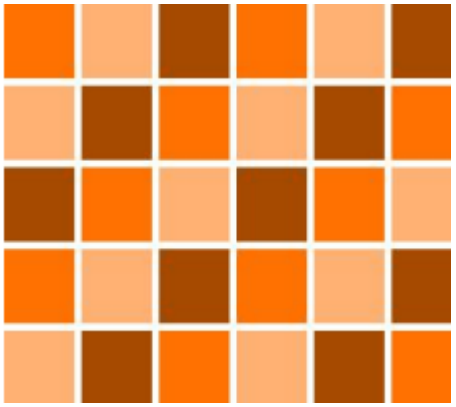
EXPERIMENT NO: 01

AIM:

Try to recreate the following patterns using HTML and CSS only.



DESCRIPTION:

1. `<html>`:
    - Defines the root of an HTML document.
    - Contains the entire content of the document.
2. `<head>`:
    - Contains meta-information about the document.
    - Includes elements like `<title>`, `<meta>`, and `<link>`.
3. `<body>`:
    - Contains the visible content of the document.
    - Includes text, images, links, and other elements.
4. `<h1>` to `<h6>`:
    - Defines headings of different levels.
    - `<h1>` is the highest level, `<h6>` is the lowest.
5. `<p>`:
    - Defines a paragraph.
    - Used for structuring text content.
6. `<a>`:
    - Creates a hyperlink to another web page or resource.
    - Uses the `href` attribute to specify the link target.
7. `<img>`:
    - Inserts an image into the document.
    - Uses the `src` attribute to specify the image source.
8. `<ul>`:
    - Defines an unordered list.
    - Contains `<li>` (list item) elements.
9. `<ol>`:
    - Defines an ordered list.
    - Contains `<li>` elements with a sequence number.
10. `<li>`:
    - Represents a list item.
    - Used within `<ul>` or `<ol>` elements.
11. `<div>`:
    - Defines a division or a section in an HTML document.
    - Used for grouping and styling content.
12. `<span>`:
    - Defines an inline container for text.
    - Often used for styling or scripting purposes.

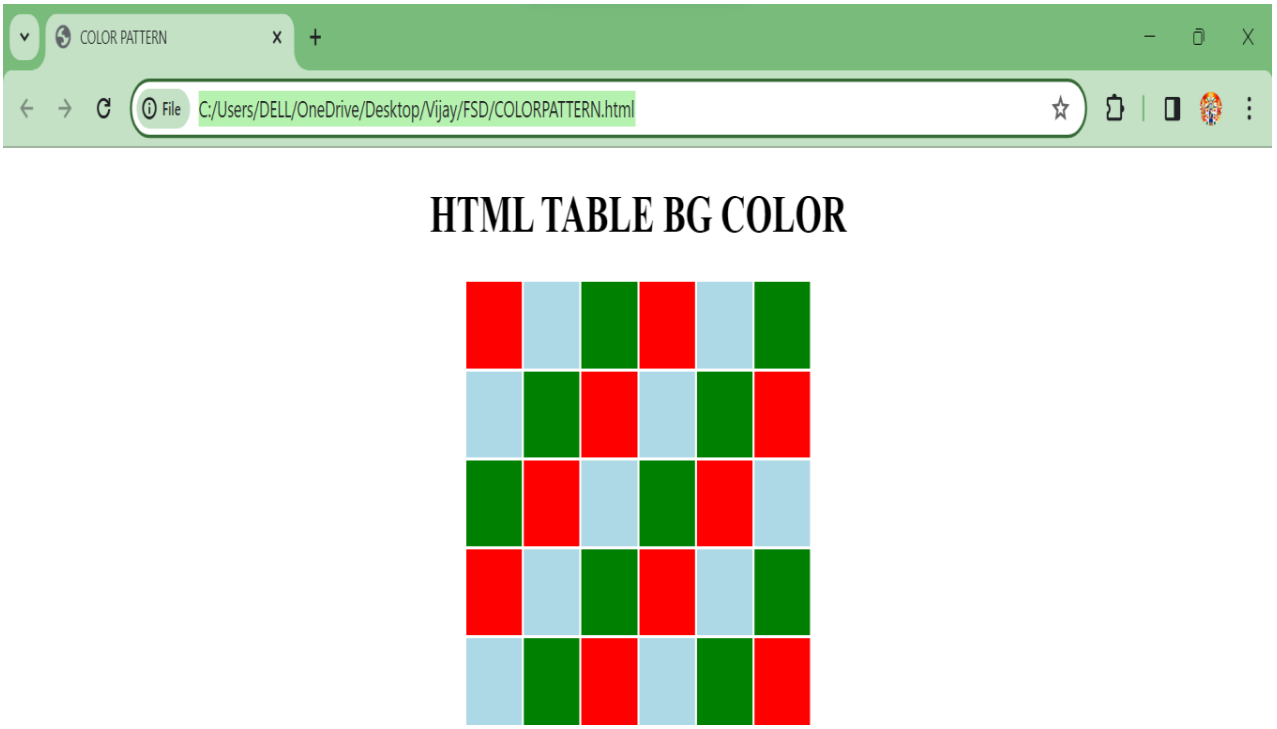
13. `<table>`:
    - Defines a table.
    - Contains `<tr>`, `<td>`, and `<th>` for rows, cells, and headers.
14. `<tr>`:
    - Represents a row in a table.
    - Contains `<td>` or `<th>` elements.
15. `<td>`:
    - Defines a standard cell in a table.
    - Contains data or content within a table row.
16. `<th>`:
    - Defines a header cell in a table.
    - Used for column or row headers.
17. `<form>`:
    - Creates an interactive form for user input.
    - Contains input fields, buttons, and other form elements.
18. `<input>`:
    - Defines an input field within a form.
    - Various types like text, password, checkbox, radio, etc.
19. `<button>`:
    - Creates a clickable button.
    - Can trigger actions or submit forms.
20. `<textarea>`:
    - Defines a multi-line text input field.
    - Used for longer text input by users.

PROGRAM:

```
<!DOCTYPE html>
<html>
  <head>
    <title>COLOR PATTERN</title>
  </head>
  <body align="center">
    <h1>HTML TABLE BG COLOR</h1>
    <table style="table-layout:fixed; width=100%" border="0" align="center">
      <tr>
        <td width="50" height="50" bgcolor="red"></td>
        <td width="50" height="50" bgcolor="lightblue"></td>
        <td width="50" height="50" bgcolor="green"></td>
        <td width="50" height="50" bgcolor="red"></td>
        <td width="50" height="50" bgcolor="lightblue"></td>
        <td width="50" height="50" bgcolor="green"></td>
      </tr>

      <tr>
        <td width="50" height="50" bgcolor="lightblue"></td>
        <td width="50" height="50" bgcolor="green"></td>
        <td width="50" height="50" bgcolor="red"></td>
        <td width="50" height="50" bgcolor="lightblue"></td>
        <td width="50" height="50" bgcolor="green"></td>
        <td width="50" height="50" bgcolor="red"></td>
      </tr>
```

```
        <tr>
          <td width="50" height="50" bgcolor="green"></td>
          <td width="50" height="50" bgcolor="red"></td>
          <td width="50" height="50" bgcolor="lightblue"></td>
          <td width="50" height="50" bgcolor="green"></td>
          <td width="50" height="50" bgcolor="red"></td>
          <td width="50" height="50" bgcolor="lightblue"></td>
        </tr>

        <tr>
          <td width="50" height="50" bgcolor="red"></td>
          <td width="50" height="50" bgcolor="lightblue"></td>
          <td width="50" height="50" bgcolor="green"></td>
          <td width="50" height="50" bgcolor="red"></td>
          <td width="50" height="50" bgcolor="lightblue"></td>
          <td width="50" height="50" bgcolor="green"></td>
        </tr>

        <tr>
          <td width="50" height="50" bgcolor="lightblue"></td>
          <td width="50" height="50" bgcolor="green"></td>
          <td width="50" height="50" bgcolor="red"></td>
          <td width="50" height="50" bgcolor="lightblue"></td>
          <td width="50" height="50" bgcolor="green"></td>
          <td width="50" height="50" bgcolor="red"></td>
        </tr>

      </table>

    </body>
</html>
```

OUTPUT:


COLOR PATTERN
File
C:/Users/DELL/OneDrive/Desktop/Vijay/FSD/COLORPATTERN.html
HTML TABLE BG COLOR

EXPERIMENT NO: 02

AIM:
Implement Drag n Drop feature in HTML 5

DESCRIPTION:

1. `<div draggable="true">`:
    - Defines an element that can be dragged.
    - The `draggable` attribute set to "true" enables dragging for the element.
    - Example: `<div draggable="true">Drag me!</div>`
2. `<div dropzone="copy">`:
    - Defines a drop zone where dragged items can be dropped.
    - The `dropzone` attribute specifies the type of operation allowed: "copy", "move", or "link".
    - Example: `<div dropzone="copy">Drop here</div>`
3. `ondragstart` event:
    - Triggered when the user starts dragging an element.
    - Used to specify what should happen when dragging starts.
    - Example: `<div draggable="true" ondragstart="drag(event)">Drag me!</div>`
4. `ondrop` event:
    - Triggered when an element is dropped into a drop zone.
    - Used to define the action to be performed upon dropping.
    - Example: `<div dropzone="copy" ondrop="drop(event)">Drop here</div>`

Here's a more detailed explanation of each tag and attribute:

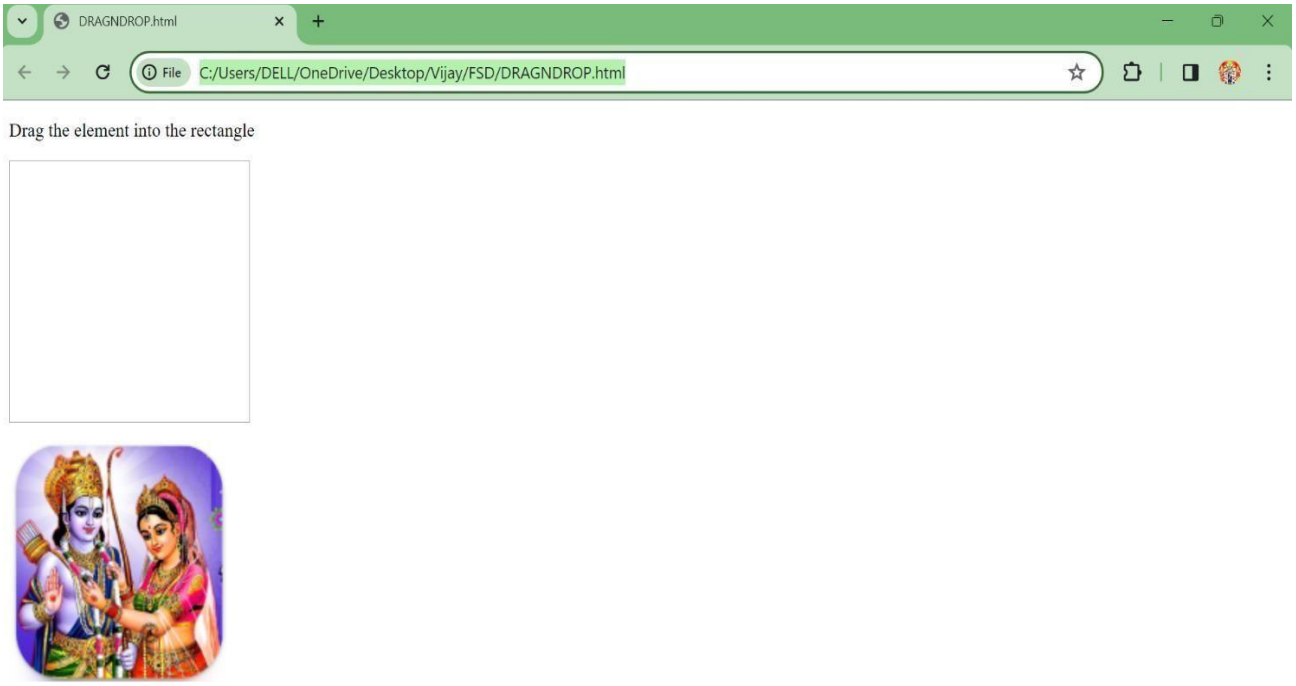
1. `<div draggable="true">`:
    - The `draggable` attribute allows an element to be draggable.
    - When set to "true", the element can be dragged by the user.
    - The element can be styled and customized to indicate that it's draggable using CSS.
2. `<div dropzone="copy">`:
    - The `dropzone` attribute specifies the type of drop operation allowed.
    - Values can be "copy", "move", or "link" to define different behaviors.
    - The drop zone can be styled using CSS to provide visual feedback to the user.
3. `ondragstart` event:
    - The `ondragstart` event is fired when the user starts dragging an element.
    - It's often used to set data that will be transferred when the element is dropped.
    - The event handler can be defined inline or attached using JavaScript.
4. `ondrop` event:
    - The `ondrop` event is triggered when an element is dropped into a drop zone.
    - It's used to handle the drop action and process the transferred data.
    - The event handler typically retrieves the dropped data and performs the necessary operations, such as displaying it or updating the UI.

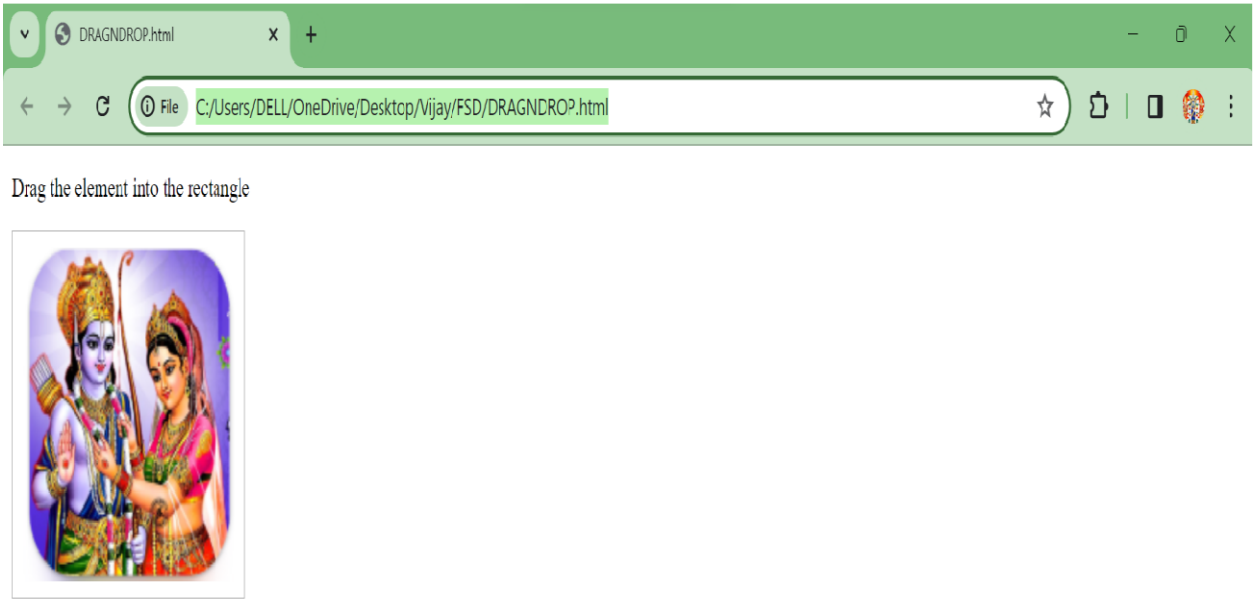
Implementing drag-and-drop functionality in HTML5 involves using these tags and event handlers to create a seamless and interactive user experience.

PROGRAM:

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      #div1{
        width: 200px;
        height: 200px;
        padding: 10px;
        border: 1px solid #aaaaaa;

      }
    </style>
    <script>
      function allowDrop(ev){
        ev.preventDefault();
      }
      function drag(ev){
        ev.dataTransfer.setData("text",ev.target.id);
      }
      function drop(ev){
        ev.preventDefault();
        var data=ev.dataTransfer.getData("text");
        ev.target.appendChild(document.getElementById(data));
      }
    </script>
  </head>
  <body>
    <p>Drag the element into the rectangle</p>
    <div id="div1" ondrop="drop(event)" ondragover="allowDrop(event)"></div>
    <br>
    <img id="drag1" src="sitaram.png" draggable="true" ondragstart="drag(event)" width="200" height="200">
    </body>
</html>
```

OUTPUT:


DRAGNDROP.html
File
C:/Users/DELL/OneDrive/Desktop/Vijay/FSD/DRAGNDROP.html
Drag the element into the rectangle


DRAGNDROP.html
File
C:/Users/DELL/OneDrive/Desktop/Vijay/FSD/DRAGNDROP.html
Drag the element into the rectangle

EXPERIMENT NO: 03

AIM:

A. **Demonstrate Event bubbling with necessary examples.**

DESCRIPTION:

Event bubbling is a mechanism in JavaScript where an event triggered on a child element "bubbles" up through its parent elements, triggering their respective event handlers. In HTML5, event bubbling is a built-in feature that occurs with many DOM events. Here are some tags and concepts related to event bubbling in HTML5:

1. **Event Listeners**:
   - Using `addEventListener()` method to attach event handlers to elements.
   - Helps in capturing and handling events during the bubbling phase.
   - Example: `element.addEventListener('click', handleClick);`
2. `event.stopPropagation()`:
   - Method used within an event handler to stop the event from bubbling up to parent elements.
   - Prevents any parent event handlers from being triggered.
   - Example: `event.stopPropagation();`
3. **Event Bubbling**:
   - Natural behavior where an event triggered on a child element propagates up to its parent elements.
   - Allows handling the event at various levels of the DOM hierarchy.
   - Example: Clicking on a child button can also trigger click events on its parent divs.
4. **Event Phase**:
   - Events go through three phases: capturing, target, and bubbling.
   - `addEventListener()` can be used with the third argument to specify the phase (`true` for capturing, `false` for bubbling).
   - Example: `element.addEventListener('click', handleClick, true);`

Detailed Explanation:

1. **Event Listeners**:
   - To implement event bubbling, you can attach event listeners to elements using the `addEventListener()` method.
   - This method takes the event type ('click', 'mouseover', etc.) and a callback function as arguments.
   - Event listeners can be added to any HTML element, making it flexible to capture and handle events during the bubbling phase.
2. `event.stopPropagation()`:
   - When handling an event, if you want to prevent it from bubbling up to parent elements, you can use the `event.stopPropagation()` method.
   - This method stops the event from propagating further, ensuring that only the specific event handler is executed.
   - It's essential when you want to handle an event on a specific element without affecting its parent elements.
3. **Event Bubbling**:

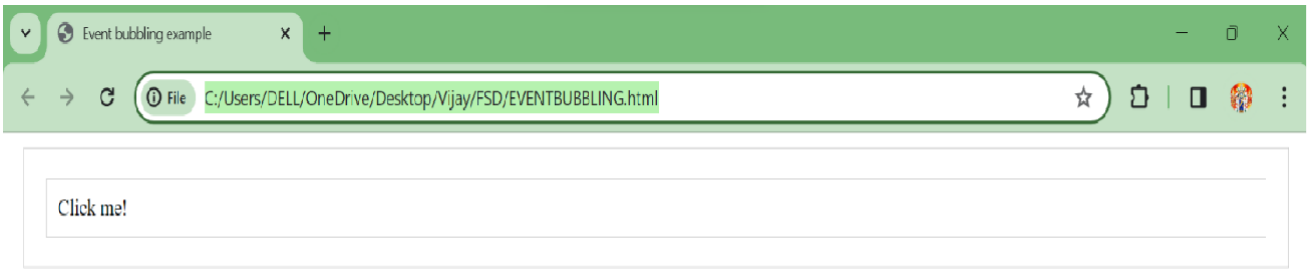
- Event bubbling is the default behavior in JavaScript where an event triggered on a child element will propagate up through its parent elements.
- This mechanism allows you to handle events at different levels of the DOM hierarchy, making it versatile for various use cases.
- By understanding event bubbling, you can design better event handling strategies and optimize event propagation in your applications.

4. **Event Phase**:
    - Events in the DOM go through three phases: capturing, target, and bubbling.
    - The capturing phase occurs before the event reaches the target element.
    - The target phase is when the event is on the target element.
    - The bubbling phase happens after the event has been handled on the target element and bubbles up through its parent elements.
    - `addEventListener()` method can be used with the third argument to specify the event phase ('true' for capturing, 'false' for bubbling), giving you more control over event handling.

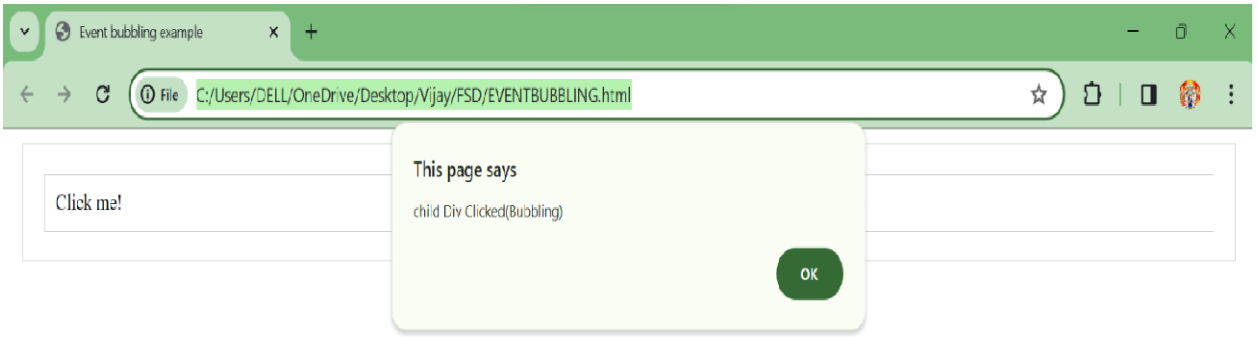
PROGRAM:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Event bubbling example</title>
    <style>
      div{
        border: 1px solid #ccc;
        padding: 10px;
        margin: 10px;
      }
    </style>
  </head>
  <body>
    <div id="parent" >
      <div id="child">Click me!</div>
    </div>
    <script>
      document.getElementById('parent').addEventListener('click',function(){alert('parent Div Clicked(Bubbling)');});
      document.getElementById('child').addEventListener('click',function(){alert('child Div Clicked(Bubbling)');});

    </script>
  </body>
</html>
```

OUTPUT:


Event bubbling example
File
C:/Users/DELL/OneDrive/Desktop/Vijay/FSD/EVENTBUBBLING.html
Click me!


Event bubbling example
File
C:/Users/DELL/OneDrive/Desktop/Vijay/FSD/EVENTBUBBLING.html
Click me!
This page says
parent Div Clicked(Bubbling)
OK


Event bubbling example
File
C:/Users/DELL/OneDrive/Desktop/Vijay/FSD/EVENTBUBBLING.html
Click me!
This page says
child Div Clicked(Bubbling)
OK

EXPERIMENT NO: 03

AIM:

B . Demonstrate Event Capturing with necessary examples.

DESCRIPTION:

Event capturing is the phase in the DOM event flow where the event is captured by the outermost element and propagated down to the target element. In HTML5, event capturing can be implemented using event listeners with the capturing phase set to `true`. Here are the tags and concepts related to event capturing in HTML5:

1. **Event Listeners with Capturing Phase**:
   - Use `addEventListener()` method with the third argument set to `true` to enable event capturing.
   - Allows you to capture events during the capturing phase before reaching the target element.
   - Example: `element.addEventListener('click', handleClick, true);`
2. `event.stopPropagation()`:
   - Method used within an event handler to stop the event from further propagation.
   - Prevents the event from reaching the target element and triggering its event handlers.
   - Example: `event.stopPropagation();`
3. **Event Bubbling vs. Event Capturing**:
   - Event capturing occurs in the initial phase of the event flow, from the outermost parent to the target element.
   - It's the opposite of event bubbling, which occurs after the event reaches the target and bubbles up to the parent elements.
   - Understanding both phases helps in implementing complex event handling scenarios effectively.
4. **Event Phase**:
   - Events in the DOM flow through three phases: capturing, target, and bubbling.
   - Capturing phase happens first, followed by the target phase, and then the bubbling phase.
   - `addEventListener()` allows you to specify the event phase ('true' for capturing, 'false' for bubbling), giving you precise control over event handling.

