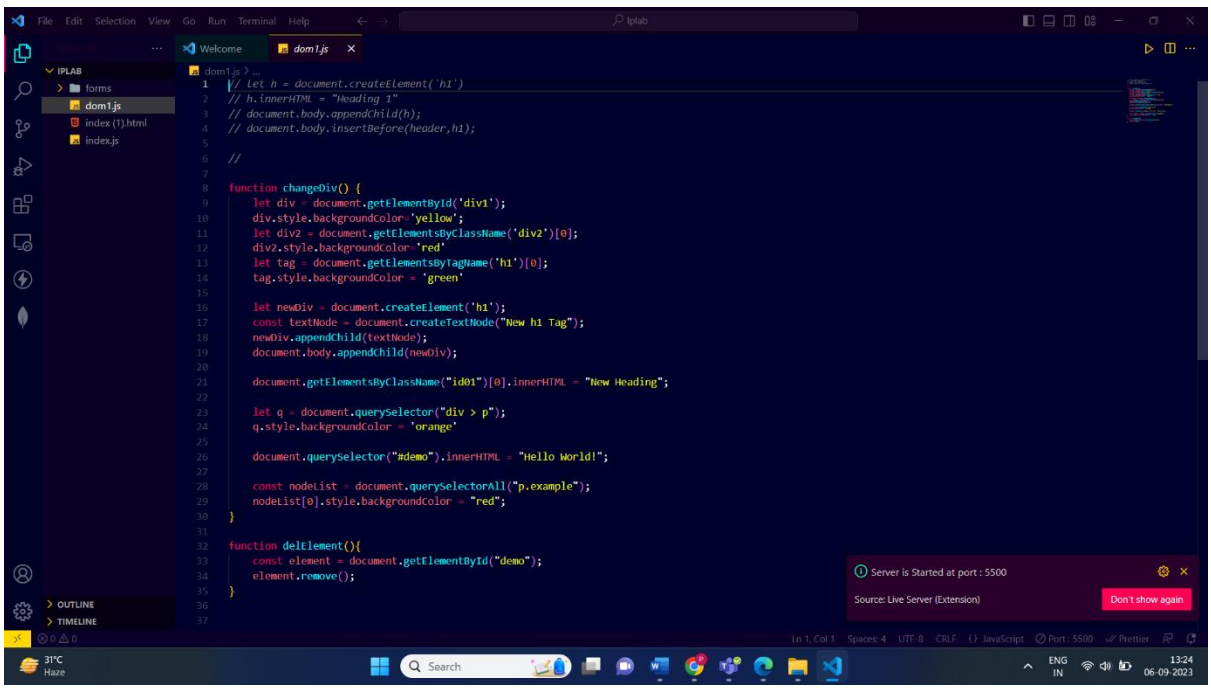
Media queries: You can change styles based on the screen size or other media-specific conditions using CSS media queries.

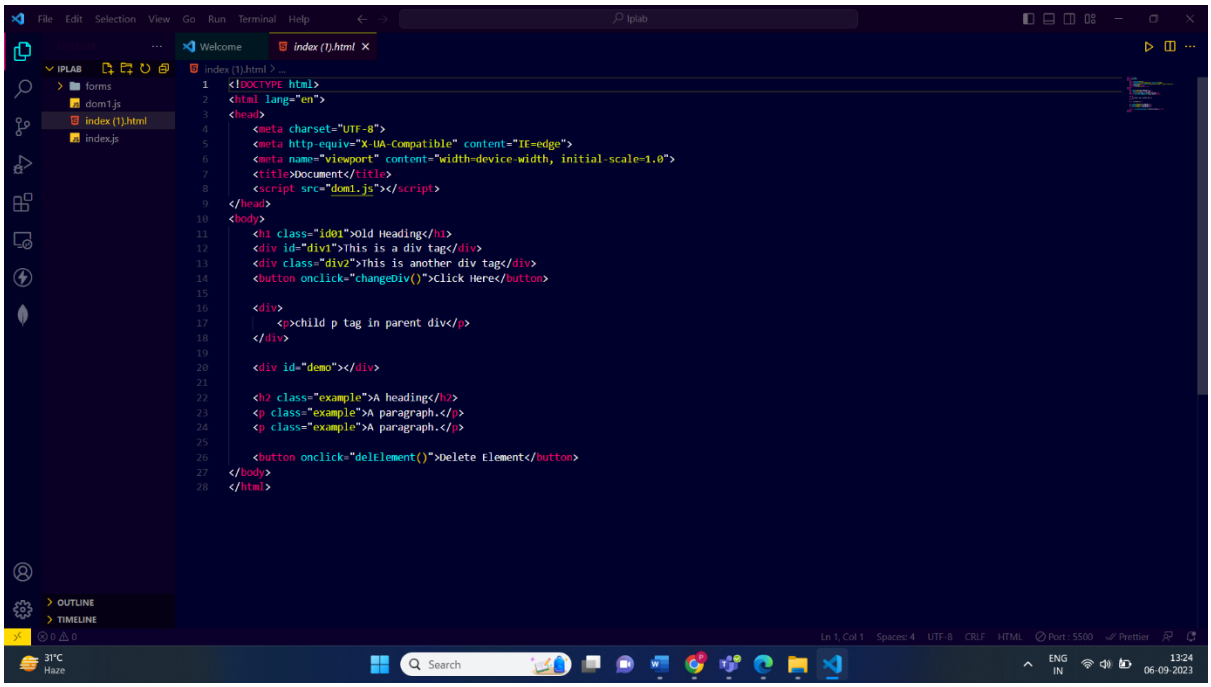
Combining DOM and CSS Manipulation:

In web development, DOM and CSS manipulation often go hand in hand. You can use DOM manipulation to access and modify elements and then use CSS manipulation to change their styles. For example, you might use DOM manipulation to toggle a class on a button click, and the CSS associated with that class defines how the element looks in its toggled state. This combination allows for highly interactive and visually appealing web applications.

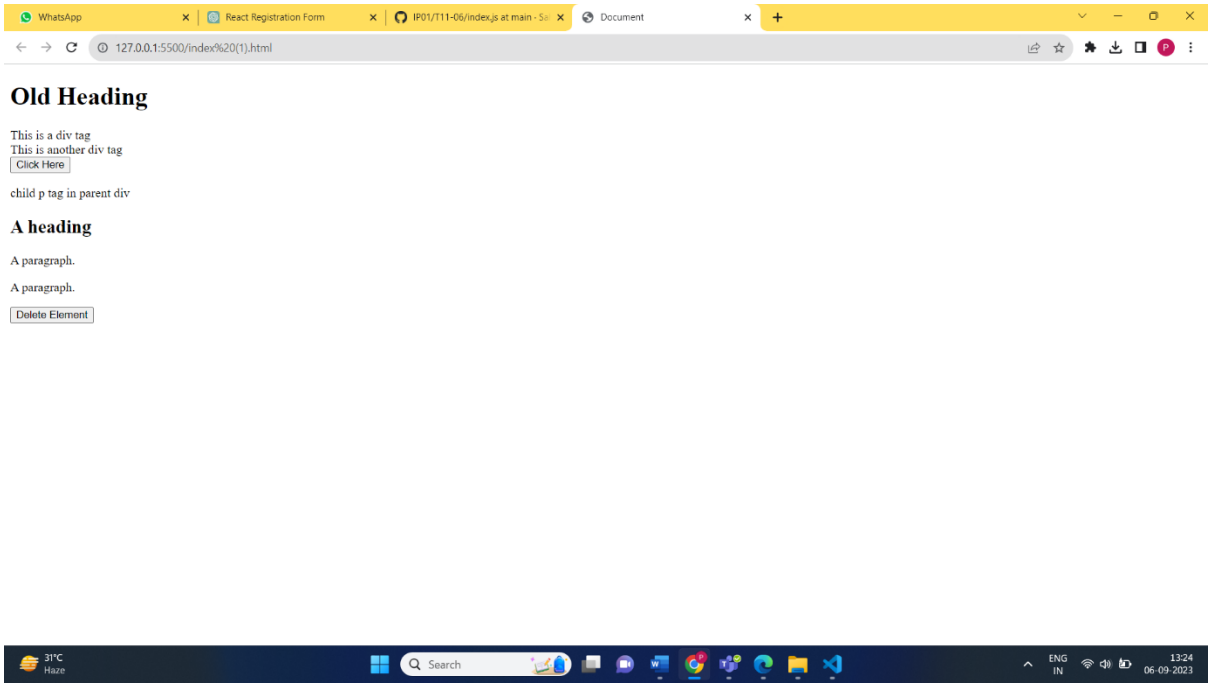
ARROW FUNCTION

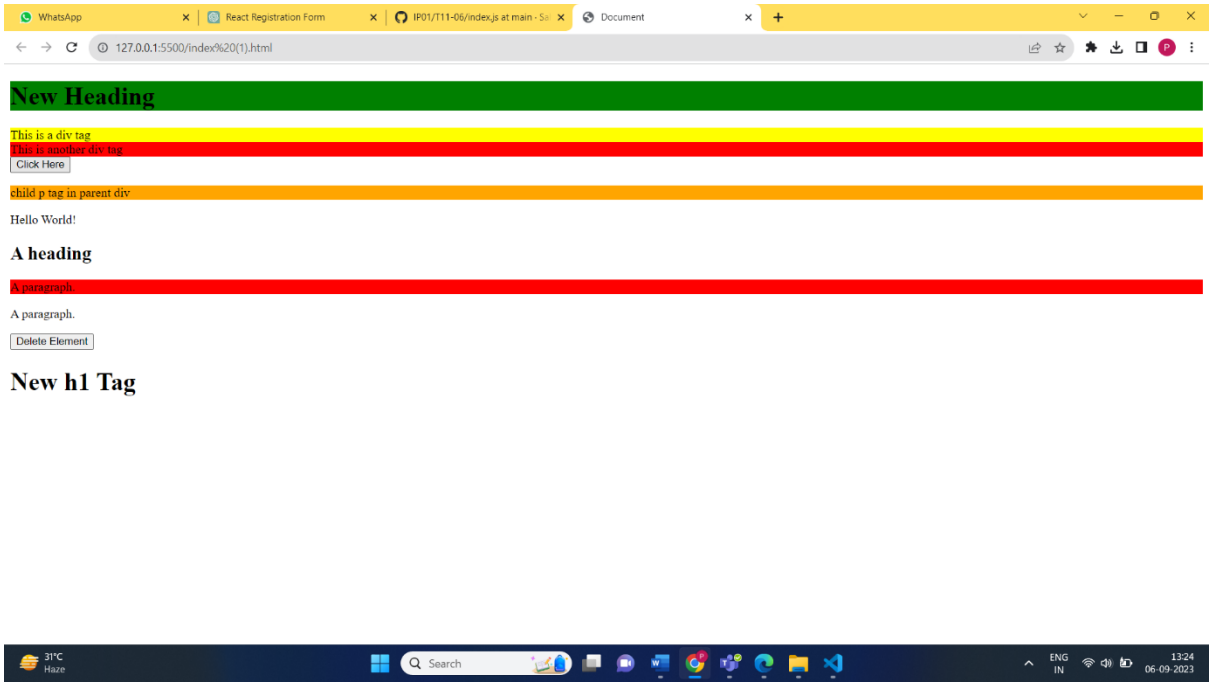
```
// Traditional function expression
const add = function (a, b) {
  return a + b;
};
// Arrow function equivalent
const addArrow = (a, b) => a + b;
// Arrow function with implicit return
const double = (number) => number * 2;
// Arrow function with no parameters
const sayHello = () => console.log("Hello!");
// Arrow function in map() method
const numbers = [1, 2, 3, 4, 5];
const squared = numbers.map((number) => number ** 2);
// Arrow function in setTimeout()
setTimeout(() => {
  console.log("Timeout complete!");
}, 1000);
```

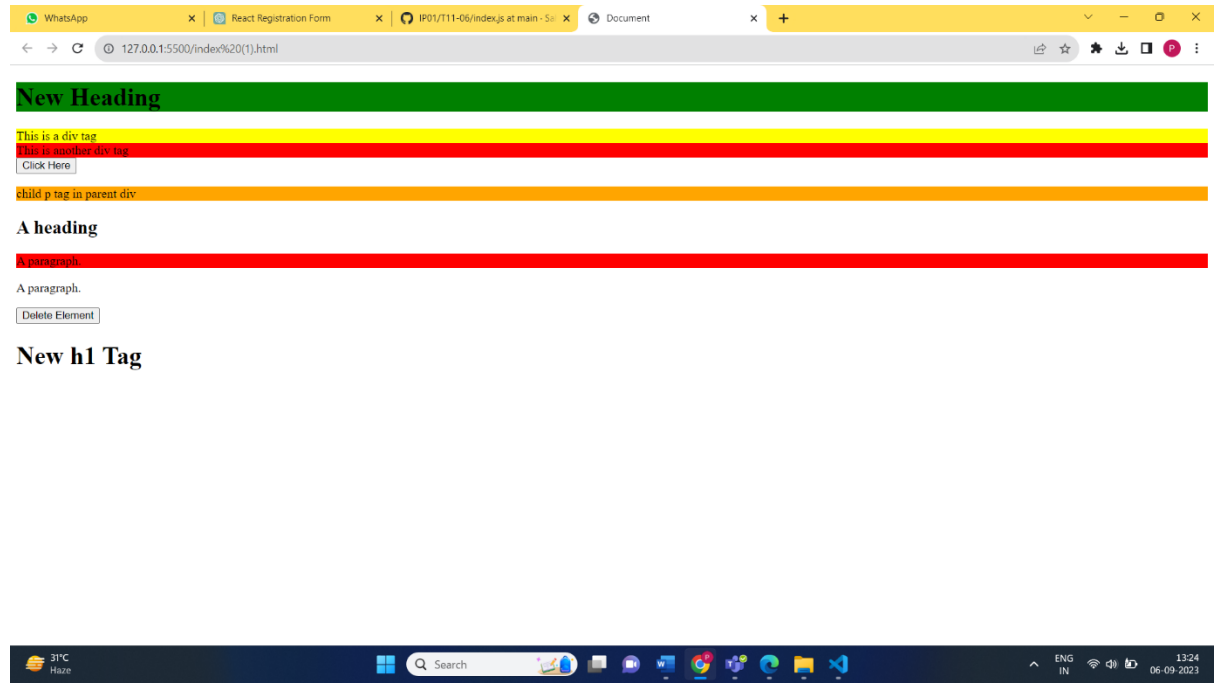




```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script src="dom1.js"></script>
</head>
<body>
    <h1 class="id01">Old Heading</h1>
    <div id="div1">This is a div tag</div>
    <div class="div2">This is another div tag</div>
    <button onclick="changeDiv()">Click Here</button>

    <div>
        <p>child p tag in parent div</p>
    </div>

    <div id="demo"></div>

    <h2 class="example">A heading</h2>
    <p class="example">A paragraph.</p>
    <p class="example">A paragraph.</p>

    <button onclick="delElement()">Delete Element</button>
</body>
</html>
```







CONCLUSION:

Here we studied and implemented DOM manipulation and CSS Manipulation in JS.

