# Industry Project

# Report

## On

# DEVS TREE

| Developed By: - | Guided By:- |
|---|---|
| Anuj jani (21162102003) | Prof. Umang Thakkar sir (Internal) |
| | Mr. Arath patel sir (External) |

## Submitted to
## Department of Computer Science & Engineering
## Institute of Computer Technology

Ganpat University
॥ विद्यया समाजोत्कर्षः ॥
Institute of Computer Technology
National Assessment and Accreditation Council NAAC A Grade Accredited University

## Year - 2024

# CERTIFICATE

This is to certify that the Industry Work as “react js inter” by Anuj jani (21162102003) of Ganpat University, towards the partial fulfillment of requirements of the degree of Bachelor of Technology – Computer Science and Engineering, carried out by them in the CSE (CBA) Department.The results/findings contained in this Project have not been submitted in part or full to any other University / Institute for award of any other Degree/Diploma.

Name & Signature of Internal Guid

Name & Signature of Head

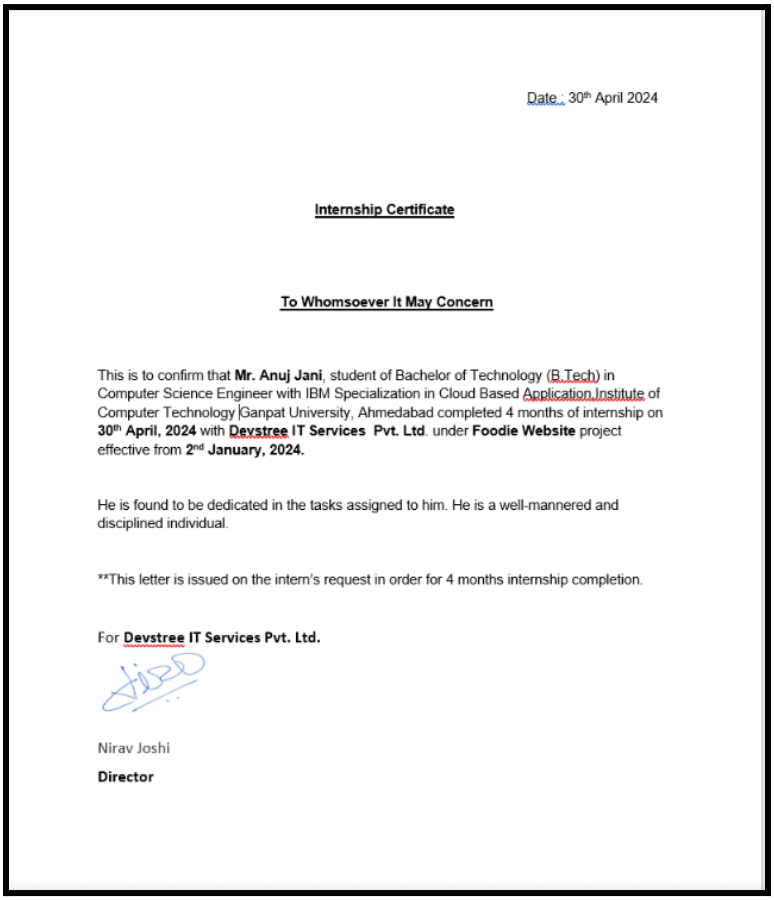
**Place: ICT – GUNI**

**Date:**

# ACKNOWLEDGEMENT

I would like to extend my sincere appreciation to my internship supervisor, Prof Umang Thakkar sir (internal) and Team leader Arath Patel sir(external) , for their invaluable guidance and support throughout my internship. I am also Grateful to Dr. Rohit Patel sir, Principal of ICT, and Prof. Dharmesh Darji sir, Head of ICT, for their mentorship and expertise in the field of React JS. Their collective wisdom and encouragement have significantly contributed to my professional growth and development during this period. I am thankful for their unwavering support and willingness to share their knowledge with me.

## Certificate



Date : 30th April 2024

**Internship Certificate**

**To Whomsoever It May Concern**

This is to confirm that **Mr. Anuj Jani**, student of Bachelor of Technology (B.Tech) in Computer Science Engineer with IBM Specialization in Cloud Based Application,Institute of Computer Technology |Ganpat University, Ahmedabad completed 4 months of internship on **30th April, 2024** with **Devstree IT Services Pvt. Ltd**. under **Foodie Website** project effective from **2nd January, 2024.**

He is found to be dedicated in the tasks assigned to him. He is a well-mannered and disciplined individual.

**This letter is issued on the intern's request in order for 4 months internship completion.

For **Devstree IT Services Pvt. Ltd.**

Nirav Joshi

**Director**

Anuj Jani
21162102003

# ABSTRACT

As an aspiring web developer, this report documents my immersive journey into the captivating world of React.js. Fueled by the desire to craft dynamic and user-centric interfaces, I embarked on a mission to leverage the power of this versatile JavaScript library. This experience was more than just building interfaces; it was a transformative exploration of component-based architecture, state management, and the art of creating engaging user experiences.My initial steps involved mastering the fundamentals of React. Understanding the concept of virtual DOM, JSX syntax, and component lifecycle became the building blocks of my learning. Pixel by pixel, I constructed reusable components, appreciating their modularity and promoting maintainability.This internship solidified my belief in React.js as a powerful tool for shaping the future of web development. Its component-based approach, developer-friendly ecosystem, and vast community empower developers to build performant, scalable, and user-centric web applications

# Index

# CHAPTER: 1 INTRODUCTION

## CHAPTER: 1 INTRODUCTION

React JS is a well-liked JavaScript library for creating user interfaces, particularly for applications with only one page. It was created by Facebook and is presently kept up with by Facebook and a local area of individual designers and organizations.

React JS's key feature is its component-based architecture. Developers can construct complex user interfaces by breaking them down into smaller, reusable components. The code is more modular, easier to maintain, and more effective because each component manages its own logic and state.

Respond JS likewise gives a strong arrangement of elements for dealing with the condition of parts, like useState and useEffect. Developers can use these hooks to declaratively manage the state of components, making the code more predictable and easier to reason about.

# CHAPTER: 2 PROJECT SCOPE

# CHAPTER: 2 PROJECT SCOPE

The food website will provide users with a platform to browse and order food from various restaurants in their area.

The second project is to build a modern landing page with smooth animations that stand out from other websites.

The landing page will include a hero section with different shapes reacting to the cursor, a navigation bar with a colorful button that transforms into an animated hamburger menu on smaller screens, and a subtle smooth scroll parallax effect that moves the elements down as we scroll

# CHAPTER: 3 Requirement

## CHAPTER: 3 Requirement

### Hardware Requirements

| | |
|---|---|
| **Processor** | i5 |
| **RAM** | 4GB |
| **SDD** | 512GB |

## Software requirement

| | |
|---|---|
| **Operating System** | Windows,mac,linux etc |
| **Programming language** | javascript |
| **Other tools & technologies** | Vs code and replit |

## React js requirement

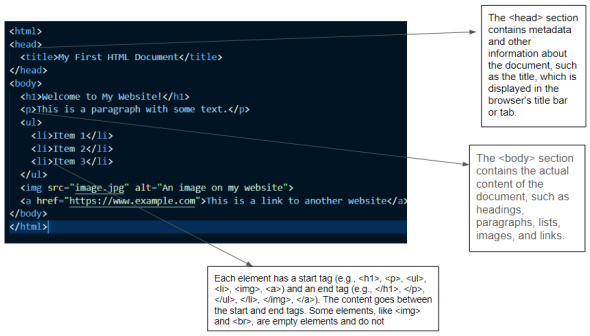
| | |
|---|---|
| **1** | html,css |
| **2** | javascript , TailwindCss |

# Chapter 4: Week 1

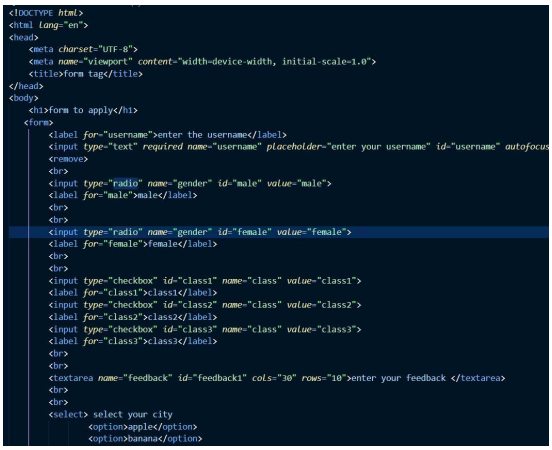
## 4.1 HTML

- ➢ The most common markup language for creating web pages is HTML (HyperText Markup Language).
- ➢ Content is organized and formatted using a tag system. Other technologies are typically utilized when describing the appearance or presentation of a web page (using CSS) or its functionality or behavior (using JavaScript)..

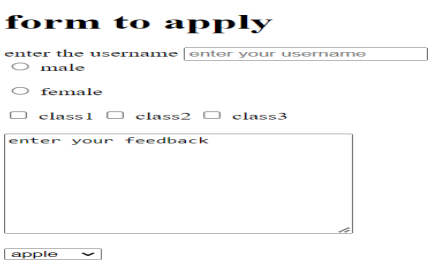
**Syntax**




**CODE**



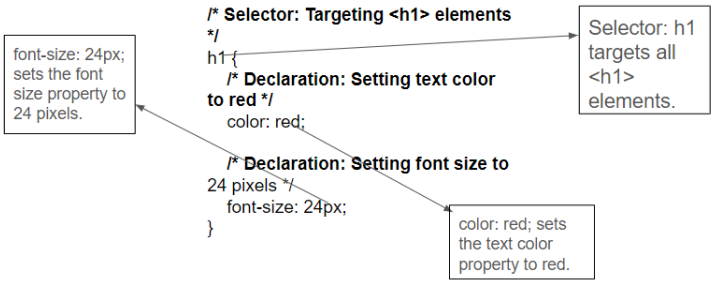
**Output**



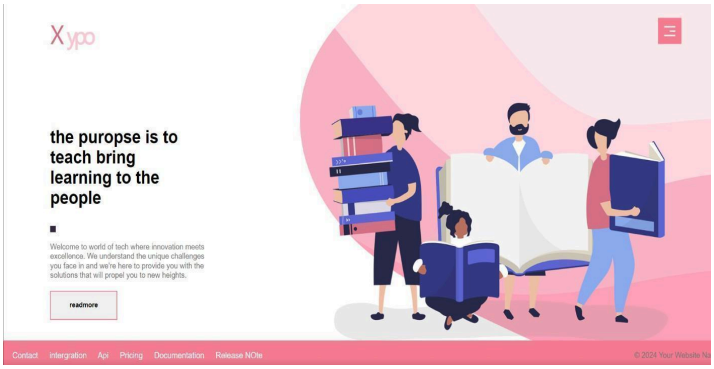
## 4.2 CSS

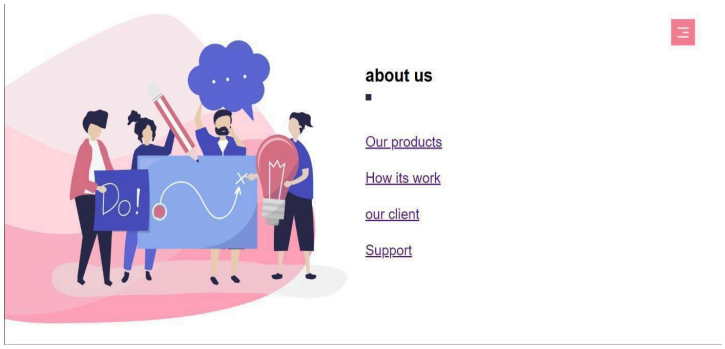
- ➢ CSS (Flowing Templates) is a template language utilized for portraying the look and designing of a record written in HTML or XML.
- ➢ It permits you to isolate the introduction of your record from its construction, making your code cleaner and more straightforward to keep up with.

**Syntax**




Output



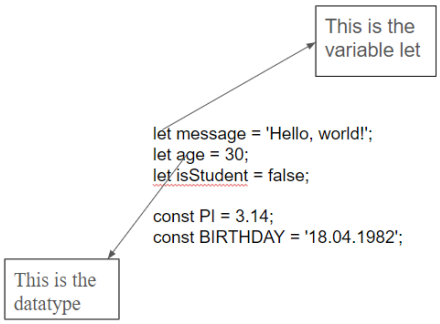



# Chapter 5: (Week 2)

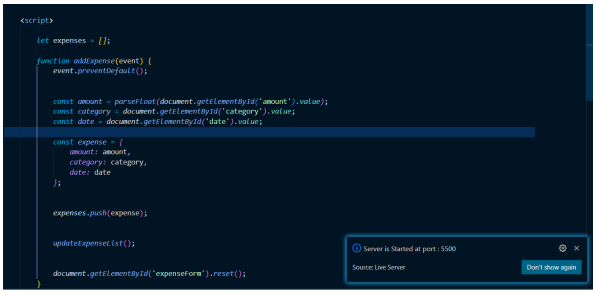
## 5.1 variable (let , const , var)

- ➢ A variable is a storage area (container) for data in programming. For instance, when declaring variables in JavaScript,
- ➢ A variable is a storage area (container) for data in programming. For instance, when declaring variables in JavaScript, we use either the var or the let keyword. Because var keywords are elevated (lifted) to the top of their scope, we did not use them. Because of this, a variable can be used before it has been declared.so for this scenarios we can used let , and const

**Syntax**




**Code**

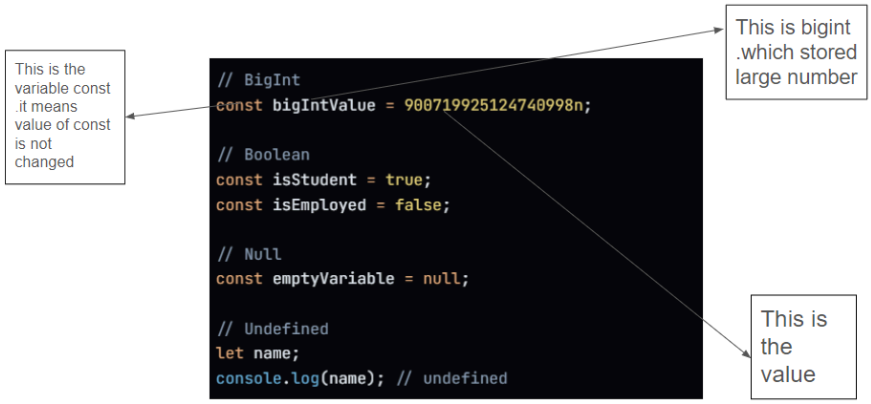



**Output**

## 5.2 Data Types()

- ➢ As a dynamically typed language, JavaScript supports a variety of data types.
- ➢ There are two sorts of information type in javascript There are two different types of data. non-primitive and primitive()

### Syntax




### Code

```
    <form id="expenseForm">
        <label for="amount">Amount:</label>
        <input type="number" id="amount" name="amount" required><br><br>

        <label for="category">Category:</label>
        <input type="text" id="category" name="category" required><br><br>

        <label for="date">Date:</label>
        <input type="date" id="date" name="date" required><br><br>

        <button type="submit">Add Expense</button>
    </form>

    <h2>Expenses:</h2>
    <ul id="expenseList"></ul>

    <script>

        // String
let message = "Hello, world!";

// Number
let age = 25;
let temperature = 98.6;

// Boolean
let isStudent = true;
let hasCar = false;

// Array
let fruits = ["apple", "banana", "orange"];

// Object
let person = {
```

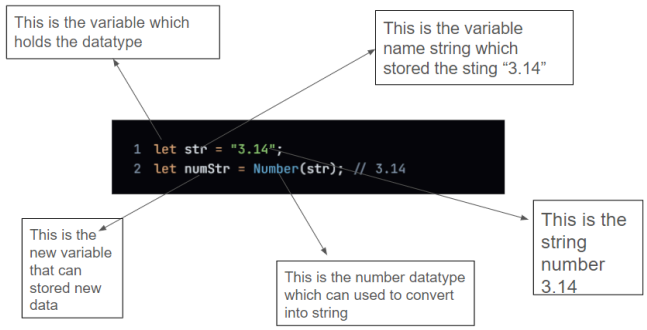
### Output

```
String: Hello, world!
Number: 25 98.6
Boolean: true false
Array:  ▶ Array(3)
Object:  ▶ Object
Function: 15
Undefined: undefined
Null: null
```

## 5.3 Data Types and conversion

- The process of converting data from one type to another is known as type conversion in programming. Changing data from a string to a number, for instance.
- There are two sorts of type transformation in JavaScript: Automatic type conversion is implicit conversion. Manual type conversion in explicit conversion.

