Thadomal Shahani Engineering College

Bandra (W.), Mumbai- 400 050.

CERTIFICATE

Certify that Mr./~~Miss~~ Altaf Alam

of Information Technology Department, Semester V with

Roll No. 02 has completed a course of the necessary

experiments in the subject IP Lab under my

supervision in the **Thadomal Shahani Engineering College**

Laboratory in the year 2023 - 2024

Saoobed

Teacher In-Charge

Head of the Department

Date 13/10/23

Principal

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

CONTENTS

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

Name : Altaf Alam

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

Assignment No 1

Aim : Develop a web app using HTML tags**.**

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

Theory :

What is HTML?

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

Description of Tags used in the example:

1. `<!DOCTYPE html>`: This is the document type declaration that tells the browser that the document is an HTML5 document. It must be the first line of an HTML document.

2. `<html>`: The root element of an HTML document, enclosing all other elements.

3. `<head>`: Contains meta-information about the document, such as the document title, character set, and links to external resources like stylesheets or scripts.

4. `<meta>`: Defines metadata about the document, such as the character encoding and viewport settings.

5. `<link>`: Used to link external resources like stylesheets to the HTML document.

6. `<title>`: Sets the title of the document, which appears in the browser's title bar or tab.

7. `<body>`: Contains the visible content of the web page, including text, images, links, and other elements.

8. `<nav>`: Represents a section of the page that contains navigation links.

9. `<img>`: Embeds an image in the document. The `src`, `alt`, `height`, and `width` attributes define the image source, alternative text, height, and width, respectively.

10. `<ul>` and `<li>`: Create an unordered list and list items, respectively. Used for creating lists of items without specific numbering.

11. `<h1>`, `<h2>`, ..., `<h6>`: Heading tags define headings of different levels, with `<h1>` being the highest level and `<h6>` being the lowest.

12. `<hr>`: Creates a horizontal rule or divider on the page.

13. `<video>`: Embeds video content in the document. The `controls`, `height`, and `width` attributes define video controls and dimensions.

14. `<source>`: Specifies the video source for the `<video>` element.

15. `<audio>`: Embeds audio content in the document. The `controls` attribute adds audio controls to the element.

16. `<fieldset>` and `<legend>`: Used together to group form elements and provide a legend or caption for the fieldset.

17. `<textarea>`: Creates a multi-line text input field for forms.

18. `<form>`: Defines an HTML form for user input. The `action` attribute specifies the URL to send the form data.

Name : Altaf Alam

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

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

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

21. `<strong>`: Renders text in bold font.

22. `<em>`: Renders text in italic font.

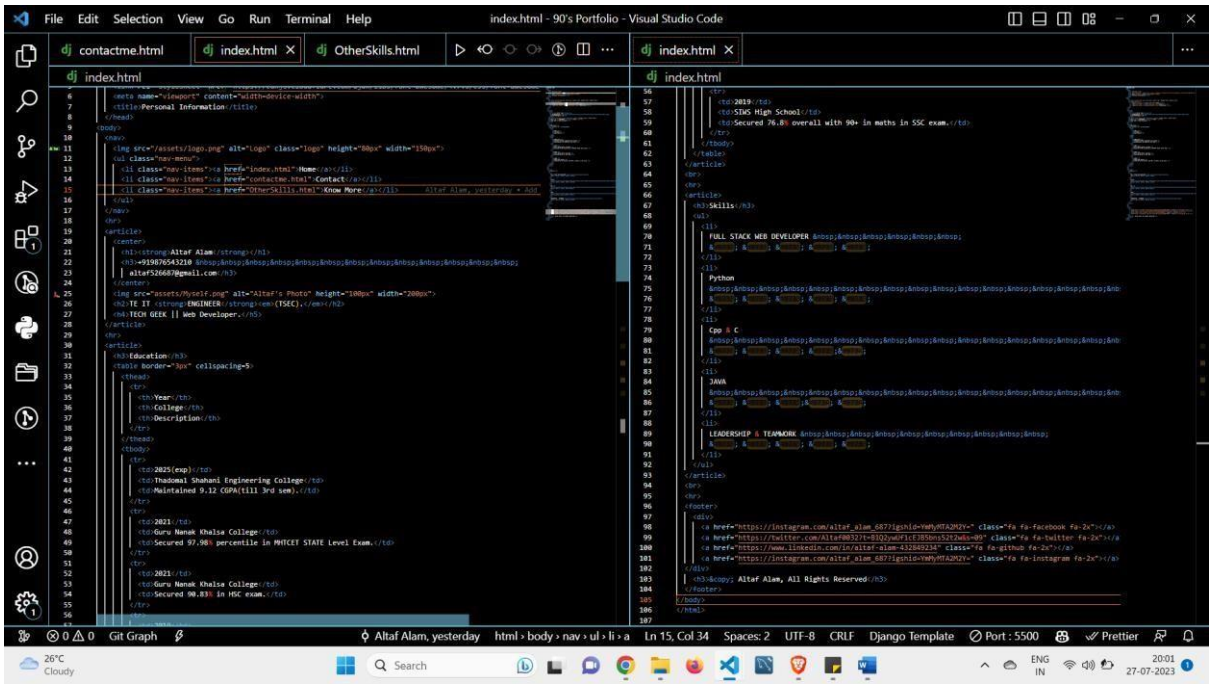
23. `<a>`: Creates hyperlinks, linking to other web pages or resources. The `href` attribute specifies the link destination.

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

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

Source Code :

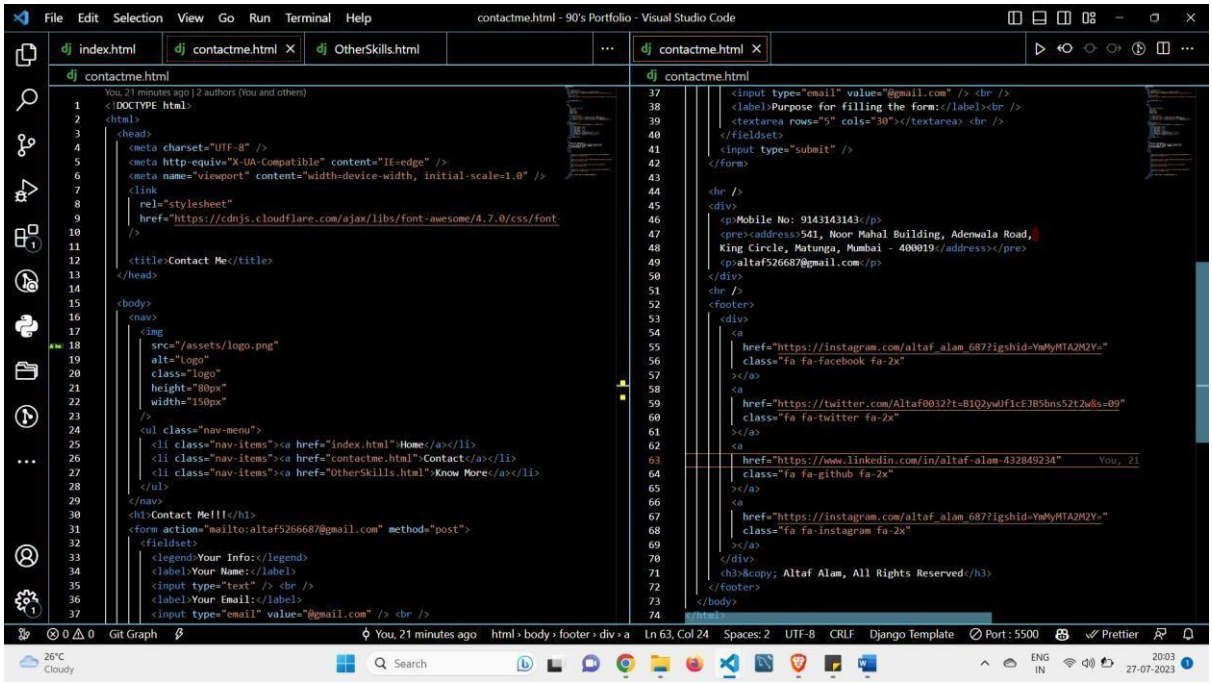
Index.html



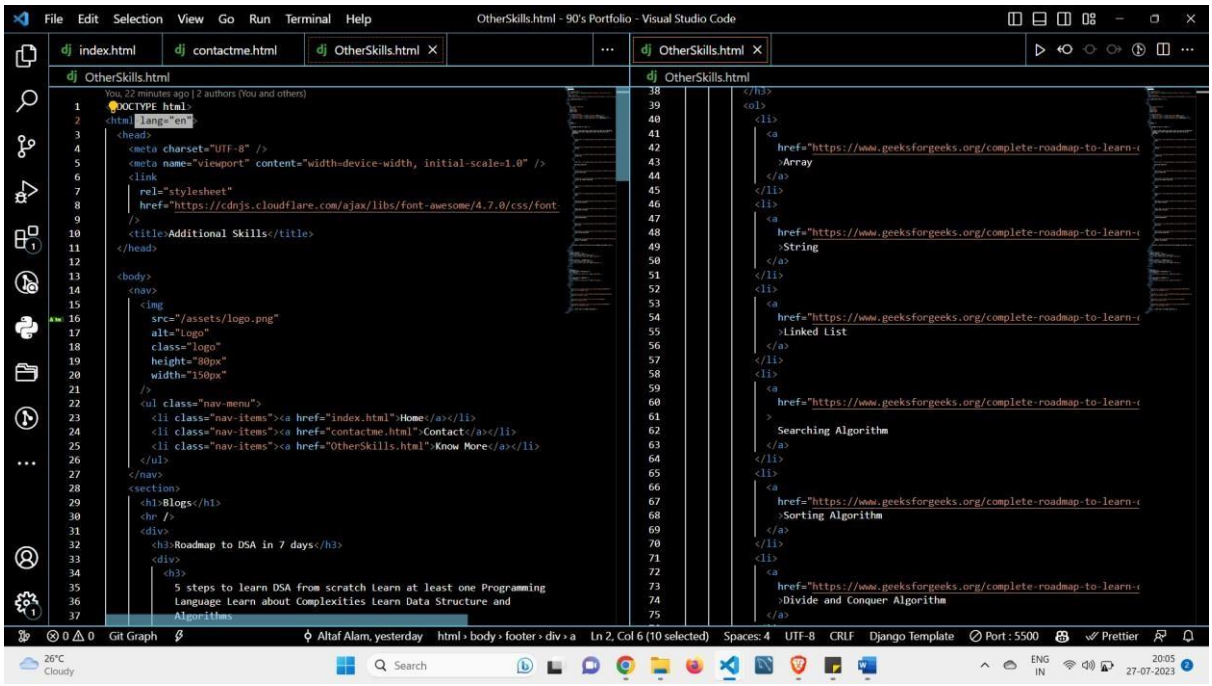
Contactme.html

Name : Altaf Alam

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

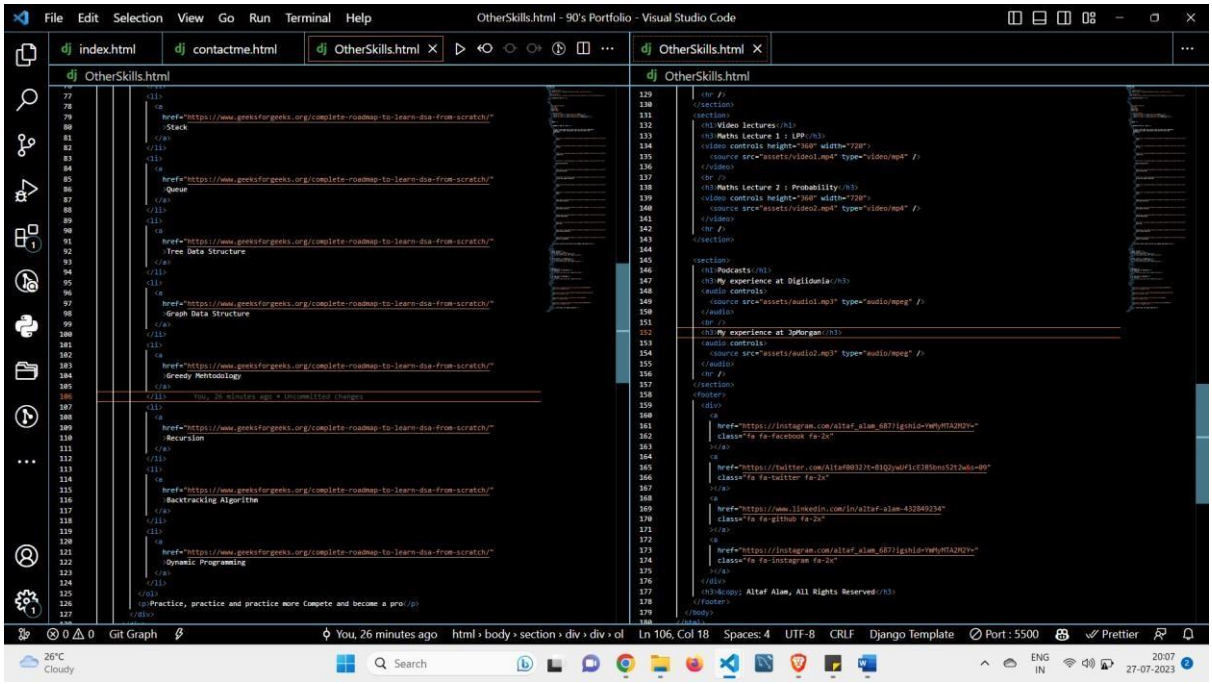


OtherSkills.html



Name : Altaf Alam

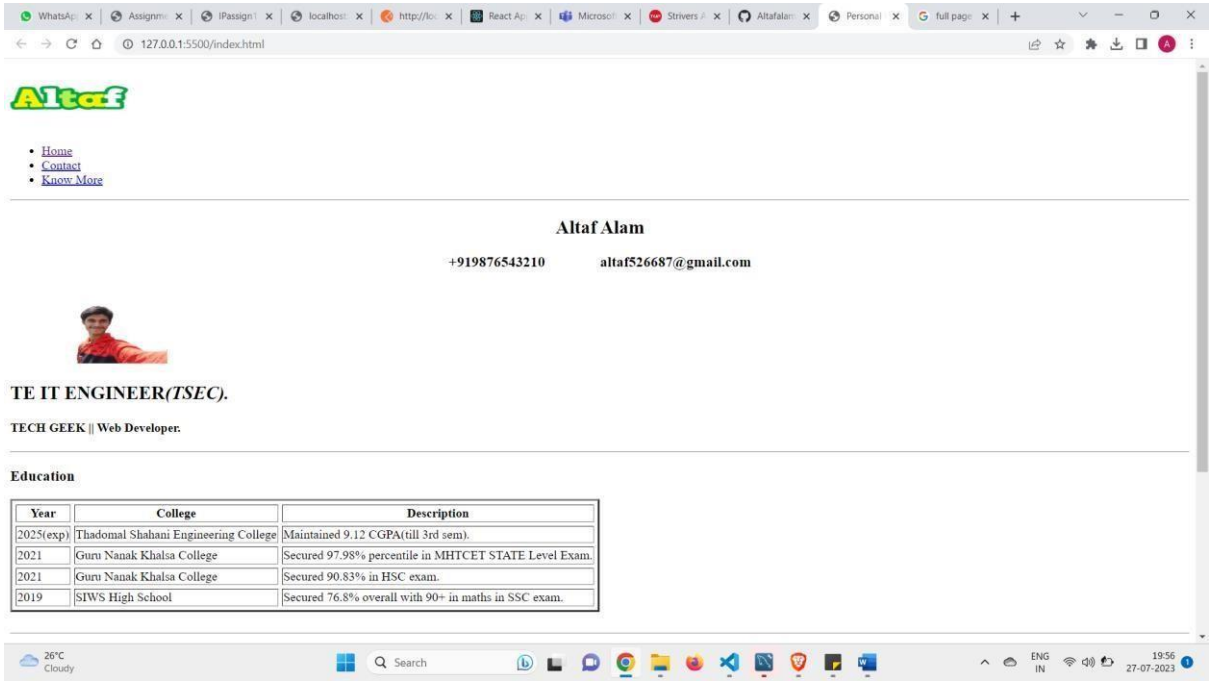
Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 18/07/23



Output :

Name : Altaf Alam

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



Altaf

- Home
- Contact
- Know More

Altaf Alam

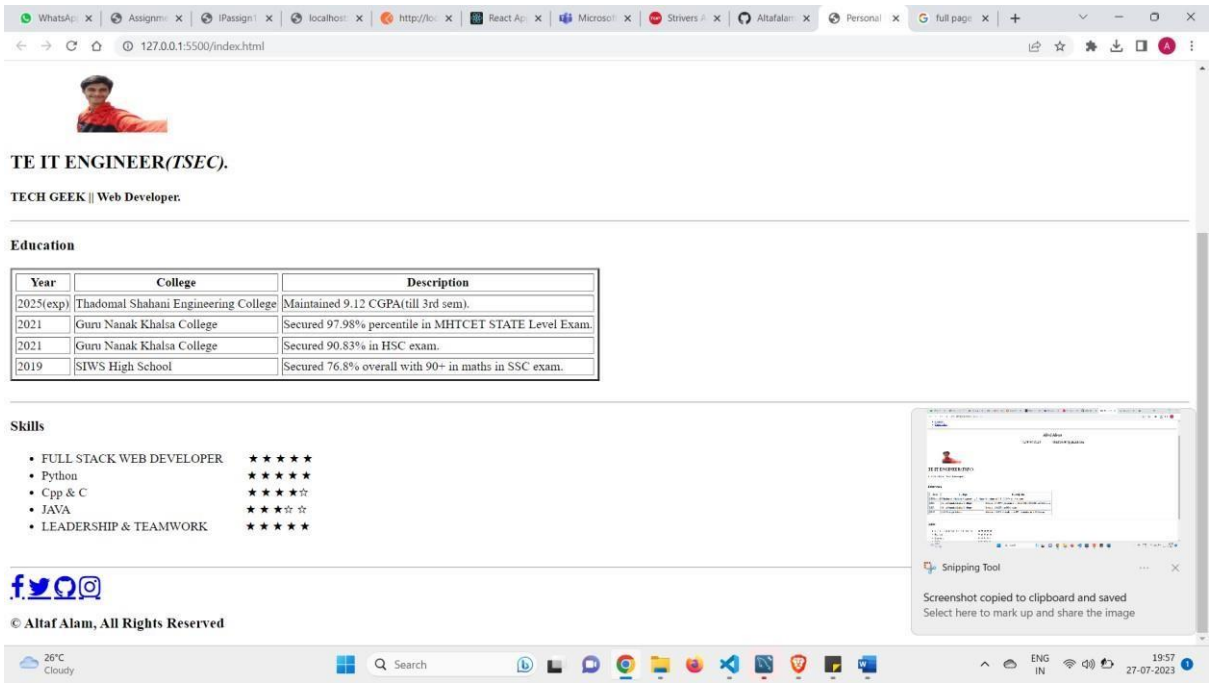
+919876543210 altaf526687@gmail.com

TE IT ENGINEER*(TSEC).*

TECH GEEK || Web Developer.

Education

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



TE IT ENGINEER*(TSEC).*

TECH GEEK || Web Developer.

Education

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

Skills

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

Snipping Tool

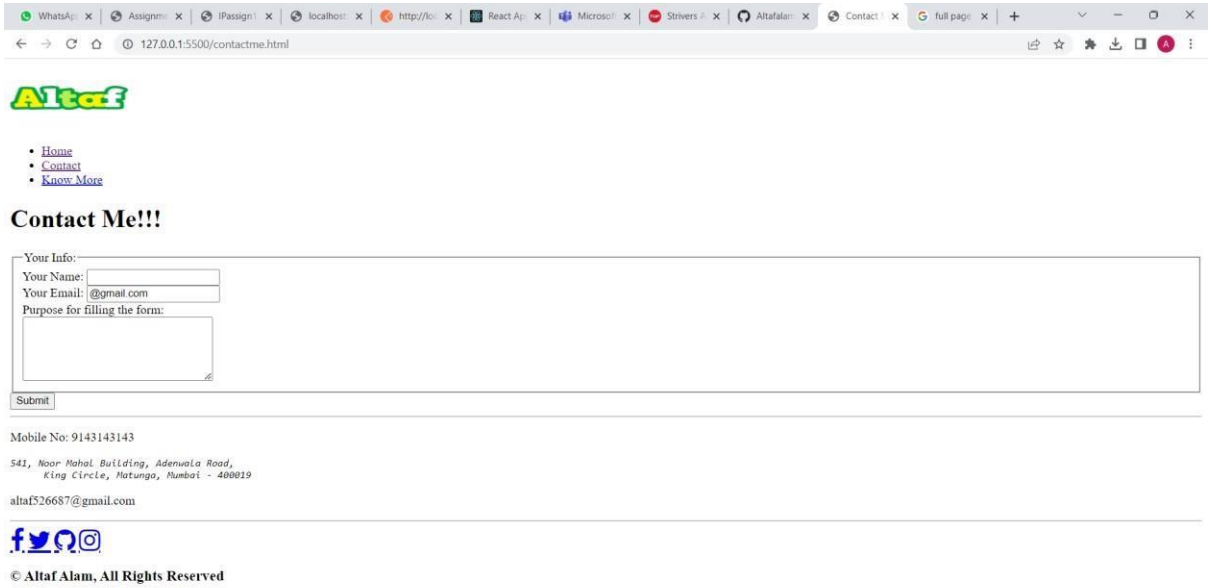
Screenshot copied to clipboard and saved

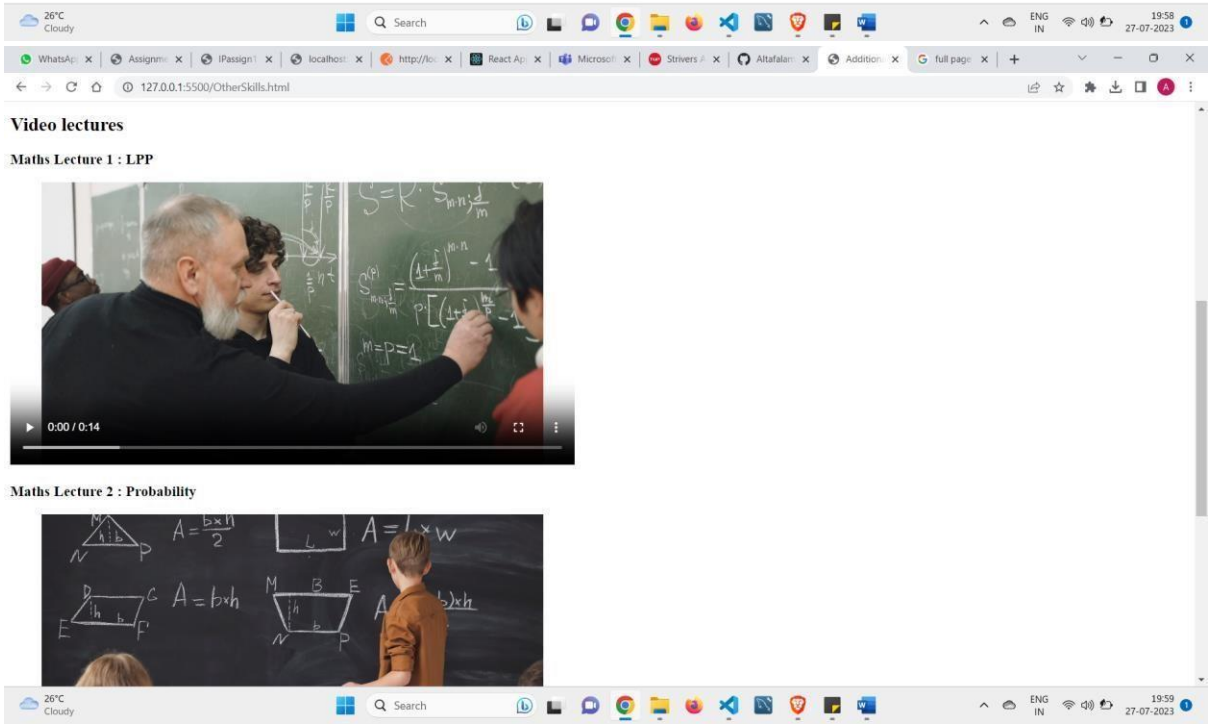
Select here to mark up and share the image

© Altaf Alam, All Rights Reserved

Name : Altaf Alam

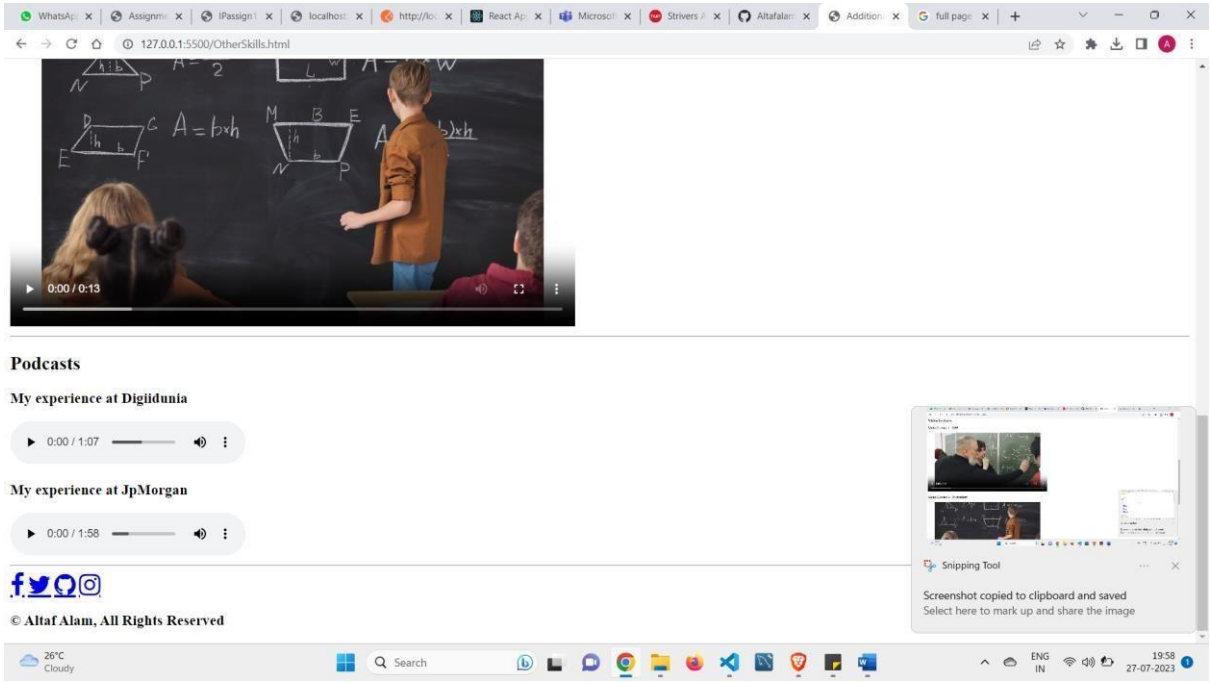
Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 18/07/23





Name : Altaf Alam

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



Conclusion :

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

Name : Altaf Alam

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

Assignment No 2

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

Lab Outcome :

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

Theory :

What is CSS?

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

Types of CSS:

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

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

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

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

Pros:
Separates the presentation (styles) from the content (HTML), leading to better code organization.
Allows for reusability of styles across multiple elements within the same HTML file.

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

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

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

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

2. CSS Font Properties

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

3. CSS Text Properties

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

4. Table Properties

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

Name : Altaf Alam

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

5. List Properties

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

6. Background

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

7. CSS Selectors

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

8. Pseudo Classes

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

9. Pseudo Elements

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

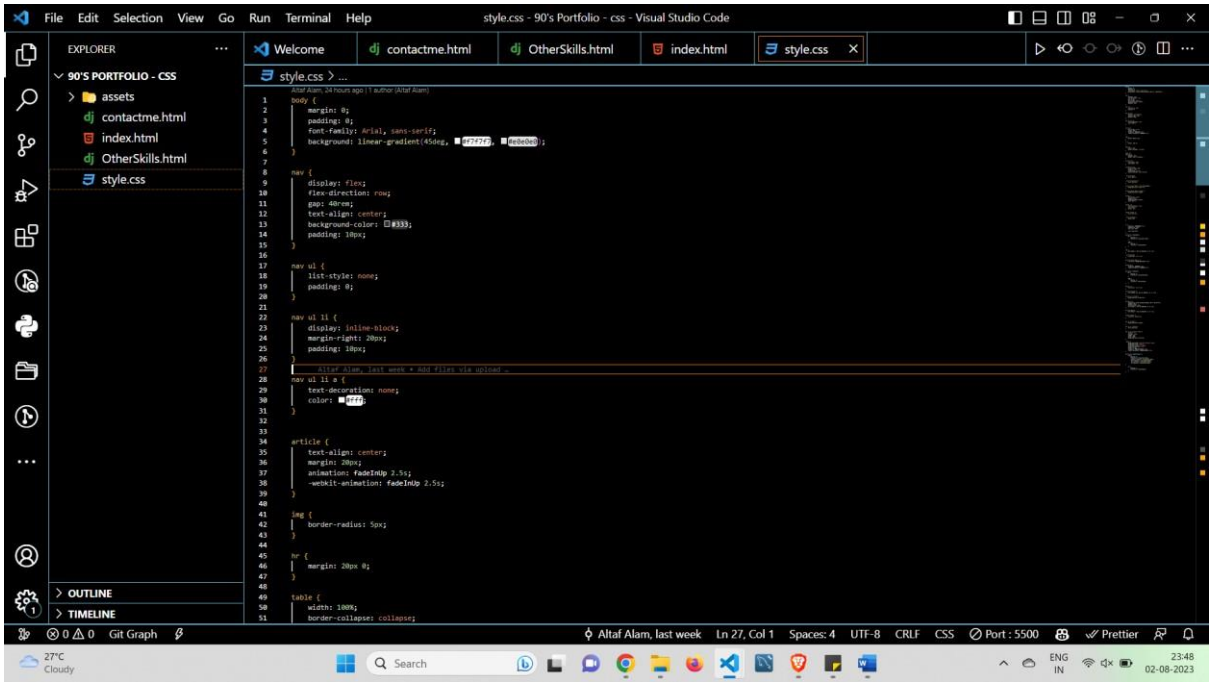
Name : Altaf Alam

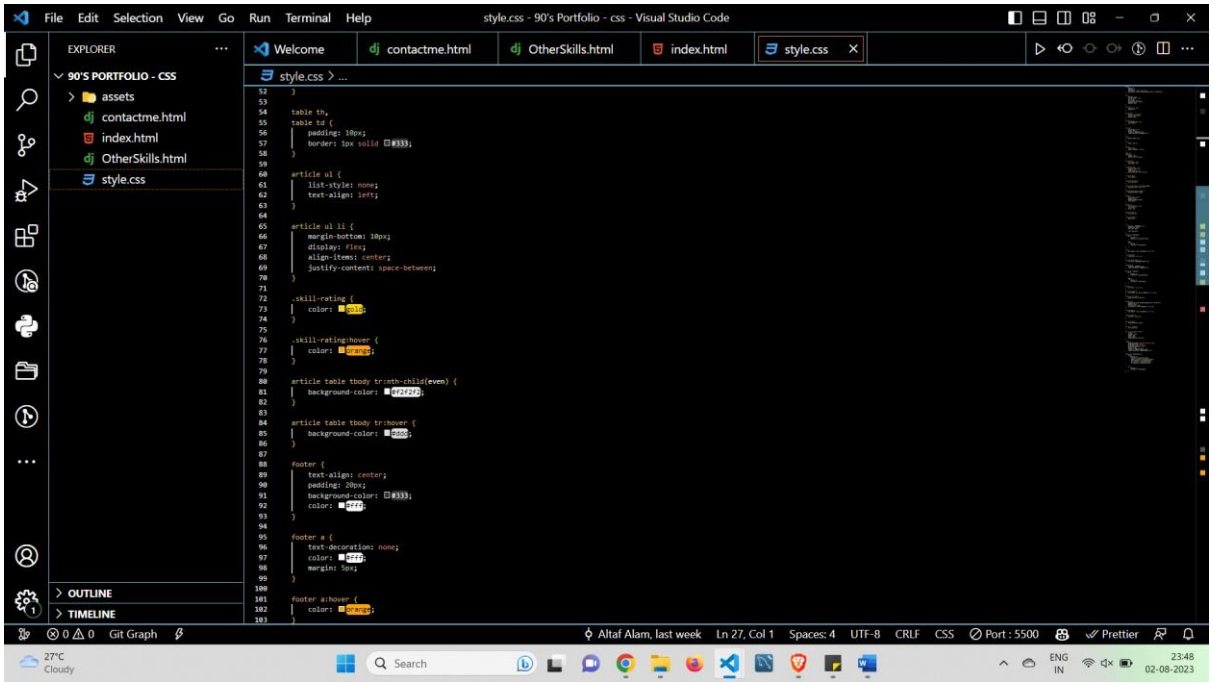
Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 26/07/23

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

Source Code :

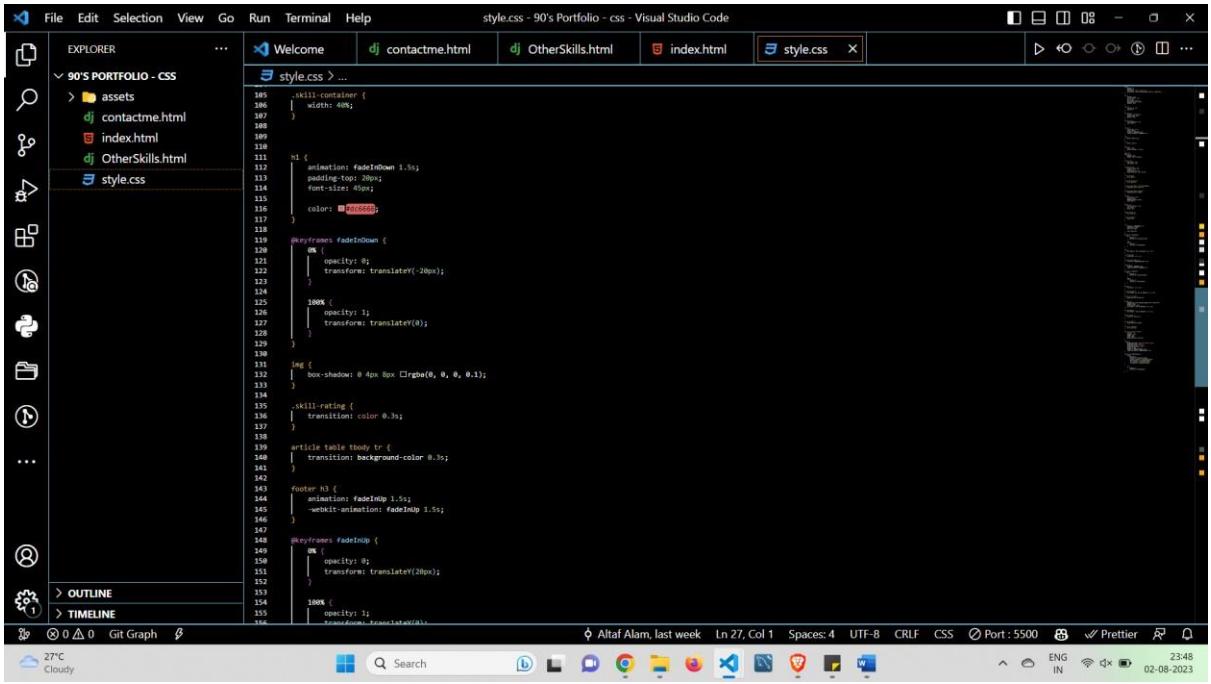
Style.css





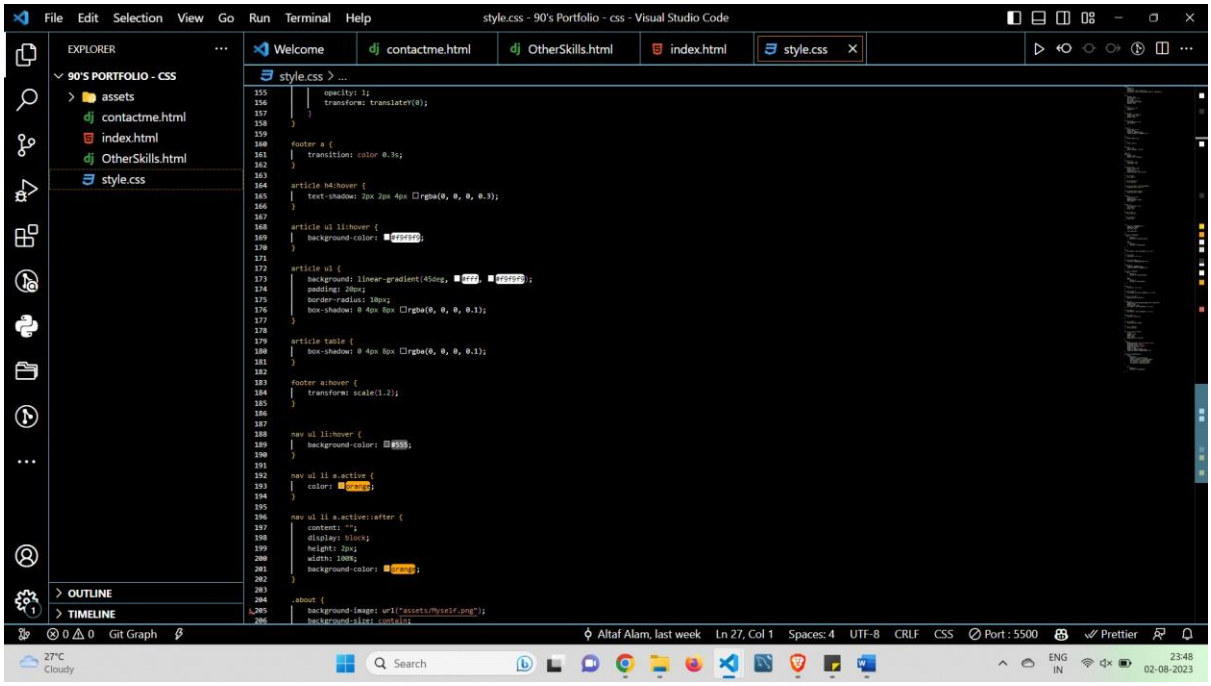
Name : Altaf Alam

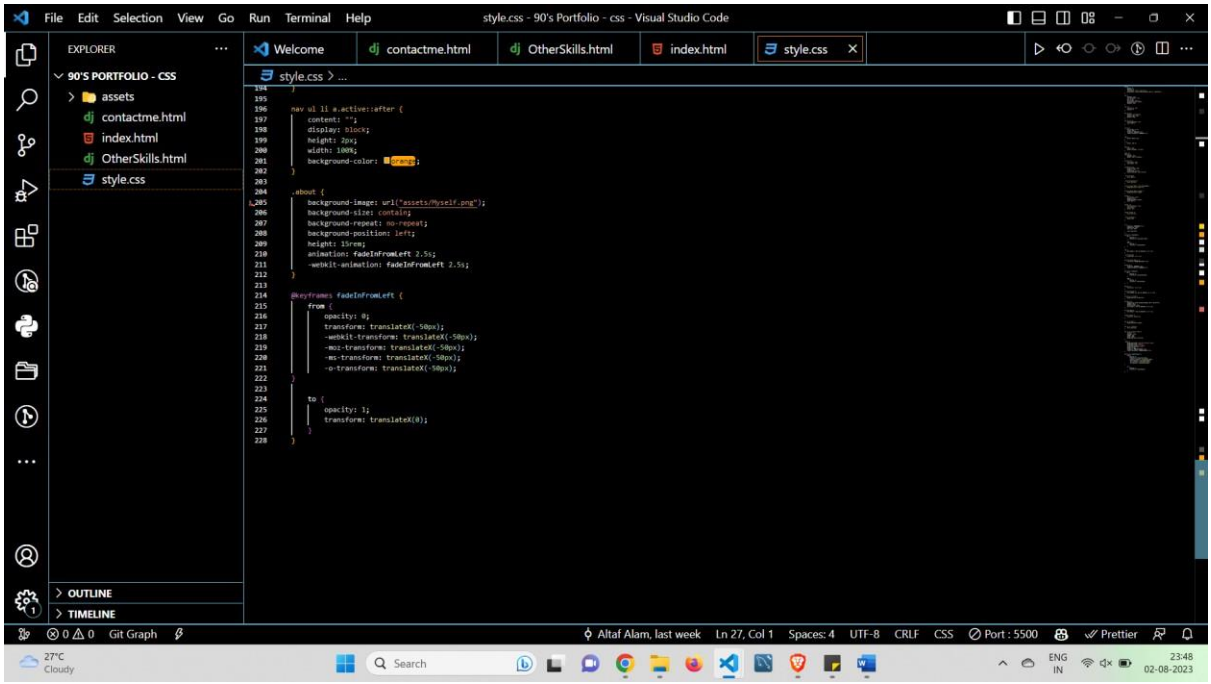
Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 26/07/23