Assignment 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.

LO4: To expose students to JavaScript to make web pages interactive**.**

THEORY:

Promises in JavaScript are a way to handle asynchronous operations more effectively and avoid callback hell (a situation where callbacks are nested within callbacks). They provide a cleaner and more structured way to work with asynchronous code. Promises represent a value that might not be available yet but will be at some point in the future.

Here are the key concepts and features of Promises in JavaScript:

States:

Pending: The initial state when a Promise is created, and the operation is still in progress.

Fulfilled (Resolved): The state when the operation completed successfully, and the Promise holds a resolved value.

Rejected: The state when the operation failed, and the Promise holds a reason for the failure.

Creating a Promise:

A Promise is created using the Promise constructor, which takes a single function (the executor) as its argument. This function receives two callbacks, resolve and reject, which can be called to transition the Promise to a fulfilled or rejected state, respectively.

```
const myPromise = new Promise((resolve, reject) => {
  // Perform an asynchronous operation
  // If successful, call resolve with the result
  // If an error occurs, call reject with an error
});
```

Consuming Promises:

You can consume Promises using the .then() and .catch() methods. The .then() method is used for handling the successful fulfillment of a Promise, and the .catch() method is used for handling any errors (rejections).

```
myPromise
  .then((result) => {
    // Handle the resolved value here
  })
  .catch((error) => {
```

```
  // Handle errors here
});
```

Chaining Promises:

Promises can be chained together to perform multiple asynchronous operations sequentially or in a specific order. This is achieved by returning a new Promise from the then method.

```
myPromise
  .then((result) => {
    // Perform another asynchronous operation and return a new Promise
    return anotherAsyncOperation(result);
  })
  .then((result2) => {
    // Handle the result of the second operation
  })
  .catch((error) => {
    // Handle errors in any step of the chain
  });
```

Promise.all:

Promise.all() is a utility function that takes an array of Promises and returns a new Promise that fulfills when all the Promises in the array have been fulfilled, or rejects if any of them is rejected. It's useful for parallelizing multiple asynchronous operations and waiting for all of them to complete.

```
const promises = [promise1, promise2, promise3];
Promise.all(promises)
  .then((results) => {
    // Handle the results when all promises are fulfilled
  })
  .catch((error) => {
    // Handle any error from any promise
  });
```

Promise.race:

Promise.race() is similar to Promise.all(), but it fulfills or rejects as soon as one of the Promises in the array fulfills or rejects. It's useful when you want to race multiple asynchronous operations and only care about the first one that completes.

const promises = [promise1, promise2, promise3];

Promise.race(promises)

.then((firstResult) => {

// Handle the result of the first promise to complete

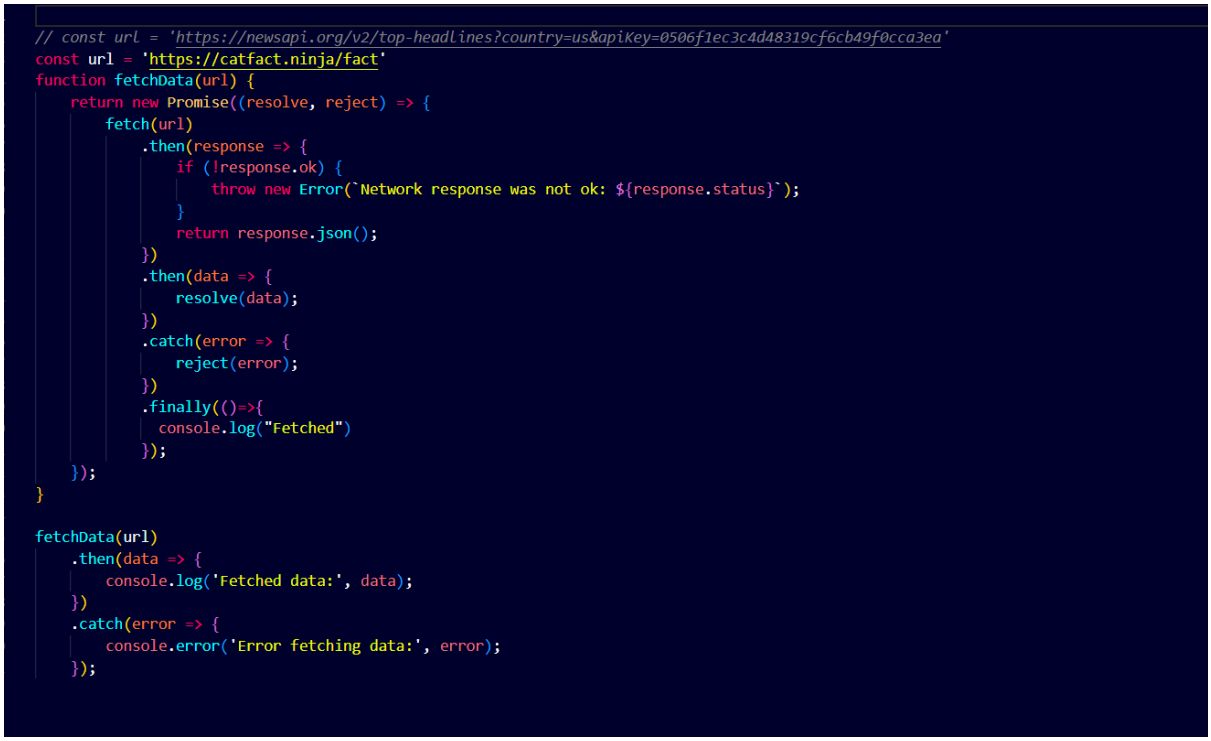
})

.catch((error) => {

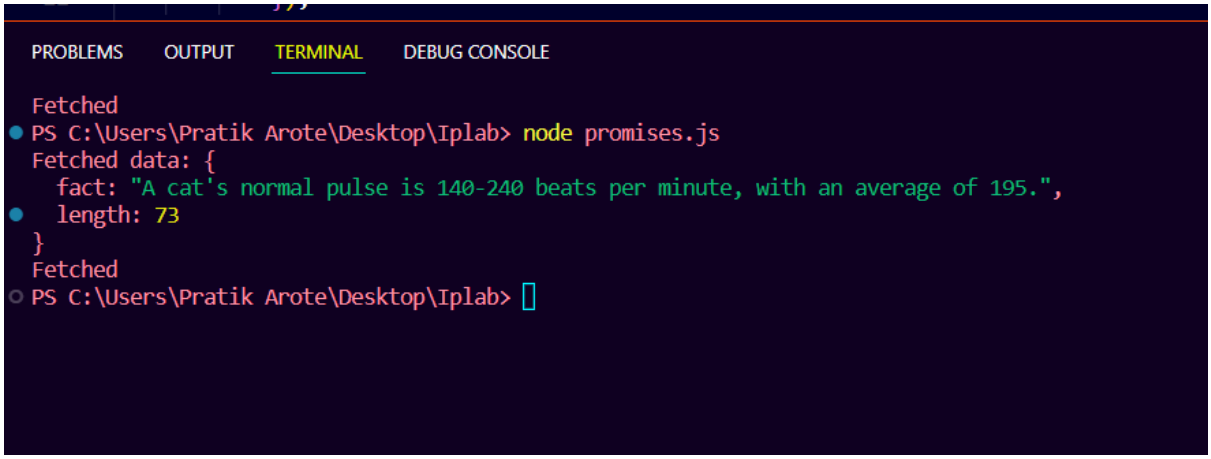
// Handle any error from any promise

});

Promises provide a structured and more readable way to work with asynchronous code in JavaScript, making it easier to handle errors and coordinate multiple asynchronous operations. They have become a fundamental building block for handling asynchronous tasks in modern JavaScript applications.



```
// const url = 'https://newsapi.org/v2/top-headlines?country=us&apiKey=0506f1ec3c4d48319cf6cb49f0cca3ea'
const url = 'https://catfact.ninja/fact'
function fetchData(url) {
    return new Promise((resolve, reject) => {
        fetch(url)
            .then(response => {
                if (!response.ok) {
                    throw new Error(`Network response was not ok: ${response.status}`);
                }
                return response.json();
            })
            .then(data => {
                resolve(data);
            })
            .catch(error => {
                reject(error);
            })
            .finally(()=>{
              console.log("Fetched")
            });
    });
}

fetchData(url)
    .then(data => {
        console.log('Fetched data:', data);
    })
    .catch(error => {
        console.error('Error fetching data:', error);
    });
```



CONCLUSION:

Here, we studied about promises in JavaScript.

Assignment 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 colour.

LO5: To orient students to React for developing frontend application.

THEORY:

In React, "state" and "props" are two fundamental concepts that play a crucial role in building dynamic and interactive user interfaces. They are used to manage and pass data within a React application. Let's explore each concept in detail:

State:

State is a built-in feature in React that allows components to store and manage their own local data. Each component can have its own state, which is used to keep track of information that can change over time. State is mutable and can be updated using the setState method. Here are some key points about state in React:

Local to Component: State is specific to a component and cannot be directly accessed or modified by other components. This encapsulation ensures data isolation and prevents unintended side effects.

Initialization: State is typically initialized in the component's constructor or by using the useState hook in functional components.

Updating State: To update the state, you should never directly modify it. Instead, use the setState method (in class components) or the state updater function (in functional components) provided by React. This triggers a re-render of the component with the updated state.

Asynchronous Updates: State updates may be asynchronous, so React batches multiple state updates for performance reasons. This can lead to unexpected behaviour when relying on the current state immediately after updating it.

Props:

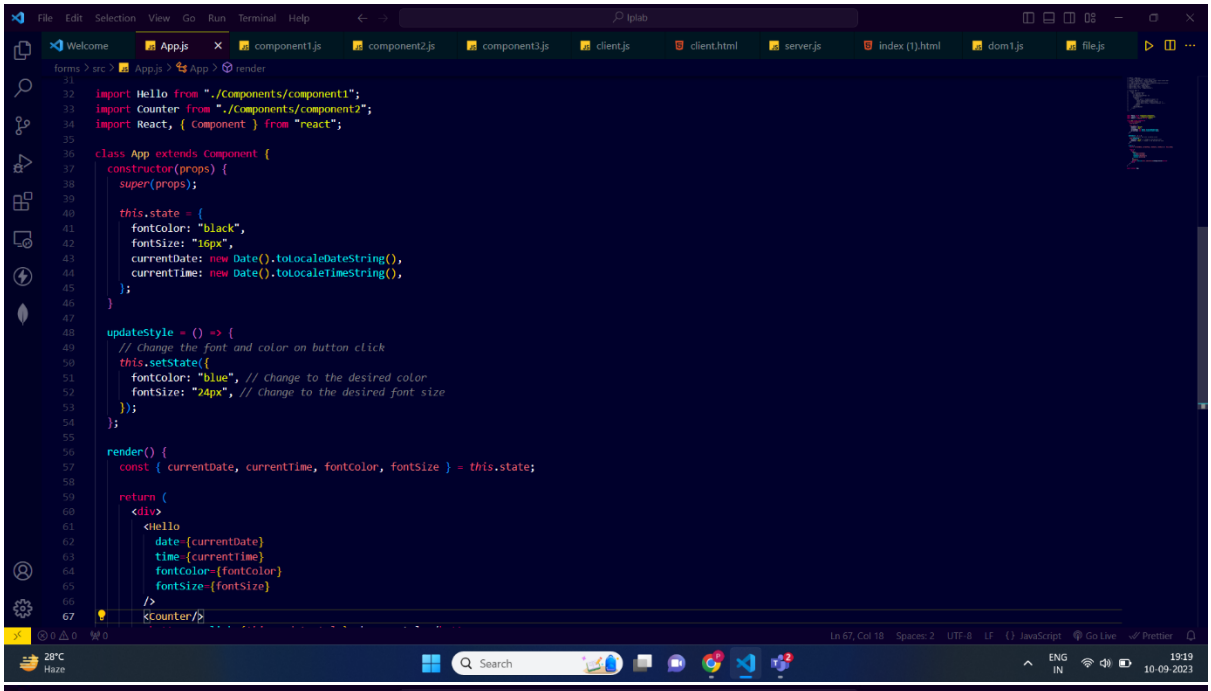
Props (short for "properties") are a way to pass data from a parent component to its child components. They are read-only and provide a mechanism for components to communicate with each other. Props are essential for creating reusable and composable components. Here are some key points about props in React:

Unidirectional Data Flow: Data flows from parent to child components through props. Child components cannot directly modify their props; they are considered immutable within the component receiving them.

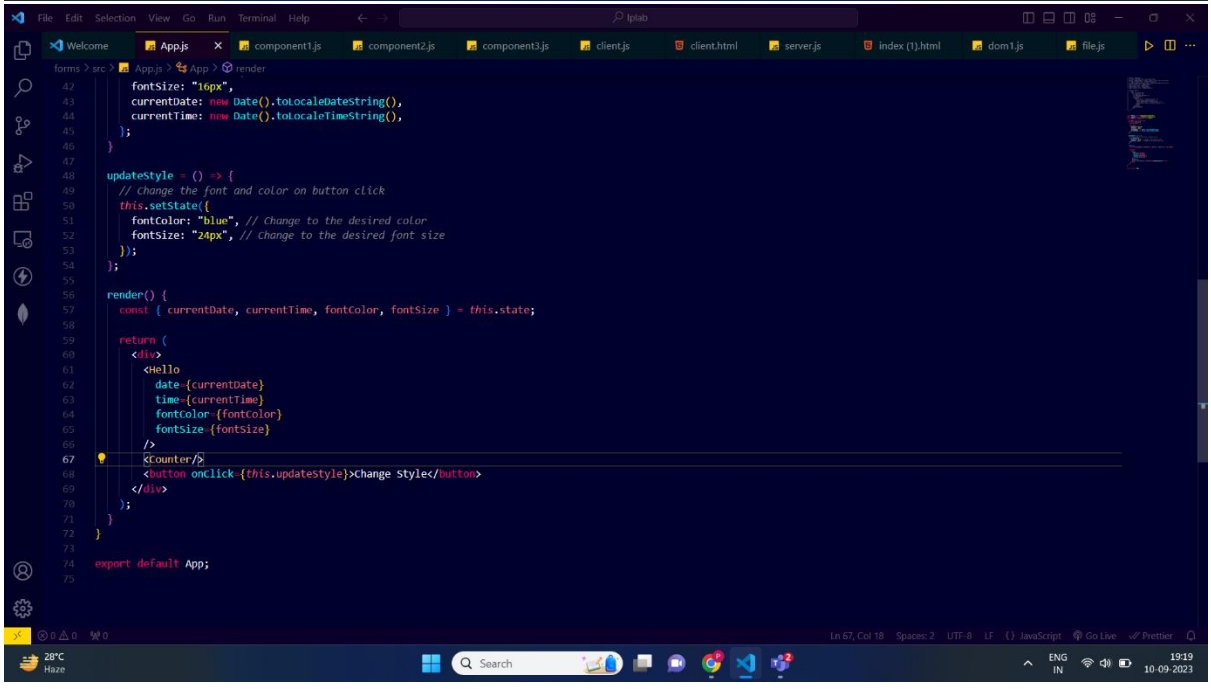
T11 PRASAD AROTE 06

Functional Components: In functional components, props are passed as an argument to the component's function.

