R20

**FULL STACK DEVELOPMENT LAB MANUAL**

**III B. Tech (CSM&AIM) – II Semester**

**Department of CSE(Artificial Intelligence & Machine Learning)**

**VASIREDDY VENKATADRI INSTITUTE OF TECHNOLOGY**

(Autonomous)

**Nambur (V), Peda Kakani (M), Guntur (Dt.) 522508**



# VASIREDDY VENKATADRI INSTITUTE OF TECHNOLOGY, NAMBUR

(Autonomous)

**DEPARTMENT OF CSE(Artificial Intelligence & Machine Learning)**

Providing quality education to enable the generation of socially conscious software engineers who can contribute to the advancement in the field of computer science and engineering.

**Vision of the Department**

**Mission of the Department**

* To equip the graduates with the knowledge and skills required to enable them to be industry ready.
* To train socially responsible, disciplined engineers who work with good leadership skills and can contribute for nation building.
* To make our graduates proficient in cutting edge technologies through student centric teaching-learning process and empower them to contribute significantly to the software industry
* To shape the department into a centre of academic and research excellence

|  |  |
| --- | --- |
| **Program Educational Objectives** | |
| **PEO-1** | To provide the graduates with **solid foundation** in Computer Science and Engineering along with the fundamentals of Mathematics and Sciences with a view to impart in them **high quality technical skills** like **modelling, analysing, designing, programming and implementation** with **global**  **competence** and helps the graduates for **life-long learning**. |
| **PEO-2** | To prepare and motivate graduates with **recent technological developments related to core subjects** like Programming, Databases, Design of Compilers and Network Security aspects and future technologies so as to contribute effectively for Research & Development by participating  in professional activities like publishing and seeking copy rights. |
| **PEO-3** | To train graduates to choose a **decent career option either in high degree of employability/Entrepreneur or, in higher education** by empowering students with ethical administrative acumen, ability to handle critical  situations and training to excel in competitive examinations. |
| **PEO-4** | To train the graduates to have basic **interpersonal skills** and **sense of social responsibility** that paves them a way to become good team members and  leaders. |

# Program Specific Outcomes (PSOs)

**PSO-1: Professional Skills:** The ability to understand, analyse and develop computer programs in the areas related to algorithms, system software, multimedia, web design, big data analytics, and networking for efficient design of computer-based systems of varying complexity.

**PSO-2: Successful Career and Entrepreneurship:** The ability to employ modern computer languages, environments, and platforms in creating innovative career paths to be an entrepreneur and a zest for higher studies/employability in the field of Computer Science & Engineering.

**Program Outcomes:**

1. **Engineering knowledge:** apply the knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems. **(L3-Apply)**
2. **Problem analysis:** identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural science and engineering sciences. **(L4-Analysis)**
3. **Design/development of solutions:** design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal and environmental considerations. **(L6-Create)**
4. **Conduct investigations of complex problems:** use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions. **(L5-Evaluation)**
5. **Modern tool usage:** create, select and apply appropriate techniques, resources and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations. **(L3-Apply)**
6. **The engineer and society:** apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice**. (L3-Apply)**
7. **Environment sustainability:** understand the impact of the professional engineering solutions in the societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development. **(L2-Understand)**
8. **Ethics:** apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice. **(L1-Remember)**
9. **Individual and team work:** function effectively as an individual and as a member or leader in diverse teams, and in multidisciplinary settings. **(L1-Remember)**
10. **Communication:** communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions. **(L1-Remember)**
11. **Project management and finance:** demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments. **(L3-Apply)**
12. **Lifelong learning:** recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broader context of technological change**. (L1-Remember)**



**Department of CSE(Artificial Intelligence & Machine Learning)**

**Internal Lab Evaluation Process:**

|  |  |
| --- | --- |
| Day to day evaluation | 5M |
| Record | 5M |
| Conduct of Experiment / Program | 3M |
| Viva Voce | 2M |
| **Total** | **15 Marks** |

**External Lab Evaluation Process:**

|  |  |
| --- | --- |
| Description | 10 M |
| Program | 10M |
| Execution | 10M |
| Viva Voce | 5M |
| **Total** | **35 Marks** |

|  |  |
| --- | --- |
| Internal Marks | 15 M |
| External Marks | 35 Marks |
| Total | 50 Marks |

|  |  |
| --- | --- |
| **LIST OF EXPERIMENTS** | |
| **Prerequisites:** HTML, CSS, Java script  **Tools Used:** Node JS Server, Any Browser | |
| **S. NO** | **PROBLEM STATEMENT** |
| **1** | Try to recreate the following patterns using HTML and CSS only. |
| **2** | Implement Drag n Drop feature in HTML 5 |
| **3** | Demonstrate Event bubbling with necessary examples. |
| **4** | Design a Calculator using Java script and relevant CSS.  Description: How To Build A Simple Calculator Application With React.JS | by Nitin Patel  | Medium |
| **5** | Demonstrate Higher order functions with necessary examples –  filter(), reduce() and map() |
| **6** | Create a Class Component for Counter in React JS |
| **7** | Create a Class component for Changing the color of the text given in React JS |
| **8** | Create a Class Component for viewing an array of objects in a tabular form. |
| **9** | Display a digital clock in React JS. |
| **10** | Demonstrate useState Hook with necessary example. |
| **11** | Demonstrate useContext Hook with necessary example. |
| **12** | Demonstrate useEffect Hook with necessary example. |
| **13** | Demonstrate consuming web API using fetch & axios (AXIOS API). Demonstrate with the help of fake URL. |
| **14** | Design a BMI calculator using React JS based on the description given below: |

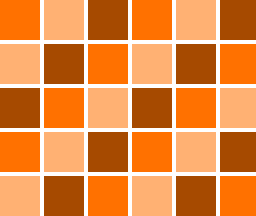
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | BMI is a measurement of a person's leanness or corpulence based on their height and weight, and is intended to quantify tissue mass. It is widely used as a general indicator of whether a person has a healthy body weight for their height.  Formula:  weight (kg) / [height (m)]2 (or) [weight (kg) / height (cm) / height (cm)] x 10,000  BMI table for adults: This is the World Health Organization's (WHO) recommended body weight based on BMI values for adults. It is used for both men and women, age 18 or older. | | | |
|  | Category | BMI range - kg/m2 |  |
| Severe Thinness | < 16 |
| Moderate Thinness | 16 - 17 |
| Mild Thinness | 17 - 18.5 |
| Normal | 18.5 - 25 |
| Overweight | 25 - 30 |
| Obese Class I | 30 - 35 |
| Obese Class II | 35 - 40 |
| Obese Class III | > 40 |
| **15** | Display a selected set of images in tabular format using React JS. | | | |
| **16** | Implement Upload & down load options on a given file. | | | |
| **17** | Create a React application to view EMI calculator. A specific view is given below:  Description: EMI Calculator | | | |
| **18** | Design the following Hotel bill screen. User can select as many items as possible from the dropdown box and is allowed to enter in the text field provided. Each transaction must be added in the table given below along with the bill amount. | | | |

|  |  |
| --- | --- |
|  |  |
| **19** | Demonstrate the procedure to create a schema in MongoDB. |
| **20** | Demonstrate CRUD operations using MongoDB. |

**\*\*\*\*\*\***

# Exp. No 1

**Problem Statement:** Draw the following pattern in HTML and CSS only.



**Description:** Here, the pattern can be obtained using a little styling (<style> </style> tags) in the HTML component. In addition to that a table can be prepared with different color patterns. However, this pattern may be obtained by writing java script code too.

## Code:

<!DOCTYPE html>

<html>

<head>

<title>Experiment-1</title>

<style> td {

width: 35px; height: 35px;

}

#t1{

background-color: orange;

}

#t2{

background-color:red

} #t3{

background-color:yellow;

}

</style>

</head>

<body>

<center>

<table style="border-width: 5px;">

<tr>

<td id="t1"></td>

<td id="t2"></td>

<td id="t3"></td>

<td id="t1"></td>

<td id="t2"></td>

<td id="t3"></td>

</tr>

<tr>

<td id="t2"></td>

<td id="t3"></td>

<td id="t1"></td>

<td id="t2"></td>

<td id="t3"></td>

<td id="t1"></td>

</tr>

<tr>

<td id="t3"></td>

<td id="t1"></td>

<td id="t2"></td>

<td id="t3"></td>

<td id="t1"></td>

<td id="t2"></td>

</tr>

<tr>

<td id="t1"></td>

<td id="t2"></td>

<td id="t3"></td>

<td id="t1"></td>

<td id="t2"></td>

<td id="t3"></td>

</tr>

<tr>

<td id="t2"></td>

<td id="t3"></td>

<td id="t1"></td>

<td id="t2"></td>

<td id="t3"></td>

<td id="t1"></td>

</tr>

<tr>

<td id="t3"></td>

<td id="t1"></td>

<td id="t2"></td>

<td id="t3"></td>

<td id="t1"></td>

<td id="t2"></td>

</tr>

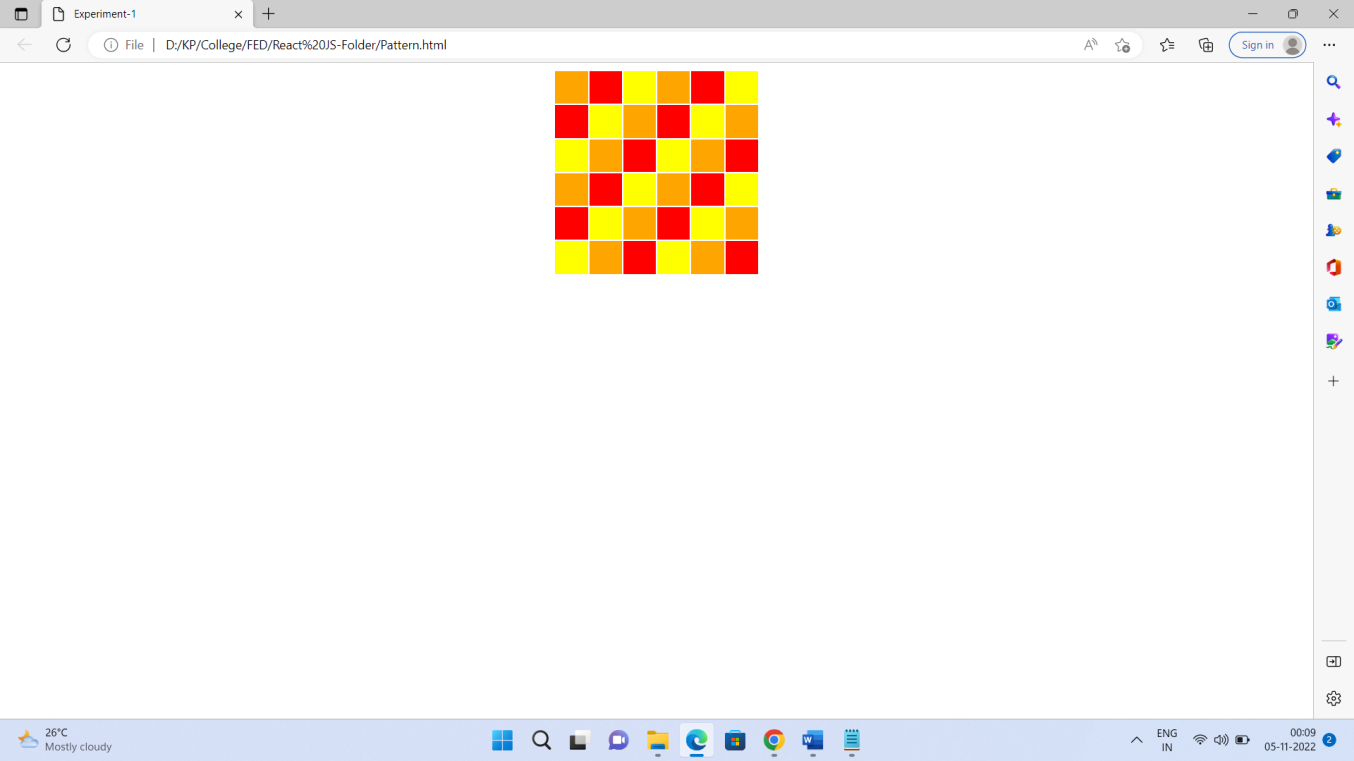
</table>

</center>

</body>

</html>

**Output:**

****

# Exp. No 2

## Problem Statement: demonstrate drag and drop feature in HTML 5. Description:

Before HTML5, if we wanted to implement drag and drop features in the browser, we had to use

libraries such as jQuery and MooTools. Now we have HTML5 Drag and Drop, which gives us the ability to drag, drop, and transfer data natively within the browser.

While Drag and Drop is a new API included in HTML5, it was – amazingly – first implemented in IE5, back in 2005 when Ian Hickson worked on the HTML5 API. To create draggable content is easy; you just add **draggable=true** on to the element you want to make moveable.

The Drag and Drop API adds seven new events HTML elements can listen out for; these can be used to monitor a drag and drop.

The events **dragstart**, **drag**, **dragend**, **dragenter**, **dragleave**, **dragover** and **drop** can be split into elements that can be dragged and elements that can have items dropped into them.

Elements that are dragged can trigger three events:

* **dragstart**: triggered when dragging a draggable element.
* **drag**: triggered when the draggable element is moved.
* **dragend**: triggered when the drag and drop operation ends.
* **dragenter**: triggered when a draggable object is dragged over an element.