**Syntax**




**Code**

```
// String to Number
let strNumber = "123";
let num = Number(strNumber);
console.log(num); // Output: 123

// Number to String
let number = 456;
let str = String(number);
console.log(str); // Output: "456"

// String to Boolean
let strBoolean = "true";
let bool = Boolean(strBoolean);
console.log(bool); // Output: true

// Boolean to Number
let boolean = true;
let numFromBool = Number(boolean);
console.log(numFromBool); // Output: 1 (true is converted to 1, false would be converted to 0)
```

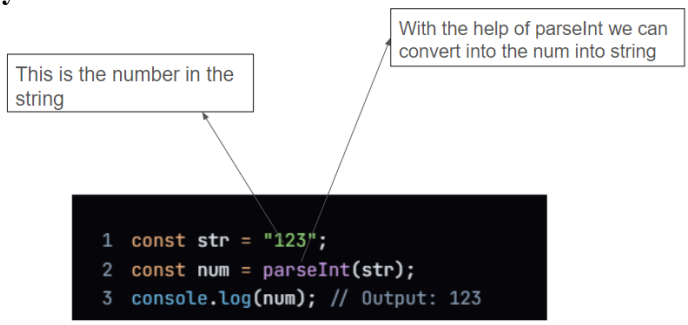
**Output**

```
123
456
true
1
false
10
10
10.5
>
```

## 5.4 Why string to number and operators used in javascript

- ➢ Switching a string over completely to a number in JavaScript is a typical activity . There are numerous reasons to convert a number to a string;
- ➢ one of these is Data manipulation: While numbers are required for mathematical operations, strings typically represent data in a format that is readable by humans.
- ➢ Switching strings over completely to numbers permits you to perform computations, correlations, and other numeric tasks.

### Syntax




### Code

```
<script>
    function performAddition() {
        let num1 = document.getElementById('num1').value;
        let num2 = document.getElementById('num2').value;

        // Convert strings to numbers
        num1 = parseFloat(num1);
        num2 = parseFloat(num2);

        let result = num1 + num2;
        document.getElementById('result').textContent = `Result: ${result}`;
    }

    function performSubtraction() {
        let num1 = document.getElementById('num1').value;
        let num2 = document.getElementById('num2').value;

        // Convert strings to numbers
        num1 = parseFloat(num1);
        num2 = parseFloat(num2);

        let result = num1 - num2;
        document.getElementById('result').textContent = `Result: ${result}`;
    }

    function performMultiplication() {
        let num1 = document.getElementById('num1').value;
        let num2 = document.getElementById('num2').value;
```

**Calculator**

Enter Number 1: 12 Enter Number 2: 23 Add Subtract Multiply Divide

**Result:**

Result: 0.5217391304347826

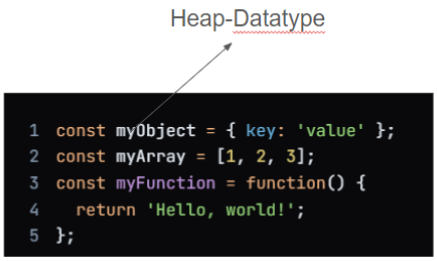
# Chapter 6: (Week 3)

## 6.1 stack (primitive) heap(non-primitive)

- ➢ The stack and the heap are the two main areas of a computer where data can be stored. The stack is a more modest, quicker memory region utilized for putting away basic information types and data about capability calls.
- ➢ It deals with a "rearward in, first out" premise, implying that the latest data added is quick to be eliminated. The pile is a bigger, more slow memory region utilized for putting away complex information types, like items and exhibits.
- ➢ It's more adaptable than the stack, however it additionally requires more work to make due. The software engineer (or on account of JavaScript, the programmed trash specialist) is liable for designating and deallocating memory in the load.
- ➢ In conclusion, simple, short-term data can be stored in the stack, while more complex, long-term data can be stored in the heap.

**Syntax(stack)**


**Syntax(heap)**



## 6.2 String

- ➢ a string in JavaScript is a grouping of characters used to address text.
- ➢ It is a crude information type, meaning a fundamental sort of information isn't an item and doesn't have strategies (in spite of the fact that JavaScript furnishes worked in techniques to work with strings).

### Syntax


### Code

```
Click here to ask Blackbox to help you code faster
const name = "John";
const age = 25;
const city = "New York";

const greeting = `Hello, my name is ${name}. I am ${age} years old and I live in ${city}.`;

console.log(greeting);
```

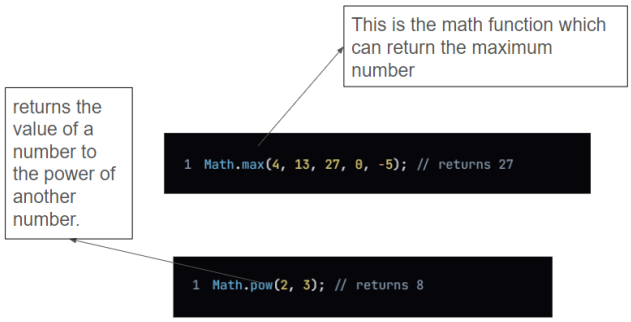
### Output

```
Hello, my name is John. I am 25 years old and I live in New York.
Live reload enabled.
>
```

## 6.3 Number and Math

- Numbers can be either floating-point numbers (decimals) or integers (whole numbers) in JavaScript.
- You can perform number-crunching activities like expansion, deduction, augmentation, and division on numbers. Here is a model:

Syntax



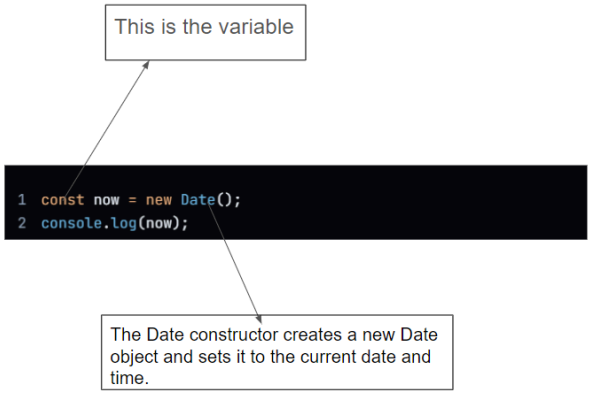

Code

```
Click here to ask Blackbox to help you code faster
function alternateCasing(str) {
    let result = '';
    for (let i = 0; i < str.length; i++) {
      if (i % 2 === 0) {
        result += str[i].toUpperCase();
      } else {
        result += str[i].toLowerCase();
      }
    }
    return result;
  }

  $('.str').on('keyup change', function() {
    const str = $(this).val();
    const result = alternateCasing(str);
    $('#alternated').html(result);
  });
```

## 6.4 Date and time

- When working with dates and times, JavaScript makes use of the Date object.
- It permits you to make, access, and control dates and times. This is an illustration of the way to make another Date object:
- A new Date object representing the current date and time will be created as a result of this. You can also pass in the year, month, day, hour, minute, second, and millisecond as arguments to create a Date object for a specific date and time:

**Syntax**




**Code**

```
let mydate = new Date()
console.log(mydate.toString()) //we can just print the data
console.log(mydate.toLocaleString()) //it can covert into the local string
console.log(mydate.toDateString());
console.log(mydate.toLocaleString());
console.log(typeof mydate); //this is the object data

//we also decalare specfic date
let mycreateddate = new Date("1-14-2023") //in javascript month start from 0 to 11 jan - 0  and end with 23
// console.log(mycreateddate.toDateString()); //it can gives with day like saturday // sat dec 23 2023
console.log(mycreateddate.toLocaleString()); //it can gives with standard date // 23/12/2023

// time stamp
let timestamp = Date.now()
console.log(timestamp);
console.log(mycreateddate.getTime()); //converted into seconds
console.log(Math.floor(Date.now()/100));


let newdate = new Date()
console.log(newdate);
console.log(newdate.getMonth()+1);
console.log(newdate.getDay());
console.log(newdate.getTime());
💡
// let newdate = new Date()
console.log(newdate.getHours());
console.log(newdate.getFullYear());
console.log(newdate.getDate()+1);
```

**Output**

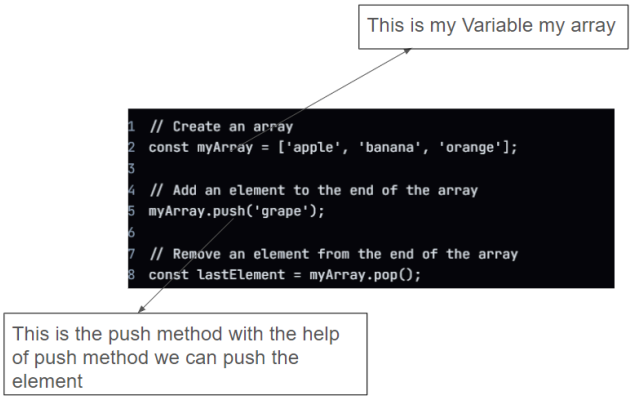
```
08/02/2024, 21:22:49
Thu Feb 08 2024
08/02/2024, 21:22:49
object
14/01/2023, 00:00:00
1707407569511
1673634600000
17074075695
Thu Feb 08 2024 21:22:49 GMT+0530 (India Standard Time)
2
4
1707407569512
21
2024
9
> |
```

# Chapter 7: (Week 4)

## 7.1 Array

- An array is a type of data structure that can hold multiple values in a single variable. These values can be of any type, such as numbers, strings, objects, or even other arrays. Arrays are useful for storing and working with sets of related data.
- There are many other methods available for working with arrays in JavaScript, including methods for sorting, filtering, and mapping. You can find more information in the JavaScript documentation.

**Syntax**




**Code**

```
<script>
//  arrays = array is the collection of same datatype and it is non-primitive datatypes
const array = [0,1,2,3,4,true] //array value start from 3
const myheros = ["spiderman","captionamerica","ironman","shaktiman","krish"]

//Array methods

// let myheros = ["spiderman","batman","superman","ironman","captionamerica"]
myheros.push("thor") //we can add at the end of the array
myheros.pop("thor","captionamerica") //pop is deleted from last no from first
myheros.unshift("krish") //unhift is like the push  method  but we can add the new element from first
myheros.shift("krish") //in shift we can remove the element from
console.log(myheros.includes("spiderman")); //by using include method it can shows the value is exsitece for not by using true or false
console.log(myheros.indexOf("spiderman")) //by using the index of it can shows the real index number
const arr = myheros.join() //by using join it can covert into string
console.log(myheros);
console.log(arr);

//slice or splice

let a = [1,2,3,4,5,6,7,8,9]
console.log(" A" , a);

const arr1 = a.slice(1,3) //The difference between slice and splice in JavaScript is that slice returns a new array with a portion
console.log(arr1);

console.log(" B" , a);

let arr2 = a.splice(1,3) //while splice modifies the original array by adding or removing elements in place.
console.log(arr2)
```

**Output**

```
true
0
▶ (5) ['spiderman', 'captionameric', 'ironman', 'shaktiman', 'krish']
spiderman,captionameric,ironman,shaktiman,krish
 A ▶ (9) [1, 2, 3, 4, 5, 6, 7, 8, 9]
▶ (2) [2, 3]
 B ▶ (9) [1, 2, 3, 4, 5, 6, 7, 8, 9]
▶ (3) [2, 3, 4]
>
```

## 7.2 Array Methods

- ➢ For working with arrays, which are lists of values that can be of any data type, JavaScript provides a number of built-in methods.
- ➢ Here are some regularly utilized exhibit techniques: push(): Adds at least one components to the furthest limit of a cluster. pop():
- ➢ Eliminates the last component from a cluster and brings it back. shift(): Eliminates the main component from a cluster and brings it back. unshift(): Adds at least one components to the start of a cluster.