Class Components: In class components, props are accessed via this.props.

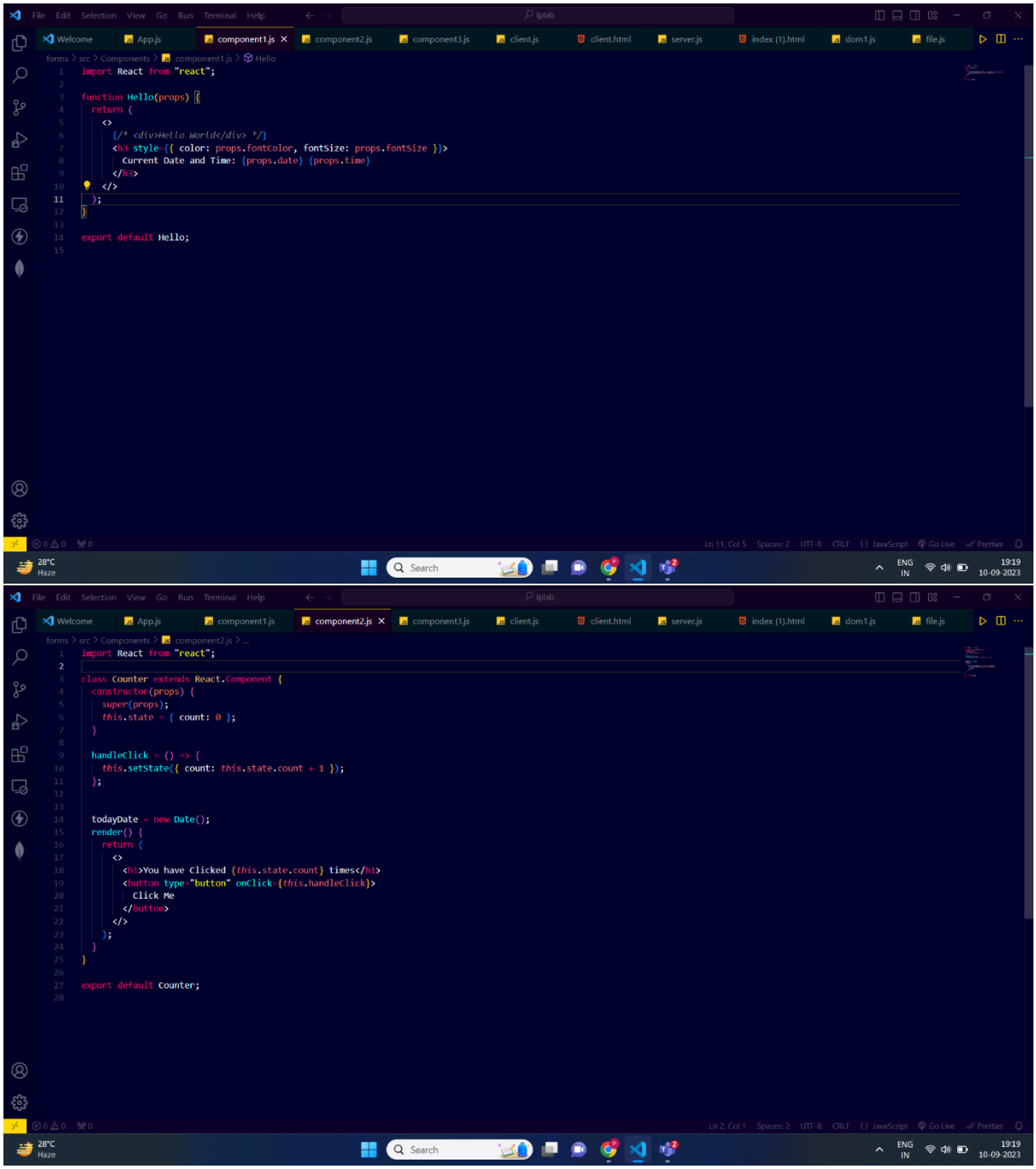


```
import Hello from "./Components/component1";
import Counter from "./Components/component2";
import React, { Component } from "react";

class App extends Component {
  constructor(props) {
    super(props);

    this.state = {
      fontColor: "black",
      fontSize: "16px",
      currentDate: new Date().toLocaleDateString(),
      currentTime: new Date().toLocaleTimeString(),
    };
  }

  updateStyle = () => {
    // Change the font and color on button click
    this.setState({
      fontColor: "blue", // Change to the desired color
      fontSize: "24px", // Change to the desired font size
    });
  };

  render() {
    const { currentDate, currentTime, fontColor, fontSize } = this.state;

    return (
      <div>
        <Hello
          date={currentDate}
          time={currentTime}
          fontColor={fontColor}
          fontSize={fontSize}
        />
        <Counter/>
```



```
      fontSize: "16px",
      currentDate: new Date().toLocaleDateString(),
      currentTime: new Date().toLocaleTimeString(),
    };
  }

  updateStyle = () => {
    // Change the font and color on button click
    this.setState({
      fontColor: "blue", // Change to the desired color
      fontSize: "24px", // Change to the desired font size
    });
  };

  render() {
    const { currentDate, currentTime, fontColor, fontSize } = this.state;

    return (
      <div>
        <Hello
          date={currentDate}
          time={currentTime}
          fontColor={fontColor}
          fontSize={fontSize}
        />
        <Counter/>
        <button onClick={this.updateStyle}>Change Style</button>
      </div>
    );
  }
}

export default App;
```

T11 PRASAD AROTE 06

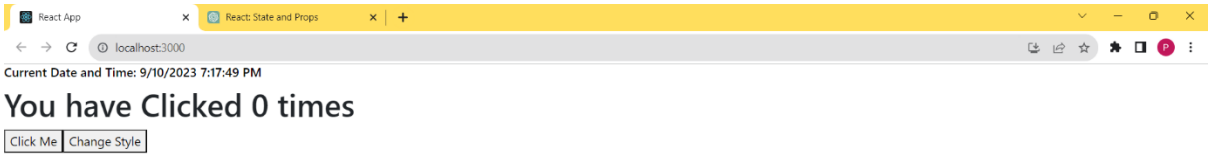


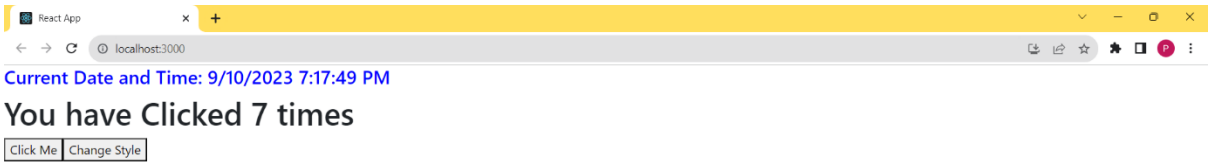
```
import React from "react";

function Hello(props) {
  return (
    <>
      {/* <div>Hello World</div> */}
      <h3 style={{ color: props.fontColor, fontSize: props.fontSize }}>
        Current Date and Time: {props.date} {props.time}
      </h3>
    </>
  );
}

export default Hello;
```



```
import React from "react";

class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
  }

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


  todayDate = new Date();
  render() {
    return (
      <>
        <h1>You have Clicked {this.state.count} times</h1>
        <button type="button" onClick={this.handleClick}>
          Click Me
        </button>
      </>
    );
  }
}

export default Counter;
```

OUTPUT:





CONCLUSION:

Hence, we learnt the concept of props and states in react. Also, we have passed the date as props and changed the style of it on button click using states.

Assignment 6b

AIM: WAP to implement the concept of forms and events. Create a React JS registration Form consisting of textbox, textarea, selection input, check box, radio button, submit button and reset button handling onSubmit, onClick and keyDown events.

LO5: To orient students to React for developing frontend application.

THEORY:

React Forms:

Controlled Components:

In React, form elements like <input>, <select>, and <textarea> are typically turned into controlled components. A controlled component is an input element whose value is controlled by the React state. Key points about controlled components include:

The component's value is stored in the component's state, allowing React to control and track the input's value.

Changes to the input's value are reflected in the component's state via event handlers.

Controlled components make it easier to perform actions such as validation and synchronization between different form elements.

Handling Form Submission (onSubmit):

The onSubmit event handler is used to handle form submissions in React. It is assigned to the <form> element and is triggered when the user submits the form.

The onSubmit handler is often used to perform actions like data validation, data submission to a server, or navigating to another page after successful form submission.

Typically, the event.preventDefault() method is used to prevent the default browser behavior of submitting the form and refreshing the page. This allows developers to control the form submission process programmatically.

Event Handling in React:

Event Handlers:

React uses event handlers to capture and respond to various user interactions, such as mouse clicks, keyboard presses, and form submissions. Common event handlers in React include:

onClick: Used for handling mouse clicks on elements like buttons and links.

onMouseEnter and onMouseLeave: Used for detecting when the mouse pointer enters or leaves an element's area.

onChange: Used for tracking changes to form input elements, like text fields, checkboxes, and radio buttons.

onKeyDown, onKeyPress, and onKeyUp: Used for handling keyboard events.

Event Object:

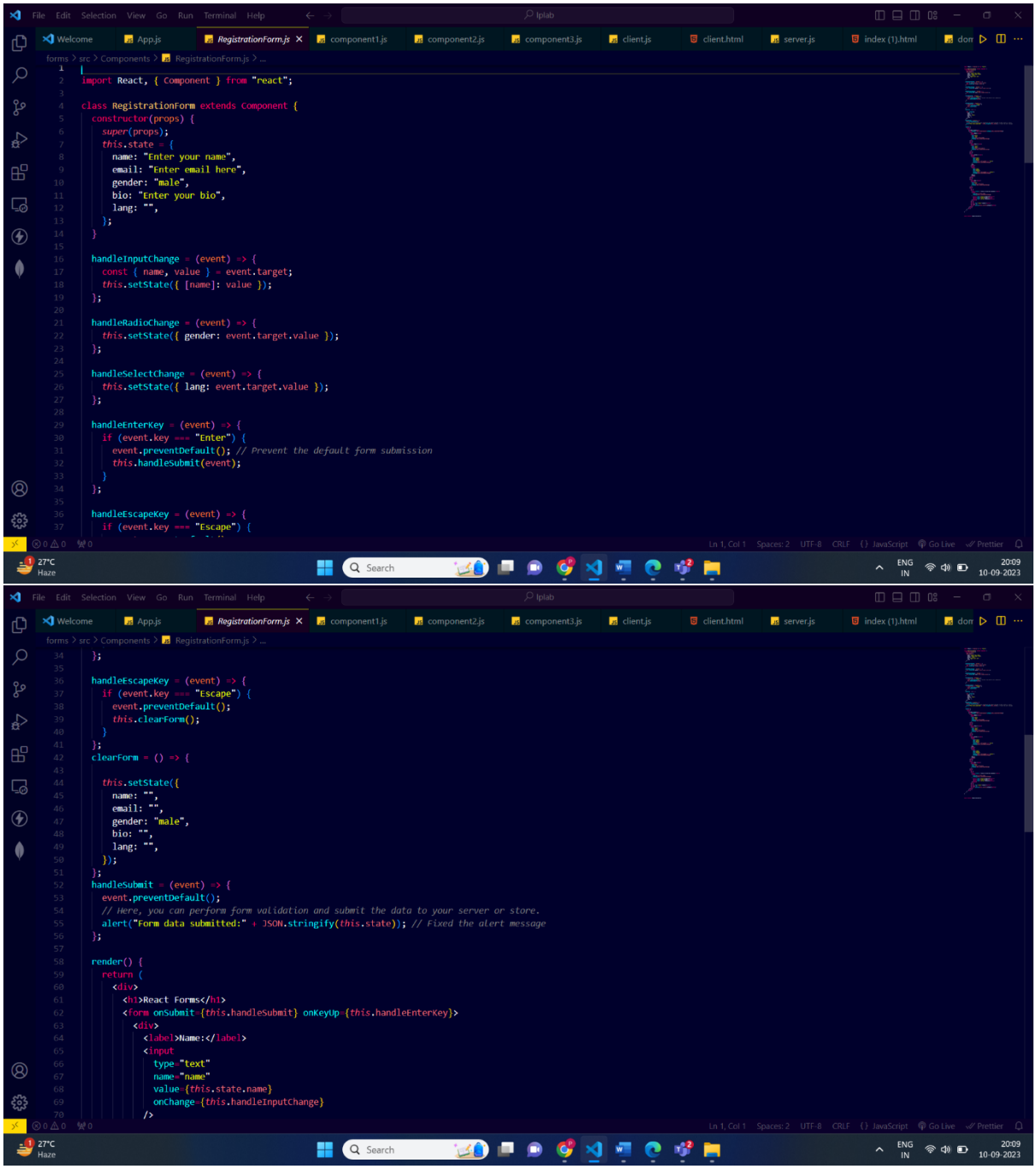
When an event occurs, React passes an event object to the event handler function. This object contains information about the event, such as the type of event, target element, and event-specific properties.

Event objects can be accessed as the first argument in event handler functions, allowing you to extract relevant data and respond accordingly.

Event Propagation:

React follows the event propagation model, which means events propagate from the innermost component to the outermost component in the component tree.

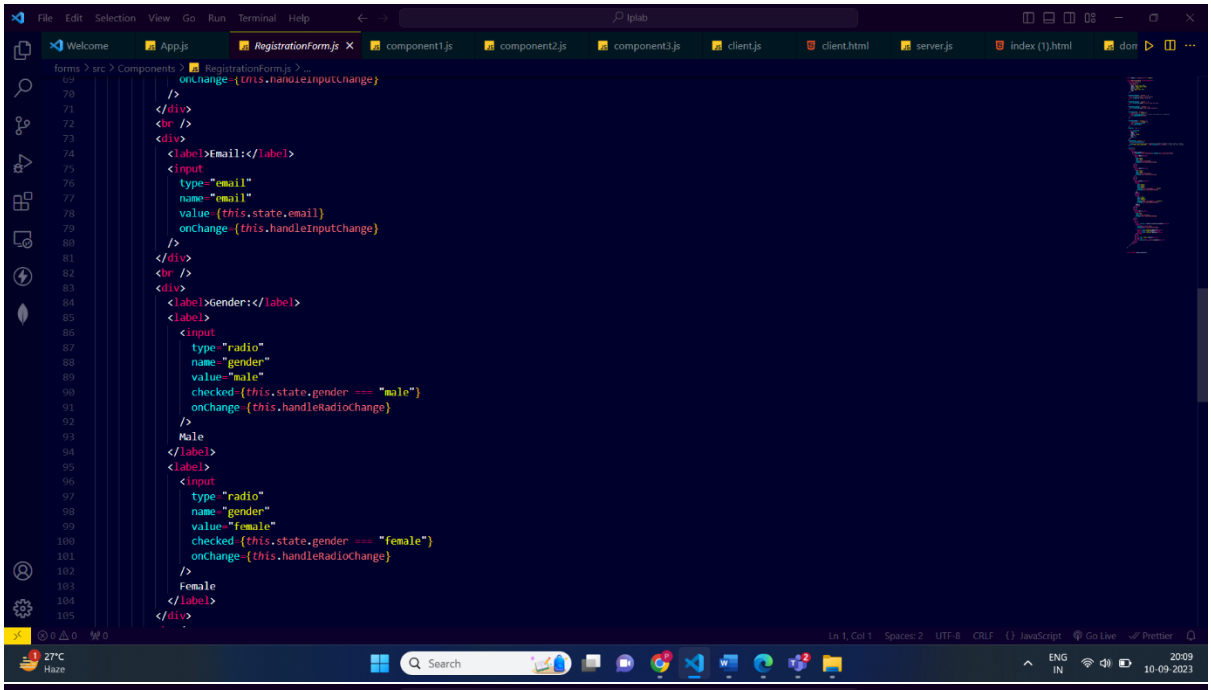
Event propagation can be controlled using event methods like stopPropagation(), which prevents the event from further propagation, and preventDefault(), which stops the default behavior of an event.