## dragleave: triggered when a draggable object is dragged outside of an element.

* **dragover**: triggered when a draggable object is moved inside an element.
* **drop**: triggered when a draggable object is dropped.

Code:

<!DOCTYPE html>

<head>

<title>Using Drag and Drop</title>

<script>

function dragStart(e) {

e.dataTransfer.effectAllowed = "move"; e.dataTransfer.setData("Text", e.target.getAttribute("id"));

}

function dragOver(e) {

e.preventDefault(); e.stopPropagation();

}

function drop(e) {

e.stopPropagation(); e.preventDefault();

var data = e.dataTransfer.getData("Text"); e.target.appendChild(document.getElementById(data));

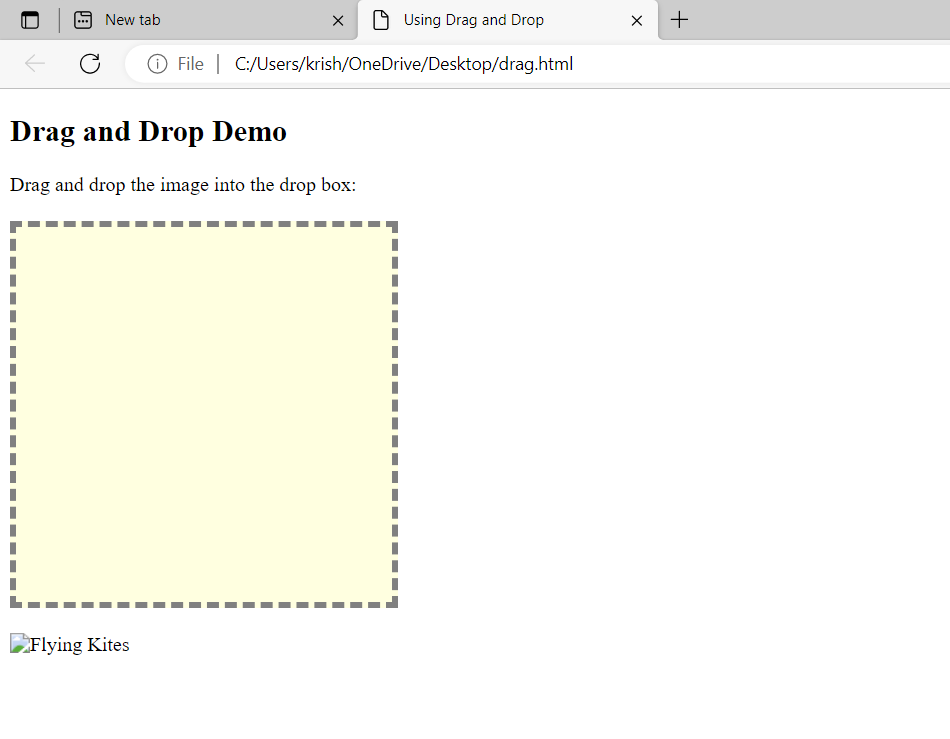
}

</script>

<style>

#dropBox { width: 300px; height: 300px;

border: 5px dashed gray; background: lightyellow; text-align: center; margin: 20px 0;



color: orange;

}

#dropBox img { margin: 25px;

}

</style>

</head>

<body>

<h2>Drag and Drop Demo</h2>

<p>Drag and drop the image into the drop box:</p>

<div id="dropBox" ondragover="dragOver(event);" ondrop="drop(event);">

</div>

<img src="../images/kites.jpg" id="dragA" draggable="true" ondragstart="dragStart(event);" width="250" height="250" alt="Flying Kites">

</body>

</html>

**Output:**

# Exp. No 3

**Problem Statement:** Demonstrate Event bubbling with necessary examples.

**Description:** Event bubbling is a method of event propagation in the HTML DOM API when an event is in an element inside another element, and both elements have registered a handle to that event. It is a process that starts with the element that triggered the event and then bubbles up to the containing elements in the hierarchy. In event bubbling, the event is first captured and handled by the innermost element and then propagated to outer elements.

Syntax: addEventListener(type, listener, useCapture)

Here,

**type:** Use to refer to the type of event.

**listener:** Function we want to call when the event of the specified type occurs. **userCapture:** Boolean value. Boolean value indicates event phase. By Default useCapture is false. It means it is in the bubbling phase.

## Source code:

<html>

<head>

<title>Bubbling Event in Javascript</title>

</head>

<body>

<h2>Bubbling Event in Javascript</h2>

<div id="parent">

<button>

<h2>Parent</h2>

</button>

<button id="child">

<p>Child</p>

</button>

</div>

<br>

<script>

document.getElementById("child").addEventListener("click", function () { alert ("You clicked the Child element!");

}, false);

document.getElementById("parent").addEventListener("click", function () { alert ("You clicked the parent element!");

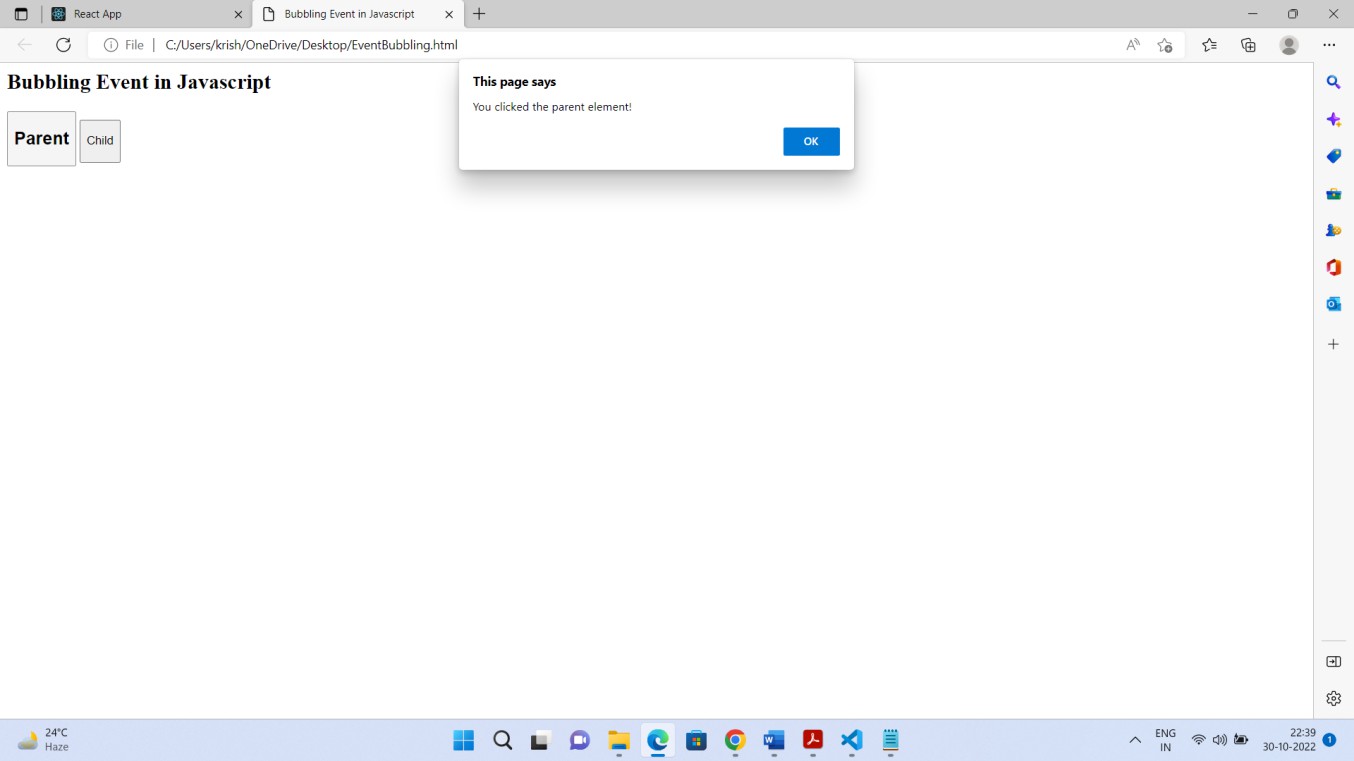
}, false);

</script>

</body>

</html>

**Output:**

****

# Exp. No 4

## Problem Statement: Prepare a calculator using HTML & Java script.

**Description:** Here are the steps to create a simple calculator using HTML and JavaScript which can evaluate simple arithmetic on integer numbers. Two types of inputs text and button are used here on a table within a form element and the OnClick event was used to insert button values on the screen or to evaluate the numbers.

## Code:

<html>

<center><h1>Calculator</h1></center>

<head>

<style>

table {

border:2px solid black; width: 400px; height:400px;

margin-left: auto; margin-right: auto;

background-color: rgb(196, 185, 185);

}

input[type="button"] { width: 100%;

font-weight: bold; height:100%;

}

input[type="text"] { font-size: medium; width: 100%; height:100%;

background-color: rgb(196, 185, 185);

}

</style>

</head>

<table>

<tr>

<td colspan="5"><input type="text" id="res"></td>

</tr>

<tr>

<td><input type="button" value="(" onclick="dis('(')"></td>

<td><input type="button" value="CE" onclick="dis('CE')"></td>

<td><input type="button" value=")" onclick="dis(')')"></td>

<td><input type="button" value="C" onclick="clr()"></td>

</tr>

<tr>

<td><input type="button" value="1" onclick="dis('1')"></td>

<td><input type="button" value="2" onclick="dis('2')"></td>

<td><input type="button" value="3" onclick="dis('3')"></td>

<td><input type="button" value="+" onclick="dis('+')"></td>

</tr>

<tr>

<td><input type="button" value="4" onclick="dis('4')"></td>

<td><input type="button" value="5" onclick="dis('5')"></td>

<td><input type="button" value="6" onclick="dis('6')"></td>

<td><input type="button" value="-" onclick="dis('-')"></td>

</tr>

<tr>

<td><input type="button" value="7" onclick="dis('7')"></td>

<td><input type="button" value="8" onclick="dis('8')"></td>

<td><input type="button" value="9" onclick="dis('9')"></td>

<td><input type="button" value="" onclick="dis('')"></td>

</tr>

<tr>

<td><input type="button" value="." onclick="dis('.')"></td>

<td><input type="button" value="0" onclick="dis('0')"></td>

<td><input type="button" value="=" onclick="solve()"></td>

<td><input type="button" value="/" onclick="dis('/')"></td>

</tr>

</table>

<script type="text/javascript"> function dis(val){

document.getElementById("res").value+=val;

}

function solve(){

let x=document.getElementById("res").value; let y=eval(x); document.getElementById("res").value=y;

}

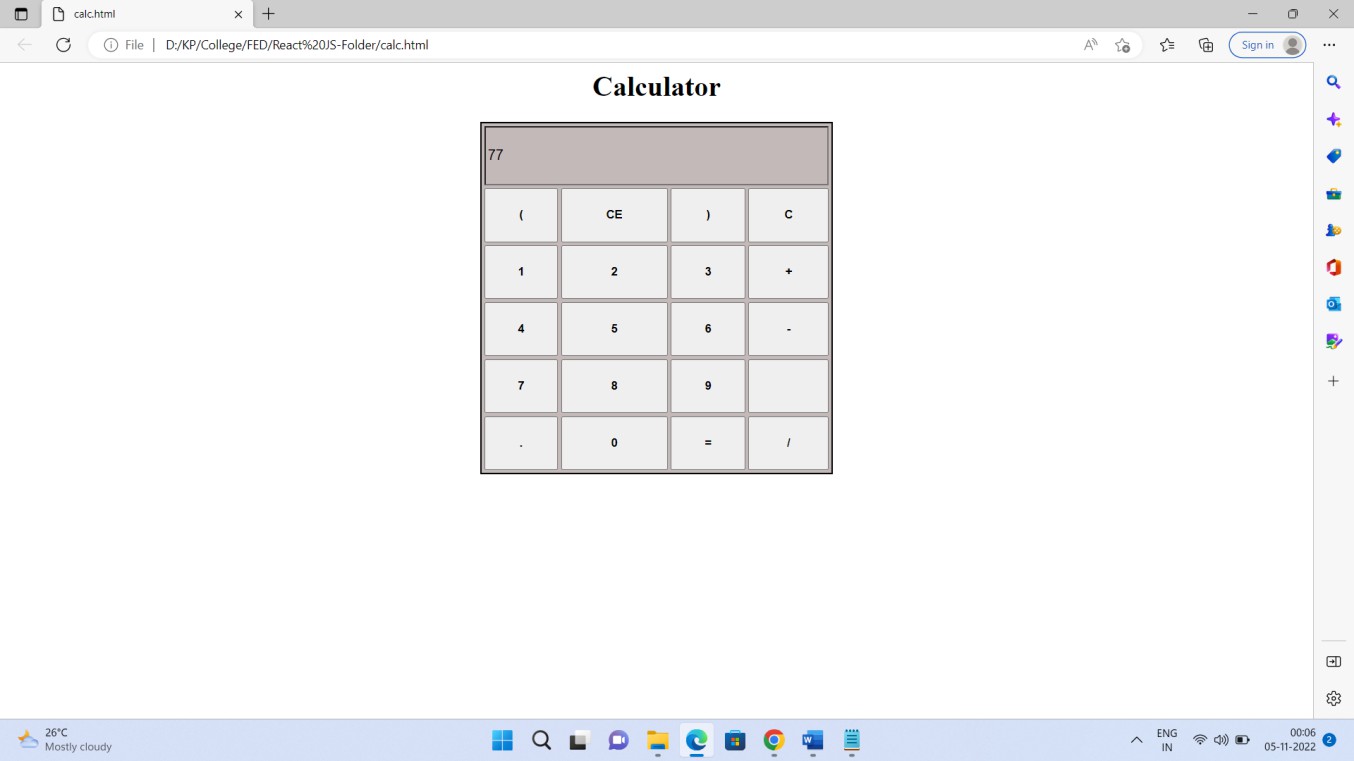
function clr(){ document.getElementById("res").value="";

}

</script>

</html>

**Output:**

****

# Exp. No 5

**Problem Statement:** Demonstrate Higher order functions with necessary examples – filter(), reduce() and map()

**Description:** map(), filter() and reduce() are popular Higher order functions in Java script.

A “higher order function” is a function that accepts functions as parameters and/or returns a function.