### Code

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Arrays</h1>
<h2>The unshift() Method</h2>

<p>The unshift() method returns the length of the new array:</p>

<p id="demo1"></p>
<p id="demo2"></p>

<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits.unshift("Lemon");
document.getElementById("demo2").innerHTML = fruits;
</script>

</body>
</html>
```

# JavaScript Arrays

## The unshift() Method

The unshift() method returns the length of the new array:

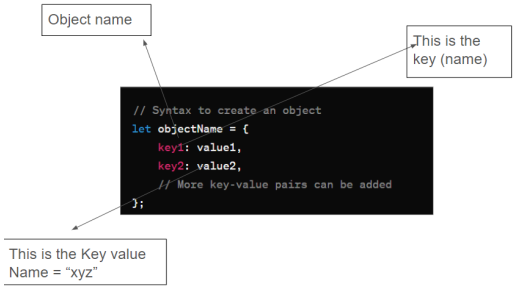
5

Lemon,Banana,Orange,Apple,Mango

## 7.3 Object

- A container for storing related data and functionality is an object. It resembles a case with named compartments, where each mark (or "key") relates to a piece of information (or "worth").
- The values can be any data type, including numbers, strings, arrays, other objects, functions, or even undefined, while the keys are always strings (or symbols). Objects are a principal information type in JavaScript and are utilized to address complex elements with properties and conduct.

**Syntax**




**Code**

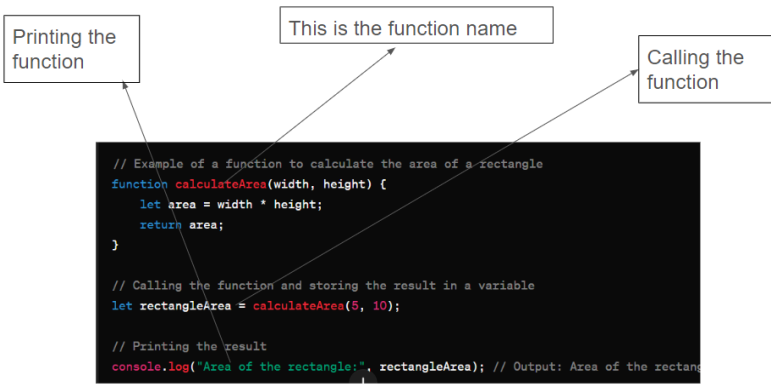
```
let person = {
    firstName: "John",
    lastName: "Doe",
    age: 30,
    email: "john@example.com",
    address: {
        street: "123 Main St",
        city: "Anytown",
        country: "USA"
    },
    // Method to display full name
    fullName: function() {
        return this.firstName + " " + this.lastName;
    }
};

// Accessing object properties
console.log(person.firstName);  // Output: John
console.log(person.age);        // Output: 30

// Accessing nested object properties
console.log(person.address.city); // Output: Anytown

// Calling object method
console.log(person.fullName());  // Output: John Doe
```

```
John
30
Anytown
John Doe
Live reload enabled.
```

## 7.4 Function

- A block of code known as a function can be defined and later executed.
- Capabilities are one of the basic structure blocks of JavaScript programming and are utilized to typify reusable bits of code, making your projects more secluded

**syntax**




**Code**

```
Click here to ask Blackbox to help you code faster
// Define a function to calculate total price
function calculateTotalPrice(cart) {
    let totalPrice = 0;

    // Iterate through each item in the cart
    for (let item of cart) {
        // Add the price of each item to the total price
        totalPrice += item.price * item.quantity;
    }

    // Return the total price
    return totalPrice;
}

// Example shopping cart with items
let shoppingCart = [
    { name: "Laptop", price: 1000, quantity: 1 },
    { name: "Headphones", price: 100, quantity: 2 },
    { name: "Mouse", price: 20, quantity: 3 }
];

// Call the function to calculate total price
let total = calculateTotalPrice(shoppingCart);

// Display the total price
console.log("Total price of items in the shopping cart:", total);
```

**Output**

```
Total price of items in the shopping cart: 1260
Live reload enabled.
```

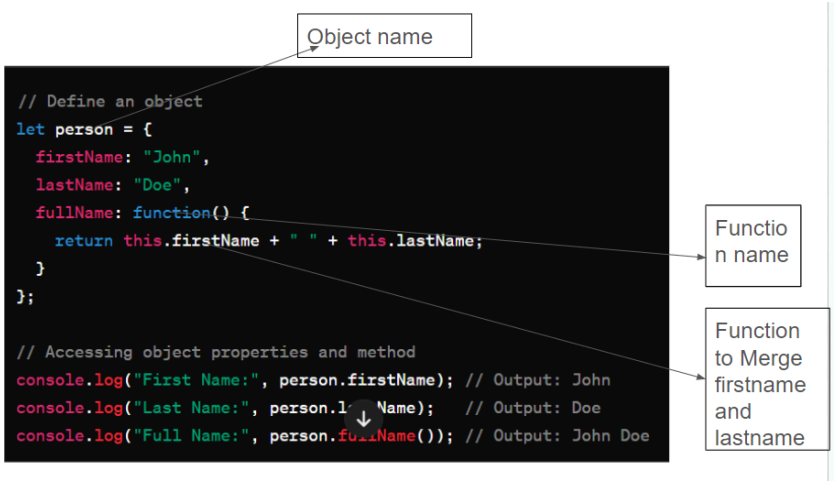
# CHAPTER NO 8 : (week5)

## 8.1 Function with object

➢ By passing the object to the function as a parameter, you can define a function in JavaScript that works on an object.

➢ A function that takes an object as a parameter and uses its properties to carry out some action is shown in the following example:

**Syntax**



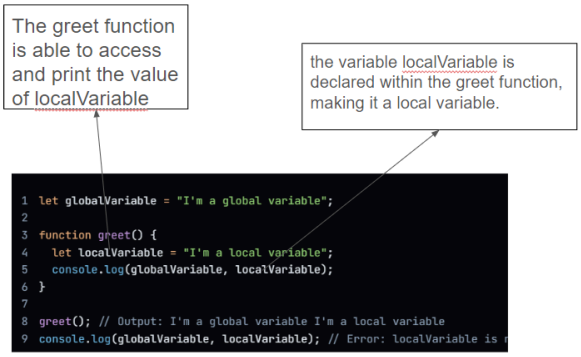

**Code**

```
function add(a,b) {
    return `${a+b} your soluction `
}
console.log((add(3,4)));

// function with objects
const name = {
    name:"anuj",
    fullname:"anuj jani",
}
name.name="wolf"
console.log(name);
                    //  rest operator or spread operator
function addcalculato (val1,val2,...num1){
     return num1
}
console.log(addcalculato(200,400,500,2000));

const user = {
    username:"anuj",
    price:34
}
function handleobject(anyobject){
   console.log(`username is ${anyobject.username} and it prices is ${anyobject.price}`);
}
handleobject(user)
```

## 8.2 Global scope and local scope

- ➢ Scope in JavaScript alludes to the perceivability and availability of factors in your code. To put it another way, it decides where a variable can be accessed and used. In JavaScript, scope is divided into two main categories:
- ➢ Global scope: Variables with global scope are declared outside of any function or block. Any section of the code can be used to access and modify them.
- ➢ Neighborhood scope: Factors pronounced within a capability or block have nearby degree. They can only be accessed and modified inside the declared function or block.

**Syntax**




**Code**

```
var globalVariable = "I am a global variable";

function foo() {
    console.log(globalVariable); // Accessible inside functions
}

console.log(globalVariable); // Accessible outside functions
```

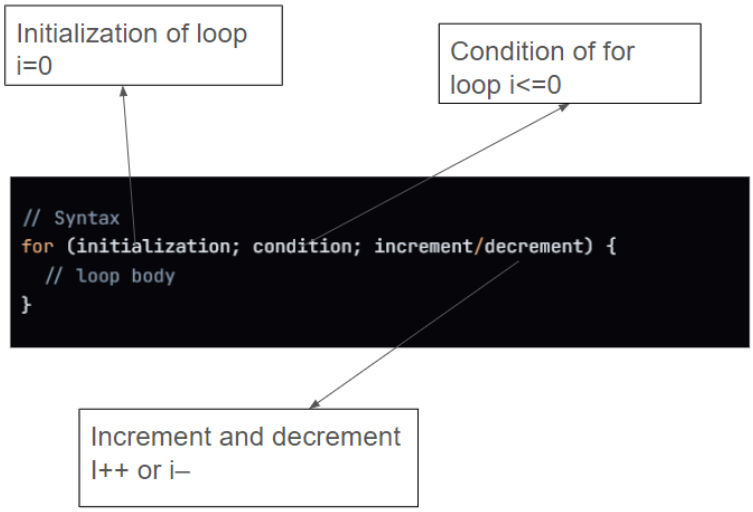
**Output**

```
I am a global variable
```

## 8.3 loops

- Loops are used in programming to repeat a block of code multiple times, as long as a certain condition is met. In JavaScript, there are several types of loops available, including for, while, and do...while loops.
- The initialization section is where you set the starting value for your loop counter. The condition section is where you specify the condition that must be true for the loop to continue executing. The increment/decrement section is where you update the loop counter after each iteration.

**Syntax**



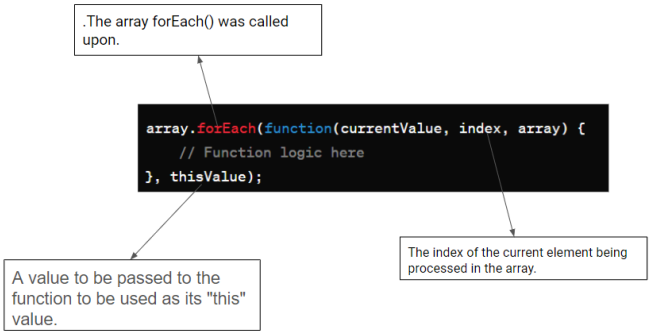

**Code**

```
// while (condition) {
//     // code block to be executed
// }
let i = 0;
do {
  console.log(i);
  i++;
} while (i < 5);
const person = {name: 'John', age: 30, city: 'New York'};
for (let key in person) {
  console.log(`${key}: ${person[key]}`);
}
const arr = [1, 2, 3, 4, 5];
for (let num of arr) {
  console.log(num);
}
```

## 8.4 Array loops (for each loop)

- Array loops are used to go through each element of an array and do something with it. There are several ways to loop through an array in JavaScript, including Traditional for loops: This is the most basic way to loop through an array.
- You can use a for loop to iterate over the array and access each element using its index. Here's an example:forEach() method: This is a method available on arrays that allows you to iterate over each element and perform an action on it. Here's an example:for...of loop: This is a newer way to loop through arrays in JavaScript.
- It allows you to iterate over the values of an array directly, without having to use an index. Here's an example:map() method: This is a method available on arrays that allows you to create a new array by applying a function to each element in the original array. Here's an example:

**Syntax**




**Code**

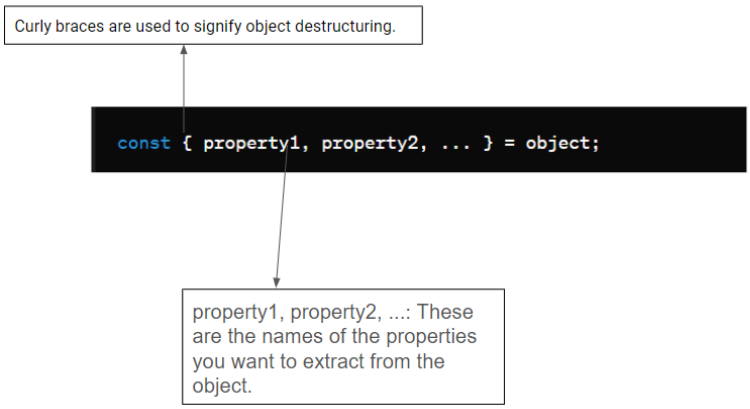
```
const array = [1, 2, 3, 4, 5];

for (let element of array) {
  console.log(element);
}
const array2 = [1, 2, 3, 4, 5];

const newArray = array.map(function(element) {
  return element * 2;
});

console.log(newArray); // Output: [2, 4, 6, 8, 10]
```

# CHAPTER 9 : (Week 6)

## 9.1 object destructuring

- ➢ Object destructuring is a way to extract values from an object and assign them to variables in a simpler and more readable way.
- ➢ It's a handy feature in JavaScript that can help you write cleaner code when working with objects.

### Syntax



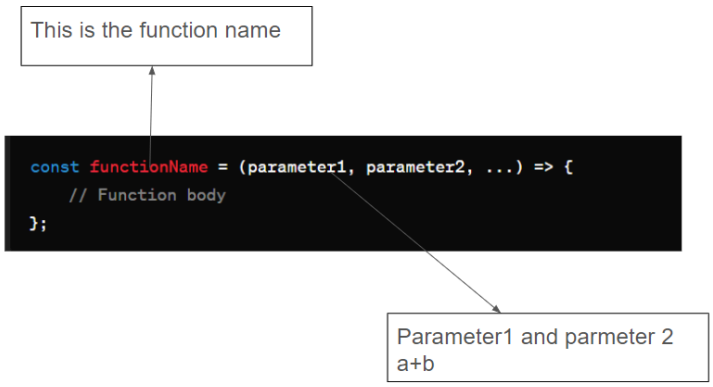

### Code

```
    <title>Document</title>
</head>
<body>
    <script>

const car = {
  make: 'Toyota',
  model: 'Corolla',
  year: 2020,
  features: {
    engine: '1.8L',
    transmission: 'Automatic',
    fuelType: 'Petrol'
  }
};

// Function using object destructuring
const printCarInfo = ({ make, model, year, features: { engine, transmission, fuelType } }) => {
  console.log(`Car: ${year} ${make} ${model}`);
  console.log(`Engine: ${engine}, Transmission: ${transmission}, Fuel Type: ${fuelType}`);
};

// Call the function with the car object
printCarInfo(car);
```

## 9.2 this with arrow function

- In JavaScript, the this keyword is used to refer to the current context or object that a function is called on. In other words, it refers to the object that "owns" the function. In regular functions,
- the value of this depends on how the function is called. If the function is called as a method of an object, this refers to that object. If the function is called as a standalone function, this refers to the global object (window in a browser or global in Node.js).

**Syntax**



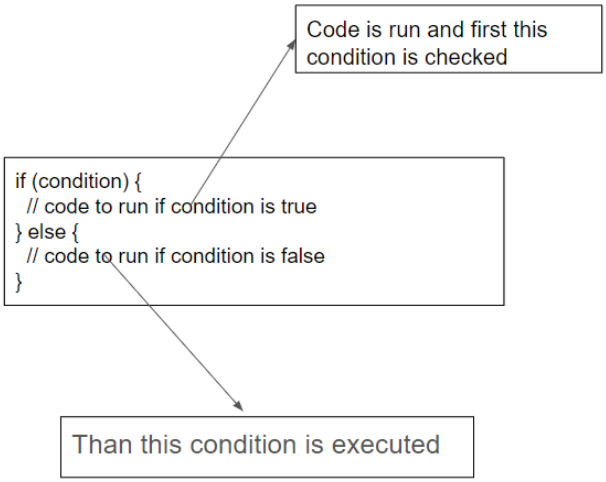

**Code**

```
class Animal {
  constructor(name) {
    this.speed = 0;
    this.name = name;
  }
  run(speed) {
    this.speed = speed;
    alert(`${this.name} runs with speed ${this.speed}.`);
  }
  stop() {
    this.speed = 0;
    alert(`${this.name} stands still.`);
  }
}

let animal = new Animal("My animal");
```

## 9.3 control flow

- ➢ Conditional statements are used in programming to make decisions based on certain conditions.
- ➢ They allow you to execute different blocks of code depending on whether a certain condition is true or false.
- ➢ The main difference between try and catch and if-else conditions is that try and catch are used for error handling, while if-else conditions are used for decision making.

**Syntax**



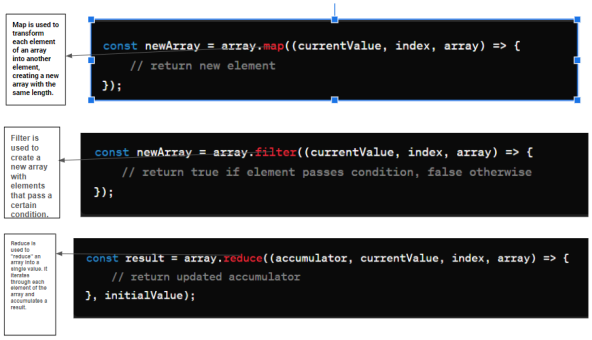

**Code**