Name : Altaf Alam

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

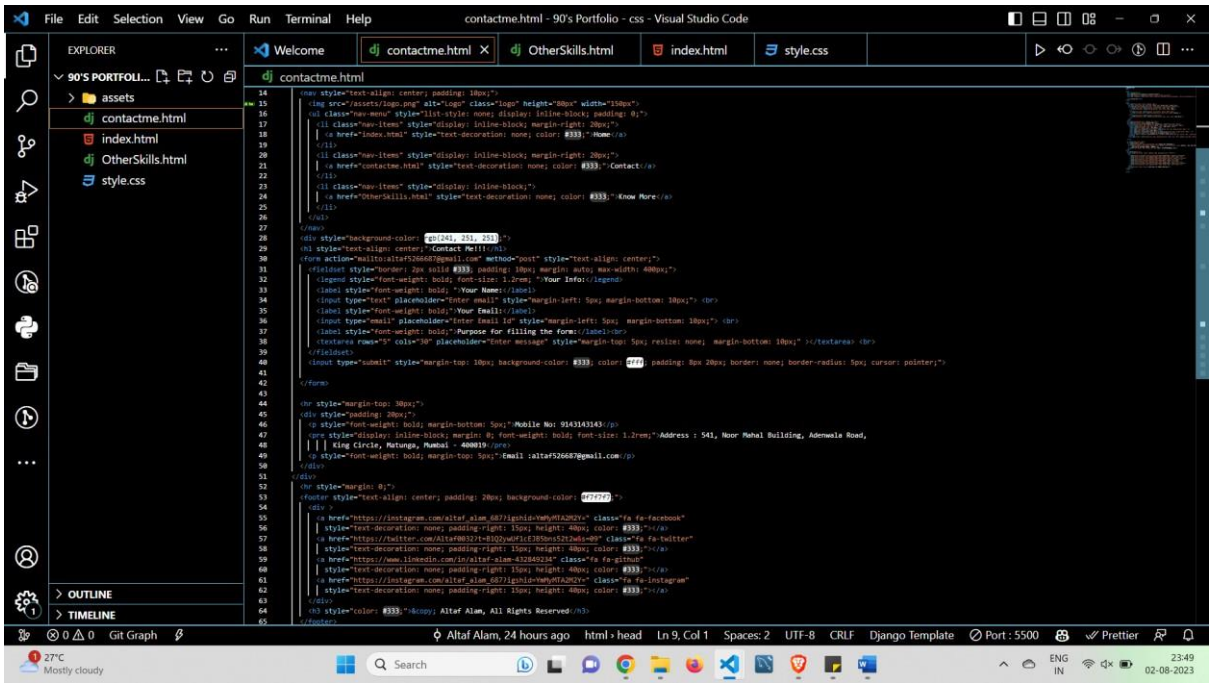




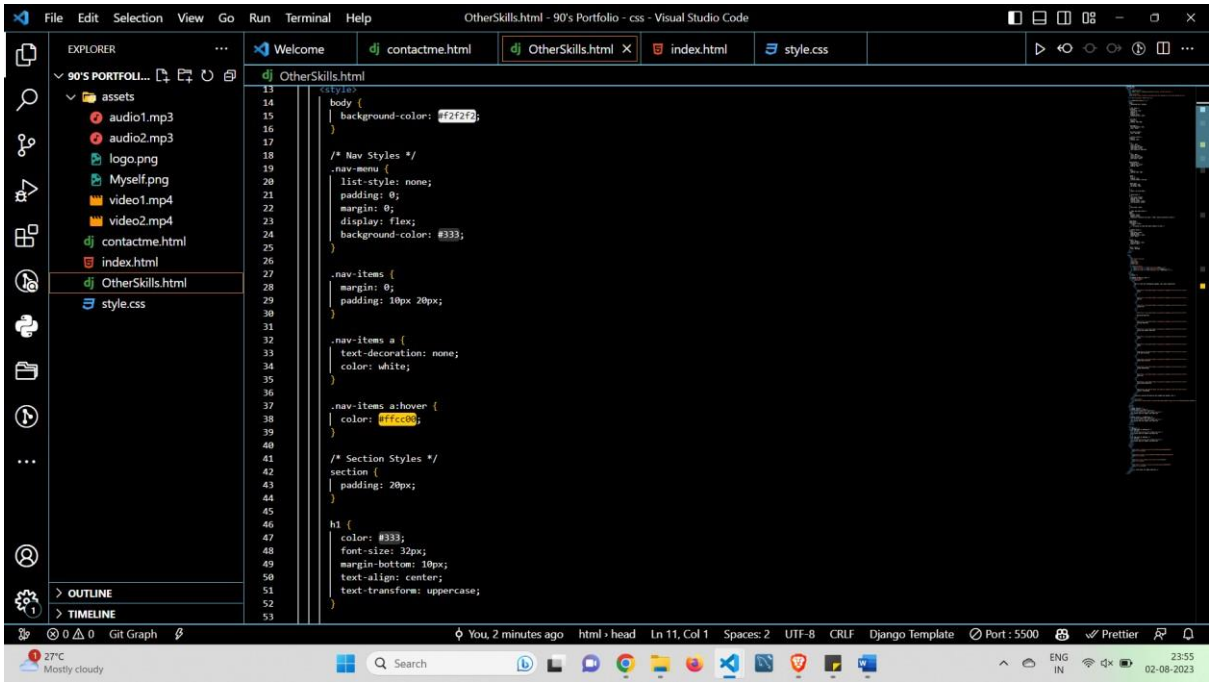
Name : Altaf Alam

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

Contactme.html

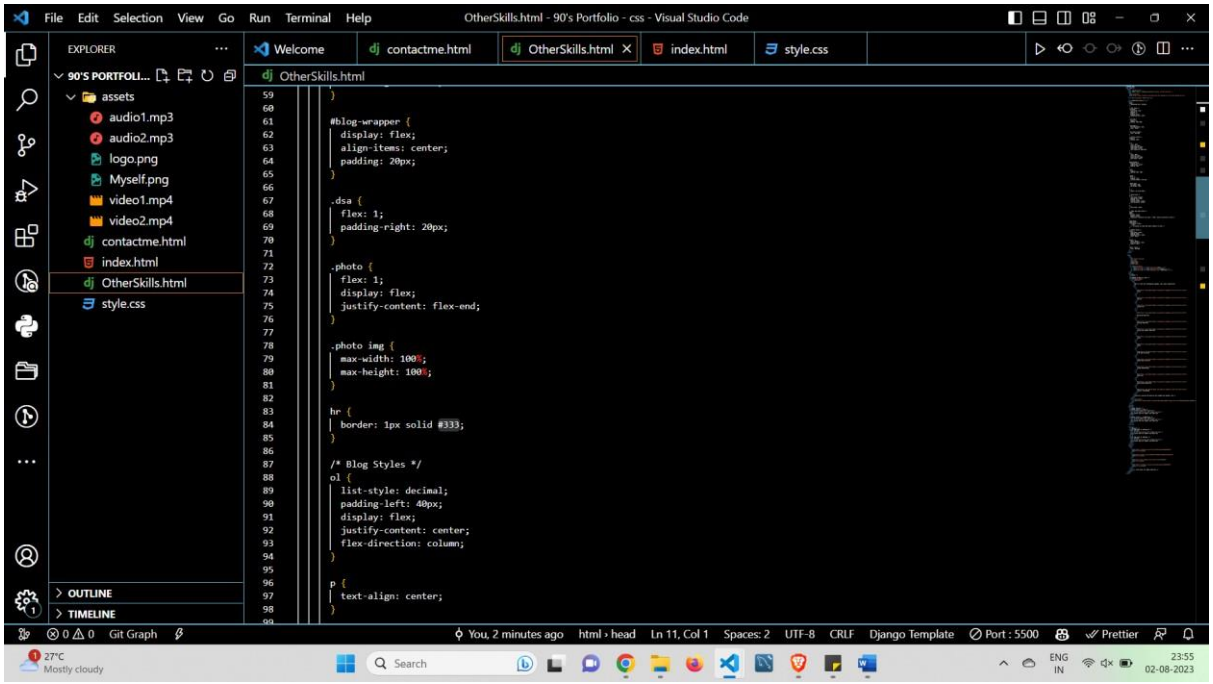


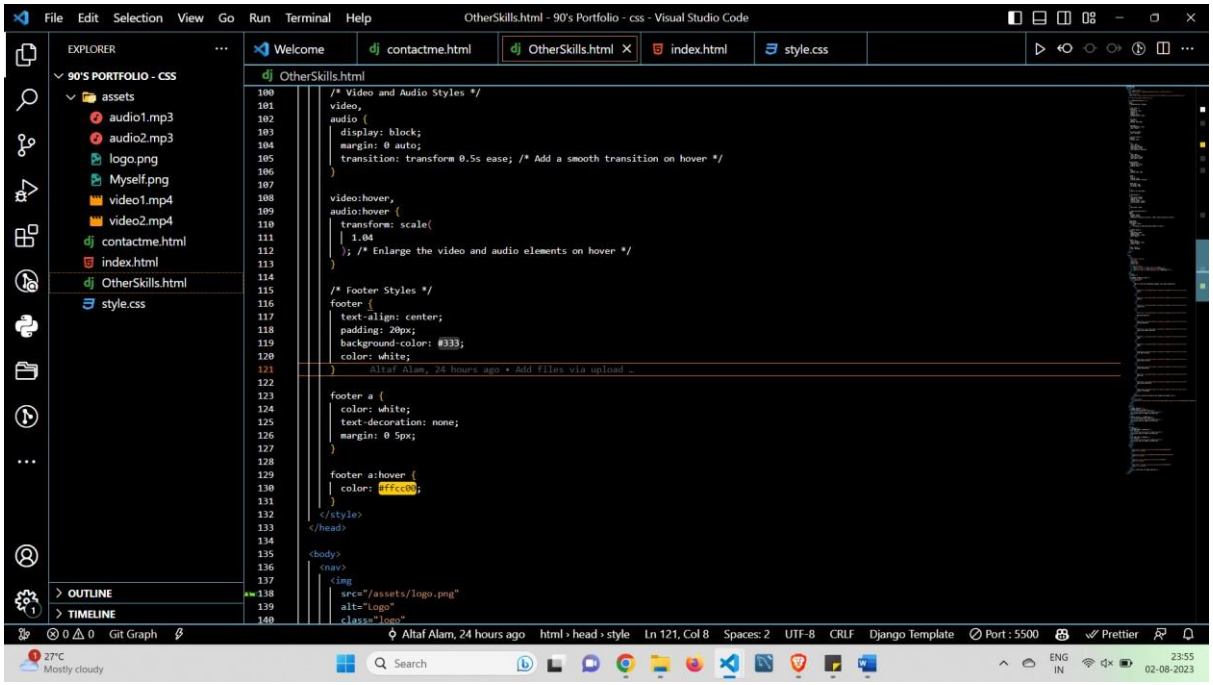
OtherSkills.html



Name : Altaf Alam

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

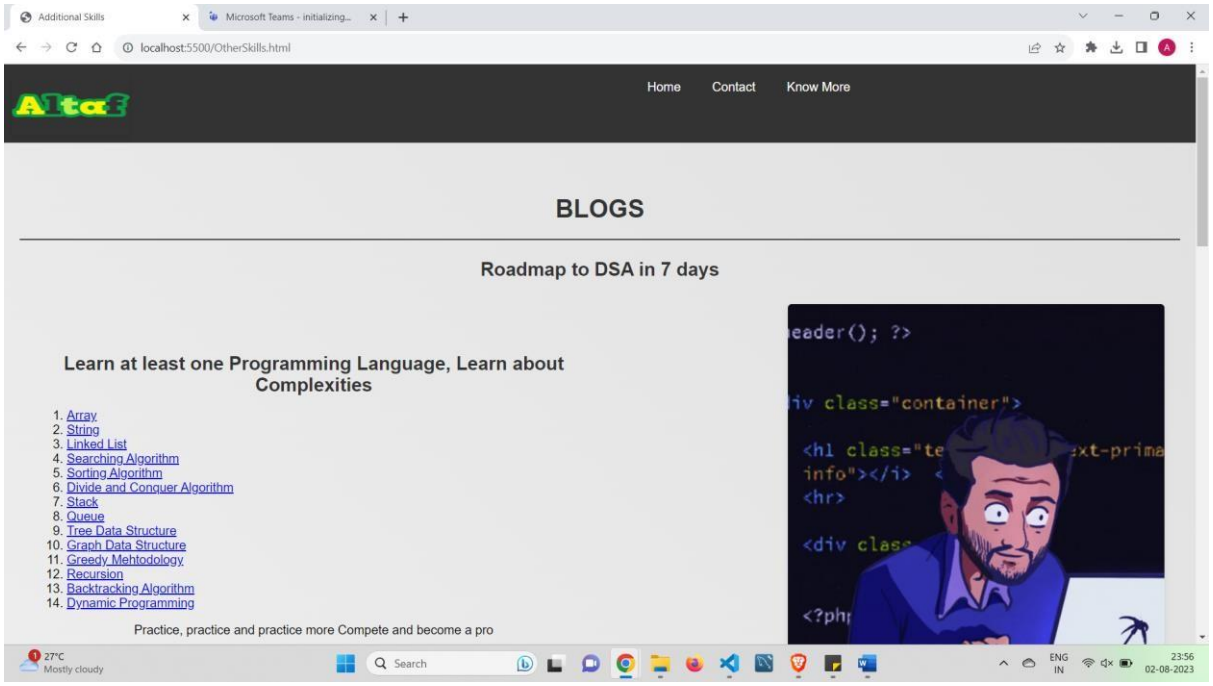


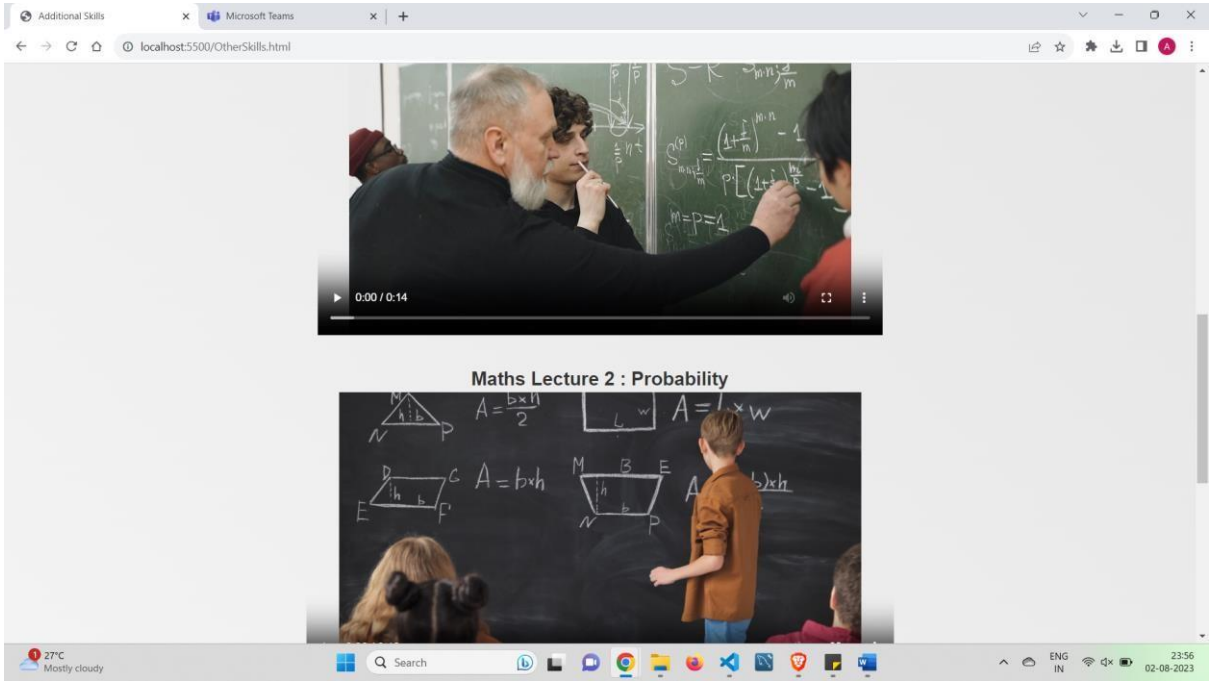


Output :

Name : Altaf Alam

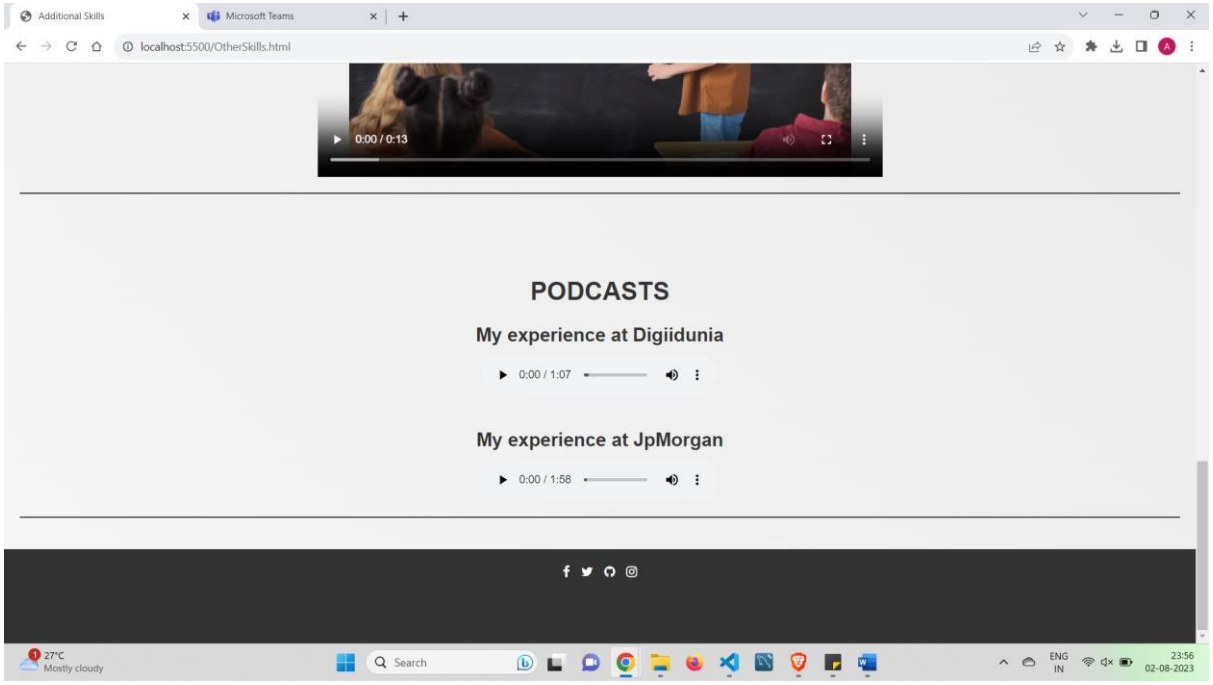
Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 26/07/23

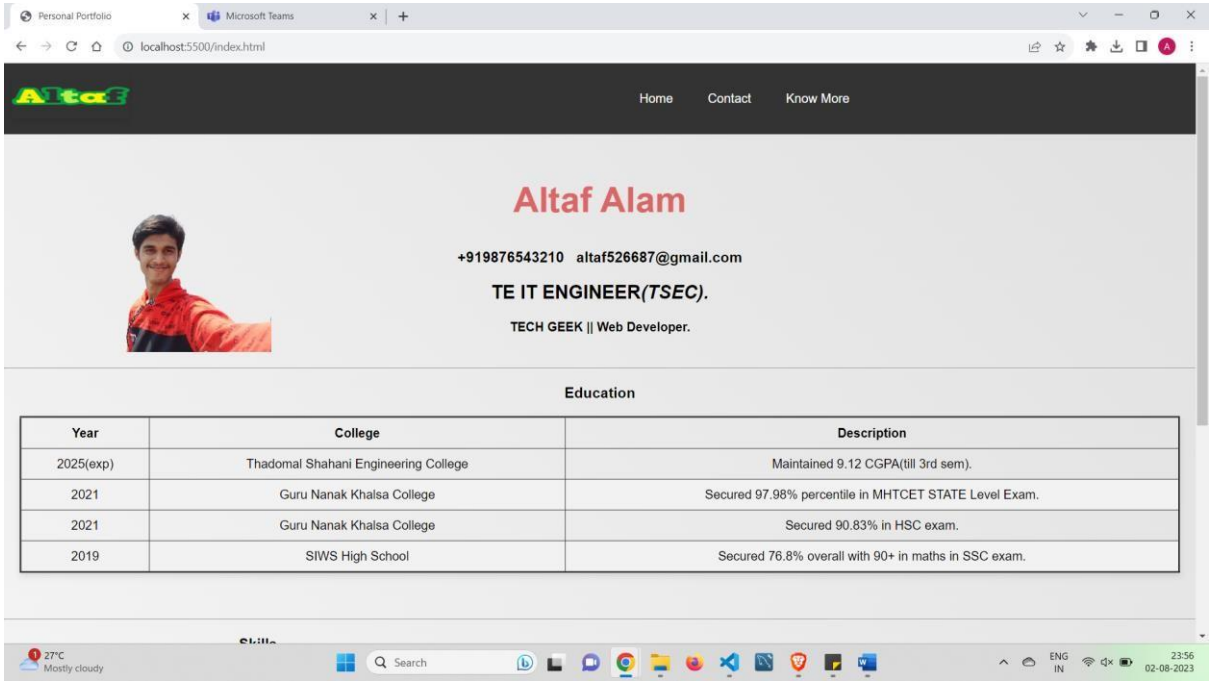




Name : Altaf Alam

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

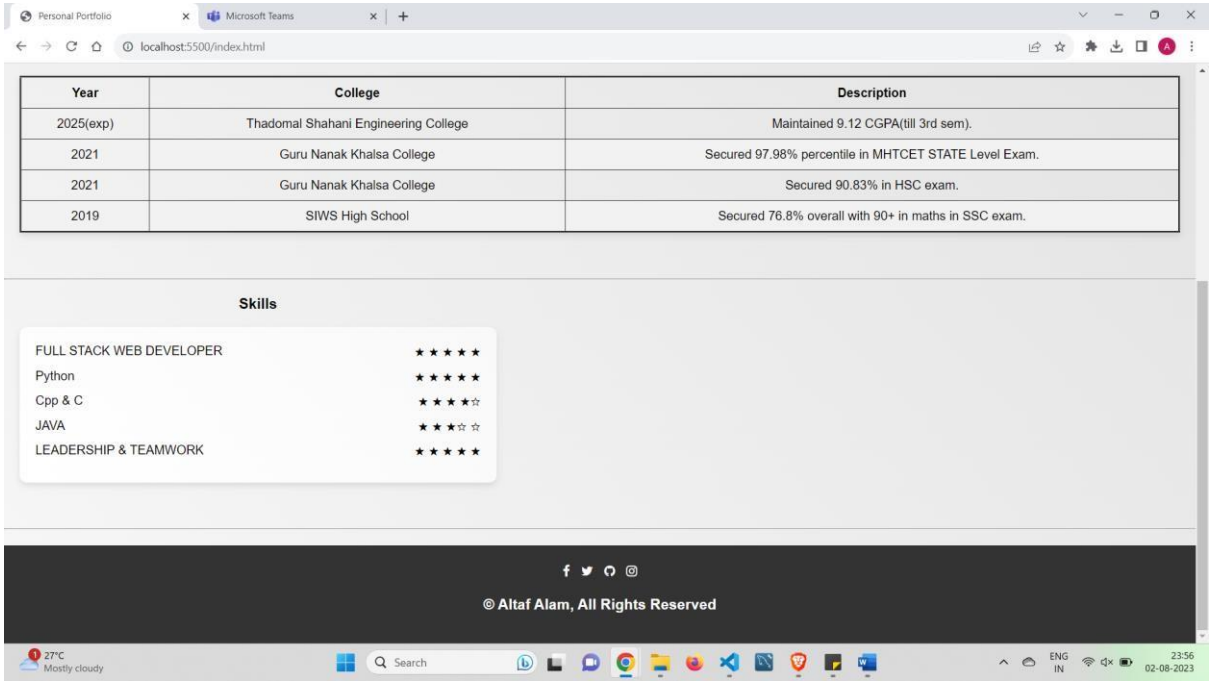


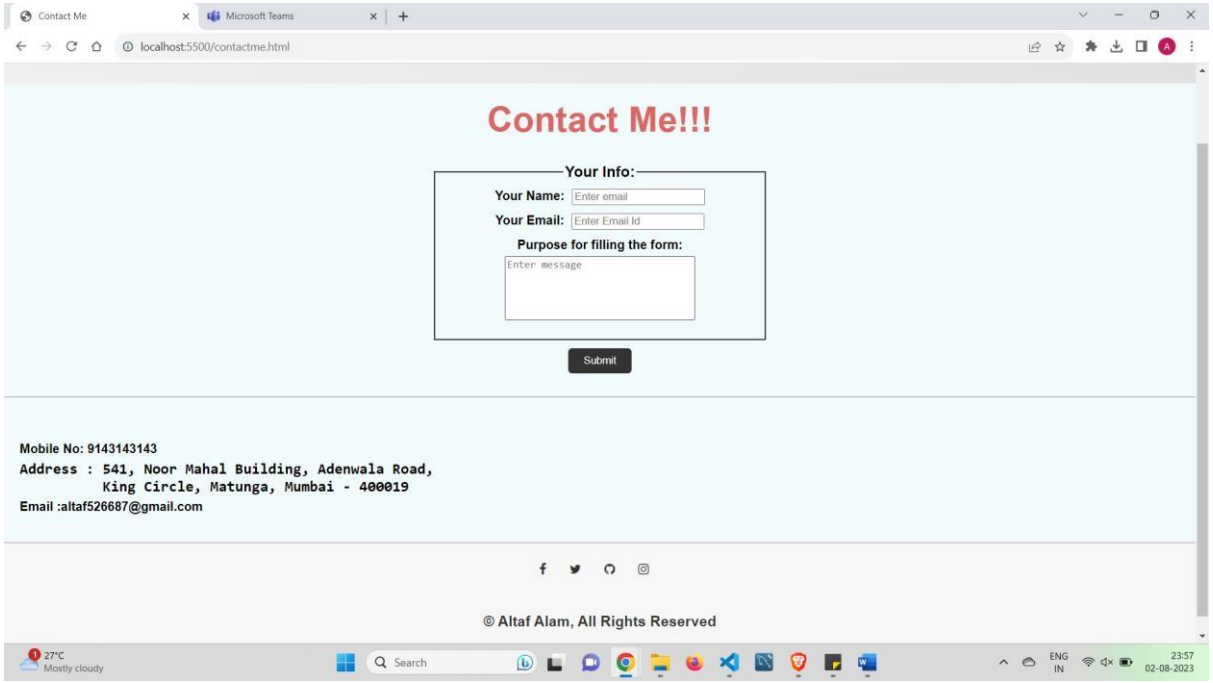


Year	College	Description
2025(exp)	Thadomal Shahani Engineering College	Maintained 9.12 CGPA(till 3rd sem).
2021	Guru Nanak Khalsa College	Secured 97.98% percentile in MHTCET STATE Level Exam.
2021	Guru Nanak Khalsa College	Secured 90.83% in HSC exam.
2019	SIWS High School	Secured 76.8% overall with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

Name : Altaf Alam

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





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 userfriendly and efficient.

Name : Altaf Alam

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

Assignment No 3

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

Lab Outcome :

LO3: To expose students to developing responsive layouts.

Theory :

What is CSS?

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

Uses of Bootstrap:

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

Bootstrap Classes in Detail:

1. Grid System:

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

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

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

2. Forms:

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

3. Buttons:

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

4. Navbar:

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

5. Breadcrumbs:

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

6. Jumbotron:

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

7. Carousel:

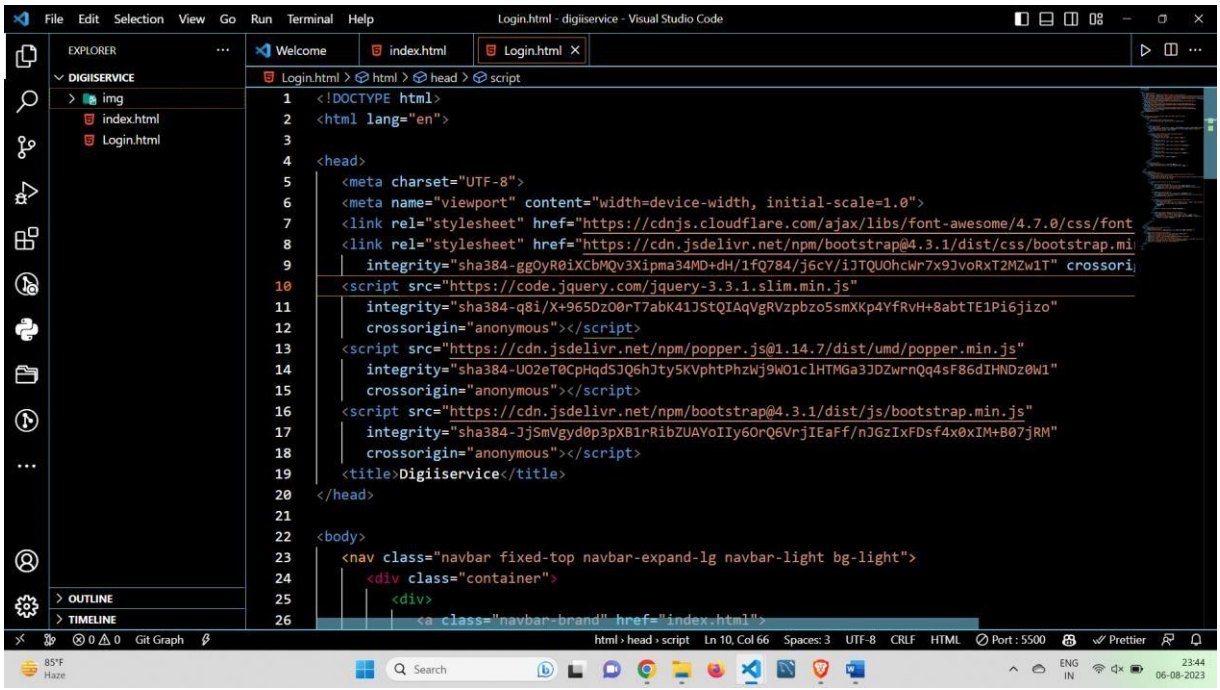
Bootstrap's carousel component allows you to create an image slider or a carousel of content. The carousel is based on a series of `.carousel-item` elements, each containing the content to be displayed. The `.carousel-indicators` class provides navigation dots, and the `.carousel-control-prev` and `.carousel-control-next`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

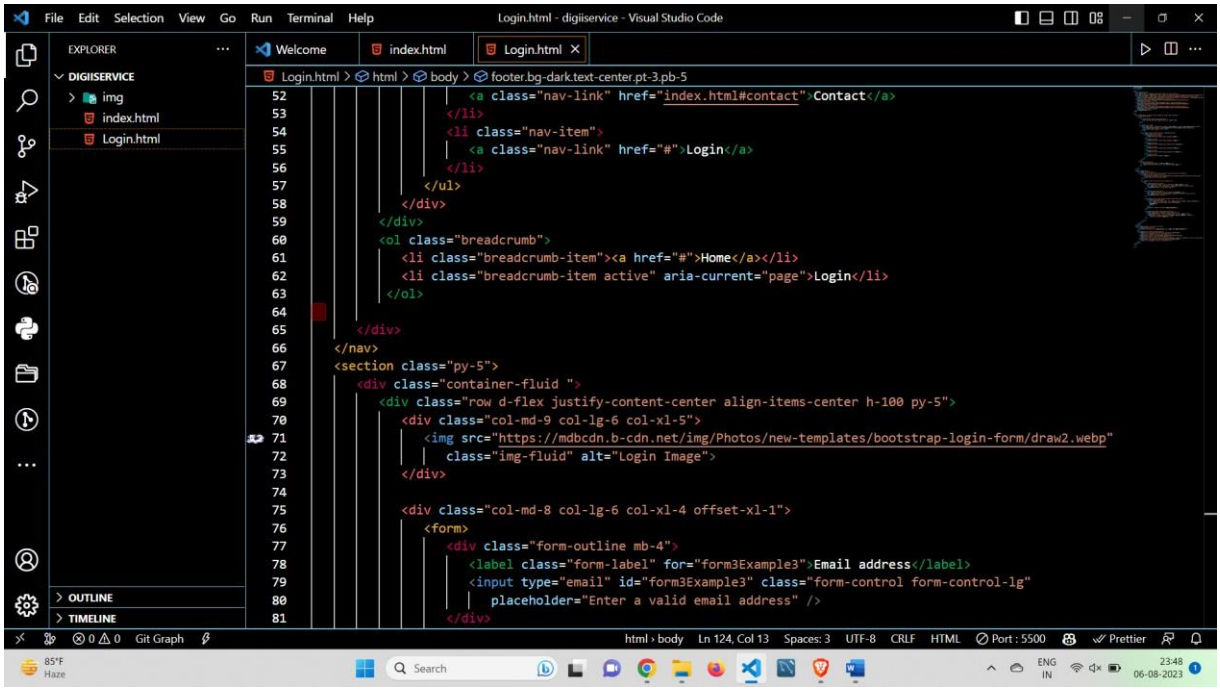
These Bootstrap classes and components simplify the process of building responsive, visually appealing, and user-friendly websites and web applications. By utilizing these elements, developers can create consistent and professional-looking interfaces with less effort and code.