Detailed Explanation:

1. **Event Listeners with Capturing Phase**:
   - To implement event capturing in HTML5, use the `addEventListener()` method with the third argument set to `true`.
   - This enables event capturing, allowing you to capture and handle events during the capturing phase before they reach the target element.
   - Capturing phase is useful for intercepting events at the parent level and executing specific actions based on the event propagation.
2. `event.stopPropagation()`:
   - Within an event handler, you can use the `event.stopPropagation()` method to stop the event from further propagation.
   - This prevents the event from reaching the target element and triggering its event handlers.
   - It's crucial when you want to handle an event at a specific phase without affecting the subsequent phases of event flow.
3. **Event Bubbling vs. Event Capturing**:
   - Event capturing and event bubbling are two phases of the DOM event flow, occurring in opposite directions.
   - Event capturing happens during the initial phase, from the outermost parent element to the target.
   - In contrast, event bubbling occurs after the event reaches the target and bubbles up through its parent elements.

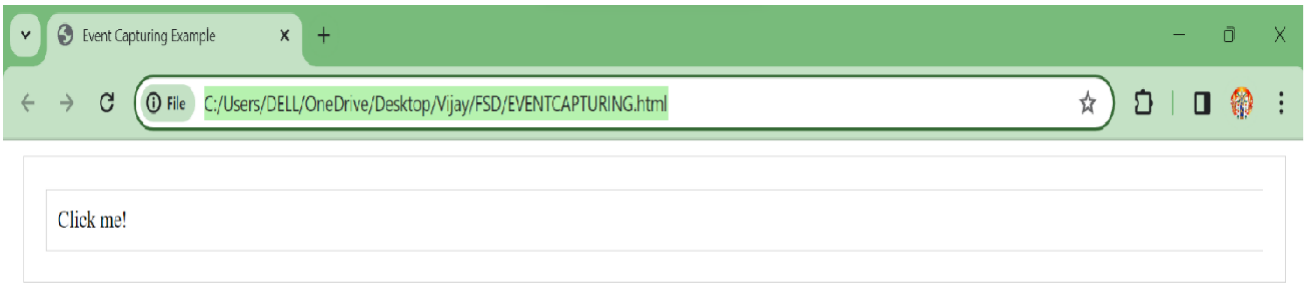
- Understanding the differences between these phases is essential for designing robust and efficient event handling mechanisms.

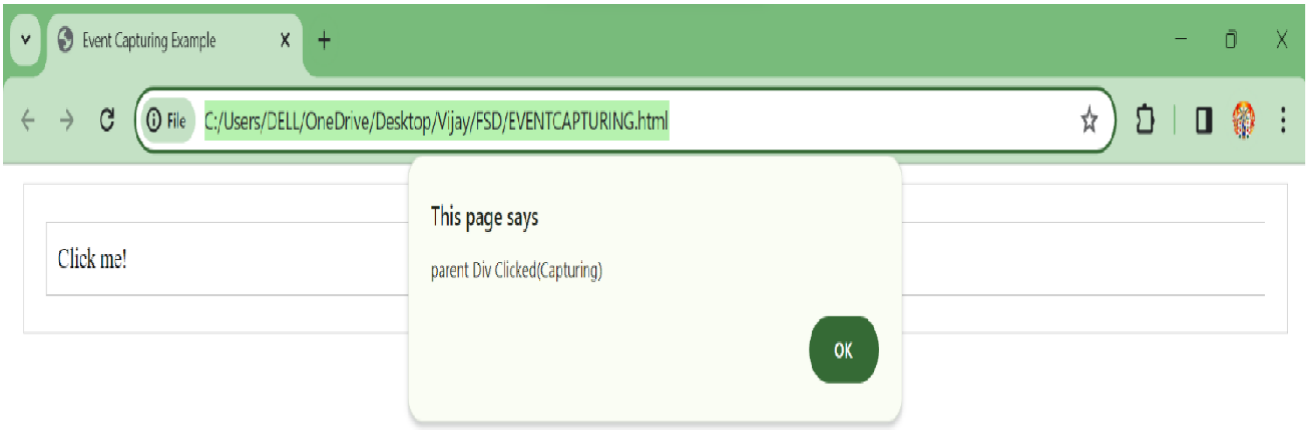
4. **Event Phase**:
    - Events in the DOM go through three distinct phases: capturing, target, and bubbling.
    - Capturing phase is the initial phase where the event is captured by the outermost parent element.
    - Target phase occurs when the event reaches the target element.
    - Bubbling phase is the final phase where the event bubbles up through the parent elements.
    - `addEventListener()` method allows you to specify the event phase ('true' for capturing, 'false' for bubbling), providing fine-grained control over event handling and propagation.

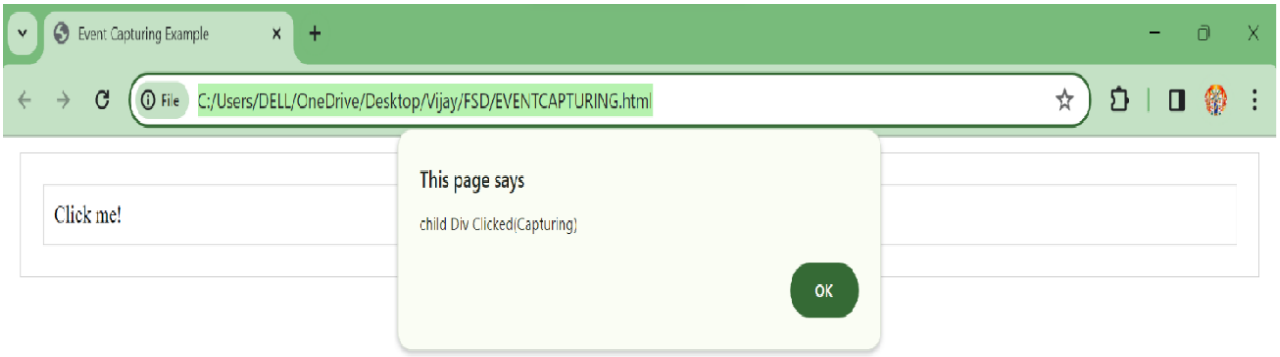
PROGRAM:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Event Capturing Example</title>
    <style>
      div{
        border: 1px solid #ccc;
        padding: 10px;
        margin: 10px;
      }
    </style>
  </head>
  <body>
    <div id="parent">
      <div id="child">Click me!</div>
    </div>
    <script>
            document.getElementById('parent').addEventListener('click',function(){alert('parent Div Clicked(Capturing)');},true);
            document.getElementById('child').addEventListener('click',function(){alert('child Div Clicked(Capturing)');},true);
    </script>
  </body>
</html>
```

OUTPUT:


Event Capturing Example
File
C:/Users/DELL/OneDrive/Desktop/Vijay/FSD/EVENTCAPTURING.html
Click me!

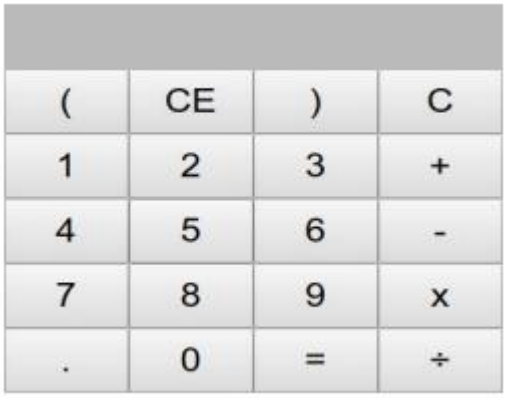

Event Capturing Example
File
C:/Users/DELL/OneDrive/Desktop/Vijay/FSD/EVENTCAPTURING.html
Click me!
This page says
parent Div Clicked(Capturing)
OK


Event Capturing Example
File
C:/Users/DELL/OneDrive/Desktop/Vijay/FSD/EVENTCAPTURING.html
Click me!
This page says
child Div Clicked(Capturing)
OK

EXPERIMENT NO: 04

AIM:

Design a Calculator using Java script and relevant CSS.



DESCRIPTION:

1. **HTML Structure**:
   - The HTML structure of the calculator typically includes an input field to display the current value or result and buttons for numbers, operators, and other functionalities.
   - The display area is usually an input field set to readonly to prevent direct user input and ensure that the value is only updated through the calculator's logic.
2. **JavaScript Calculation Logic**:
   - The JavaScript part handles the core functionality of the calculator, including capturing user input, performing calculations, and updating the display.
   - Functions are defined to append numbers and operators to the expression, evaluate the expression to get the result, and clear the display when needed.
   - The `eval()` function can be used to evaluate the expression string and calculate the result. However, it's essential to handle potential errors, such as division by zero or invalid expressions, to ensure the calculator's robustness.
3. **CSS Styling**:
   - CSS is used to style the calculator's appearance, making it visually appealing and user-friendly.
   - You can style the display area, buttons, and overall layout using CSS properties like `width`, `padding`, `margin`, `border`, `font-size`, `text-align`, and `background-color`.
   - Grid or flexbox layouts can be used to arrange the calculator buttons in rows and columns, ensuring a neat and organized appearance.
4. **Event Handling**:
   - Event listeners are attached to the calculator buttons to capture user interactions.
   - When a button is clicked, its corresponding value (number or operator) is appended to the expression, updating the display.
   - For the equal button '=', the expression is evaluated to calculate the result, which is then displayed in the input field.
   - Clear or reset buttons can be added to allow users to clear the display and start a new calculation.
5. **Additional Features**:

- You can enhance the calculator by adding support for decimal numbers, memory functions (e.g., memory recall, memory clear), keyboard input, and more advanced mathematical operations.
- Error handling is crucial to provide feedback to users in case of invalid input or operations, ensuring a smooth user experience.
- Styling can be further improved with animations, themes, and responsive design to make the calculator accessible and visually appealing across different devices and screen sizes.

By understanding these concepts and principles, you can design and implement a functional and user-friendly calculator using JavaScript and CSS. This approach allows you to create a custom calculator tailored to your specific requirements, whether it's a basic calculator for simple calculations or a more advanced one with additional features and functionalities.

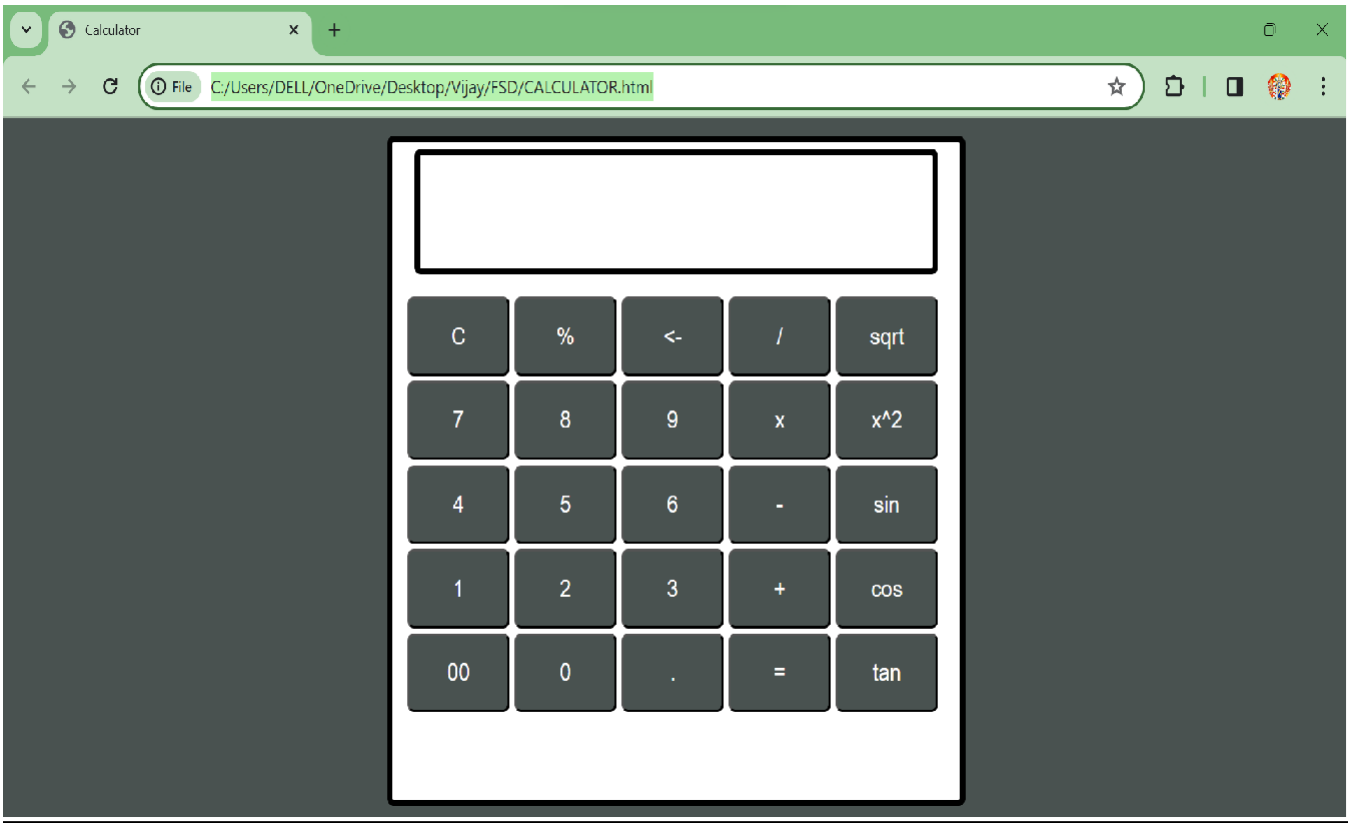
PROGRAM:

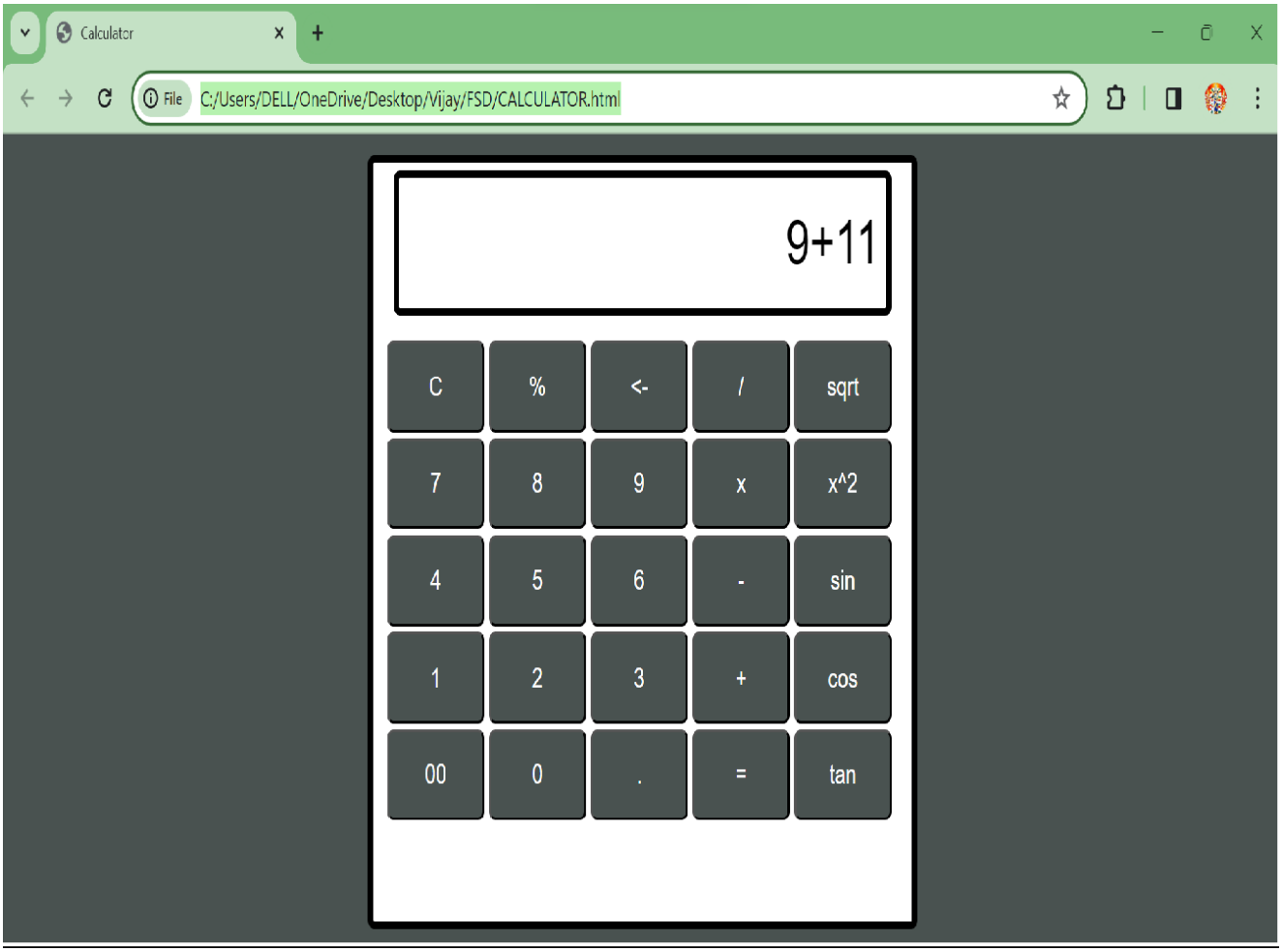
```
<!DOCTYPE html>
<html>
  <head>
    <style>
      *{
        padding:0;
        margin:0;
        font-family: "poppins",sans-serif;
      }
      body{
        background-color: #495250;
        display: grid;
        height:100vh;
        place-items: center;
      }
      .main{
        width:500px;
        height:500px;
        background-color: white;
        position: absolute;
        border: 5px solid black;
        border-radius:6px;
      }
      .main input[type="text"]{
        width:88%;
        position: relative;
        height:80px;
        top:5px;
        text-align: right;
        padding: 3px 6px;
        outline: none;
        font-size: 40px;
        border: 5px solid black;
        display: flex;
        margin: auto
        border-radius: 6px;
        color:black;