```
let num = 5;

switch (num) {
    case 0:
        console.log("Number is zero.");
        break;
    case 1:
        console.log("Nuber is one.");
        break;
    case 2:
        console.log("Number is two.");
        break;
    default:
        console.log("Number is greater than 2.");
};
```

## 9.4 map reduce and filter

- ➢ Map, reduce, and filter are JavaScript array methods that can be used to transform or compute new arrays based on an existing one. Every strategy takes a capability as a contention and applies it to every component in the array.The map technique makes another exhibit with the consequences of calling a gave capability on each component in the cluster.
- ➢ The lessen strategy applies a capability against a gatherer and every component in the cluster (from left to right) to decrease it to a solitary result value.
- ➢ The channel technique makes another exhibit with all components that breeze through the assessment executed by the gave capability.

### Syntax




### Code

```
const numbers = [1, 2, 3, 4, 5];

// Using map to double each element, then filter to get only even numbers, and
    finally reduce to calculate their sum
const sumOfDoubledEvenNumbers = numbers
    .map(num => num * 2) // Double each element
    .filter(num => num % 2 === 0) // Filter only even numbers
    .reduce((accumulator, currentValue) => accumulator + currentValue, 0); //
        Calculate sum

console.log(sumOfDoubledEvenNumbers); // Output: 24 (2 * 2 + 4 * 2)
```

### Output

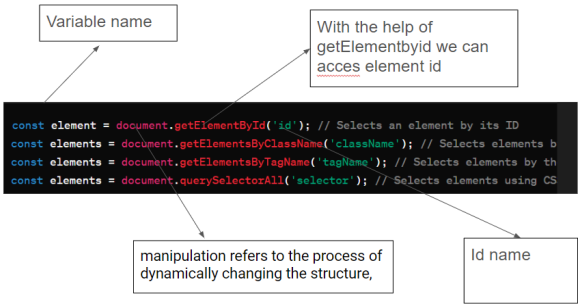
```
node /tmp/S9c3tUR42Q.js
30
```

# CHAPTER 10 : (Week 7)

## 10.1 Dom Manipulation

- ➢ When creating web pages and applications, it's common to modify the structure of the documents in some way.
- ➢ This is usually done using the Document Object Model (DOM), which is a set of APIs for controlling HTML and styling data using the Document object.
- ➢ The DOM represents the document as a tree of nodes, where each node corresponds to an element or a piece of text in the HTML. You can manipulate the DOM by selecting nodes, adding or removing nodes, and changing their attributes or content.

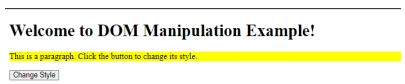
### Syntax




### Code

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>DOM Manipulation Example</title>
<style>
    .highlight {
        background-color: yellow;
    }
</style>
</head>
<body>
    <h1 id="mainHeading">Welcome to DOM Manipulation Example!</h1>
    <p>This is a paragraph. Click the button to change its style.</p>
    <button id="changeStyleButton">Change Style</button>

    <script>
        // Function to change the style of the paragraph
        function changeStyle() {
            var paragraph = document.querySelector('p');
            paragraph.classList.toggle('highlight');
        }

        // Add event listener to the button
        document.getElementById('changeStyleButton').addEventListener('click', changeStyle);
    </script>
</body>
</html>
```

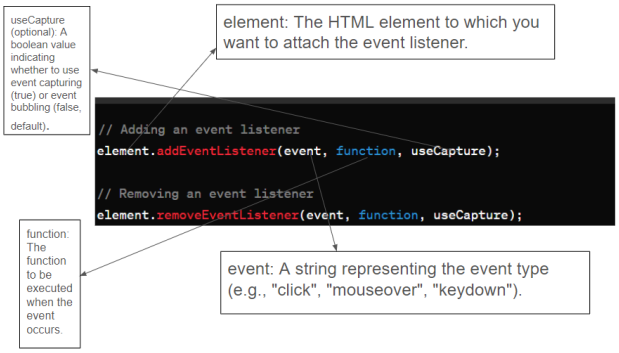
### Output



## 10.2 Events

- ➢ Events in JavaScript are actions or occurrences that happen in the browser, such as clicking a button or loading a page.
- ➢ You can write code that runs in response to these events, allowing you to create interactive and dynamic web pages. To use events in JavaScript, you first need to attach an event listener to an element by using element.addeventlistener

### Syntax




### Code

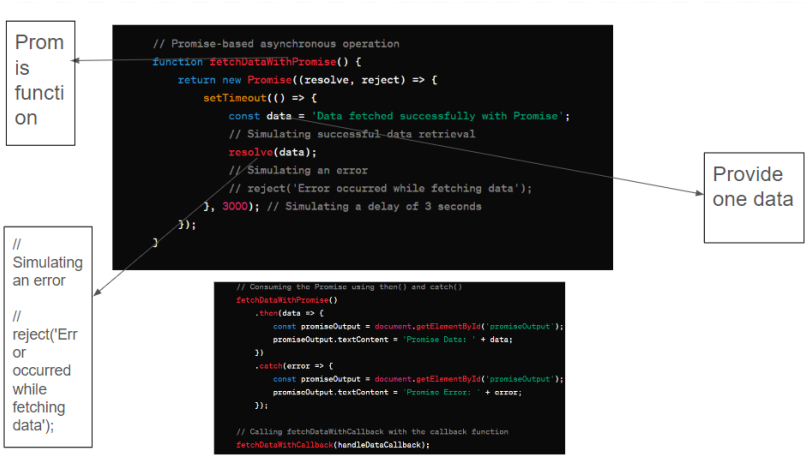
```
</head>
<body>
    <button id="myButton">Click Me!</button>
    <div id="output"></div>

    <script>
        // Get a reference to the button and output div
        var button = document.getElementById('myButton');
        var output = document.getElementById('output');

        // Function to be executed when the button is clicked
        function handleClick() {
            output.textContent = 'Button clicked!';
        }

        // Add an event listener to the button
        button.addEventListener('click', handleClick);

        // Function to be executed when the mouse enters the button
        function handleMouseEnter() {
            button.textContent = 'Mouse over!';
        }

        // Function to be executed when the mouse leaves the button
        function handleMouseLeave() {
            button.textContent = 'Click Me!';
        }

        // Add event listeners for mouse enter and leave events
        button.addEventListener('mouseenter', handleMouseEnter);
        button.addEventListener('mouseleave', handleMouseLeave);
    </script>
```

hey i am the box
change content

## 10.3 Callbacks and Promises

- ➢ Promises are utilized in JavaScript to manage asynchronous actions. It is an affirmation that something will be finished.
- ➢ The commitment is utilized to monitor regardless of whether the nonconcurrent occasion has been executed and figures out what occurs after the occasion has happened.
- ➢ A promise is like a pledge or a guarantee that something will be done in the future.

### Syntax




### Code

```
<script>
    function handleDataCallback(error, data) {
        const callbackOutput = document.getElementById('callbackOutput');
        if (error) {
            callbackOutput.textContent = 'Callback Error: ' + error;
        } else {
            callbackOutput.textContent = 'Callback Data: ' + data;
        }
    }

    // Promise-based asynchronous operation
    function fetchDataWithPromise() {
        return new Promise((resolve, reject) => {
            setTimeout(() => {
                const data = 'Data fetched successfully with Promise';
                // Simulating successful data retrieval
                resolve(data);
                // Simulating an error
                // reject('Error occurred while fetching data');
            }, 3000); // Simulating a delay of 3 seconds
        });
    }

    // Consuming the Promise using then() and catch()
    fetchDataWithPromise()
        .then(data => {
            const promiseOutput = document.getElementById('promiseOutput');
            promiseOutput.textContent = 'Promise Data: ' + data;
        })
        .catch(error => {
            const promiseOutput = document.getElementById('promiseOutput');
            promiseOutput.textContent = 'Promise Error: ' + error;
        });
```

**Callback and Promise Example**

Callback Data: Data fetched successfully with callback
Promise Data: Data fetched successfully with Promise

## 10.4 Aysic and fetch api

- ➢ In JavaScript, promises are used to handle asynchronous operations, such as making API requests or reading files.
- ➢ To work with promises more easily, we can use the async and await keywords. The async keyword is used before a function to indicate that it returns a promise.
- ➢ This means that we can use the await keyword inside the function to pause its execution and wait for a promise to resolve before continuing.

### Syntax


### Code

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Async/Await with Fetch API Example</title>
</head>
<body>
    <h2>Async/Await with Fetch API Example</h2>
    <div id="output"></div>

    <script>
        async function fetchData() {
            try {
                const response = await fetch('https://jsonplaceholder.typicode.com/posts/1');
                if (!response.ok) {
                    throw new Error('Network response was not ok');
                }
                const data = await response.json();
                return data;
            } catch (error) {
                throw new Error('Error fetching data: ' + error.message);
            }
```

**Async/Await with Fetch API Example**

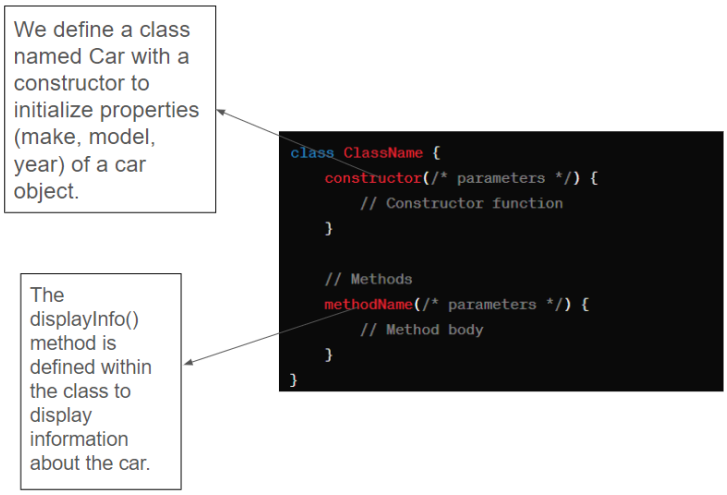
{"userId":1,"id":1,"title":"sunt aut facere repellat provident occaecati excepturi optio reprehenderit","body":"qu

# Chapter 11 : (week8)

## 11.1 Classes And Object

- ➢ In programming, a class is like a blueprint or a plan that defines the characteristics and features of an object. Think of it like a architectural design for a house, which outlines the details of doors, windows, floors, and more. Just as one house plan can be used to build multiple houses, a single class can be used to create multiple objects.
- ➢ The class itself isn't a real entity, but rather a template or a mold that shapes the objects created from it.
- ➢ This concept was introduced in ES6, replacing the traditional use of functions, and allows developers to create objects with specific properties and behaviors, making it a powerful tool in programming.

### Syntax



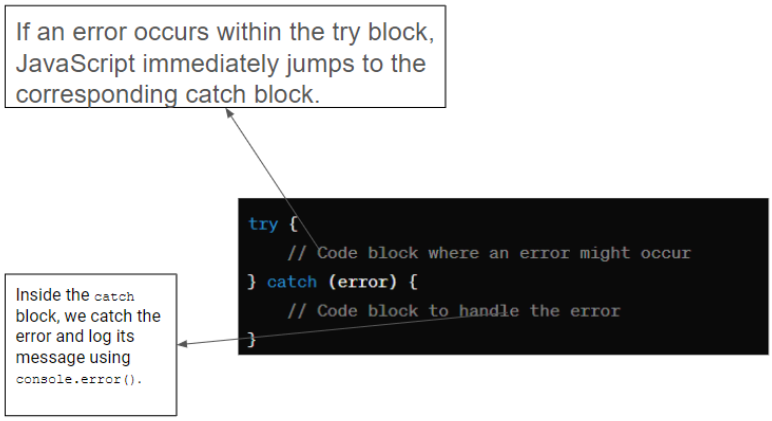

### Code

```
class OOPs {
    constructor(name) {
        this.name = name;
    }

    // Getter method
    get langName() {
        return this.name;
    }

    // Setter method
    set langName(x) {
        this.name = x;
    }
    hello(){
        console.log(`Hello ${this.name}`)
    }
}

let obj = new OOPs('JavaScript');
console.log(obj.name);

obj.langName = 'Java';
console.log(obj.name);
```

## 11.2 try and catch

- ➢ When coding in JavaScript, errors can happen, but we can use the try...catch...finally combo to handle them without stopping the script.
- ➢ The try part is like a test zone where we put the code that might cause an error. If an error occurs, the catch part kicks in to fix the issue or provide a backup plan.
- ➢ And finally, the finally part runs regardless of whether an error happened or not, like a cleanup crew. It's important to note that try...catch is for handling unexpected runtime errors, whereas if...else statements are for controlling the program's flow based on specific conditions - two different tools for different jobs!

**Syntax**



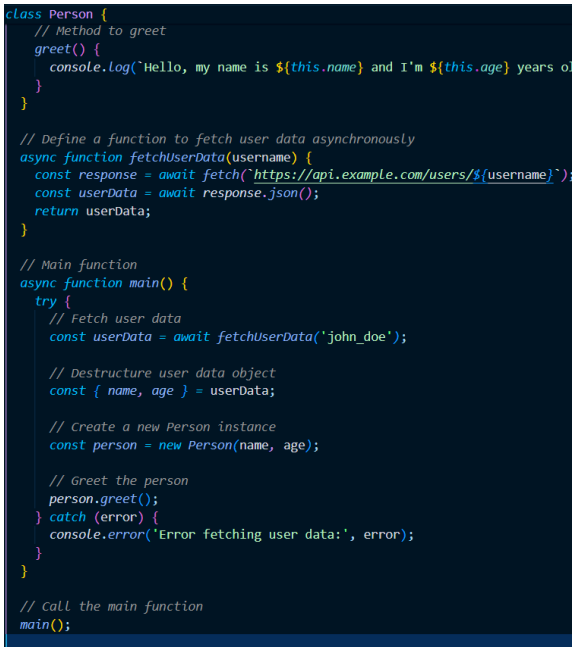

**Code**