Name : Altaf Alam

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

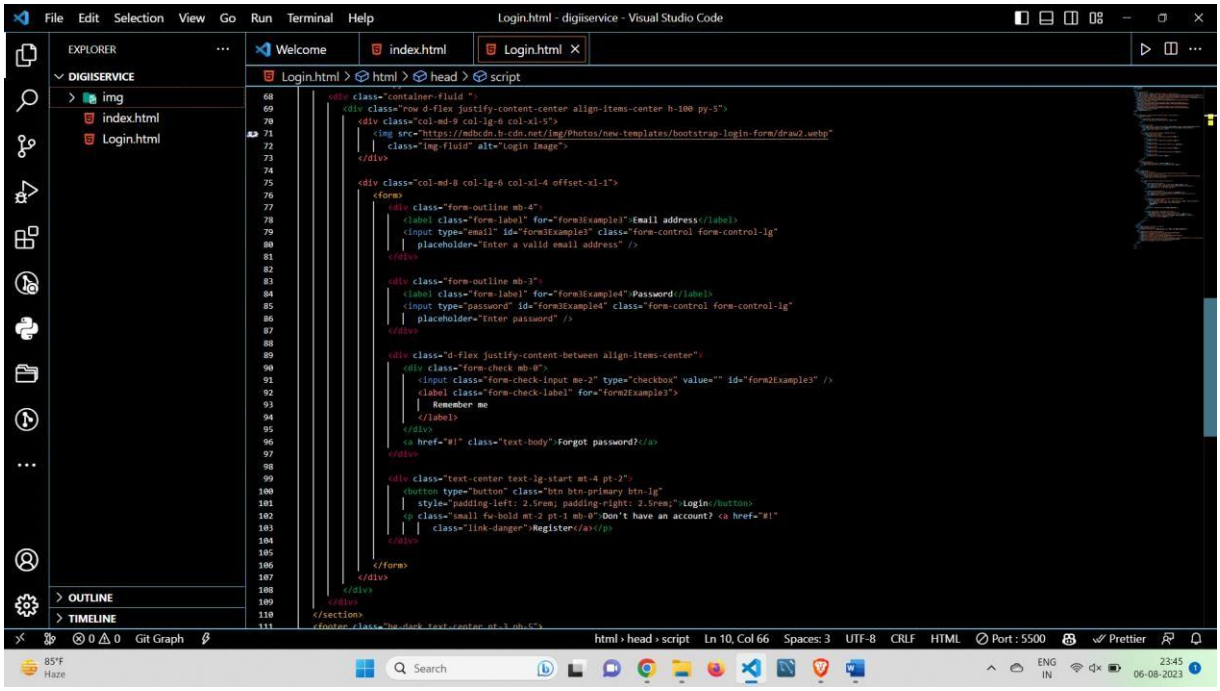
Source Code : Index.html

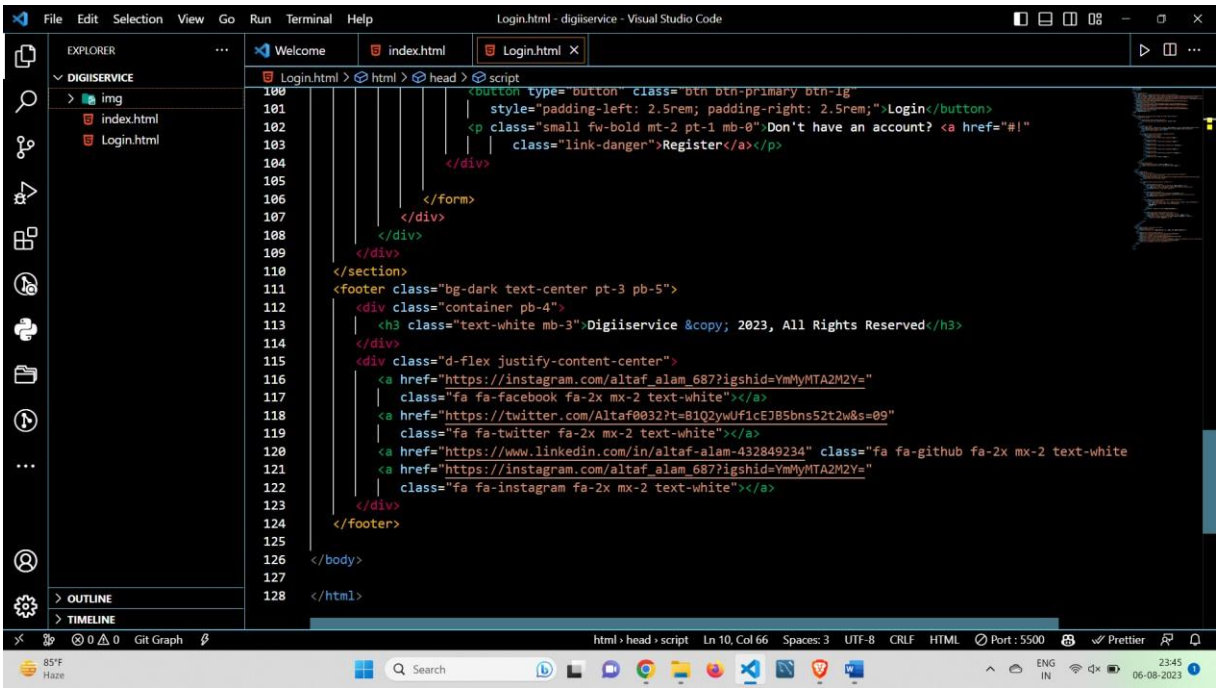




Name : Altaf Alam

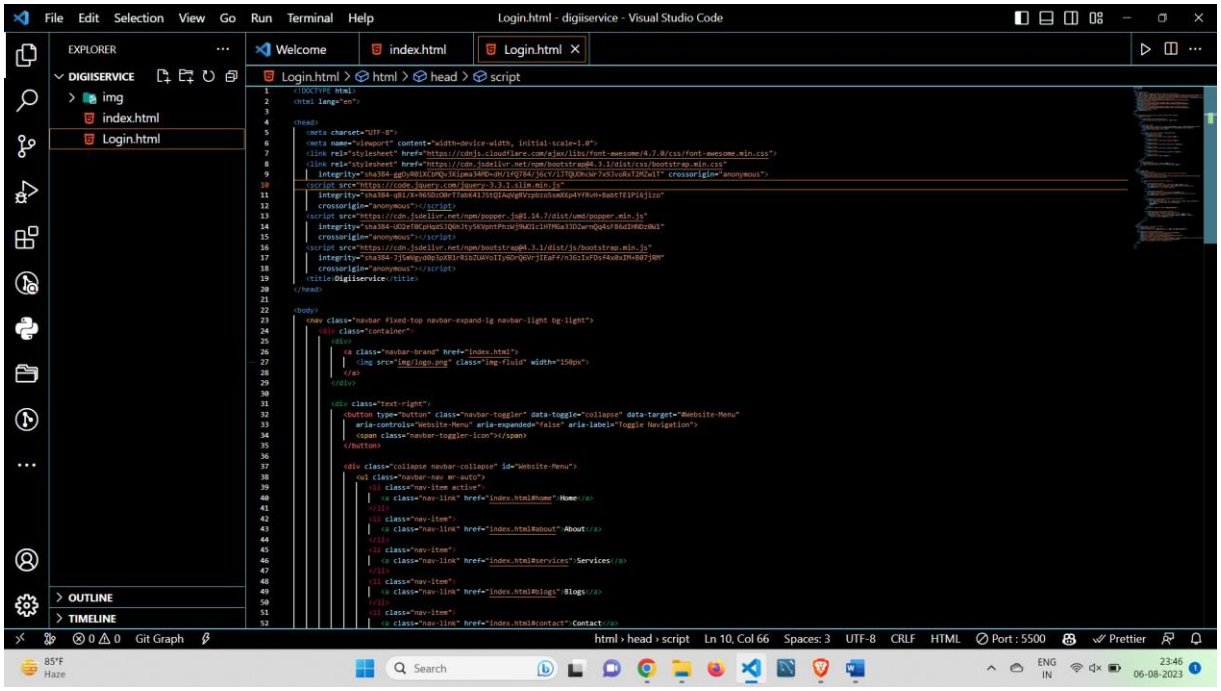
Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

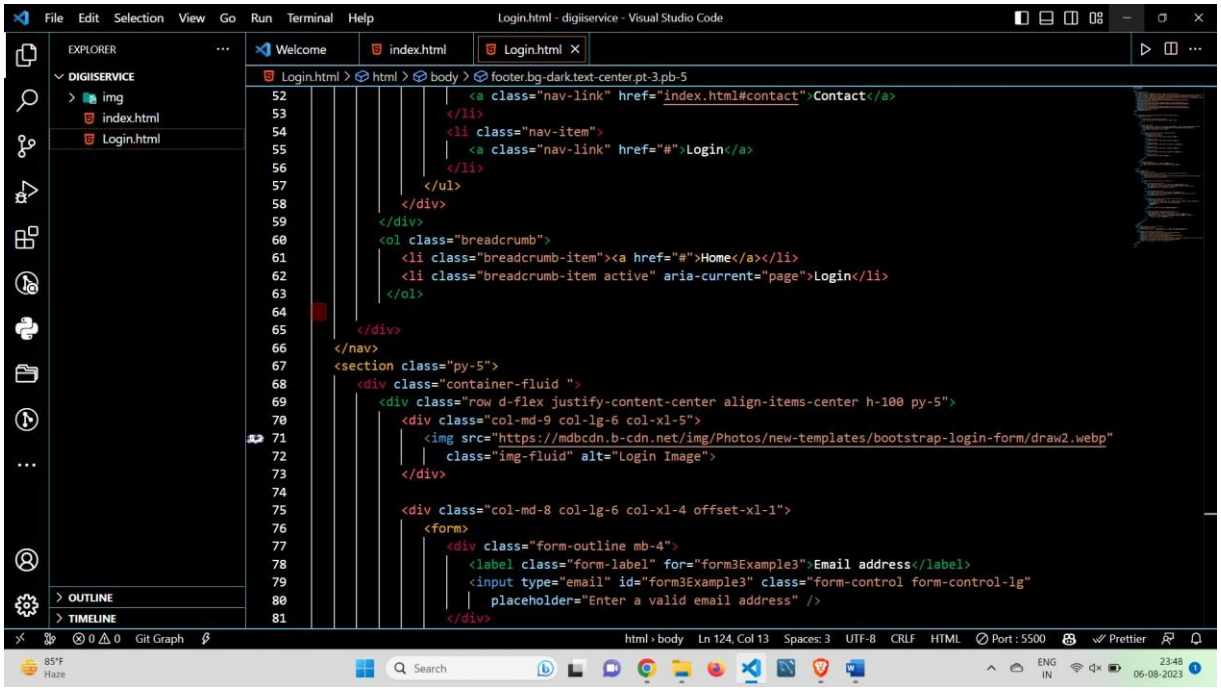




Name : Altaf Alam

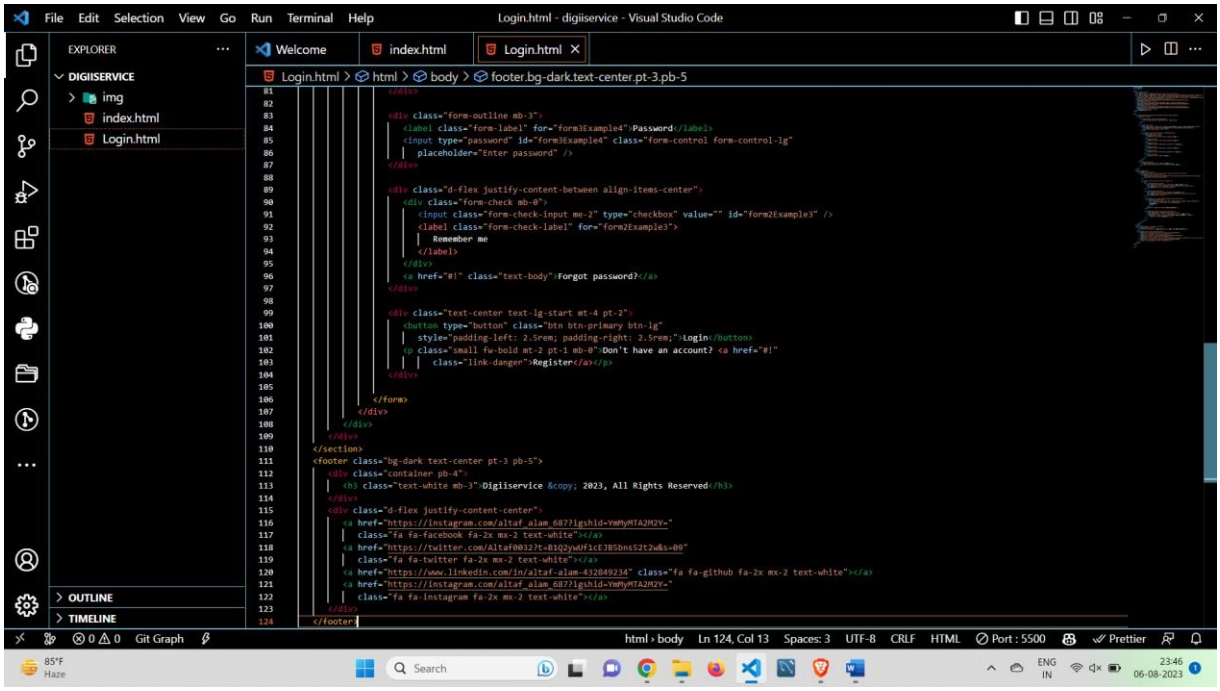
Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23



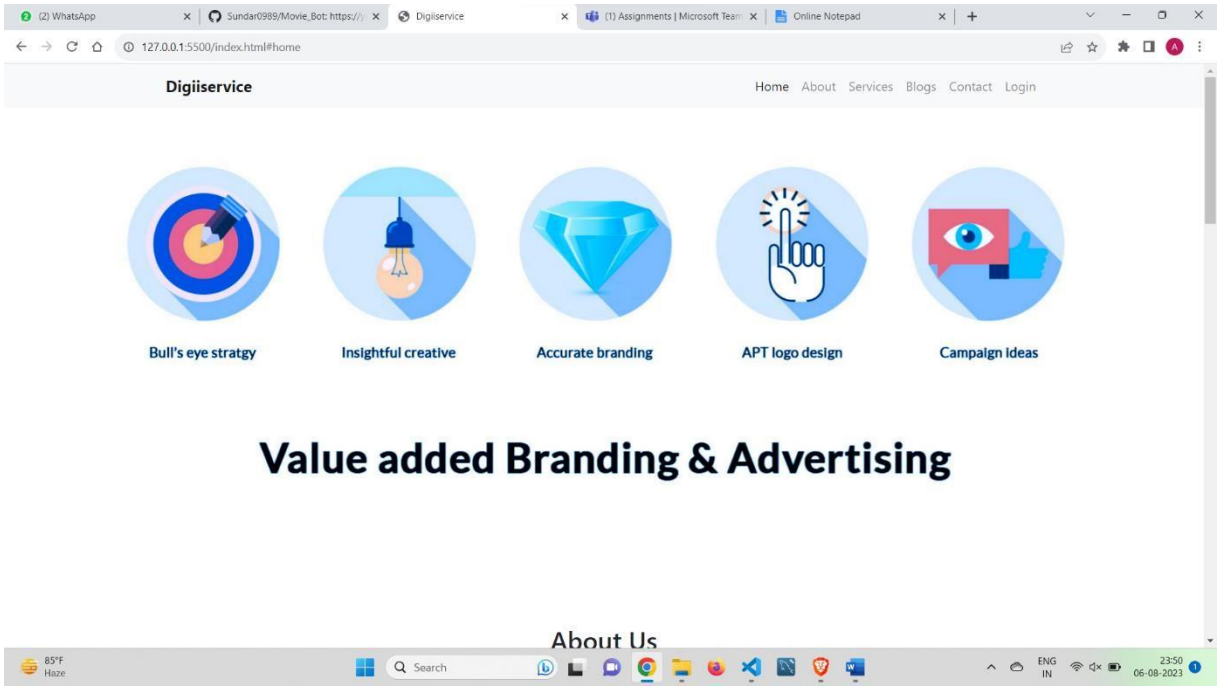


Name : Altaf Alam

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

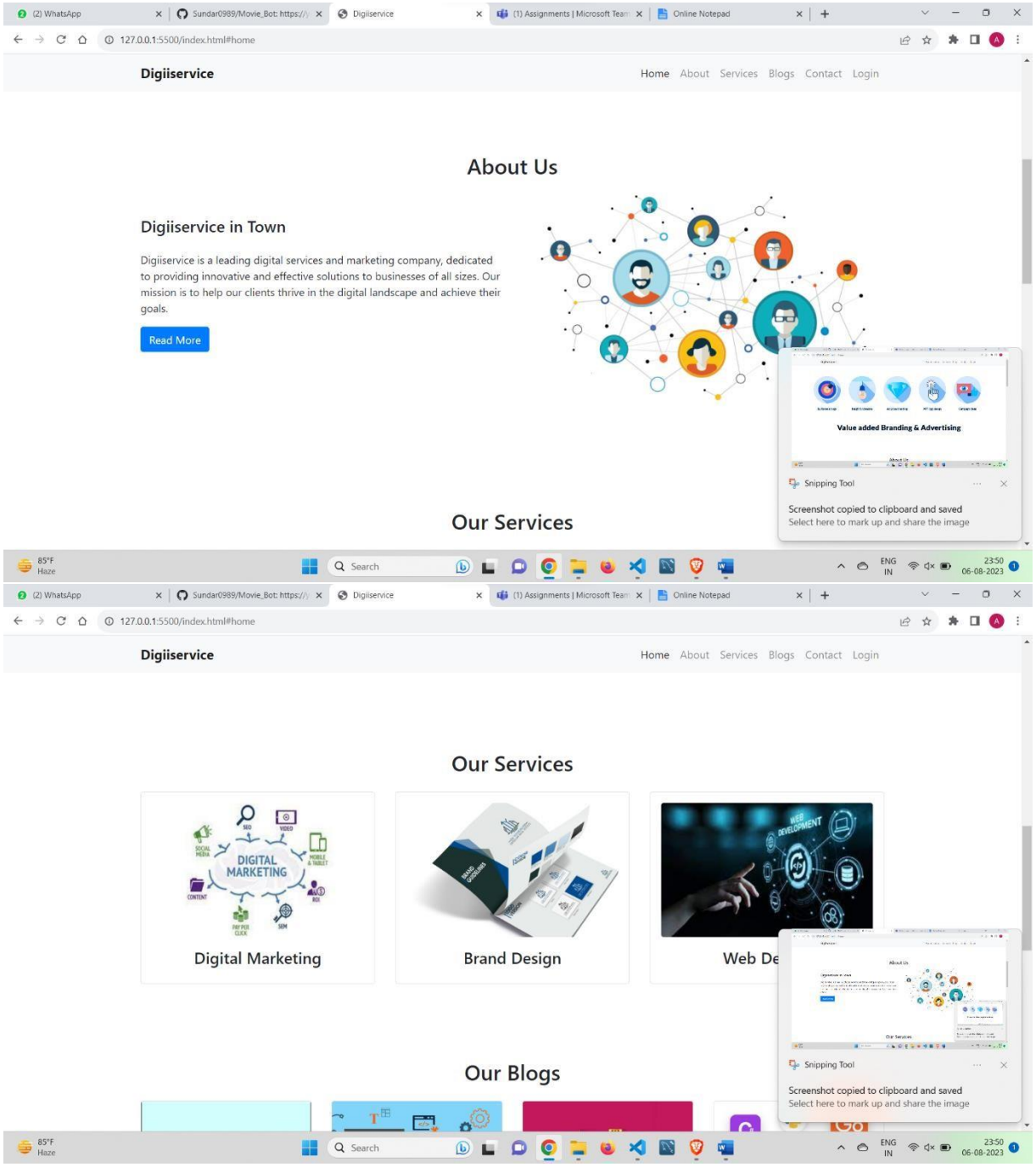


Output :



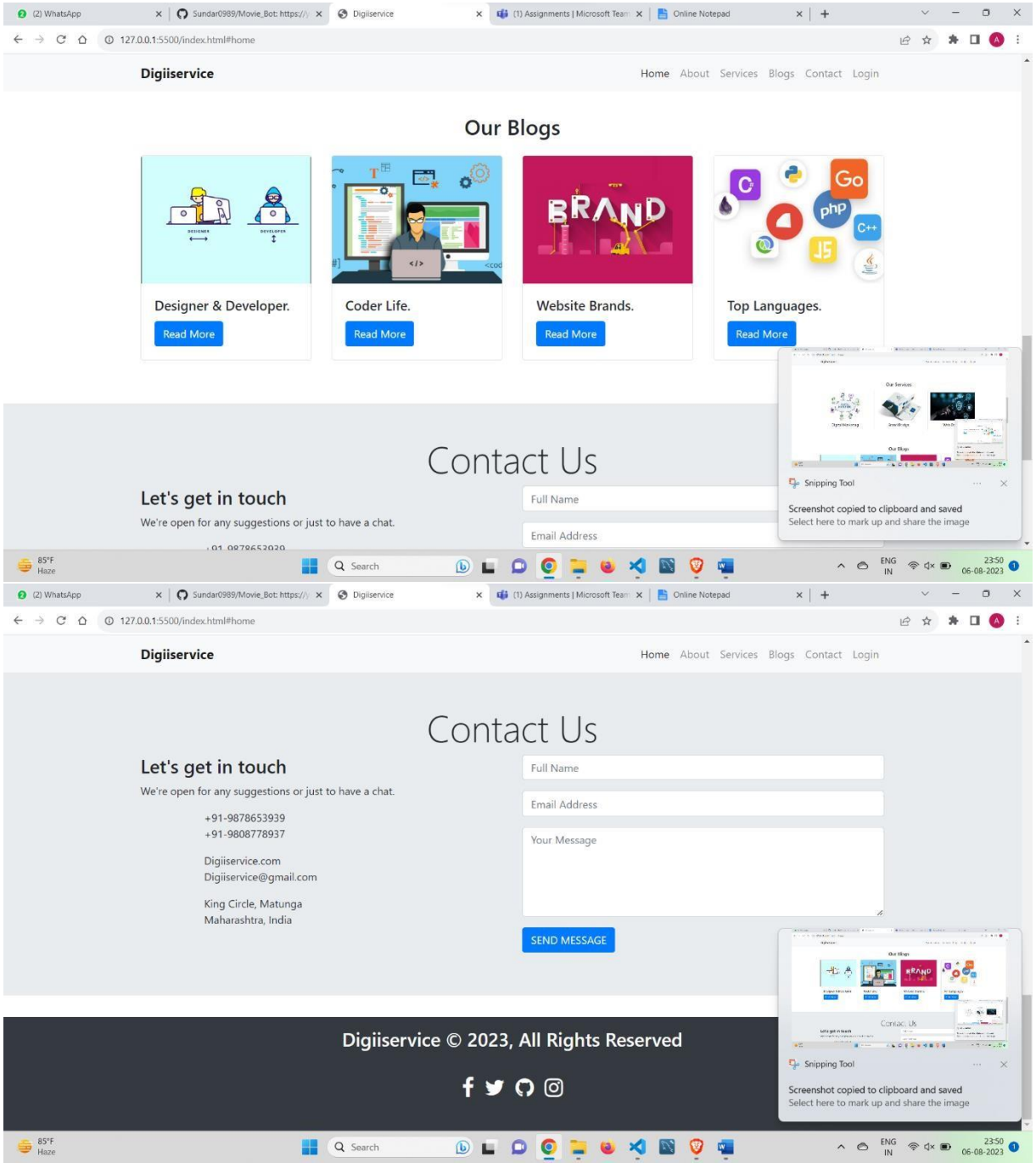
Name : Altaf Alam

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



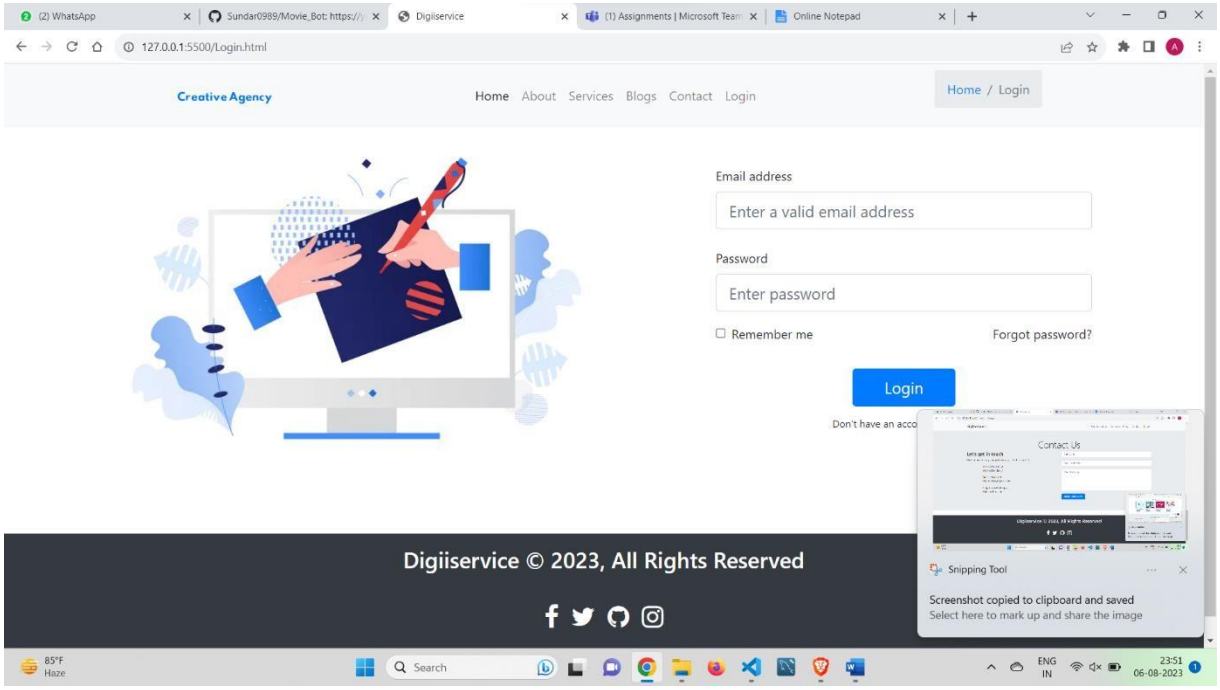
Name : Altaf Alam

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



Name : Altaf Alam

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



Conclusion :

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

Name : Altaf Alam

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

Assignment No 4 (a)

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

Lab Outcome :

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

Theory :

Conditionals :

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

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

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

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

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

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

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

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

Loops :

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

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

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

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

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

5. for...of Loop:

Name : Altaf Alam

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

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

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

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

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

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

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

4. IIFE (Immediately Invoked Function Expression):
   An IIFE is a function that is defined and executed immediately after its creation. It's enclosed in parentheses to ensure it's treated as an expression rather than a declaration.

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

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

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

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

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

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

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

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

13. Rest Parameters and Spread Operator:
Functions can use the rest parameter syntax to accept an indefinite number of arguments as an array. Conversely, the spread operator can spread an array into individual arguments when calling a function.

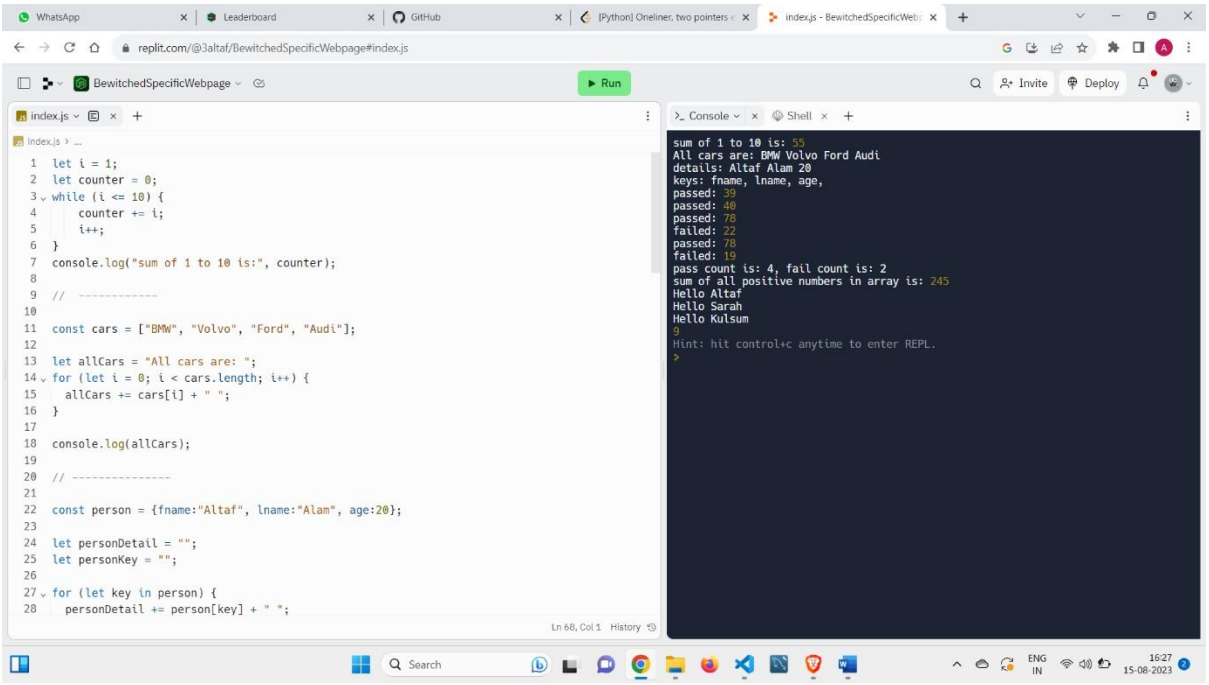
Understanding these different types of functions in JavaScript equips developers with the tools to create well-structured, efficient, and maintainable code. Each type has its own advantages and use cases, contributing to the flexibility and expressiveness of the language.

Name : Altaf Alam

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

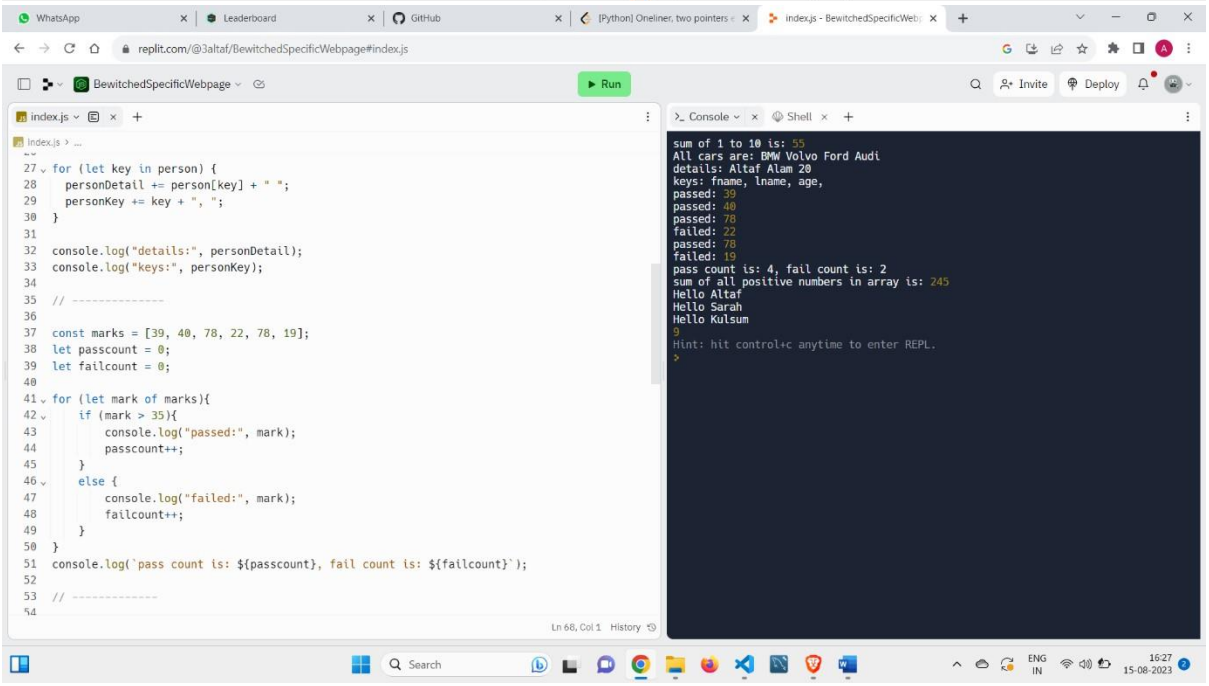
Source Code & Output :

index.js



```
let i = 1;
let counter = 0;
while (i <= 10) {
    counter += i;
    i++;
}
console.log("sum of 1 to 10 is:", counter);

//  ------------

const cars = ["BMW", "Volvo", "Ford", "Audi"];

let allCars = "All cars are: ";
for (let i = 0; i < cars.length; i++) {
  allCars += cars[i] + " ";
}

console.log(allCars);

// ---------------

const person = {fname:"Altaf", lname:"Alam", age:20};

let personDetail = "";
let personKey = "";

for (let key in person) {
  personDetail += person[key] + " ";
```

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

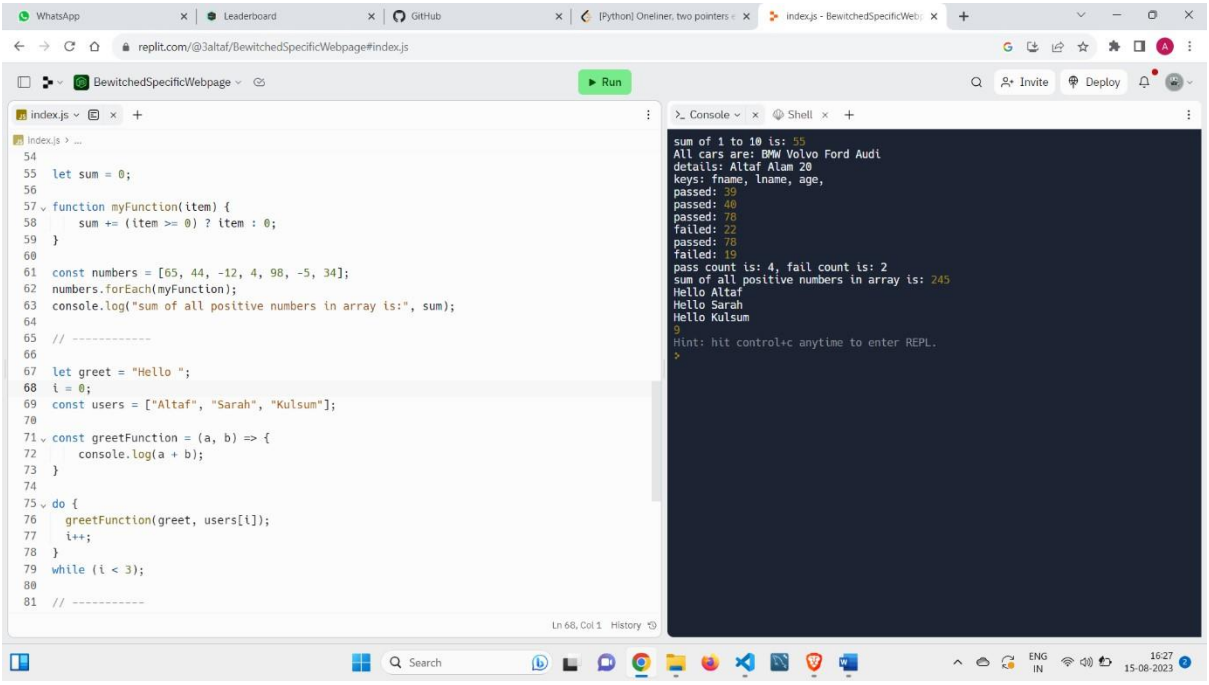


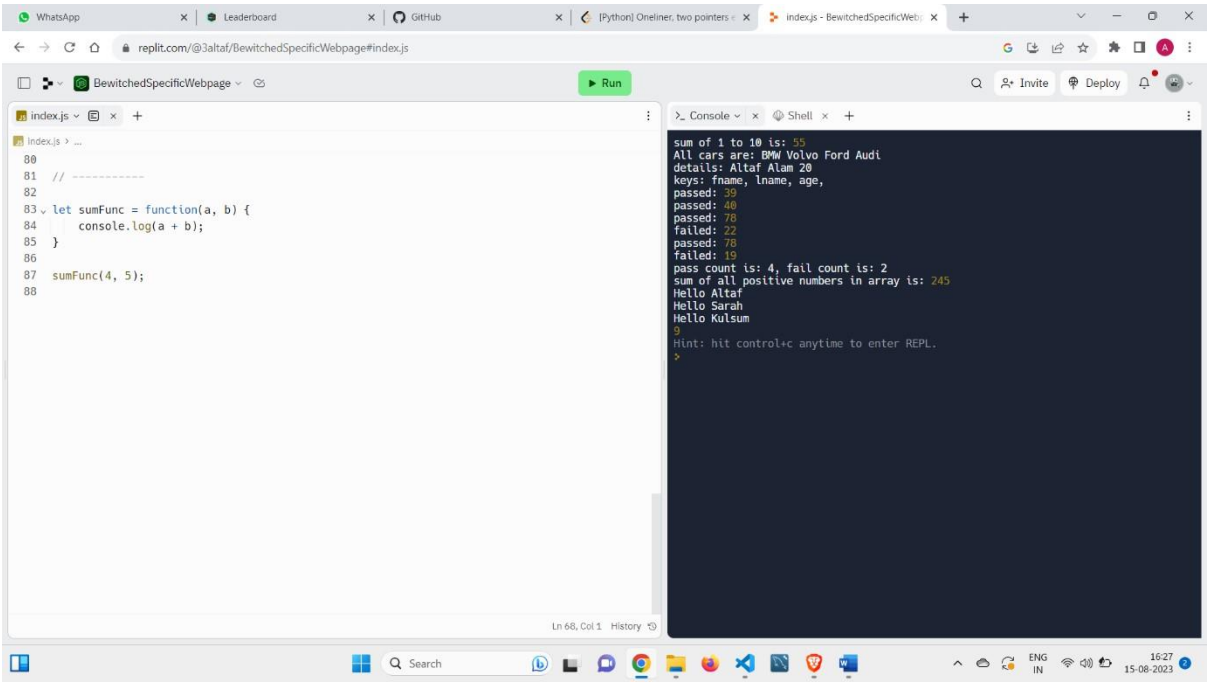
```
for (let key in person) {
  personDetail += person[key] + " ";
  personKey += key + ", ";
}

console.log("details:", personDetail);
console.log("keys:", personKey);

// ---------------

const marks = [39, 40, 78, 22, 78, 19];
let passcount = 0;
let failcount = 0;

for (let mark of marks){
    if (mark > 35){
        console.log("passed:", mark);
        passcount++;
    }
    else {
        console.log("failed:", mark);
        failcount++;
    }
}
console.log(`pass count is: ${passcount}, fail count is: ${failcount}`);

// -------------
```

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

Name : Altaf Alam

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





Conclusion :

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

Name : Altaf Alam

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

Assignment No 4 (b)

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

Lab Outcome :

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

Theory :

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

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

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

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

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

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

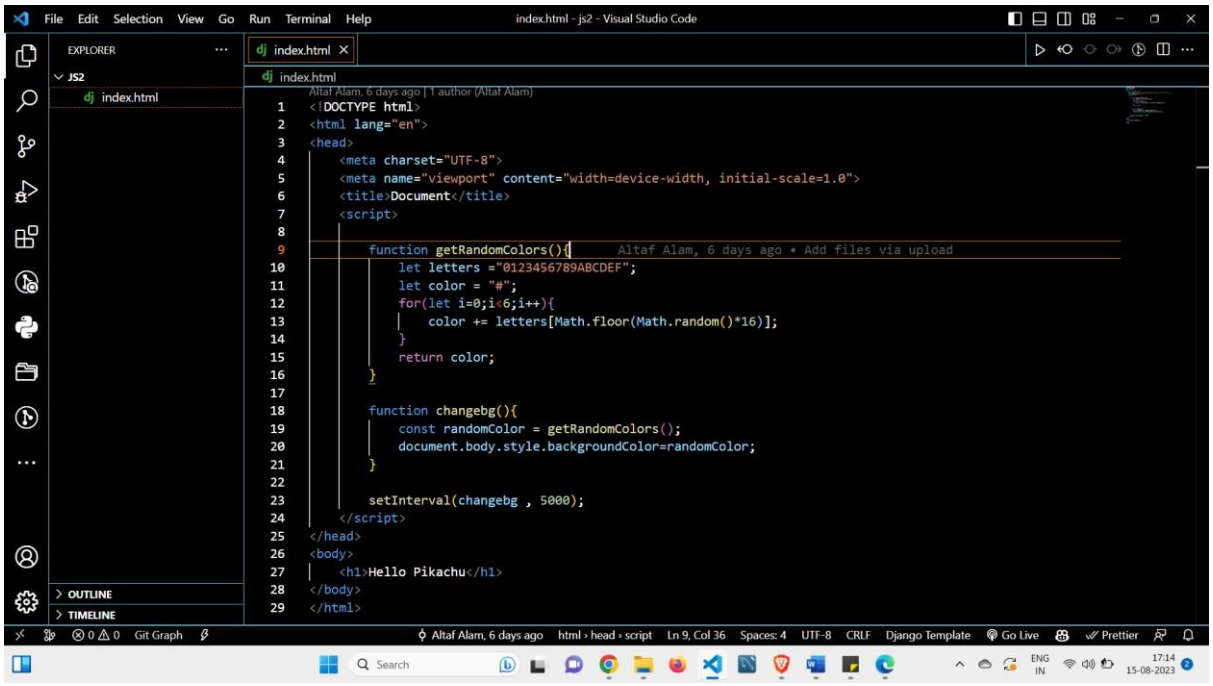
Name : Altaf Alam

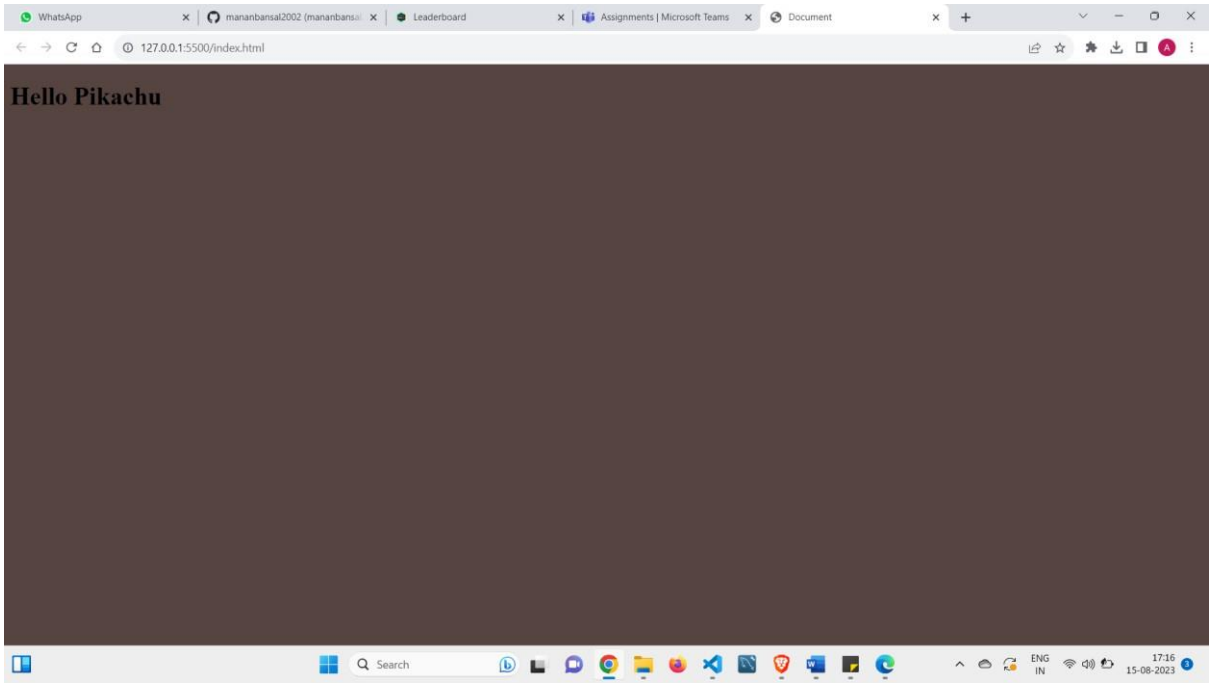
Roll no : 02 , Batch : T11 , Subject : IP , Date : 16/08/23

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

Source Code & Output :

index.html





Name : Altaf Alam

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

Conclusion :

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

Name : Altaf Alam

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

Assignment No 4b

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

Lab Outcome :

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

Theory :

1. Inheritance:

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

- Superclass (Parent Class): The superclass is the original class that defines the properties and methods that can be inherited.