**filter():**

1. The filter() method creates a new array filled with elements that pass a test provided by a function
2. The filter() method does not execute the function for empty elements
3. The filter() method does not change the original array

Syntax: array.filter(function( currentValue index, arr thisValue))

**map():** Java script map() creates a new array, which contains the results obtained from iterating over the elements of the specified array and calling the provided function once for each element in order.

Syntax:

const newArray = oldArray.map((currentValue, index, array)=> {

// Do stuff with currentValue

}

);

**reduce ():** The reduce () method executes a user supplied callback function on each element of the array, in order, passing in the return value from the calculation on the preceding element The final result of running

the reducer across all elements of the array is a single value.

The reduce () method:

* executes a reducer function for array element.
* returns a single value: the function's accumulated result.
* does not execute the function for empty array elements.
* does not change the original array.

Ex:

const a = [33,66,44,88,11];

console.log(a.reduce((a,b)=>(a>b?a:b));

**Source code:**

Ex-1:

const input = [

{

name: 'John', age: 13

},

{

name: 'Mark', age: 56,

},

{

name: 'Rachel', age: 45,

},

{

name: 'Nate', age: 67,

},

{

name: 'Jeniffer', age: 65,

}

];

const ages = input.map(person => person.age);

[Math.min(...ages), Math.max(...ages), Math.max(...ages) - Math.min(...ages)];

## Ex-2:

let studentRecords = [ {name: 'John', id: 123, marks: 98 },

{name: 'Baba', id: 101, marks: 23 },

{name: 'yaga', id: 200, marks: 45 },

{name: 'Wick', id: 115, marks: 75 } ]

let names = studentRecords.map( stu => stu.name.toUpperCase()); console.log(names); // logs:

let names = studentRecords.filter(stu => stu.marks > 50); console.log(names); //logs:

let totalMarks = studentRecords.reduce( ((acc,emp) => acc+emp.marks), 0) console.log(totalMarks);

## Output:

The result logs the following….

[ 'JOHN', 'BABA', 'JOHN', 'WICK' ]

[ { name: 'John', id: 123, marks: 98 } ] 241

# Exp. No. 6

**Problem Statement:** Incrementing the counter using Class Components in React JS.

**Description:** React Components: Components are independent and reusable bits of code. They serve the same purpose as JavaScript functions, but work in isolation and return HTML via a render() function.

Components come in two types, Class components and Function components. Before React 16.8, Class components were the only way to track state and lifecycle on a React component.

The component has to include the extends React.Component statement, this statement creates an inheritance to React.Component, and gives your component access to React.Component's functions.

The component also requires a render () method, this method returns HTML.

## Source code:

import React from 'react';

class CounterDemo extends React. Component { state = {

count: 0

};

handleClick = e => {

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

};

render() { return (

<div>

<h1>Counter Demo - Class Components</h1>

<button id = "kpbtn" className = "block" onClick = {this. handleClick}>

<div className="counter"> {this. state. Count} </div>

</button>

</div>

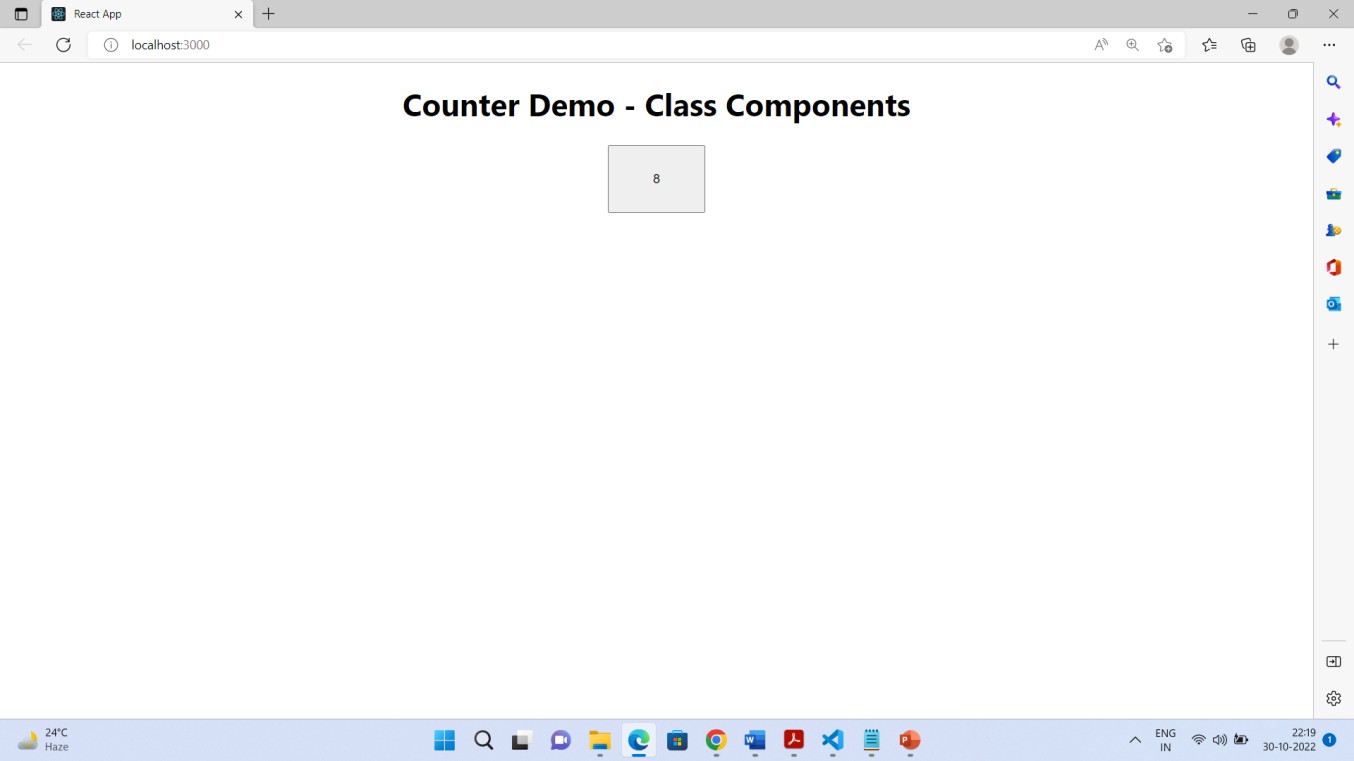
);

}

}

export default CounterDemo;

**Output:**

****

# Exp. No. 7

**Problem Statement:** Create Class component for Changing the color of the text given in React JS

**Description:** React Components: Components are independent and reusable bits of code. They serve the same purpose as JavaScript functions, but work in isolation and return HTML via a render () function.

Components come in two types, Class components and Function components. Before React 16.8, Class components were the only way to track state and lifecycle on a React component.

The component has to include the extends React.Component statement, this statement creates an inheritance to React.Component, and gives your component access to React.Component's functions.

The component also requires a render() method, this method returns HTML.

## Source code:

import React, {Component} from 'react'; class TextColorDemo extends Component {

constructor(props) {

super(props); this.state = { color : "#FF00FF"

}

}

setRed = (e) => {

this.setState({color: "#FF0000"});

}

setGreen = (e) => {

this.setState({color: "#00FF00"});

}

setBlue = (e) => {

this.setState({color: "#0000FF"});

}

render () {

return (

<div>

<h1 style={{color:this.state.color}}>This is CSE B class room </h1>

<button onClick = {this.setRed}>Red</button>

<button onClick = {this.setGreen}>Green</button>

<button onClick = {this.setBlue}>Blue</button>

</div>

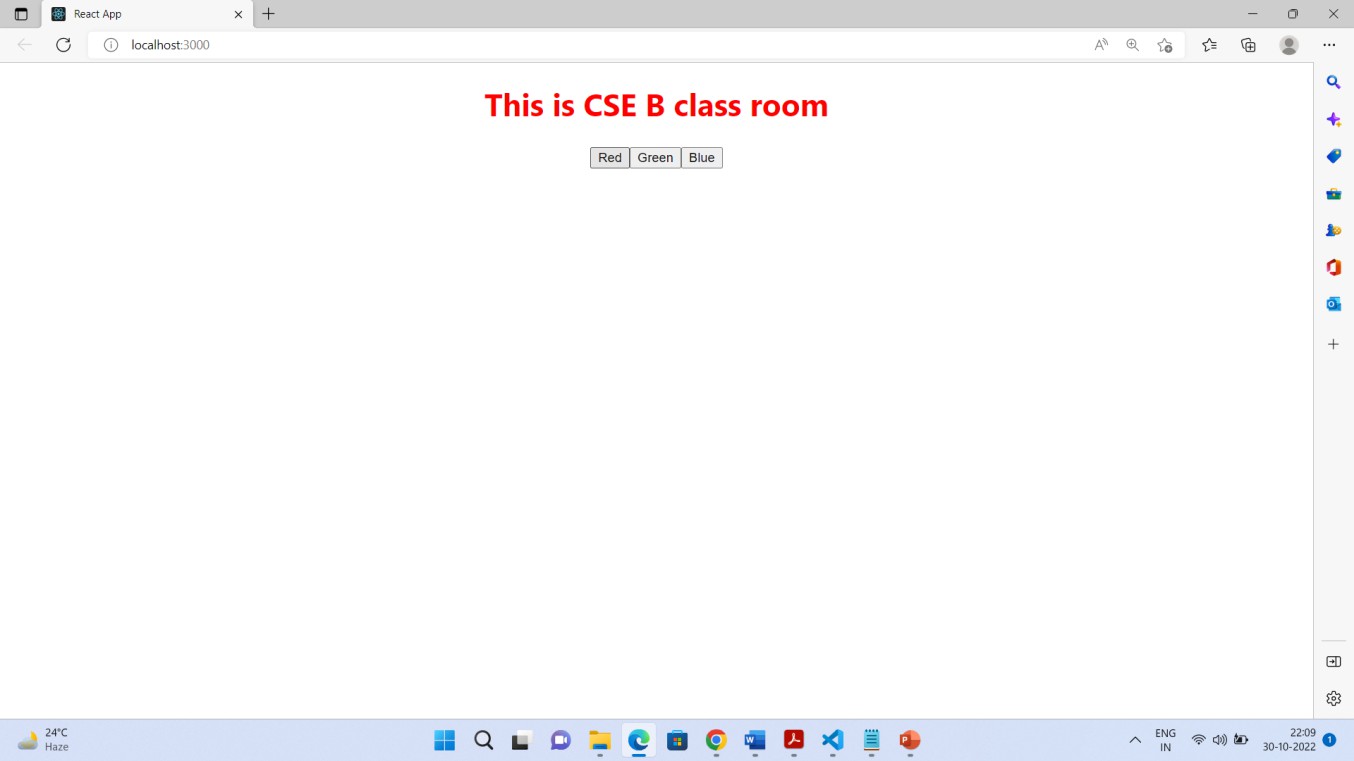
);

}

}

export default TextColorDemo;

**Output:**

****

# Exp. No. 8

**Problem Statement:** Create a class component for viewing an array of objects in tabular form.

**Description:** It stores multiple values in a single variable. The object can contain anything in the real world, such as person names, cars, game characters. Objects are very easy to use in some situations, if you know where the data is being processed. The character set of objects are known as Properties. Properties of an object can be called by using DOT(“.”) notation and (“[]”) notation. An array of objects can be represented as follows:

## Syntax:

var object = [

{

"property1":"value1", "property2":"value2"

}

]

## Source code:

import React from 'react';

function ArrayOfObjectsDemo() {

const items = [

{sid:'101', name:'Krishna', course:'CSE'},

{sid:'102', name:'Aadya', course:'CSE'},

{sid:'103', name:'Hrudya',course:'IT'},

{sid:'104', name:'Varun',course:'ECE'},

{sid:'105', name:'Manasa',course:'IT'},

{sid:'106', name:'Vijay', course:'CSE'}

];

return (

<div>

<h1>Array of Objects View - Demo</h1>

<table border="4" cellpadding="10px" align="center">

{

items.map((item)=> (

<tr key={item.id}>

<td>{item.sid}</td>

<td> {item.name} </td>

<td> {item.course} </td>

</tr>

))