```
<p id="demo"></p>

<script>
try {
  adddlert("Welcome guest!");
}
catch(err) {
  document.getElementById("demo").innerHTML = err.message;
}
</script>
```

## 11.3 Advances javascript

- ➢ Get a firm grasp of closures and scope in JavaScript, from the fundamentals to advanced methods. This guide will help you understand how they operate and how to apply them to create more efficient and maintainable code. Additionally, learn about the latest features of ES6 in JavaScript and what the future holds for this programming language. Explore real-world code examples and discover the advantages of using the latest version. Stay ahead of the curve with this informative article.
- ➢ Closures and scope are essential concepts in JavaScript. A closure is a function that has access to variables in its parent scope, even after the parent function has finished executing. This feature allows for powerful programming techniques, such as creating private variables and encapsulating functionality. Scope, on the other hand, refers to the visibility of variables in different parts of the code. Understanding scope is crucial for avoiding naming conflicts and ensuring that variables are accessible when and where they are needed.

**Code**

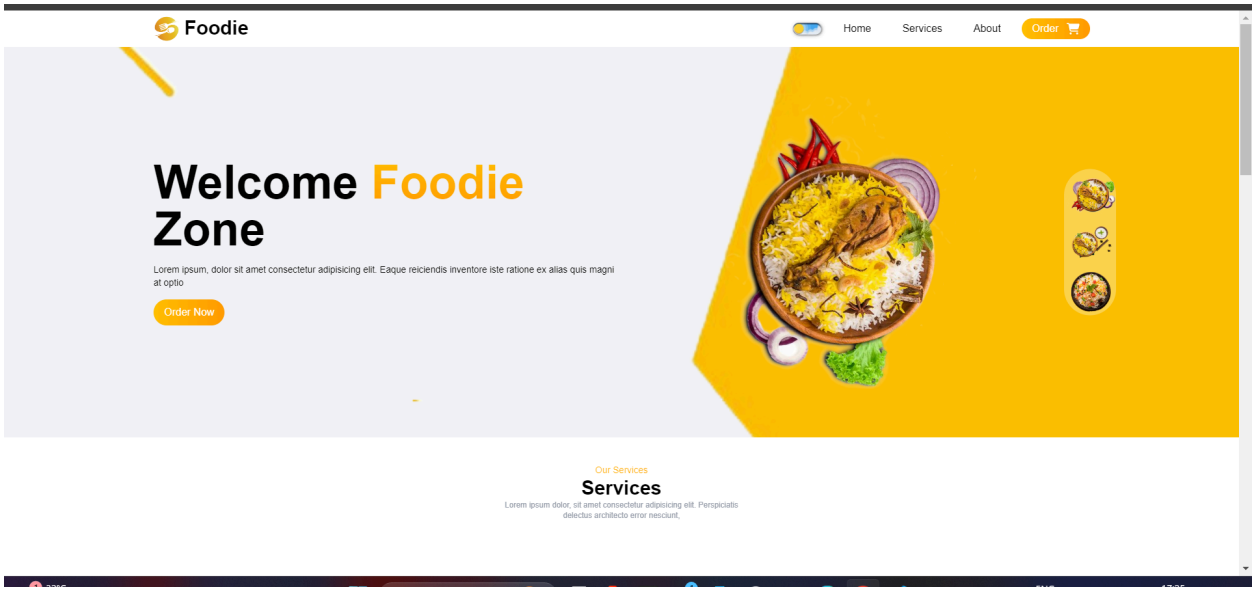


```
class Person {
    // Method to greet
    greet() {
      console.log(`Hello, my name is ${this.name} and I'm ${this.age} years o
    }
  }

  // Define a function to fetch user data asynchronously
  async function fetchUserData(username) {
    const response = await fetch(`https://api.example.com/users/${username}`);
    const userData = await response.json();
    return userData;
  }

  // Main function
  async function main() {
    try {
      // Fetch user data
      const userData = await fetchUserData('john_doe');

      // Destructure user data object
      const { name, age } = userData;

      // Create a new Person instance
      const person = new Person(name, age);

      // Greet the person
      person.greet();
    } catch (error) {
      console.error('Error fetching user data:', error);
    }
  }

  // Call the main function
  main();
```

## Foodie-website

Tools = vs code , Rapid Api , Freepik , free icons

Technology = Javascript , react , html , Tailwind Css



# CHAPTER NO 12 : (week 9)

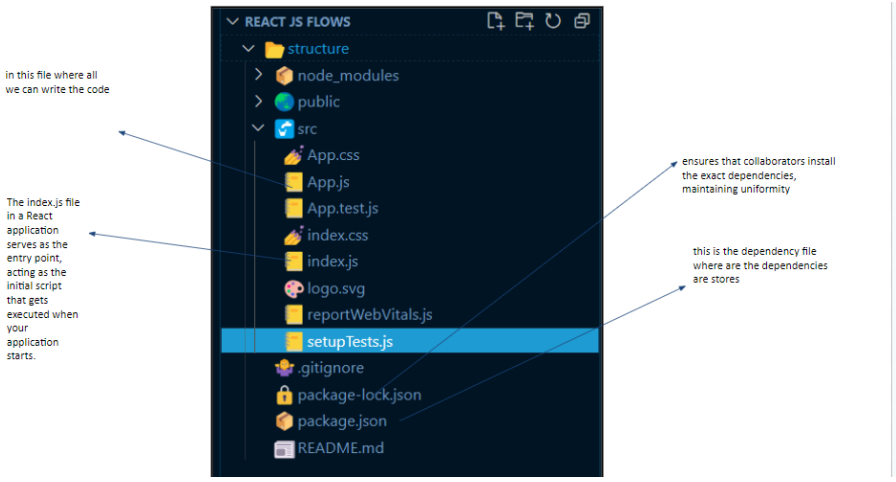
## 12.1 Introduction To The React js

- ReactJS is a JavaScript library for creating user interfaces that's frequently updated with new features. The latest stable version, 18.2.0, was released in June 2022, with the first version coming out in May 2013. ReactJS is well-known for its speed, flexibility, and strong community support.
- What makes ReactJS stand out is its component-based structure. This means that complex code is broken down into smaller, individual pieces called components, making it easier for developers to organize and manage their code.
- Many companies are adopting ReactJS, making it a valuable skill for both beginners and experienced developers to learn. With its frequent updates, ReactJS continues to be a popular choice for building user interfaces, offering new features and improvements to help developers create better, more efficient code. In simple terms, ReactJS is a tool for building user interfaces in JavaScript. It's easy to use, flexible, and has a large community of developers who can help if you get stuck. By breaking down code into smaller components, ReactJS makes it easier to manage and maintain, making it a popular choice for building modern user interfaces.

```
<!--what is the react js for
=>ReactJS is updated so frequently, that it is quite difficult to navigate the version,
which comes with new features every time, each time it comes with new features.
The current stable version of ReactJS is 18.2.0
and released on June 14, 2022 and the
first release was on May 29, 2013.

->
```

## 12.2 React js flows and structure

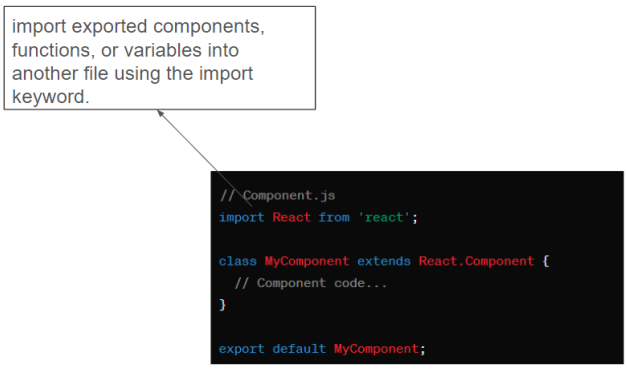
- ➢ The structure of a ReactJS project is organized into an "src" directory that contains the source code for the app. Within this directory, there are several folders that hold different types of code:
- ➢ The "actions" and "reducers" folders contain Redux-specific code, which is used for managing the state of the app.
- ➢ The "components" folder contains the individual React components that make up the app.
- ➢ The "styles" folder holds the CSS styles for the app.
- ➢ The "utils" folder contains utility functions that can be used throughout the app.
- ➢ By organizing the code in this way, it's easier to manage and maintain, especially as the app grows in size and complexity. Each folder has a specific purpose, making it easy to find and modify the code as needed.



## 12.3 Import and export the components in react js

- ➢ In React, you can share components with other parts of your code using the export keyword.Named Exports:
- ➢ You can give a name to the component and then use the export keyword before its declaration. This way, you can import the component by its name in other parts of your code. Named exports are useful when you want to export multiple components from the same file, as you can give each component a unique name and import them individually.

**Syntax**




**Code**

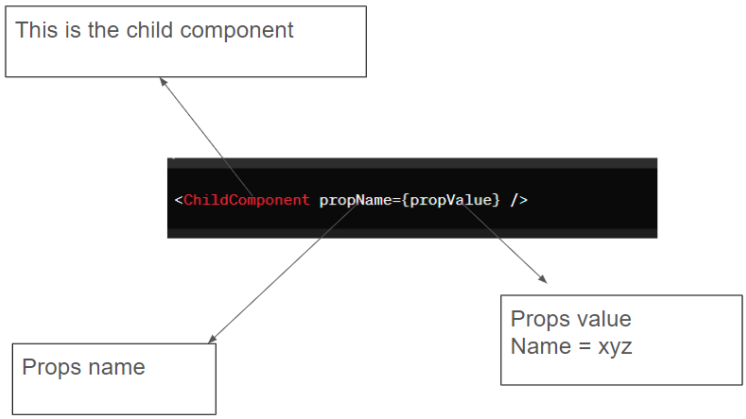
```jsx
import React from 'react'

const Footer = () => {
  return (
    <footer>
        <p>&copy; {new Date().getFullYear()} your website is safe </p>

    </footer>
  )
}

export default Footer
```

```jsx
import React from 'react'

const Header = () => {
  return (
    <header>
      <ul>

        <li><a href="#">Home</a></li>
        <li><a href="#">contact</a></li>
        <li><a href="#">People</a></li>
        <li><a href="#">Contact me</a></li>
      </ul>
    </header>
  )
}

export default Header
```

## 12.4 Props

- ➢ React, "props" (short for properties) are a way of passing data from parent components to child components.
- ➢ they are read-only and immutable, meaning that the child components cannot modify the props they receive.
- ➢ Props are used to customize and configure child components, allowing for dynamic and reusable UI elements.

**Syntax**




**Program**

```
Click here to ask Blackbox to help you code faster
import React from 'react'

const UserName = (props) => {
  return (
    <div>
        <h1>{props.name}</h1>
        <h1>{props.mailid}</h1>
        <h1>{props.passwordS}</h1>
        <h1></h1>
    </div>
  )
}

export default UserName
```

```
Click here to ask Blackbox to help you code faster
import React from 'react'
import UserName from './Components/UserName'

function App() {
  return (
    <div>
      <UserName name="anuj"  mailid = "XXyz@gmail.com" password = "12345" />

    </div>
  )
}

export default App
```

# CHAPTER 13 (week10)

## 13.1 Conditional Rendering

- ➢ React makes it possible to render different components on the screen based on certain conditions. This is called conditional rendering
- ➢ .For example, you can show a message to the user if they haven't entered any data, or you can show a loading spinner while data is being fetched from a server. Conditional rendering is a powerful feature that allows you to create dynamic user interfaces that adapt to the user's actions and input.
- ➢ To implement conditional rendering in React, you can use JavaScript's conditional statements, such as if and ternary operators, inside the render method of a component. This way, you can check if a certain condition is true or false, and render a component accordingly.

**Code**

```
Click here to ask Blackbox to help you code faster
import PropTypes from 'prop-types'

function Greeting(props){


    const welcomemessage =  <h2 className="true">WELCOME {props.username}
                            </h2>

     const welcomemessage2 = <h3 className="false">pleased logged in </h3>

    return (props.isloogedin ? welcomemessage : welcomemessage2);
}

Greeting.prototype = {
    isloogedin:PropTypes.bool,
    username:PropTypes.string,
}
Greeting.defaultProps = {
    isloogedin:false,
    username:"guest",
}
export default Greeting;
```

## 13.2 Rendering

- In React JS, the render method is a crucial part of class components. It is used to display the component on the user interface (UI) as HTML or JSX code.

- The render method returns a single parent element that contains all the other child elements that make up the component. This parent element can be a div, section, article, or any other HTML element.

- To display the component on the UI, you can use the ReactDOM.render() function, which takes two arguments: the HTML code returned by the render method, and an HTML element where the component should be inserted.

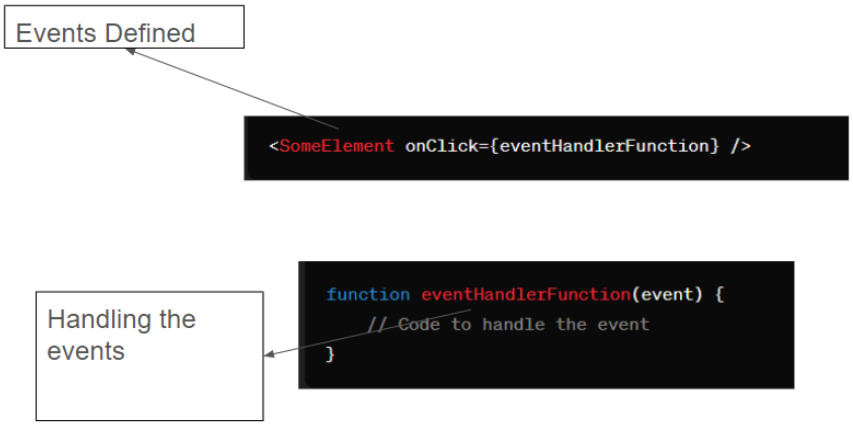
### Code

```
Click here to ask Blackbox to help you code faster
import React, { useState } from 'react';

const List = () => {
  const [fruits, setFruits] = useState([
    { id: 1, name: "apple", calories: 95 },
    { id: 2, name: "orange", calories: 45 },
    { id: 3, name: "banana", calories: 105 },
    { id: 4, name: "coconut", calories: 159 },
    { id: 5, name: "pineapple", calories: 37 },
  ]);

  const listItems = fruits.map(fruit => (
    <li key={fruit.id}>
      {fruit.name}:  
      <b>{fruit.calories}</b>
    </li>
  ));

  return <ol>{listItems}</ol>;
};

export default List;
```

## 13.3 events

- ➢ In React, an "event" is when something happens on the user interface, like clicking a button or submitting a form. React supports many of the same events as HTML, and you can use them in a similar way. However, instead of using HTML attributes to attach event listeners,
- ➢ in React you use JavaScript. To do this, you add an event listener function to the JSX element using the "on" prefix, like "onClick" for a click event or "onSubmit" for a form submit event.

### Syntax

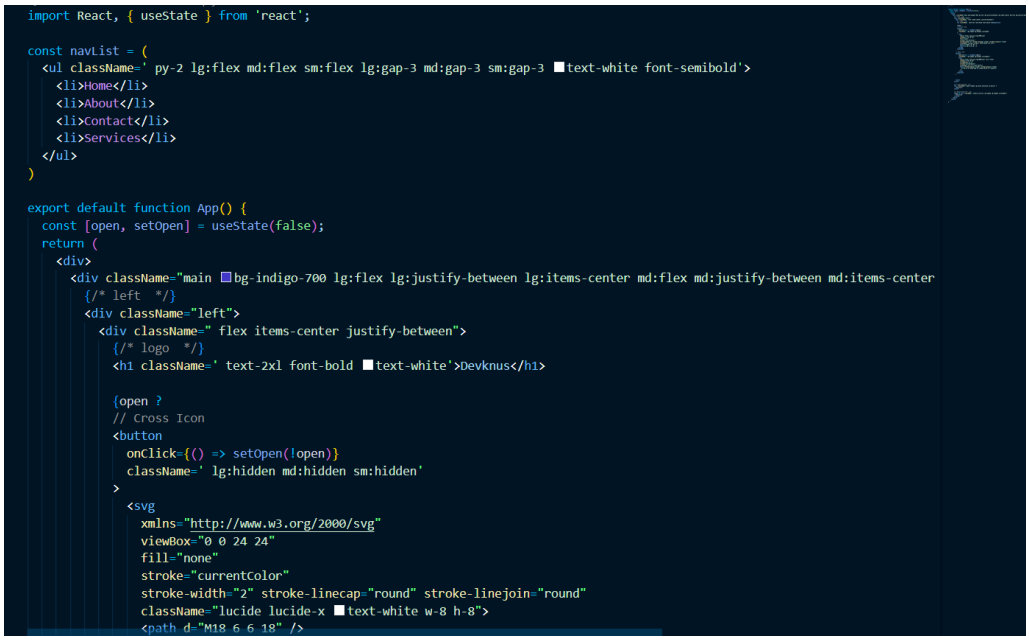



### Code

```
import React from 'react';

class MyButton extends React.Component {
  constructor(props) {
    super(props);
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    console.log('Button clicked!');
  }

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

## 13.4 Understanding About Tailwind Css

- Tailwind CSS is a tool for quickly building user interfaces (UI) with custom styles. It's a "utility-first" framework, which means it provides a lot of small, reusable CSS classes that you can use to style your HTML elements.
- With Tailwind CSS, you don't have to write your own CSS code. Instead, you can use the pre-built classes to style your elements directly in your HTML. For example, you can use the "text-red-500" class to make some text red, or the "bg-gray-200" class to make a background gray.
- You can change the colors, fonts, and other settings to match your brand or design. This makes it a great choice for building custom UIs quickly and efficiently. Overall, Tailwind CSS is a powerful tool for building UIs with inline styling. It's a low-level framework that gives you all the building blocks you need to create a unique and modern interface, without having to write a lot of custom CSS code.