- Subclass (Child Class): The subclass is a new class that inherits properties and methods from a superclass. It can also add its own properties and methods or override the inherited ones.

- `extends` Keyword: In modern JavaScript, you can use the `extends` keyword to create a subclass that inherits from a superclass.

1. Single Inheritance:

- In Single inheritance, classes are organized in a hierarchical structure.

- Each class can inherit properties and methods from a single parent class.

2. Multiple Inheritance:

- In multiple inheritance, a class can inherit properties and methods from multiple parent classes.

- While powerful, it can lead to complex inheritance hierarchies and the "diamond problem," which makes it challenging to resolve conflicts when two parent classes define the same method or property.

3. Multilevel Inheritance:

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

4. Hierarchical Inheritance:

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

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

2. Iterators:

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

- Iterable: An iterable is an object that has an associated iterator. Arrays, strings, maps, sets, and custom iterable objects are examples of iterables.

- Iterator: An iterator is an object that provides a `next()` method. When called, the `next()` method returns the next item in the collection along with a `done` flag to indicate if the iteration is complete.

- `Symbol.iterator`: In JavaScript, objects that are iterable must implement the `Symbol.iterator` method, which returns an iterator.

3. Generators:

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

Name : Altaf Alam

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

- `function*` Syntax: You define a generator function using the `function*` syntax. Inside a generator function, you can use the `yield` keyword to pause the function's execution and yield a value.

- `yield`: The `yield` keyword is used within a generator function to produce a value and temporarily pause the function. When the generator is resumed, it continues execution from where it left off.

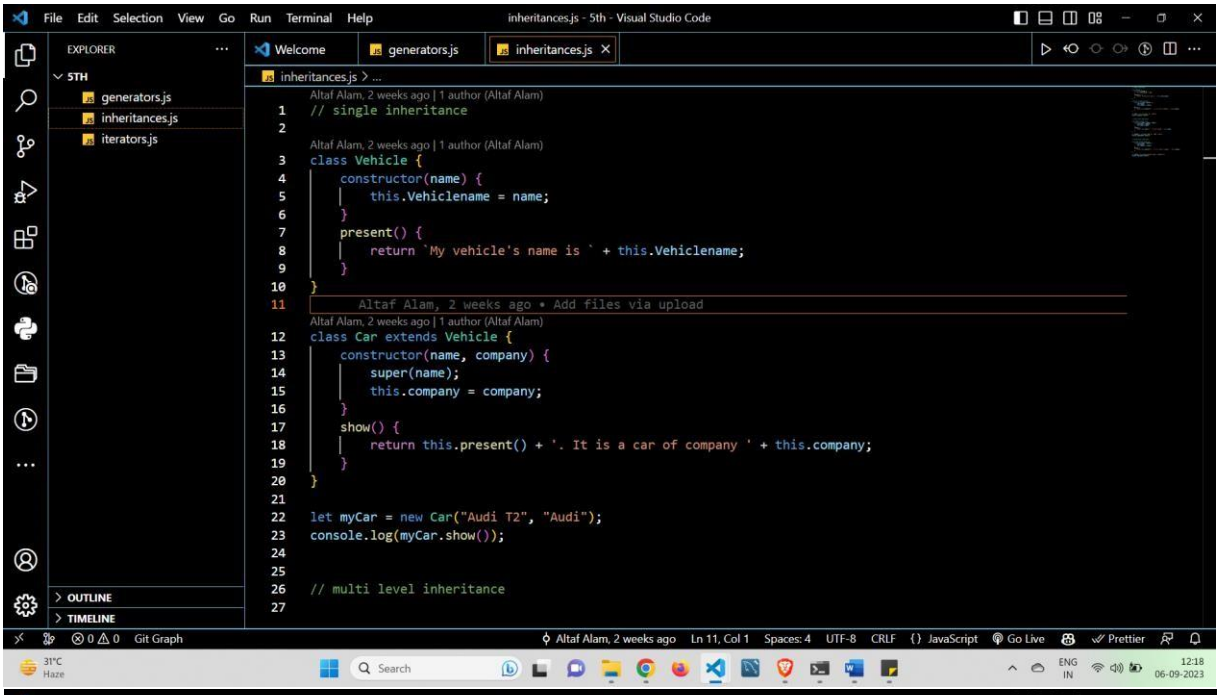
- Lazy Evaluation: Generators enable lazy evaluation, meaning that they produce values ondemand rather than all at once. This is useful for efficiently handling large datasets or asynchronous operations.

- Iterating with Generators: Generators are often used to create custom iterators, providing a more concise and readable way to define iterators for custom data structures.

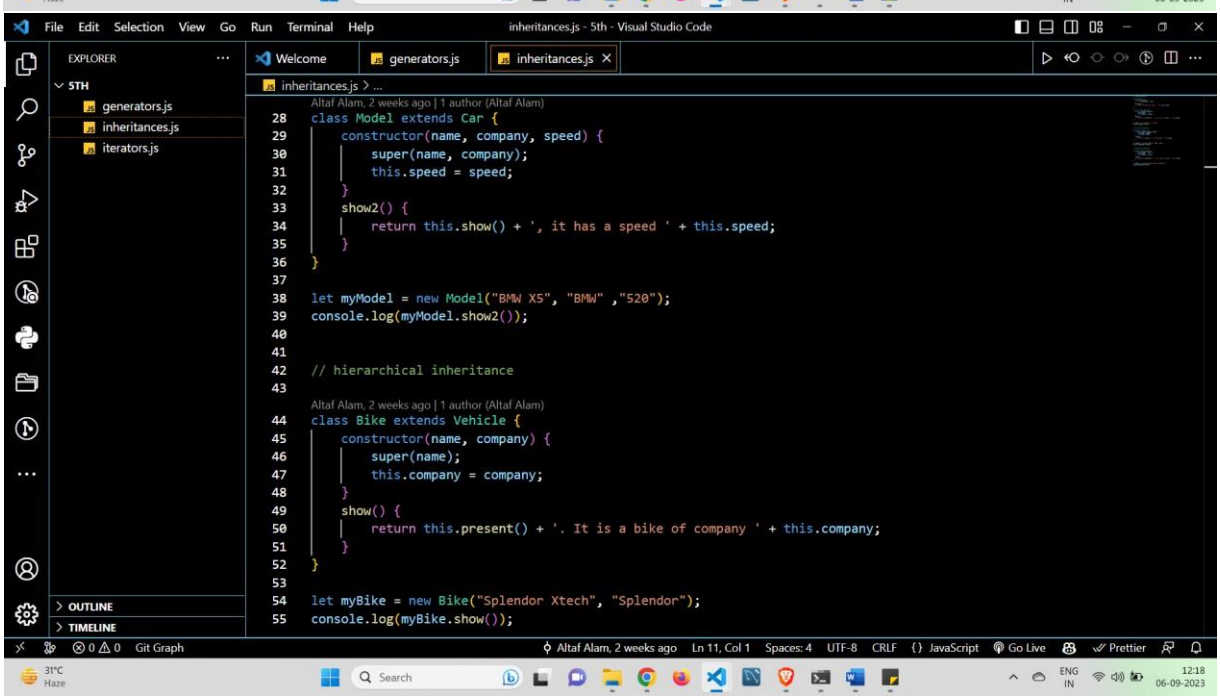
Source Code :

Name : Altaf Alam

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



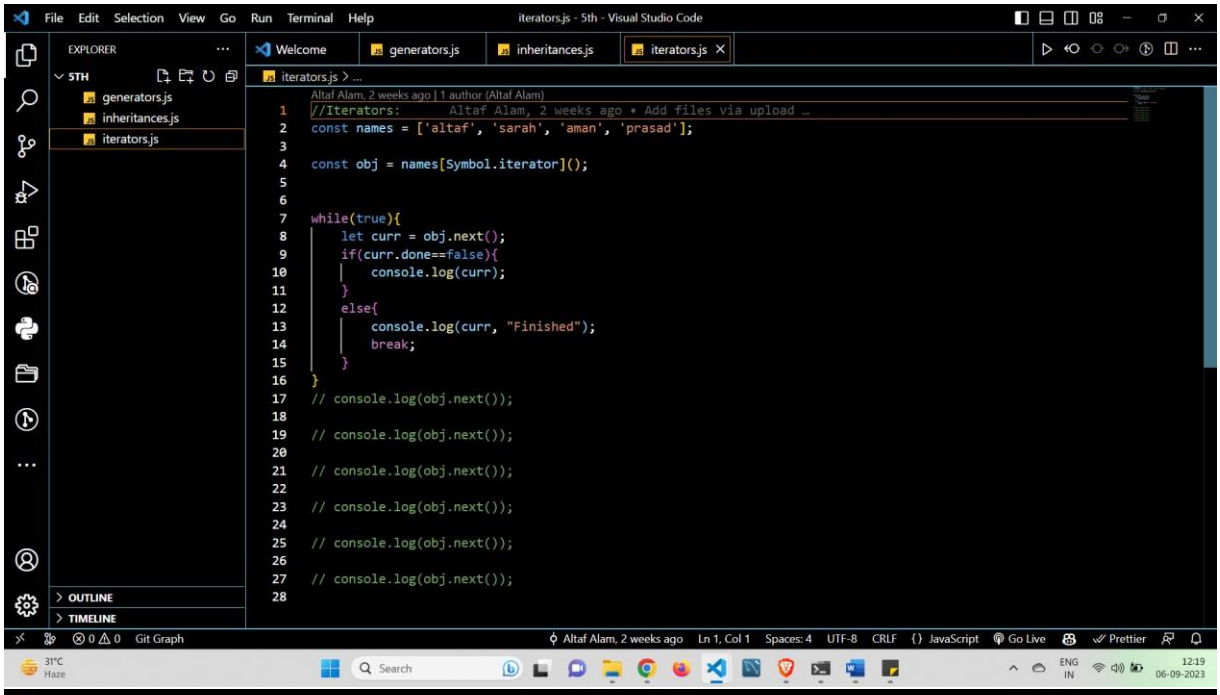
```
Altaf Alam, 2 weeks ago | 1 author (Altaf Alam)
// single inheritance

Altaf Alam, 2 weeks ago | 1 author (Altaf Alam)
class Vehicle {
    constructor(name) {
        this.Vehiclename = name;
    }
    present() {
        return `My vehicle's name is ` + this.Vehiclename;
    }
}
    Altaf Alam, 2 weeks ago • Add files via upload
Altaf Alam, 2 weeks ago | 1 author (Altaf Alam)
class Car extends Vehicle {
    constructor(name, company) {
        super(name);
        this.company = company;
    }
    show() {
        return this.present() + '. It is a car of company ' + this.company;
    }
}

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


// multi level inheritance

```

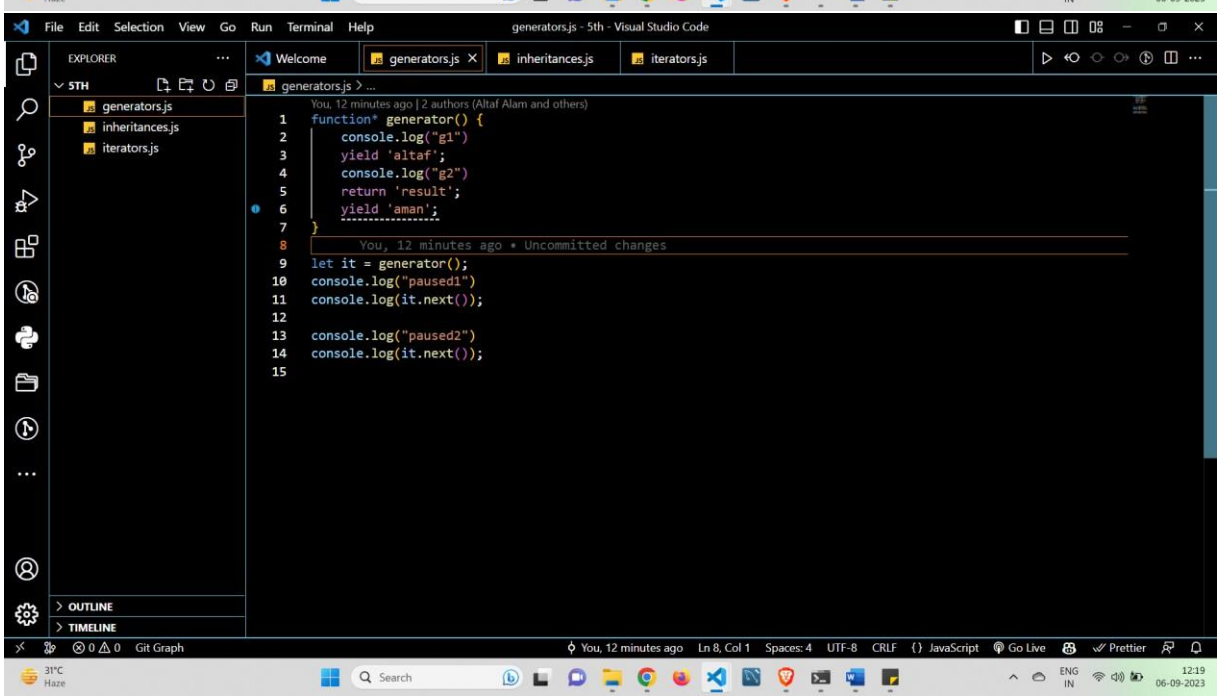


```
Altaf Alam, 2 weeks ago | 1 author (Altaf Alam)
class Model extends Car {
    constructor(name, company, speed) {
        super(name, company);
        this.speed = speed;
    }
    show2() {
        return this.show() + ', it has a speed ' + this.speed;
    }
}

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


// hierarchical inheritance

Altaf Alam, 2 weeks ago | 1 author (Altaf Alam)
class Bike extends Vehicle {
    constructor(name, company) {
        super(name);
        this.company = company;
    }
    show() {
        return this.present() + '. It is a bike of company ' + this.company;
    }
}

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

Name : Altaf Alam

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

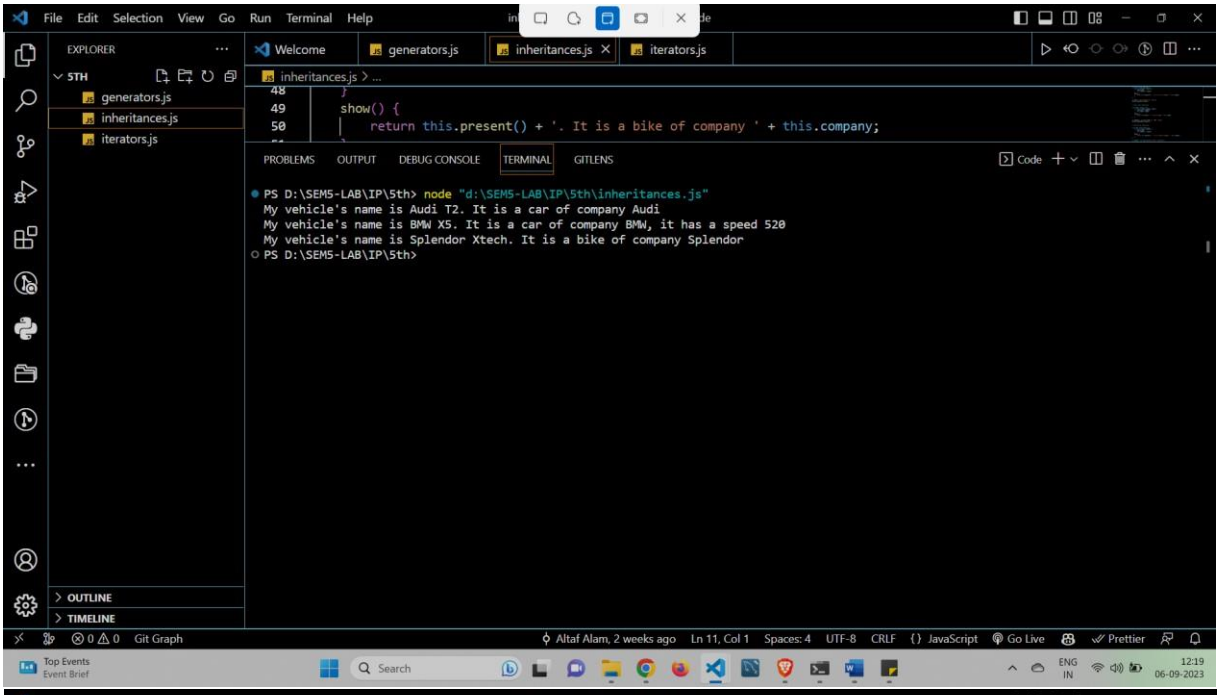


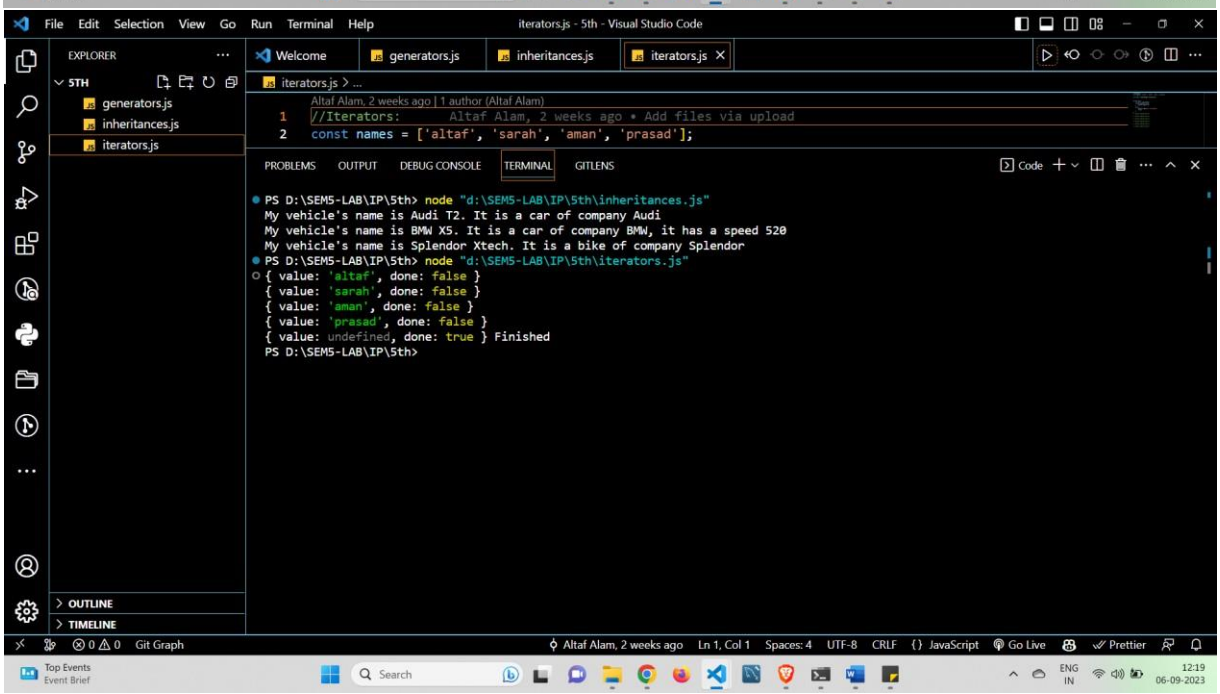


Output :

Name : Altaf Alam

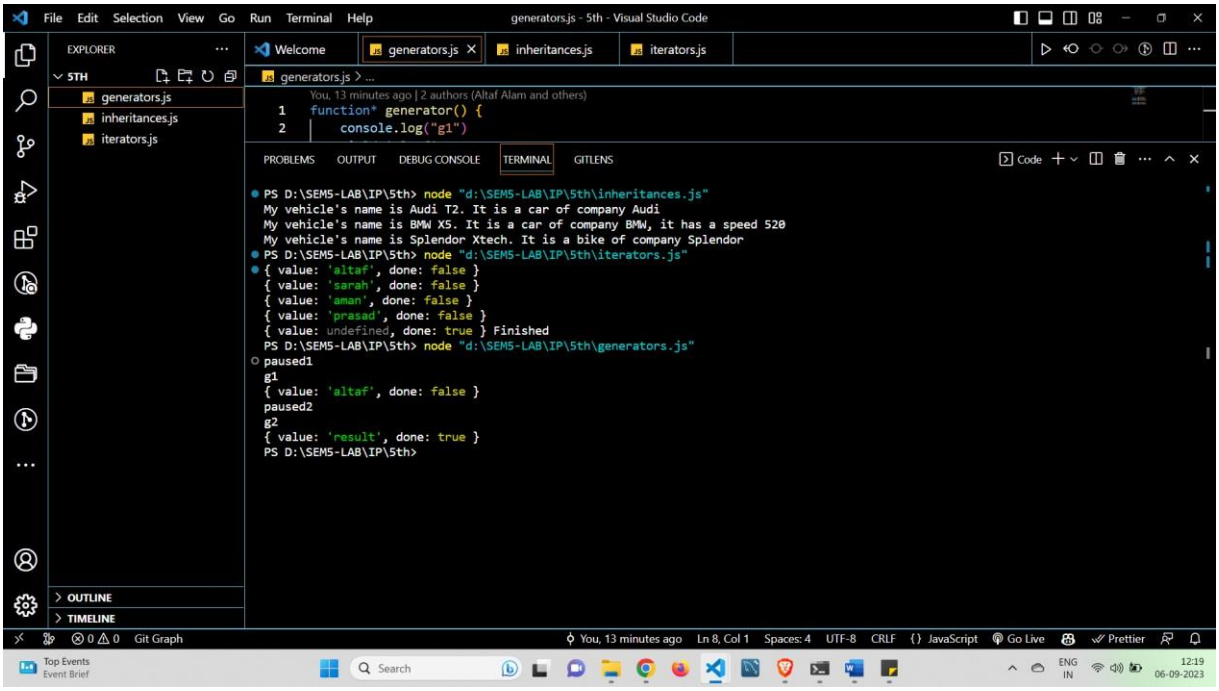
Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23





Name : Altaf Alam

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



Conclusion :

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

Name : Altaf Alam

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

Assignment No 5a

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

Lab Outcome :

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

Theory :

1. Arrow Functions:

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

- Syntax:

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

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

- No `this` Binding:

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

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

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

Name : Altaf Alam

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

2. DOM Manipulation (Document Object Model):

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

- Accessing Elements:

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

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

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

- Modifying Content:

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

- Adding and Removing Elements:

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

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

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

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

- Event Handling:

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

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

3. CSS Manipulation:

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

- Changing Styles:

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

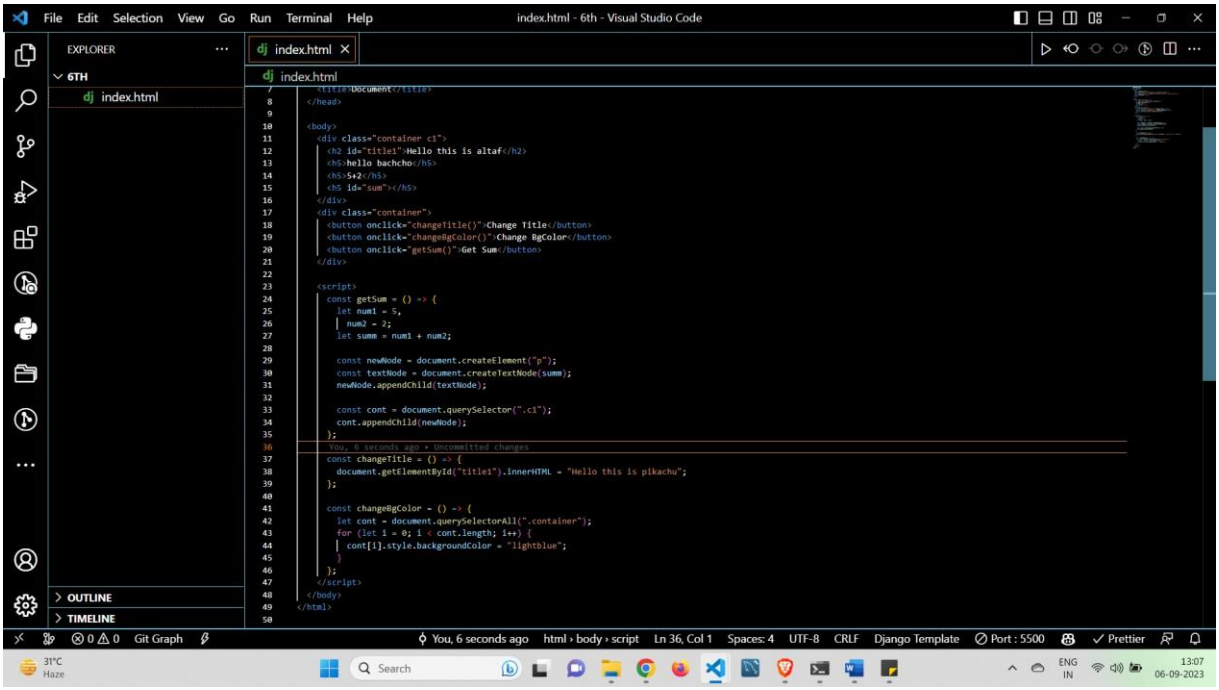
```
// Modify font size
element.style.fontSize = '16px';
```

- Adding and Removing Classes:

```
element.classList.add('active');

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

Name : Altaf Alam

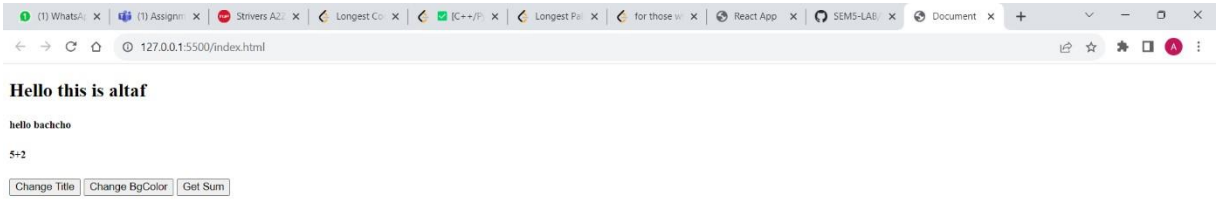
Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

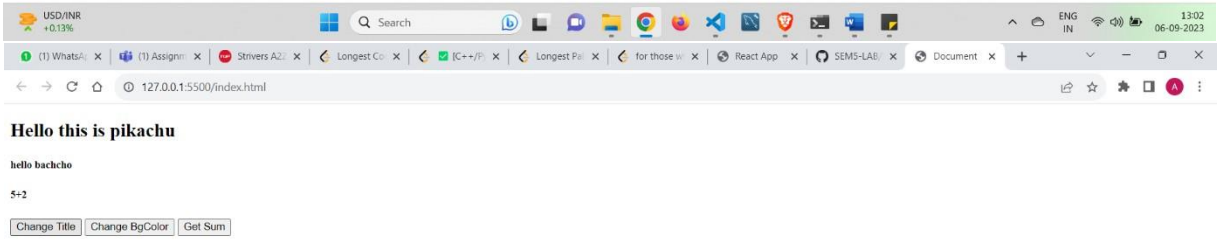
```

## Code :


```

Name : Altaf Alam

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

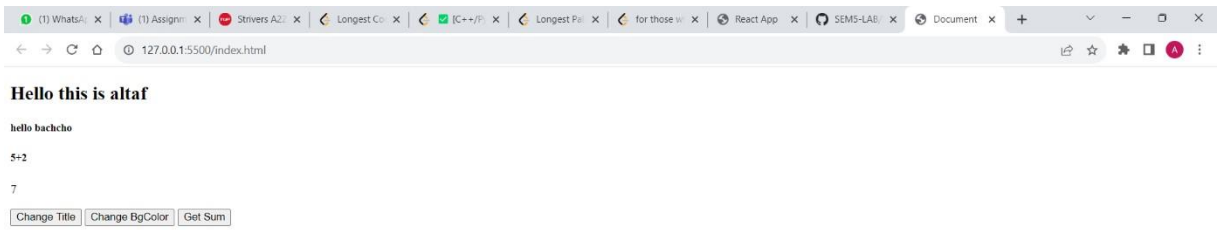
Output :

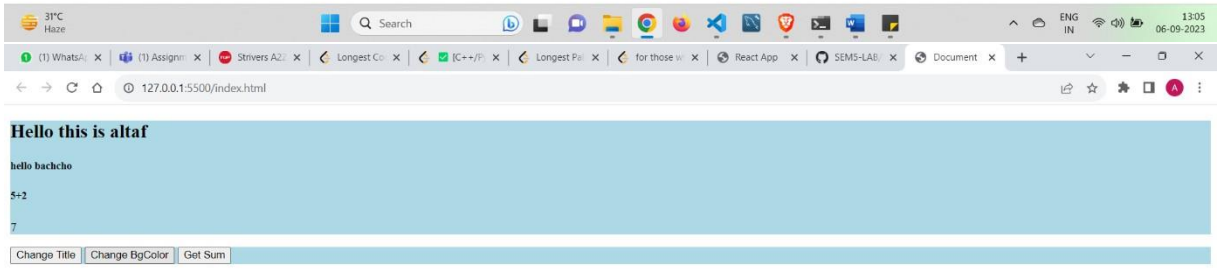


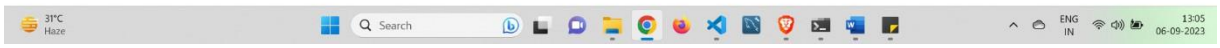


Name : Altaf Alam

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







Conclusion :

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

Name : Altaf Alam

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

Assignment No 5b

Aim : Write a Javascript program to implement the concept of Promise.
Access the data from any API using Promise and either resolve or reject the request by taking appropriate action.
.

Lab Outcome :

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

Theory :

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

1. Promise States:

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

2. Asynchronous Operations:

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

3. Chaining Promises:

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

4. Error Handling:

Name : Altaf Alam

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

- Promises provide a central mechanism for error handling through the `.catch()` method. Errors that occur during promise execution can be caught and handled in a single place.

- This helps improve code maintainability by separating error handling from the logic of asynchronous operations.

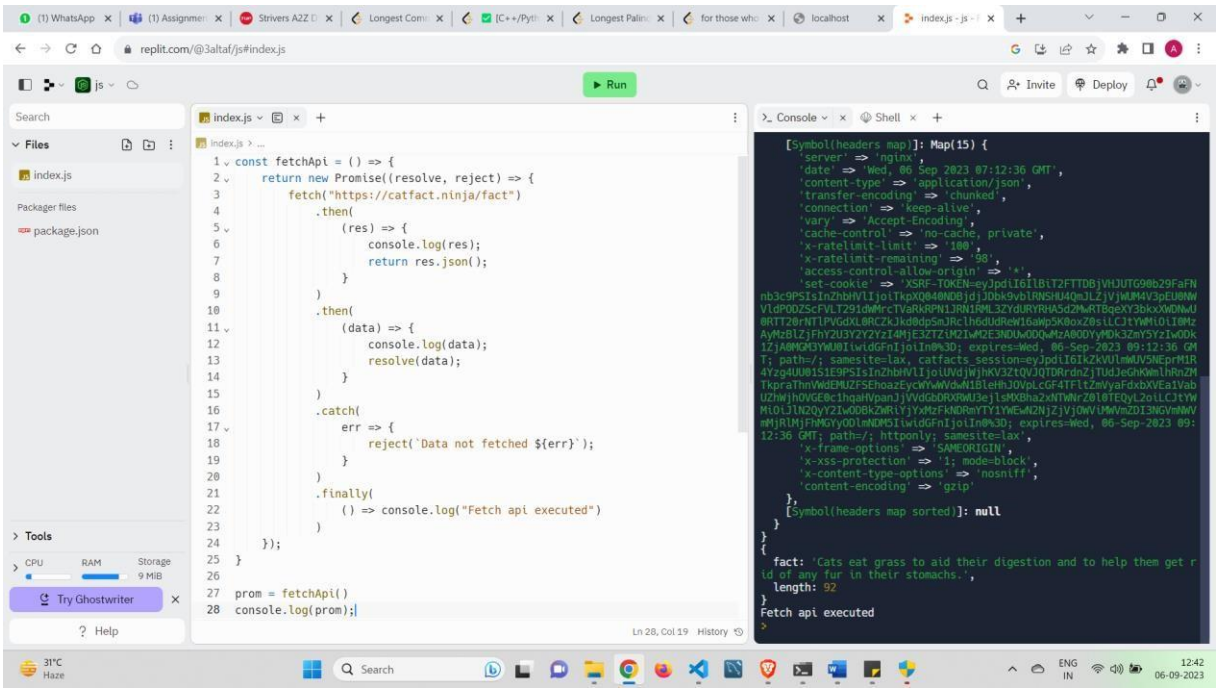
5. Promise.all and Promise.race:

- `Promise.all()` takes an array of promises and returns a new promise that fulfills when all the input promises have fulfilled or rejects when any one of them rejects.

- `Promise.race()` returns a promise that fulfills or rejects as soon as one of the input promises fulfills or rejects. It's useful for tasks where you're interested in the first result.

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

Output :



Name : Altaf Alam

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

Conclusion :

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

Name : Altaf Alam

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

Assignment No 6a

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

Lab Outcome :

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

Theory :

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

Props:

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

1. Immutable: Props are immutable, meaning they cannot be modified within the child component. They are passed down from parent components and are read-only.

2. Functional Components: Props are commonly used in functional components to receive data and render it. You define props as parameters in the component's function signature.

3. Data Flow: Props facilitate the flow of data from a parent component to a child component. This allows you to create reusable and composable components.

4. Usage: To access props in a functional component, you simply refer to them as properties of the `props` object (e.g., `props.name`).