      }
      .btn input[type="button"]{
        width:90px;
```

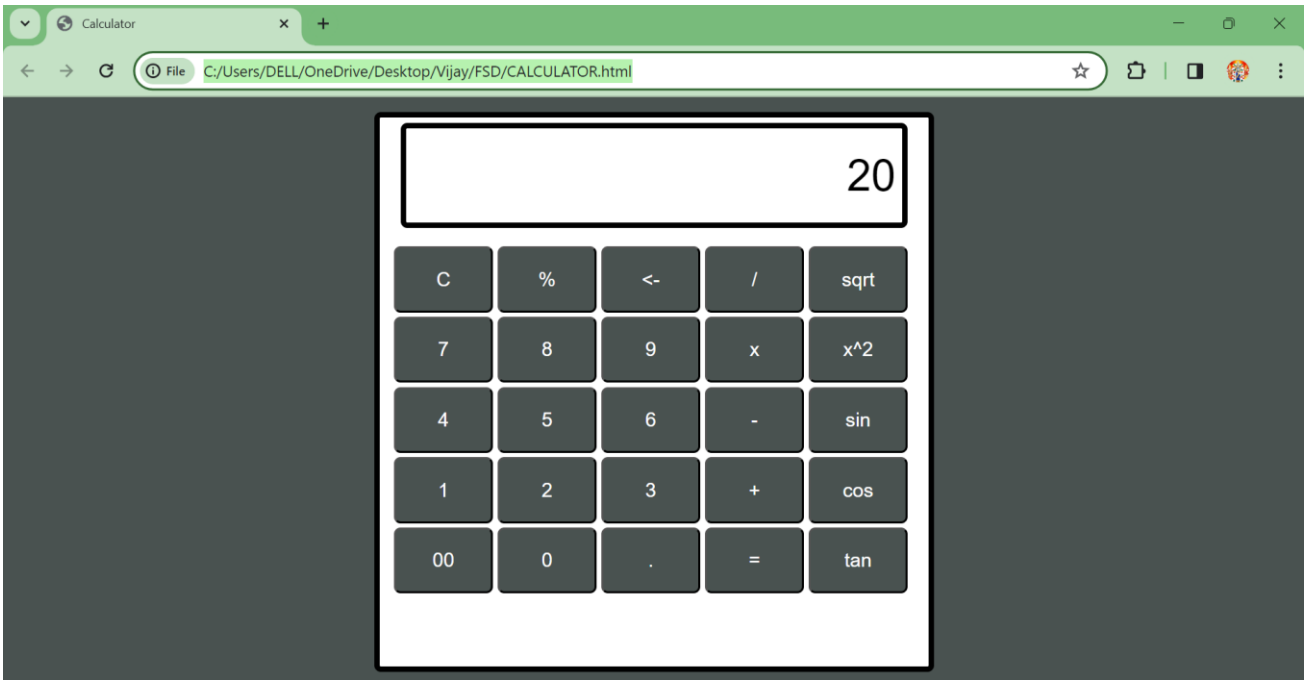
```
            padding:2px;
            margin: 2px 0px;
            position: relative;
            left: 13px;
            top:20px;
            height: 60px;
            cursor: pointer;
            font-size: 18px;
            transition: 0.5s;
            background-color: #495250;
            border-radius:6px;
            color:#ffffff;
        }
        .btn input[type='button']:hover{
            background-color: black;
            color:#ffffff;
        }
    </style>
    <script>
        function Solve(val){
            var v = document.getElementById('res');
            v.value +=val;
        }
        function Result(){
            var num1 = document.getElementById('res').value;
            var num2 = eval(num1);
            document.getElementById('res').value = num2;
        }
        function Clear(){
            var inp = document.getElementById('res');
            inp.value = ' ';
        }
        function Back(){
            var ev = document.getElementById('res');
            ev.value = ev.value.slice(0,-1);
        }
        function SquareRoot(){
            var v = document.getElementById('res');
            v.value = Math.sqrt(eval(v.value));
        }
        function Power(){
            var v = document.getElementById('res');
            v.value = Math.pow(eval(v.value),2);
        }
        function Sine(){
            var v = document.getElementById('res');
            v.value = Math.sin(eval(v.value));
        }
        function Cosine(){
            var v = document.getElementById('res');
            v.value = Math.cos(eval(v.value));
        }
        function Tangent(){
            var v = document.getElementById('res');
            v.value = Math.tan(eval(v.value));
        }
    </script>
    <title>Calculator</title>
</head>
```

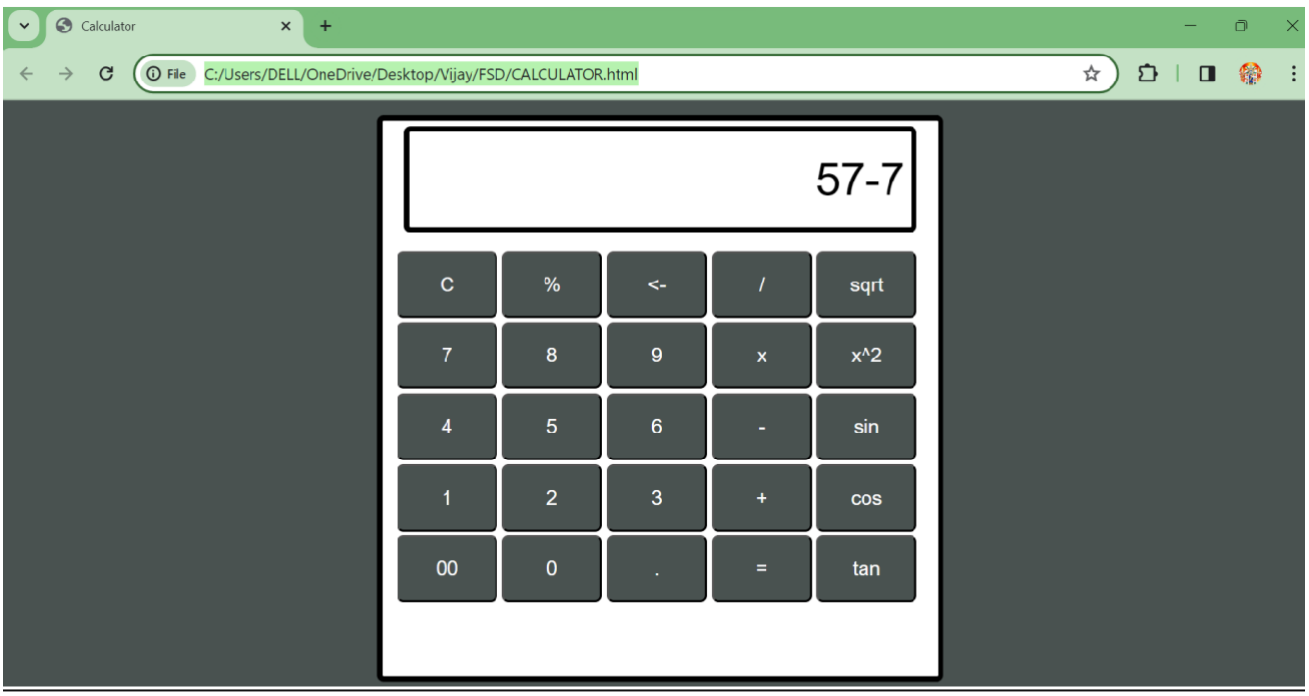
```
    <body>
      <div class="main">
        <input type="text" id="res">
        <div class="btn">
          <input type="button" value="C" onclick="Clear()">
          <input type="button" value="%" onclick="Solve('%')">
          <input type="button" value="<-" onclick="Back('<-')">
          <input type="button" value="/" onclick="Solve('/')">
          <input type="button" value="sqrt" onclick="SquareRoot()">
          <br>
          <input type="button" value="7" onclick="Solve('7')">
          <input type="button" value="8" onclick="Solve('8')">
          <input type="button" value="9" onclick="Solve('9')">
          <input type="button" value="x" onclick="Solve('*')">
          <input type="button" value="x^2" onclick="Power()">
          <br>
          <input type="button" value="4" onclick="Solve('4')">
          <input type="button" value="5" onclick="Solve('5')">
          <input type="button" value="6" onclick="Solve('6')">
          <input type="button" value="-" onclick="Solve('-')">
          <input type="button" value="sin" onclick="Sine()">
          <br>
          <input type="button" value="1" onclick="Solve('1')">
          <input type="button" value="2" onclick="Solve('2')">
          <input type="button" value="3" onclick="Solve('3')">
          <input type="button" value="+" onclick="Solve('+')">
          <input type="button" value="cos" onclick="Cosine()">
          <br>
          <input type="button" value="00" onclick="Solve('00')">
          <input type="button" value="0" onclick="Solve('0')">
          <input type="button" value="." onclick="Solve('.')">
          <input type="button" value="=" onclick="Result('+')">
          <input type="button" value="tan" onclick="Tangent()">
          <br>
        </div>
      </div>
    </body>
  </html>
```

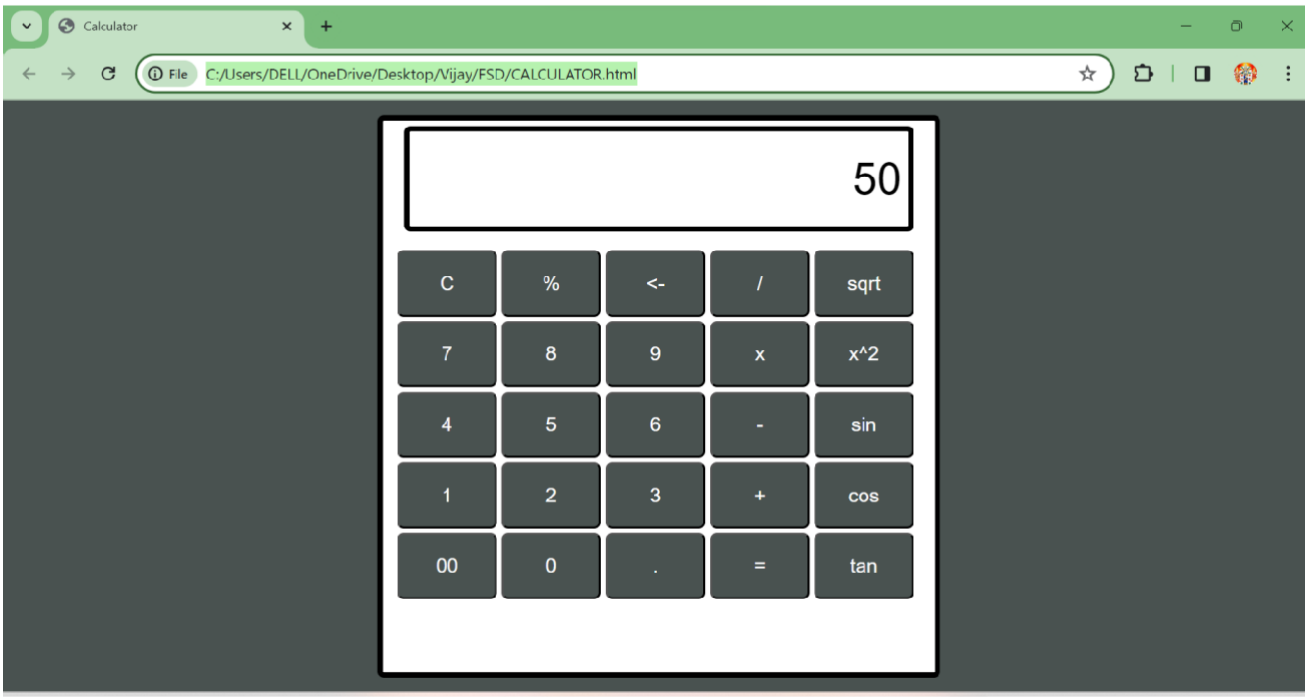
OUTPUT:

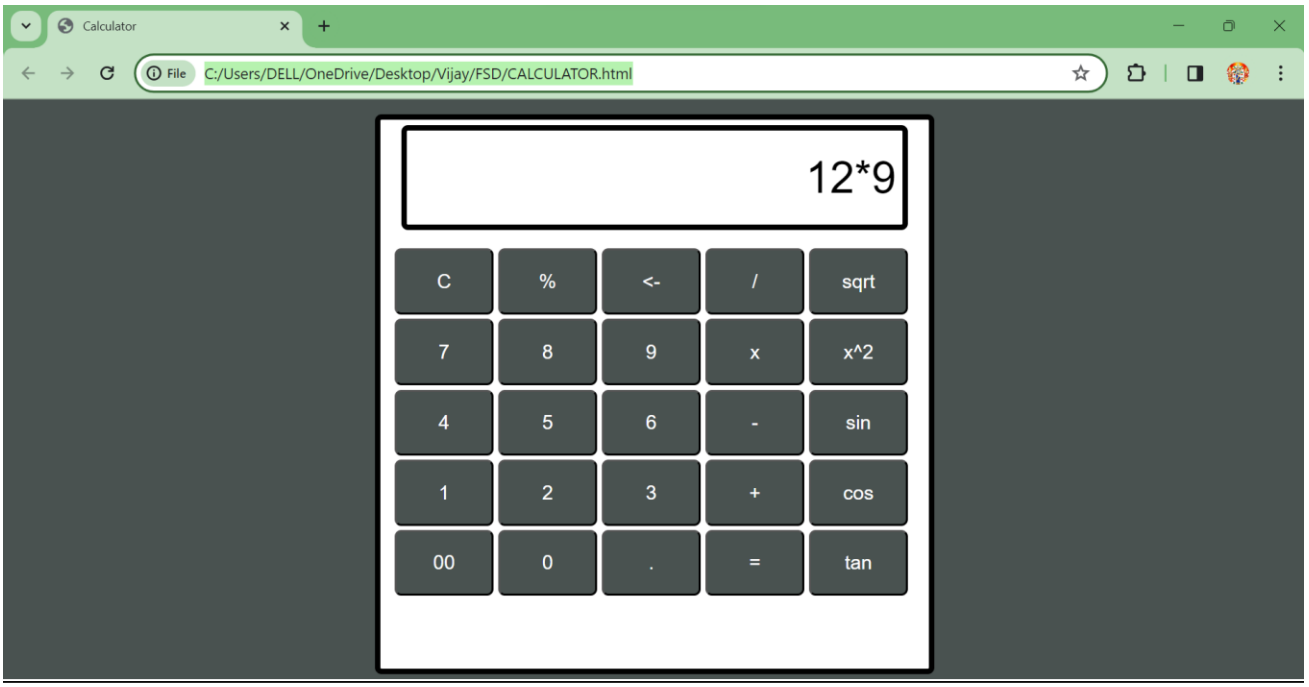

Calculator
File
C:/Users/DELL/OneDrive/Desktop/Vijay/FSD/CALCULATOR.html
C
%
<-
/
sqrt
7
8
9
x
x^2
4
5
6
-
sin
1
2
3
+
cos
00
0
.
=
tan

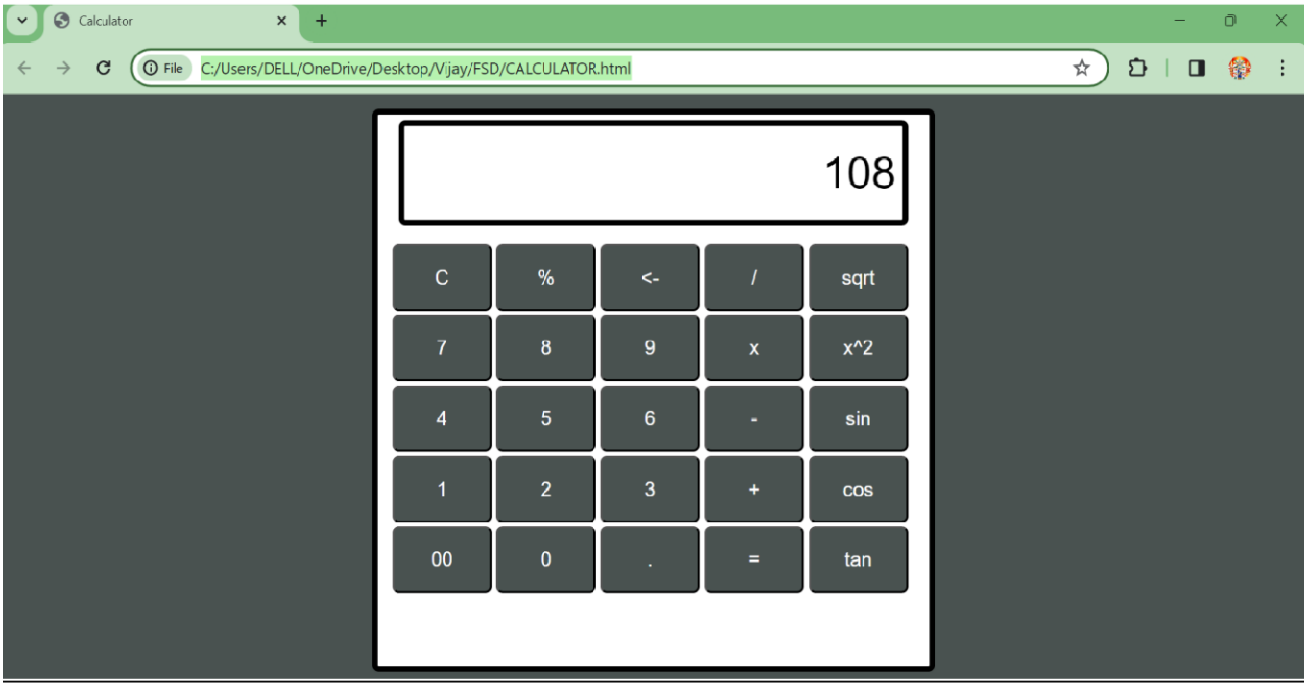

Calculator
File
C:/Users/DELL/OneDrive/Desktop/Vijay/FSD/CALCULATOR.html
9+11
C
%
<-
/
sqrt
7
8
9
x
x^2
4
5
6
-
sin
1
2
3
+
cos
00
0
.
=
tan

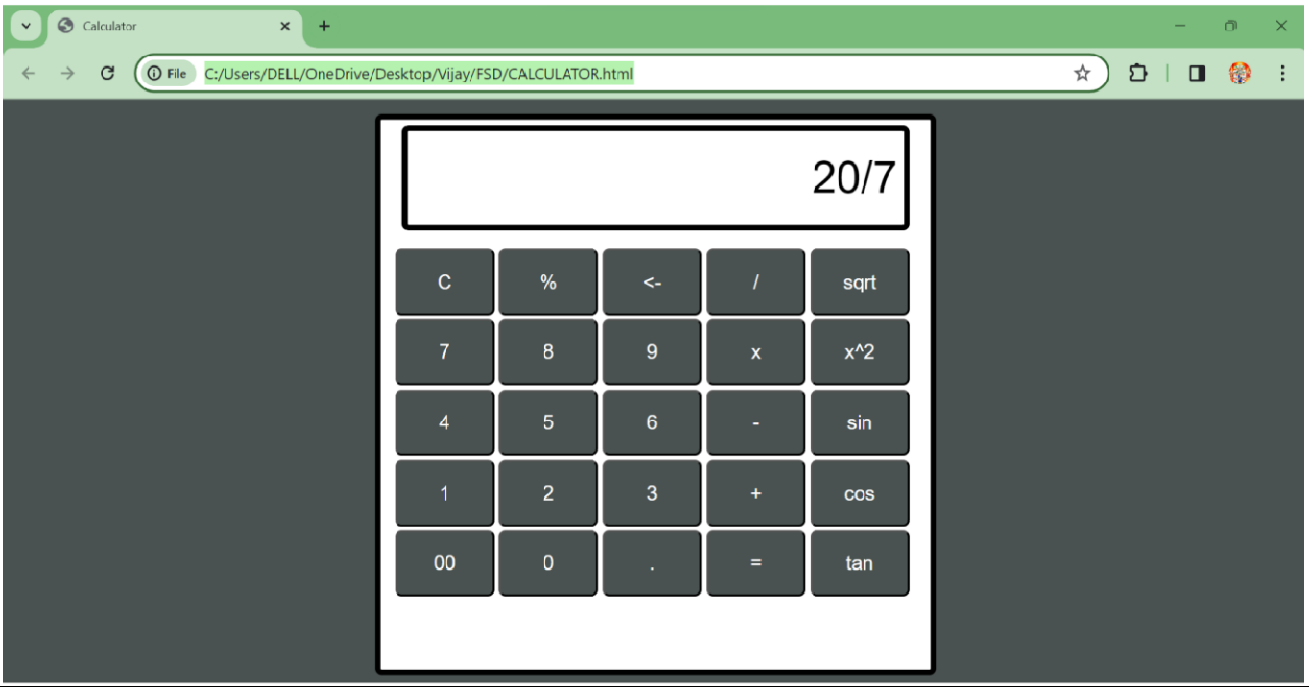

Calculator
File
C:/Users/DELL/OneDrive/Desktop/Vijay/FSD/CALCULATOR.html
20
C
%
<-
/
sqrt
7
8
9
x
x^2
4
5
6
-
sin
1
2
3
+
cos
00
0
.
=
tan

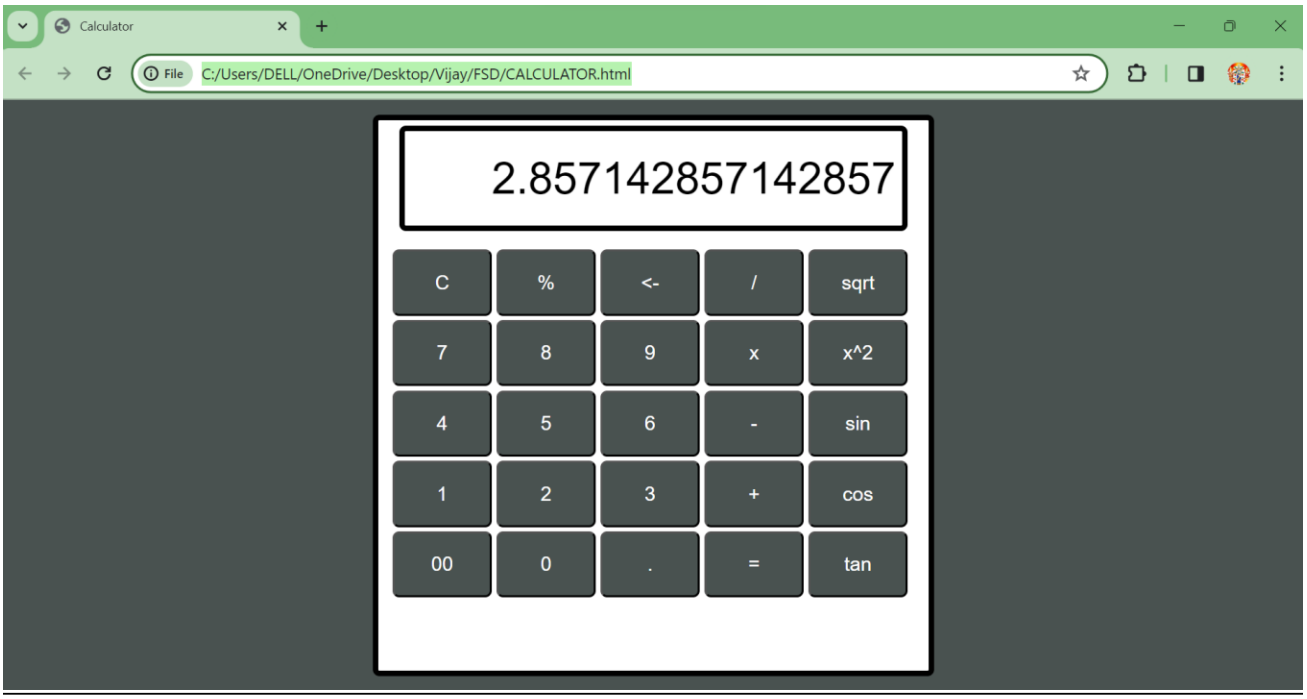

Calculator
File
C:/Users/DELL/OneDrive/Desktop/Vijay/FSD/CALCULATOR.html
57-7
C
%
<-
/
sqrt
7
8
9
x
x^2
4
5
6
-
sin
1
2
3
+
cos
00
0
.
=
tan

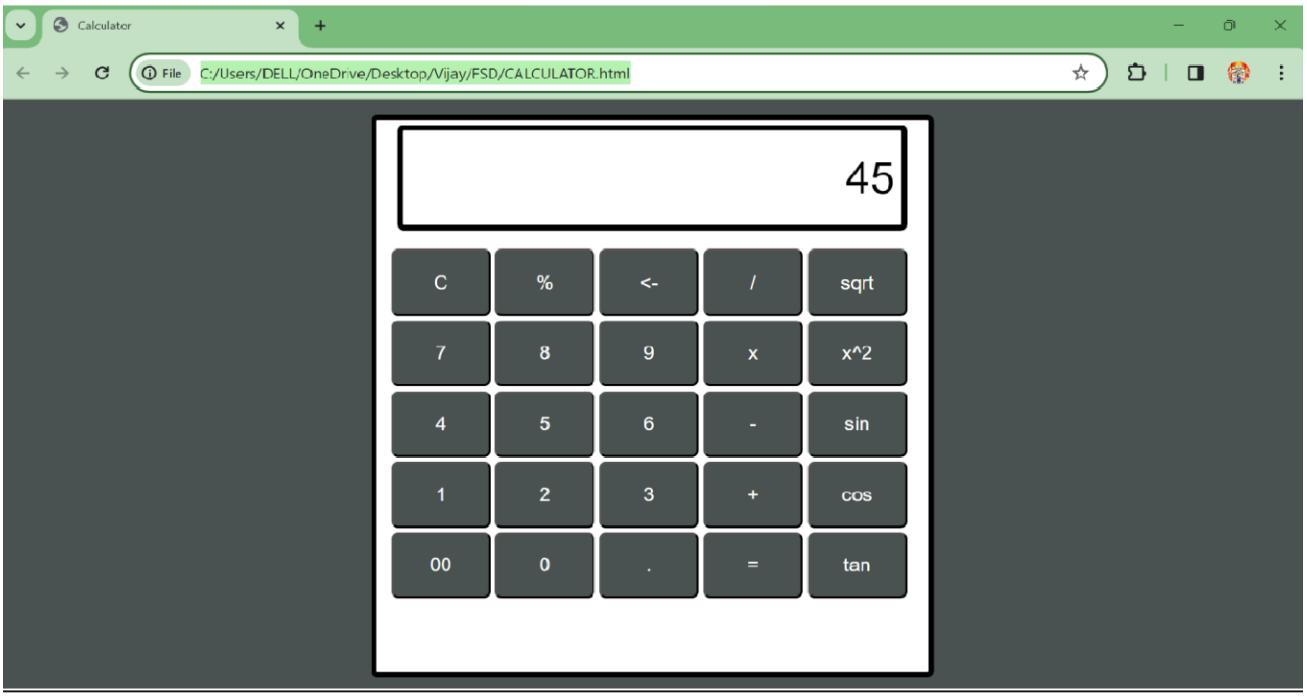

Calculator
File
C:/Users/DELL/OneDrive/Desktop/Vijay/FSD/CALCULATOR.html
50
C
%
<-
/
sqrt
7
8
9
x
x^2
4
5
6
-
sin
1
2
3
+
cos
00
0
.
=
tan

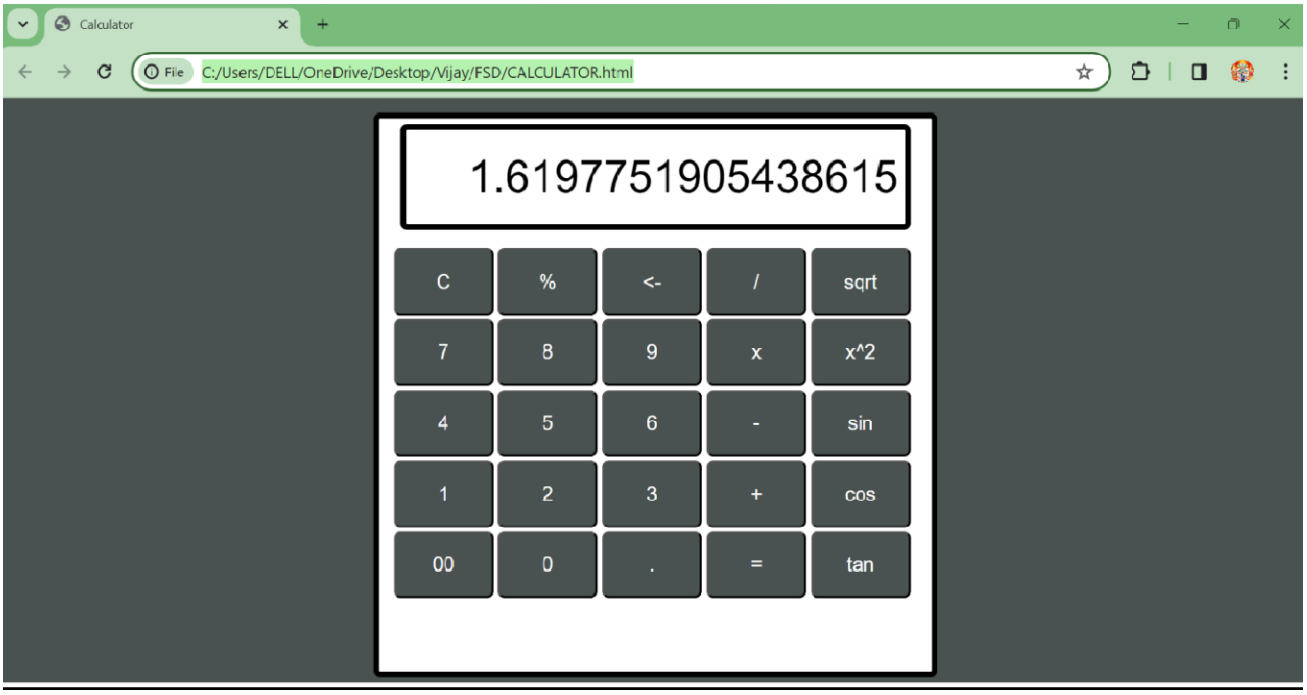

Calculator
File
C:/Users/DELL/OneDrive/Desktop/Vijay/FSD/CALCULATOR.html
12*9
C
%
<-
/
sqrt
7
8
9
x
x^2
4
5
6
-
sin
1
2
3
+
cos
00
0
.
=
tan


Calculator
File
C:/Users/DELL/OneDrive/Desktop/Vijay/FSD/CALCULATOR.html
108
C
%
<-
/
sqrt
7
8
9
x
x^2
4
5
6
-
sin
1
2
3
+
cos
00
0
.
=
tan


Calculator
File
C:/Users/DELL/OneDrive/Desktop/Vijay/FSD/CALCULATOR.html
20/7
C
%
<-
/
sqrt
7
8
9
x
x^2
4
5
6
-
sin
1
2
3
+
cos
00
0
.
=
tan


Calculator
File
C:/Users/DELL/OneDrive/Desktop/Vijay/FSD/CALCULATOR.html
2.857142857142857
C
%
<-
/
sqrt
7
8
9
x
x^2
4
5
6
-
sin
1
2
3
+
cos
00
0
.
=
tan


Calculator
File
C:/Users/DELL/OneDrive/Desktop/Vijay/FSD/CALCULATOR.html
45
C
%
<-
/
sqrt
7
8
9
x
x^2
4
5
6
-
sin
1
2
3
+
cos
00
0
.
=
tan


Calculator
File
C:/Users/DELL/OneDrive/Desktop/Vijay/FSD/CALCULATOR.html
1.6197751905438615
C
%
<-
/
sqrt
7
8
9
x
x^2
4
5
6
-
sin
1
2
3
+
cos
00
0
.
=
tan

EXPERIMENT NO: 05

AIM:

Demonstrate Higher order functions with necessary examples – filter(), reduce() and map().

DESCRIPTION:

1. `filter()`:
   - The `filter()` method creates a new array with all elements that pass the test implemented by the provided function.
   - It iterates through each item in the array and applies the provided function. If the function returns `true`, the item is included in the new array; otherwise, it's excluded.
   - It doesn't modify the original array but returns a new filtered array.
2. `map()`:
   - The `map()` method creates a new array populated with the results of calling a provided function on every element in the calling array.
   - It iterates over each item in the array and applies the provided function to each item, producing a new value.
   - It returns a new array with the same length as the original array but with modified values based on the function's logic.
3. `reduce()`:
   - The `reduce()` method executes a reducer function on each element of the array, resulting in a single output value.
   - It takes an accumulator and the current value as arguments and returns a single value that accumulates the results of the function over each iteration.
   - It's often used for tasks like summing up values, calculating averages, or transforming arrays into objects.

Key Points:

- **Higher-Order Function**: All these methods are higher-order functions because they take another function as an argument or return a function.
- **Immutability**: `filter()` and `map()` do not modify the original array; instead, they return a new array with the desired elements or values. This approach promotes immutability, which is beneficial for maintaining the integrity of data.
- **Transformation and Computation**: `map()` is primarily used for transforming array elements, while `filter()` is used for selecting specific elements based on a condition. `reduce()` is more versatile and can perform various computations, including aggregation and transformation.
- **Flexibility**: These methods provide flexibility in manipulating arrays, allowing for concise and expressive code. They are fundamental in functional programming paradigms and are widely used in modern JavaScript development.

Understanding and mastering these higher-order functions can greatly enhance your ability to manipulate arrays and perform complex operations efficiently in JavaScript. They enable you to write cleaner, more readable, and maintainable code by leveraging functional programming principles.

PROGRAM:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Higher Order Functions</title>
  </head>
  <body>
    <h1>
      Higher Order Functions
    </h1>
    <button onclick="filterEven()">Filter Even Numbers</button>
    <button onclick="squareNumber()">Square Numbers</button>
    <button onclick="calculateProduct()">Calculate Product</button>
    <script>
      const numbers = [1,2,3,4, 5, 6, 7, 8,9,10];
      function displayResult(title,result){
        alert(`${title}`,result);
        const resulltElement = document.createElement('p');
        resulltElement.textContent =`${title}:${result}`;
        document.body.appendChild(resulltElement);
      }
      function filterEven(){
        const isEven = num => num%2 ===0;
        const filteredNumbers = numbers.filter(isEven);
        displayResult("Filtered Even Numbers are ",filteredNumbers);
      }
      function squareNumber(){
        const  square = num => num **2;
        const squaredNumbers=numbers.map(square);
        displayResult ("Squared Numbers are ",squaredNumbers );
      }
      function calculateProduct(){
        const multiply = (x,y) => x*y;
        const product = numbers.reduce(multiply);
        displayResult ("Product of all numbers are ",product);
      }
    </script>
  </body>
</html>
```