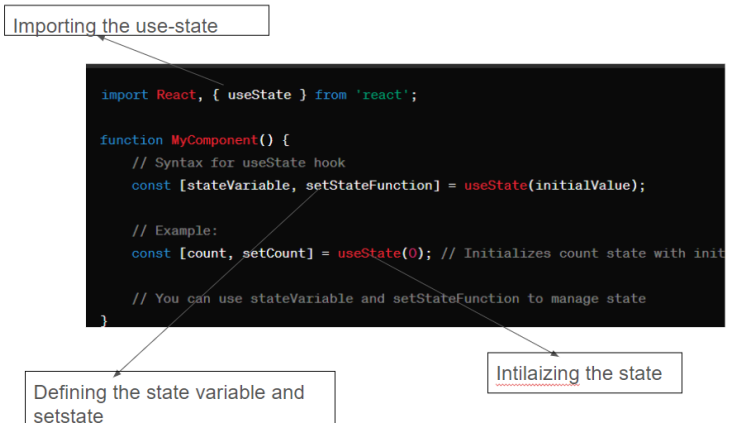
```
import React, { useState } from 'react';

const navList = (
  <ul className=' py-2 lg:flex md:flex sm:flex lg:gap-3 md:gap-3 sm:gap-3 text-white font-semibold'>
    <li>Home</li>
    <li>About</li>
    <li>Contact</li>
    <li>Services</li>
  </ul>
)

export default function App() {
  const [open, setOpen] = useState(false);
  return (
    <div>
      <div className="main bg-indigo-700 lg:flex lg:justify-between lg:items-center md:flex md:justify-between md:items-center
        {/* left  */}
        <div className="left">
          <div className=" flex items-center justify-between">
            {/* logo  */}
            <h1 className=' text-2xl font-bold text-white'>Devknus</h1>

            {open ?
            // Cross Icon
            <button
              onClick={() => setOpen(!open)}
              className=' lg:hidden md:hidden sm:hidden'
            >
              <svg
                xmlns="http://www.w3.org/2000/svg"
                viewBox="0 0 24 24"
                fill="none"
                stroke="currentColor"
                stroke-width="2" stroke-linecap="round" stroke-linejoin="round"
                className="lucide lucide-x text-white w-8 h-8">
                <path d="M18 6 6 18" />
```

# CHAPTER 14 (week11)

## 14.1 States and hooks

- In React.js, "state" is the data that describes the current condition of a component. Components in React.js can have state, which means they can store and manage their own data. State is mutable, which means it can change over time.
- This usually happens because of user interactions, network responses, or other events. When state changes, React.js automatically updates the component to reflect the new data. Before React.js version 16.8,
- only class components could have state. But with the introduction of the useState hook, you can now add state to functional components as well. The useState hook allows you to declare state variables in your functional components. For example, you can declare a state variable called count and set its initial value to 0. Then, you can use the setState function to update the count variable whenever the user clicks a button.

### Syntax




### Code

```
Click here to ask Blackbox to help you code faster
import React, { useState } from 'react'

function App() {

  const[count , setcount] = useState(0)
  return (
    <div>
        <h1>this is {count}</h1>
        <button onClick={()=>{setcount(count+1)}}>update the count</button>
    </div>
  )
}

export default App;
```

## 14.2 UseEffect

- ➢ useEffect is a special function in React that helps you connect your component to things outside of React, like a database or a web API. It lets you run some code after the component has rendered, and you can choose when to run it again based on certain conditions.
- ➢ useEffect takes two arguments: a function to run, and an array of things to watch. The function runs after every render, but if you include an array of dependencies, it will only run again if one of those things changes
- ➢ . This is useful for things like fetching data from a server or updating the local storage. Speaking of local storage, React has a similar feature called useLocalStorage, which works just like localStorage but is integrated with React. It lets you store data in the browser's local storage, which persists even after the user closes the browser. This is different from sessionStorage, which only lasts for the current session.

### Code

```
import { useState, useEffect } from "react";
import ReactDOM from "react-dom/client";

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

  useEffect(() => {
    setCalculation(() => count * 2);
  }, [count]); // <- add the count variable here

  return (
    <>
      <p>Count: {count}</p>
      <button onClick={() => setCount((c) => c + 1)}>+</button>
      <p>Calculation: {calculation}</p>
    </>
  );
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Counter />);
```

### Output

Count: 4

+

Calculation: 8

## 14.3 working in the Nav bar

Navbar.jsx

src > components > Navbar > Navbar.jsx > Menu

Click here to ask Blackbox to help you code faster

```jsx
import React from "react";
import Logo from "../../assets/food-logo.png";
import { FaCartShopping } from "react-icons/fa6";
import DarkMode from "./DarkMode";

const Menu = [
  {
    id: 1,
    name: "Home",
    link: "/#",
  },
  {
    id: 2,
    name: "Services",
    link: "/#services",
  },
  {
    id: 3,
    name: "About",
    link: "/#about",
  },
];
const Navbar = () => {
  return (
    <>
      <div className="shadow-md bg-white dark:bg-gray-900 dark:text-white duration-200">
        <div className="container py-3 sm:py-0">
          <div className="flex justify-between items-center">
            <div>
              <a href="#" className="font-bold text-2xl sm:text-3xl flex gap-2">
                <img src={Logo} alt="Logo" className="w-10" />
                Foodie
              </a>
            </div>
            <div className="flex justify-between items-center gap-4">
              <div>
```

## 14.4 working on the Dark mode

➢ Learning how to make darkmode in my website

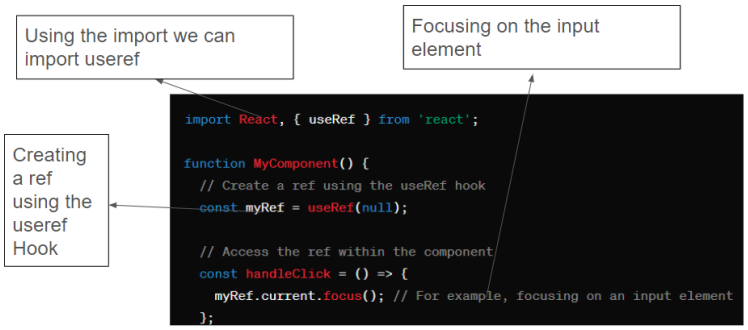
```
<>
  <div className="relative ">
    <img
      // src={theme === "dark" ? darkPng : lightPng}
      src={lightPng}
      alt="dark"
      onClick={() =>
        setTheme((data) => (data === "dark" ? "light" : "dark"))
      }
      className={`w-12 cursor-pointer drop-shadow-[1px_1px_1px_rgba(0,0,0,0.1)] transition-all duration-300 a
        theme === "dark" ? "opacity-0" : "opacity-100"
      } `}
    />
    <img
      src={darkPng}
      alt="dark"
      onClick={() =>
        setTheme((data) => (data === "dark" ? "light" : "dark"))
      }
      className="w-12 cursor-pointer drop-shadow-[1px_1px_2px_rgba(0,0,0,0.5)] duration-300 "
    />
  </div>
</>
);
```

# CHAPTER 15 (week12)

## 15.1 use ref

- ➢ In React, you can use "refs" to access and manipulate DOM elements or React components directly.
- ➢ This is useful when you need to interact with a specific element, like focusing an input field or triggering an animation. There are two ways to create a ref in React: using React.createRef() or the useRef() hook with a DOM element:

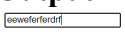
### Syntax




### Code

```
import { useState, useEffect, useRef } from "react";
import ReactDOM from "react-dom/client";

function App() {
  const [inputValue, setInputValue] = useState("");
  const count = useRef(0);

  useEffect(() => {
    count.current = count.current + 1;
  });

  return (
    <>
      <input
        type="text"
        value={inputValue}
        onChange={(e) => setInputValue(e.target.value)}
      />
      <h1>Render Count: {count.current}</h1>
    </>
  );
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<App />);
```

### Output

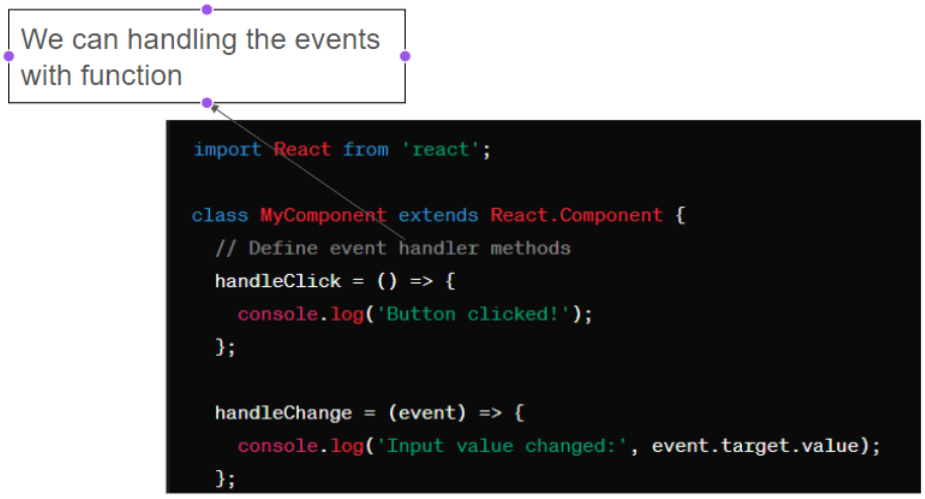



**Render Count: 13**

## 15.2 handling the events in react

- ➢ In React, when you want to respond to user interactions like clicks or keyboard input, you use "event handlers".
- ➢ These are functions that you define in your React component, and then attach to specific elements in your JSX code. React provides its own set of "synthetic" event listeners, which work just like regular DOM events, but with some added benefits.
- ➢ For example, synthetic events are cross-browser compatible, and they only trigger one event instead of multiple events for the same interaction

**Syntax**

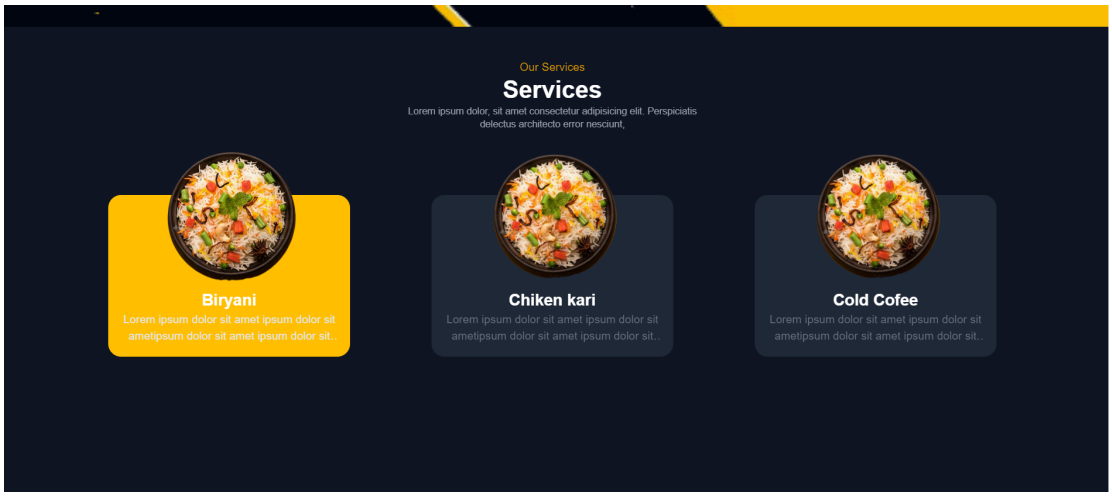



**Code**

```jsx
import React from 'react';

function Button() {
  const handleClick = () => {
    console.log('Button clicked');
  }

  return (
    <button onClick={handleClick}>Click me</button>
  );
}
```

## 15.3 Working on the hero-section of Foodi ui

## 15.4 Working on the Services section of Foodi ui

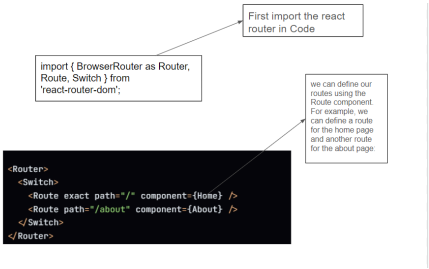


# CHAPTER 16 (week13)

## 16.1 React Router

- ➢ React Router is a tool that helps you navigate between different pages or views in a React application. It lets you define different routes for your app, and display different components based on the current URL.
- ➢ For example, you might have a route for the homepage, another route for a list of products, and a third route for a detailed view of a single product. When the user clicks a link or enters a URL, React Router figures out which component to display based on the current URL.
- ➢ To use React Router, you first need to install it as a separate library. Then, you can define your routes using the <Route> component. Each route specifies a path and a component to render when the path matches the current URL.