5. Example:

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

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

```

**State:**

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

1. Class Components: In class components, you manage state using the `this.state` object and the `this.setState()` method. State can be modified using `setState()` to trigger re-renders.

2. Functional Components with Hooks: With the introduction of hooks in React, functional components can also manage state using the `useState` hook. It returns an array with the current state value and a function to update it.

3. Local to a Component: State is local to a component and cannot be directly accessed or modified by other components. It encapsulates a component's internal data.

4. Reactive UI: When state changes, React re-renders the component, ensuring that the UI reflects the latest data.

5. Example (Functional Component with `useState`):
```jsx
import React, { useState } from 'react';

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

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

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

Name : Altaf Alam

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

```

useRef:

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

1. Mutable Refs: `useRef` creates a mutable object with a `.current` property. This property can be assigned a value, and it retains its value between renders without causing re-renders.

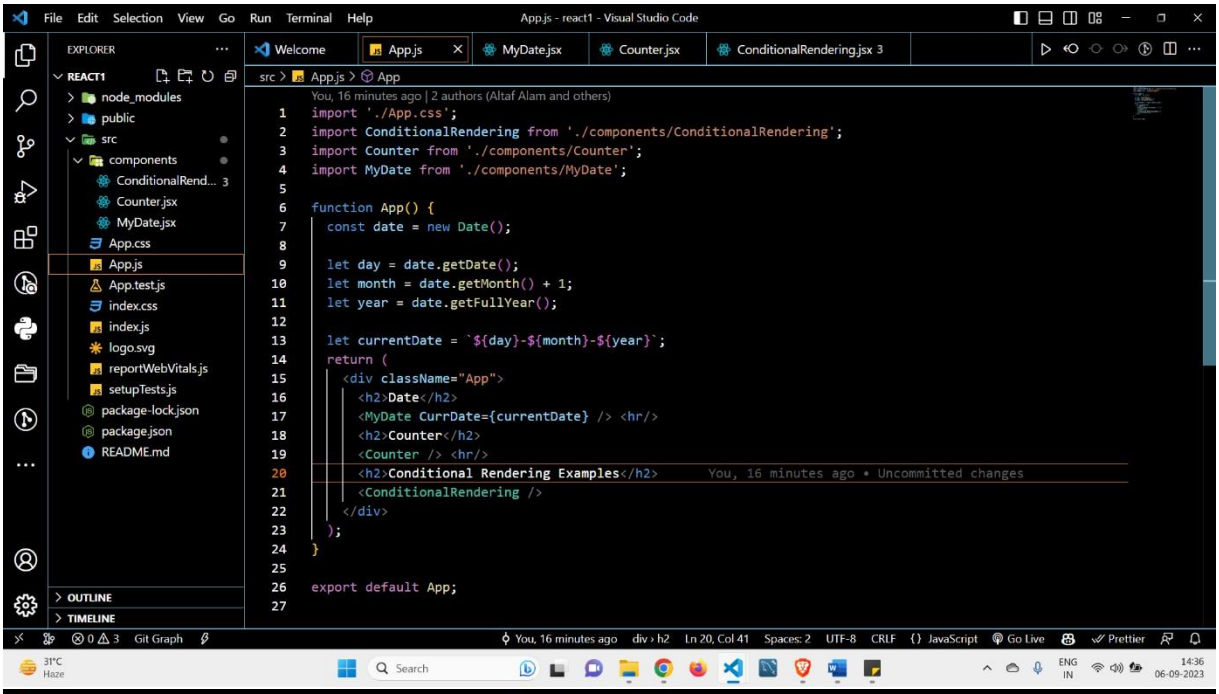
2. Accessing DOM Elements: One common use of `useRef` is to access and manipulate DOM elements directly. You can attach a `ref` attribute to a JSX element and assign it to a `useRef` variable.

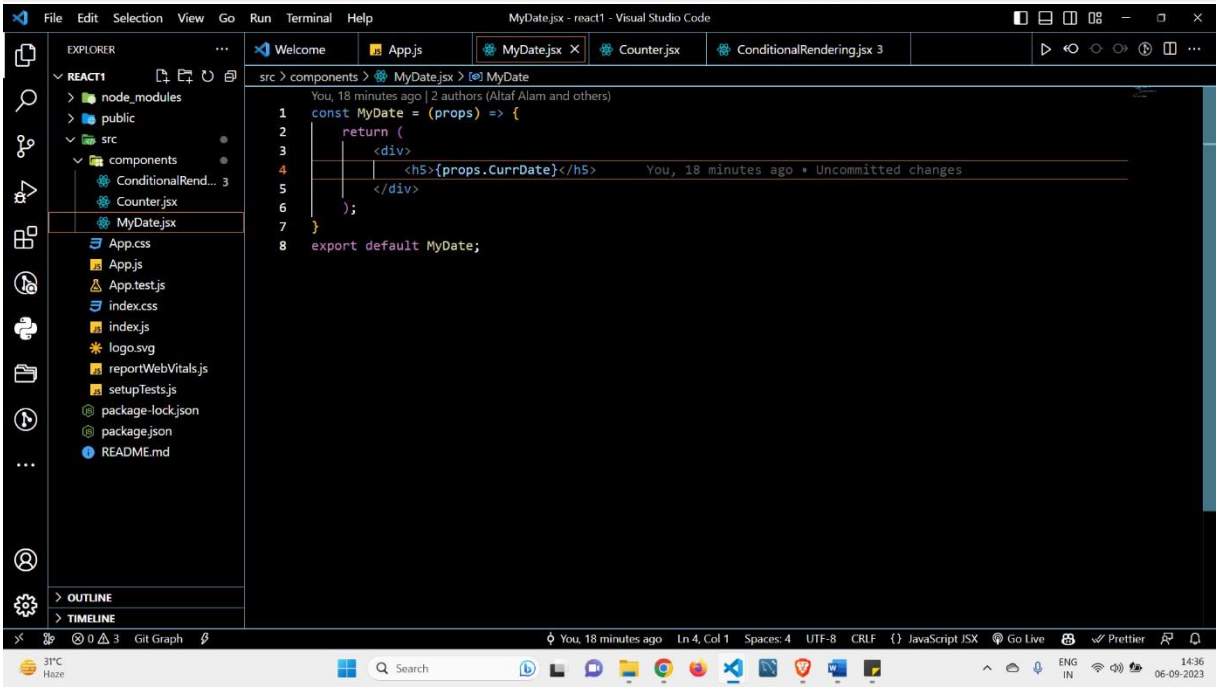
3. Preserving Values: `useRef` is suitable for preserving values or state between renders without causing re-renders. This makes it different from `useState`.
```

Name : Altaf Alam

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

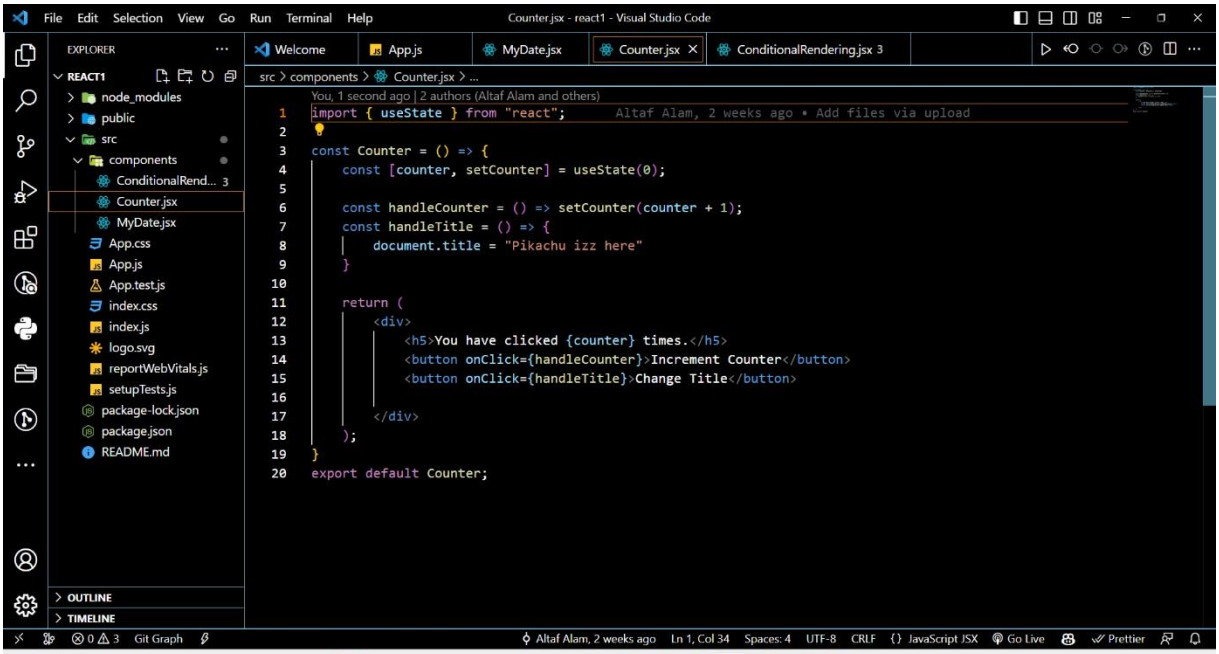
Code :



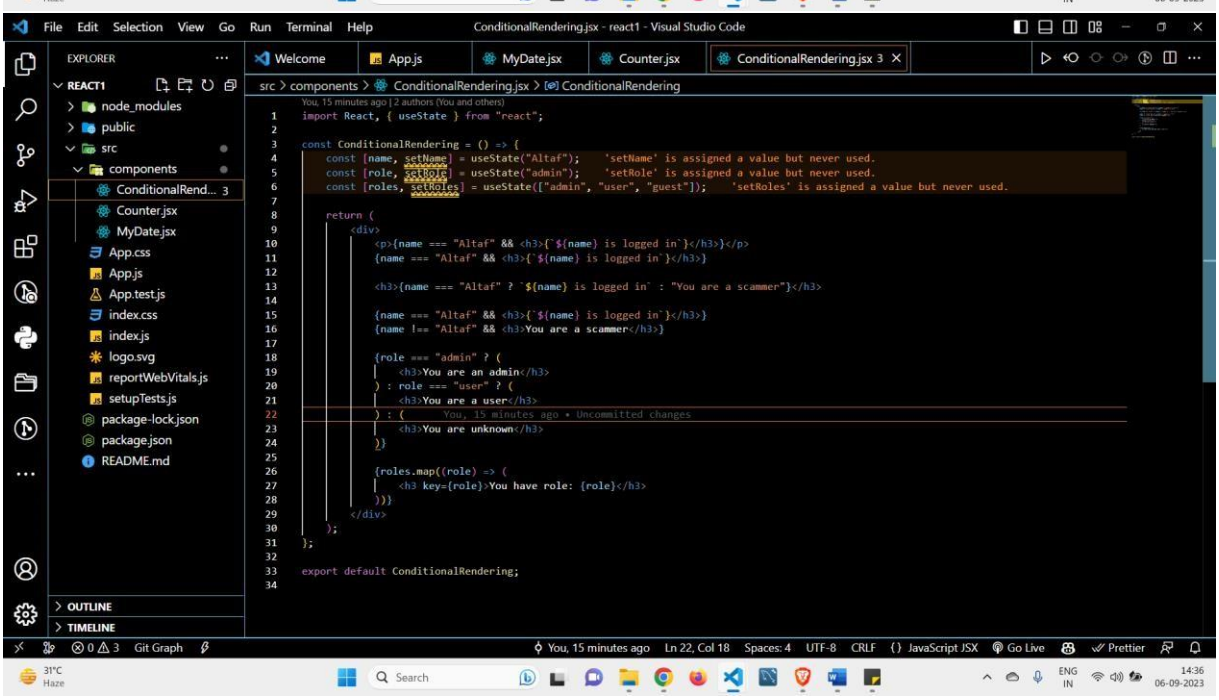


Name : Altaf Alam

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



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

const Counter = () => {
    const [counter, setCounter] = useState(0);

    const handleCounter = () => setCounter(counter + 1);
    const handleTitle = () => {
        document.title = "Pikachu izz here"
    }

    return (
        <div>
            <h5>You have clicked {counter} times.</h5>
            <button onClick={handleCounter}>Increment Counter</button>
            <button onClick={handleTitle}>Change Title</button>

        </div>
    );
}
export default Counter;
```

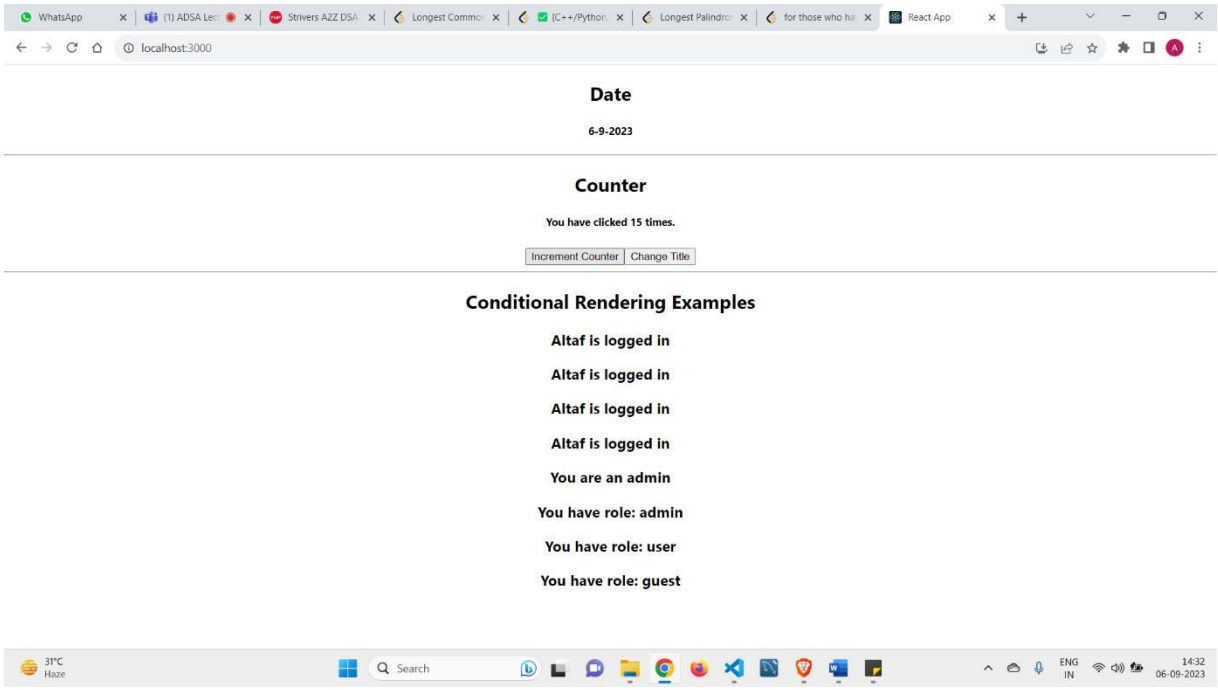


```jsx
import React, { useState } from "react";

const ConditionalRendering = () => {
    const [name, setName] = useState("Altaf");    'setName' is assigned a value but never used.
    const [role, setRole] = useState("admin");    'setRole' is assigned a value but never used.
    const [roles, setRoles] = useState(["admin", "user", "guest"]);    'setRoles' is assigned a value but never used.

    return (
        <div>
            <p>{name === "Altaf" && <h3>{`${name} is logged in`}</h3>}</p>
            {name === "Altaf" && <h3>{`${name} is logged in`}</h3>}

            <h3>{name === "Altaf" ? `${name} is logged in` : "You are a scammer"}</h3>

            {name === "Altaf" && <h3>{`${name} is logged in`}</h3>}
            {name !== "Altaf" && <h3>You are a scammer</h3>}

            {role === "admin" ? (
                <h3>You are an admin</h3>
            ) : role === "user" ? (
                <h3>You are a user</h3>
            ) : (
                <h3>You are unknown</h3>
            )}

            {roles.map((role) => (
                <h3 key={role}>You have role: {role}</h3>
            ))}
        </div>
    );
};

export default ConditionalRendering;
```

Name : Altaf Alam

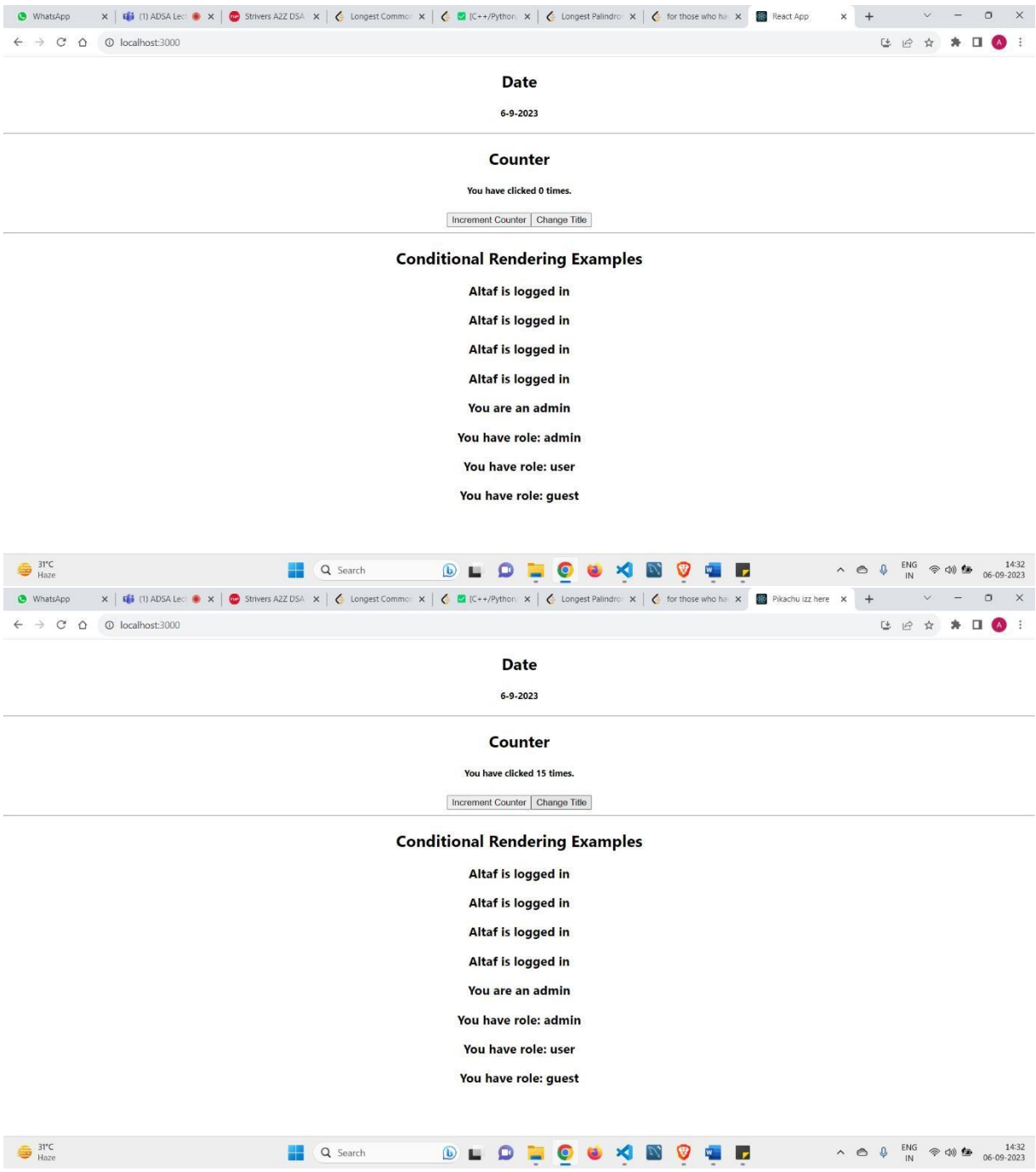
Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

Output :



Name : Altaf Alam

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



Conclusion :

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

Name : Altaf Alam

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

Assignment No 6b

Aim : Write a Program to implement forms and events.

Lab Outcome :

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

Theory :

Forms:

1. What is a Form?

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

2. Common Form Elements:

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

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

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

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

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

3. Form Attributes:

- `action`: Specifies the URL where the form data will be sent upon submission.
- `method`: Defines the HTTP method to be used for the request (e.g., GET or POST).
- `name`: Provides a name for the form, often used for identification in JavaScript.
- `enctype`: Specifies how form data should be encoded for submission (e.g., `application/xwww-form-urlencoded` for simple text data).
- `target`: Specifies where the response should be displayed (e.g., in the same window or a new tab).

Name : Altaf Alam

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

4. Form Submission:

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

Events:

1. What are Events?

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

2. Common Types of Events:

- Mouse Events: Events triggered by mouse actions, such as `click`, `mousedown`, `mouseup`, `mousemove`, and `mouseenter`.

- Keyboard Events: Events related to keyboard input, such as `keydown`, `keyup`, and `keypress`.

- Form Events: Events related to form interactions, like `submit`, `reset`, `change`, and `input`.

- Document and Window Events: Events related to the document and window, including `load`, `unload`, `resize`, and `scroll`.

- Custom Events: Developers can create custom events to handle specific interactions within their applications.

3. Event Handling:

Event handling in JavaScript involves the following steps:

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

Name : Altaf Alam

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

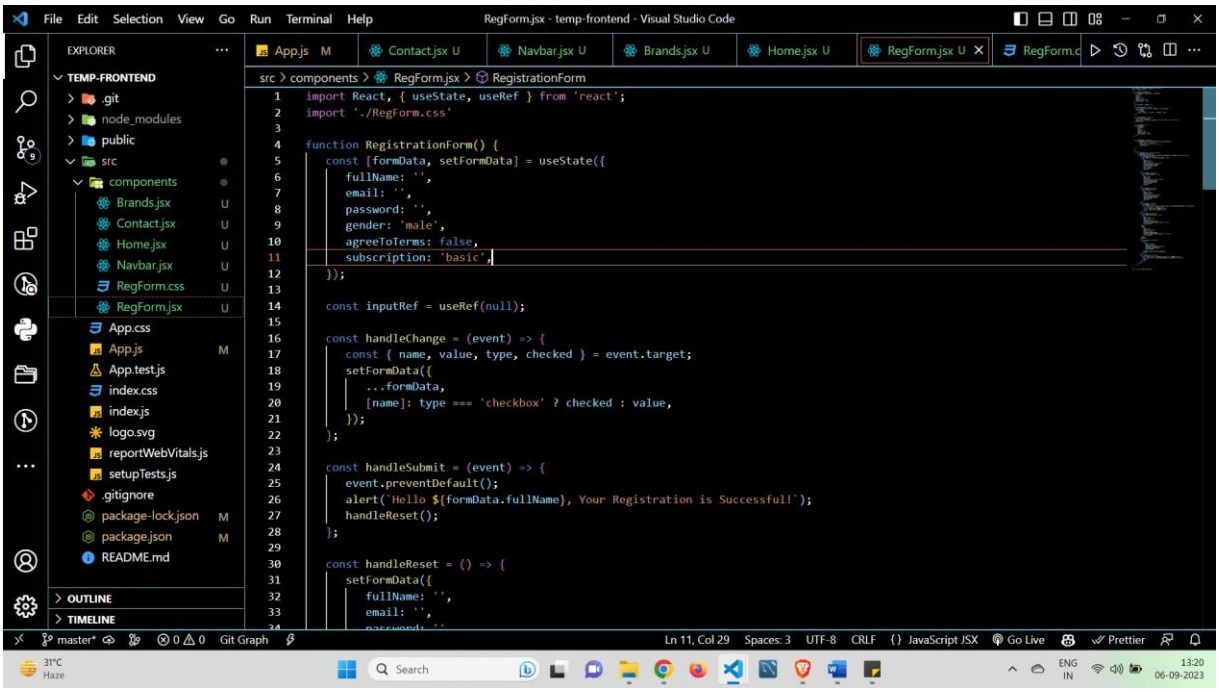
- Event Handler: The event handler is a JavaScript function that defines what should happen when the event occurs. It can perform actions like updating the DOM, making AJAX requests, or triggering animations.

- Event Object: When an event occurs, an event object is automatically created and passed to the event handler. This object contains information about the event, such as the target element and event-specific data.

4. Event Propagation:

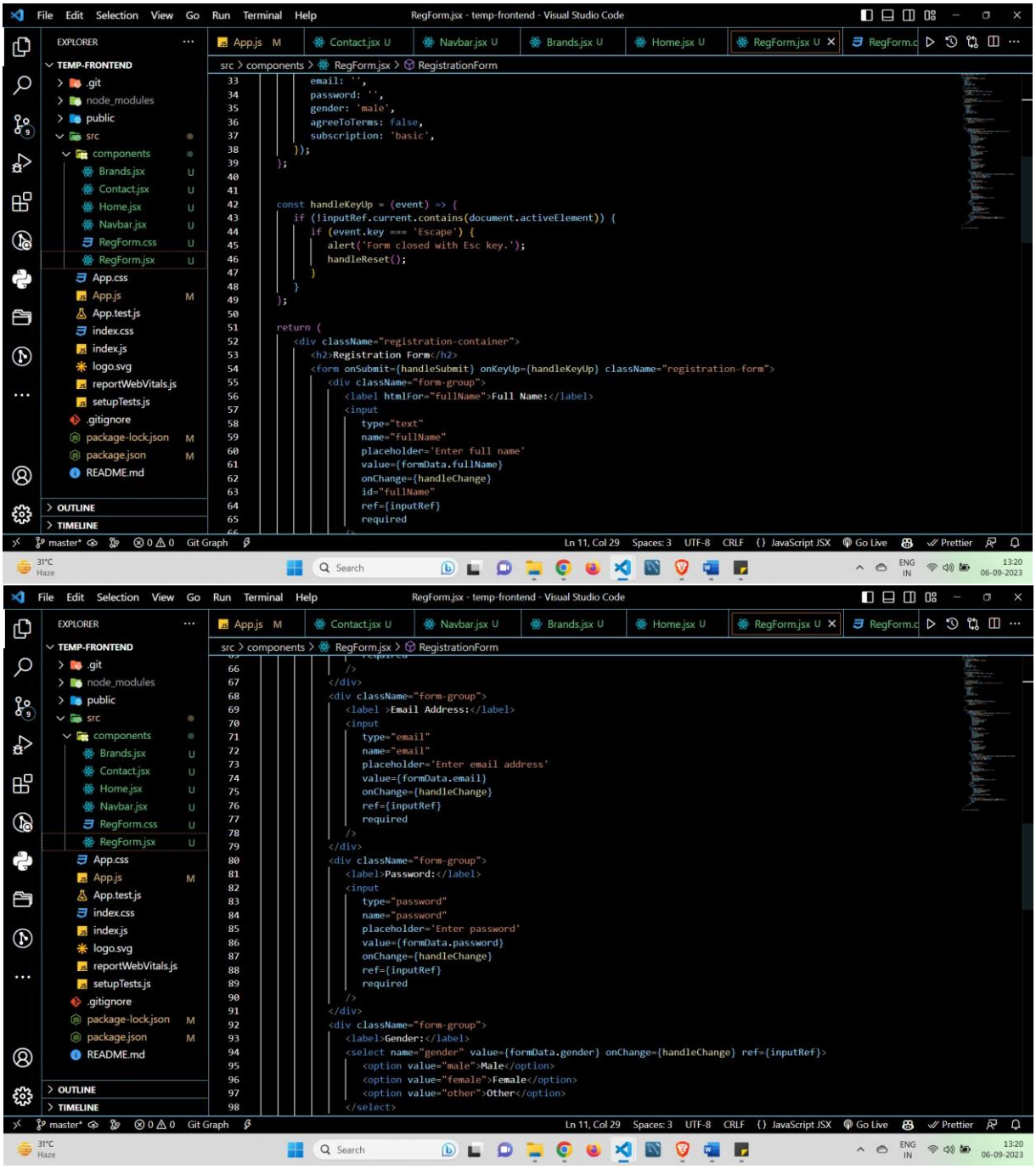
Events can propagate through the DOM tree in two phases: capturing phase and bubbling phase. Event propagation determines the order in which event listeners are invoked. You can control event propagation using the `stopPropagation` method.

Code :



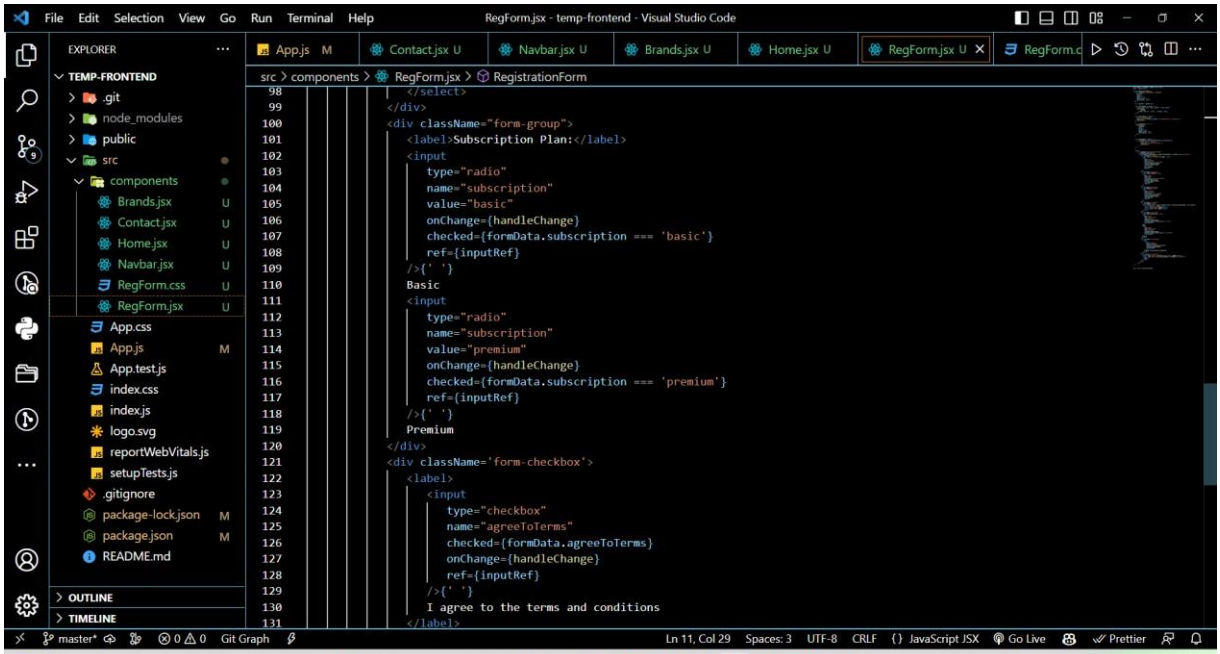
Name : Altaf Alam

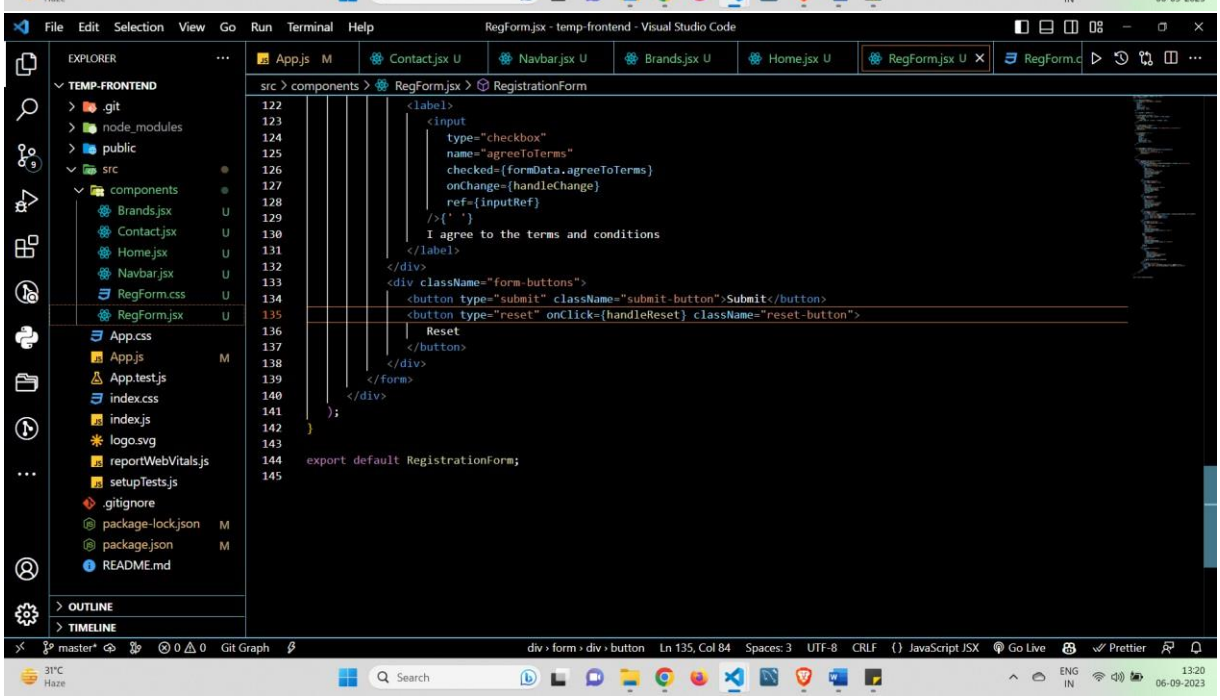
Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23



Name : Altaf Alam

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

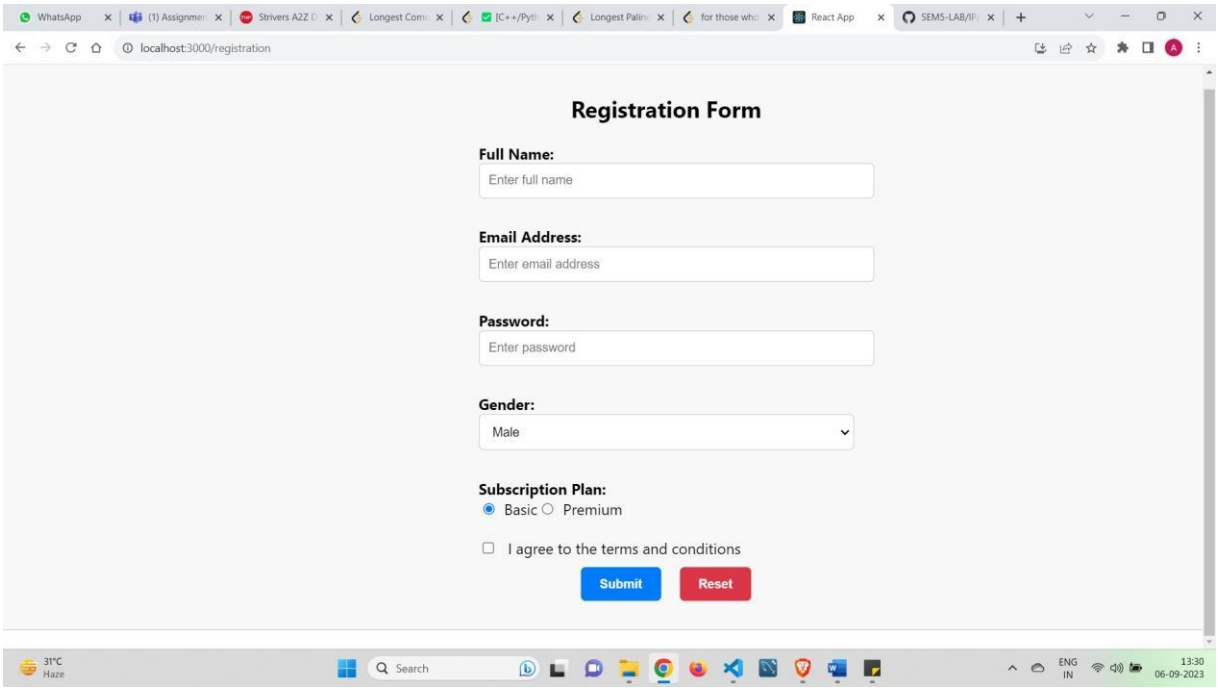


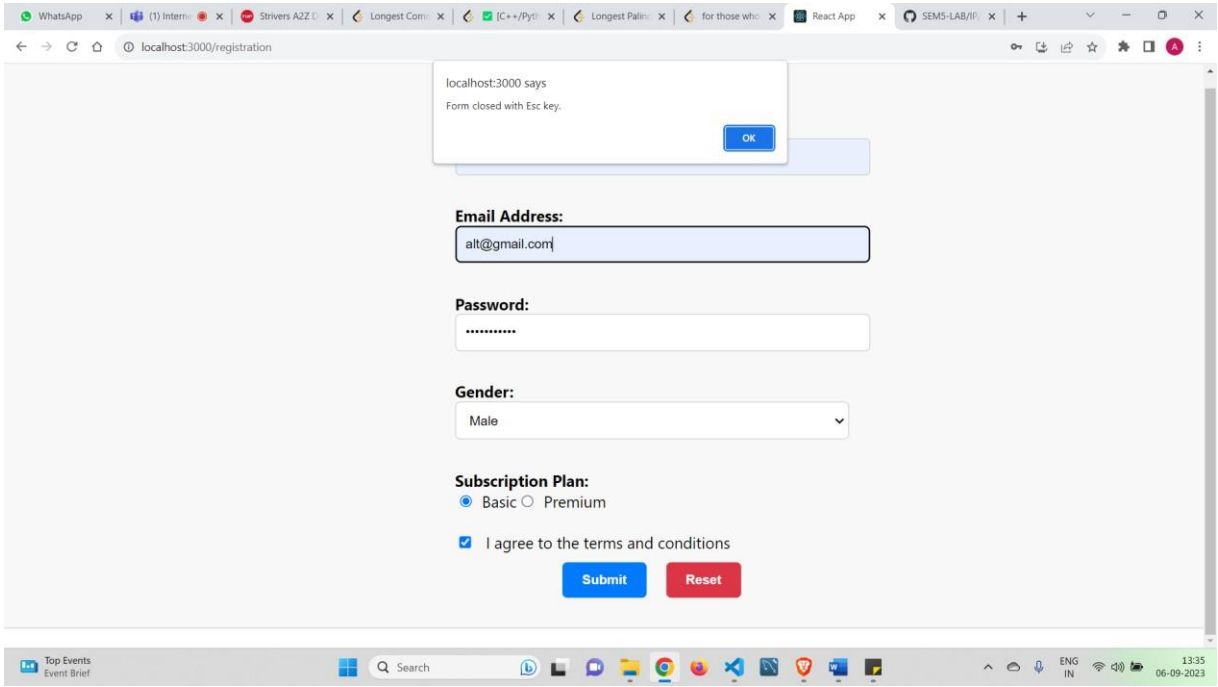


Name : Altaf Alam

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

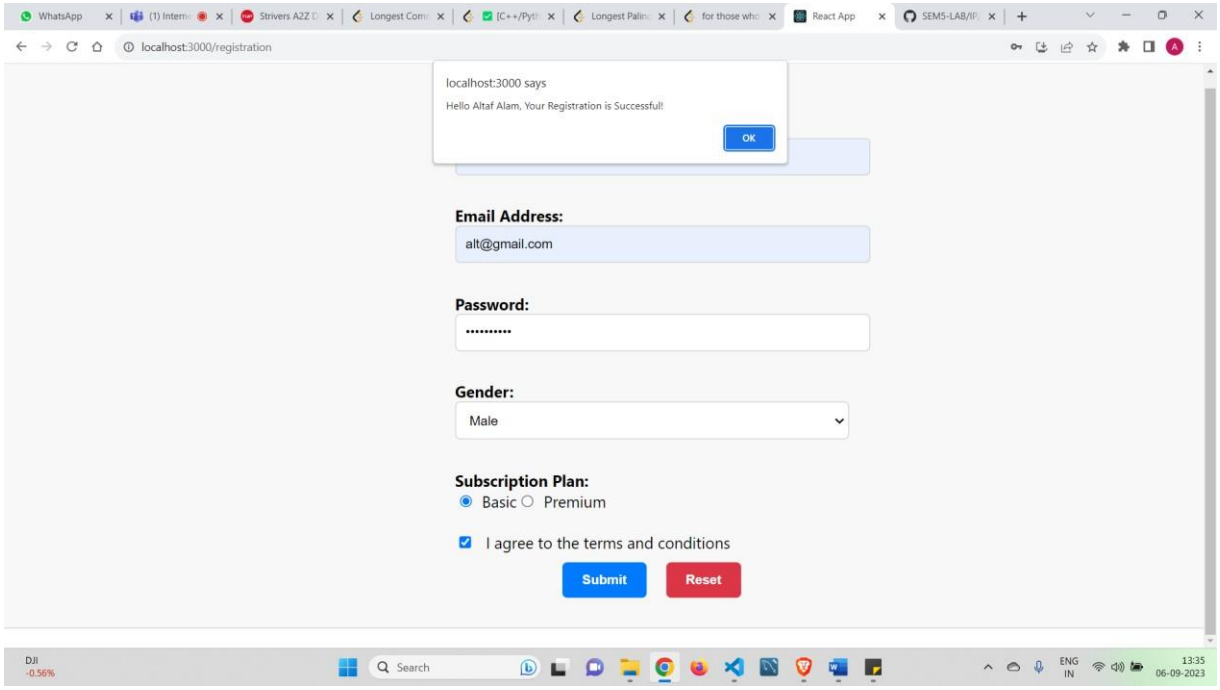
Output :





Name : Altaf Alam

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



Conclusion :

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

Name : Altaf Alam

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

Assignment No 7a

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

Lab Outcome :

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

Theory :

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

1. Installation:

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

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

2. Importing BrowserRouter:

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

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

3. Wrapping the App Component:

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

Name : Altaf Alam

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

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

1. `BrowserRouter` Component: The `<BrowserRouter>` component wraps the entire application. It provides the context for routing and is responsible for synchronizing the application's UI with the URL.

2. `Navbar` Component: This component is rendered outside of the `<Routes>` component, which means it will appear on every page of your application. It likely contains navigation links (e.g., Home, Contact, Brands, Registration), allowing users to navigate to different parts of your app.

3. `Routes` Component: Inside the `<BrowserRouter>`, the `<Routes>` component is used to define the route configuration for your application. It acts as a container for individual `<Route>` components.

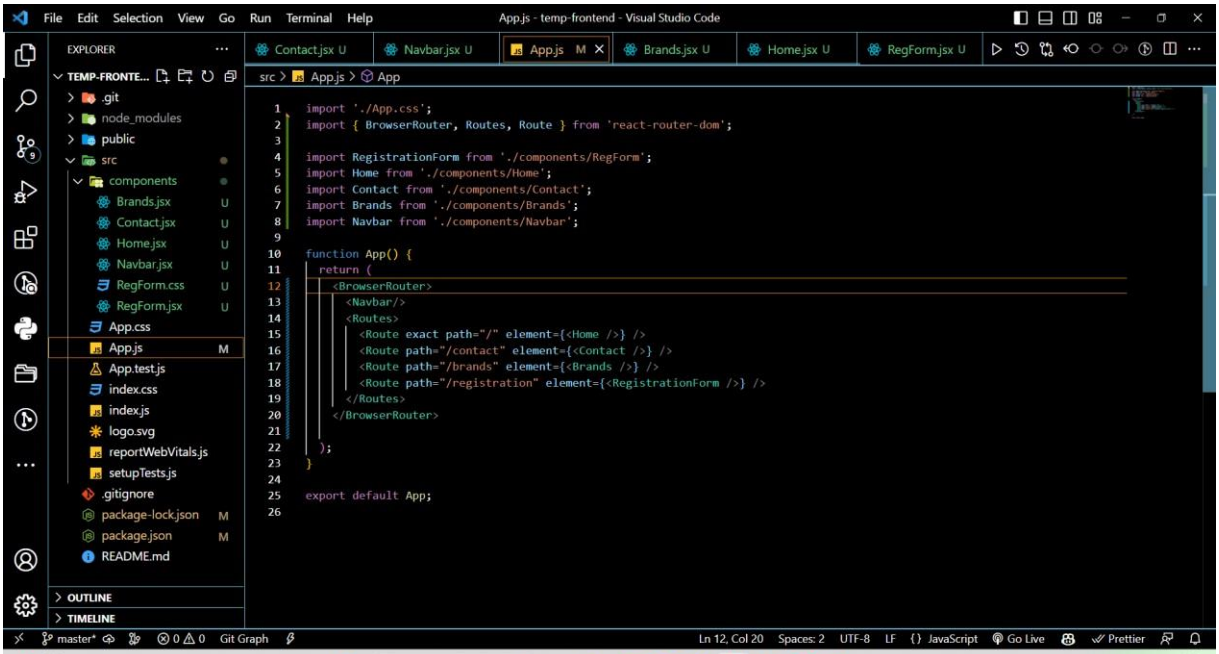
4. `Route` Components: Each `<Route>` component specifies how a particular URL should map to a React component. For example:
- The `<Route>` with `path="/" element={<Home />}` indicates that when the URL is exactly "/", the `Home` component should be rendered.
- The `<Route>` with `path="/contact" element={<Contact />}` maps the "/contact" URL to the `Contact` component, and so on.

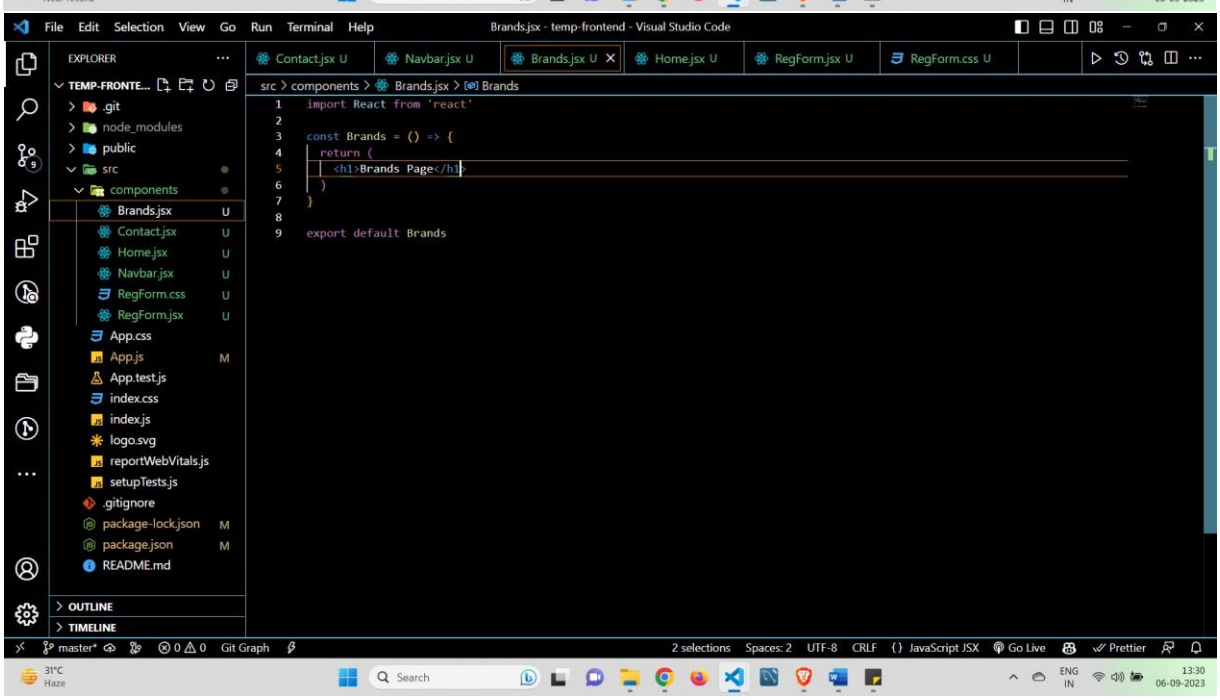
By using `BrowserRouter`, you enable client-side routing. When a user clicks a link or enters a URL, React Router will determine which component to render based on the URL's path. This approach provides a seamless user experience without full page reloads.

Code :

Name : Altaf Alam

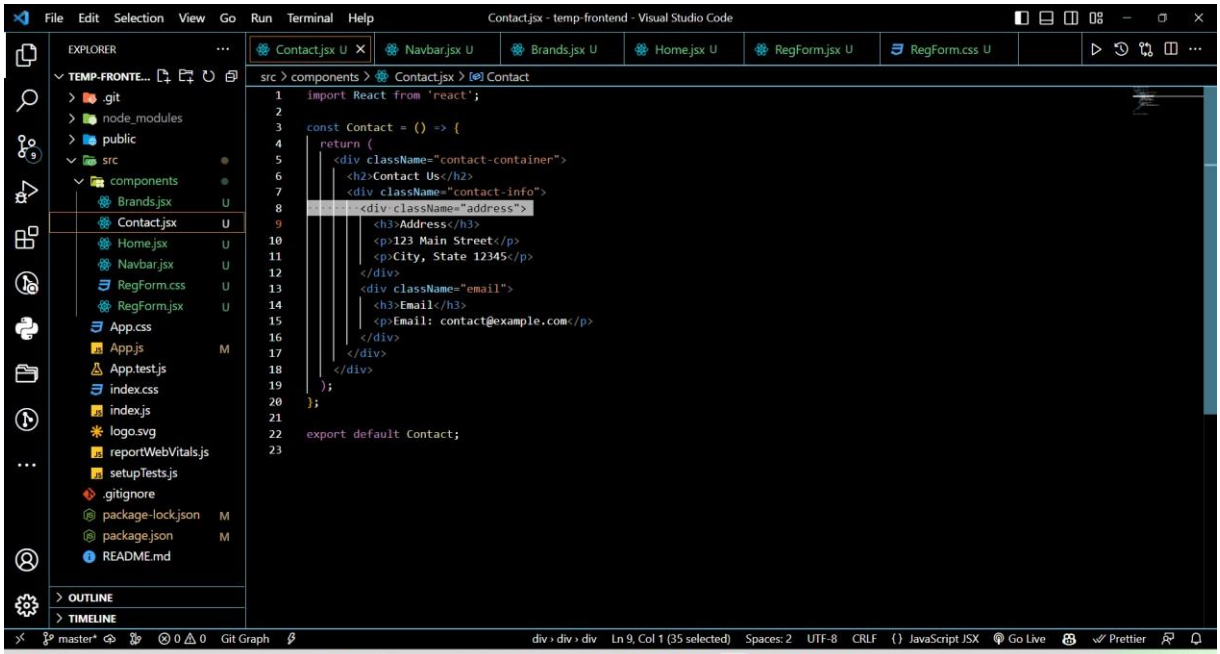
Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

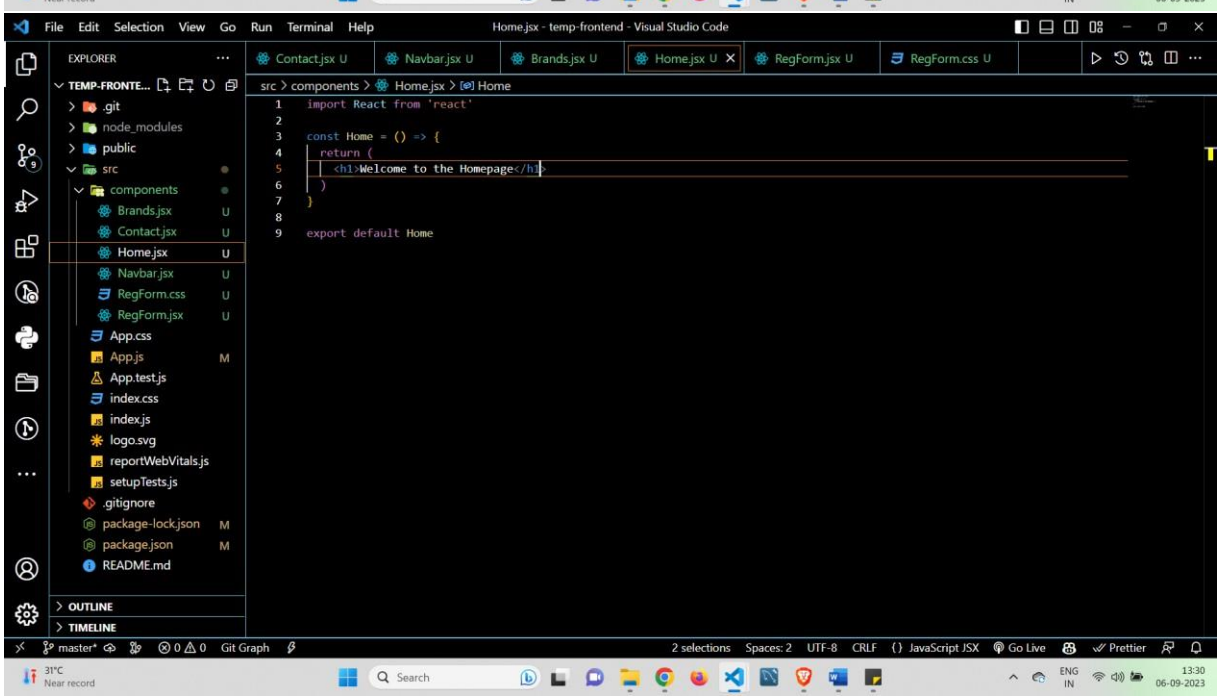




Name : Altaf Alam

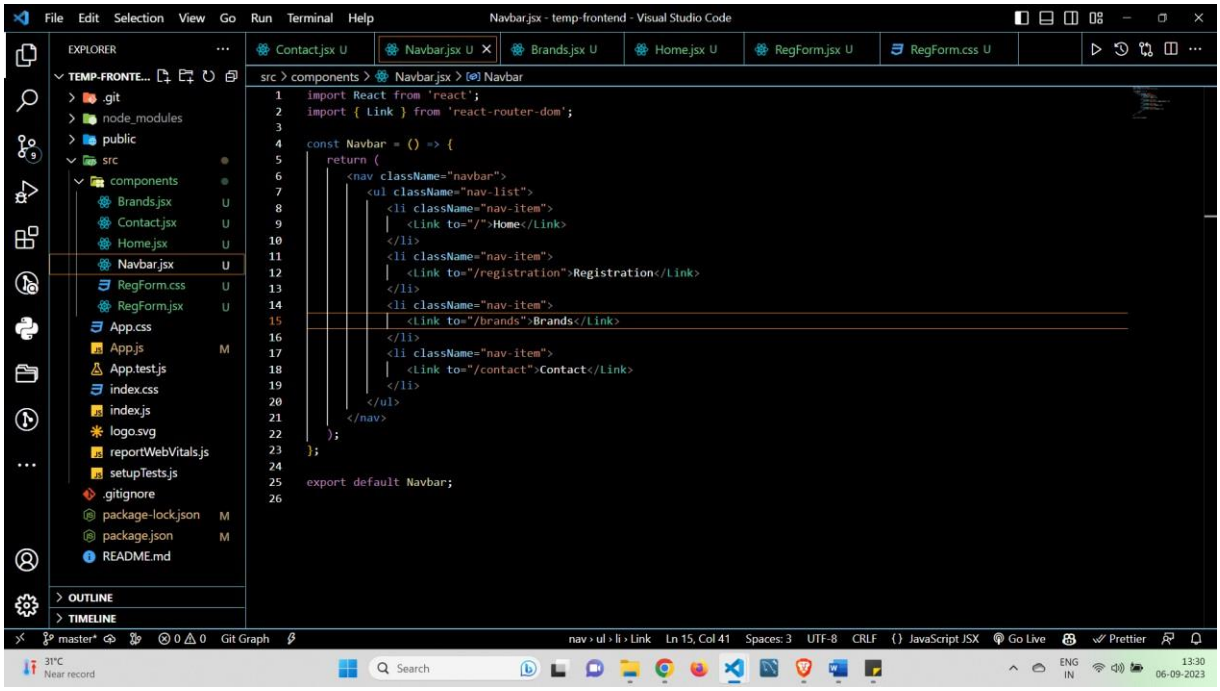
Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23





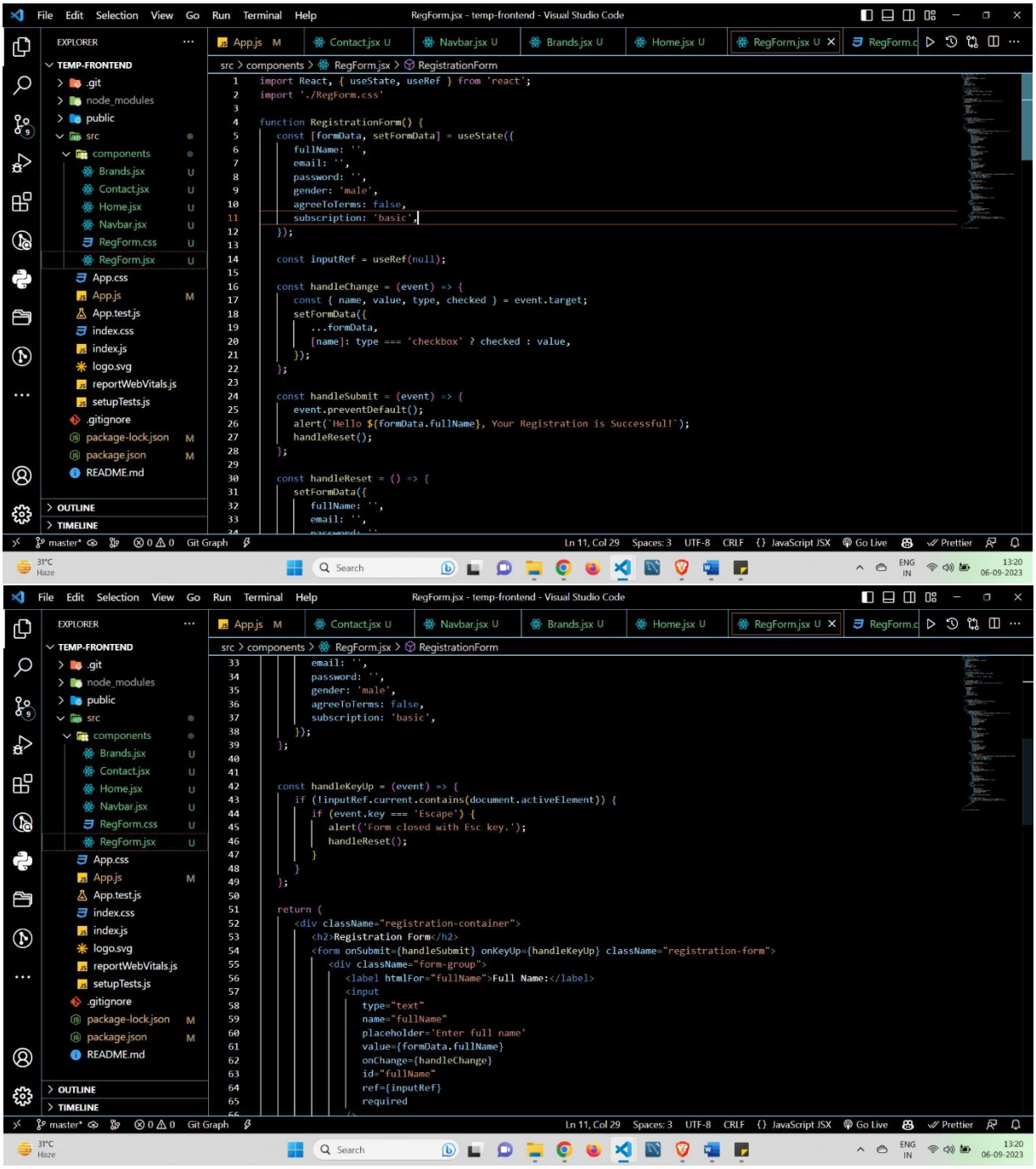
Name : Altaf Alam

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



Name : Altaf Alam

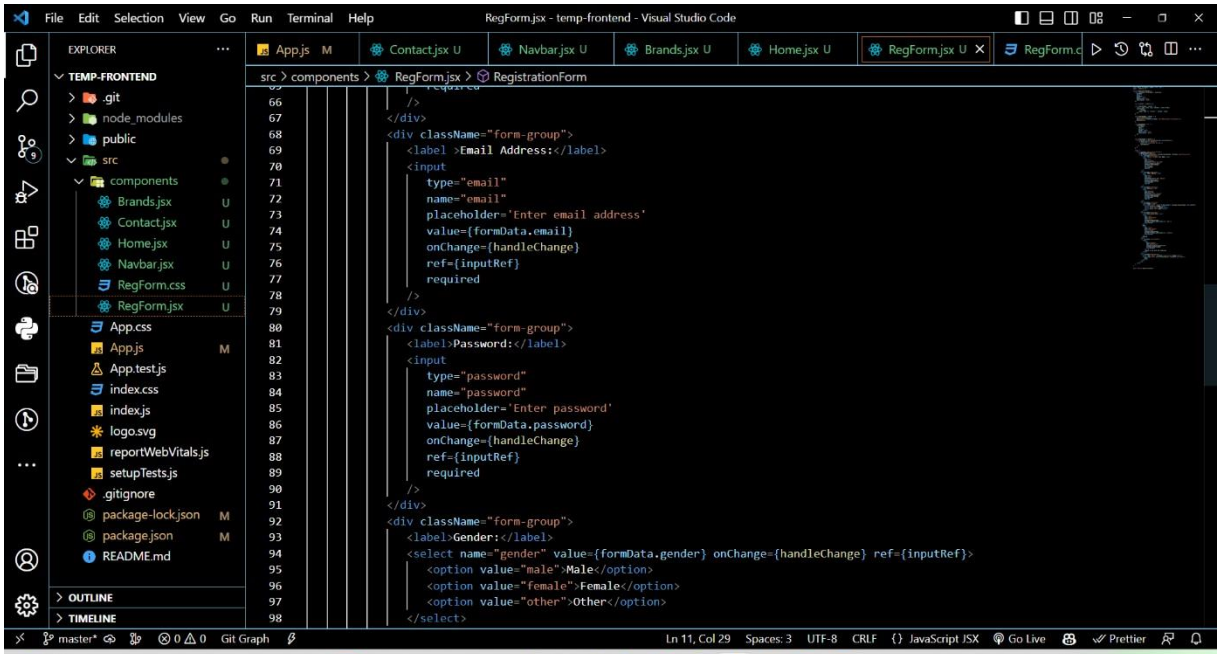
Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23



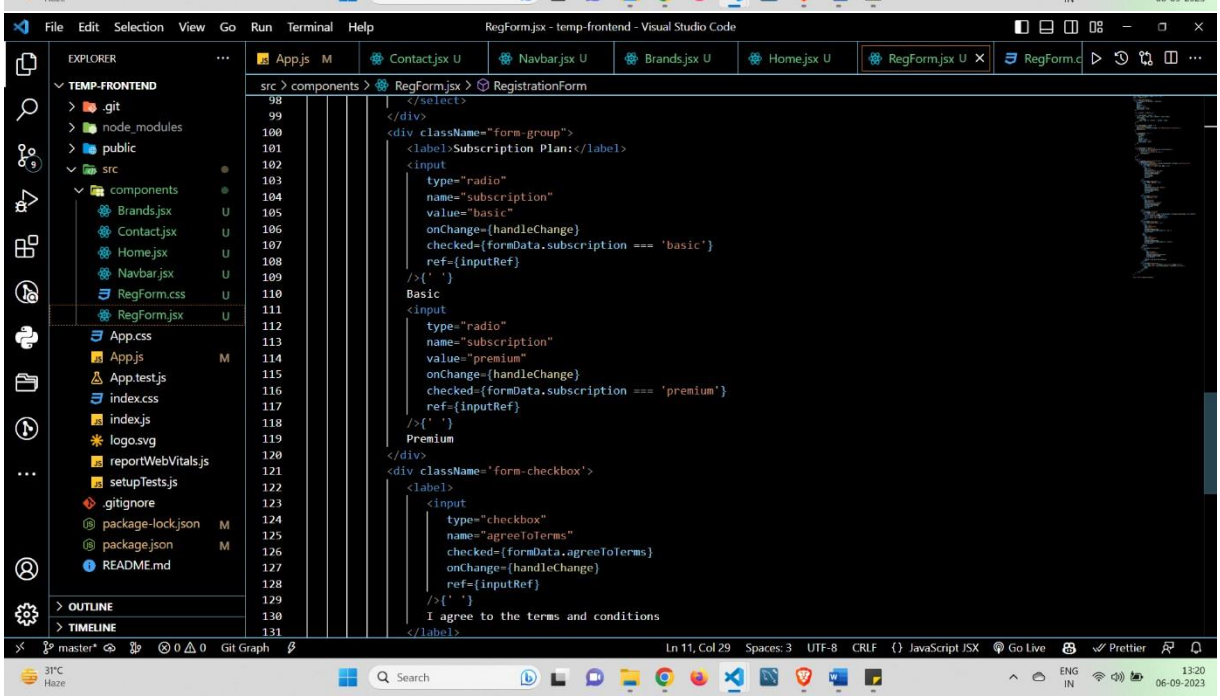
```
import React, { useState, useRef } from 'react';
import './RegForm.css'