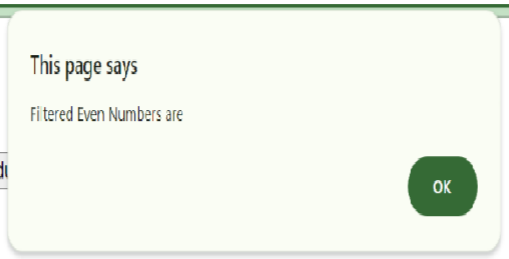
OUTPUT:

Higher Order Functions

Filter Even Numbers | Square Numbers | Calculate Product

Higher Order Functions

Filter Even Numbers | Square Numbers | Calculate Prod



Higher Order Functions

Filter Even Numbers | Square Numbers | Calculate Product

Filtered Even Numbers are :2,4,6,8,10

Higher Order Functions

Filter Even Numbers | Square Numbers | Calculate Prod

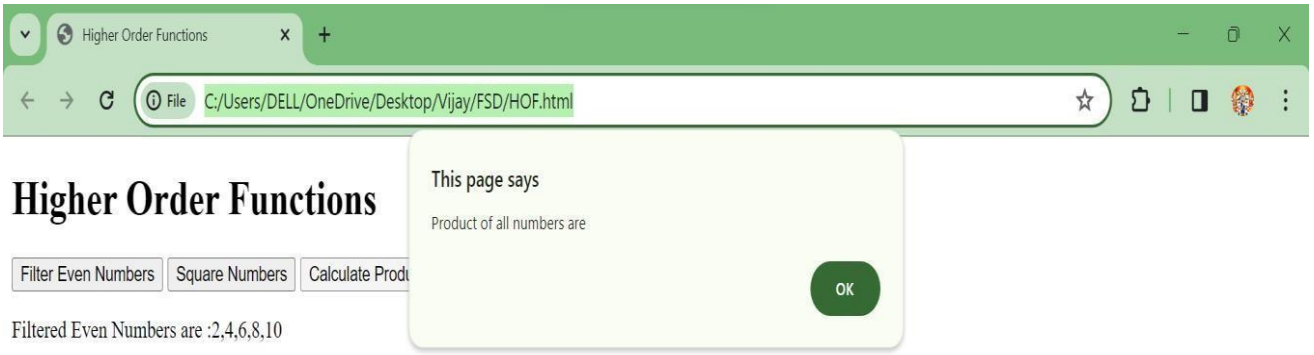
Filtered Even Numbers are :2,4,6,8,10

Higher Order Functions

Filter Even Numbers | Square Numbers | Calculate Product

Filtered Even Numbers are :2,4,6,8,10

Squared Numbers are :1,4,9,16,25,36,49,64,81,100


Higher Order Functions
File
C:/Users/DELL/OneDrive/Desktop/Vijay/FSD/HOF.html
This page says
Product of all numbers are
OK
Higher Order Functions
Filter Even Numbers
Square Numbers
Filtered Even Numbers are :2,4,6,8,10
Squared Numbers are :1,4,9,16,25,36,49,64,81,100


Higher Order Functions
File
C:/Users/DELL/OneDrive/Desktop/Vijay/FSD/HOF.html
Higher Order Functions
Filter Even Numbers
Square Numbers
Calculate Product
Filtered Even Numbers are :2,4,6,8,10
Squared Numbers are :1,4,9,16,25,36,49,64,81,100
Product of all numbers are :3628800

EXPERIMENT NO: 06

AIM:

Create a Class Component for Counter in React JS.

DESCRIPTION:

Class Components in ReactJS:

Class components are one of the two primary ways to define React components, the other being functional components. Class components are defined using ES6 classes and extend the `React.Component` class provided by React.

Key Characteristics:

1. **State Management**: Class components can have a local state using `this.state` and update it using `this.setState()`.
2. **Lifecycle Methods**: They support lifecycle methods like `componentDidMount`, `componentDidUpdate`, and `componentWillUnmount`, allowing you to perform side-effects, manage subscriptions, or clean up resources.
3. **Props**: Like functional components, class components can receive props from their parent component and use them within their `render` method.

Key Steps:

1. **Import React**: Import the `React` library at the beginning of your file.
2. **Define Class**: Define a new class that extends `React.Component`.
3. **Constructor**: Implement a constructor to initialize state and bind methods if necessary.
4. **Render Method**: Implement a `render` method to define the component's UI using JSX.
5. **Export Component**: Export the class component for use in other parts of your application.

Class components were the traditional way to write components in React before the introduction of hooks in React 16.8. While functional components with hooks have become more popular due to their simplicity and flexibility, class components are still widely used, especially in legacy codebases or for components that require lifecycle methods or local state management.

Button.jsx:

```
import React from "react";

export default function Button(props) {
   let { action, title } = props;
   return <button onClick={action}>{title}</button>
}
```

App.js:

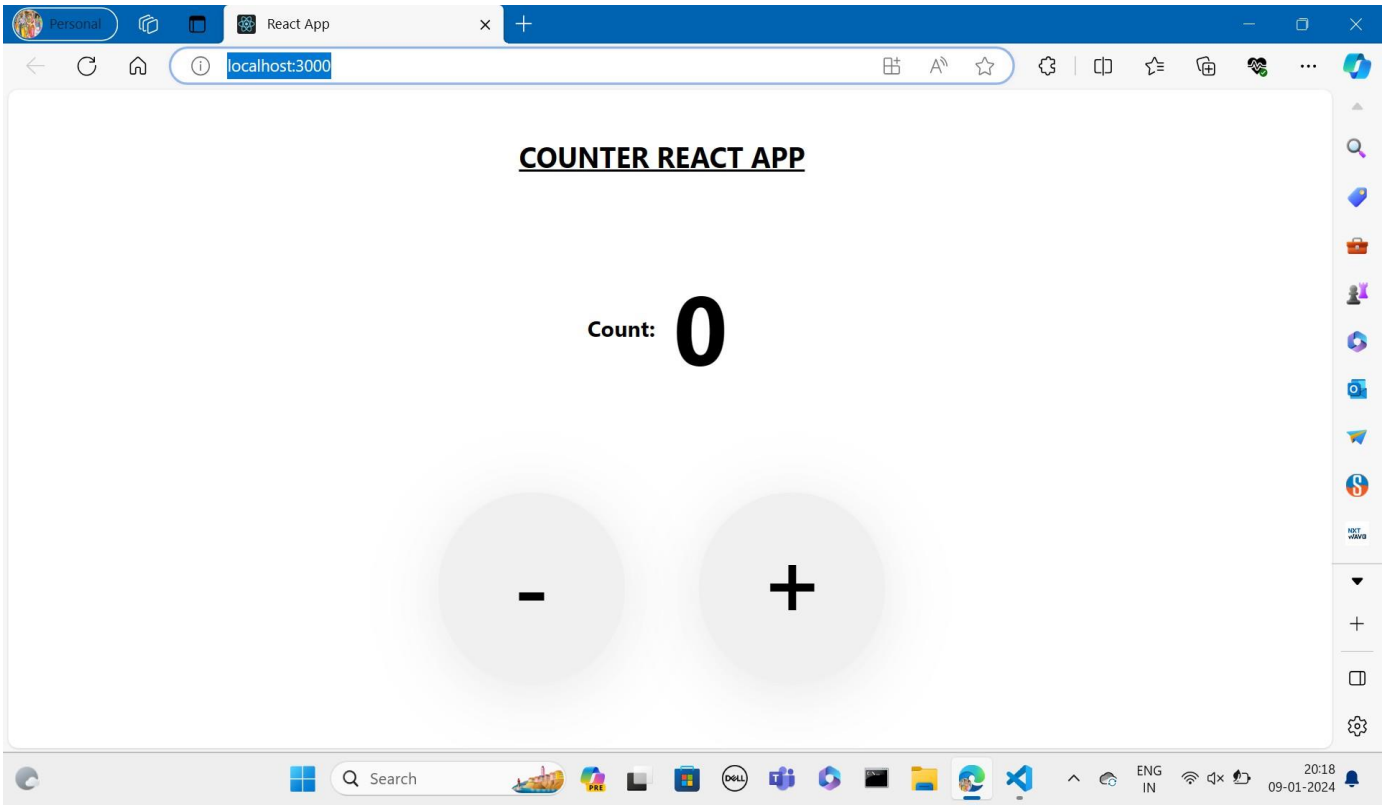
```
import React, { useState } from "react";
import Button from "./components/Button";
import "./assets/style.css";

export default function App() {
  const [count, setCount] = useState(0);

  let incrementCount = () => {
    setCount(count + 1);
  };

  let decrementCount = () => {
    setCount(count - 1);
  };

  return (
    <div className="app">
      <div>
        <div className="title-container">
          <h2>COUNTER REACT APP</h2>
          <div className="title-underline"></div>
        </div>
        <div class="count">
          <h3>Count:</h3>
          <h1>{count}</h1>
        </div>
        <div class="buttons">
          <Button title={"-"} action={decrementCount} />
          <Button title={"+"} action={incrementCount} />
        </div>
      </div>
    </div>
  );
}
```

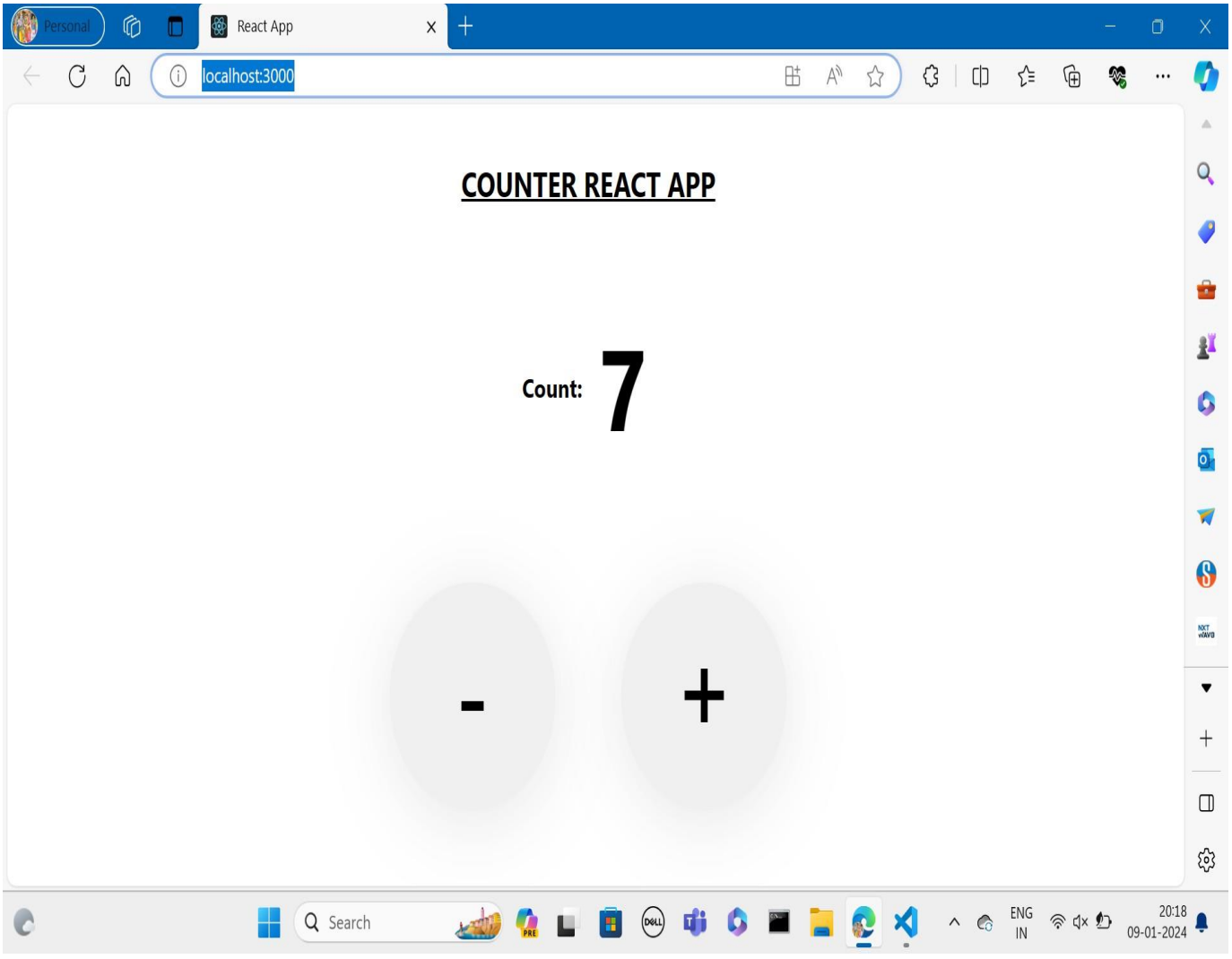
Style.css:

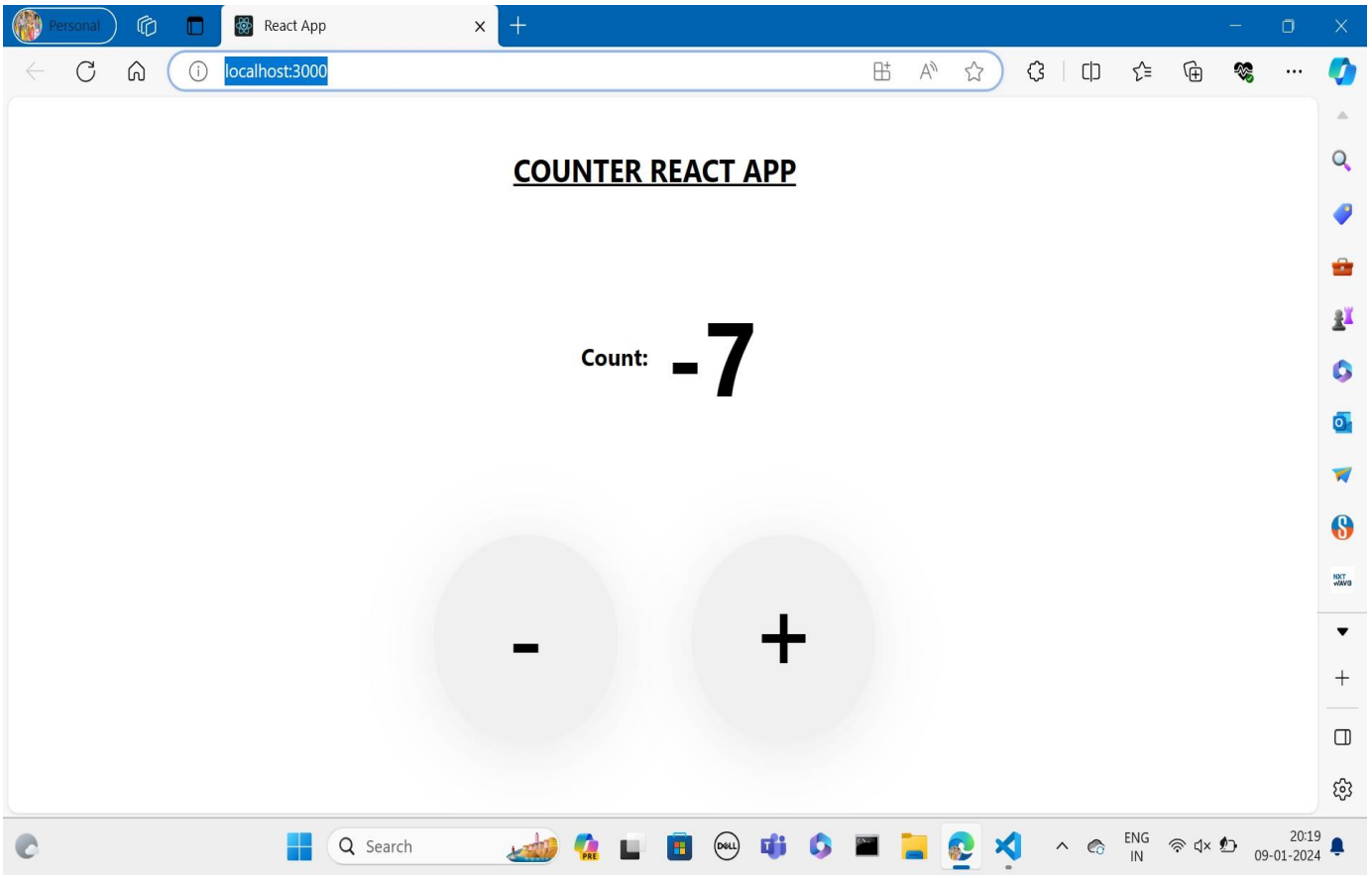
```
.app {
    display: flex;
    flex-direction: row;
    justify-content: center;
    margin-top: 10rem;
}

.app .count {
    margin-bottom: 2rem;
    text-align: center;
    display: flex;
    margin-left: 10rem;
}

.app .count h3{
    line-height: 8.6rem;
    margin-right: 1rem;
}

.app .count h1{
    font-size: 5rem;
    margin-top: 2rem;
```

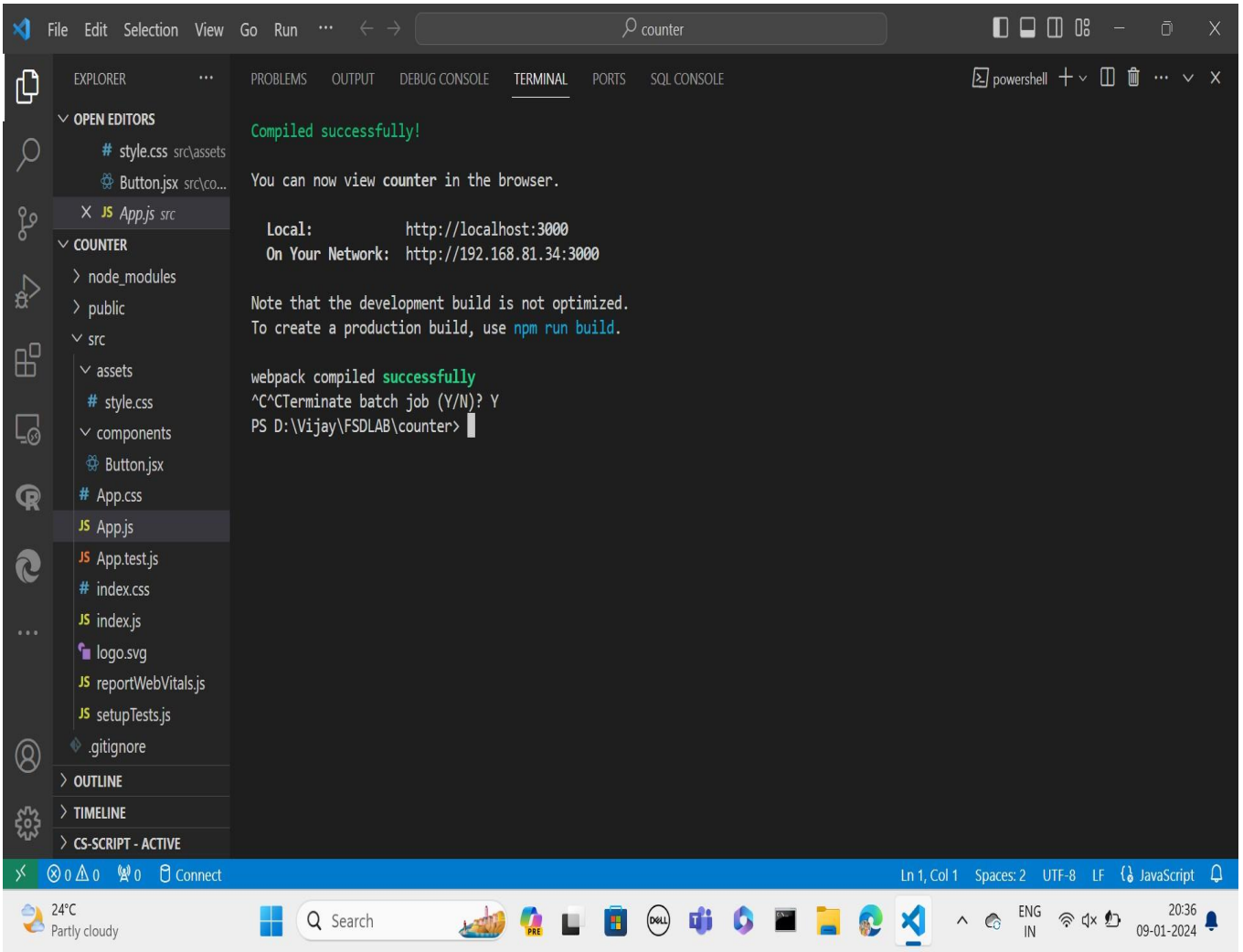
```
}

.app .buttons button {
    width: 10rem;
    height: 10rem;
    border-radius: 50%;
    border: none;
    margin: 0 2rem;
    font-size: 5em;
    box-shadow: 0 4px 32px 0 rgba(10,14,29,.02), 0 8px 64px 0 rgba(10,14,29,.08);
    cursor: pointer;
}

.app .buttons button:hover{
    font-size: 7em;
}

.app .buttons button:focus{
    outline: 0;
}

/* style.css */

body {
    display: flex;
    align-items: flex-start;
    justify-content: center;
    height: 100vh;
    margin: 0;
}

.app {
    text-align: center;
    margin-top: 20px; /*Adjust the top margin as needed*/
}

.title-container {
    margin-bottom: 20px;
}

.title-container h2 {
    text-decoration: underline;
    display: inline-block; /* Prevents the underline from extending full width*/
}

.count {
    margin-bottom: 20px;
}
```

OUTPUT:


Personal
React App
localhost:3000
COUNTER REACT APP
Count: 0
-
+
Search
ENG
IN
20:18
09-01-2024


Personal
React App
localhost:3000
COUNTER REACT APP
Count: 7
-
+
Search
ENG
IN
20:18
09-01-2024


React App
localhost:3000
COUNTER REACT APP
Count: -7
-
+
Search
ENG
IN
20:19
09-01-2024


File Edit Selection View Go Run
counter
EXPLORER
OPEN EDITORS
style.css src\assets
Button.jsx src\co...
App.js src
COUNTER
node_modules
public
src
assets
style.css
components
Button.jsx
App.css
App.js
App.test.js
index.css
index.js
logo.svg
reportWebVitals.js
setupTests.js
.gitignore
OUTLINE
TIMELINE
CS-SCRIPT - ACTIVE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL CONSOLE
powershell
Compiled successfully!
You can now view counter in the browser.
Local: http://localhost:3000
On Your Network: http://192.168.81.34:3000
Note that the development build is not optimized.
To create a production build, use npm run build.
webpack compiled successfully
^C^CTerminate batch job (Y/N)? Y
PS D:\Vijay\FSDLAB\counter>
Connect
Ln 1, Col 1 Spaces: 2 UTF-8 LF JavaScript
24°C
Partly cloudy
Search
ENG
IN
20:36
09-01-2024

EXPERIMENT NO: 07

AIM:

Create a Class component for Changing the color of the text given in React JS.