### Syntax




### Code

```
import ReactDOM from "react-dom/client";
import { BrowserRouter, Routes, Route } from "react-router-dom";
import Layout from "./pages/Layout";
import Home from "./pages/Home";
import Blogs from "./pages/Blogs";
import Contact from "./pages/Contact";
import NoPage from "./pages/NoPage";

export default function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Layout />}>
          <Route index element={<Home />} />
          <Route path="blogs" element={<Blogs />} />
          <Route path="contact" element={<Contact />} />
          <Route path="*" element={<NoPage />} />
        </Route>
      </Routes>
    </BrowserRouter>
  );
}
```

### Output

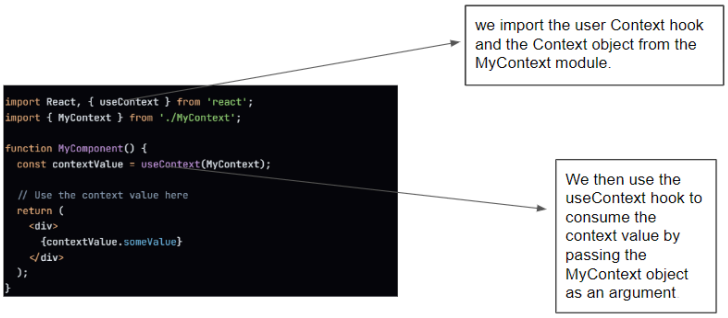
Home
Blogs
Contact

**Blog Articles**

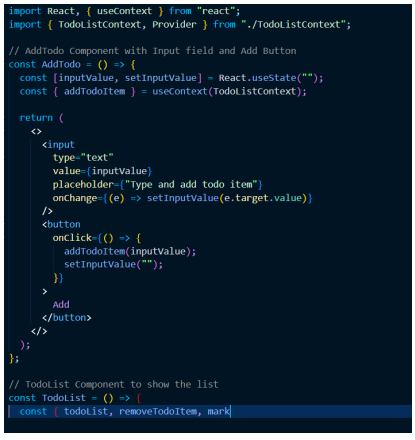
## 16.2 Context-hook

- The React Context API is a way to share data between different components in a React application, without having to pass props down manually through every level of the component tree. This is useful when you have data that needs to be accessed by multiple components, but you don't want to pass it down through every level of the component hierarchy.
- To use the Context API, you first create a context object using the React.createContext() method. Then, you define a provider component that wraps the components that need access to the shared data. The provider component has a value prop that holds the shared data.

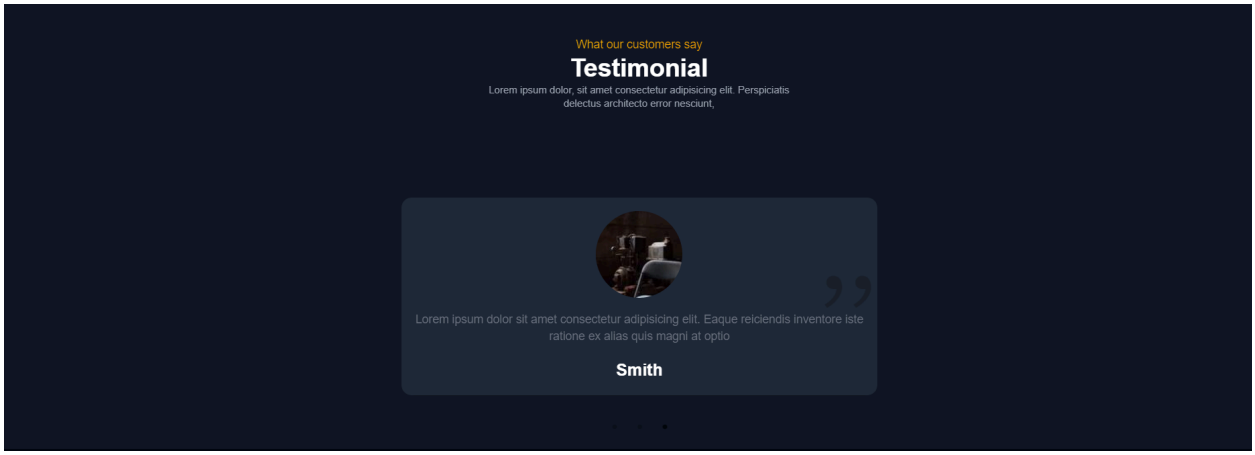
**Syntax**

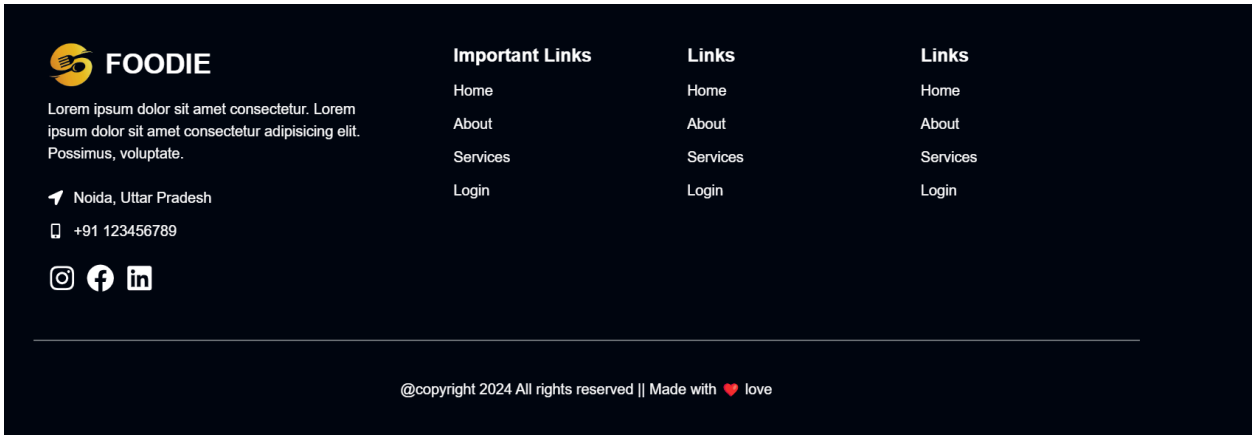



**Example**



## 16.3 Working on the Feedback and footer-section of food ui (wrapping up the project)



# CHAPTER 17 (week14)

## 17.1 Use-Hooks

- ➢ Hooks are a way to use state and other features in React without having to write a class. They were added in React version 16.8. One of the most commonly used hooks is the useState hook. This hook lets you add state to a functional component. It takes an initial state value as an argument, and returns a pair:
- ➢ the current state value, and a function that lets you update it. For example, let's say you want to create a counter component that starts at 0. You can use the useState hook to add state to the component, and update the state when the user clicks a button.

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

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

## 17.2 useCallback hook

- ➢ The useCallback hook is a way to improve the performance of your React application by preventing unnecessary re-renders of child components.
- ➢ It works by "memoizing" a function, which means that it only creates a new version of the function when its dependencies change.

**Example**

```
import { useCallback } from 'react';

function Component({ todos, addTodo }) {
  const handleAddTodo = useCallback((text) => {
    addTodo(text);
  }, [addTodo]);

  return (
    <div>
      <input type="text" onKeyDown={(e) => {
        if (e.key === 'Enter') {
          handleAddTodo(e.target.value);
          e.target.value = '';
        }
      }} />
      <ul>
        {todos.map((todo, index) => (
          <li key={index}>{todo}</li>
        ))}
      </ul>
    </div>
  );
}

function App() {
  const [todos, setTodos] = useState([]);

  const addTodo = useCallback((text) => {
    setTodos((prevTodos) => [...prevTodos, text]);
  }, []);

  return (
    <Component todos={todos} addTodo={addTodo} />
  );
}
```

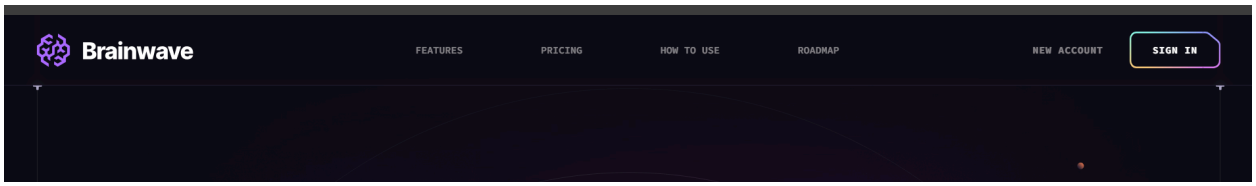
## 17.3 Start working on the new project (admin panel ) (discussion)

**Technology = Javascript , react js , material ui ,barchar.js , html , Tailwind css**

**Tools = vs code , Free- peka , font- awesome**

# CHAPTER 18 (week15)

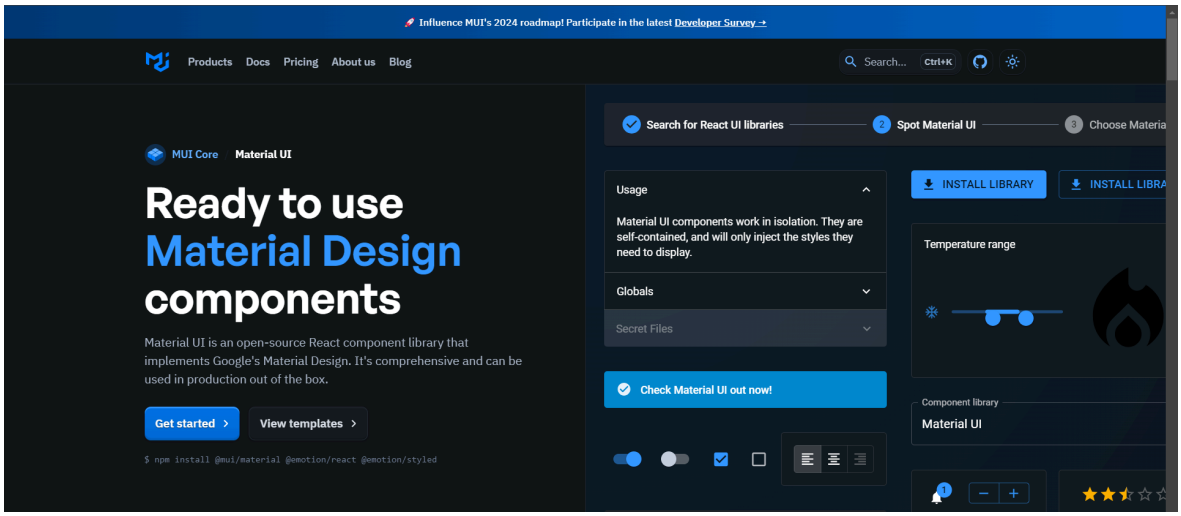
## 18.1 Navbar



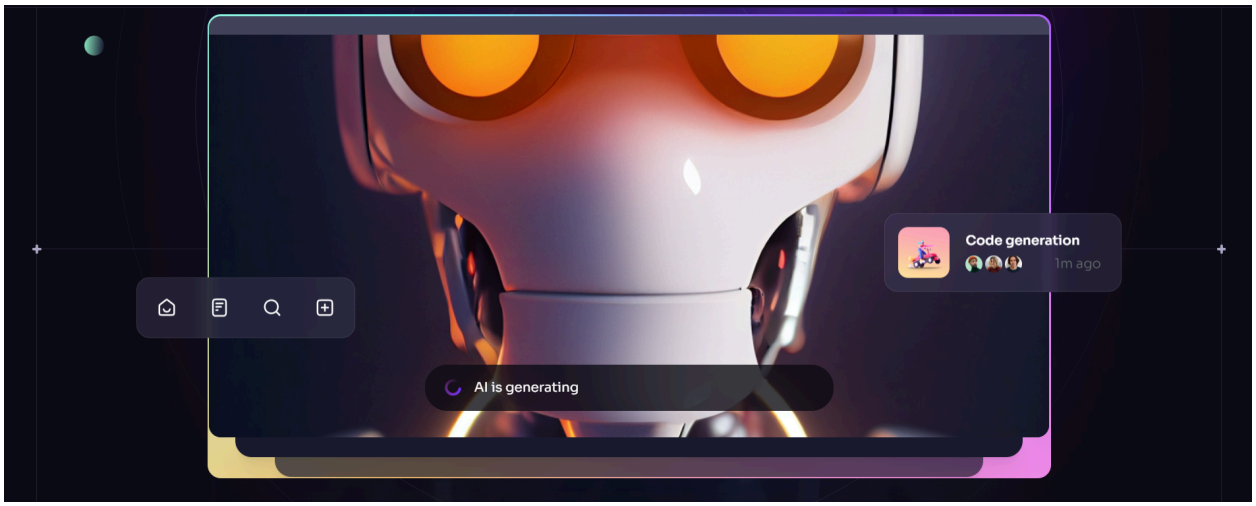
## 18.2 Material - ui

➢ Material User Interface (MUI) is an open-source React component library that implements Google's Material Design. It offers a comprehensive collection of prebuilt components that are ready for use in production right out of the box. MUI supports Material Design 2, and the adoption of Material Design 3 is tentatively planned for MIUI v6.

➢ MUI is known for its ease of use and customization options, making it a popular choice among developers. It can be installed into your application using package managers like yarn or npm. Some of the reasons why developers prefer MUI include its compatibility with popular tools like Node.js, React, Next.js, and Emotion, and its ability to optimize the design process.