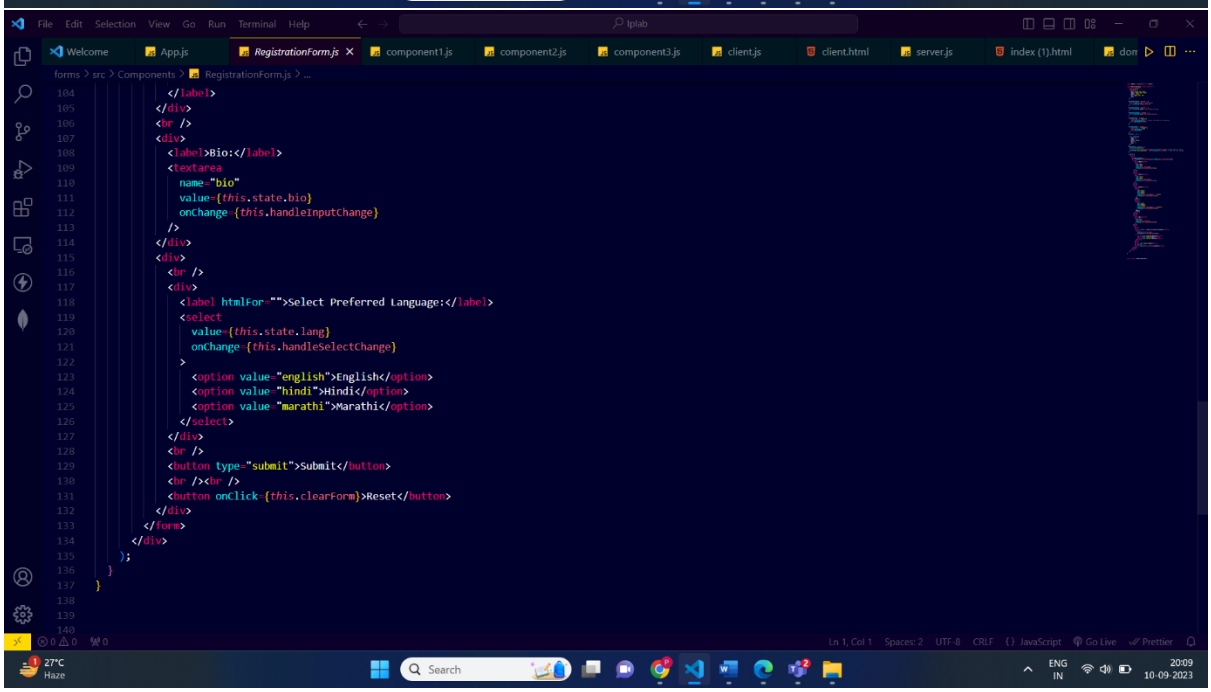
```
import React, { Component } from "react";

class RegistrationForm extends Component {
  constructor(props) {
    super(props);
    this.state = {
      name: "Enter your name",
      email: "Enter email here",
      gender: "male",
      bio: "Enter your bio",
      lang: "",
    };
  }

  handleInputChange = (event) => {
    const { name, value } = event.target;
    this.setState({ [name]: value });
  };

  handleRadioChange = (event) => {
    this.setState({ gender: event.target.value });
  };

  handleSelectChange = (event) => {
    this.setstate({ lang: event.target.value });
  };

  handleEnterKey = (event) => {
    if (event.key === "Enter") {
      event.preventDefault(); // Prevent the default form submission
      this.handleSubmit(event);
    }
  };

  handleEscapeKey = (event) => {
    if (event.key === "Escape") {
```



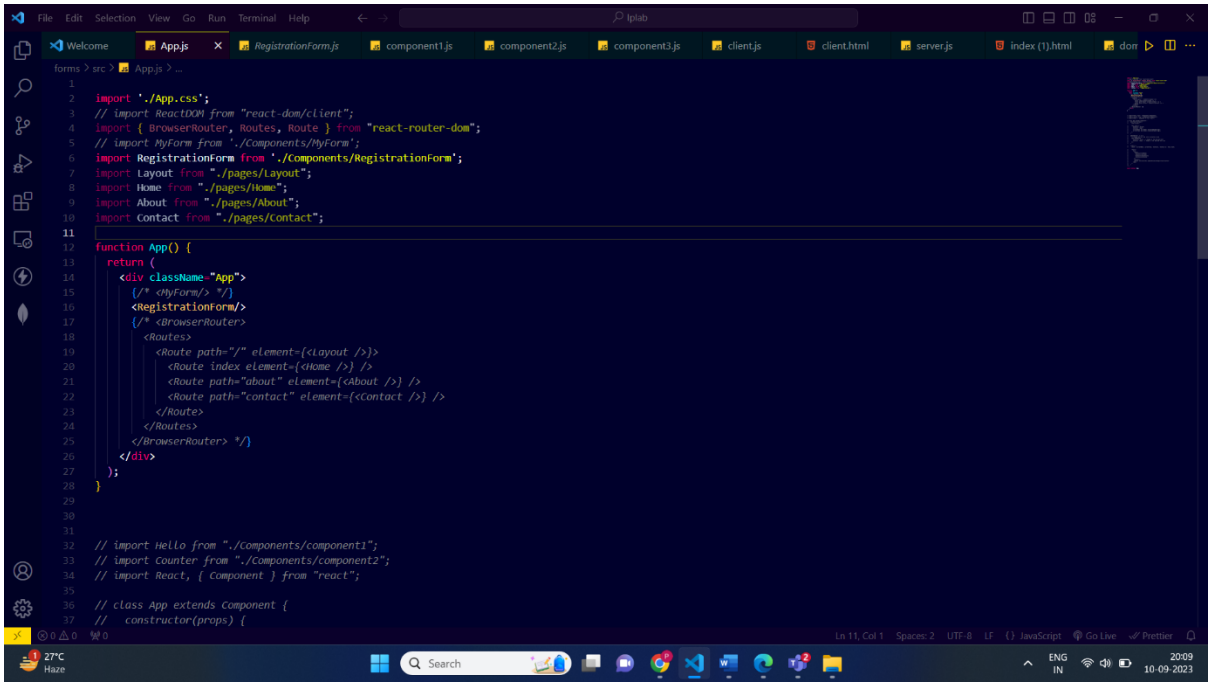
```
  };

  handleEscapeKey = (event) => {
    if (event.key === "Escape") {
      event.preventDefault();
      this.clearForm();
    }
  };
  clearForm = () => {

    this.setState({
      name: "",
      email: "",
      gender: "male",
      bio: "",
      lang: "",
    });
  };
  handleSubmit = (event) => {
    event.preventDefault();
    // Here, you can perform form validation and submit the data to your server or store.
    alert("Form data submitted:" + JSON.stringify(this.state)); // Fixed the alert message
  };

  render() {
    return (
      <div>
        <h1>React Forms</h1>
        <form onSubmit={this.handleSubmit} onKeyUp={this.handleEnterKey}>
          <div>
            <label>Name:</label>
            <input
              type="text"
              name="name"
              value={this.state.name}
              onChange={this.handleInputChange}
            />
```



```
                onChange={this.handleInputChange}
              />
            </div>
            <br />
            <div>
              <label>Email:</label>
              <input
                type="email"
                name="email"
                value={this.state.email}
                onChange={this.handleInputChange}
              />
            </div>
            <br />
            <div>
              <label>Gender:</label>
              <label>
                <input
                  type="radio"
                  name="gender"
                  value="male"
                  checked={this.state.gender === "male"}
                  onChange={this.handleRadioChange}
                />
                Male
              </label>
              <label>
                <input
                  type="radio"
                  name="gender"
                  value="female"
                  checked={this.state.gender === "female"}
                  onChange={this.handleRadioChange}
                />
                Female
              </label>
            </div>
```

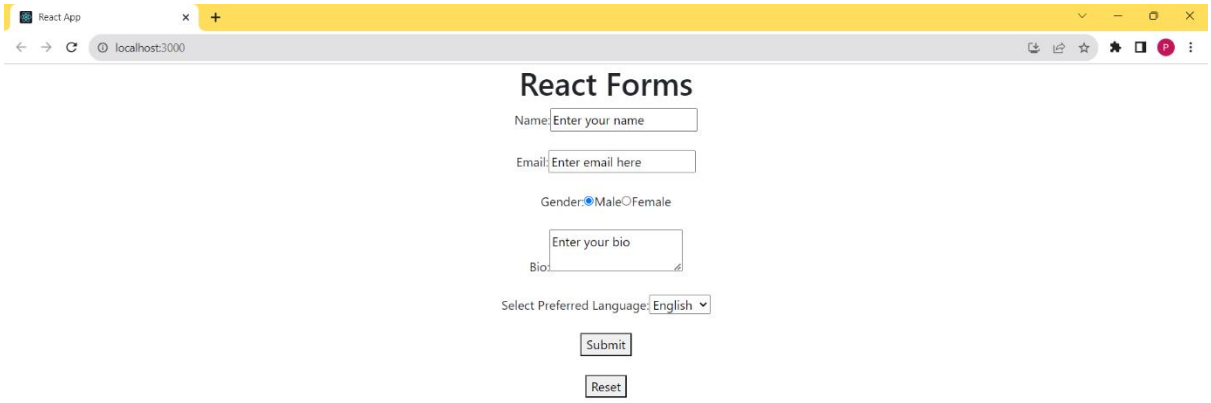


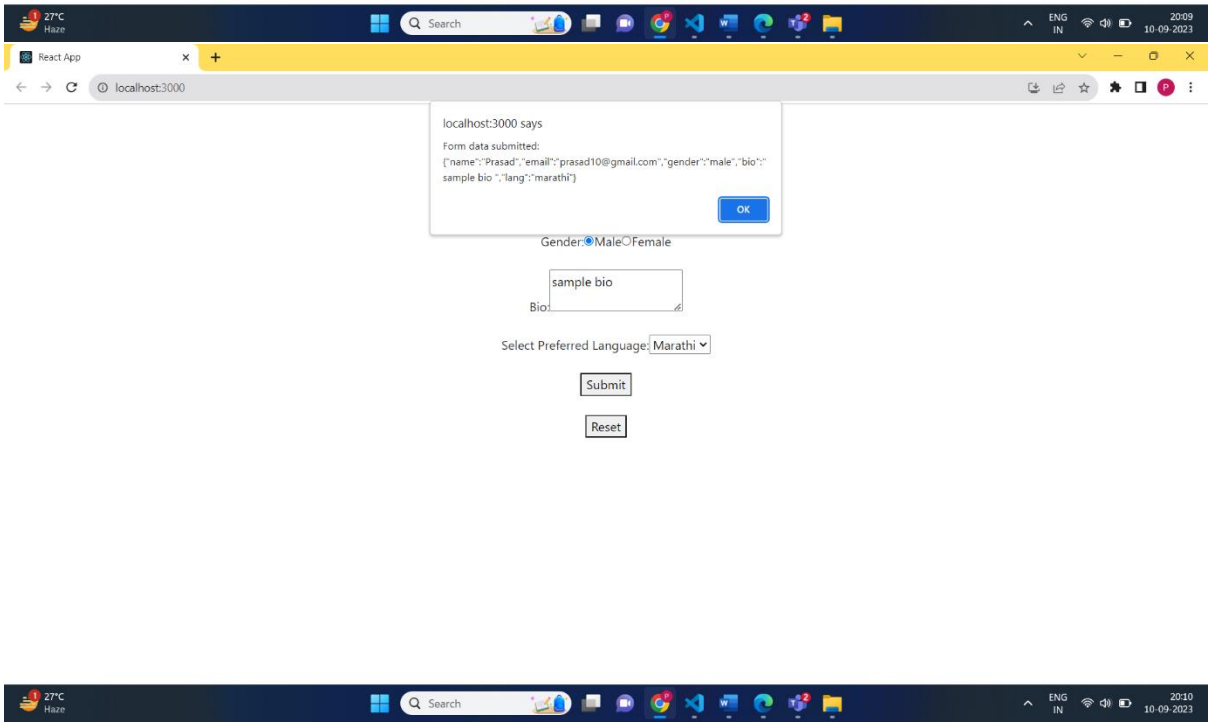
```
              </label>
            </div>
            <br />
            <div>
              <label>Bio:</label>
              <textarea
                name="bio"
                value={this.state.bio}
                onChange={this.handleInputChange}
              />
            </div>
            <div>
              <br />
              <div>
                <label htmlFor="">Select Preferred Language:</label>
                <select
                  value={this.state.lang}
                  onChange={this.handleSelectChange}
                >
                  <option value="english">English</option>
                  <option value="hindi">Hindi</option>
                  <option value="marathi">Marathi</option>
                </select>
              </div>
              <br />
              <button type="submit">Submit</button>
              <br /><br />
              <button onClick={this.clearForm}>Reset</button>
            </div>
          </form>
        </div>
      );
    }
  }
```



```
import './App.css';
// import ReactDOM from "react-dom/client";
import { BrowserRouter, Routes, Route } from "react-router-dom";
// import MyForm from './Components/MyForm';
import RegistrationForm from './Components/RegistrationForm';
import Layout from "./pages/Layout";
import Home from "./pages/Home";
import About from "./pages/About";
import Contact from "./pages/contact";

function App() {
  return (
    <div className="App">
      {/* <MyForm/> */}
      <RegistrationForm/>
      {/* <BrowserRouter>
        <Routes>
          <Route path="/" element={<Layout />}>
            <Route index element={<Home />} />
            <Route path="about" element={<About />} />
            <Route path="contact" element={<Contact />} />
          </Route>
        </Routes>
      </BrowserRouter> */}
    </div>
  );
}



// import Hello from "./Components/component1";
// import Counter from "./Components/component2";
// import React, { Component } from "react";

// class App extends Component {
//   constructor(props) {
```

OUTPUT





CONCLUSION:

Here we have understood Forms and event handling in React. We have created a form using various form elements and handled various click events and Keyboard events.

Assignment 7a

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

LO5: To orient students to React for developing frontend application.