}

</table>

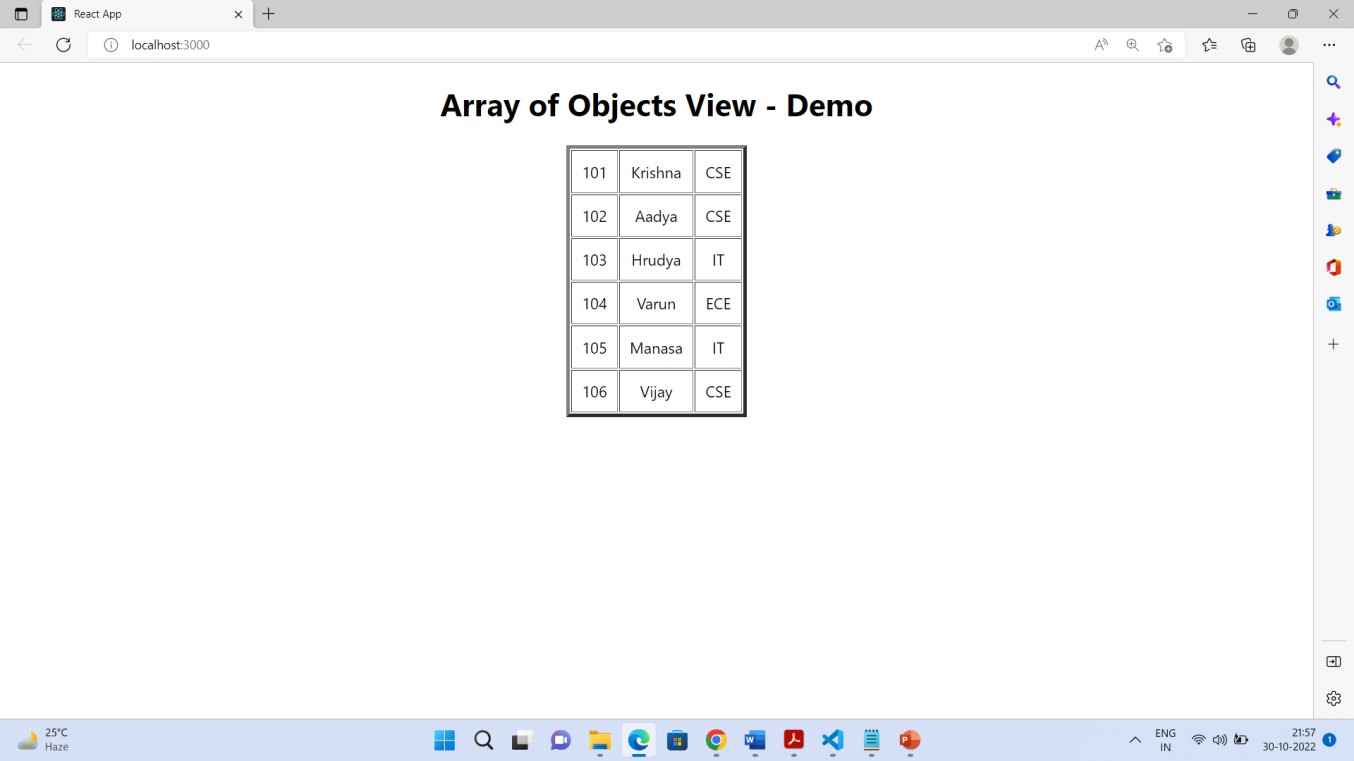
</div>

);

}

export default ArrayOfObjectsDemo;

**Output:**

****

# Exp No. 9

## Problem Statement: Demonstrate digital clock in React JS

**Description:** to implement the digital clock, one should understand life cycle methods of React in Class components.

## Life cycle API

Each React component has three distinct stages.

* **Mounting** − Mounting represents the rendering of the React component in the given DOM node.
* **Updating** − Updating represents the re-rendering of the React component in the given DOM node during state changes / updates.
* **Unmounting** − Unmounting represents the removal of the React component.

In the class component, React JS has the life cycle methods. Here, the **useEffect** is equivalent to **componentDidMount()**, **componentDidUpdate()**, **componentWillUnmount()** life-cycle methods as part of class component. Use Date() class object, setInterval() and clearInterval().

The setInterval() function is used to invoke a function or a piece of code repeatedly after a specific amount of time.

## Code:

import React, {Component} from "react";

export class DigitalClock extends Component { constructor(props) {

super(props); this.state = { time: ""

};

}

componentDidMount() { this.tick();

}

tick = () => {

const hours = new Date().getHours(); const minutes = new Date().getMinutes(); const seconds = new Date().getSeconds();

const updatedTime = `${hours}:${minutes}:${seconds}`; this.setState({ time: updatedTime });

};

componentDidUpdate(prevProps, prevState) { if (this.state.time !== prevState.time) { this.interval = setInterval(() => {

this.tick();

}, 1000);

}

}

componentWillUnmount() { clearInterval(this.interval);

}

render () { return <div>

<h1>Digital Clock</h1>

<h1>{this.state.time}</h1></div>;

}

}

export default DigitalClock;

**Output:**

****

# Exp. No 10

**Problem Statement:** Demonstrate useState Hook with the help sample text.

**Description:** useState is a React Hook that lets you add a state variable to your component.

Syntax: const [state, setState] = useState(initialState)

## Usage:

* Adding state to a component
* Updating state based on the previous state
* Updating objects and arrays in state
* Avoiding recreating the initial state
* Resetting state with a key
* Storing information from previous renders

## Code:

import React, {useState} from 'react'

function ViewItems() {

const [item, setItem] = useState(""); const [items, setItems] = useState([]);

const handleAddItem = () => { setItems([...items, {"item":item}]); setItem("");

};

const handleDelete = (item) => {

const fitems = items.filter((it) => it.item !== item.item); setItems([...fitems]);

};

const tdata = items.map((it, index) => (

<tr key={index}>

<td cellpadding="20px">

<span>{it.item}

</span>

</td>

<td>

<button onClick={() => handleDelete(it)}>x</button>

</td>

</tr>

), []);

return (

<div>

<input type="text" size="20" value={item}

onChange={(e) => setItem(e.target.value)}

/>

<button onClick={() => handleAddItem()}>Add</button>

<h1>useState() Demo</h1>

<h1>Add User List</h1>

<table border=" " align="center">{tdata}</table>

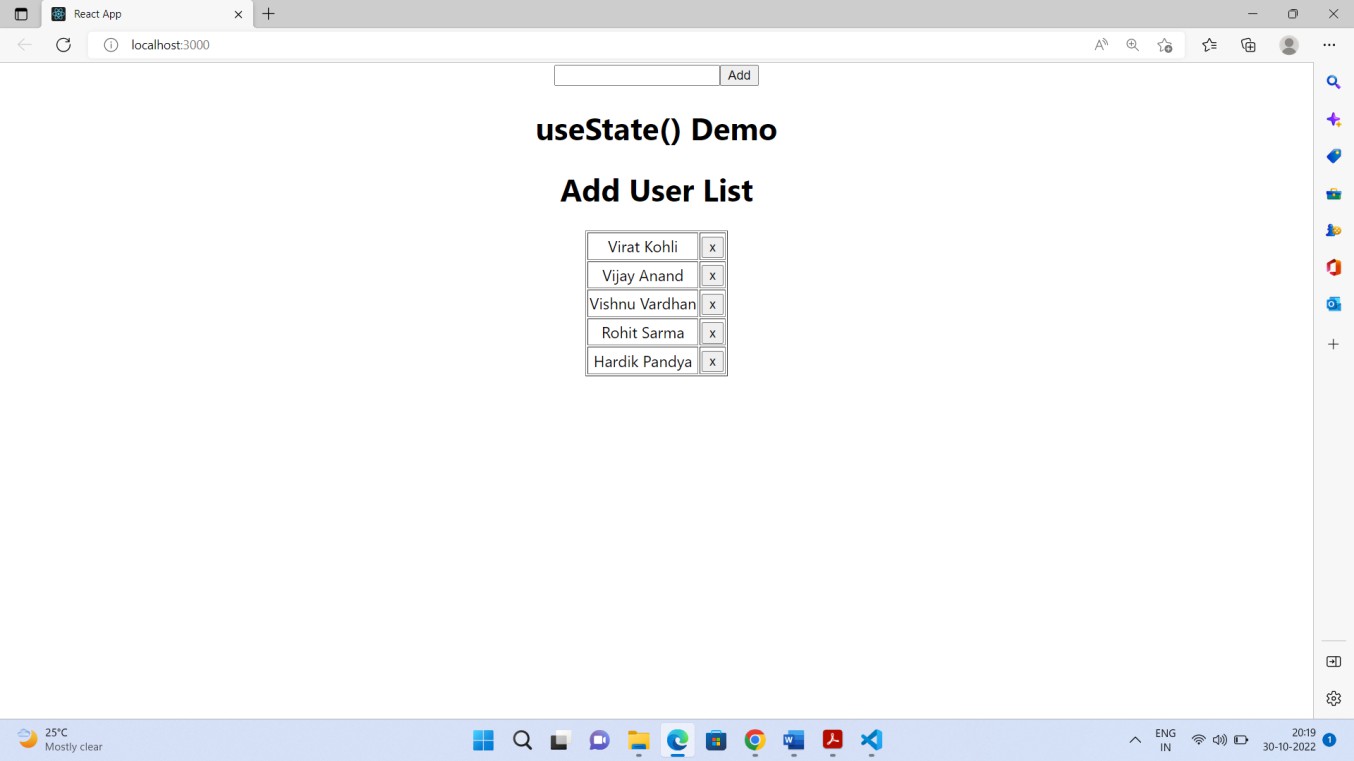
</div>

);

}

export default ViewItems;

**Output:**

****

# Exp. No. 11

**Problem Statement:** Demonstrate useEffect Hook with necessary example.

## Description:

The motivation behind the introduction of useEffect Hook is to eliminate the side-effects of using class-based components. For example, tasks like updating the DOM, fetching data from API end- points, setting up subscriptions or timers, etc can be led to unwarranted side-effects. Since the render method is quick to produce a side-effect, one needs to use life cycle methods to observe the side effects.

Common side effects include:

* Making a request to an API for data from a backend server
* To interact with browser APIs (that is, to use document or window directly)
* Using unpredictable timing functions like setTimeout or setInterval

## Code:

import React, {useState, useEffect} from 'react'; function DigiClock() {

const [mytime, getMytime] = useState(" "); const tick = () => {

let time = new Date().getHours()-12+":"+new Date().getMinutes()+":"+new Date().getSeconds();

if (new Date().getHours()< 12) time = time+" AM"

else

time = time+" PM"

getMytime(time);

}

useEffect(() => {

const t = setInterval(tick,1000);

return () => { clearTimeout(t);

}

}, [mytime])

return (

<div>

<h1>DigitalClock</h1>

<h2>{mytime}</h2>

</div>

)

}

export default DigiClock;

**Output:**

****

# Exp. No 12

**Problem Statement:** Demonstrate useContext Hook with necessary example.

**Description: useContext()** is one of the popular hooks in React JS.

useContext is a React Hook that lets you read and subscribe to context from your component. const value = useContext(SomeContext)

## Usage

* Passing data deeply into the tree
* Updating data passed via context
* Specifying a fallback default value
* Overriding context for a part of the tree
* Optimizing re-renders when passing objects and functions

## Reference

**useContext(SomeContext)**

**Code:** There are two files: App.js & BlogContextDemo.js

**App.js**

import React from 'react';

import BlogContextDemo from './BlogContextDemo'

const blogInfo = { React: {

post: "Learn useContext Hooks", author: "Varun K"

},

NodeJS: {

post: "Node Commands", author: "Veena M"

}

};

export const BlogContext = React.createContext(blogInfo); export default function App() {

return (

<div className="App">

<div>

<h1>useContext() Demo</h1>

<BlogContext.Provider value={blogInfo}>

<BlogContextDemo />

</BlogContext.Provider>

</div>

</div>

);

}

**BlogContextDemo.js**

import React, {useContext} from "react"; import {BlogContext} from './App';

function BlogContextDemo() {

const binfo = useContext(BlogContext); return (

<div>

<p><h2>Topic : {binfo.React.post}</h2></p>

<p><h2>Author : {binfo.React.author}</h2></p>

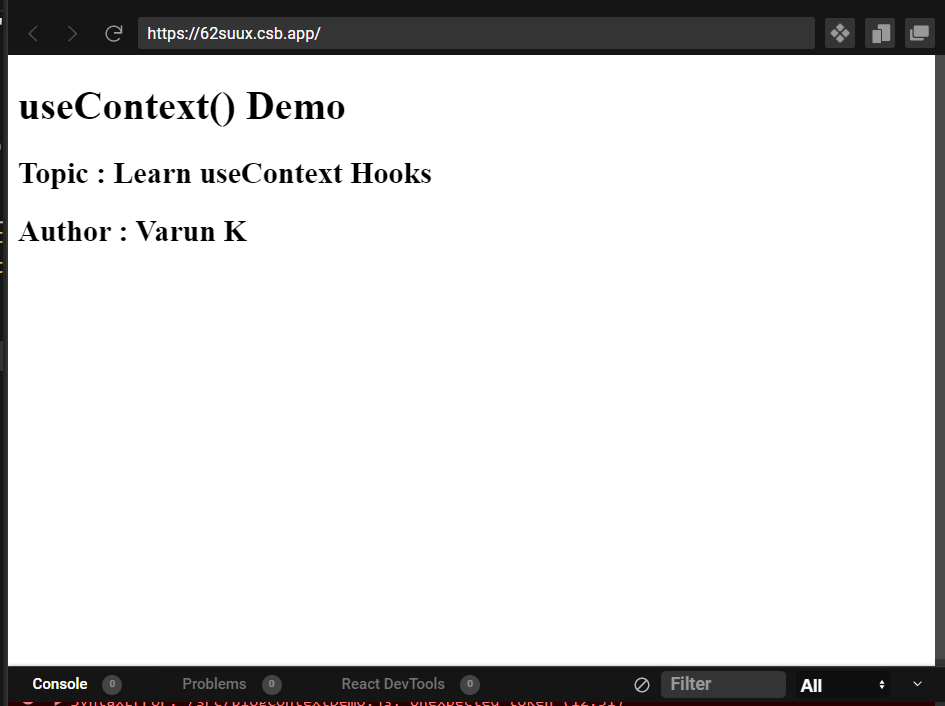
</div>

);

}

export default BlogContextDemo;

**Output:**

****

# Exp. No 13

**Problem Statement:** Demonstrate consuming web API using fetch &axios (AXIOS API). Demonstrate with the help of fake URL.

