Task

Notification.js

```
import React, { useState, useEffect } from "react";
import "./Notification.css";
function NotificationComponent() {
  const [notifications, setNotifications] = useState([]);
  const [prevLength, setPrevLength] = useState(0);
  const storedPrevLength = localStorage.getItem("prevLength");

  useEffect(() => {
    fetch("https://databytess.com/api/adsapi/notifications?user=70")
      .then((response) => response.json())
      .then((data) => {
        // Calculate the number of new notifications
        const newNotificationsCount = data.length - storedPrevLength;
        if (newNotificationsCount > 0) {
          // Get the latest notifications
          const latestNotifications = data.slice(
            data.length - newNotificationsCount
          );
          // Show popup for each new notification
          latestNotifications.forEach((notification, index) => {
            showPopup(notification.message, index);
          });
```

```
      }
      // Update state with new notifications and store the current length
      setNotifications(data);

      // Store the current length in local storage
      localStorage.setItem("prevLength", data.length);
     })
     .catch((error) => console.error("Error fetching notifications:", error));
  }, []); // Fetch notifications only once on component mount

  const showPopup = (message, index) => {
    // Create a new popup for the given message
    const popup = document.createElement("div");
    popup.className = "popup1234";
    popup.style.bottom = `${index * 70 + 20}px`; // Adjust the spacing between popups
    popup.innerHTML = `<span class="close" onClick="this.parentNode.remove()">&times;</span>
      <p class="mass">${message}</p>`;
    document.body.appendChild(popup);

    // Close the popup after 5 seconds
    setTimeout(() => {
```

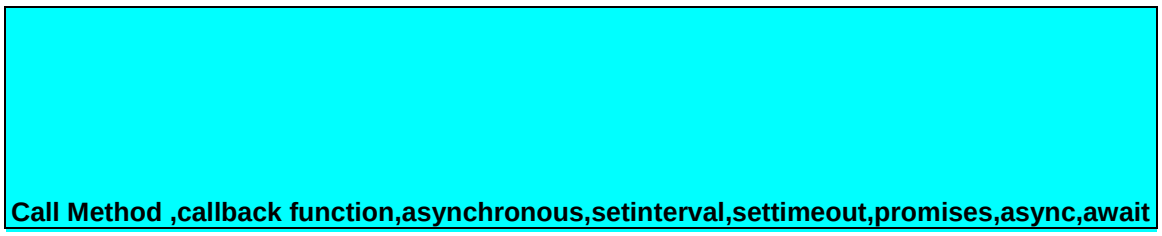
```
      popup.remove();
    }, 50000);
  };

  //   return (
  //     <div>
  //       {/* Render your notifications here */}
  //       <ul>
  //         {notifications.map(notification => (
  //           <li key={notification.id}>{notification.message}</li>
  //         ))}
  //       </ul>
  //     </div>
  //   );
}

export default NotificationComponent;
```

Notification.css

```
.popup1234 {
    position: fixed;
    bottom: 20px; /* Adjust as needed */
    right: 20px;
    background-color: rgba(0, 0, 0, 0.8); /* Black with transparency */
    color: #fff; /* Text color */
    padding: 10px;
    border-radius: 5px;
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.2);
    max-width: 300px; /* Adjust as needed */
    z-index: 9; /* Ensure it's on top of other elements */
    transition: opacity 0.3s ease; /* Add transition for smooth
appearance */
  }
  .mass{
    color: aliceblue;
    font-size: 70%;
  }
  .popup1234 .close {
    float: right;
    cursor: pointer;
    color: #fff; /* Close button color */
  }
```

```
.popup1234 .close:hover {
  color: #ccc; /* Close button color on hover */
}
```

Tranning Sessions for Feb Batch




```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>asynchronous</title>
</head>

<body>
  <p id="demo"></p>
  <!-- <h1 id="setasy"></h1>
  <h2 id="setintervaldemo"></h2> -->
  <script src="democallback.js"></script>
```

```
</body>


</html>
/*
A callback is a function passed as an argument to another function
This technique allows a function to call another function
A callback function can run after another function has finished
 */
//Example
// function greet(name, callback) {
//   console.log("Hello " + name);
//   callback();
// }
// function sayGoodBye() {
//   console.log("Goodbye");
// }

// greet("Ganesh", sayGoodBye);

// Aynchronous
/*
Functions running in parallel with other functions are called asynchronous
setTimeout()
```

```
 */
// /Example 1
// function myDisplayer(something) {
//   document.getElementById("demo").innerHTML = something;
// }

// function myCalculator(num1, num2, myCallback) {
//   let sum = num1 + num2;
//   myCallback(sum);
// }
// myCalculator(5, 5, myDisplayer);

// Example 2
// setTimeout(
//   myFuction,
//   3000
// );
/*3000  is the number of miliseconds so my fuction will called after 2 seconds.*/

// function myFuction() {
//   document.getElementById("setasy").innerHTML = "Hi how r u !!!";
// }
```

```
//Setinterval():- u can specify a callback function to be excuted for each interval.


//Exmaple 1
// setInterval(updatetime, 1000);
// function updatetime() {
//   let d = new Date();
//   document.getElementById("setintervaldemo").innerHTML =
//     d.getHours() + ":" + d.getMinutes() + ":" + d.getSeconds();
// }

// Exmaple 2
// asynchronous
// function fetchData(callback) {
//   setTimeout(() => {
//     const data = "Data from server";
//     callback(data);
//   }, 2000);
// }
// function processData(data) {
//   console.log("Received data", data);
// }
```

```
// fetchData(processData);

//Promises
/*
"Producing code" is code that can take some time
"Consuming code" is code that must wait for the result
A Promise is an Object that links Producing code and
Consuming code.
 */
// object properties
// 1 pending
// 2 fulfilled
// 3 rejected

// it support 2 properties
// 1 state
// 2 result

//when a promise is "pending " the result is undefined
// when a promise is "fulfilled" the result is a value
// when a promise is "rejected" the result is error as object .

/*
myPromise.state                    myPromise.result
```

```
"Pending"                    undefined
"fulfilled"                   as value
"rejected"                   error object
 */

// note that u can not the Promises properties state & object
// u must use a promise method handle process.

/*
How to use:
myPromise.then(
   function(value){/ code if successful/ }
   function(error){/ code if some error/ }
)
 */

// here u cnan see that 2 arguments , a callback for success &
another for failure

// Both r optional so u can add a caalback for success or failure only

//Example

// function myDisplayer(some) {
```