THEORY:

In React, Router and BrowserRouter are two essential components provided by the React Router library, which is a popular routing solution for building single-page applications (SPAs). These components help manage and navigate between different views or pages within your application. Let's explore what each of these components does:

Router:

The Router component is the core component of React Router. It doesn't render anything itself but serves as a container for the routes in your application. It acts as a context provider that makes routing-related information available to other components within your app.

Typically, you'll wrap your entire application with a Router component, and it will determine how routing works for your application, including whether you use a hash-based routing or a browser history-based routing (which is what BrowserRouter uses).

BrowserRouter:

BrowserRouter is a specific type of router that uses the HTML5 History API for navigation. It allows you to create cleaner and more user-friendly URLs without the hash symbol (#) commonly seen in hash-based routing.

With BrowserRouter, you can create routes with simple, readable paths that resemble traditional URLs. For example, you can have URLs like http://example.com/about instead of http://example.com/#/about.

```jsx
import './App.css';
// import ReactDOM from "react-dom/client";
import { BrowserRouter, Routes, Route } from "react-router-dom";
// impo
// impo
import Layout from "./pages/Layout";
import Home from "./pages/Home";
import About from "./pages/About";
import Contact from "./pages/Contact";

function App() {
  return (
    <div className="App">
      {/* <RegistrationForm/> */}
      <BrowserRouter>
        <Routes>
          <Route path="/" element={<Layout />}>
            <Route index element={<Home />} />
            <Route path="about" element={<About />} />
            <Route path="contact" element={<Contact />} />
          </Route>
        </Routes>
      </BrowserRouter>
    </div>
  );
}
```

```
(alias) const Layout: () => React.JSX.Element
import Layout
```

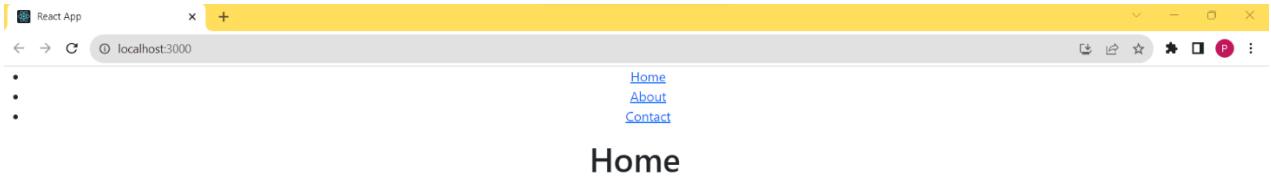
```jsx
import { Outlet, Link } from "react-router-dom";

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

      <Outlet />
    </>
  );
};

export default Layout;
```

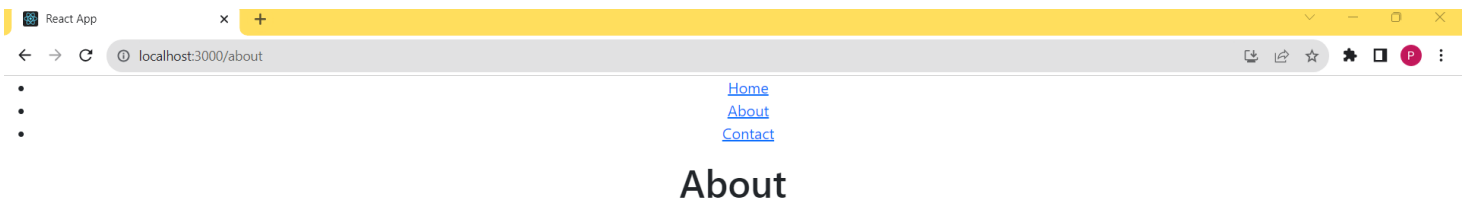
```jsx
const About = () => {
  return <h1>About</h1>;
};

export default About;
```

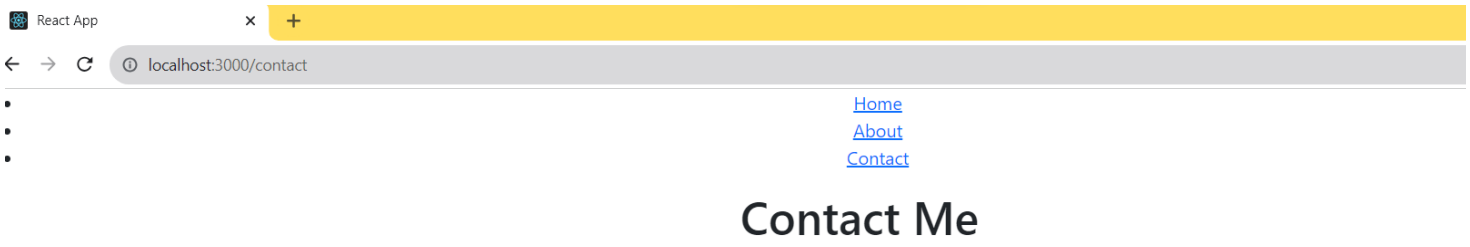
```jsx
const Contact = () => {
  return <h1>Contact Me</h1>;
};

export default Contact;
```

```jsx
const Home = () => {
  return <h1>Home</h1>;
};

export default Home;
```

OUTPUT







CONCLUSION:

Here we have understood the concept of Router and BrowserRouter in React and have created different pages and created different routes using React.

Assignment 7b

AIM: To implement the concept of React Hooks.

LO5: To orient students to React for developing frontend application.

THEORY:

React Hooks is a feature introduced in React 16.8 that allows you to use state and other React features in function components, which were previously stateless. Before Hooks, state management and lifecycle methods were only available in class components, making it challenging to manage state and side effects in function components. React Hooks aims to simplify state management, side effect handling, and code organization in function components.

Here's a detailed explanation of React Hooks:

State Hook (useState):

useState allows you to add state to a functional component.

It returns an array with two elements: the current state value and a function to update it.

You can have multiple useState calls to manage multiple pieces of state in a single component.

Effect Hook (useEffect):

useEffect enables you to perform side effects in function components, such as data fetching, DOM manipulation, or subscribing to external events.

It takes two arguments: a function to run the side effect and an optional array of dependencies to control when the effect runs.

Context Hook (useContext):

useContext allows you to access a React context from within a function component.

It provides a cleaner way to consume context values compared to the older Context.Consumer approach.

Reducer Hook (useReducer):

useReducer is an alternative to useState for managing more complex state logic.

It takes a reducer function and an initial state, returning the current state and a dispatch function.

It's especially useful when the state transitions depend on the previous state or involve multiple sub-values.

Custom Hooks:

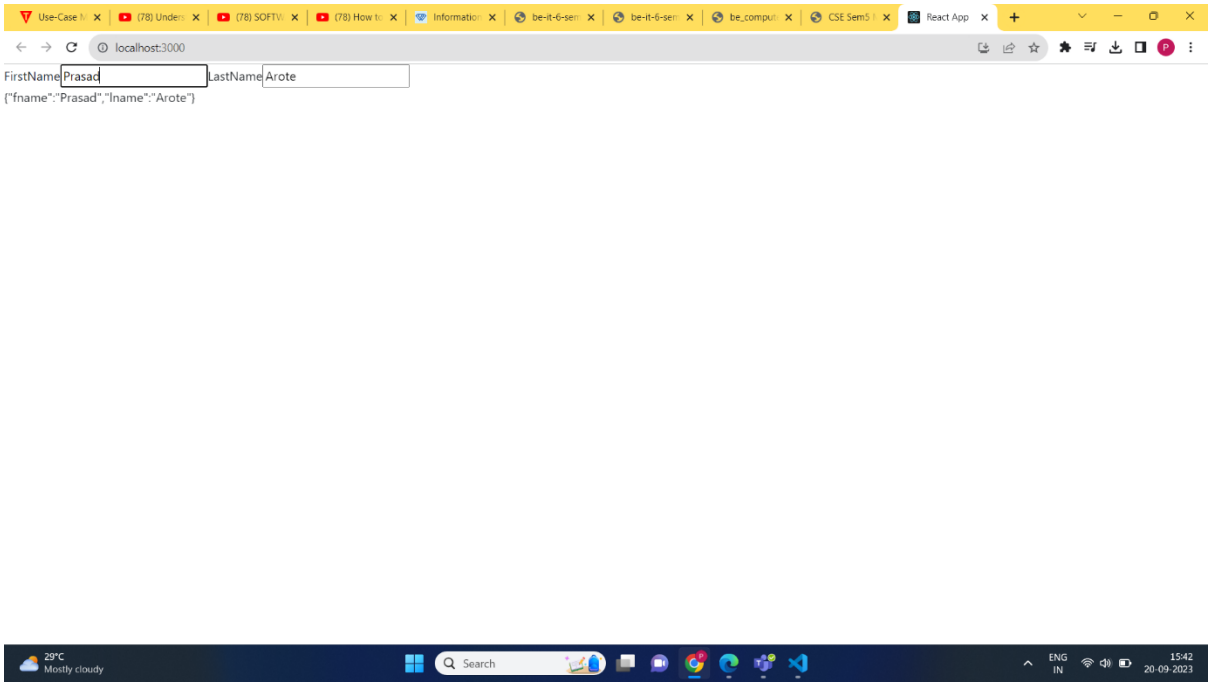
Custom Hooks allow you to extract reusable logic from your components. You can create custom hooks to share stateful logic across different components.

Custom hooks should start with the word "use" to follow the hook naming convention.

Ref Hook (useRef):

useRef allows you to create mutable references to DOM elements or any value that persists across renders.

It's often used to access or manipulate DOM elements imperatively or to store values that don't trigger re-renders.



CONCLUSION:

Here we have studied the concept of hooks in React.

Assignment 8

AIM: To implement Node JS REPL.

LO6: To orient students to Node.js for developing backend applications.

THEORY:

Introduction to Node.js REPL:

Node.js REPL, which stands for Read-Eval-Print Loop, is an interactive environment provided by Node.js for executing JavaScript code. It allows you to enter JavaScript commands or code snippets, which are then interpreted and executed by the Node.js runtime. REPL is a valuable tool for learning, prototyping, debugging, and experimenting with JavaScript and Node.js.

Components of Node.js REPL:

1. Read (Read Input):

The Read phase of the REPL takes user input. You can type JavaScript code, expressions, or commands directly into the REPL, similar to typing in a regular text editor or IDE.

2. Eval (Evaluate):

The Eval phase evaluates the JavaScript code or expressions entered by the user. It processes the input and executes it within the Node.js runtime environment.

The results of the evaluation are stored in a temporary variable (typically _), which can be accessed in subsequent commands to refer to the result of the last evaluation.

3. Print (Display Output):

The Print phase displays the result of the evaluated code on the console. It shows the output, including the return values, errors, or any other relevant information, to the user.

4. Loop (Repeat):

The Loop phase loops back to the Read phase, allowing the user to enter more code or commands. This process continues until the user decides to exit the REPL.

Features and Use Cases of Node.js REPL:

Interactive Learning and Testing:

Node.js REPL provides an excellent platform for learning JavaScript and experimenting with code in an interactive manner. You can quickly test code snippets to see how they work.

Debugging and Troubleshooting:

Developers can use the REPL for debugging purposes by interactively running code and examining variables or functions to identify and fix issues in their Node.js applications.

Prototyping:

When developing new functionality, developers can use the REPL to prototype and explore ideas before integrating them into larger codebases. It allows for rapid development and experimentation.

API Exploration:

You can interactively explore and test Node.js and third-party library APIs without writing a complete application. This is useful for understanding how different modules and functions work.

Mathematical and Scientific Calculations:

The REPL can be used as a simple calculator for performing mathematical and scientific calculations, making it handy for various quantitative tasks.

Basic Usage of Node.js REPL:

To start a Node.js REPL session, open your terminal or command prompt and type node followed by the Enter key. You'll see the Node.js REPL prompt (> or ... for multi-line input), indicating that it's ready to accept your JavaScript code. You can start typing code and press Enter to execute it.

Exiting Node.js REPL:

To exit the Node.js REPL, you can use the following methods:

Type .exit or press Ctrl + C twice.

Type .quit.

Press Ctrl + D on Unix-like systems (including macOS and Linux).

OUTPUTS:

```
PS C:\Users\Pratik Arote\Desktop\Iplab> node
Welcome to Node.js v18.13.0.
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
});

console.log("Simple Calculator\n");

function calculator() {
  rl.question("Enter the first number: ", (num1) => {
    if (num1.toLowerCase() === 'exit') {
      rl.close();
      return;
    }

    num1 = parseFloat(num1);

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

      num2 = parseFloat(num2);

      rl.question("Enter the operation (+, -, *, /): ", (operation) => {
        if (operation.toLowerCase() === 'exit') {
          rl.close();
          return;
        }

        try {
          let result;
```

```
        try {
          let result;
          switch (operation) {
            case '+':
              result = num1 + num2;
              break;
            case '-':
              result = num1 - num2;
              break;
            case '*':
              result = num1 * num2;
              break;
            case '/':
              if (num2 !== 0) {
                result = num1 / num2;
              } else {
                console.error("Division by zero is not allowed.");
                calculator();
                return;
              }
              break;
            default:
              console.error("Invalid operation. Please use +, -, *, or /.");
              calculator();
              return;
          }
          console.log(`Result: ${result}`);
        } catch (error) {
          console.error("Invalid expression. Please try again.");
        }

        calculator();
      });
    });
  });
}

calculator();
```

```
calculator();

Simple Calculator

Enter the first number: undefined
> 1
1
> Enter the second number: 2
2
> Enter the operation (+, -, *, /): +
... Result: 3
Enter the first number: 5
5
> Enter the second number: 8
8
> Enter the operation (+, -, *, /): *
*
^

Uncaught SyntaxError: Unexpected token '*'
> Result: 40
Enter the first number: 3
3
> Enter the second number: 1
1
> Enter the operation (+, -, *, /): -
... Result: 2
... s
```

CONCLUSION:

Here we have implemented calculator program using Node JS REPL.

Assignment 9

AIM: Write a program to implement

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

LO5: To orient students to React for developing frontend application.

THEORY:

In React, "Refs" (short for references) are a powerful feature that allows you to access and interact with the DOM (Document Object Model) elements and components directly. They provide a way to bridge the gap between the virtual representation of your UI in React and the actual HTML elements rendered in the browser. Refs can be used for various purposes, such as focusing input fields, triggering animations, measuring elements, and integrating with third-party libraries that require direct DOM access. Here's a theory and explanation of how refs work in React:

Introduction to Refs:

Refs are an advanced and low-level feature in React. They provide a way to access and manipulate DOM elements and React components directly. While React promotes a declarative and data-driven approach to building user interfaces, there are cases where you need to interact with the DOM imperatively. This is where refs come into play.

Creating Refs:

You can create refs in two ways: using the React.createRef() method or using the callback function approach. Here's how each method works:

React.createRef() Method:

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.myRef = React.createRef();
  }
  render() {
    return <div ref={this.myRef}>Hello, world!</div>;
```

```
  }
}
```

Callback Function Approach:

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.myRef = null;
  }
  setMyRef = (element) => {
    this.myRef = element;
  };
  render() {
    return <div ref={this.setMyRef}>Hello, world!</div>;
  }
}
```

Both methods achieve the same result: they create a reference to the DOM element associated with the div element in this example.