## Description:

**Using fetch Request:** The Fetch API through the fetch() method allows us to make an HTTP request to the backend. With this method, we can perform different types of operations using HTTP methods like the GET method to request data from an endpoint, POST to send data to an endpoint, and more. “fetch()” requires the URL of the resource we want to fetch and an optional parameter:

fetch(url, options) or

fetch(url, {

method: "GET" // default, so we can ignore

})

Ex:

useEffect(() => {

fetch(`https://jsonplaceholder.typicode.com/posts`)

.then((response) => console.log(response));

}, []);

## Code:

import { useEffect, useState } from "react";

export const FetchFunctional = () => { const [data, setData] = useState([]);

const getData = () => { fetch("https://jsonplaceholder.typicode.com/posts/")

.then((response) => response.json())

.then((result) => setData(result))

.catch((err) => console.log("Error occured", err));

};

useEffect(()=>{ getData();

},[]);

return (

<div>

<table border="1" cellpadding="10px" align="center">

<th>S.No</th><th>Posts</th>

{

data.map((d) => (

<tr key={d.id}>

<td align="center">{d.id}</td>

<td align="left">{d.title}</td>

</tr>

))

}

</table>

</div>

);

};

export default FetchFunctional;

**Using Axios.get():** Many projects on the web need to interface with a REST API at some stage in their development. Axios is a lightweight HTTP client based on the $http service.

Step-1:

npx create-react-app react-axios-example

To add Axios to the project, open your terminal and change directories into your project: cd react-axios-example

Then run this command to install Axios: npm install [axios@0.24.0](mailto:axios@0.24.0)

Step-2:

Making a GET Request

In this example, you create a new component and import Axios into it to send a GET request.

The user use axios.get(url) with a URL from an API endpoint to get a promise which returns a response object.

## Code:

import React, {useState, useEffect} from 'react'; import axios from 'axios'

function DataFetching() {

const [posts, setPosts] = useState([]);

useEffect(()=> { axios.get("https://jsonplaceholder.typicode.com/posts")

.then(res => { setPosts(res.data);

})

.catch(err => { console.log(err); })

}, [])

return(

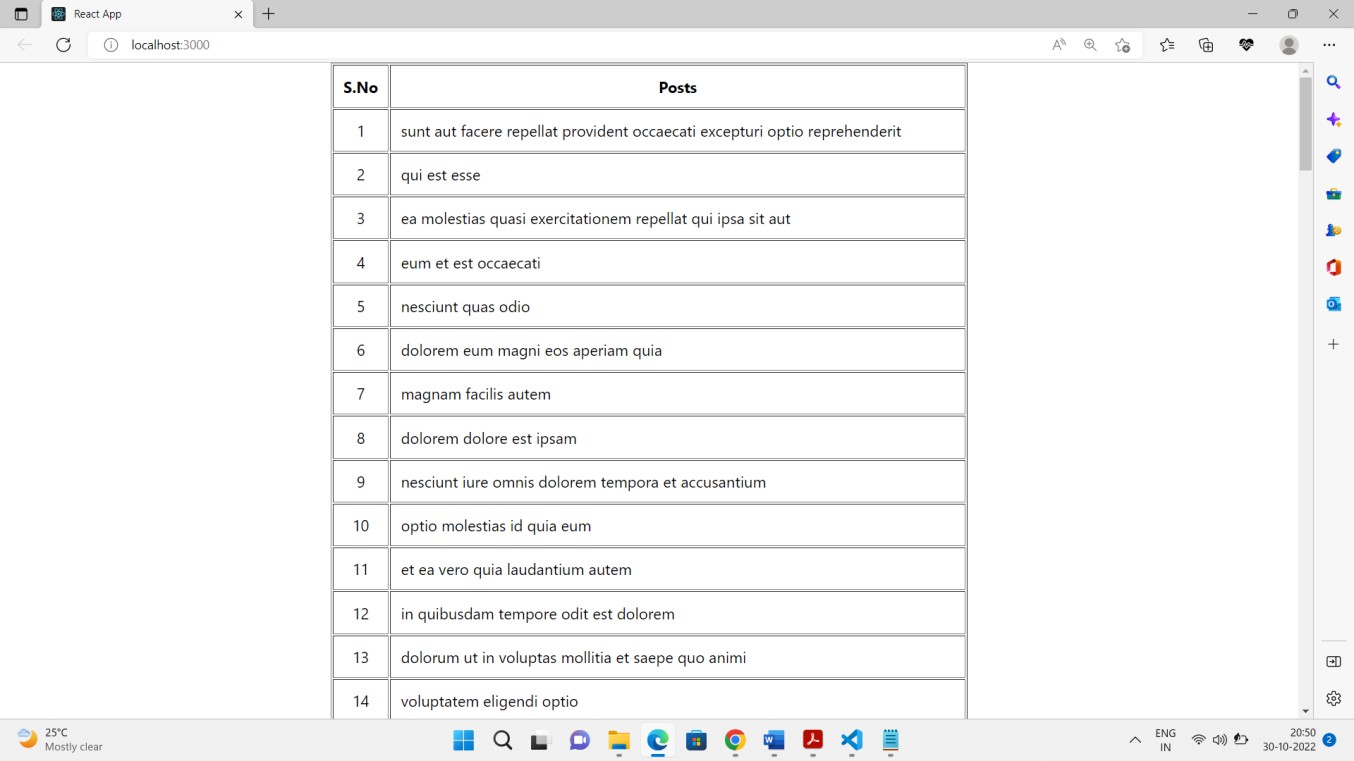
<div>

<h1> Data Fetching Demo </h1>

<ol>

{

posts.map( post => (



<li key={post.id}>

{post.id} - {post.title}

</li>

)) }

</ol>

</div>

);

}

export default DataFetching;

**Output:**

# Exp No. 14

**Problem Statement:** Design a BMI calculator using React JS based on the description given below:

BMI is a measurement of a person's leanness or corpulence based on their height and weight, and is intended to quantify tissue mass. It is widely used as a general indicator of whether a person has a healthy body weight for their height.

## Formula:

**Code:**

weight (kg) / [height (m)]2 (or)

[weight (kg) / height (cm) / height (cm)] x 10,000

**BMI table for adults:** This is the World Health Organization's (WHO) recommended body weight based on BMI values for adults. It is used for both men and women, age 18 or older.

|  |  |
| --- | --- |
| Category | BMI range - kg/m2 |
| Severe Thinness | < 16 |
| Moderate Thinness | 16 - 17 |
| Mild Thinness | 17 - 18.5 |
| Normal | 18.5 - 25 |
| Overweight | 25 - 30 |
| Obese Class I | 30 - 35 |
| Obese Class II | 35 - 40 |
| Obese Class III | > 40 |

import React, {useState} from 'react';

function BMICalcDemo() {

const [height,setHeight] = useState(0); const [weight, setWeight] = useState(0); const [name, setName] = useState("");

const calculateBMI = ()=> {

var heightSquared = (height/100 \* height/100); var bmi = weight / heightSquared;

if( bmi < 16)

window.alert("Hi.."+name+"...You are completely UnderWeight (Severe Thinner)"); else

if( bmi >=16 && bmi < 17)

window.alert("Hi.."+name+"...You are Moderately UnderWeight (Moderate Thinner)"); else

if( bmi >=17 && bmi < 18.5)

window.alert("Hi.."+name+"...You are little UnderWeight (Mild Thinness)"); else

if( bmi >= 18.5 && bmi <= 24.99 ){ window.alert("Hi.."+name+"...You are in a healthy weight range");

}

else if(bmi >= 25 && bmi <= 29.9){ window.alert("Hi.."+name+"...You are overweight");

}

else if(bmi >= 30){ window.alert("Hi.."+name+"...You are obese");

}

else if(bmi < 18.5){

window.alert("Hi.."+name+"...You are under weight");

}

bmi = Math.round(bmi \* 100) / 100;

}

const submitMe = (e) =>{ e.preventDefault(); calculateBMI();

}

const handleName = (e) =>{ setName(e.target.value);

}

const handleHeight = (e) =>{ setHeight(e.target.value);

}

const handleWeight = (e) =>{ setWeight(e.target.value);

}

return (

<div className="App">

<h1>BMI Calculator</h1>

<form onSubmit={submitMe}>

<label>

Please enter your name

</label>

<input type="text" name="name" value={name} onChange={handleName}/>

<br/><br/>

<label>

Enter your height in cm:

</label>

<input type="text" name="height" value={height} onChange={handleHeight} />

<br/><br/>

<label>

Enter your weight in kg :

</label>

<br/>

<input type="text" name="weight" value={weight} onChange={handleWeight}/>

<br/>

<br/>

<input type="submit" value="Submit"/>

</form>

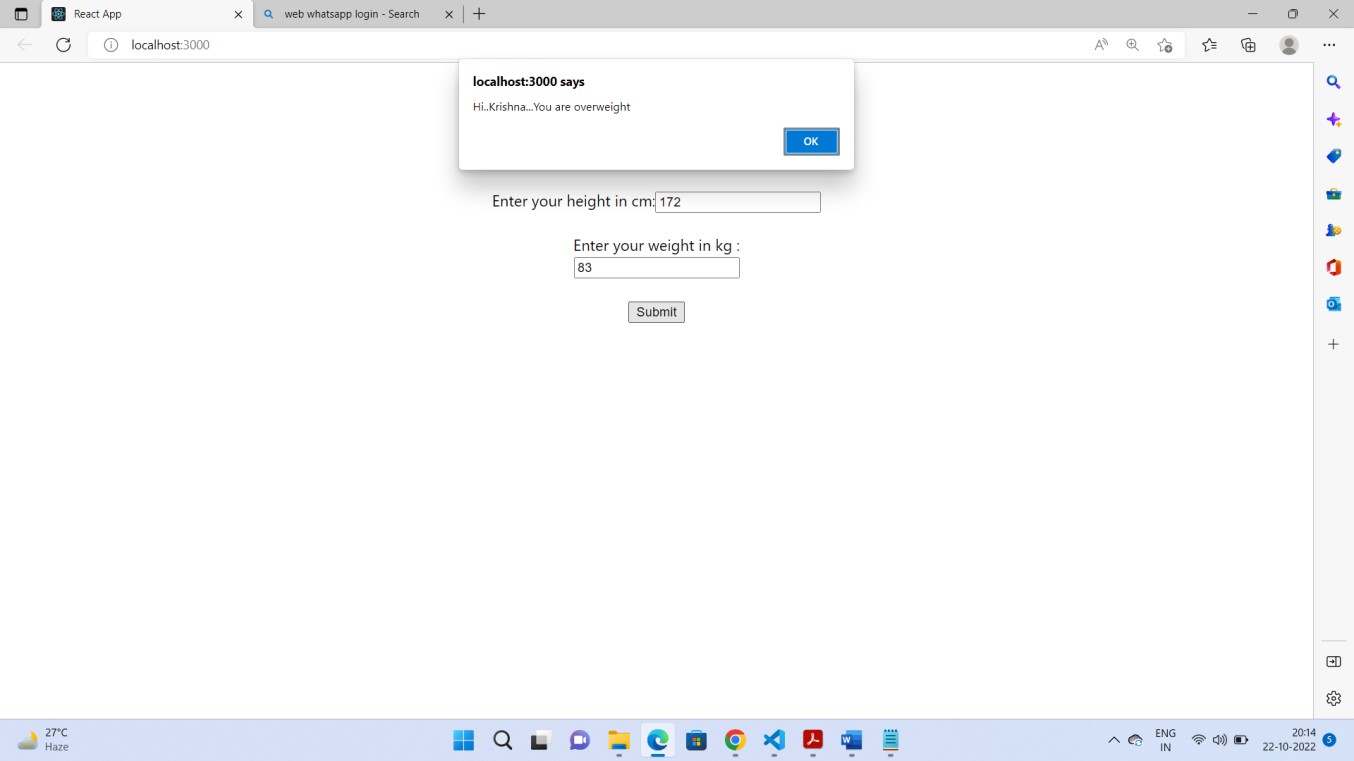
</div>

);

}

export default BMICalcDemo;

**Output:**

****

# Exp No. 15

**Problem Statement:** Display a selected set of images in tabular format using React JS.

**Description:** Here we try to display a set of images in a grid form with the help of image links provided as an array of objects.