DESCRIPTION:

Creating a Class Component for Changing Text Color:

1. **Import React**: Import the `React` library at the beginning of your file.
2. **Define Class**: Define a new class that extends `React.Component`.
3. **State Initialization**: Implement a constructor to initialize state. You can have a state property like `color` to store the current text color.
4. **Method for Color Change**: Implement a method, say `changeColor`, to change the color of the text. This method can update the `color` state based on user input or other conditions.
5. **Render Method**: Implement a `render` method to define the component's UI using JSX. Use inline styles or CSS classes to apply the current color to the text.
6. **User Interaction**: You can add buttons or other UI elements with event handlers to trigger the color change method.
7. **Export Component**: Export the class component for use in other parts of your application.
8. **Library for UI**: ReactJS is a JavaScript library developed by Facebook for building user interfaces, particularly for single-page applications.
9. **Component-Based**: It follows a component-based architecture, allowing developers to build reusable UI components.
10. **Virtual DOM**: React uses a virtual DOM to improve performance by updating only the changed parts of the actual DOM.
11. **JSX**: React uses JSX (JavaScript XML), a syntax extension that allows you to write UI components using HTML-like syntax within JavaScript.
12. **Unidirectional Data Flow**: React promotes a unidirectional data flow, making it easier to understand and manage state changes.
13. **Hooks**: Introduced in React 16.8, hooks allow functional components to use state and other React features without writing a class.
14. **Declarative**: React uses a declarative programming style, where developers describe what they want to achieve, and React takes care of how to achieve it.
15. **Community and Ecosystem**: React has a large and active community, along with a rich ecosystem of libraries and tools that enhance its capabilities.
16. **Server-Side Rendering**: React can be used with server-side rendering frameworks like Next.js, improving SEO and initial page load times.
17. **Mobile Development**: With frameworks like React Native, you can use React principles to build mobile applications for iOS and Android platforms.

In this approach, you can manage the text color as a part of the component's state and provide user controls to change it. The component will re-render whenever the state changes, reflecting the updated text color in the UI.

App.js :

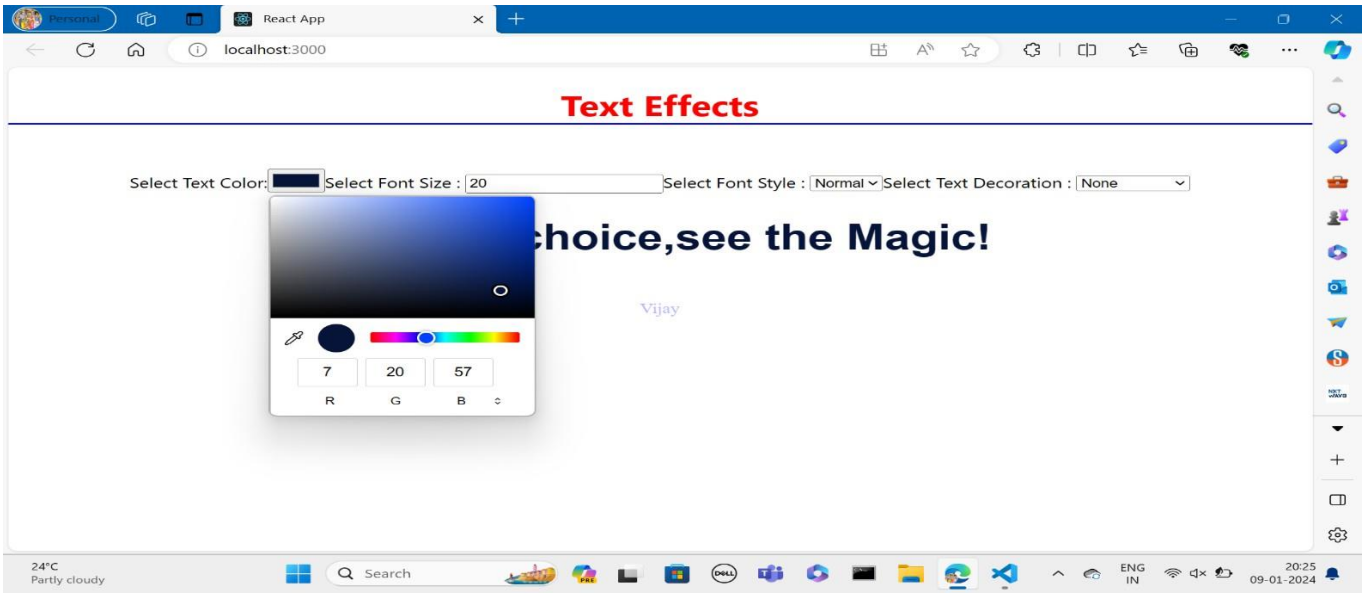
```
import React from 'react';
import ColorChanger from './ColorChanger';
function App() {
  return (
<div style={{textAlign:'center',marginTop:'10px'}}>
  <h1 style={{borderBottom:'2px solid blue',color:'Red'}}>Text Effects</h1>
  <ColorChanger/>
</div>
  );
}

export default App;
```

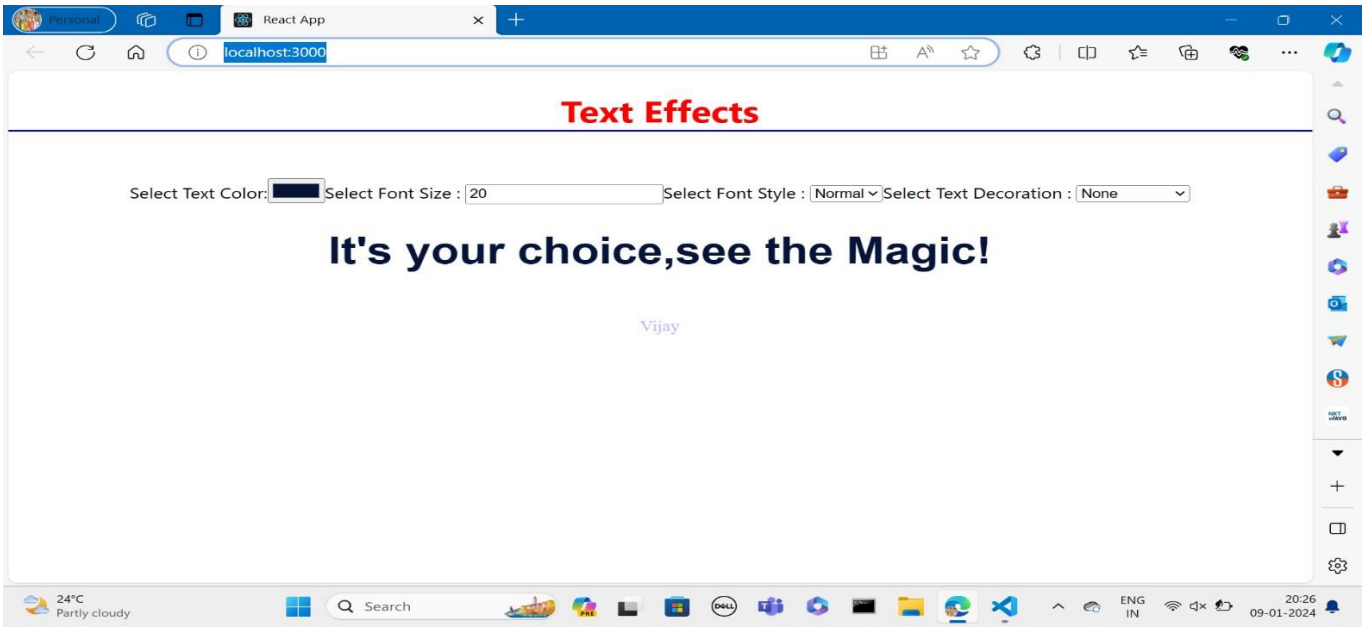
ColorChanger.js:

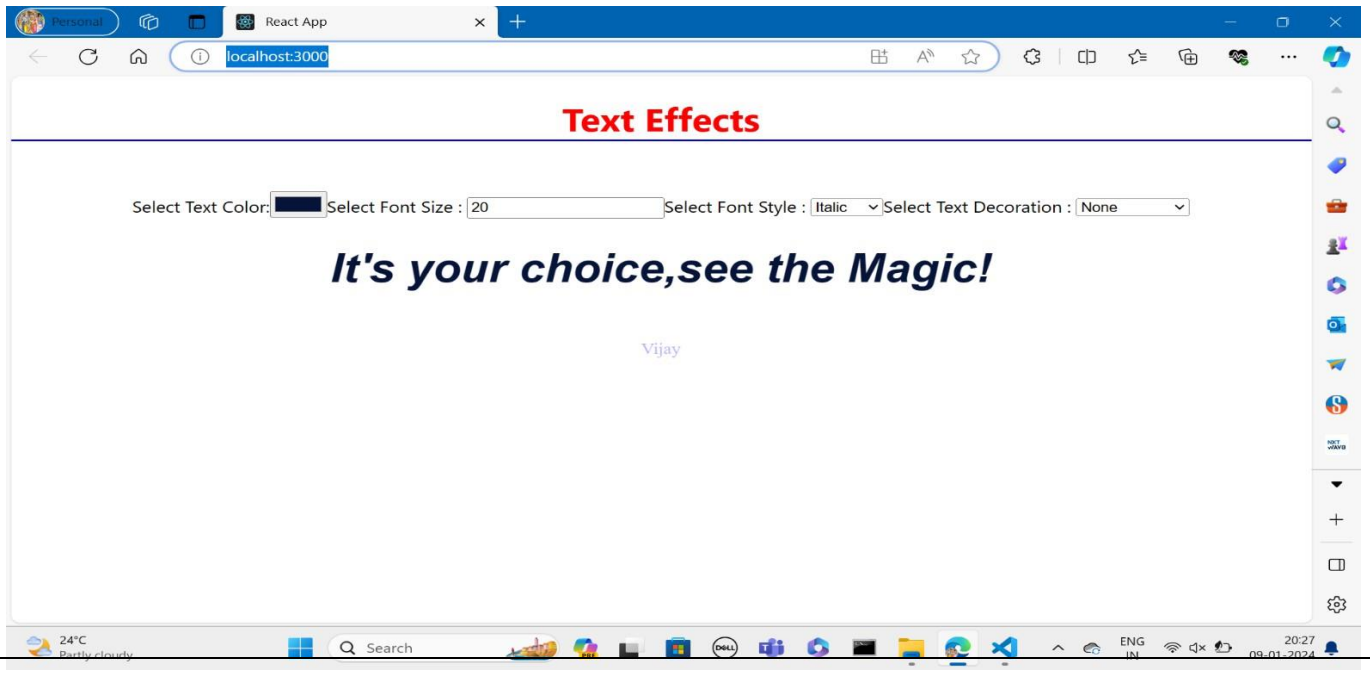
```
import React,{ Component} from 'react';
class ColorChanger extends Component{
   constructor(props){
      super(props);
      this.state = {
         textColor: 'black',
         fontSize: '20px',
         fontStyle: 'normal',
         textDecoration: 'none'
      };
      this.watermarkStyle = {
         position: 'absolute',
         top:'50%',
         left:'50%',
         transform:'translate(-50%,-50%)',
         zIndex:'-1',
         opacity:'0.3',
         fontFamily:'Arial sans-serif',
         fontSize:'Normal',
         color:'blue',
      };
   }

   handleColorChange = (event) => {
      this.setState({ textColor: event.target.value});
   };

   handleSizeChange = (event) => {
      this.setState({ fontSize: event.target.value + 'px '});
   };

   handleStyleChange = (event) => {
      this.setState({ fontStyle: event.target.value});
   };

   handleDecorationChange = (event) => {
      this.setState({ textDecoration: event.target.value});
   };

   render(){
```

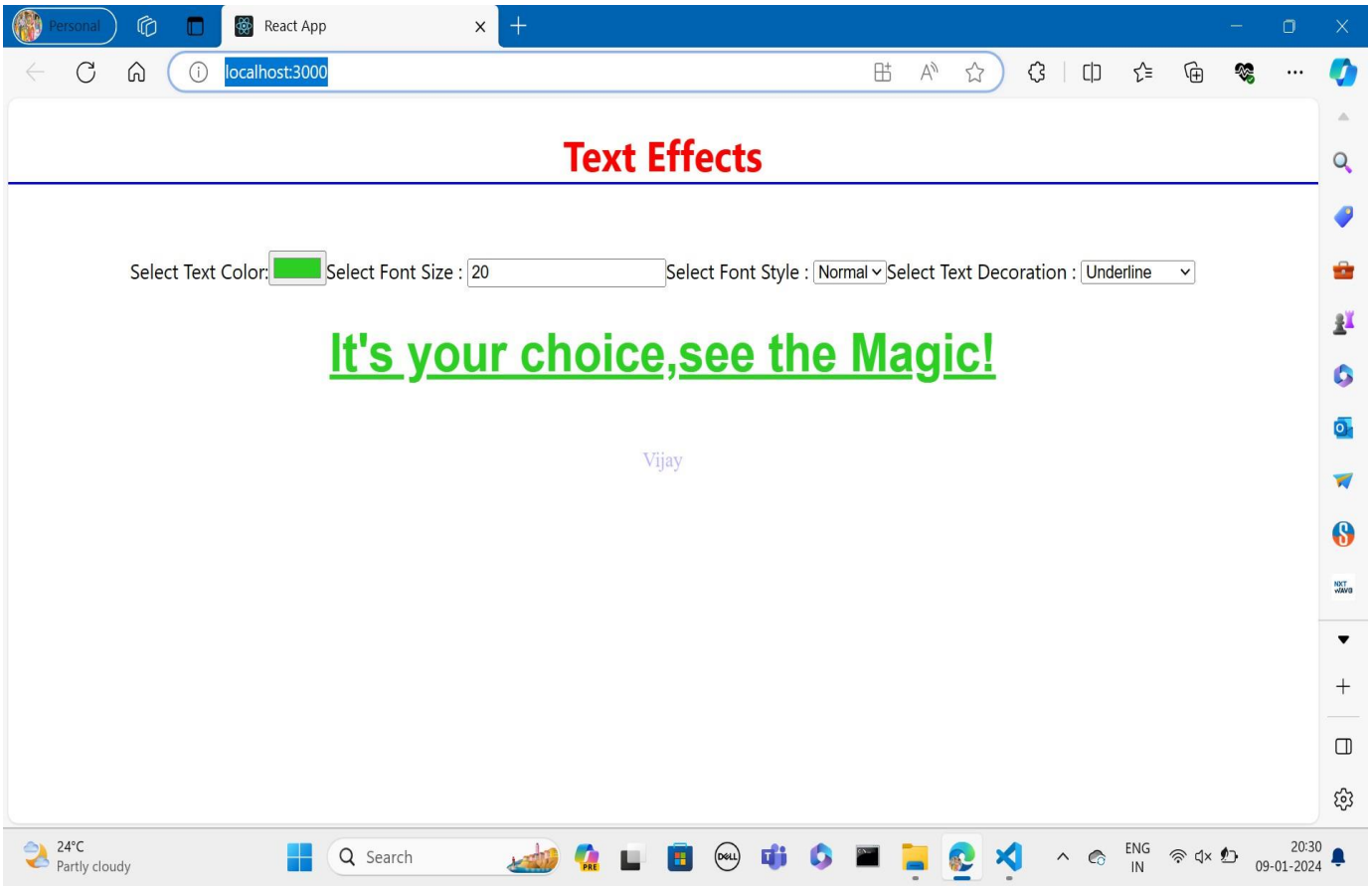
```
        const { textColor,fontSize,fontStyle,textDecoration } = this.state;

        return(
          <div style={{textAlign:'center', marginTop :'50px'}}>
            <div style={this.watermarkStyle}>**Vijay**</div>
            <label htmlFor='colorpicker'>Select Text Color:</label>
            <input
            type='color'
            id='colorpicker'
            value={textColor}
            onChange={this.handleColorChange}
            />

            <label htmlFor='fontSize'>Select Font Size : </label>
            <input
            type='number'
            id='fontSize'
            value={parseInt(fontSize)}
            onChange={this.handleSizeChange}
            />

            <label htmlFor='fontStyle'>Select Font Style : </label>
            <select id='fontStyle' value={fontStyle} onChange={this.handleStyleChange}>
              <option value = 'normal'>Normal</option>
              <option value ='italic'>Italic</option>
            </select>

            <label htmlFor='textDecoration'>Select Text Decoration : </label>
            <select id='textDecoration' value={textDecoration} onChange={this.handleDecorationChange}>
              <option value={"none"}>None</option>
              <option value={"underline"}>Underline</option>
              <option value={"overline"}>Overline</option>
              <option value={"line-through"}>Line Through</option>
            </select>
            <div style={{color:textColor,marginTop:'20px',fontFamily:'Arial,sans-
serif',fontSize:fontSize,fontStyle,textDecoration:textDecoration}}>
              <h1>It's your choice,see the Magic!</h1>
            </div>

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

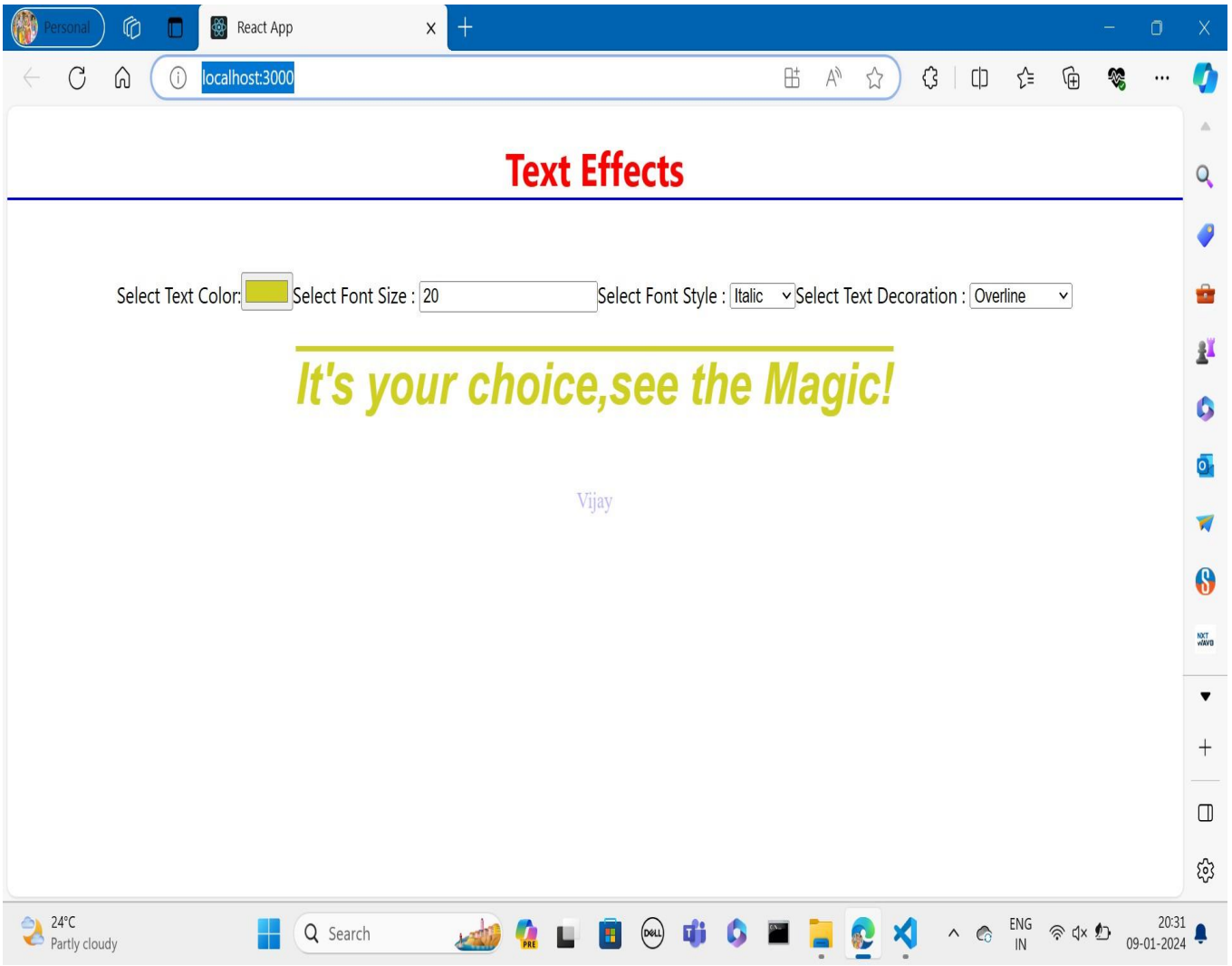
OUTPUT:

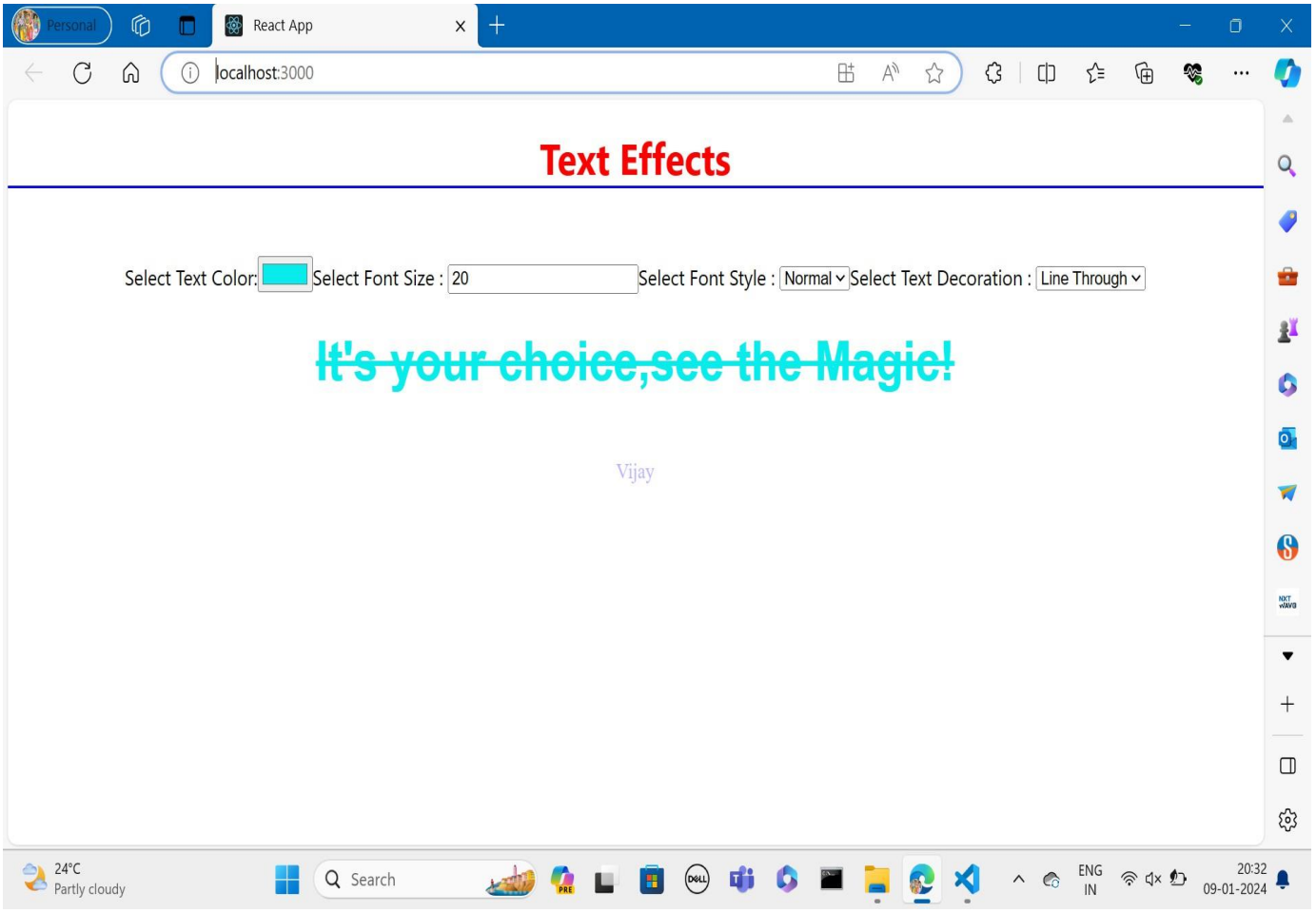

React App
localhost:3000
Text Effects
Select Text Color:
Select Font Size :
20
Select Font Style :
Normal
Select Text Decoration :
None
choice,see the Magic!
Vijay
7
20
57
R
G
B
24°C
Partly cloudy
Search
ENG
IN
20:25
09-01-2024

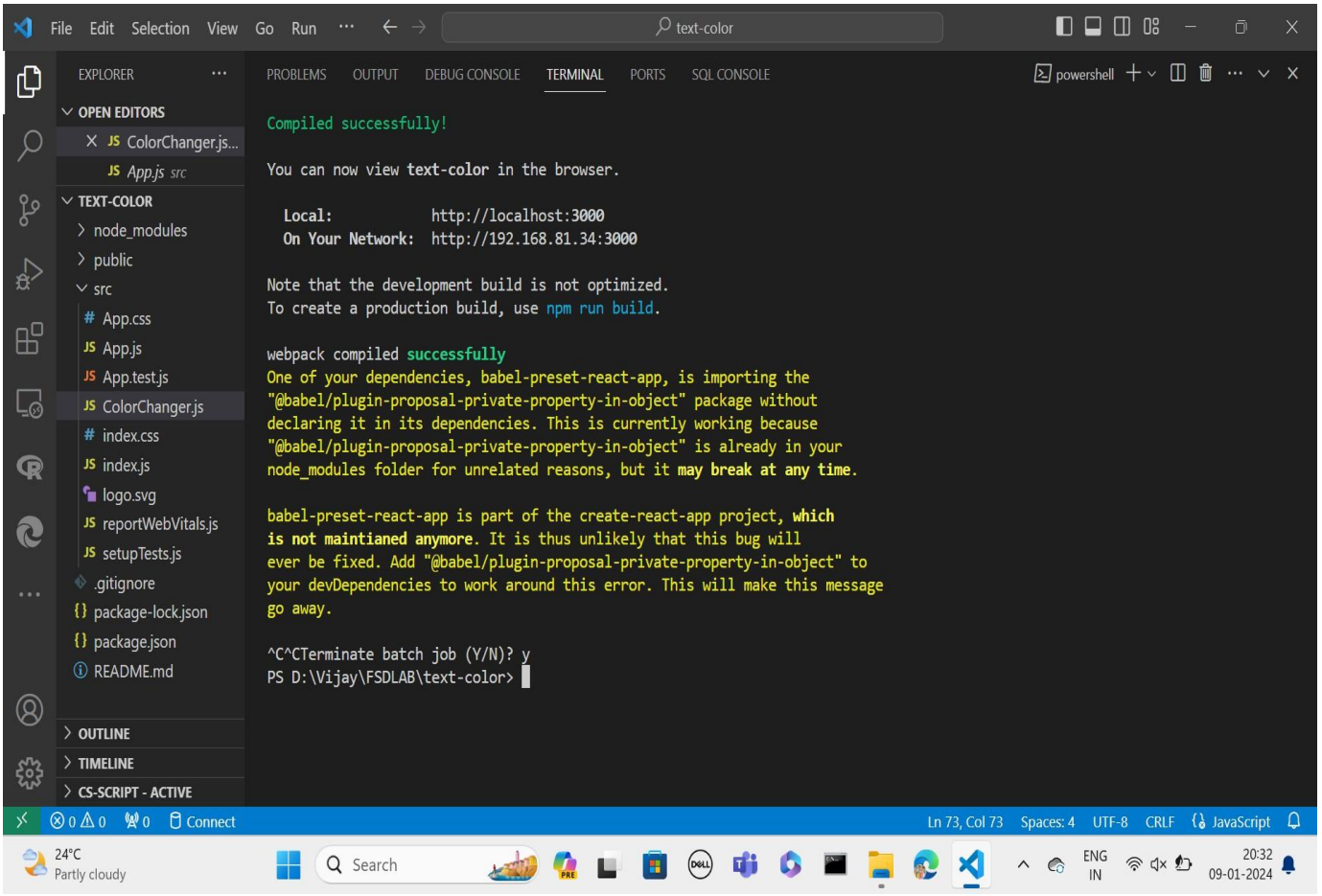

React App
localhost:3000
Text Effects
Select Text Color:
Select Font Size :
20
Select Font Style :
Normal
Select Text Decoration :
None
It's your choice,see the Magic!
Vijay
24°C
Partly cloudy
Search
ENG
IN
20:26
09-01-2024


React App
localhost:3000
Text Effects
Select Text Color:
Select Font Size :
20
Select Font Style :
Italic
Select Text Decoration :
None
It's your choice,see the Magic!
Vijay
24°C
Partly cloudy
Search
ENG
IN
20:27
09-01-2024


React App
localhost:3000
Text Effects
Select Text Color:
Select Font Size : 20
Select Font Style : Normal
Select Text Decoration : Underline
It's your choice,see the Magic!
Vijay
24°C
Partly cloudy
Search
ENG
IN
20:30
09-01-2024


React App
localhost:3000
Text Effects
Select Text Color:
Select Font Size : 20
Select Font Style : Italic
Select Text Decoration : Overline
It's your choice,see the Magic!
Vijay
24°C
Partly cloudy
Search
ENG
IN
20:31
09-01-2024


Personal
React App
localhost:3000
Text Effects
Select Text Color:
Select Font Size :
20
Select Font Style :
Normal
Select Text Decoration :
Line Through
It's your choice,see the Magic!
Vijay
24°C
Partly cloudy
Search
ENG
IN
20:32
09-01-2024


File Edit Selection View Go Run
text-color
EXPLORER
OPEN EDITORS
ColorChanger.js...
App.js src
TEXT-COLOR
node_modules
public
src
App.css
App.js
App.test.js
ColorChanger.js
index.css
index.js
logo.svg
reportWebVitals.js
setupTests.js
.gitignore
package-lock.json
package.json
README.md
OUTLINE
TIMELINE
CS-SCRIPT - ACTIVE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL CONSOLE
powershell
Compiled successfully!
You can now view text-color in the browser.
Local: http://localhost:3000
On Your Network: http://192.168.81.34:3000
Note that the development build is not optimized.
To create a production build, use npm run build.
webpack compiled successfully
One of your dependencies, babel-preset-react-app, is importing the
"@babel/plugin-proposal-private-property-in-object" package without
declaring it in its dependencies. This is currently working because
"@babel/plugin-proposal-private-property-in-object" is already in your
node_modules folder for unrelated reasons, but it may break at any time.
babel-preset-react-app is part of the create-react-app project, which
is not maintianed anymore. It is thus unlikely that this bug will
ever be fixed. Add "@babel/plugin-proposal-private-property-in-object" to
your devDependencies to work around this error. This will make this message
go away.
^C^CTerminate batch job (Y/N)? y
PS D:\Vijay\FSDLAB\text-color>
Connect
Ln 73, Col 73 Spaces: 4 UTF-8 CRLF JavaScript
24°C
Partly cloudy
Search
ENG
IN
20:32
09-01-2024

EXPERIMENT NO: 08

AIM:

Class a Class Component for viewing an array of objects in a tabular form.

DESCRIPTION:

Import React: Import the React library at the beginning of your file.

Define Class: Define a new class that extends React.Component.

State Initialization: Implement a constructor to initialize state. You can have a state property like data to store the array of objects that you want to display in the table.

Render Method: Implement a render method to define the component's UI using JSX. Create a table structure with headers and rows to display the object properties.

Mapping Data: Inside the render method, map over the data state to generate table rows. For each object in the array, create a table row and populate it with the object's properties as table data.

Table Headers: Define table headers based on the object keys or custom headers if needed.

Export Component: Export the class component for use in other parts of your application.

1. **Library for UI**: ReactJS is a JavaScript library developed by Facebook for building user interfaces, particularly for single-page applications.
2. **Component-Based**: It follows a component-based architecture, allowing developers to build reusable UI components.
3. **Virtual DOM**: React uses a virtual DOM to improve performance by updating only the changed parts of the actual DOM.
4. **JSX**: React uses JSX (JavaScript XML), a syntax extension that allows you to write UI components using HTML-like syntax within JavaScript.
5. **Unidirectional Data Flow**: React promotes a unidirectional data flow, making it easier to understand and manage state changes.
6. **Hooks**: Introduced in React 16.8, hooks allow functional components to use state and other React features without writing a class.
7. **Declarative**: React uses a declarative programming style, where developers describe what they want to achieve, and React takes care of how to achieve it.
8. **Community and Ecosystem**: React has a large and active community, along with a rich ecosystem of libraries and tools that enhance its capabilities.
9. **Server-Side Rendering**: React can be used with server-side rendering frameworks like Next.js, improving SEO and initial page load times.
10. **Mobile Development**: With frameworks like React Native, you can use React principles to build mobile applications for iOS and Android platforms.

App.js :