Forward Refs

Forward refs are a way to pass a ref object created in a parent component to a child component, allowing the parent to gain direct access to the child's DOM element or React component instance. This feature is particularly valuable when you're working with functional components or when you want to abstract the implementation details of a child component from its parent.

Creating a Forward Ref:

To create a forward ref, you typically use the React.forwardRef() function. This function accepts a function as an argument, and this function receives the props and ref as its arguments. Inside this function, you can return the JSX for your component, making sure to pass the ref to the element you want to forward it to.

Accessing DOM Elements:

Once you have a ref to a DOM element, you can access it using the .current property of the ref. For example:

const element = this.myRef.current;

Now, you can perform imperative operations on element, like changing its styles, adding event listeners, or calling native DOM methods.

Common Use Cases:

Focusing Input Fields: Refs are often used to focus input fields when a component mounts or in response to user actions.

Animating Elements: You can use refs to trigger animations by manipulating CSS classes or using animation libraries.

Measurements: Refs are useful for measuring the size and position of DOM elements, which can be helpful in building responsive layouts.

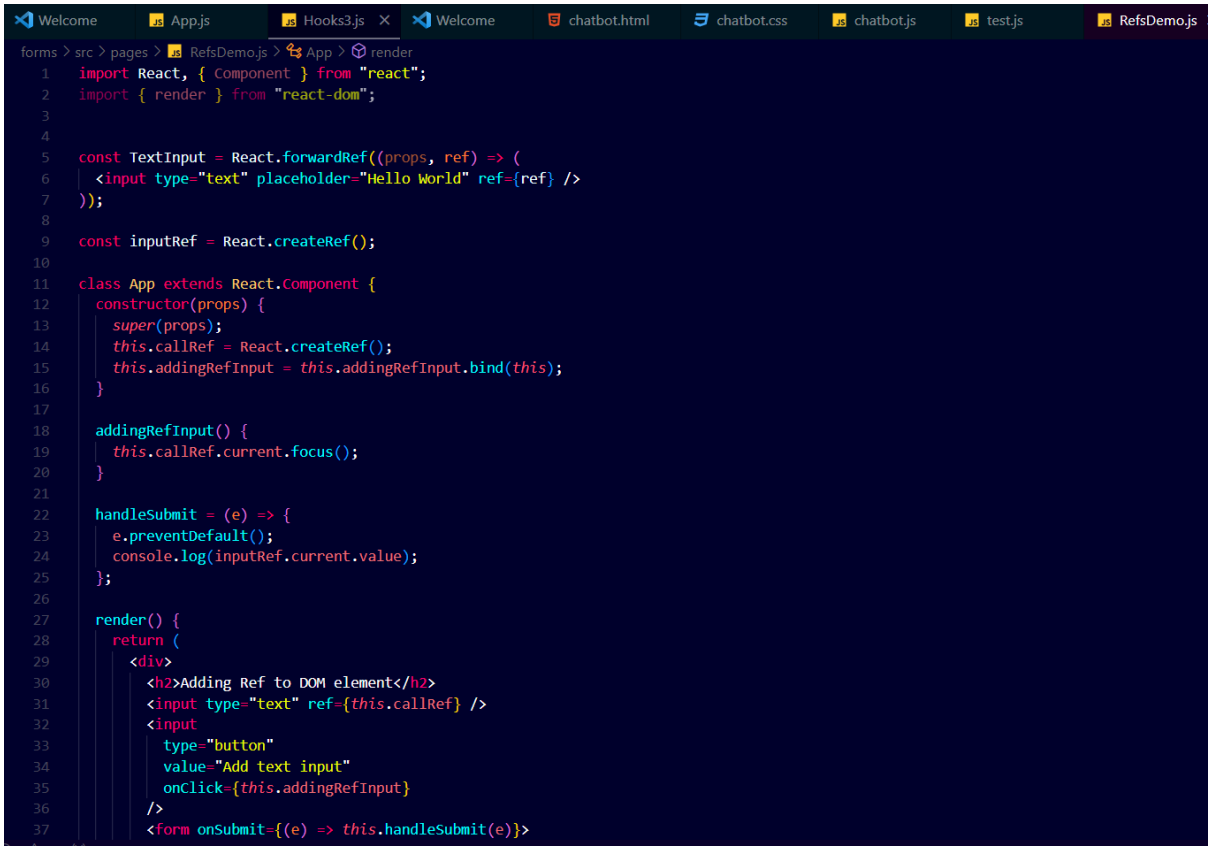
Integrating with Third-Party Libraries: Many third-party libraries, like D3.js or Chart.js, require direct access to the DOM. Refs make it possible to integrate such libraries into your React application.

Cautions and Best Practices:

Avoid excessive use of refs. React's declarative approach should be preferred whenever possible.

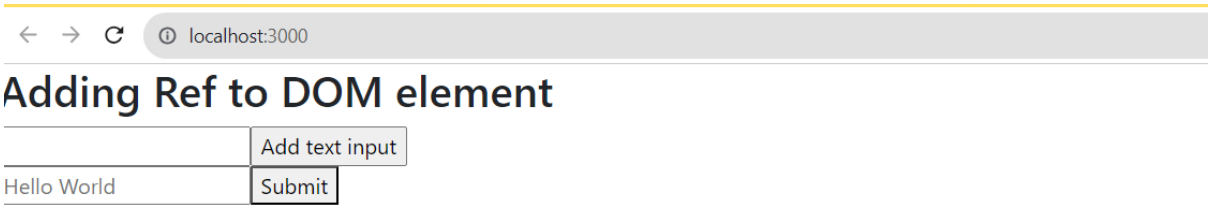
Refs should be used sparingly for integrating with non-React code and for solving specific, isolated problems.

Avoid manipulating the DOM directly unless it's necessary. React's reconciliation process may conflict with direct DOM manipulation.



```
import React, { Component } from "react";
import { render } from "react-dom";


const TextInput = React.forwardRef((props, ref) => (
  <input type="text" placeholder="Hello World" ref={ref} />
));

const inputRef = React.createRef();

class App extends React.Component {
  constructor(props) {
    super(props);
    this.callRef = React.createRef();
    this.addingRefInput = this.addingRefInput.bind(this);
  }

  addingRefInput() {
    this.callRef.current.focus();
  }

  handleSubmit = (e) => {
    e.preventDefault();
    console.log(inputRef.current.value);
  };

  render() {
    return (
      <div>
        <h2>Adding Ref to DOM element</h2>
        <input type="text" ref={this.callRef} />
        <input
          type="button"
          value="Add text input"
          onClick={this.addingRefInput}
        />
        <form onSubmit={(e) => this.handleSubmit(e)}>
```

```
  addingRefInput() {
    this.callRef.current.focus();
  }

  handleSubmit = (e) => {
    e.preventDefault();
    console.log(inputRef.current.value);
  };

  render() {
    return (
      <div>
        <h2>Adding Ref to DOM element</h2>
        <input type="text" ref={this.callRef} />
        <input
          type="button"
          value="Add text input"
          onClick={this.addingRefInput}
        />
        <form onSubmit={(e) => this.handleSubmit(e)}>
          <TextInput ref={inputRef} />
          <button>Submit</button>
        </form>
      </div>
    );
  }
}
export default App;
```



CONCLUSION:

Here we understood the concepts of refs in react and we successfully implemented it.

Assignment 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

LO6: To orient students to Node.js for developing backend applications.

THEORY:

a. Create a file:

In Node.js, you can create a file using the fs (file system) module. The fs.writeFile() method is commonly used for this purpose. It takes the filename as a parameter along with the content you want to write to the file. If the file already exists, it will be overwritten by default. You can also specify options like encoding and a callback function to handle errors or perform actions after the file is created.

b. Read the data from a file:

You can read the data from a file in Node.js using the fs.readFile() method. It takes the filename and an optional encoding parameter (usually 'utf8' for text files). You provide a callback function that receives the file's content or an error if something goes wrong.

c. Write data to a file:

To write data to a file in Node.js, you can use the fs.writeFile() method. It takes the filename, the content to write, and an optional callback function for error handling. This method overwrites the file's content if it already exists.

d. Rename a file:

You can rename a file in Node.js using the fs.rename() method. Provide the current filename and the new filename as arguments, along with a callback function to handle errors or perform actions after renaming.

e. Append data to a file:

To append data to an existing file in Node.js, you can use the fs.appendFile() method. This method takes the filename, the data to append, and an optional callback function for error handling.

f. Delete a file:

To delete a file in Node.js, you can use the fs.unlink() method. Provide the filename as an argument, along with a callback function to handle errors or perform actions after deletion.

test.js > ...

```
const fs = require('fs')

fs.writeFile('./myfiles/file1.txt','New File Created!',(err)=>{
    if(err){
        console.log(err);
    }
});

fs.readFile('./myfiles/file1.txt',(err,data)=>{
   console.log(data.toString());
});

fs.rename('./myfiles/file1.txt','./myfiles/myFile.txt',()=>{
    console.log("File Renamed.");
});

fs.appendFile('./myfiles/myFile.txt','New Appended Text',()=>{
    console.log("Appended Content successfully");
});

fs.unlink("input.txt", (err) => {
  if (err) {
    console.log(err);
  } else {
    console.log(`File deleted`);
  }
});
```

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE

```
PS C:\Users\Pratik Arote\Desktop\Iplab> node test.js
File deleted
File Renamed.
Appended Content successfully
New File Created!
PS C:\Users\Pratik Arote\Desktop\Iplab>
```

CONCLUSION:

Here we studied file operations in NodeJS. We studied how to create, write, read, append, rename, delete.

Assignment 11

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

THEORY:

OVERVIEW

The contact manager website developed using the MERN stack offers a comprehensive solution for users to efficiently manage their contacts. To get started, users must register and log in to access the platform. Once logged in, they can easily create new contacts, each with essential fields such as name, email, and phone number.

Additionally, users have the flexibility to update or delete their contacts as needed, ensuring their contact list remains up-to-date. The website's powerful search functionality enables users to quickly find specific contacts within their list.

Notably, users can only view and interact with contacts they've created, ensuring data privacy and a personalized experience. This feature-rich contact manager streamlines the process of managing and organizing personal or professional contacts, enhancing user convenience and efficiency.

FUNCTIONS

1. Create Contact
2. Edit Contact
3. Delete Contact
4. Search Contact
5. View Contacts
6. Verification of user (Login/Register) – JWT Token

TECHNOLOGIES USED

1. FRONTEND

   REACT JS (Semantic UI for CSS)

2. BACKEND

   NODE JS / EXPRESS JS

3. DATABASE

   MONGO DB

SCREENSHOTS

App.js | ContactList.js | contactController.js | ContactCard.js | Header.js | Register.js | Login.js

contact-app > src > components > App.js > App

```
import React, { useState, useEffect } from "react";
import { BrowserRouter as Router, Routes, Route } from "react-router-dom";
import api from "../api/contact";
import { v4 as uuidv4 } from "uuid";
import "./App.css";
import Header from "./Header";
import AddContact from "./AddContact";
import ContactList from "./ContactList";
import ContactDetail from "./ContactDetail";
import EditContact from "./EditContact";
import Login from "./Login";
import Register from "./Register";


function App() {
  const LOCAL_STORAGE_KEY = "contacts";

  const [contacts, setContacts] = useState(() => {
    const retrieveContacts = JSON.parse(
      localStorage.getItem(LOCAL_STORAGE_KEY)
    );
    return retrieveContacts || [];
  });
  const [searchTerm, setSearchTerm] = useState("");
  const [searchResults, setSearchResults] = useState([]);
  const [updatedContact, setUpdatedContact] = useState({});
  const [DContact, setDContact] = useState({});


  const retrieveContacts = async () => {
    const myValue = localStorage.getItem("userId");

    const headers = {
      "X-My-Value": myValue,
    };
    const response = await api.get("/api/contacts",{headers});
    return response.data;
```

0 0 0

App.js | ContactList.js | contactController.js | ContactCard.js | Header.js | Register.js | Login.js

contact-app > src > components > App.js > App

```
const addContactHandler = async (contact) => {
  const request = {
    id: localStorage.getItem("userId"),
    ...contact,
  };
  const response = await api.post("/api/contacts", request);
  setContacts([...contacts, response.data]);
};


const updateContactHandler = async (contact) => {
  const myValue = localStorage.getItem("userId");
  const headers = {
    "X-My-Value": myValue,
  };
  const response = await api.put(`/api/contacts/${contact.id}`, contact,{headers});
  const { id, name, email } = response.data;
  setContacts(
    contacts.map((contact) => {
      return contact.id === id ? { ...response.data } : contact;
    })
  );
};


const editContact = (contact) => {
  setUpdatedContact(contact);
};


const removeContactHandler = async (id) => {
  const myValue = localStorage.getItem("userId");
  const headers = {
    "X-My-Value": myValue,
  };
  await api.delete(`/api/contacts/${id}`,{headers});
```

App.js | ContactList.js | contactController.js | ContactCard.js | Header.js | Register.js | Login.js | userController.js

contact-app > src > components > App.js > App

```
const clickedContact = (contact) => {
  setDContact(contact);
};

const searchHandler = (searchTerm) => {
  setSearchTerm(searchTerm);
  if (searchTerm !== "") {
    const newContactList = contacts.filter((contact) => {
      return Object.values(contact)
        .join(" ")
        .toLowerCase()
        .includes(searchTerm.toLowerCase());
    });
    setSearchResults(newContactList);
  }
  else{
    setSearchResults(contacts);
  }
};

useEffect(() => {
  const getAllContacts = async () => {
    const allContacts = await retrieveContacts();
    if (allContacts) setContacts(allContacts);
  };

  getAllContacts();
}, []);


useEffect(() => {
  const getAllContacts = async () => {
    const allContacts = await retrieveContacts();
    if (allContacts) setContacts(allContacts);
  };
```

App.js | ContactList.js | ContactCard.js | Header.js | Register.js | Login.js | userController.js

contact-app > src > components > ContactCard.js > ContactCard

```
const ContactCard = (props) => {
    const {name,email,phone} = props.contact;
    const id = props.contact._id;
    return (
      <div className="item">
        <i className="large user circle icon"></i>
        <div
          className="content"
          onClick={() => props.getClickContact(props.contact)}
        >
          <Link
            to={{
              pathname: `/contact/${id}`,
              state: { contact: props.contact },
            }}
          >
            <div className="header">{name}</div>
            <div>{email}</div>
            <div>{phone}</div>
          </Link>
        </div>
        <i
          className="large trash alternate outline icon "
          style={{
            color: "red",
            marginTop: "7px",
          }}
          onClick={() => {
            props.clickHandler(id);
          }}
        ></i>
        <Link
          to={{
            pathname: `/edit`,
            state: { contact: props.contact },
          }}
        >
```

```
const ContactList = (props) => {

    let isLogin = localStorage.getItem("userId");

    const inputEl = useRef("");
    const deleteContactHandler = (id) => {
      props.getContactId(id);
    };

    const clickContact = (contact) => {
      props.getClickedContact(contact);
    }

    const updateContact = (contact) => {
       props.editContact(contact);
    }

    const getSearchTerm = () => {
        props.searchKeyword(inputEl.current.value);
    }

     const renderContactList = props.contacts.map((contact)=>{
        return (
          <>
            <ContactCard
              contact={contact}
              clickHandler={deleteContactHandler}
              getClickContact = {clickContact}
              updateContact = {updateContact}
            />
          </>
        );
      });
  return (
    <div className="main">
      <h2>
```