function RegistrationForm() {
  const [formData, setFormData] = useState({
    fullName: '',
    email: '',
    password: '',
    gender: 'male',
    agreeToTerms: false,
    subscription: 'basic',
  });

  const inputRef = useRef(null);

  const handleChange = (event) => {
    const { name, value, type, checked } = event.target;
    setFormData({
      ...formData,
      [name]: type === 'checkbox' ? checked : value,
    });
  };

  const handleSubmit = (event) => {
    event.preventDefault();
    alert(`Hello ${formData.fullName}, Your Registration is Successful!`);
    handleReset();
  };

  const handleReset = () => {
    setFormData({
      fullName: '',
      email: '',
      password: '',
      gender: 'male',
      agreeToTerms: false,
      subscription: 'basic',
    });
  };


  const handleKeyUp = (event) => {
    if (!inputRef.current.contains(document.activeElement)) {
      if (event.key === 'Escape') {
        alert('Form closed with Esc key.');
        handleReset();
      }
    }
  };

  return (
    <div className="registration-container">
      <h2>Registration Form</h2>
      <form onSubmit={handleSubmit} onKeyUp={handleKeyUp} className="registration-form">
        <div className="form-group">
          <label htmlFor="fullName">Full Name:</label>
          <input
            type="text"
            name="fullName"
            placeholder='Enter full name'
            value={formData.fullName}
            onChange={handleChange}
            id="fullName"
            ref={inputRef}
            required
```

Name : Altaf Alam

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



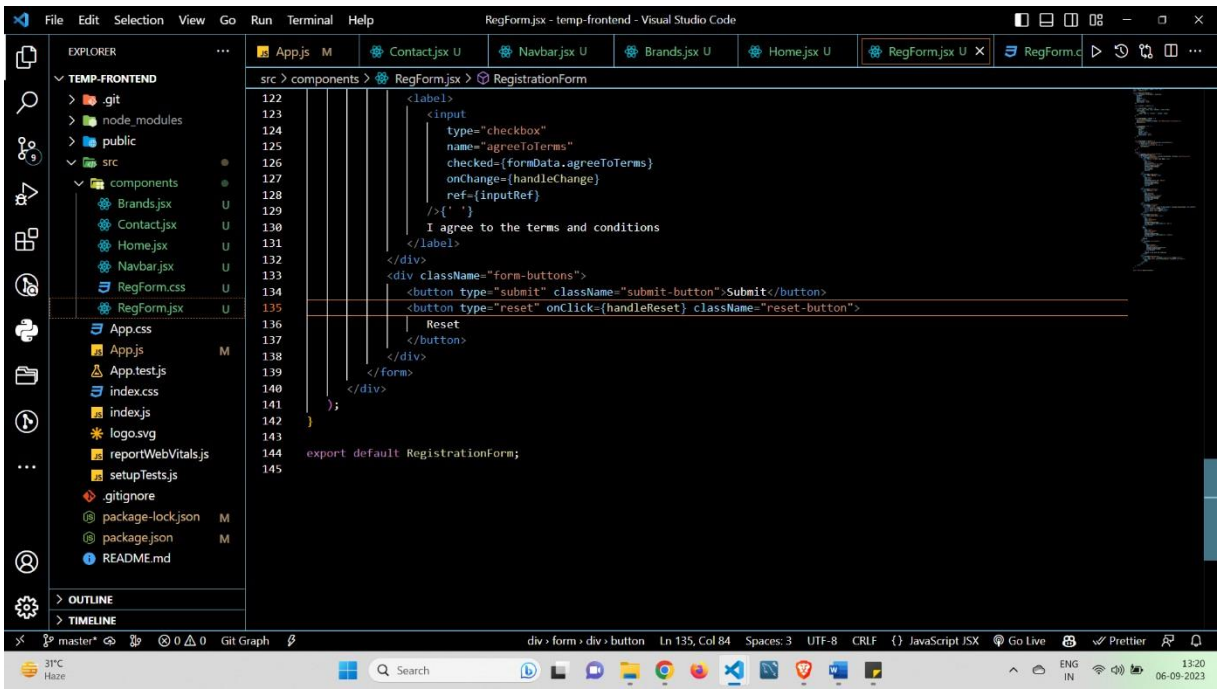
```
                />
            </div>
            <div className="form-group">
                <label >Email Address:</label>
                <input
                    type="email"
                    name="email"
                    placeholder='Enter email address'
                    value={formData.email}
                    onChange={handleChange}
                    ref={inputRef}
                    required
                />
            </div>
            <div className="form-group">
                <label>Password:</label>
                <input
                    type="password"
                    name="password"
                    placeholder='Enter password'
                    value={formData.password}
                    onChange={handleChange}
                    ref={inputRef}
                    required
                />
            </div>
            <div className="form-group">
                <label>Gender:</label>
                <select name="gender" value={formData.gender} onChange={handleChange} ref={inputRef}>
                    <option value="male">Male</option>
                    <option value="female">Female</option>
                    <option value="other">Other</option>
                </select>
```



```
                </select>
            </div>
            <div className="form-group">
                <label>Subscription Plan:</label>
                <input
                    type="radio"
                    name="subscription"
                    value="basic"
                    onChange={handleChange}
                    checked={formData.subscription === 'basic'}
                    ref={inputRef}
                />{' '}
                Basic
                <input
                    type="radio"
                    name="subscription"
                    value="premium"
                    onChange={handleChange}
                    checked={formData.subscription === 'premium'}
                    ref={inputRef}
                />{' '}
                Premium
            </div>
            <div className='form-checkbox'>
                <label>
                    <input
                        type="checkbox"
                        name="agreeToTerms"
                        checked={formData.agreeToTerms}
                        onChange={handleChange}
                        ref={inputRef}
                    />{' '}
                    I agree to the terms and conditions
                </label>
```

Name : Altaf Alam

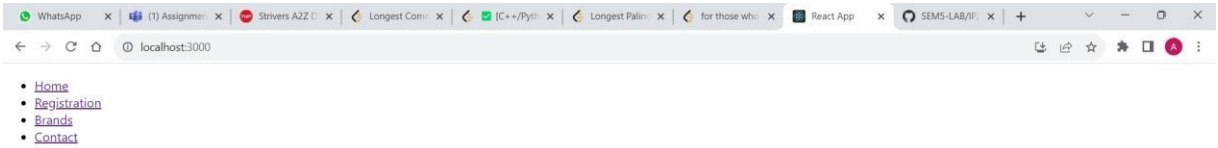
Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

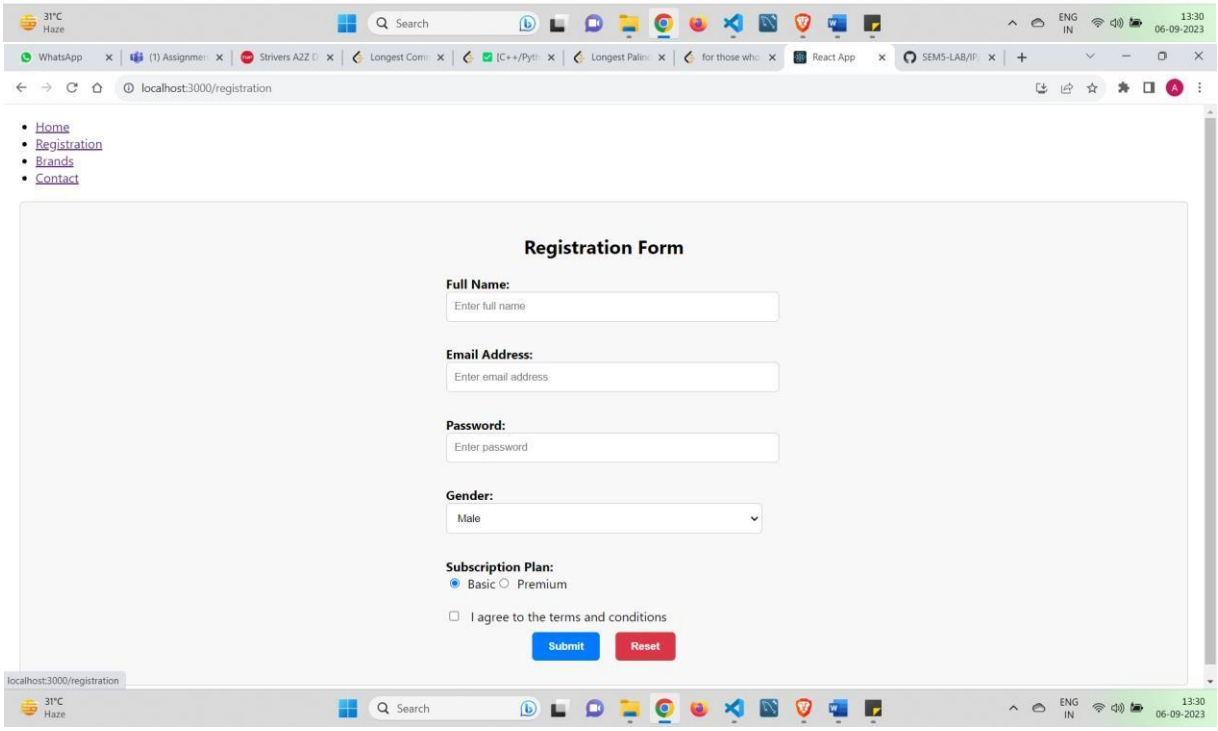


Output :

Name : Altaf Alam

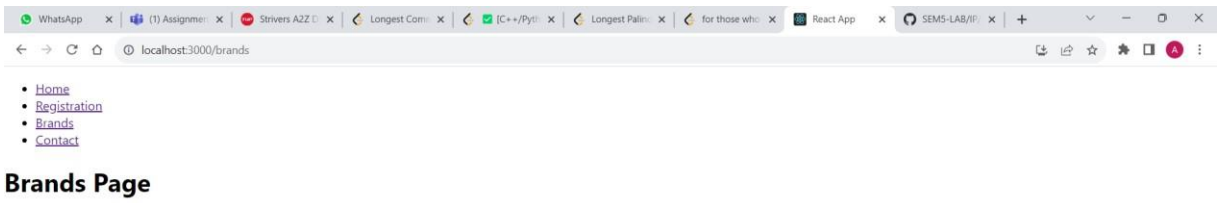
Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 02/08/23

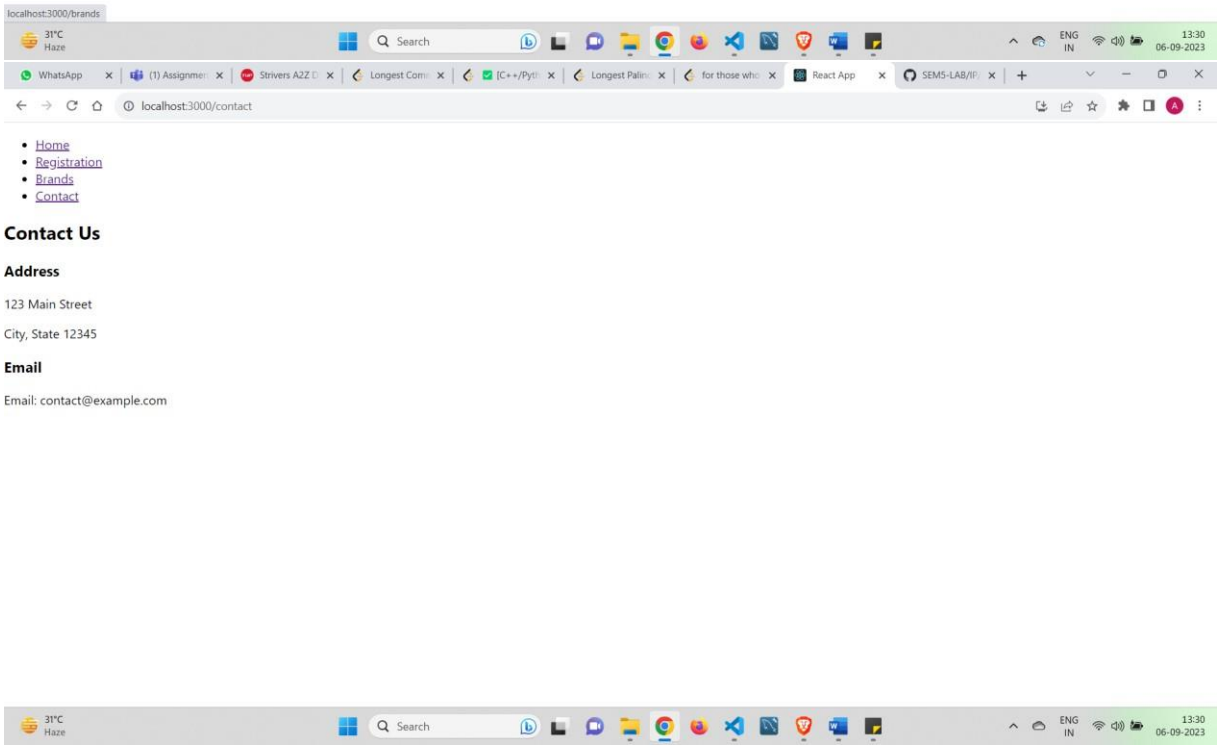




Name : Altaf Alam

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





Conclusion :

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

Name : Altaf Alam

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

Assignment No 7b

Aim : WAP to implement the concept of React Hooks.

Lab Outcome :

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

Theory :

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

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

1. State Management

2. Effect Handling

3. Context

4. Custom Hooks

5. Rules of Hooks

React Hooks offer several advantages:

- Readability: Functional components with Hooks are often more concise and easier to read than equivalent class components. They reduce boilerplate code.

- Reuse: Hooks promote logic reuse. Custom Hooks can be shared across different components and projects, enhancing code maintainability.

- Separation of Concerns: Hooks separate concerns by allowing logic related to state, effects, and context to be declared separately within a component.

Name : Altaf Alam

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

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

UseState:

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

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

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

useRef:

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

1. Mutable Refs: `useRef` creates a mutable object with a `.current` property. This property can be assigned a value, and it retains its value between renders without causing re-renders.

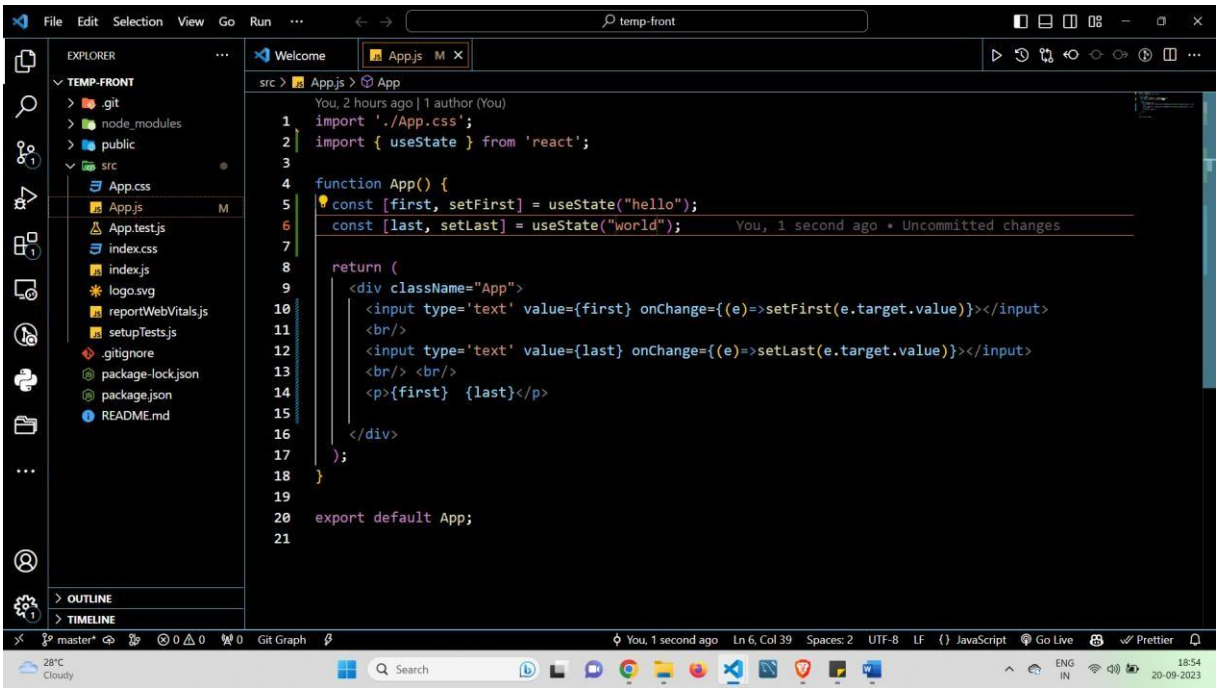
2. Accessing DOM Elements: One common use of `useRef` is to access and manipulate DOM elements directly. You can attach a `ref` attribute to a JSX element and assign it to a `useRef` variable.

3. Preserving Values: `useRef` is suitable for preserving values or state between renders without causing re-renders. This makes it different from `useState`.

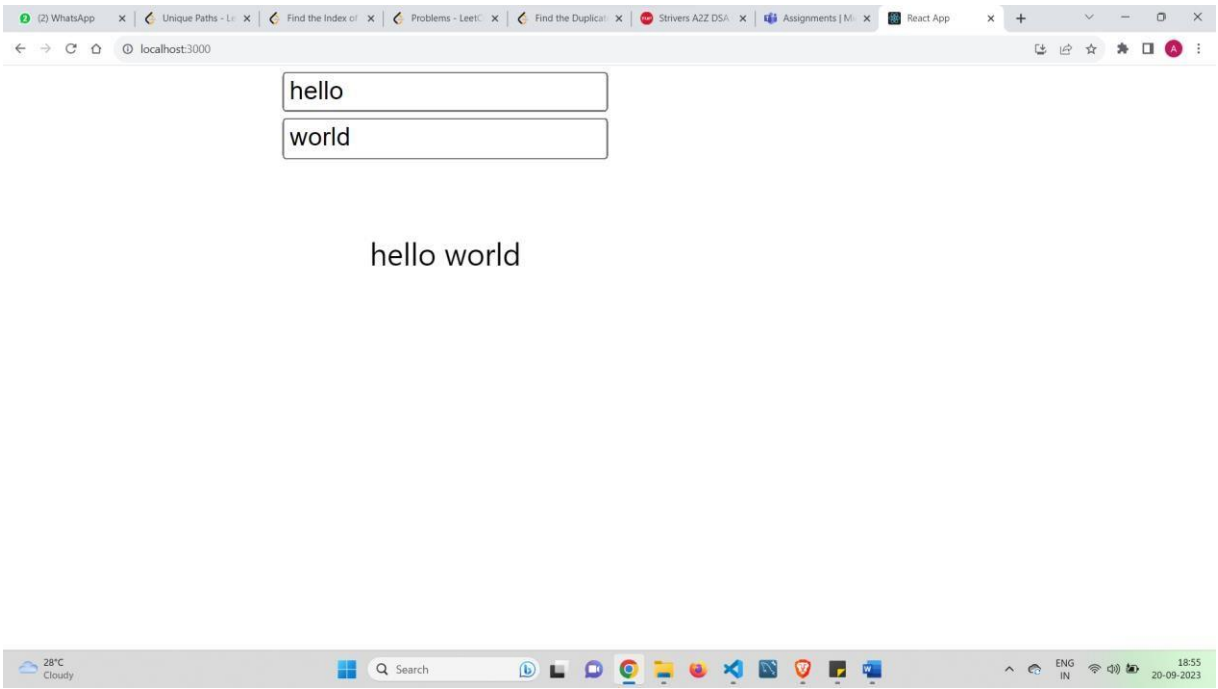
Name : Altaf Alam

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

Code :



Output :



Conclusion :

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

Name : Altaf Alam

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

Assignment No 8

Aim : WAP to implement the calculator on commandline REPL.

Lab Outcome :

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

Theory :

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

1. REPL (Read-Eval-Print Loop): The program utilizes a REPL environment, which stands for Read-Eval-Print Loop. This is an interactive command-line interface where users can input commands or expressions, and the system immediately evaluates and returns results. It's commonly used for experimentation, testing, and quick code execution.

2. Importing Modules: The program begins by importing the 'readline' module, which is a core Node.js module for reading input from the user in the command line. This module provides an interface for user interaction.

3. Creating the Readline Interface: An instance of the 'readline' interface is created, specifying that it should read input from 'stdin' (standard input) and display output in 'stdout' (standard output). The 'terminal: false' option is set to disable certain terminal-specific features.

4. User Input and Prompts: The program uses the 'rl.question' function to prompt the user for input in a series of steps. It starts by asking for the first number, then the second number, and finally the desired mathematical operation (addition, subtraction, multiplication, or division).

5. Parsing User Input: After gathering user input, the program parses the entered values into numeric format (floats) to ensure they are suitable for mathematical operations. This step involves using 'parseFloat' to convert the input.

6. Mathematical Operations: Depending on the user's chosen operation, the program performs the corresponding mathematical calculation (e.g., addition, subtraction, multiplication, or division). This is achieved using conditional statements, such as a 'switch' statement, to determine the operation and execute the appropriate calculation.

7. Result Display: The calculated result is then displayed on the command line for the user to see. It provides immediate feedback on the outcome of the operation.

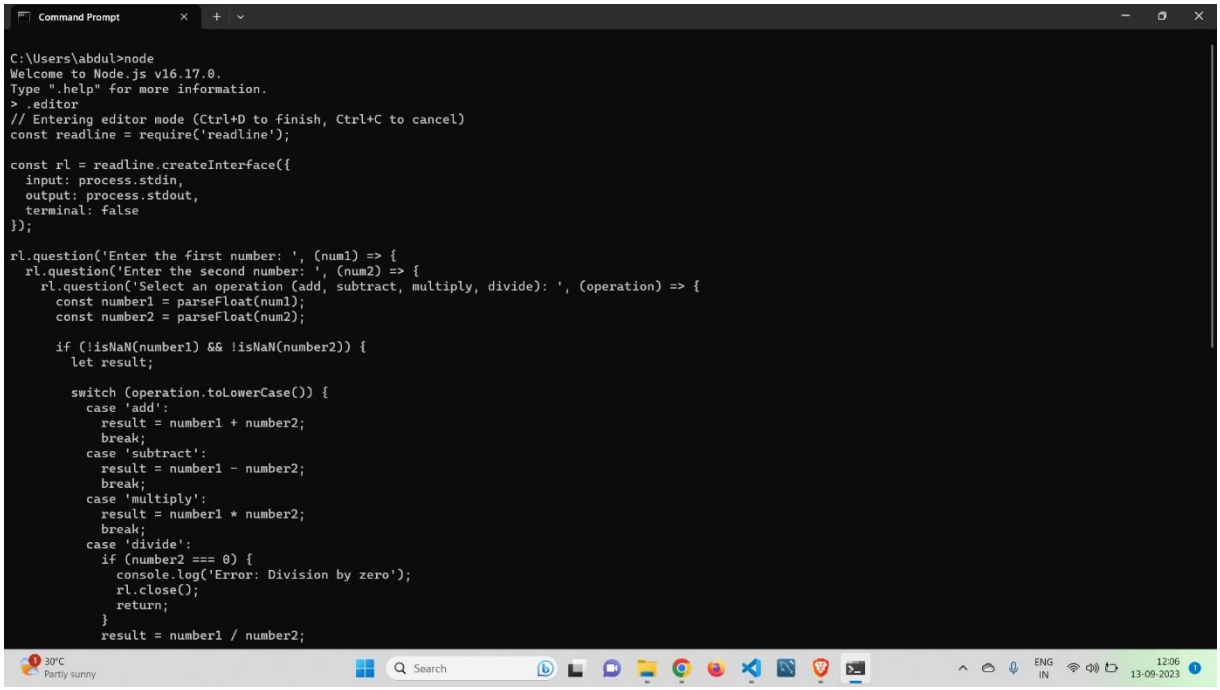
8. Error Handling: The program includes error handling to address potential issues. For instance, it checks for invalid inputs (non-numeric values) and handles division by zero to prevent errors and crashes.

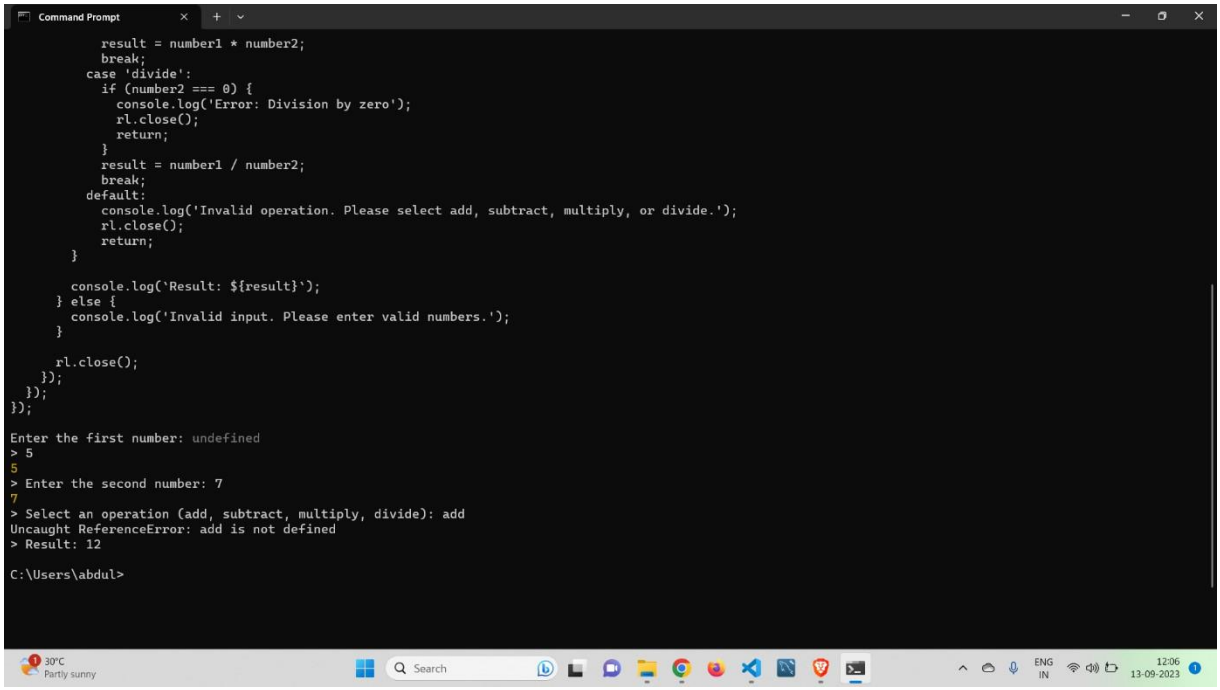
9. Interactive User Experience: Users interact with the program by entering input at each prompt, and the program responds with real-time feedback, making it a user-friendly calculator tool.

Output :

Name : Altaf Alam

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





Conclusion :

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

Name : Altaf Alam

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

Assignment No 9

Aim : Write a program to implement a.
Create React refs
b. How to access Refs
c. Forward Refs
d. Callback Refs.

Lab Outcome :

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

Theory :

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

a. Creating React Refs:

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

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

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

b. How to Access Refs:

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

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