## Code:

import React from "react"; import "./imageGrid.css"; const images = [

{

src: "https://images.unsplash.com/photo-1509420316987- d27b02f81864?dpr=1&auto=format&fit=crop&w=1500&q=80&cs=tinysrgb&ixid=dW5zcGxhc2gu Y29tOzs7Ozs%3D",

width: 450,

height: 500,

},

{

src: "https://images.unsplash.com/photo-1509641498745- 13c26fd1ed89?dpr=1&auto=format&fit=crop&w=1000&q=80&cs=tinysrgb&ixid=dW5zcGxhc2gu Y29tOzs7Ozs%3D",

width: 450,

height: 500,

},

{

src: "https://images.unsplash.com/photo-1491146179969- d674118945ff?dpr=1&auto=format&fit=crop&w=1500&q=80&cs=tinysrgb&ixid=dW5zcGxhc2gu Y29tOzs7Ozs%3D",

width: 450,

height: 500,

},

{

src: "https://images.unsplash.com/photo-1509420316987- d27b02f81864?dpr=1&auto=format&fit=crop&w=1500&q=80&cs=tinysrgb&ixid=dW5zcGxhc2gu Y29tOzs7Ozs%3D",

width: 450,

height: 500,

},

{

src: "https://images.unsplash.com/photo-1509641498745- 13c26fd1ed89?dpr=1&auto=format&fit=crop&w=1000&q=80&cs=tinysrgb&ixid=dW5zcGxhc2gu Y29tOzs7Ozs%3D",

width: 450,

height: 500,

},

];

const ImageGridDemo = () => { return (

<div> ImageGridDemo

<div id="img-wrapper">

{images.map((image,index) => { return (

<div key={index}>

<img src={image.src} height='200px' width='300px' alt=""

/>

</div>

);

})}

</div>

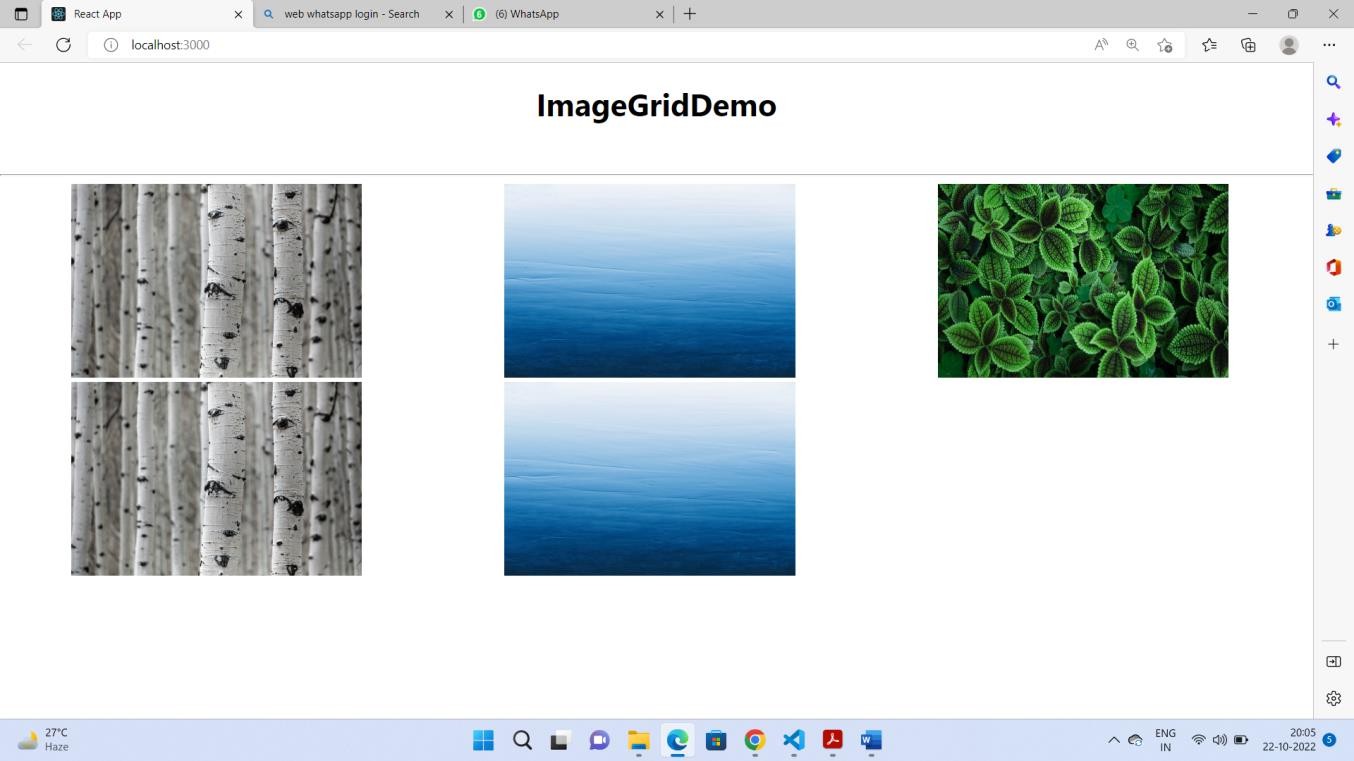
</div>

);

};

export default ImageGridDemo;

**Output:**

****

# Exp No 16

**Problem Statement: Implement upload & download a file using React JS. Description:** Here we have two options: upload & download.

## Downloading the file

With HTML5, some prior complications are eliminated. Eg, the download attribute of the anchor element is easily used to set the suggested filename. Here’s the pattern:

1. The user initiates the download via a button connected to a JavaScript method.
2. The data is converted to the output format. The result is a string.
3. A Blob is created from the string.
4. An Object URL is created from the Blob using the URL.createObjectURL method.
5. A hidden anchor element’s href attribute is set to the Object URL.
6. The anchor element’s click method is invoked. Normally the click method is invoked when the user clicks on the element. In this case, we programmatically click the element so the user only needs to initiate the download in step 1.
7. After the click method completes, the Object URL can be freed.

## Uploading a file to the app

Instead of uploading a file to the server, we can upload the file to the application running in the browser. The application can process the file locally. The app can further upload the file to the server via Ajax if desired.

## Code:

class App extends React.Component { constructor(props) {

super(props)

const defaultFileType = "json"; this.fileNames = {

json: "states.json", csv: "states.csv", text: "states.txt"

}

this.state = {

fileType: defaultFileType, fileDownloadUrl: null, status: "",

data: [

{state: "Arizona", electors: 11 },

{state: "Florida", electors: 29 },

{ state: "Iowa", electors: 6 },

{ state: "Michigan", electors: 16 },

{ state: "North Carolina", electors: 15 },

{ state: "Ohio", electors: 18 },

{ state: "Pennsylvania", electors: 20 },

{ state: "Wisconsin", electors: 10 },

]

}

this.changeFileType = this.changeFileType.bind(this); this.download = this.download.bind(this);

this.upload = this.upload.bind(this); this.openFile = this.openFile.bind(this);

}

changeFileType (event) {

const value = event.target.value; this.setState({fileType: value});

}

download (event) { event.preventDefault();

// Prepare the file let output;

if (this.state.fileType === "json") {

output = JSON.stringify({states: this.state.data}, null, 4);

} else if (this.state.fileType === "csv"){

// Prepare data:

let contents = [];

contents.push (["State", "Electors"]); this.state.data.forEach(row => {

contents.push([row.state, row.electors])

});

output = this.makeCSV(contents);

} else if (this.state.fileType === "text"){

// Prepare data:

output = ''; this.state.data.forEach(row => {

output += `${row.state}: ${row.electors}\n`

});

}

// Download it

const blob = new Blob([output]);

const fileDownloadUrl = URL.createObjectURL(blob); this.setState ({fileDownloadUrl: fileDownloadUrl},

() => {

this.dofileDownload.click();

URL.revokeObjectURL(fileDownloadUrl); // free up storage--no longer needed. this.setState({fileDownloadUrl: ""})

})

}

makeCSV (content) { let csv = '';

content.forEach(value => { value.forEach((item, i) => {

let innerValue = item === null ? '' : item.toString(); let result = innerValue.replace(/"/g, '""');

if (result.search(/("|,|\n)/g) >= 0) { result = '"' + result + '"'

}

if (i > 0) {csv += ','} csv += result;

})

csv += '\n';

})

return csv

}

upload() {

event.preventDefault(); this.dofileUpload.click()

}

openFile(evt) {

let status = []; // Status output const fileObj = evt.target.files[0]; const reader = new FileReader();

let fileloaded = e => {

// e.target.result is the file's content as text const fileContents = e.target.result;

status.push(`File name: "${fileObj.name}". Length: ${fileContents.length} bytes.`);

// Show first 80 characters of the file

const first80char = fileContents.substring(0,80);

status.push (`First 80 characters of the file:\n${first80char}`)

this.setState ({status: status.join("\n")})

}

// Mainline of the method fileloaded = fileloaded.bind(this); reader.onload = fileloaded; reader.readAsText(fileObj);

}

render() { return (

<div>

<h2>2020 US Swing States</h2>

<table>

<thead>

<tr><th>State</th><th>Electors</th></tr>

</thead>

<tbody>

{this.state.data.map(item => (

<tr key={item.state}>

<td>{item.state}</td><td>{item.electors}</td>

</tr>

))}

</tbody>

</table>

<form>

<span className="mr">File type:</span>

<select name="fileType" onChange={this.changeFileType} value={this.state.fileType} className="mr"

>

<option value="csv">CSV</option>

<option value="json">JSON</option>

<option value="text">Text</option>

</select>

<button onClick={this.download}> Download the file!

</button>

<a className="hidden" download={this.fileNames[this.state.fileType]} href={this.state.fileDownloadUrl} ref={e=>this.dofileDownload = e}

>download it</a>

<p><button onClick={this.upload}> Upload a file!

</button> Only json, csv, and text files are ok.</p>

<input type="file" className="hidden" multiple={false}

accept=".json,.csv,.txt,.text,application/json,text/csv,text/plain" onChange={evt => this.openFile(evt)} ref={e=>this.dofileUpload = e}

/>

</form>

<pre className="status">{this.state.status}</pre>

</div>

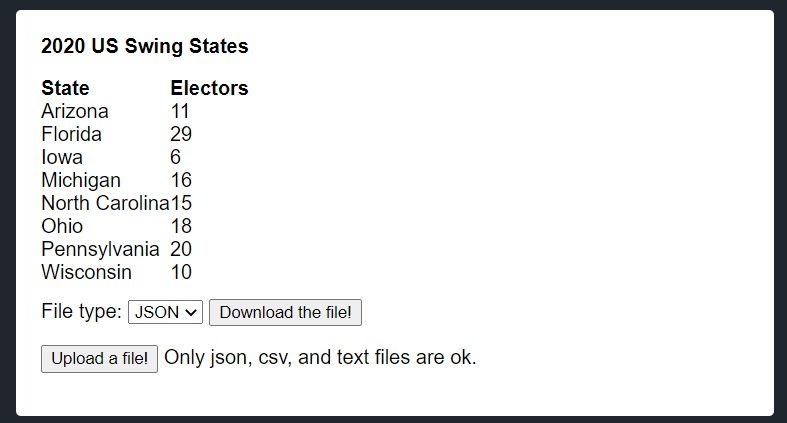
)

}

}

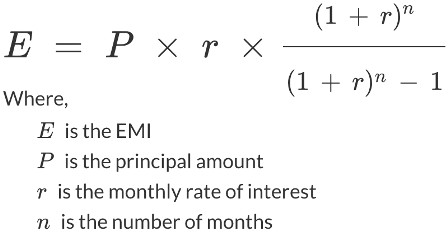
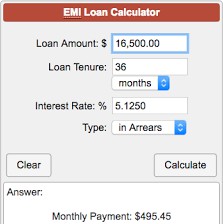
export default App;

**Output:**

****

# Exp No. 17

**Problem Statement:** Create a React application to view EMI calculator. A specific view is given below:



## Description:

According to the formula given, here we are supposed to calculate the EMI loan with the help of given formula. A proper UI should be designed with event handling. To perform the state management, useState() hook is mandatory.