## 18.3 Hero-section

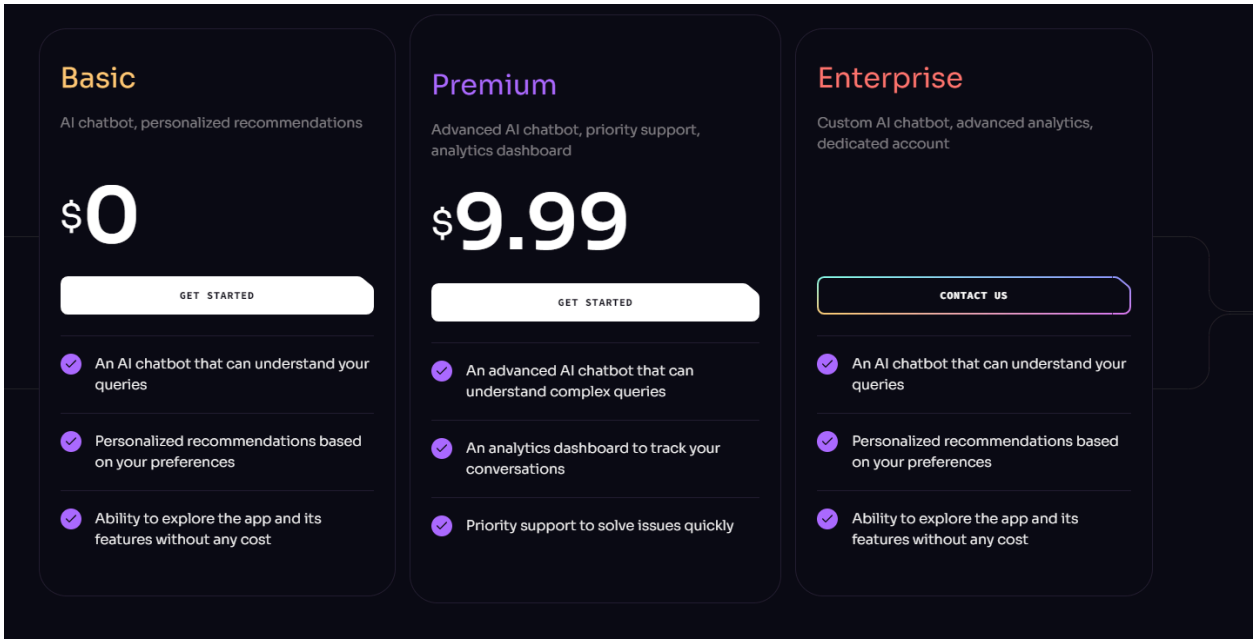


# CHAPTER 19 (week16)

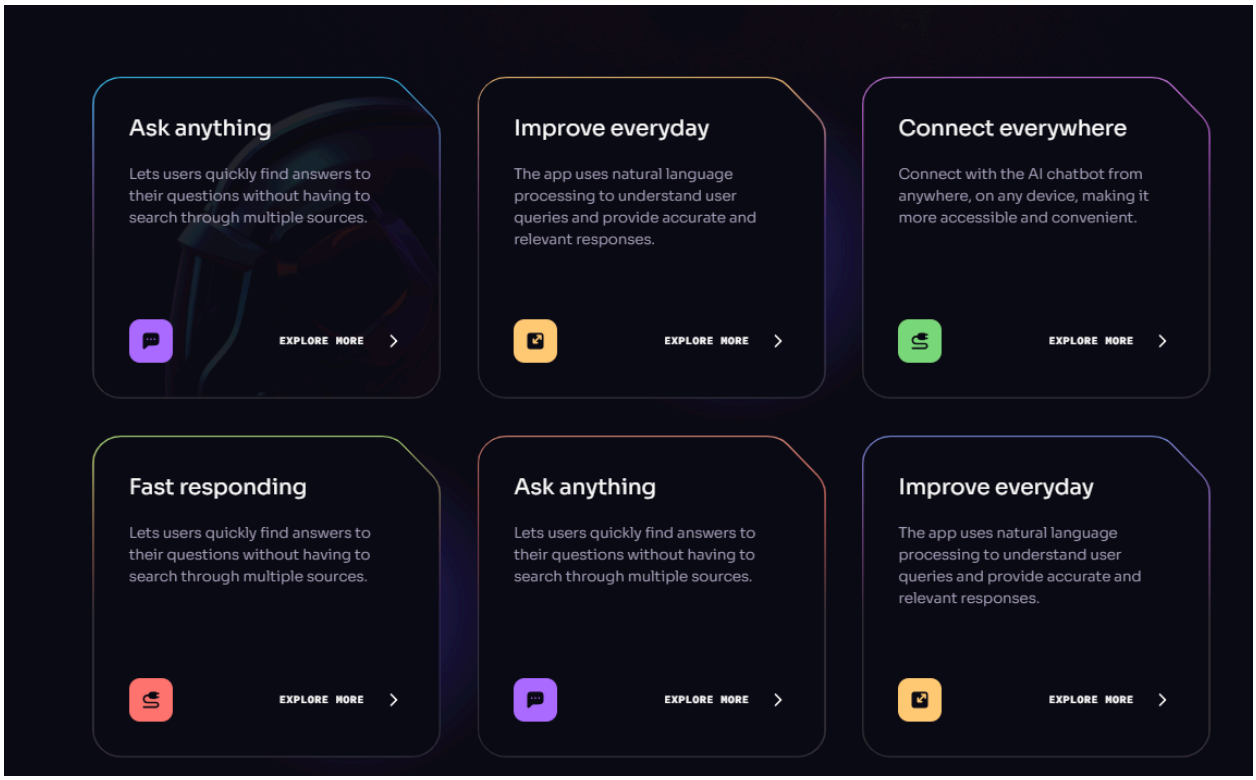
## 19.1 Understanding The Barchart.js

- ➢ Bar Chart JS is a JavaScript-based method for creating bar charts. Picturing information with even rectangular bars can be utilized.
- ➢ The value that each bar represents is inversely proportional to its length. By altering the properties of the data and options, you can further customize the chart. You can, for instance, alter the axis labels, add a title to the chart, and change the colors of the bars.

## 19.2 Working On the Pricing-section

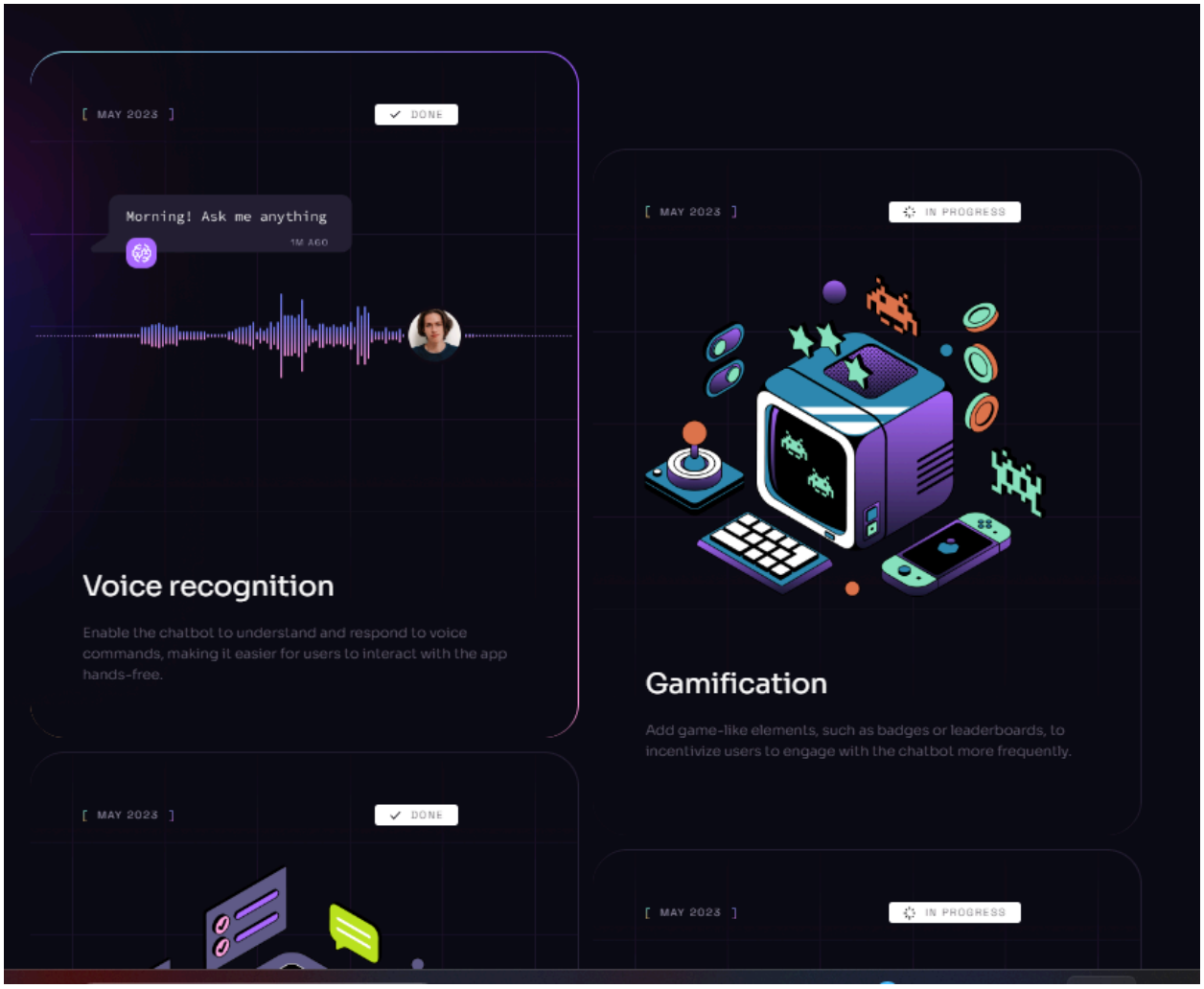


## 19.3 Working On the Feature-section

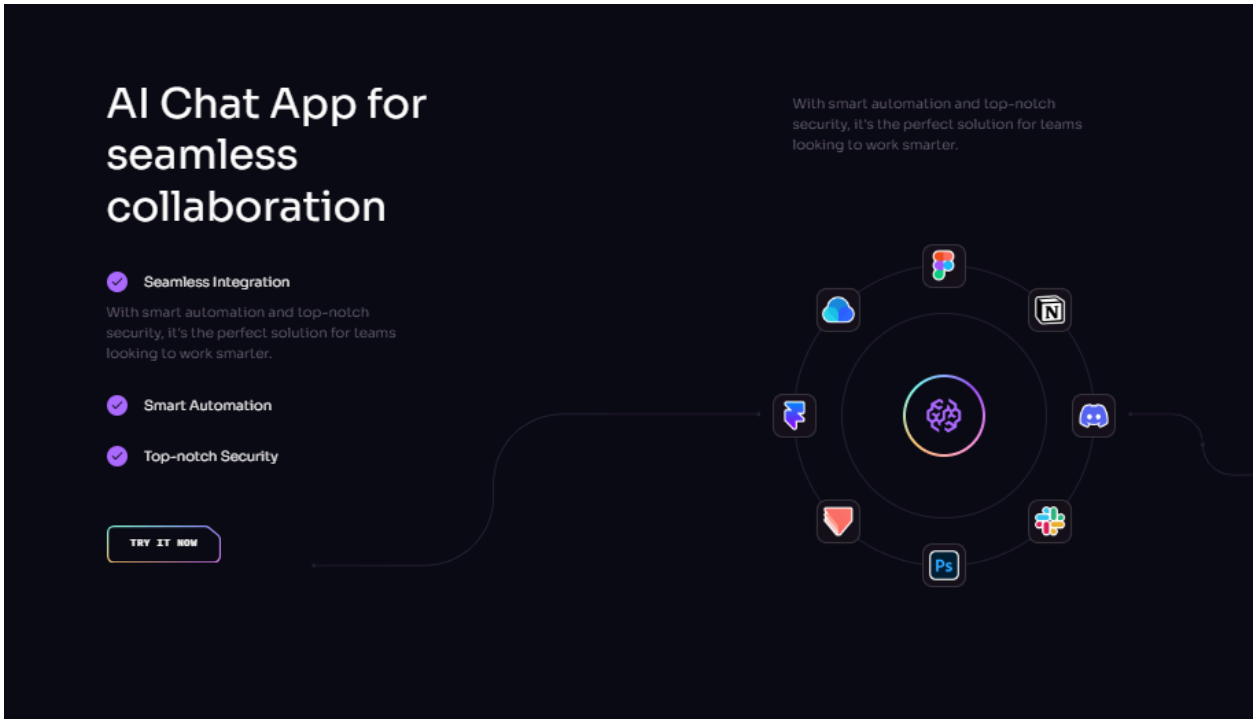


# CHAPTER 20 (week17)

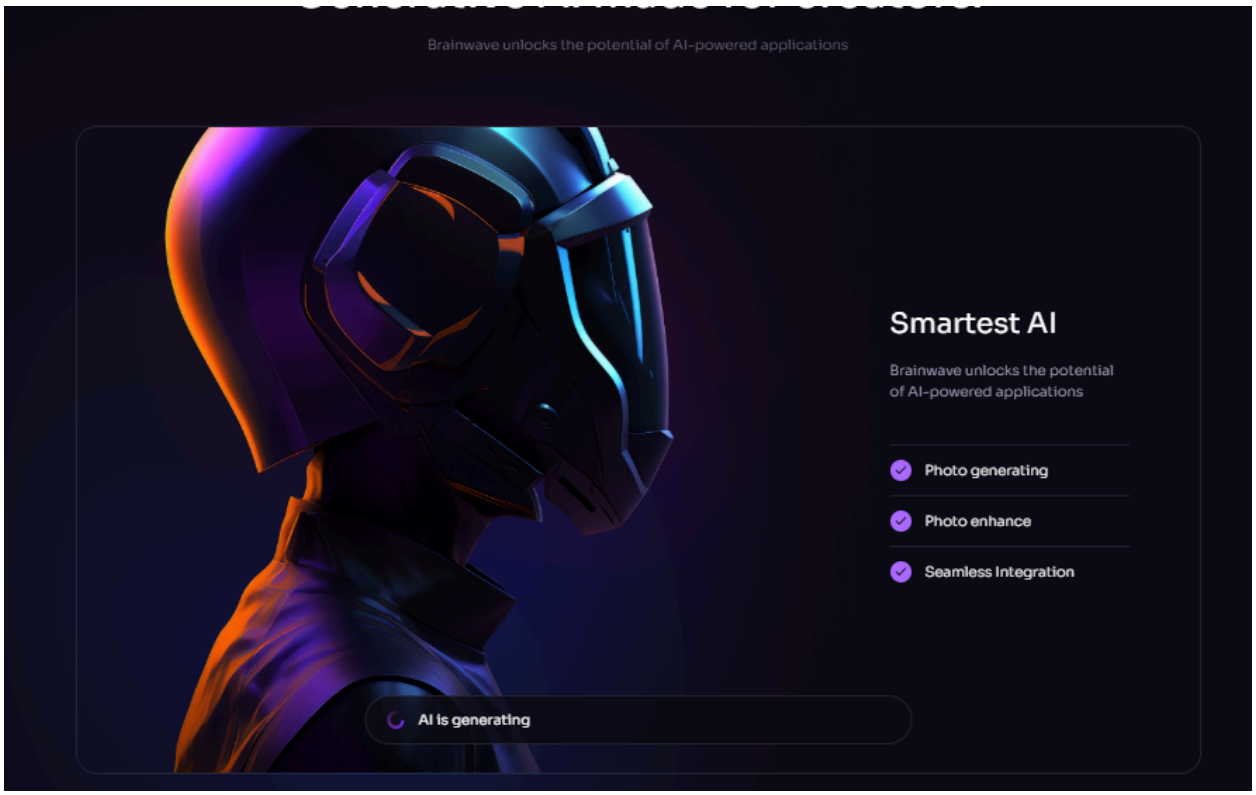
## 20.1 Working On the Footer Section



## 20.2 Working on the  Collaboration-section



## 20.3 Working on the How-to use -section



# Conclusion

In conclusion, this internship experience has provided valuable insights into the world of React.js development and its application in modern web development projects. Throughout the internship, various aspects of React.js were explored, including component-based architecture, state management, routing, and handling of asynchronous operations.

The hands-on experience gained through working on real-world projects has been instrumental in solidifying theoretical knowledge and honing practical skills. From building interactive user interfaces to optimizing performance and maintaining code quality, each task presented unique challenges and learning opportunities.

# Reference


https://www.w3schools.com/js/DEFAULT.asp

JavaScript | MDN (mozilla.org)

https://tailwindcss.com/docs/installation

https://react.dev/learn/passing-props-to-a-component

**https://www.chartjs.org/docs/latest/charts/bar.html**

**https://mui.com/material-ui/**