```
It's
imp
orta
nt to
chec
k if
the
elem
ent
exist
s
befo
re
acce
ssin
g
and
mani
pulat
ing
it
beca
use
the
```

`current` property may initially be `null`.

c. Forward Refs:

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

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

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

```javascript function
ParentComponent() {
  const childRef = useRef(null);

  // Attach ref to the child component
return <ChildComponent ref={childRef} />;
}
```

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

d. Callback Refs:

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

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

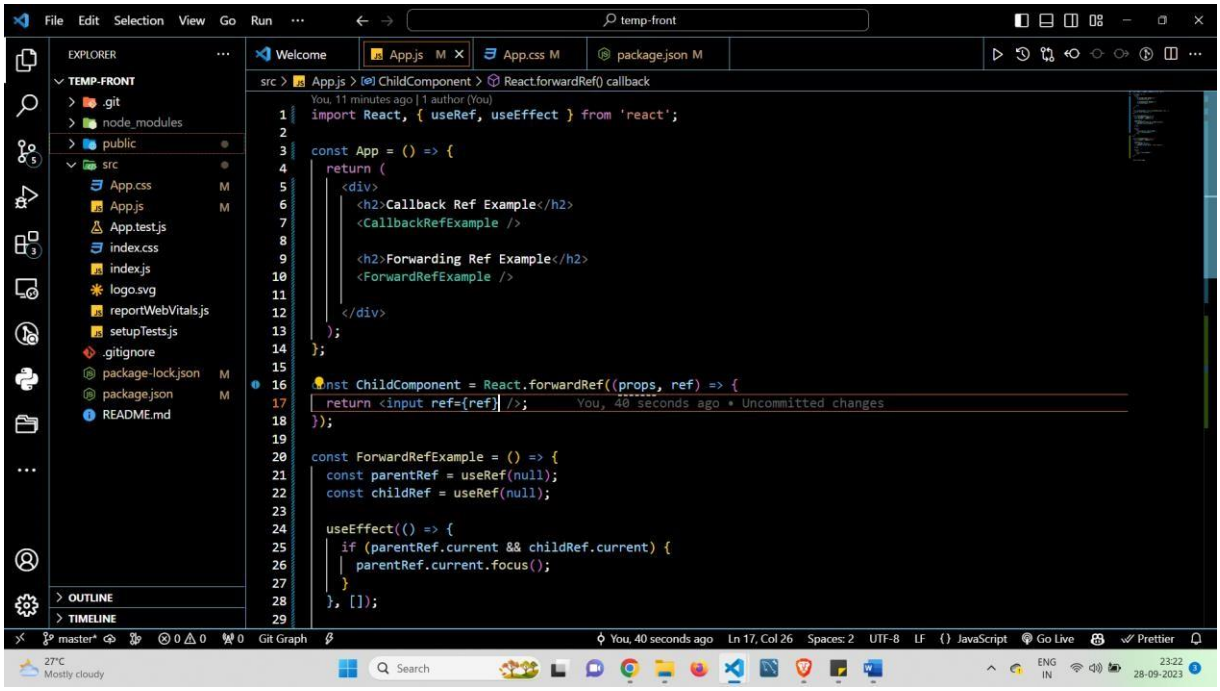
In this example, `setMyRef` is a callback function that receives the DOM element as an argument and assigns it to the `myRef` variable. You can then use `myRef` to access and manipulate the DOM element. Callback refs are typically defined inside the functional component.

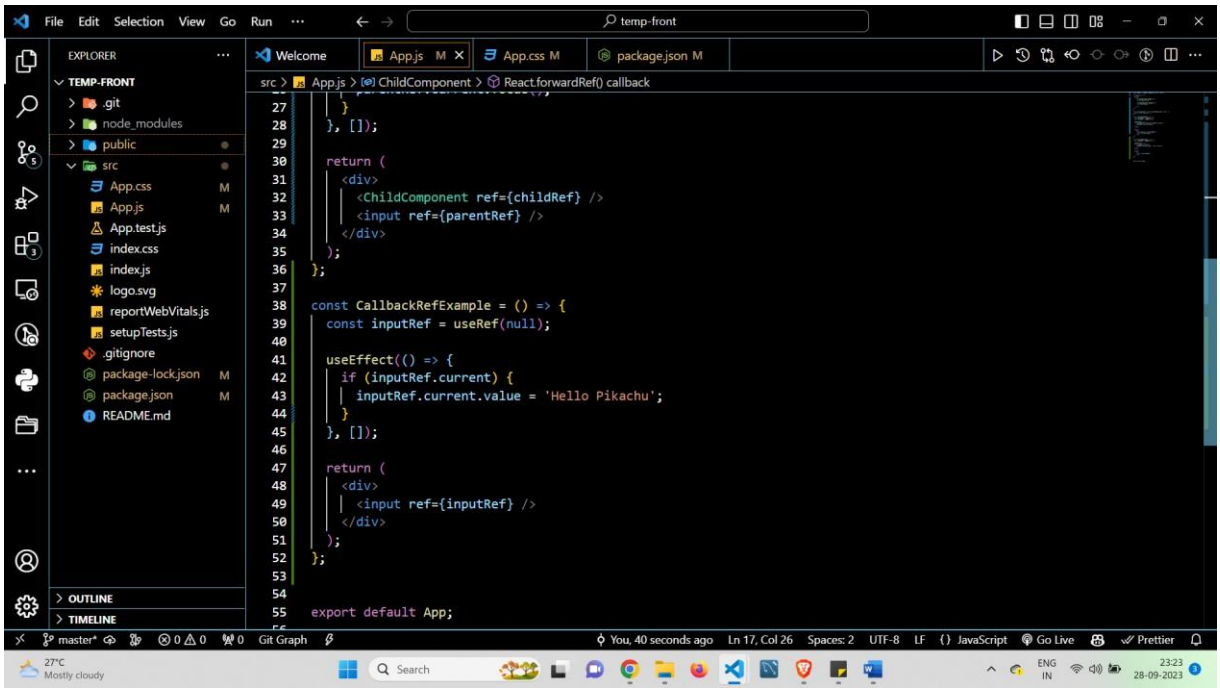
React Hooks and functional refs provide a flexible and powerful way to work with DOM elements in your React applications. Whether you're creating refs, accessing DOM elements, forwarding refs to child components, or using callback refs, these techniques empower you to build dynamic and interactive user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

Code :

Name : Altaf Alam

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

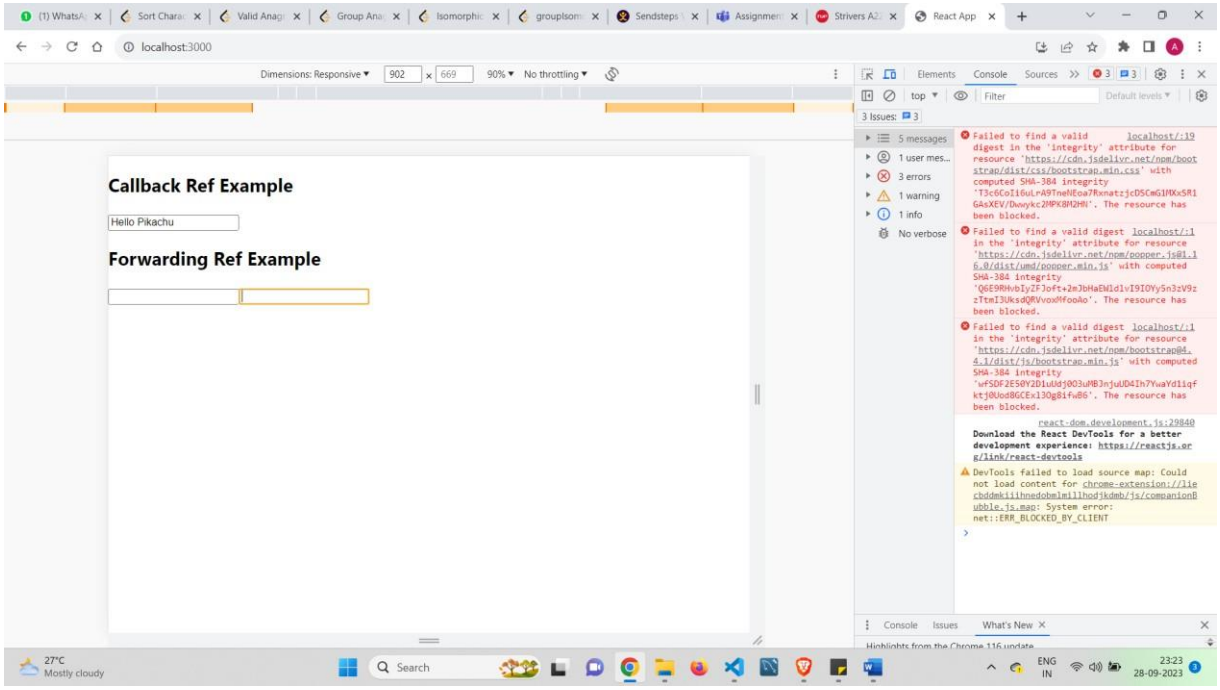




Output :

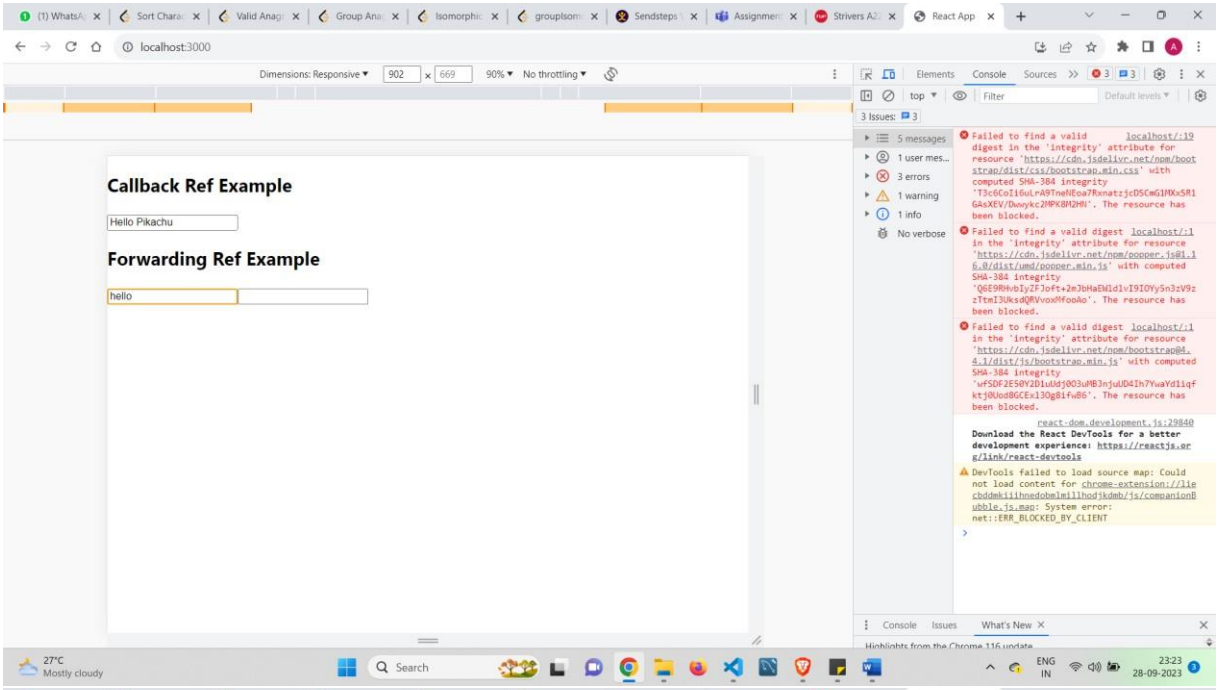
Name : Altaf Alam

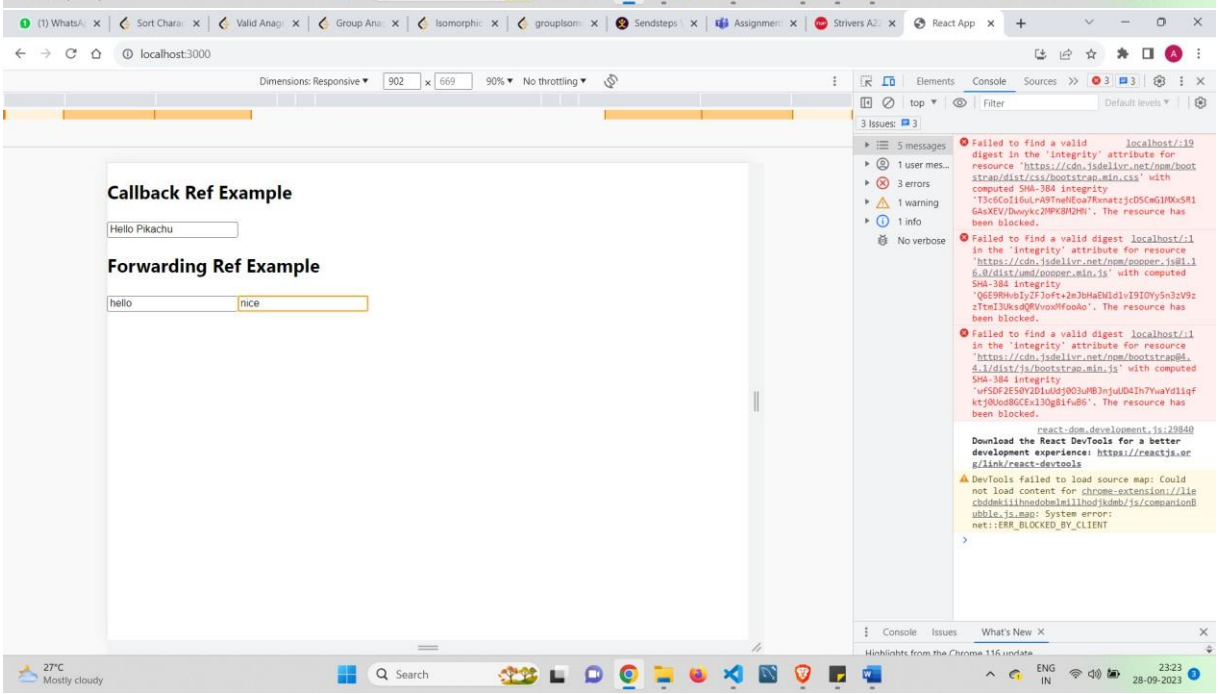
Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 28/09/23



Name : Altaf Alam

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





Conclusion :

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

Name : Altaf Alam

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

Assignment No 10

Aim : Write a program in Node JS to
a. Create a file
b. Read the data from file
c. Write the data to a file
d. Rename a file
e. Append data to a file
f. Delete a file

Lab Outcome :

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

Theory :

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

a. Creating a File

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

b. Reading Data from a File

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

c. Writing Data to a File

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

d. Renaming a File

Name : Altaf Alam

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

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

e. Appending Data to a File

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

f. Deleting a File

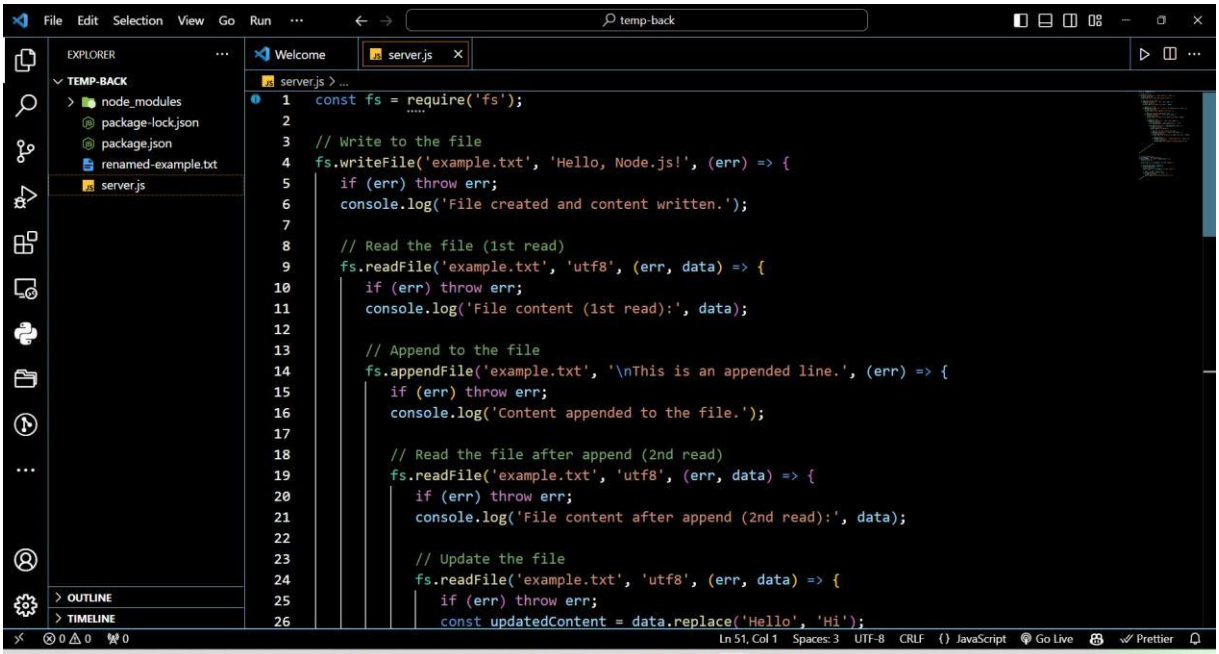
To delete a file, you utilize the 'fs' module's `fs.unlink()` method. You specify the file name as the argument, and this method removes the file from the file system.

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

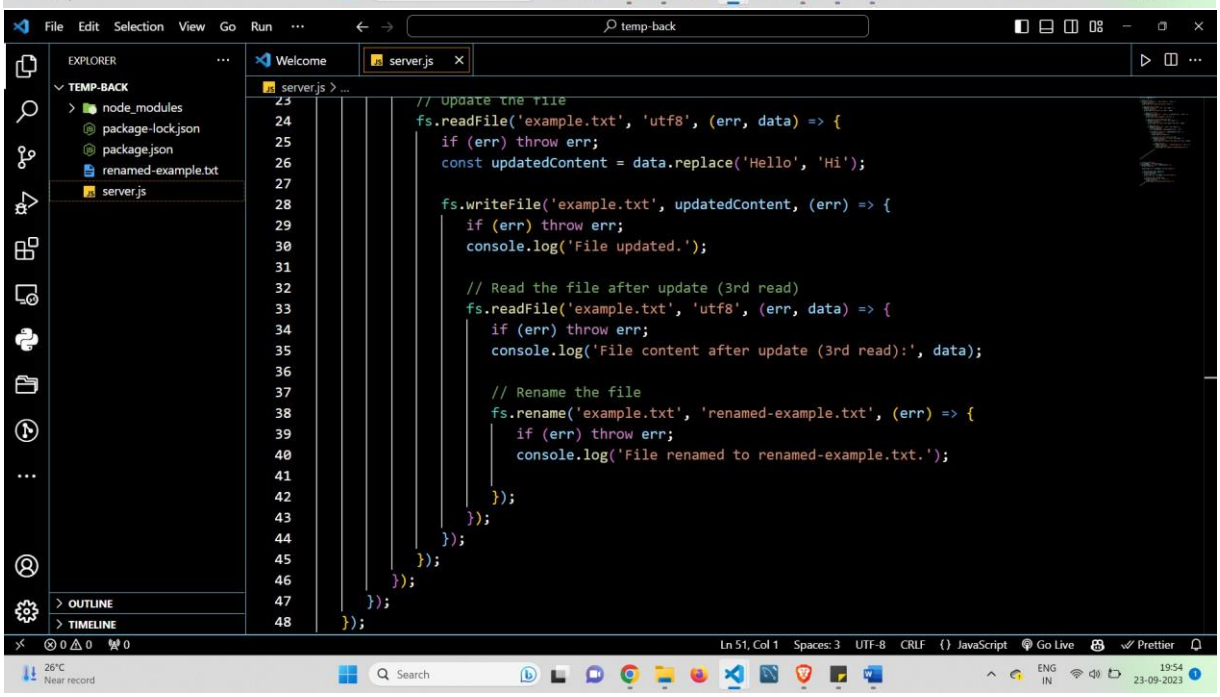
Code :

Name : Altaf Alam

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



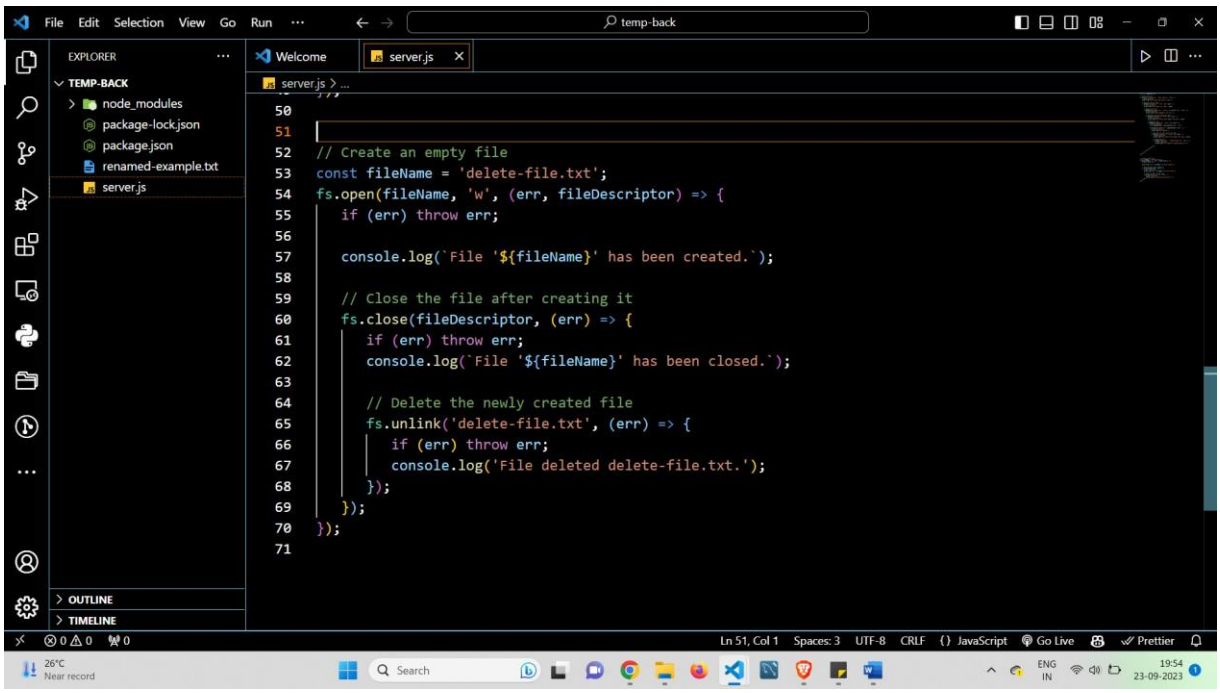
```
const fs = require('fs');

// Write to the file
fs.writeFile('example.txt', 'Hello, Node.js!', (err) => {
  if (err) throw err;
  console.log('File created and content written.');

  // Read the file (1st read)
  fs.readFile('example.txt', 'utf8', (err, data) => {
    if (err) throw err;
    console.log('File content (1st read):', data);

    // Append to the file
    fs.appendFile('example.txt', '\nThis is an appended line.', (err) => {
      if (err) throw err;
      console.log('Content appended to the file.');

      // Read the file after append (2nd read)
      fs.readFile('example.txt', 'utf8', (err, data) => {
        if (err) throw err;
        console.log('File content after append (2nd read):', data);

        // Update the file
        fs.readFile('example.txt', 'utf8', (err, data) => {
          if (err) throw err;
          const updatedContent = data.replace('Hello', 'Hi');
```



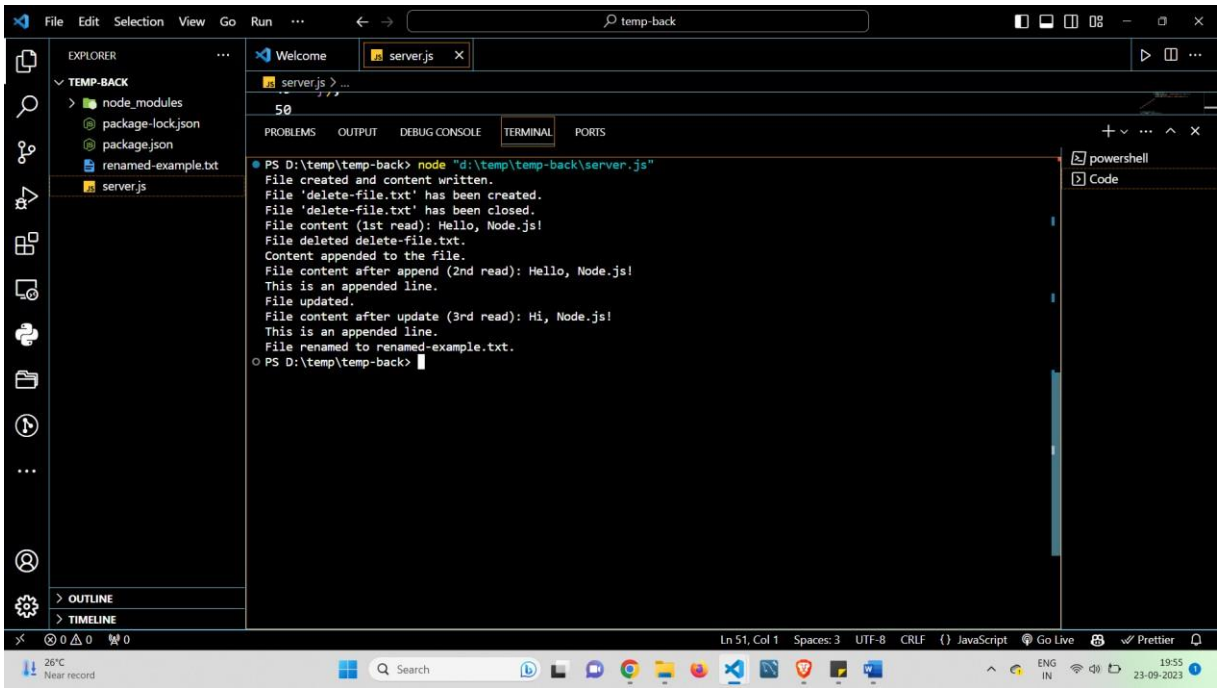
```
        // Update the file
        fs.readFile('example.txt', 'utf8', (err, data) => {
          if (err) throw err;
          const updatedContent = data.replace('Hello', 'Hi');

          fs.writeFile('example.txt', updatedContent, (err) => {
            if (err) throw err;
            console.log('File updated.');

            // Read the file after update (3rd read)
            fs.readFile('example.txt', 'utf8', (err, data) => {
              if (err) throw err;
              console.log('File content after update (3rd read):', data);

              // Rename the file
              fs.rename('example.txt', 'renamed-example.txt', (err) => {
                if (err) throw err;
                console.log('File renamed to renamed-example.txt.');