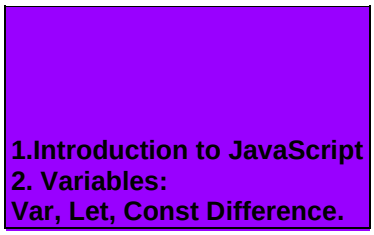
```
//   document.getElementById("demo").innerHTML = some;
// }

// let myPromise = new Promise(function (myResolve, myReject) {
//   let x = 0;

//   if (x == 1) {
//     myResolve("OK");
//   } else {
//     myReject("Error");
//   }
// });

// myPromise.then(
//   function (value) {
//     myDisplayer(value);
//   },
//   function (error) {
//     myDisplayer(error);
//   }
// );

// Async & Await :
// async makes a function return promise
```

```
// await makes a function wait for promise


// Async
// the keywords before a function makes the function eturn promise.


// Example
// async function myFuction() {
//   return "Hello, ";
// }


// // is the same as:
// function myFuction() {
//   return Promise.resolve("Hello");
// }


// //here is how to use Prmoises.
// myFunction().then(
//   function (value) {
//     /* code if successful */
//   },
//   function (error) {
//     /* code if some error */
//   }
```

```
// );

//Example
// function myDisplayer(some) {
//   document.getElementById("demo").innerHTML = some;
// }
// async function displayMessage() {
//   return "Hello";
// }
// displayMessage().then(
//   function (value) {
//    {
//      myDisplayer(value);
//    }
//   },
//   function (error) {
//    {
//      myDisplayer(error);
//    }
//   }
// );

// Await
//Syntax
```

```
// let value = await promise

// await keyword makes the function pause the excecution and wait for a resolved promise before it contiues.

//Example 1 : await

async function myDisplay() {

  let myPromise = new Promise(function (resolve, reject) {

    resolve("Hi How r u ???");

  });

  document.getElementById("demo").innerHTML = await myPromise;

}

myDisplay();
```

Tranning Sessions for March Batch




```
<!DOCTYPE html>

<html lang="en">


<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>js file</title>
</head>

<body>
    <!--
        innerHtml
        document.write()
        window.alert()
        console.log()
     -->
    <h1 id="demo"></h1>

    <script src="/main.js">

    </script>
</body>

</html>
// console.log("Hello");
// document.write("karan");
// window.alert("are y want to exit?");
// document.getElementById("demo").innerHTML = "ram";

// Variables
```

```
// it is used for storing data
//automatically
//using var
//using let
// using const

//example1 : automically
// x = 5;
// y = 6;
// z = x + y;
// console.log(z);

//example2 : var
var a = 10;
var b = 20;
var c = a - b;
console.log(c);

//example2 : let
let d = 5;
let e = 5;
let f = d * e;
console.log(f);
```

```
//example2 : const
const xx = 5;
const yy = 10;
const zz = xx / yy;
console.log(zz);

// when we have to use var , let const
// always declare varibale
//// Bad practice: Using variable without declaration
// myVar = 10;

// Good practice: Declaring the variable before using it
// let myVar = 10;

// always use const if the value should not be change
//const PI = 3.14;
// always use const if the value should not be changed (Arrays & Objects)
const person = {
  name: "Ramesh",
  age: 30,
};
person.age = 31; // it valid , mutating the object property
console.log(person);
```

```
// use let if u can not use const
let counter = 0; //0
counter = counter + 1; //1
console.log(counter); //1
// use var if u must support old browsers

const newdate = "07 March 2024";
console.log(newdate);

// rules for naming vairables
//Name - can contain lettes, digits ,undersocres & doller signs.
// begin with the letter
// name case sensitive (y or Y)
//reserved keywords can not used as names.
let firstName = "Yash";

let y = 10;
let Y = 20;
console.log(y + Y);
```