App.js | ContactList.js | ContactCard.js | Header.js | Register.js

contact-app > src > components > Header.js > ...

```jsx
const Header = (props) => {
  //global state
  let isLogin = useSelector((state) => state.isLogin);
  isLogin = isLogin || localStorage.getItem("userId");
  const dispatch = useDispatch();
  const navigate = useNavigate();
  //state

  //Logout
  const handleLogout = () => {
    try {
      dispatch(authActions.logout());
      toast.success("Logout Successfully");
      navigate("/login");
      localStorage.clear();
    } catch (error) {
      console.log(error);
    }
  };

  return (
    <div className="ui segment">
      <div className="ui center aligned header">
        <h2>{props.header}</h2>
        {!isLogin && (
          <>
            <button className="ui primary button">
              <Link to="/login" style={{ color: "white" }}>
                Login
              </Link>
            </button>
            <button className="ui button">
              <Link to="/register">Register</Link>
            </button>
          </>
        )}
```

```jsx
const Register = () => {
    const navigate = useNavigate();
    const [inputs ,setInputs] = useState({
        username:"",
        email:"",
        password:"",
    });

    const handleChange = (e) => {
        setInputs((prevState)=>({
            ...prevState,
            [e.target.name]: e.target.value
        }));
    }

    const handleSubmit = async(e) => {
        e.preventDefault();
        try{
            fetch("http://localhost:5001/api/users/register", {
              method: "POST",

              body: JSON.stringify({
                username: inputs.username,
                email: inputs.email,
                password: inputs.password,
              }),

              headers: {
                "Content-type": "application/json",
              },
            });
            toast.success("User Registered Successfully")
            navigate("/login");
        }catch(err){
            console.log(`HandleSubmit Register is giving error ${err}`);
        }
    }
```

BACKEND

```
const  express = require('express');
const colors = require('colors');
const dotenv = require("dotenv").config();
const app = express();
const errorHandler = require("./middleware/errorHandler");
const connectDB = require('./config/dbConnection');
const cors = require('cors');
connectDB();
const PORT  = process.env.PORT||5000;


app.use(cors());
app.use(express.json());
app.use("/api/contacts",require("./routes/contactRoutes"));
app.use("/api/users", require("./routes/userRoutes"));
app.use(errorHandler);

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

config > JS dbConnection.js > ...

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

const connectDB = async () => {
    try{
        const connect = await mongoose.connect(process.env.CONNECTION_STRING);
        console.log(`Database connected: ${connect.connection.host}  ${connect.connection.name}`.bgYellow.black);
    }catch(err){
        console.log(err);
        process.exit(1);
    }
}

module.exports = connectDB;
```

utes > JS contactRoutes.js > ...

```
const express = require("express");
const {
  getContacts,
  createContact,
  getContact,
  updateContact,
  deleteContact,
} = require("../controllers/contactController");
// const validateToken = require("../middleware/validateTokenHandler");
const router = express.Router();
// router.use(validateToken);
router
  .route("/")
  .get(getContacts)
  .post(createContact);
router
  .route("/:id")
  .put(updateContact)
  .delete(deleteContact)
  .get(getContact);

module.exports = router;
```

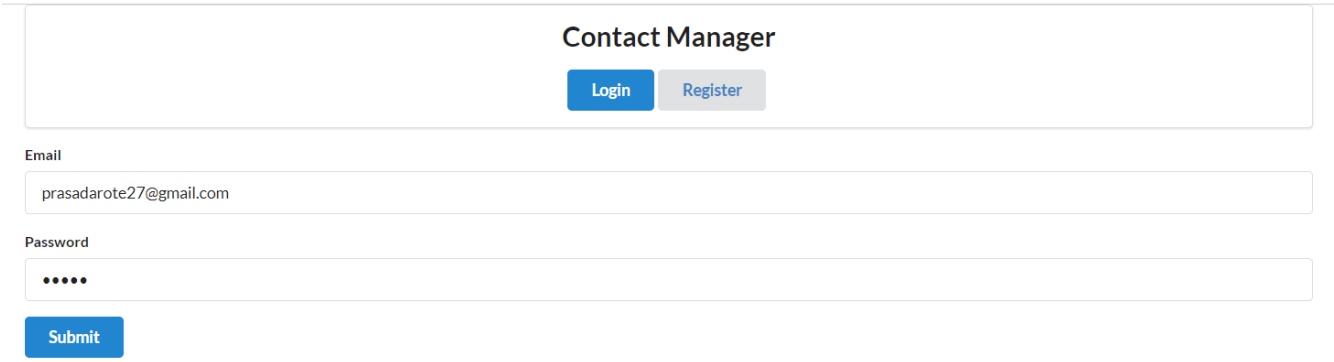
```
const mongoose = require('mongoose');

const contactSchema = new mongoose.Schema(
  {
    user_id:{
      type:mongoose.Schema.Types.ObjectId,
      required: true,
      ref: "User",
    },
    name: {
      type: String,
      required: [true, "Please add the contact name"],
    },
    email: {
      type: String,
      required: [true, "Please add the contact email address"],
    },
    phone: {
      type: String,
      required: [true, "Please add the contact phone number"],
    },
  },
  { timestamps: true }
);

module.exports = mongoose.model("Contact",contactSchema);
```

```
const asyncHandler = require("express-async-handler");
const Contact = require("../models/contactModel");
const { default: mongoose } = require("mongoose");

//@desc Get all contacts
//@route GET /api/contacts
//@access private
const getContacts = asyncHandler(async (req, res) => {
  const myValue = req.headers["x-my-value"];
  const contacts = await Contact.find({ user_id: myValue });
  res.status(200).json(contacts);
});

//@desc Create New contact
//@route POST /api/contacts
//@access private
const createContact =asyncHandler (async (req, res) => {
  const { name, email, phone } = req.body;
  if (!name || !email || !phone) {
    res.status(400);
    throw new Error("All fields are mandotary");
  }
  const contact = await Contact.create({
    name,
    email,
    phone,
    user_id: req.body.id,
  });
  res.status(201).json(contact);
});
```

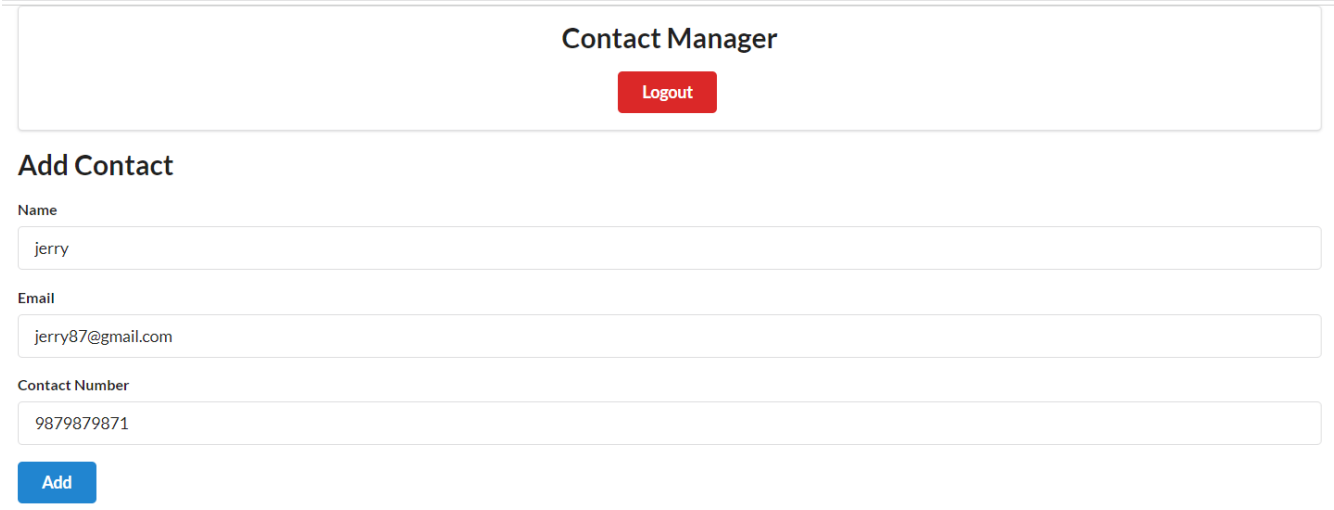
```
//@route PUT /api/contacts/:id
//@access private
const updateContact =asyncHandler( async (req, res) => {
  const contact = await Contact.findById(req.params.id);
  if (!contact) {
    res.status(404);
    throw new Error("Contact Not Found");
  }
  if (contact.user_id.toString() !== req.headers["x-my-value"]) {
    res.status(403);
    throw new Error("You don't have permission to do this operation.");
  }
  const updatedContact = await Contact.findByIdAndUpdate(req.params.id,req.body,{new:true});
  res.status(200).json(updatedContact);
});


//@desc Delete contact
//@route DELETE /api/contacts/:id
//@access private
const deleteContact =asyncHandler( async (req, res) => {
  const contact = await Contact.findById(req.params.id);
  if (!contact) {
    res.status(404);
    throw new Error("Contact Not Found");
  }
  if (contact.user_id.toString() !== req.headers["x-my-value"]) {
    res.status(403);
    throw new Error("You don't have permission to do this operation.");
  }
  const Deletedcontact = await Contact.findByIdAndDelete(req.params.id);
  res.status(200).json(Deletedcontact);
});


module.exports = {getContacts,createContact,getContact,updateContact,deleteContact};
```

```
const errorHandler = (err,req,res,next) => {
    const statusCode = res.statusCode ? res.statusCode : 500;
    switch (statusCode) {
      case constants.VALIDATION_ERROR:
        res.json({
          title: "Validation Failed",
          message: err.message,
          stackTrace: err.stack,
        });
        break;
      case constants.NOT_FOUND:
        res.json({
          title: "Not Found",
          message: err.message,
          stackTrace: err.stack,
        });
        break;
      case constants.UNAUTHORIZED:
        res.json({
          title: "Unauthorized",
          message: err.message,
          stackTrace: err.stack,
        });
        break;
      case constants.FORBIDDEN:
        res.json({
          title: "Forbidden",
          message: err.message,
          stackTrace: err.stack,
        });
        break;
      case constants.SERVER_ERROR:
        res.json({
          title: "Forbidden",
          message: err.message,
```

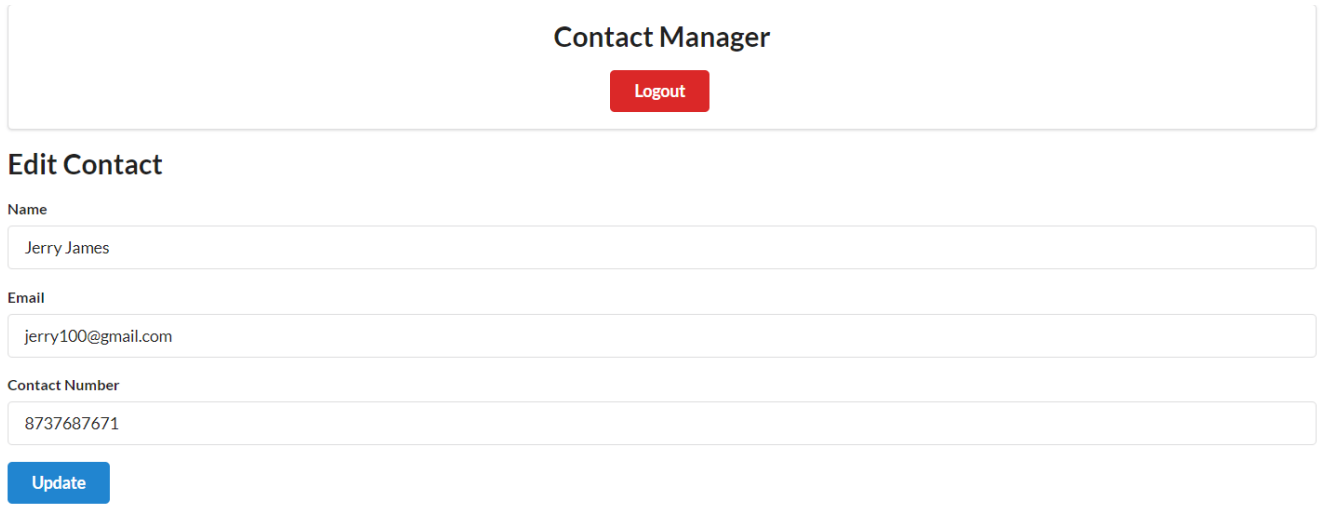
OUTPUT:

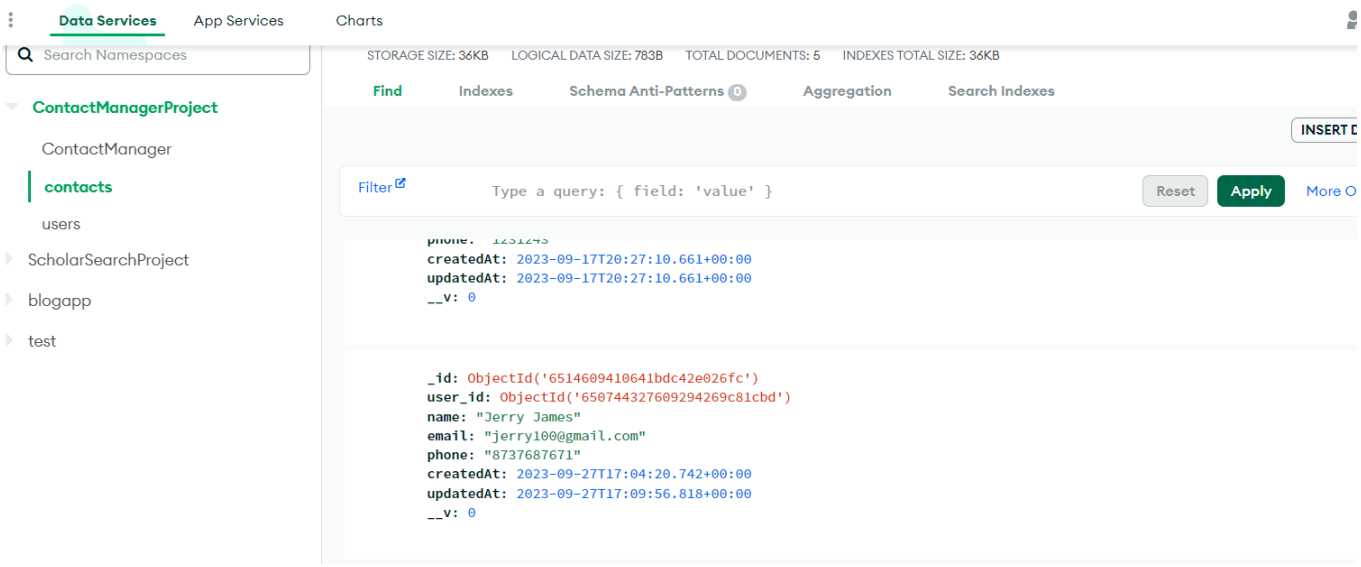


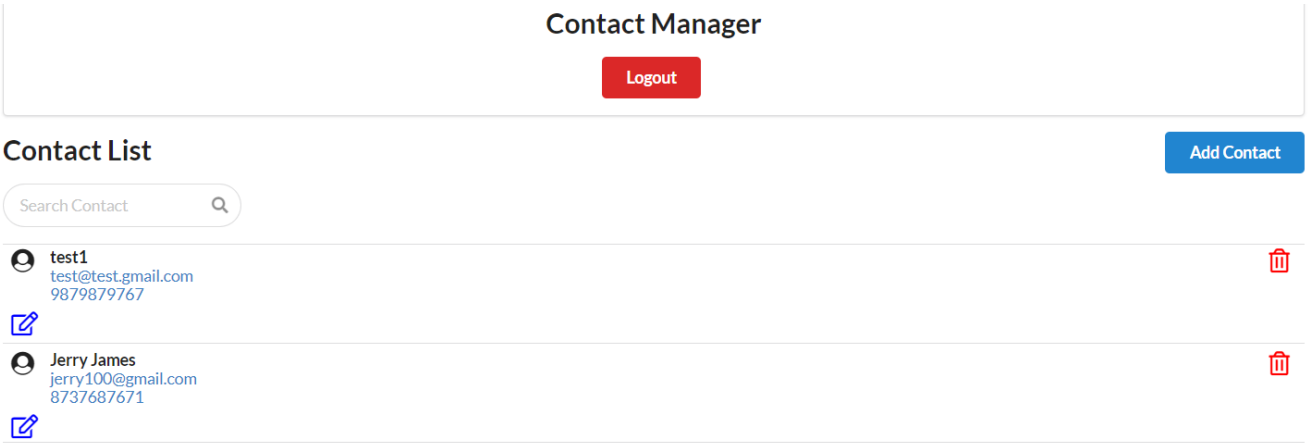
ADD CONTACT



UPDATE CONTACT







CONCLUSION:

Here, we have made Contact Manager Website using MERN which performs CRUD operations successfully.

Written Assignment 01

1. Compare XML and JSON.

JSON	XML
It is JavaScript Object Notation	It is Extensible markup language
It is based on JavaScript language.	It is derived from SGML.
It is a way of representing objects.	It is a markup language and uses tag structure to represent data items.
It does not provides any support for namespaces.	It supports namespaces.
It supports array.	It doesn't supports array.
Its files are very easy to read as compared to XML.	Its documents are comparatively difficult to read and interpret.
It doesn't use end tag.	It has start and end tags.
It is less secured.	It is more secured than JSON.
It doesn't supports comments.	It supports comments.
It supports only UTF-8 encoding.	It supports various encoding.

2. Explain different types of arrow functions

   Arrow functions, also known as arrow function expressions, are a feature introduced in JavaScript to provide a more concise syntax for writing functions. They are often used in modern JavaScript code for their brevity and the way they handle the this keyword. There are different types of arrow functions based on their structure and behavior:

a. Arrow Function with One Argument:

If the arrow function takes only one argument, you can omit the parentheses around the argument list. This is particularly concise when the argument is a single identifier.

E.g.- const square = x => x * x;

b. Arrow Function with No Arguments:

If the arrow function doesn't take any arguments, you still need to provide empty parentheses to indicate that it's a function.

E.g.- const getRandomNumber = () => Math.random();

c. Arrow Function as an Expression:

Arrow functions are often used for simple expressions that can be directly returned without using curly braces or the return keyword.

E.g.- const isEven = number => number % 2 === 0;

d. Arrow Function with Block Body:

If the arrow function's body consists of multiple statements or more complex logic, you can use a block body enclosed in curly braces. In this case, if you want to return a value, you need to use the return keyword.

E.g.- const greet = name => {
  const greeting = `Hello, ${name}!`;
  return greeting;
};

e. Arrow Function with Rest Parameters:

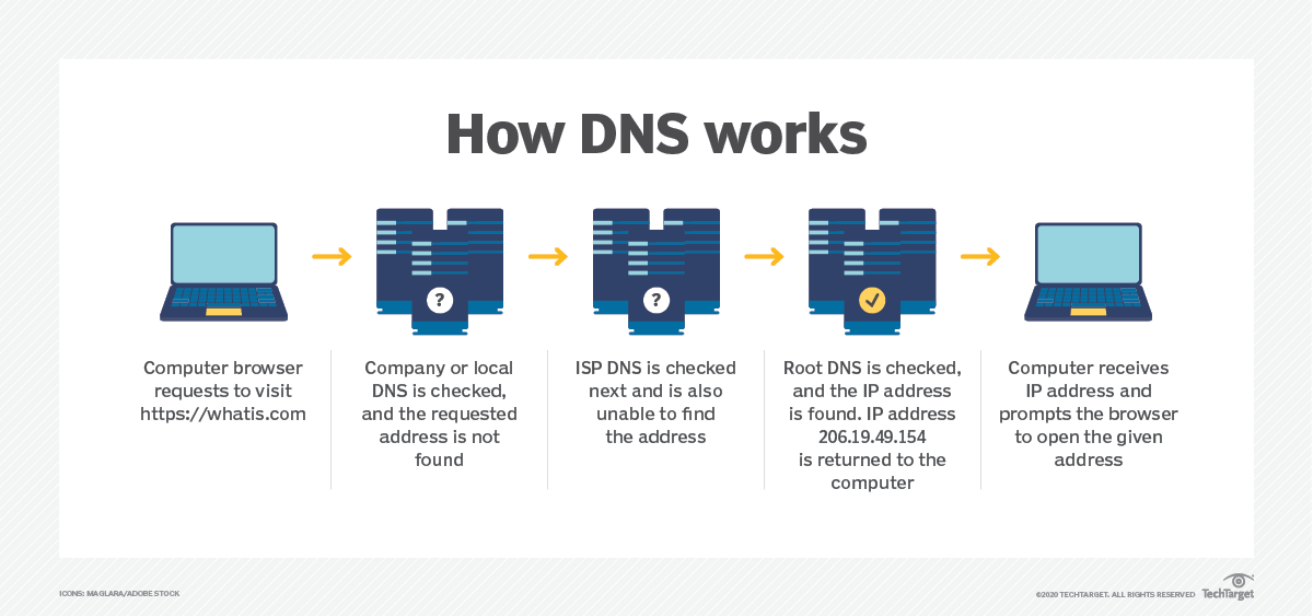
Arrow functions can also use the rest parameter syntax (...) to capture multiple arguments into an array.

E.g.- const sum = (...numbers) => {
    return numbers.reduce((total, num) => total + num, 0);
};

3. What is DNS? Explain working of DNS.

DNS, which stands for Domain Name System, is a fundamental protocol used on the internet to translate human-readable domain names (like www.example.com) into IP addresses (like 192.0.2.1), which are used by computers to locate and communicate with each other. It serves as a distributed directory that helps users access websites and other resources using familiar domain names rather than having to remember numeric IP addresses.

Working



1) Domain Name Resolution Request:

When you enter a URL (Uniform Resource Locator) in your web browser, such as "www.example.com," your computer needs to find the corresponding IP address to connect to the server hosting that website. It sends a DNS resolution request to your local DNS resolver.

2) Local DNS Resolver:

Your internet service provider (ISP) or network usually provides a local DNS resolver. This resolver maintains a cache of recent domain name lookups to speed up future requests. If the requested domain name is found in the cache, the resolver can immediately provide the corresponding IP address.

3) Recursive DNS Query:

If the domain name is not in the local cache, the local resolver initiates a recursive DNS query. It sends a request to one of the root DNS servers, asking for the top-level domain (TLD) server responsible for the specific domain extension (like ".com" or ".org").

4) Root DNS Servers:

The root DNS servers are a critical part of the DNS infrastructure. They provide information about the TLD servers that store information for specific domain extensions. These servers do not store the actual IP addresses of websites; they only point to the authoritative DNS servers for each TLD.

5) TLD DNS Servers:

The TLD DNS server receives the query from the root server and directs the resolver to the authoritative DNS server for the second-level domain (like "example.com"). TLD servers know which authoritative DNS servers are responsible for each specific domain.

6) Authoritative DNS Server:

The authoritative DNS server for the requested domain contains the mapping between the domain name and the corresponding IP address. When the resolver queries this server, it provides the IP address for the requested domain.

7) Caching and Response:

The local DNS resolver receives the IP address from the authoritative server. It stores this mapping in its cache for a specified period, known as the Time To Live (TTL). The resolver then provides the IP address to your computer, allowing your browser to initiate a connection to the web server hosting the website.

8) Accessing the Website:

With the IP address in hand, your computer can communicate directly with the web server, retrieving the requested web page and its associated resources (images, scripts, stylesheets, etc.).

4. Explain Promises in ES6.

Promises are a feature introduced in ES6 (ECMAScript 2015) to help manage asynchronous operations in JavaScript in a more organized and readable way. They provide a way to handle the results (fulfilled) or errors (rejected) of asynchronous operations when they complete, making it easier to write asynchronous code that doesn't become deeply nested (a condition known as "callback hell"). Promises provide a cleaner and more structured approach to handling asynchronous operations compared to traditional callback functions.

Here's how Promises work:

a) <u>Creating a Promise:</u>

To create a Promise, you use the Promise constructor, which takes a function as an argument. This function is called the "executor" function. The executor function usually contains asynchronous code that performs an operation and resolves the Promise when the operation is successful, or rejects it if there's an error.

E.g.-
```
const myPromise = new Promise((resolve, reject) => {
  // Asynchronous operation
  if (/* operation successful */) {
    resolve(result);
  } else {
```

```
    reject(error);
  }
});
```

b) Handling Promise Results:

You can attach handlers to a Promise to handle the resolved value or catch errors. These handlers are specified using the .then() method for successful results and the .catch() method for errors.

```
myPromise
  .then(result => {
    // Handle resolved result
  })
  .catch(error => {
    // Handle error
  });
```

c) Chaining Promises:

One of the powerful features of Promises is chaining. You can return another Promise from a .then() handler, allowing you to chain multiple asynchronous operations together. This helps to avoid deeply nested callback structures.

E.g. -

```
myPromise
  .then(result => {
    // Perform another asynchronous operation and return a Promise
    return anotherPromise;
  })
  .then(anotherResult => {
    // Handle result of the second Promise
  })
  .catch(error => {
    // Handle errors from either Promise
  });
```

d) Promise States:

A Promise can be in one of three states:

I) Pending: The initial state, before the Promise is either resolved or rejected.
II) Fulfilled: The state when the asynchronous operation is successful, and the Promise is resolved with a value.
III)Rejected: The state when an error occurs during the asynchronous operation, and the Promise is rejected with an error.

e) Async/Await with Promises:

ES6 also introduced the async and await keywords, which work in conjunction with Promises to provide a more synchronous-like syntax for handling asynchronous operations. async functions always return Promises, and await is used within these functions to pause execution until a Promise is resolved.

E.g. -

```
async function fetchData() {
  try {
    const result = await myPromise;
    // Use result
  } catch (error) {
    // Handle error
  }
}
```

Promises have significantly improved the way asynchronous code is written and managed in JavaScript. They make it easier to handle both successful and error cases, prevent callback hell, and allow for more readable and maintainable asynchronous code.

Assignment 2

1. **What are Refs? When to use Refs and when not to use refs.**

   Refs in React:

   Refs in React are a way to access and interact with the DOM (Document Object Model) directly. They provide a way to reference a DOM element or a React component instance created in the render method. Refs are typically used when you need to:

   Accessing DOM elements: Refs allow you to access and manipulate DOM elements directly, which is useful for things like focusing on an input field, playing video/audio, or animating elements.

   Interacting with third-party libraries: If you are integrating React with non-React code or libraries that rely on direct DOM manipulation, refs can be used to bridge the gap.

   Managing focus, text selection, or media playback: Refs can be used for managing user interactions that require fine-grained control over the DOM.

   However, it's important to note that using refs should be a last resort in React. In most cases, you can achieve what you need by managing state and props within your components, which follows the React way of handling interactions. Using refs can lead to code that's harder to maintain, test, and understand, and it may not work well with React's reconciliation mechanism.

2. **Write short note on**
   a) **NPM**
   b) **REPL**

      NPM (Node Package Manager):
      NPM is the default package manager for Node.js, and it is one of the largest software registries in the world. It allows developers to easily manage and distribute JavaScript packages and libraries. Some key points about NPM:

      Package Management: NPM is used to install, update, and manage packages (libraries and modules) for Node.js and JavaScript projects. It simplifies the process of dependency management.

      Command-Line Tool: NPM provides a command-line interface that allows developers to interact with packages and perform various tasks like installing packages, running scripts, and managing versions.

Package.json: NPM uses a package.json file to define project metadata and dependencies. This file contains information about the project, its dependencies, and scripts to run various tasks.

Registry: NPM hosts a vast collection of public packages in the NPM registry. Developers can publish their own packages to the registry for others to use.

REPL (Read-Eval-Print Loop):

REPL is an interactive programming environment that allows you to enter and execute code one line at a time. It stands for Read-Eval-Print Loop and is commonly used for quickly testing code snippets, experimenting with language features, and debugging. Node.js comes with a built-in REPL environment.

Key characteristics of a REPL:

Read: It reads user input, typically a single line of code or expression.

Eval: It evaluates the input and executes it.

Print: It prints the result of the evaluation to the console.

Loop: It repeats the process, allowing users to enter more code.

A REPL is handy for trying out code, exploring APIs, and testing small pieces of code without having to write full programs.

3. **Explain Routing in ExpressJS along with an example.**

Routing in Express.js:
Express.js is a popular web framework for Node.js that simplifies the process of building web applications and APIs. Routing in Express.js refers to the mechanism by which you define how your application responds to different HTTP requests. It involves mapping URLs (routes) to specific functions or handlers that will process the requests and generate responses.

Here's a basic example of routing in Express.js:

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

// Define a route for the root URL
app.get('/', (req, res) => {
  res.send('Hello, World!');
```

```
});

// Define a route for /about
app.get('/about', (req, res) => {
  res.send('About Us');
});

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

In this example:

We create an Express.js application using express().
We define two routes using app.get(). One responds to the root URL ('/') and the other to '/about'.
Each route specifies a callback function that is executed when the corresponding URL is requested.
We start the server listening on port 3000.
When you visit 'http://localhost:3000/' in your browser, you will see 'Hello, World!' as the response. Visiting 'http://localhost:3000/about' will display 'About Us'. This is the fundamental concept of routing in Express.js, where you define how your application handles different URLs and their corresponding HTTP methods.