              });
            });
          });
        });
      });
    });
  });
});
```

Name : Altaf Alam

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



Output :



Conclusion :

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

Assignment No 11

Name : Altaf Alam

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

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

Lab Outcome :

LO1, LO2, LO3, LO4, LO5, LO6

Theory :

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

1. Create: Users can add new to-do items by entering text and clicking the "Add" button.

2. Read: The app displays a list of existing to-do items, and users can see their to-do list.

3. Update: Users can edit existing to-do items by clicking the "Edit" button, making changes, and clicking "Update."

4. Delete: Users can delete to-do items by clicking the "Delete" button.

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

Here's a brief overview for the project:

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

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

- `ToDo.js` is a functional component for rendering individual to-do items.

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

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

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

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

1. `server.js` sets up the Express.js server, establishes a connection to the MongoDB database, and listens on a specified port (in this case, port 5000).

2. `ToDoRoute.js` defines the routes for CRUD operations using Express Router. It specifies the routes for `get`, `post` (create), `post` (update), and `post` (delete) operations.

3. `ToDoController.js` contains controller functions for handling the CRUD operations.

- `getToDo` retrieves all to-do items from the database.
- `saveToDo` creates a new to-do item.
- `updateToDo` updates an existing to-do item.
- `deleteToDo` deletes a to-do item.

Database (`models/ToDoModel.js`):

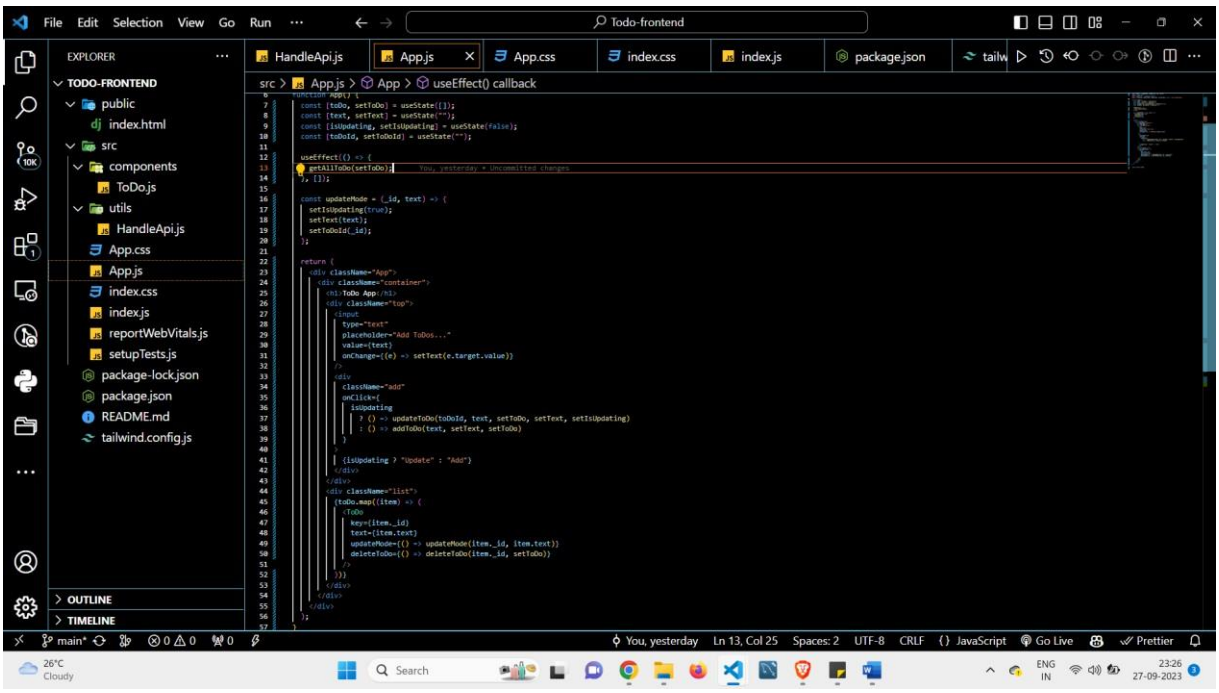
1. `ToDoModel.js` defines the MongoDB schema for to-do items, specifying that each item has a `text` field.

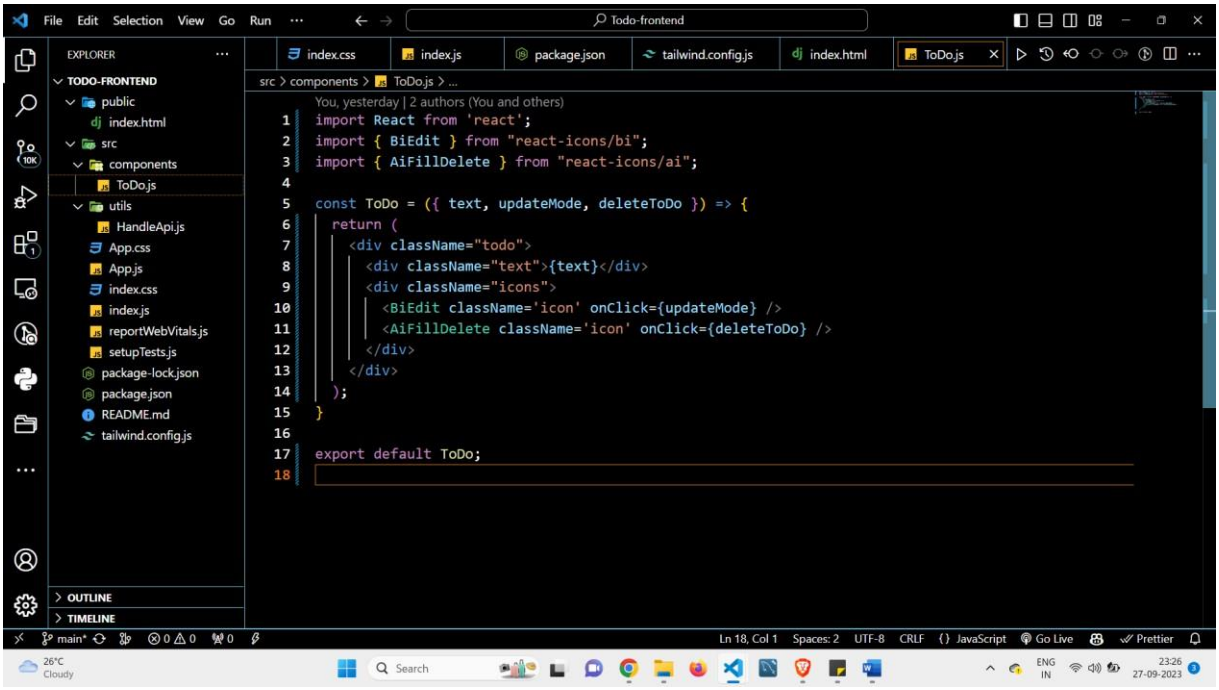
Code :

Frontend :

Name : Altaf Alam

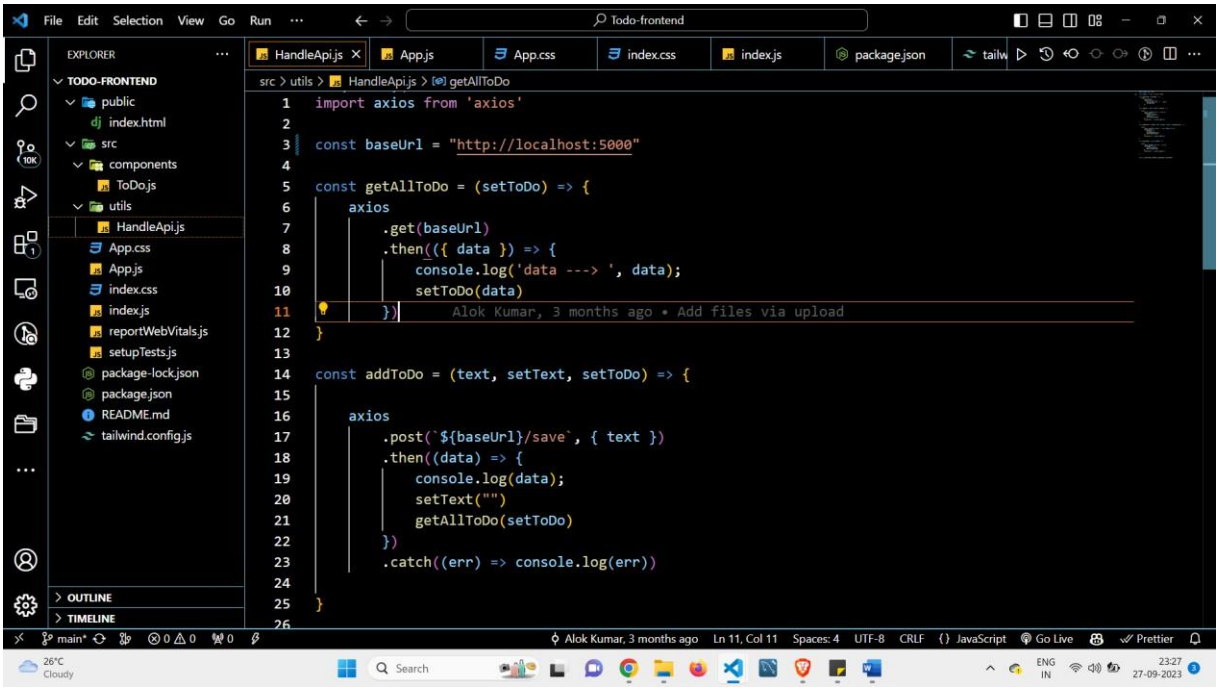
Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 23/09/23



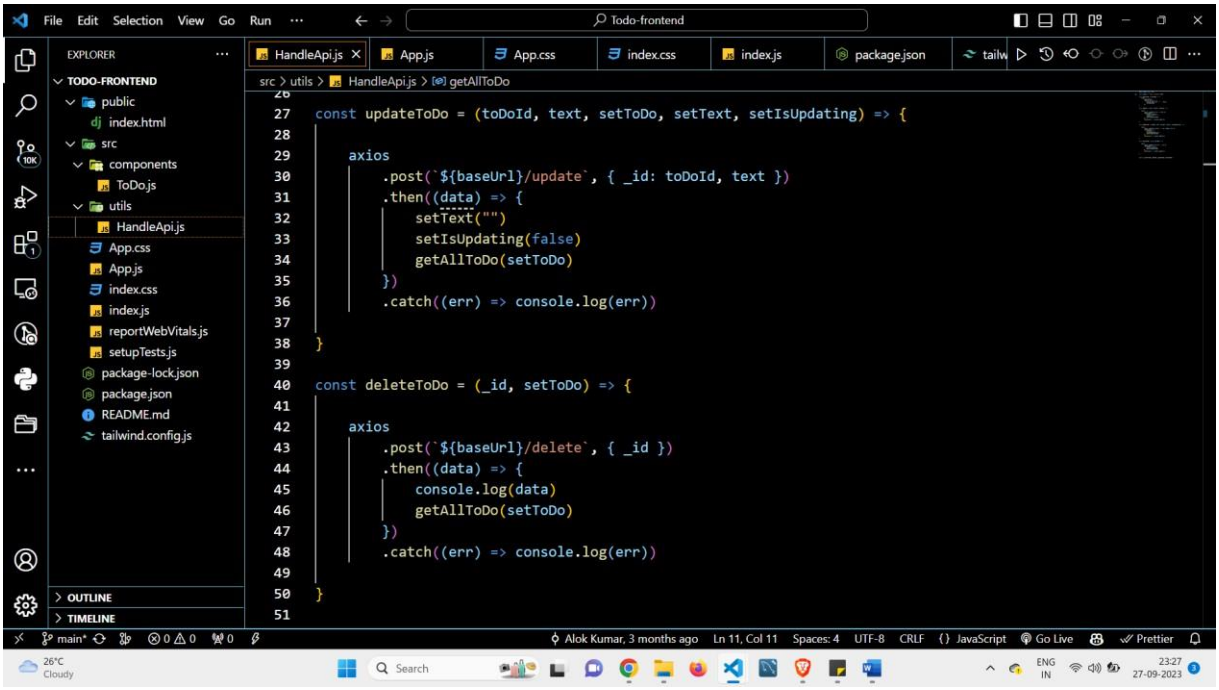


Name : Altaf Alam

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



```
import axios from 'axios'

const baseUrl = "http://localhost:5000"

const getAllToDo = (setToDo) => {
    axios
        .get(baseUrl)
        .then(({ data }) => {
            console.log('data ---> ', data);
            setToDo(data)
        })
}

const addToDo = (text, setText, setToDo) => {

    axios
        .post(`${baseUrl}/save`, { text })
        .then((data) => {
            console.log(data);
            setText("")
            getAllToDo(setToDo)
        })
        .catch((err) => console.log(err))

}
```

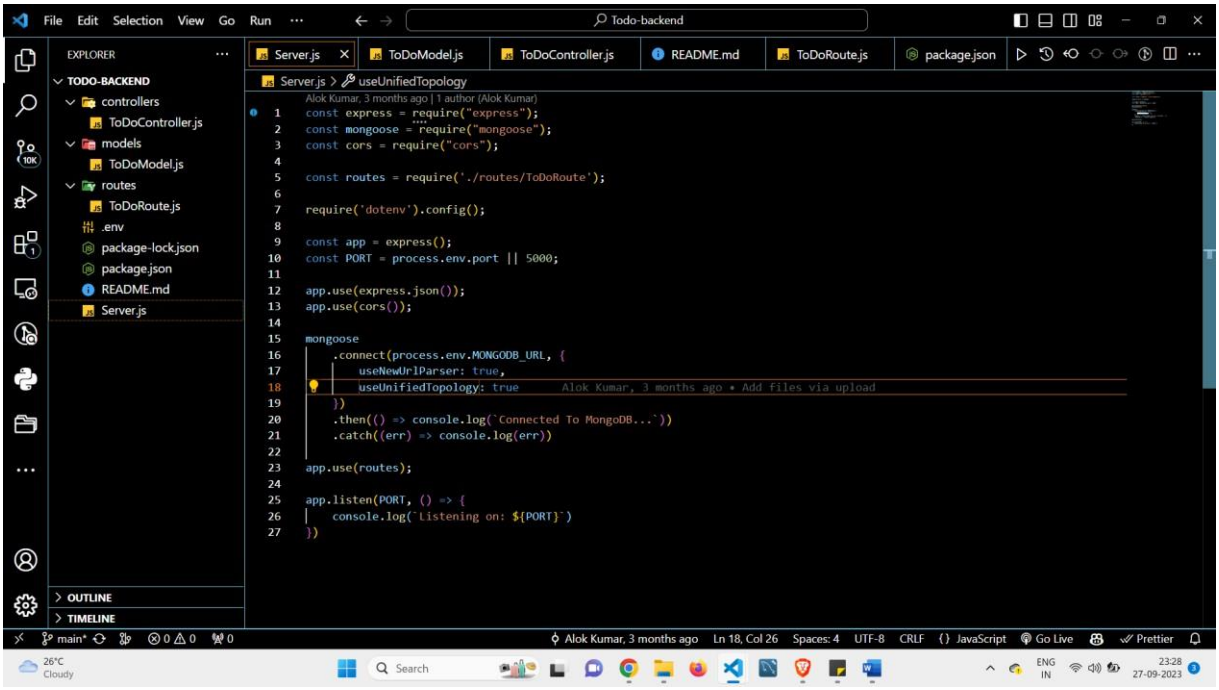


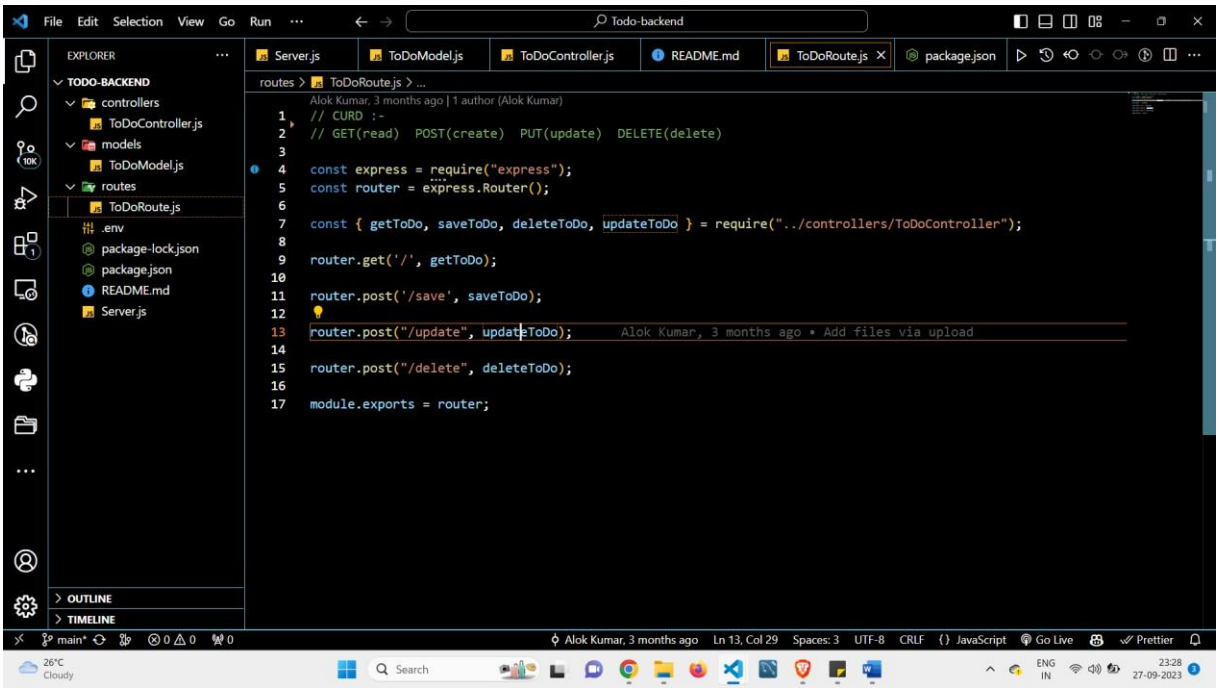
```
const updateToDo = (toDoId, text, setToDo, setText, setIsUpdating) => {

    axios
        .post(`${baseUrl}/update`, { _id: toDoId, text })
        .then((data) => {
            setText("")
            setIsUpdating(false)
            getAllToDo(setToDo)
        })
        .catch((err) => console.log(err))

}

const deleteToDo = (_id, setToDo) => {

    axios
        .post(`${baseUrl}/delete`, { _id })
        .then((data) => {
            console.log(data)
            getAllToDo(setToDo)
        })
        .catch((err) => console.log(err))

}
```

Name : Altaf Alam

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

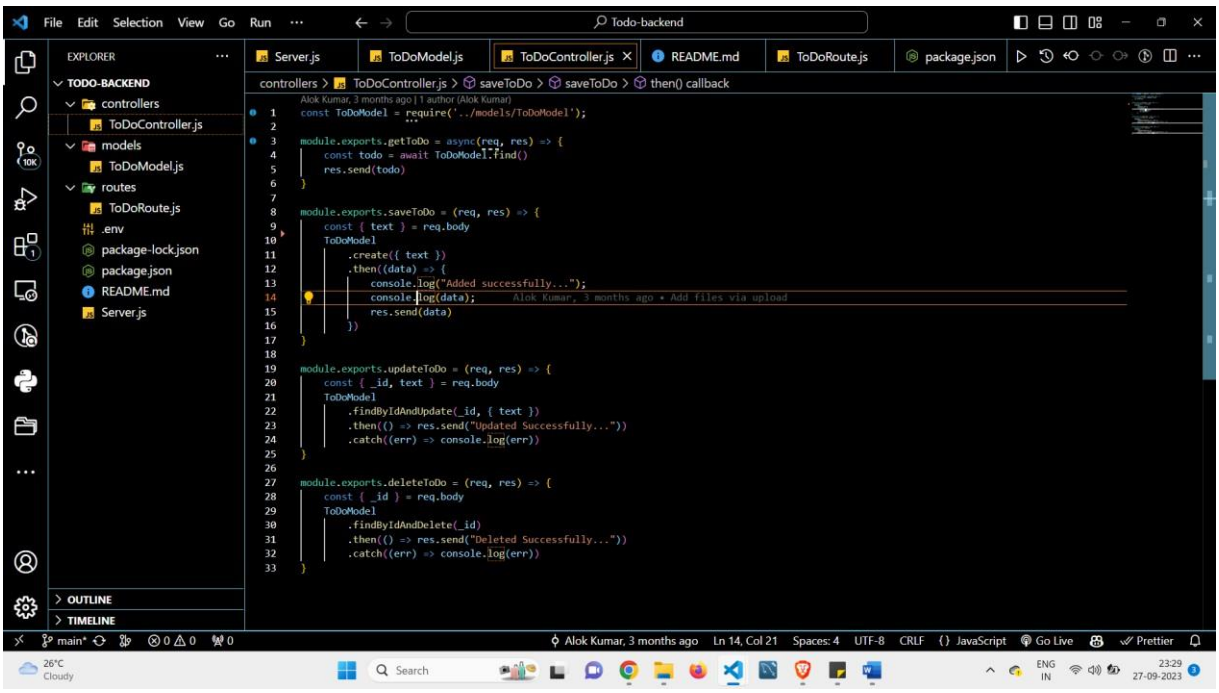
Backend :

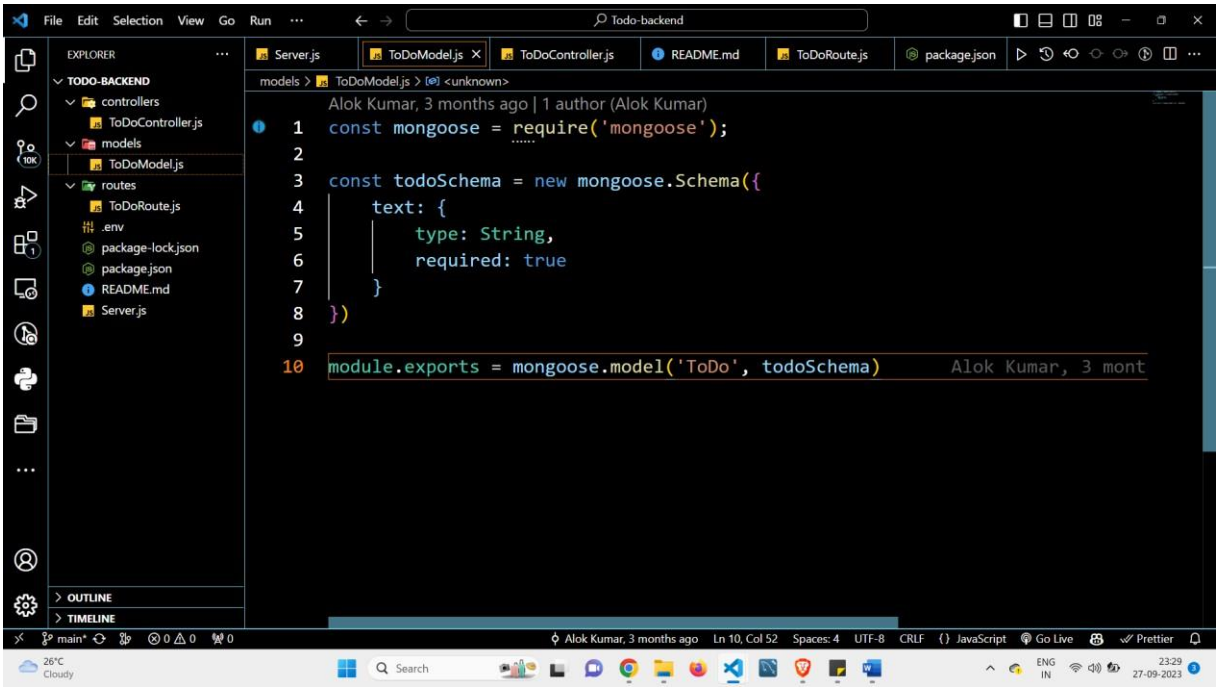




Name : Altaf Alam

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

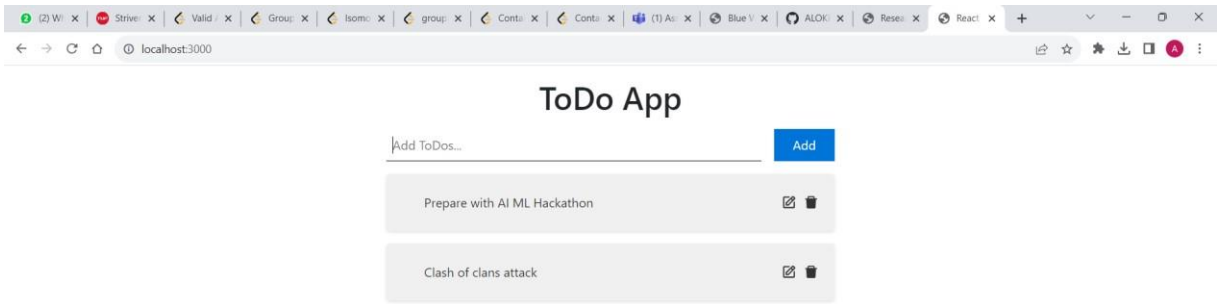


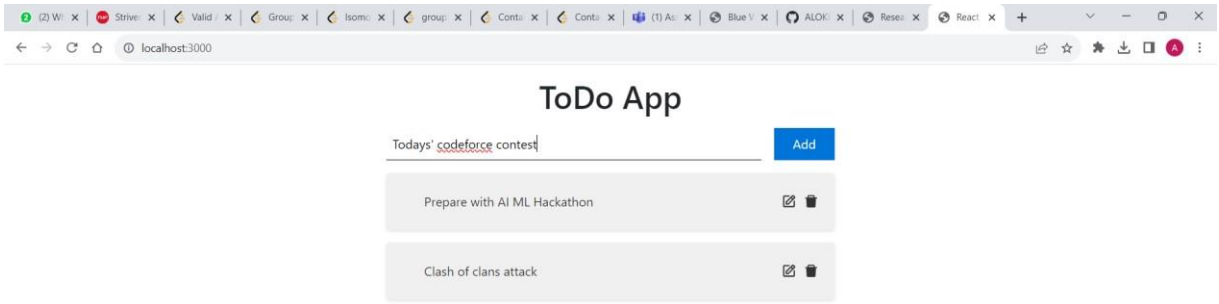


Name : Altaf Alam

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

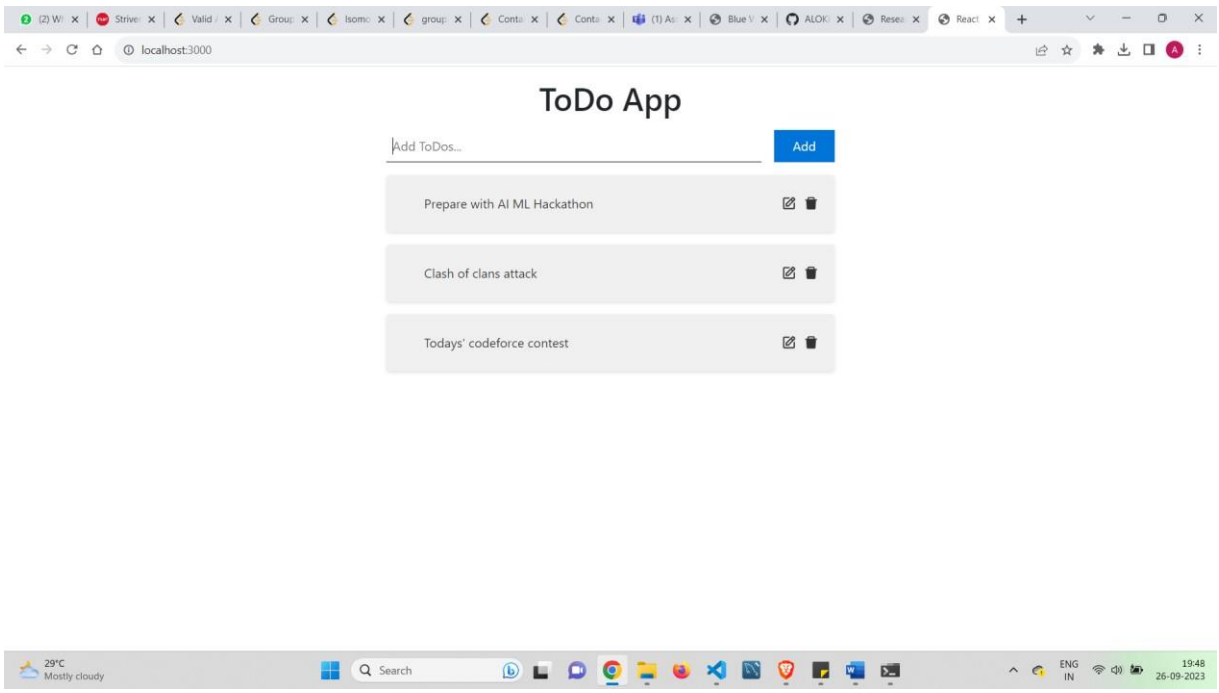
Output :

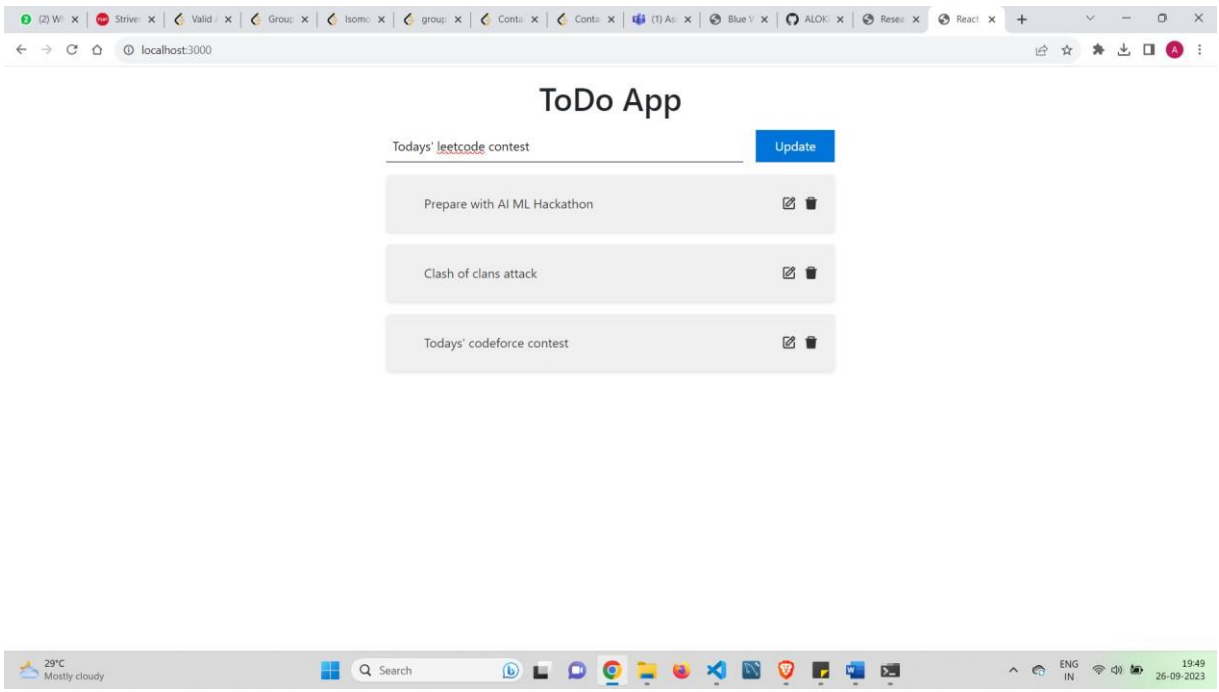




Name : Altaf Alam

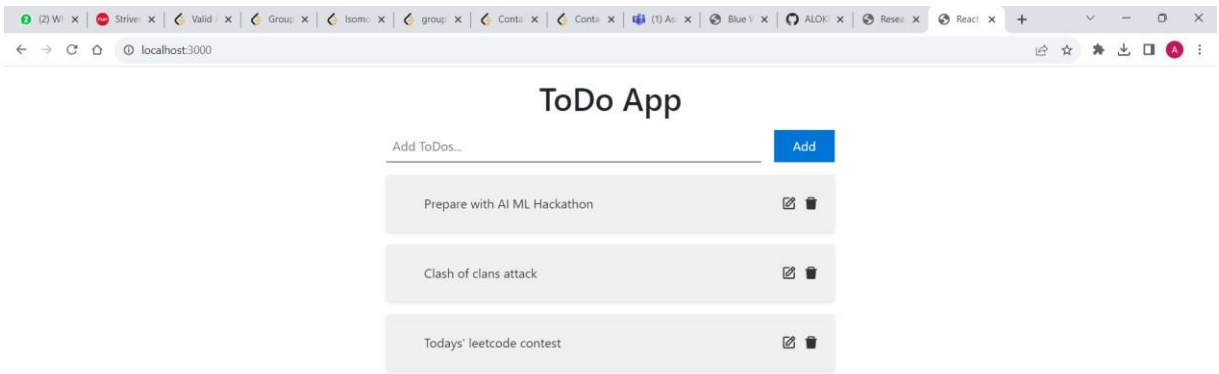
Roll no : 02 , Batch : T11 , Subject : IP Lab , Date : 23/09/23

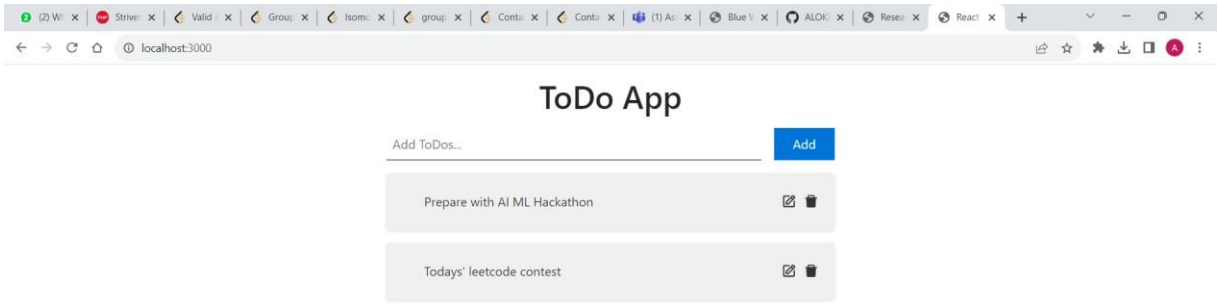




Name : Altaf Alam

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





Conclusion :

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

Name : Altaf Alam

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

Written Assignment No 1

Compare XML and JSON.

Comparing XML and JSON: Structured Data Representation

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

Similarities:

1. Human-Readable Format: Both XML and JSON are designed to be human-readable, making them easy to understand and edit by developers and users alike.

2. Hierarchical Structure: Both formats support a hierarchical structure, enabling the representation of complex data relationships and nested elements.

3. Platform-Independence: XML and JSON can be used across various programming languages and platforms, making them versatile choices for data interchange.

4. Support for Arrays/Collections: Both formats allow for the representation of arrays or collections of data elements.

Differences:

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

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

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

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

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

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

Use Cases:

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

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

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

Explain different types of arrow functions

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

Name : Altaf Alam

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

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

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

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

Arrow Function with Multiple Statements:
If you need to perform multiple statements inside an arrow function, you can use curly braces and specify the `return` keyword explicitly.

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

Arrow Function with No Parameters:
You can create an arrow function with no parameters by providing an empty set of parentheses.

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

Arrow Functions with Object literals:

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

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

Arrow Function with Rest Parameters:
You can use the rest parameter syntax to capture an arbitrary number of arguments into an array.

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

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

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

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

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

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

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

What is DNS? Explain working of DNS.

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

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

Working of DNS:

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

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

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

4. Response and Caching:

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

5. Data Transmission:

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

Benefits and Importance:

1. User-Friendly Experience: DNS enables users to access websites, services, and resources using easy-to-remember domain names instead of complex IP addresses.

2. Scalability: DNS is a distributed system, ensuring efficient and reliable resolution for the billions of domain names on the Internet.

3. Redundancy and Load Balancing: Multiple DNS servers exist for each domain, providing redundancy. Load balancing can distribute user requests across various servers for improved performance.

4. Dynamic IP Addresses: DNS allows dynamic IP addresses to be associated with a domain name, making it possible to host websites and services on changing IP addresses.

5. Global Accessibility: DNS ensures worldwide access to websites and services, regardless of users' geographic locations.

6. Security Considerations: DNS also plays a role in security by implementing techniques like Domain Name System Security Extensions (DNSSEC) to prevent DNS-related attacks like DNS spoofing.

Explain Promises in ES6.

Promises in ES6: Managing Asynchronous Operations with Confidence

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

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

Understanding Promises:

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

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

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

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

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

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

```
const promise1 = fetch(url1); const
promise2 = fetch(url2);

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

Async/Await with Promises:
ES2017 introduced the `async` and `await` keywords, which provide a more synchronouslike syntax for handling promises. The `async` keyword is used to define an asynchronous function, while `await` is used inside that function to pause execution until a promise is resolved.

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

Benefits of Promises:

1. Readability: Promises improve code readability by eliminating nested callbacks and providing a more linear flow.

2. Error Handling: Promises provide a structured way to handle errors using `.catch()`, making error management more organized.

3. Chaining: Chaining promises simplifies handling sequential asynchronous tasks, making code easier to understand and maintain.

4. Parallelism: `Promise.all()` allows parallel execution of multiple promises, enhancing performance when dealing with independent tasks.

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

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

Name : Altaf Alam

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

Assignment No 2

Aim :

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

Theory :

1: Refs in JavaScript

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

When to Use Refs

Refs are typically used in the following scenarios:

1. Accessing DOM Elements: When you need to access a specific DOM element directly. This is often necessary for tasks like setting focus on an input field, reading the dimensions of an element, or manually triggering events.

2. Managing Third-y Libraries: Integrating third-y libraries or plugins that don't play well with React's virtual DOM can require the use of refs to access and manipulate their elements.

3. Animations: When you need to animate an element or apply animations using libraries like GreenSock Animation Platform (GSAP), you may need to use refs to target the DOM element to animate.

4. Managing Uncontrolled Components: In some cases, you may need to work with uncontrolled components (e.g., HTML input elements) where React state is not directly linked. Refs allow you to interact with these components without relying on React's state management.

When Not to Use Refs

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

1. Overusing Refs: Avoid using refs for common tasks that can be managed through React's state and props. Overusing refs can make your code harder to maintain and debug.

2. Breaking Component Isolation: Refs can break the encapsulation and isolation of React components. It's often better to pass data and callbacks as props instead of using refs to directly manipulate child components.

3. Avoiding Functional Components: With the introduction of React Hooks, functional components provide a more modern and concise way to manage component state and side effects. In functional components, you can use the `useRef` hook instead of the older `createRef` method.

2: NPM (Node Package Manager)

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

Key Features of NPM

1. Package Installation: NPM allows developers to install packages and libraries from the NPM registry. You can use the `npm install` command followed by the package name to add dependencies to your project.

2. Dependency Management: NPM manages project dependencies by maintaining a `package.json` file, which lists all the packages your project relies on, along with their versions. This makes it easy to share projects and ensure consistent dependencies across development teams.

3. Scripts: NPM provides a powerful scripting system defined in the `package.json` file. You can define custom scripts for tasks like building, testing, or running your application. These scripts can be executed using the `npm run` command.

4. Version Control: NPM allows you to specify version ranges for your project dependencies. This ensures that your project can easily update to the latest compatible versions of packages while maintaining stability.

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

Basic NPM Commands

Here are some essential NPM commands:

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

3: REPL (Read-Eval-Print Loop)

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

How to Use the Node.js REPL

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

- Entering Code: Simply type JavaScript code and press `Enter` to execute it. The result will be displayed immediately.

- Variables: You can declare and work with variables within the REPL.

```javascript
> let x = 10;
undefined
> x + 5
15
```

```

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

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

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

```javascript
> 2 + 3
5
> _ 2
10
```

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

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

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

4: Routing in Express.js
```

Name : Altaf Alam

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

Express.js Routing Basics

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

Setting Up Express.js

1. Install Express: Use `npm install express` to add Express as a dependency.

2. Create an Express App:

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

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

Basic Routing

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

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

Route Parameters

Define routes with parameters to capture values from the URL:

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

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

Handling POST Requests

Handle POST requests with Express:

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

Route Middleware

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

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

app.use(logRequest);
```

Error Handling

Define error-handling middleware for error responses:

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

```
});
```

Building a Simple API

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

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

app.use(express.json());

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

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

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

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

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

Name : Altaf Alam

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

```
```