## Code:

import React, {useState} from "react";

function EMICalc() {

const [loan, setLoan] = useState(0); const [tenure, setTenure] = useState(0); const [rate, setRate] = useState(0);

const [type, setSelectType] = useState(" "); const [emi, setEMI] = useState(0);

const calculateEMI = () => {

var em = loan \* rate \* ((Math.pow((1+rate),tenure)/(Math.pow((1+rate),tenure)-1))); setEMI(em);

};

const submitMe = (e) => { e.preventDefault(); calculateEMI();

};

const myselect = () => {

var item = document.getElementById("tenureType"); setSelectType(item.options[item.selectedIndex].text);

console.log(type);

}

const handleLoanAmount = (e) => { setLoan(e.target.value);

};

const handleTenure = (e) => { setTenure(e.target.value);

};

const handleInterestRate = (e) => { setRate(e.target.value);

};

const clear = () => {

}

return (

<div className="App">

<h1>EMI Calculator</h1>

<table border="0px" align="Center" style = {{width: "50%"}}>

<tr>

<td> <label>Loan Amount: Rs </label></td>

<td> <input type="text" name="loan" value={loan} onChange={handleLoanAmount}/></td>

</tr>

<tr>

<td><label> Loan Tenure: </label></td>

<td><input type="text" name="name" value={tenure} onChange={handleTenure} /></td>

</tr>

<tr>

<td></td>

<td>

<select name="tenureType" id="tenureType" onChange={myselect}>

<option value="Months">Months</option>

<option value="Years">Years</option>

</select>

</td>

</tr>

<tr>

<td><label> Interest Rate: </label> </td>

<td> <input type="text" name="rate" value={rate} onChange={handleInterestRate}

/></td>

</tr>

<tr>

<td></td>

</tr>

<tr>

<td>

<button onClick={clear}>Clear</button>

</td>

<td>

<button onClick={submitMe}>Calculate</button>

</td>

</tr>

</table>

<div id="result">The Every Monthly Installment (EMI):{emi}</div>

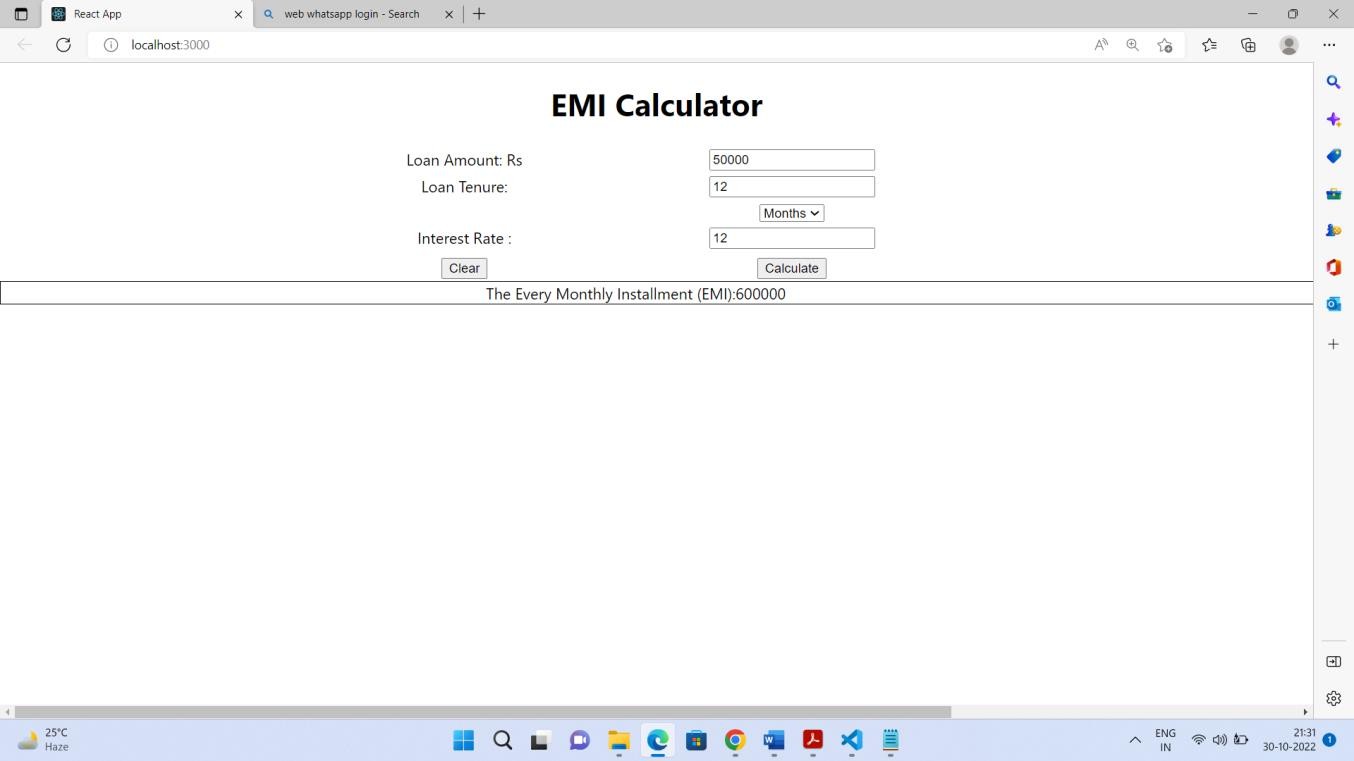
</div>

);

}

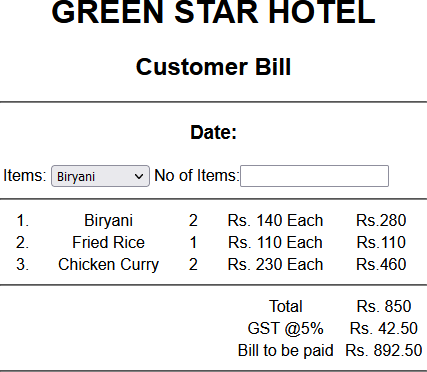
export default EMICalc;

**Output:**

****

# Exp No. 18

**Problem Statement:** Design the following Hotel bill screen. User can select as many items as possible from the dropdown box and is allowed to enter in the text field provided. Each transaction must be added in the table given below along with the bill amount.



## Description:

Here we are supposed to use useState() & useRef() hooks while performing event handling. A little calculation is required to implement the resultant UI. The UI design is given more priority here.

## Code:

import { useRef, useState } from "react";

function HotelBillDemo() { const selectRef = useRef();

const [quantity, setQuantity] = useState(); const [listItems, setListItems] = useState([]); const [name, setName] = useState("");

const menu = [

{ itemName: "Briyani", price: 210 },

{ itemName: "Fried Rice", price: 180 },

{ itemName: "Manchuria", price: 120 },

{ itemName: "Gobi Manchurian", price: 130},

{ itemName: "Baby Corn Masala", price: 130},

];

const sdate = new Date();

const handleChange = (e) => { e.preventDefault(); setName(e.target.value); setQuantity("");

};

const findSum = () =>{

return listItems?.reduce((acc,item)=> acc+item.totalPrice, 0);

}

const findDuplicate = (name) => {

const index = listItems.findIndex((item)=>{ return item.name === name

})

return index;

};

const handleAddItem = (e) => { e.preventDefault();

const item = menu.find((item) => { return item.itemName === name;

});

const itemPrice = item.price

if (name !== "Please Select") { setQuantity(e.target.value);

const index = findDuplicate(name);

if (listItems.length === 0)

setListItems([{ name, noOfItems: parseInt(e.target.value), totalPrice: itemPrice \* e.target.value}]);

else

if(index!== -1){

const myList = [...listItems];

myList[index].noOfItems = parseInt(myList[index].noOfItems) + parseInt(e.target.value); myList[index].totalPrice = myList[index].noOfItems \* itemPrice;

setListItems(myList);

}

else

setListItems([...listItems, { name, noOfItems: parseInt(e.target.value), totalPrice: itemPrice \* e.target.value }]);

}

};

const handleDelete = (item) => {

const fitems = listItems.filter((it) => it !== item); setQuantity('');

setListItems([...fitems]);

};

const tdata = listItems.map((it, index) => (

<tr key={index}>

<td>

{" "}

<span>{it.name} </span>{" "}

</td>

<td>

{" "}

<span>{it.noOfItems} </span>{" "}

</td>

<td>

{" "}

<span>{it.totalPrice} </span>{" "}

</td>

<td>

<button onClick={(e) => {e.preventDefault(); handleDelete(it);}}>x</button>

</td>

</tr>

));

return (

<div className="App">

<h1>GREEN STAR HOTEL</h1>

<h2>Customer Bill</h2>

<h3>

{" "}

Date:{sdate.getDate()}-{sdate.getMonth()}-{sdate.getYear() + 1900}

</h3>

<br />

<hr />

<label for="Items">Items:</label>

<select name="itemList" id="itemList" ref={selectRef} onChange={handleChange}>

<option value='Please Select'>Please Select</option>

{

menu.map((item) => {

return <option value={item.itemName}>{item.itemName}</option>;

})

}

</select> No of Items:

<input type="number" onChange={handleAddItem} value={quantity} />

<hr />

<table align="center" border="1" cellPadding="10%" cellSpacing="10px">

<tr><th>Item Name</th><th>No of Items</th><th>Price</th></tr>

{tdata}

<tr> <td colspan="4">Bill Amount : {findSum()}</td> </tr>

<tr> <td colspan="4">GST (5%): {(0.05\*findSum())}</td> </tr>

<tr> <td colspan="4">Amount to be Paid : {findSum()+(0.05\*findSum())}</td> </tr>

</table>

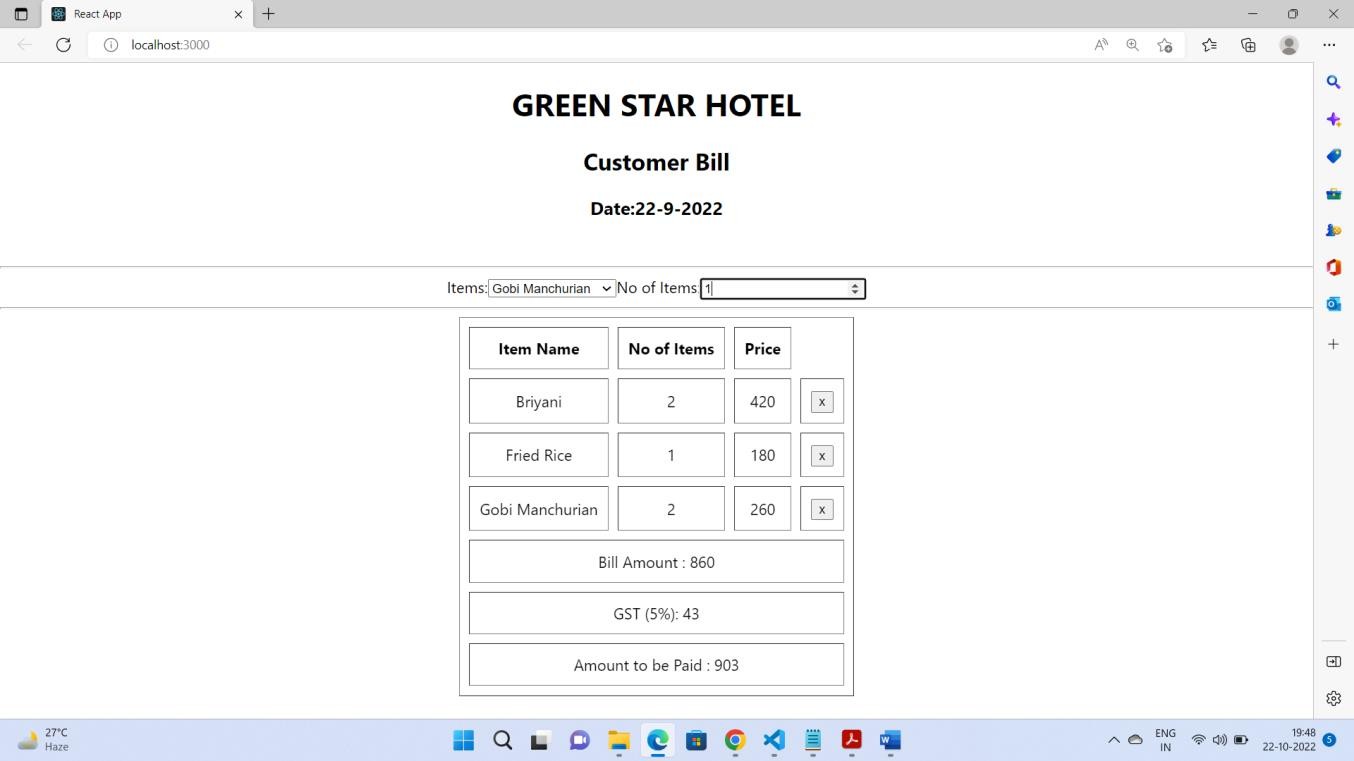
</div>

);

}

export default HotelBillDemo;

**Output:**

****

# Exp No. 19

**Problem Statement:** Demonstrate the procedure to create a schema in MongoDB.

MongoDB is a document-oriented open-source NoSQL database. It is one of the most popular and widely used NoSQL databases. In this tutorial, you will learn how to create a new MongoDB database, or switch to an existing one.

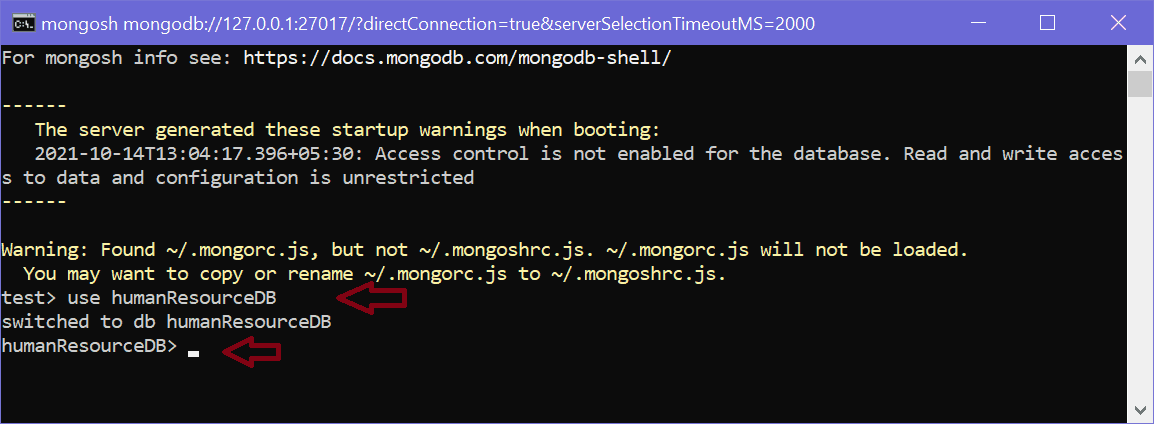
A database is a place where data is stored in an organized way. In MongoDB, databases are used to store collections. A single MongoDB server can have multiple databases and a single MongoDB database can have multiple collections.