```
import React from 'react';
import TableView from './TableView';
import './index.css';

const App = () => {
  const data = [
    {ID : 1 , Name : 'Vijay' ,   Age : 20},
    {ID : 2 , Name : 'Ram'   ,   Age : 21},
    {ID : 3 , Name : 'Jaanu' ,   Age : 20},
    {ID : 4 , Name : 'Shiva' ,   Age : 21},
    {ID : 5 , Name : 'Nani'  ,   Age : 21},

  ];

  return (
    <div>
      <h1 style = {{ textAlign : 'center' , borderBottom : '2px solid blue' , paddingTop : '20px',marginBottom : '100px'}}>
        Array of Objects - TableView
      </h1>
      <TableView data= {data} />
    </div>
  );
};

export default App;
```

TableView.js :

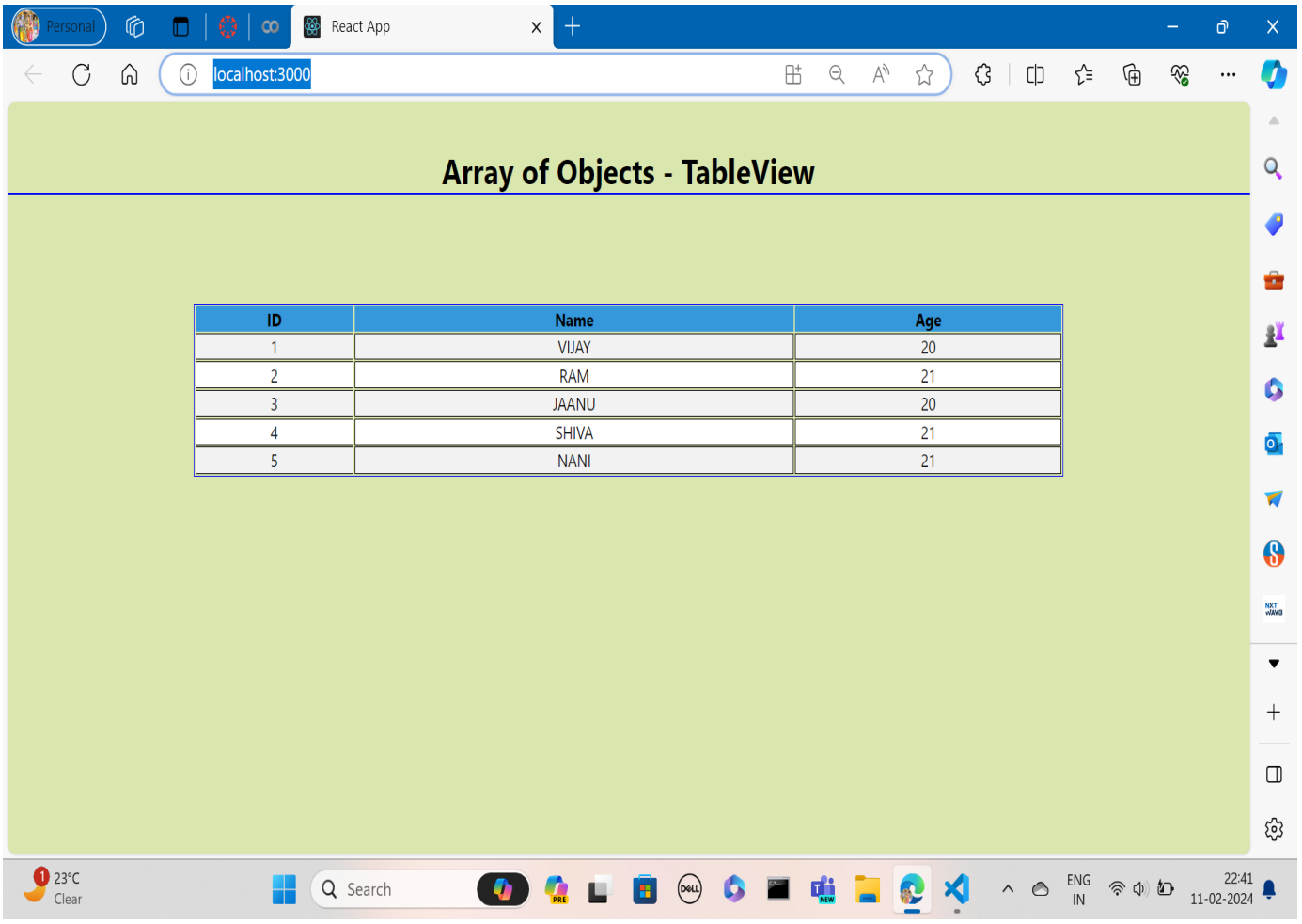
```
import React,{Component} from 'react';

class TableView extends Component {
   renderTableHeader() {
      const header = Object.keys(this.props.data[0]);
      return header.map((key) => <th key={key}>{key}</th>);
}

renderTableData() {
   return this.props.data.map((item,index) => {
      const rowStyle = {
         backgroundColor : index % 2 === 0 ? '#f2f2f2' : 'white', // Alternating background colors
      };

      return (
         <tr key ={index} style = {rowStyle}>
            {Object.values(item).map((value,index) => (
               <td key={index} style={{ border:'1px solid black',textAlign:'center'}}>
                  {value}
               </td>
            ))}
         </tr>
      );
   });
};

   render() {
      const tableStyle = {
```

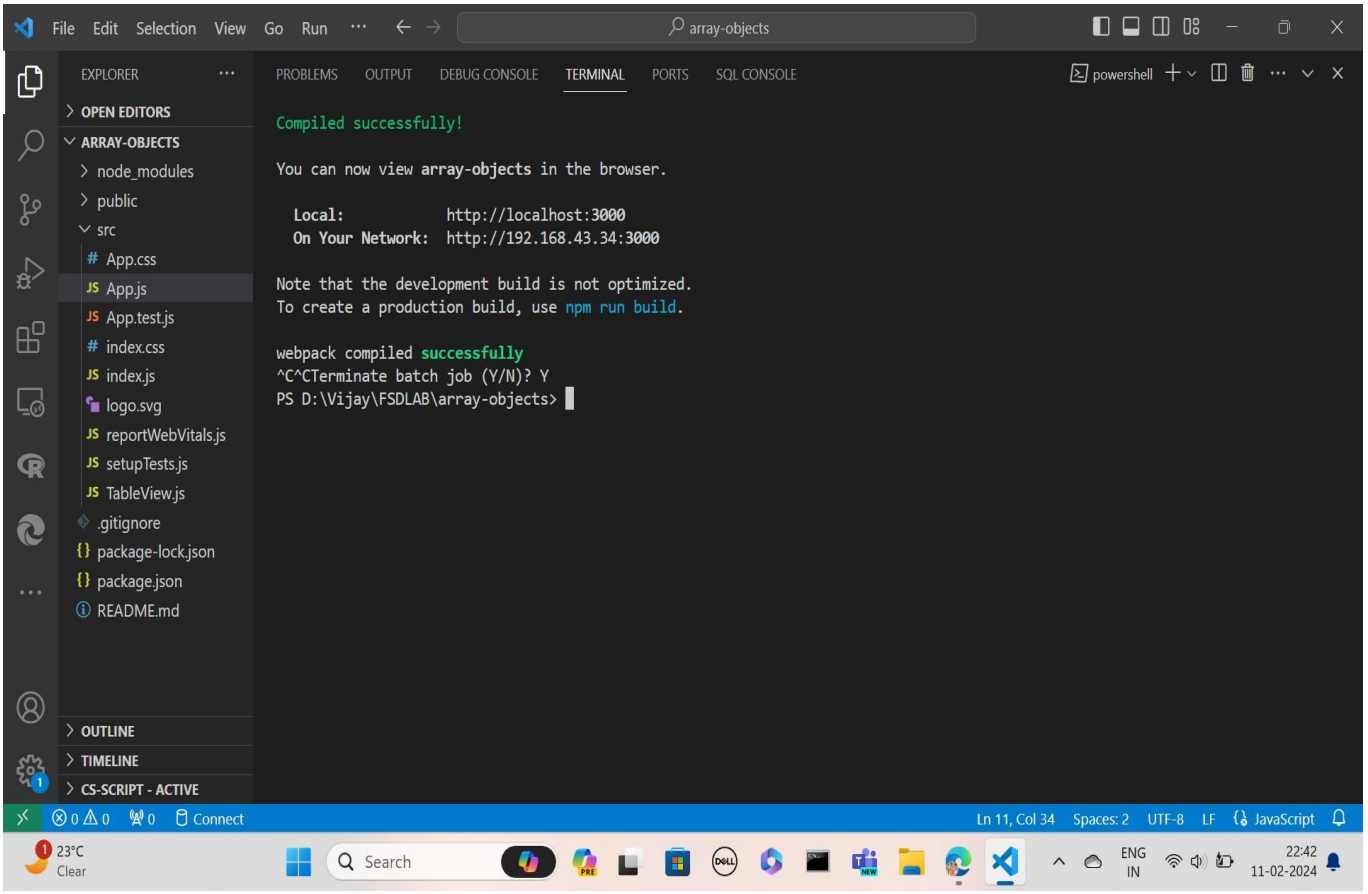
```
            borderCollapse : 'collaspe',
            width : '70%',
            top : 30,
            margin : '20px auto',
            border : '1px solid blue',
        };
        return (
            <div>
                <table style={tableStyle}>
                    <thead style={{ backgroundColor : '#3498db'}}>
                        <tr>{this.renderTableHeader()}</tr>
                    </thead>
                    <tbody>{this.renderTableData()}</tbody>
                </table>
            </div>
        )
    }
}

export default TableView;
```

index.css :

```
body {
  margin: 0;
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',
    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
    sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  background-color: rgb(217,230,173);
}

code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
    monospace;
}
```

OUTPUT:


React App
localhost:3000
Array of Objects - TableView
ID
Name
Age
1
VIJAY
20
2
RAM
21
3
JAANU
20
4
SHIVA
21
5
NANI
21
Search
ENG
IN
22:41
11-02-2024


File Edit Selection View Go Run
array-objects
EXPLORER
OPEN EDITORS
ARRAY-OBJECTS
node_modules
public
src
App.css
App.js
App.test.js
index.css
index.js
logo.svg
reportWebVitals.js
setupTests.js
TableView.js
.gitignore
package-lock.json
package.json
README.md
OUTLINE
TIMELINE
CS-SCRIPT - ACTIVE
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
SQL CONSOLE
powershell
Compiled successfully!
You can now view array-objects in the browser.
Local: http://localhost:3000
On Your Network: http://192.168.43.34:3000
Note that the development build is not optimized.
To create a production build, use npm run build.
webpack compiled successfully
^C^CTerminate batch job (Y/N)? Y
PS D:\Vijay\FSDLAB\array-objects>
Ln 11, Col 34
Spaces: 2
UTF-8
LF
JavaScript
Search
ENG
IN
22:42
11-02-2024

EXPERIMENT NO: 09

AIM:

Display a digital clock in React JS.

DESCRIPTION:

Implementing Analog and Digital Clock in ReactJS:

Analog Clock:

Canvas Element: Use the HTML5 canvas element to draw the clock face and hands.

Mathematics: Calculate the positions of the clock hands (hour, minute, and second) using trigonometric functions like sine and cosine.

Interval: Use setInterval to update the clock hands every second by redrawing the canvas.

Digital Clock:

State & Lifecycle: Maintain a state property to store the current time, which will be updated every second using the setInterval method.

Formatting: Use JavaScript's Date object to get the current time and format it into a readable string representing hours, minutes, and seconds.

Rendering: Display the formatted time string in the component's render method to show the digital clock.

General Steps:

State Initialization: Initialize state properties for both clocks, such as currentTime for the digital clock and second, minute, and hour for the analog clock.

Lifecycle Methods: Use componentDidMount to start the clock's ticking when the component mounts and componentWillUnmount to clear the interval when the component unmounts to avoid memory leaks.

Rendering: Render the analog clock using a canvas element and the digital clock using a div or span element to display the time string.

Styling: Apply CSS to style the clock components, making them visually appealing and ensuring they fit well within the layout of your application.

By following these steps, you can create both analog and digital clocks in ReactJS. The analog clock provides a visual representation of time using graphics, while the digital clock displays the time in a readable format. Implementing these clocks involves using JavaScript to handle time calculations and ReactJS to manage component rendering and state updates.

App.js :

```
import React from "react";
import "./style.css";
import Clock from "./clock";

class DigitalClock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      time: new Date()
    };
  }

  componentDidMount() {
    this.timerId = setInterval(() => {
      this.setState({
        time: new Date()
      });
    }, 1000);
  }

  componentWillUnmount() {
    clearInterval(this.timerId);
  }

  render() {
    const { time } = this.state;
    const hours = time.getHours().toString().padStart(2, "0");
    const minutes = time.getMinutes().toString().padStart(2, "0");
    const seconds = time.getSeconds().toString().padStart(2, "0");
    const ampm = hours >= 12 ? 'PM' : 'AM';
    const displayHours = (hours % 12 || 12).toString().padStart(2, '0');
    const options = { day: 'numeric', month: 'short', year: 'numeric' };
    const formattedDate = new Intl.DateTimeFormat('en-US', options).format(time);

    return (
      <div className="digital-clock">
        <h2>
          {`${displayHours}:${minutes}:${seconds}`}
          <sup style={{ fontSize: '0.4em', verticalAlign: 'top' }}>{ampm}</sup>
        </h2>
        <div className="date-container" style={{ textAlign: 'center' }}>
          <b><p style={{ fontSize: '0.9em', margin: '0' }}>{formattedDate}</p></b>
        </div>
      </div>
    );
  }
}

function App() {
  return (
    <div>
      <h1 style={{ textAlign: 'center', borderBottom: '2px solid orange', paddingTop: '20px', color: 'blue' }}>
        Analog - Digital Clock
      </h1>
      <div className="clocks-container">
        <Clock />
```