You can use MongoDB Shell or MongoDB Compass to create a new database.

MongoDB provides the use <database-name command to connect with the database. If the specified database name does not exist then it creates it and set it as a current database.

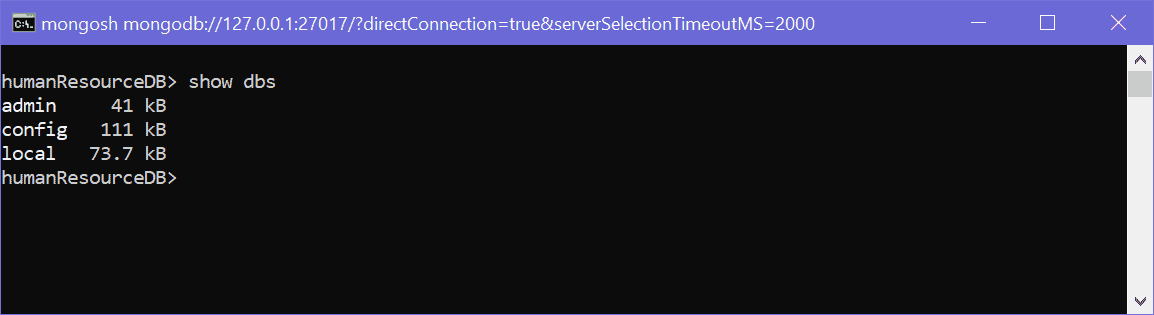
For example, the following command switch to the "humanResouredb" database. If it does not exist then creates it.

use humanResourceDB



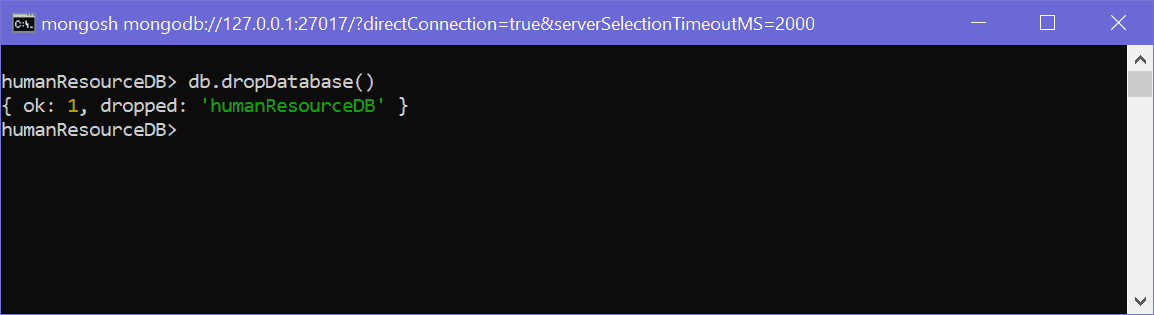
MongoDB will automatically switch to the newly created database. Notice that it promts to humanResourceDB> now.

To check all the databases, use the "show dbs" command, as shown below.



As you can see above, the "admin", "config", and "local" are default databases. As of now, "humanResourceDB" is not visible. This is because there is no collection in it.

To delete a database, use the db.dropDatabase() method which deletes a current database.

[](https://www.tutorialsteacher.com/Content/images/mongodb/create-db4.png)

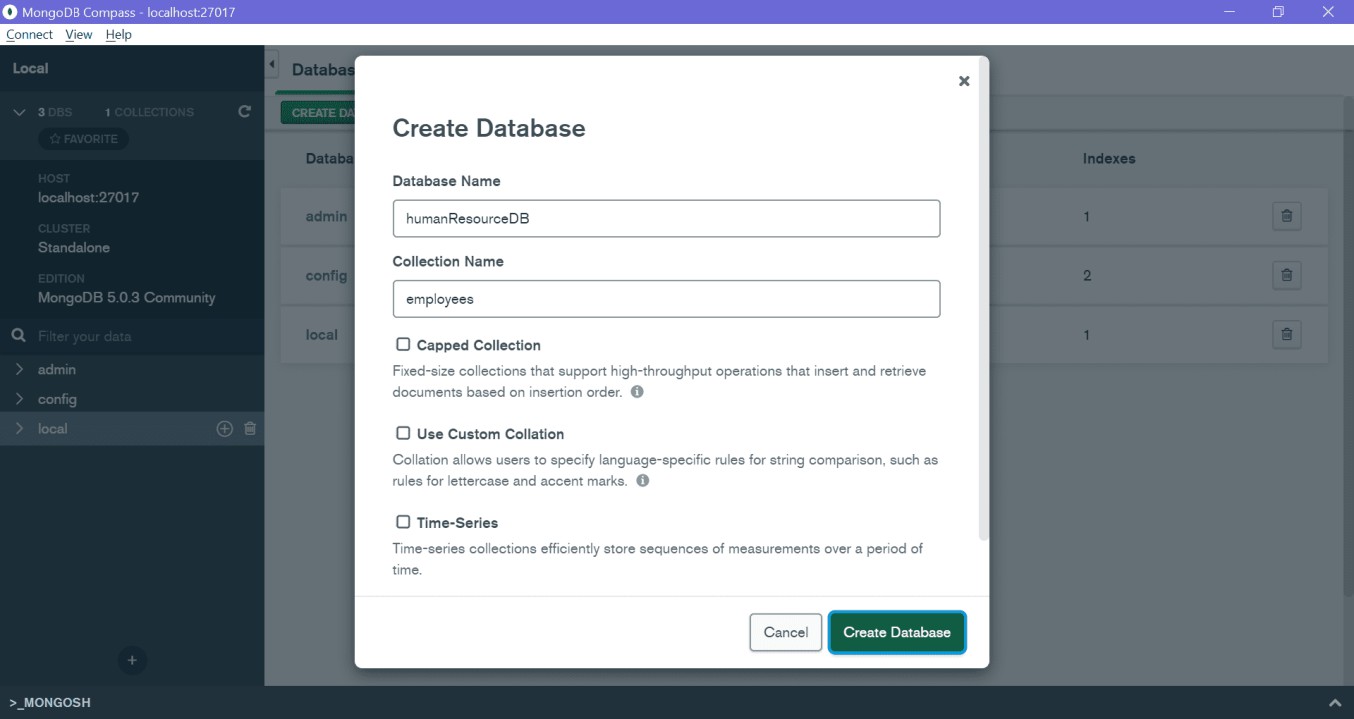
Delete Database

Above, { ok: 1, dropped: 'humanResourcedb' } indicates that the database deleted successfully.

Note: Method names are case sensitive. So, executing db.dropdatabase() will throw an error.

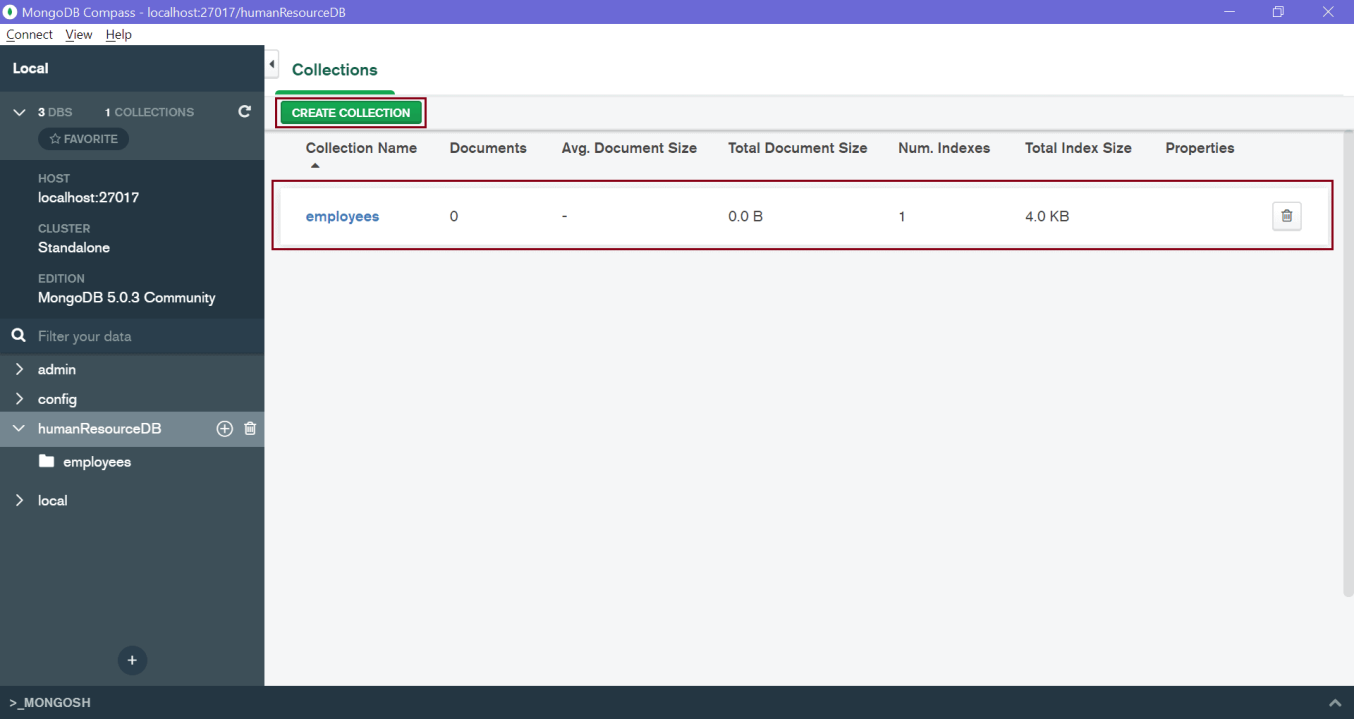
Create Database using MongoDB Compass

You can create a new database using MongoDB Compass. For that, open Compass and connect with your local or remote database. Once it connects with the MongoDB server, click on the top "CREATE DATABASE" button which will open the popup window, as shown below.

[](https://www.tutorialsteacher.com/Content/images/mongodb/compass4.png)

MongoDB Compass - Create Database

Enter your database name and collection name and click Create Database. This will create a new database humanResourceDB with the new employees collection shown below.

[](https://www.tutorialsteacher.com/Content/images/mongodb/compass5.png)

MongoDB Compass - Collections

Thus, you can create a new database in MongoDB.

# Exp No. 20

**Problem Statement:** Demonstrate CRUD operations using MongoDB.

## Description:

**MongoDB CRUD Operations**

CRUD operations *create*, *read*, *update*, and *delete* documents.

## Create Operations

Create or insert operations add new documents to a collection. If the collection does not currently exist, insert operations will create the collection.

MongoDB provides the following methods to insert documents into a collection:

* *New in version 3.2*

db.collection.insertOne()

* *New in version 3.2*

db.collection.insertMany()

In MongoDB, insert operations target a single collection. All write operations in MongoDB are atomic on the level of a single document.

For examples, see Insert Documents.

## Read Operations

Read operations retrieve documents from a collection; i.e. query a collection for documents. MongoDB provides the following methods to read documents from a collection:

[db.collection.find()](https://www.mongodb.com/docs/manual/reference/method/db.collection.find/#mongodb-method-db.collection.find)

You can specify [query filters or criteria](https://www.mongodb.com/docs/manual/tutorial/query-documents/#std-label-read-operations-query-argument) that identify the documents to return.

## Update Operations

Update operations modify existing [documents](https://www.mongodb.com/docs/manual/core/document/#std-label-bson-document-format) in a [collection](https://www.mongodb.com/docs/manual/core/databases-and-collections/#std-label-collections). MongoDB provides the following methods to update documents of a collection:

* *New in version 3.2*

db.collection.updateOne()

* *New in version 3.2*

db.collection.updateMany()

* *New in version 3.2*

db.collection.replaceOne()

In MongoDB, update operations target a single collection. All write operations in MongoDB are [atomic](https://www.mongodb.com/docs/manual/core/write-operations-atomicity/) on the level of a single document.

You can specify criteria, or filters, that identify the documents to update. These filters use the same syntax as read operations.

For examples, see Update Documents.

## Delete Operations

Delete operations remove documents from a collection. MongoDB provides the following methods to delete documents of a collection:

* *New in version 3.2*

db.collection.deleteOne()

* *New in version 3.2*

db.collection.deleteMany()

In MongoDB, delete operations target a single collection. All write operations in MongoDB are atomic on the level of a single document.

You can specify criteria, or filters, that identify the documents to remove. These filters use the same syntax as read operations.

**References:**

1. W3schools for HTML & CSS
2. JavaScript Tutorial (javascripttutorials.net)
3. Java script Docs
4. MDN Docs for React JS
5. https://javascript.plainenglish.io/how-to-create-download-and-upload-files-in-react-apps- 80893da4247a
6. MongoDB: tutorialteachers, geeks for geeks

\*\*\*\*\*@\*\*\*\*\*