```
      <DigitalClock />
    </div>
    <div className="watermark" style={{color:"blue"}}>GOLI VIJAY KUMAR</div>
  </div>
 );
}

export default App;
```

clock.js :

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

export default class Clock extends Component {
 constructor(props) {
  super(props);
  this.state = {
   time: new Date()
  };
 }

 componentDidMount() {
  this.timerId = setInterval(() => {
   this.setState({
    time: new Date()
   });
  }, 1000);
 }

 componentWillUnmount() {
  clearInterval(this.timerId);
 }

 render() {
  return (
   <div className="clock">
    <div className="analog-clock">
     <div
      className="hour_hand"
      style={{
       transform: `rotateZ(${this.state.time.getHours() * 30}deg)`
      }}
     />
     <div
      className="min_hand"
      style={{
       transform: `rotateZ(${this.state.time.getMinutes() * 6}deg)`
      }}
     />
     <div
      className="sec_hand"
      style={{
       transform: `rotateZ(${this.state.time.getSeconds() * 6}deg)`
      }}
     />
     <span className="twelve">12</span>
     <span className="one">1</span>
     <span className="two">2</span>
     <span className="three">3</span>
```

```
          <span className="four">4</span>
          <span className="five">5</span>
          <span className="six">6</span>
          <span className="seven">7</span>
          <span className="eight">8</span>
          <span className="nine">9</span>
          <span className="ten">10</span>
          <span className="eleven">11</span>
        </div>
      </div>
    );
  }
}
```

style.css :

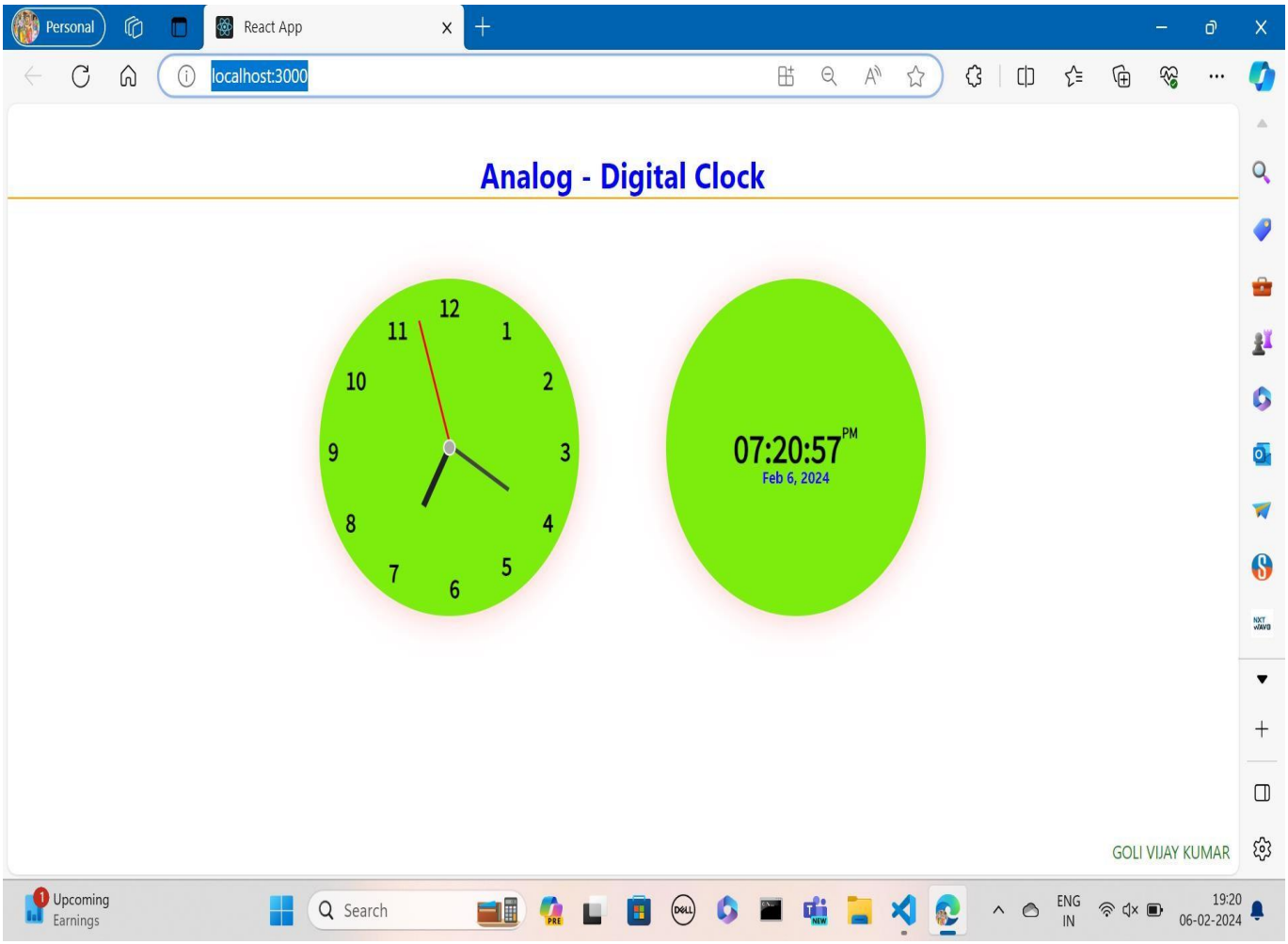
```
@import url("https://fonts.googleapis.com/css?family=Source+Sans+Pro&display=swap");

body {
  margin: 0; /* Remove default body margin */
}

.App {
  font-family: sans-serif;
  position: relative;
}

.watermark {
  position: absolute;
  bottom: 10px;
  right: 10px;
  color: rgba(0, 0, 0, 0.5); /* Adjust the color and opacity of the watermark */
  font-size: 16px;
}

.clocks-container {
  display: flex;
  justify-content: center;
  align-items: center;
}

.clock, .digital-clock {
  width: 300px;
  height: 300px;

  border-radius: 50%;
  position: relative;
  box-shadow: 0 2px 30px rgba(238, 5, 5, 0.2);
  font-size: 24px;
  color: #000000;
  text-align: center;
  margin: 50px; /* Add some margin for spacing between clocks */
  background-color:skyblue;
}

.clock::after {
  background: #aaa;
  content: "";
```

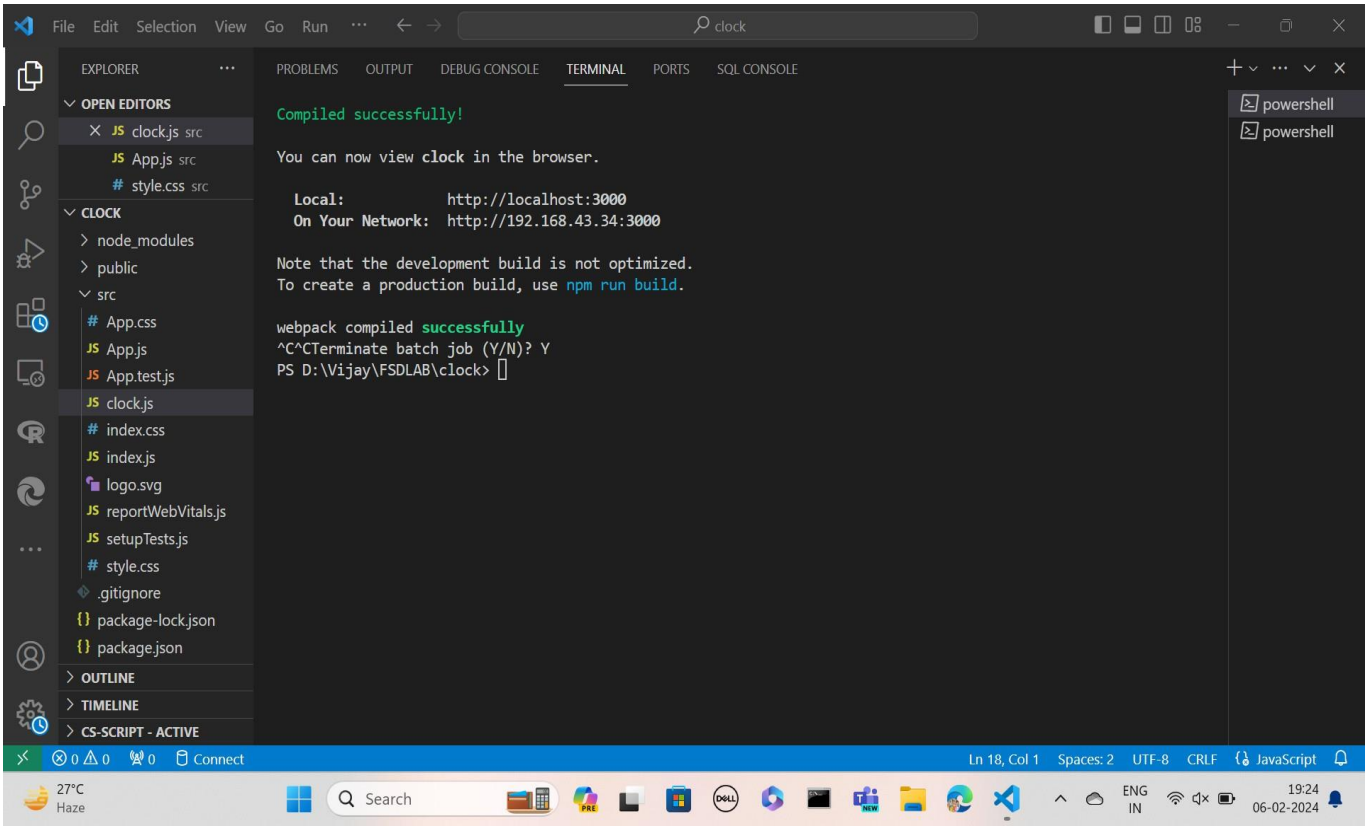
```
  width: 12px;
  height: 12px;
  border-radius: 50%;
  position: absolute;
  z-index: 2;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
  border: 2px solid #fff;
}

.digital-clock h2 {
  font-family: "Perfograma","Source Sans Pro", sans-serif;
  font-weight: 700;
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
  margin: 0;
}


.hour_hand {
  position: absolute;
  width: 6px;
  height: 60px;
  background: #222;
  top: 30%;
  left: 49%;
  transform-origin: bottom;
}

.min_hand {
  position: absolute;
  width: 4px;
  height: 80px;
  background: #444;
  top: 22.5%;
  left: 49%;
  transform-origin: bottom;
}

.sec_hand {
  position: absolute;
  width: 2px;
  height: 118px;
  background: red;
  top: 10.5%;
  left: 50%;
  transform-origin: bottom;
}

.clock span {
  position: absolute;
  font-family: "Source Sans Pro", sans-serif;
  font-weight: 700;
}
```

```
.twelve {
  top: 10px;
  left: 46%;
}

.one {

  top: 10%;
  right: 26%;
}

.eleven {
  top: 10%;
  left: 26%;
}

.two {
  top: 25%;
  right: 10%;
}

.three {
  right: 10px;
  top: 46%;
}

.four {
  right: 30px;
  top: 67%;
}

.five {
  right: 78px;
  top: 80%;
}

.six {
  bottom: 10px;
  left: 50%;
}

.seven {
  left: 80px;
  top: 82%;
}

.eight {
  left: 30px;
  top: 67%;
}

.nine {
  left: 10px;
  top: 46%;
}
```

```
.clock__text-ampm {
  font-size: var(--tiny-font-size);
  color: var(--title-color);
  font-weight: var(--font-medium);
  margin-left: var(--mb-0-25);
}

.date-container {
  position: absolute;
  bottom: 35%; /* Adjust the position of the date container */

  left: 50%;
  transform: translate(-50%, -50%);
  margin: 0;
  font-size: 16px;
  color: #2b08f7;
}

.clock__date {
  text-align: center;
  font-size: var(--small-font-size);
  font-weight: var(--font-medium);
}

.ten {
  top: 25%;
  left: 10%;
}
```

OUTPUT:


Personal
React App
localhost:3000
Analog - Digital Clock
12
1
2
3
4
5
6
7
8
9
10
11
07:20:57PM
Feb 6, 2024
GOLI VIJAY KUMAR
Upcoming
Earnings
Search
ENG
IN
19:20
06-02-2024


File Edit Selection View Go Run
clock
EXPLORER
OPEN EDITORS
clock.js src
App.js src
style.css src
CLOCK
node_modules
public
src
App.css
App.js
App.test.js
clock.js
index.css
index.js
logo.svg
reportWebVitals.js
setupTests.js
style.css
.gitignore
package-lock.json
package.json
OUTLINE
TIMELINE
CS-SCRIPT - ACTIVE
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
SQL CONSOLE
Compiled successfully!
You can now view clock in the browser.
Local: http://localhost:3000
On Your Network: http://192.168.43.34:3000
Note that the development build is not optimized.
To create a production build, use npm run build.
webpack compiled successfully
^C^CTerminate batch job (Y/N)? Y
PS D:\Vijay\FSDLAB\clock>
powershell
powershell
Connect
Ln 18, Col 1
Spaces: 2
UTF-8
CRLF
JavaScript
27°C
Haze
Search
ENG
IN
19:24
06-02-2024

EXPERIMENT NO: 10

AIM:

Demonstrate hooks concepts using ReactJS.

DESCRIPTION:

useState

Initialization: useState allows you to declare state variables in functional components. You can initialize a state variable with an initial value.

Updating State: It provides a function to update the state variable, ensuring immutability by replacing the old state with the new state.

Array Destructuring: useState returns an array with the current state value and a function to update it. You can use array destructuring to easily access these values.

Multiple States: You can declare multiple state variables by calling useState multiple times in a single component.

Lazy Initialization: You can provide a function as an argument to useState to compute the initial state lazily.

Functional Updates: The update function can accept a function to update the state based on the previous state, which is useful for complex state transitions.

State Batching: React batches multiple state updates into a single re-render to optimize performance.

State Hooks Isolation: Each call to useState is independent and doesn't share state with other useState calls or affect props or other state variables.

Derived State: You can derive a state variable from props or other state variables by computing the value inside the component's body.

Debugging: useState simplifies debugging by isolating state logic within functional components, making it easier to understand and maintain.

useContext

Context Creation: useContext allows you to access the value of a React context from within a functional component.

Context Consumer: It provides a way to consume context values without using the Context.Consumer component.

Context Provider: useContext works in conjunction with Context.Provider to provide context values to nested child components.

Dynamic Context: You can use useContext with dynamically changing contexts where the value can be updated over time.

Multiple Contexts: You can use useContext to consume multiple contexts within a single component by calling it multiple times.

Avoids Prop Drilling: useContext helps in avoiding prop drilling by providing a more direct way to access shared values across components.

Refactoring: It simplifies component refactoring by reducing the need to pass props down through intermediate components.

Custom Hooks: You can create custom hooks that leverage useContext to provide more granular control over context consumption.

Optimization: useContext can improve performance by preventing unnecessary re-renders when context values change.

Context Composition: It allows for flexible composition of contexts, enabling complex state management patterns without additional libraries.

useEffect

Lifecycle Method: useEffect is a lifecycle method in functional components that replaces lifecycle methods like componentDidMount, componentDidUpdate, and componentWillUnmount.

Side Effects: It's used for executing side effects like data fetching, DOM manipulation, and subscriptions in functional components.

Dependency Array: useEffect accepts a dependency array to control when the effect runs based on changes to its dependencies.

Cleanup: It provides a cleanup function to perform necessary cleanup tasks when the component unmounts or before re-running the effect.

Conditional Effects: You can conditionally run effects based on certain conditions by using conditional statements inside useEffect.

Multiple Effects: You can use multiple useEffect hooks in a single component to separate different concerns.

Async Effects: useEffect can work with asynchronous operations by using async functions or promises inside the effect.

Global Effects: It enables global side effects that run once when the component mounts, simulating componentDidMount with an empty dependency array.

Effect Dependencies: Understanding the dependency array is crucial to avoid unintended effects or infinite loops caused by improper dependencies.

Debugging: useEffect simplifies debugging by isolating side effects and providing a clear structure to manage component lifecycle and state effects.

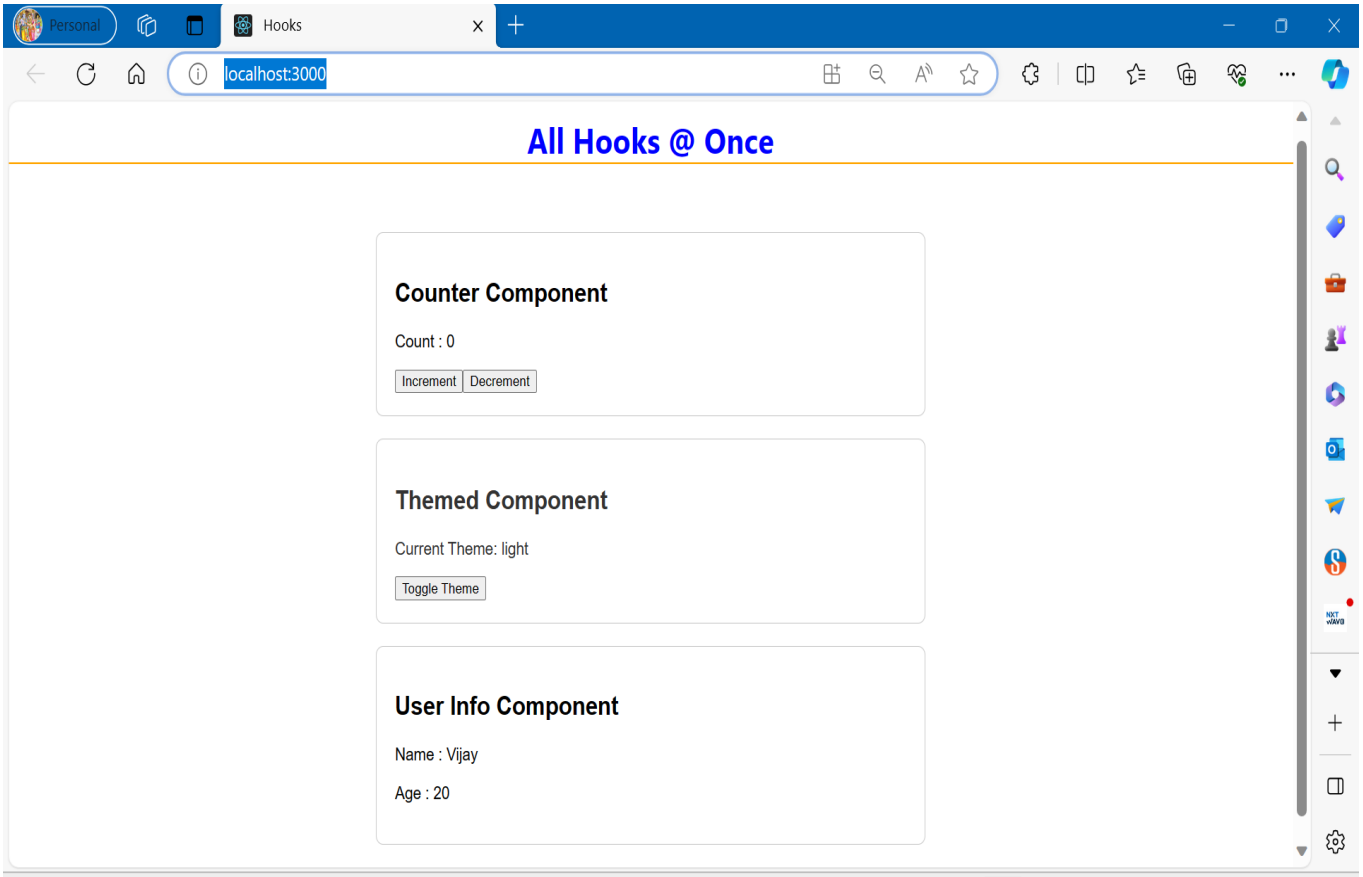
These points provide a comprehensive overview of useState, useContext, and useEffect, highlighting their key features, benefits, and best practices in React functional components.

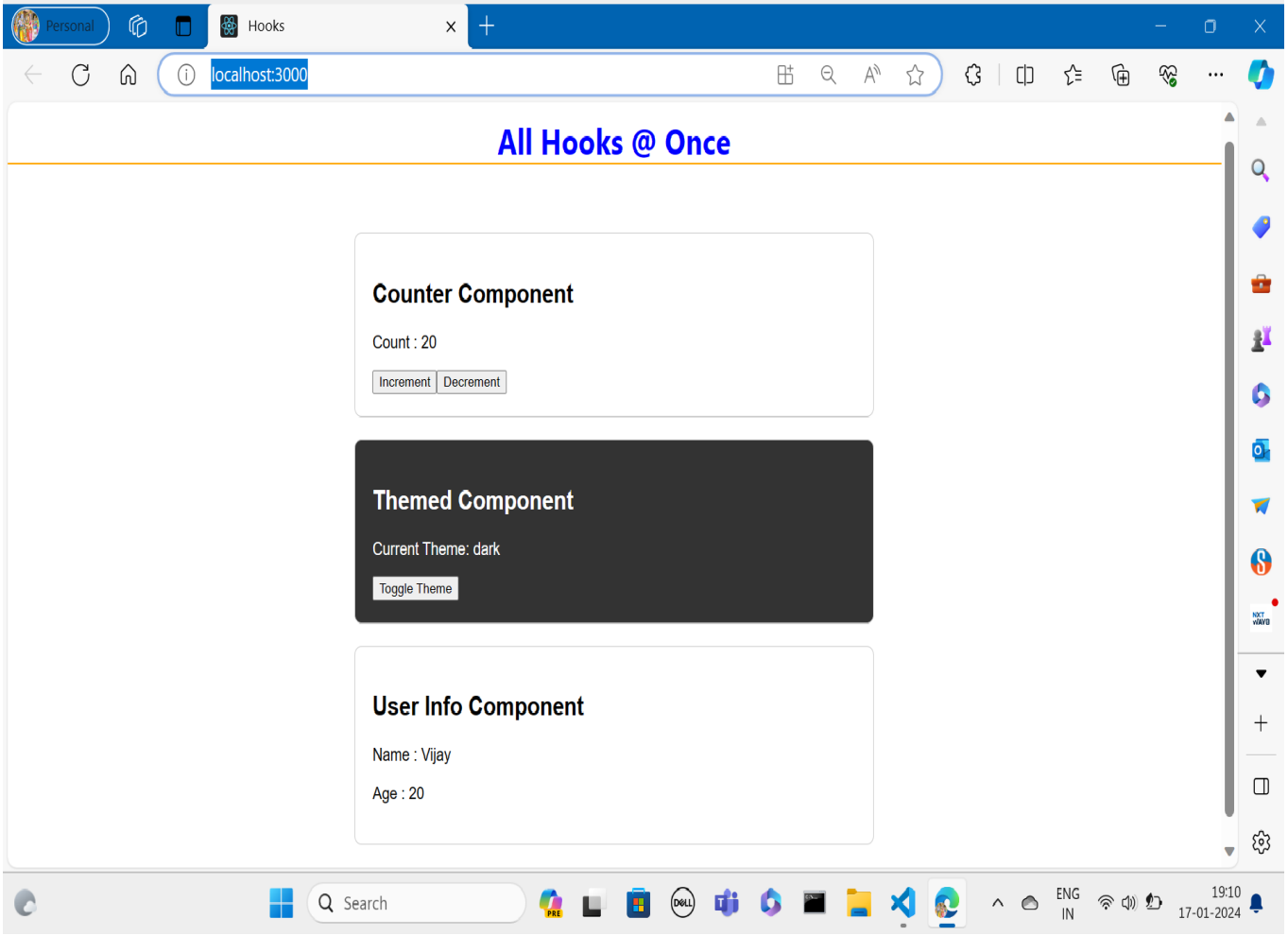
App.js :

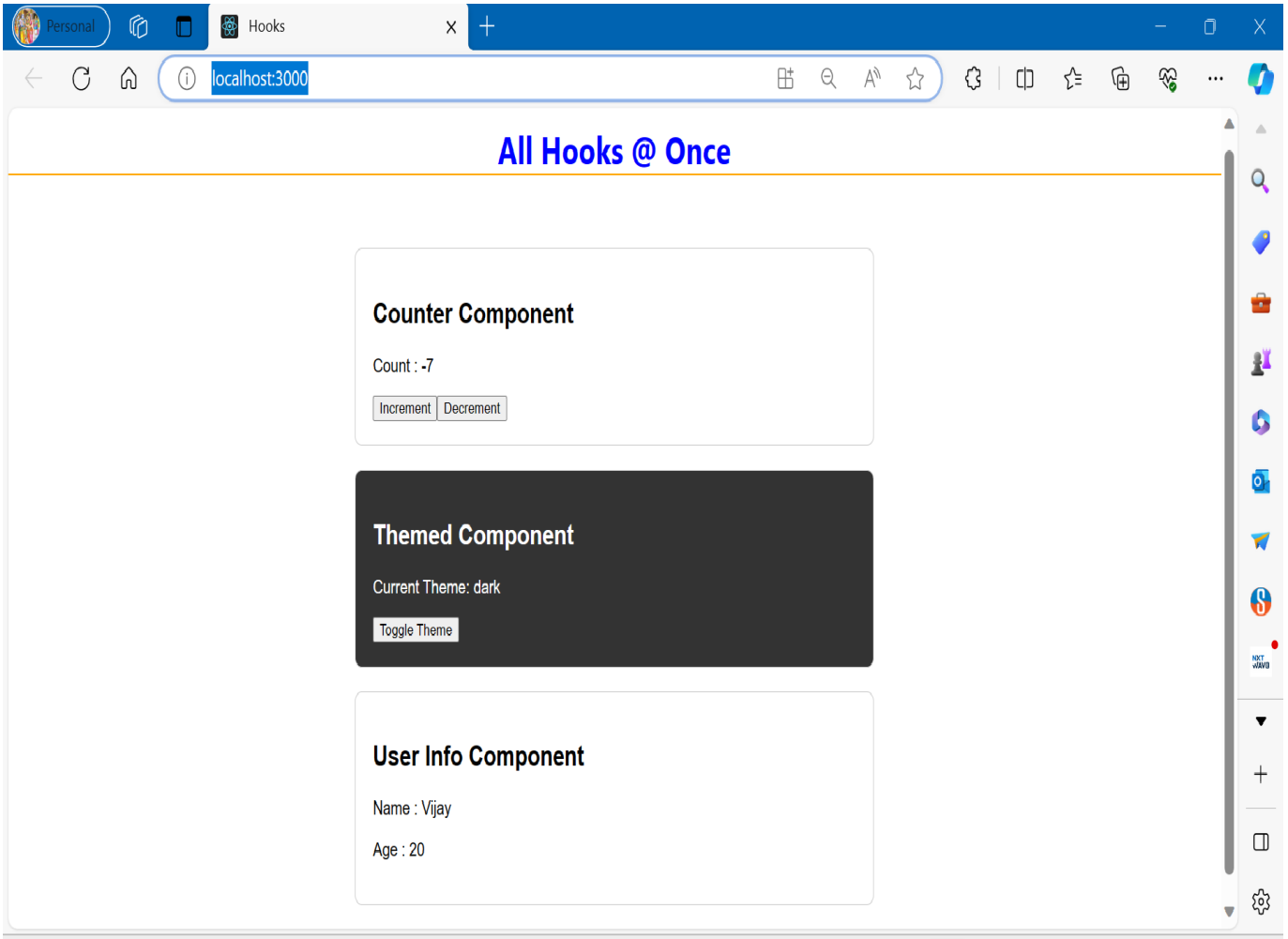
```
import React,{useState,useContext,useEffect,createContext} from 'react';

const ThemeContext=createContext();

const ThemeProvider=({children}) => {
  const[theme,setTheme]=useState('light');

const toggleTheme = () => {
  setTheme((prevTheme) => (prevTheme === 'light' ? 'dark' : 'light'));
};

return (
  <ThemeContext.Provider value={{theme,toggleTheme}}>
    {children}
  </ThemeContext.Provider>
);
};

const ThemedComponent = () => {
  const {theme,toggleTheme} = useContext(ThemeContext);

  const ThemedComponentStyle = {
    background: theme === 'light' ? '#fff' : '#333',
    color: theme === 'light' ? '#333' : '#fff',
    border:'1px solid #ccc',
    borderRadius:'8px',
    padding: '20px',
    margin: '20px 0',
  };

  return (
    <div style={ThemedComponentStyle}>
      <h2>Themed Component</h2>
      <p>Current Theme: {theme}</p>
      <button onClick={toggleTheme}>Toggle Theme</button>
    </div>
  );
};

const CounterComponent = () => {
  const [count,setCount]=useState(0);

  useEffect(() => {
    document.title=`Hooks`
  },[count]);

  const counterStyle = {
    margin:'20px 0',
    padding:'20px',
    border:'1px solid #ccc',
    borderRadius:'8px',
  };

  return (
    <div style={counterStyle}>
```

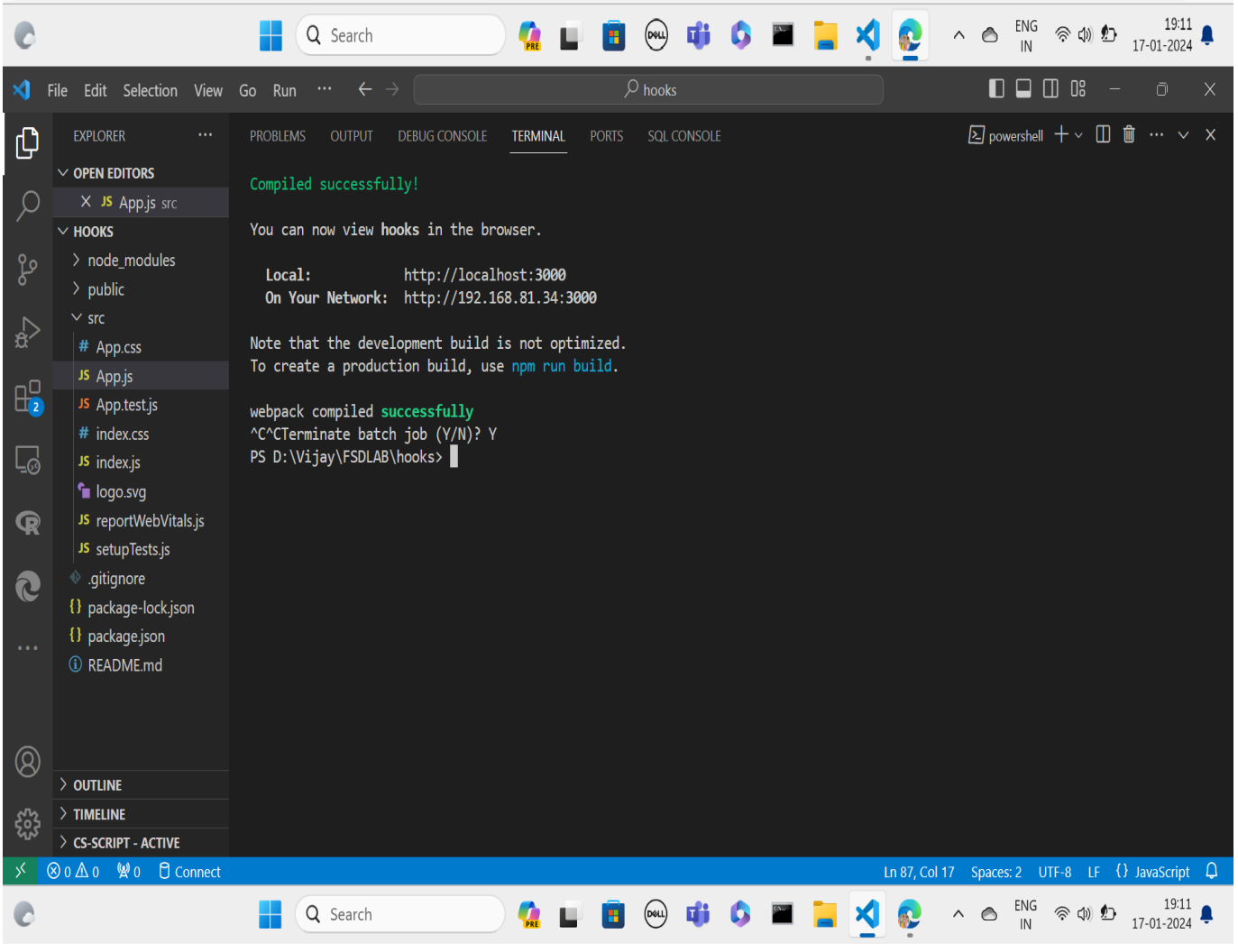
```
      <h2>Counter Component</h2>
      <p>Count : {count}</p>
      <button onClick={() => setCount(count+1)}>Increment</button>
      <button onClick={() => setCount(count-1)}>Decrement</button>

    </div>
  );
};

const UserInfoComponent = () => {
  const [userInfo,setUserInfo] = useState({name:'',age:0});
  useEffect(() => {
    const fetchUserInfo = async () => {
      setTimeout(() => {
        setUserInfo({name:'Vijay',age:20});
      },1000);
    };

    fetchUserInfo();
  },[]);

  const userInfoStyle = {
    margin:'20px 0',
    padding:'20px',
    border: '1px solid #ccc',
    borderRadius:'8px',
  };

  return (
    <div style={userInfoStyle}>
      <h2>User Info Component</h2>
      <p>Name   : {userInfo.name}</p>
      <p>Age    : {userInfo.age}</p>
    </div>
  );
};

const App = () => {
  const appStyle = {
    fontFamily:'Arial,sans-serif',
    maxWidth:'600px',
    margin:'auto',
    padding:'20px',
  };

  return (
    <ThemeProvider>
      <div style={{textAlign:'center',marginTop:'10px'}}>
        <h1 style={{borderBottom:'2px solid orange',color:'blue'}}>All Hooks @ Once</h1>
      </div>
      <div style={appStyle}>
        <CounterComponent />
        <ThemedComponent />
        <UserInfoComponent />
      </div>
    </ThemeProvider>
  );
};
export default App;
```

OUTPUT:


Personal
Hooks
localhost:3000
All Hooks @ Once
Counter Component
Count : 0
Increment
Decrement
Themed Component
Current Theme: light
Toggle Theme
User Info Component
Name : Vijay
Age : 20
Search
ENG
IN
19:10
17-01-2024


Personal
Hooks
localhost:3000
All Hooks @ Once
Counter Component
Count : 20
Increment
Decrement
Themed Component
Current Theme: dark
Toggle Theme
User Info Component
Name : Vijay
Age : 20
Search
ENG
IN
19:10
17-01-2024


Personal
Hooks
localhost:3000
All Hooks @ Once
Counter Component
Count : -7
Increment
Decrement
Themed Component
Current Theme: dark
Toggle Theme
User Info Component
Name : Vijay
Age : 20


File Edit Selection View Go Run
hooks
EXPLORER
OPEN EDITORS
App.js src
HOOKS
node_modules
public
src
App.css
App.js
App.test.js
index.css
index.js
logo.svg
reportWebVitals.js
setupTests.js
.gitignore
package-lock.json
package.json
README.md
OUTLINE
TIMELINE
CS-SCRIPT - ACTIVE
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
SQL CONSOLE
powershell
Compiled successfully!
You can now view hooks in the browser.
Local: http://localhost:3000
On Your Network: http://192.168.81.34:3000
Note that the development build is not optimized.
To create a production build, use npm run build.
webpack compiled successfully
^C^CTerminate batch job (Y/N)? Y
PS D:\Vijay\FSDLAB\hooks>
Connect
Ln 87, Col 17
Spaces: 2
UTF-8
LF
JavaScript

EXPERIMENT NO: 11

AIM:

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

DESCRIPTION:

Using fetch

Import: You can use the fetch function provided by the global window object in browsers to make HTTP requests.

URL: Specify the fake URL of the API endpoint you want to fetch data from.

GET Request: Use the fetch function with the URL to make a GET request to the API.

Promise: fetch returns a Promise that resolves with the Response object representing the API response.

Response Handling: You can handle the response by chaining .then() to the fetch Promise to parse the JSON data or handle errors with .catch().

Async/Await: fetch can also be used with async/await for more readable asynchronous code.

Using axios

Install: First, you need to install axios by running npm install axios or include it via a CDN.

Import: Import the axios library at the beginning of your file.

URL: Specify the fake URL of the API endpoint you want to fetch data from.

GET Request: Use the axios.get() method with the URL to make a GET request to the API.

Promise: axios.get() returns a Promise that resolves with the API response data.

Response Handling: You can handle the response by chaining .then() to the axios Promise to access the data or handle errors with .catch().

Error Handling: axios automatically transforms JSON data and handles errors with HTTP status codes.

Interceptors: axios provides request and response interceptors for custom logic, such as setting headers or handling authentication.

Async/Await: axios can also be used with async/await for more concise and readable asynchronous code.

Cancel Requests: axios supports canceling requests using the CancelToken feature, allowing you to cancel pending requests when necessary.

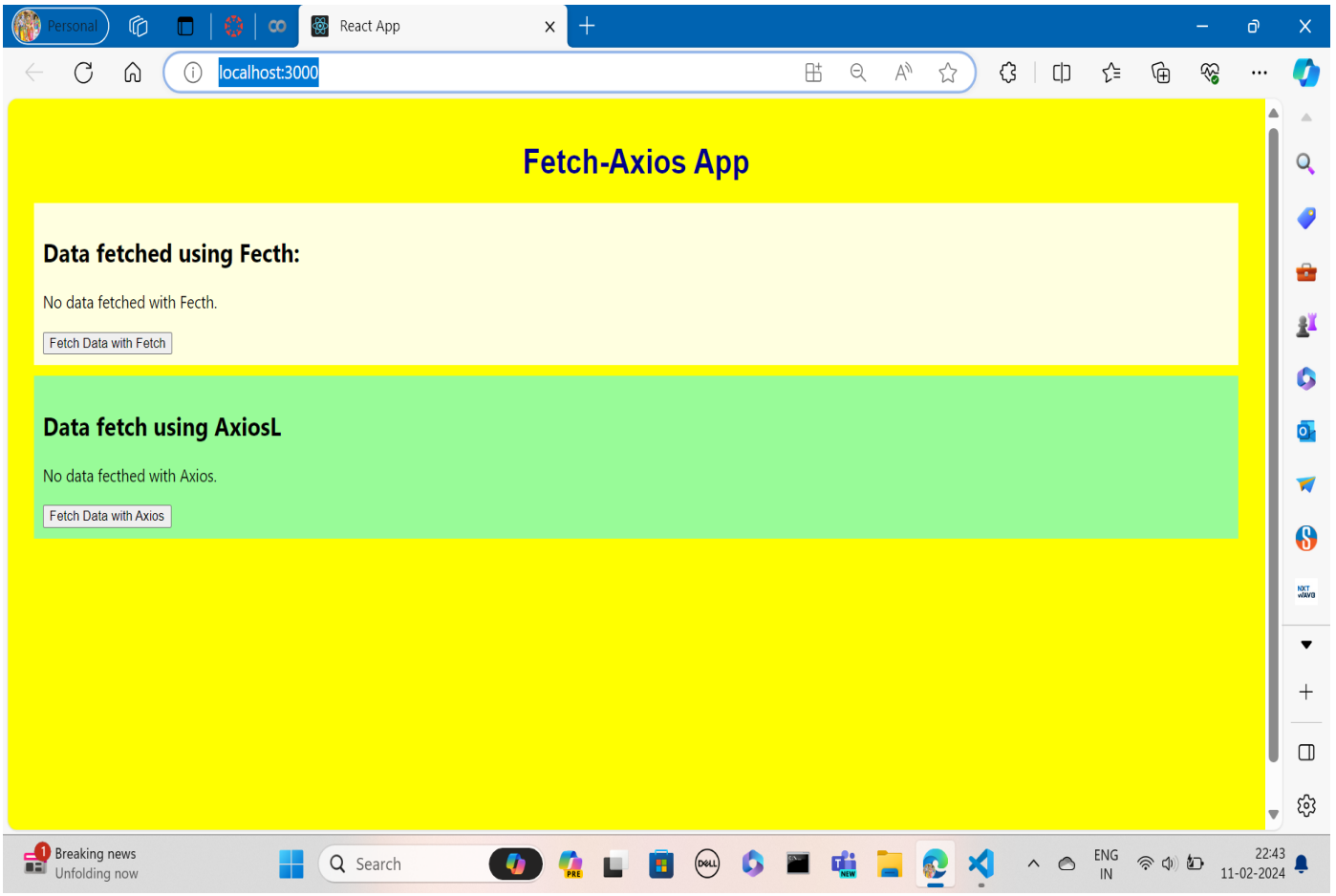
Note: When working with real APIs, always ensure that you handle authentication, error responses, and data parsing properly. Also, consider implementing loading states and error handling in your applications to provide a better user experience.

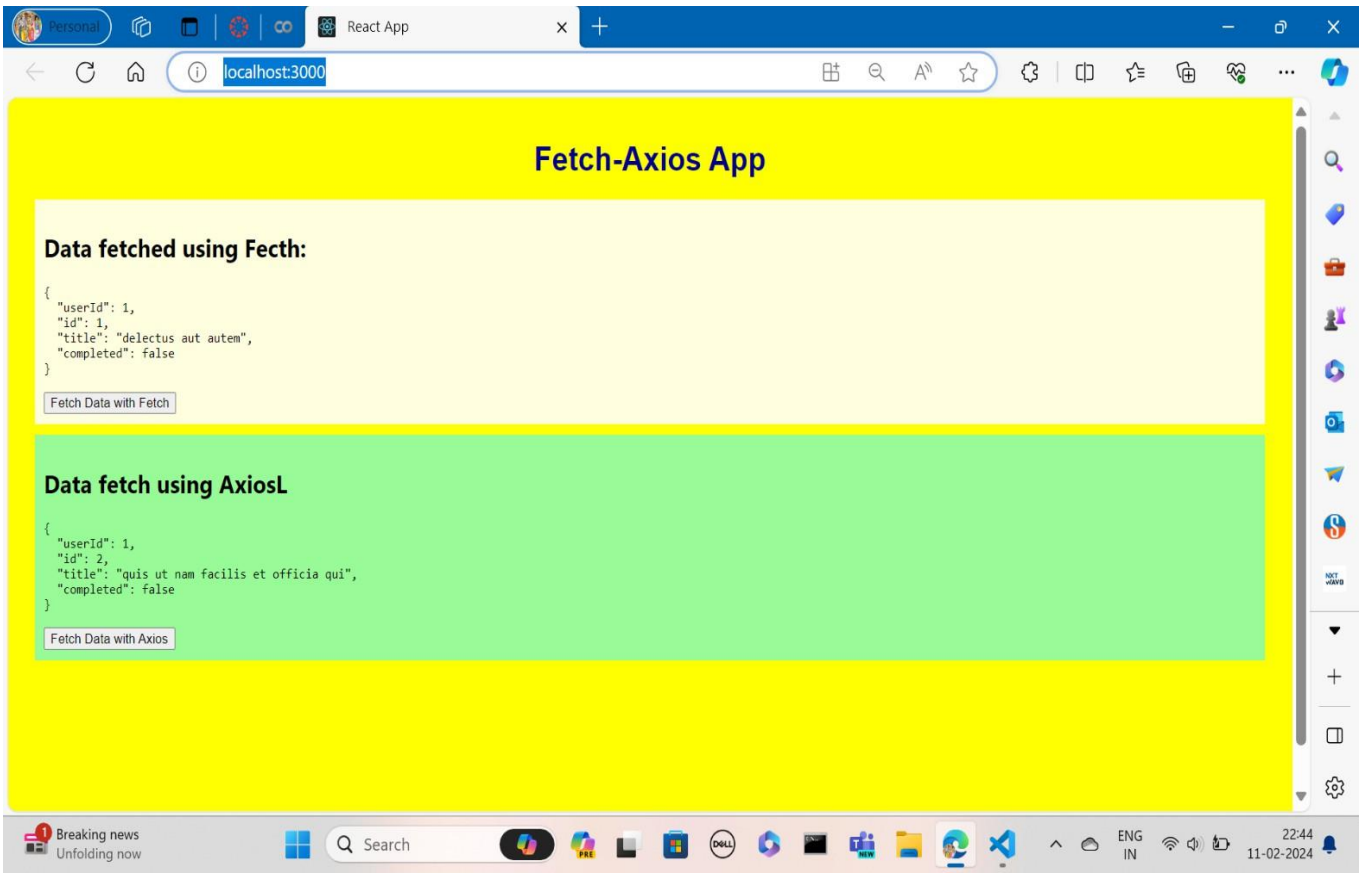
By following these steps, you can consume a web API using both fetch and axios, fetching data from a fake URL in a React application or any JavaScript environment.

App.js :

```
import React,{ useState } from 'react';
import axios from 'axios';
function App(){
const [fetchData , setFetchData] = useState(null);
const [axiosData , setAxiosData] = useState(null);
const fetchDataWithFetch = async()=>{
try{
const response = await fetch('https://jsonplaceholder.typicode.com/todos/1');
const data = await response.json();
setFetchData(data);
}catch(error){
console.error('Error fetching data with Fetch:',error);
}
};
const fetchDatawithAxios = async()=>{
try{
const response = await axios.get('https://jsonplaceholder.typicode.com/todos/2');
setAxiosData(response.data);
} catch(error){
  console.error('Error fecthing data with Axios:',error);
}
};
return(
  <div style={{backgroundColor:'blue',padding:'20px',minHeight:'100vh'}}>
    <h1 style={{marginBottom:'20px',textAlign:'center',fontFamily:'Helvetica,sans-serif',color:'darkblue'}}>Fetch-Axios
App</h1>
    <section style={{backgroundColor : 'lightyellow',padding:'10px',margin:'10px'}}>
      <h2>Data fetched using Fecth:</h2>
      {fetchData ?(
        <pre>{JSON.stringify(fetchData,null,2)}</pre>
      ):(
        <p>No data fetched with Fecth.</p>
      )}
      <button onClick={fetchDataWithFetch}>Fetch Data with Fetch
    </button>
    </section>
    <section style={{backgroundColor:'palegreen',padding:'10px',margin:'10px'}}>
      <h2>Data fetch using AxiosL</h2>
      {axiosData ? (
        <pre>{JSON.stringify(axiosData,null,2)}</pre>
      ):(
        <p>No data fecthed with Axios.</p>
      )}
      <button onClick={fetchDatawithAxios}>Fetch Data with Axios</button>
    </section>
  </div>
)
};
export default App;
```

OUTPUT:


React App
localhost:3000
Fetch-Axios App
Data fetched using Fecth:
No data fetched with Fecth.
Fetch Data with Fetch
Data fetch using AxiosL
No data fecthed with Axios.
Fetch Data with Axios
Breaking news
Unfolding now
Search
ENG
IN
22:43
11-02-2024


React App
localhost:3000
Fetch-Axios App
Data fetched using Fecth:
{
"userId": 1,
"id": 1,
"title": "delectus aut autem",
"completed": false
}
Fetch Data with Fetch
Data fetch using AxiosL
{
"userId": 1,
"id": 2,
"title": "quis ut nam facilis et officia qui",
"completed": false
}
Fetch Data with Axios
Breaking news
Unfolding now
Search
ENG
IN
22:44
11-02-2024

EXPERIMENT NO: 12

AIM:

Design a BMI calculator using React JS based on the description given below.

DESCRIPTION:

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.

Category	BMI range - kg/m^2
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

Steps to Design BMI Calculator in ReactJS:

Initialize Project: Create a new React project using Create React App or any other method**.**

Component Structure: Plan the component structure for the BMI calculator, including input fields for height and weight, a button to calculate BMI, and an area to display the result.

State Management: Use React state to manage the height, weight, and BMI values within the component.

Input Fields: Create controlled input fields for height and weight where users can enter their values.

Calculate BMI: Implement a function to calculate BMI using the formula: BMI = weight / (height * height). Convert height to meters if needed.

Button & Event Handling: Add a button that triggers the BMI calculation when clicked. Implement an event handler to perform the calculation and update the state with the result.

Display Result: Create a section to display the calculated BMI result to the user.

Styling: Apply CSS or a styling library to design the BMI calculator interface, making it user-friendly and visually appealing.

Validation: Implement validation checks to ensure that users enter valid height and weight values before calculating BMI. Display error messages for invalid inputs.

Additional Features: Enhance the calculator by adding features like unit conversion (imperial to metric), BMI categories (underweight, normal weight, overweight, etc.), and interpretation of BMI results.

Example Components:

App Component: Main component that holds the state and logic for BMI calculation.

Input Component: A reusable component for text inputs, used for height and weight**.**

Button Component: A reusable button component to trigger the BMI calculation.

Result Component: A component to display the calculated BMI result and interpretation.

By following these steps and principles, you can design a BMI calculator in ReactJS that provides a simple and effective way for users to calculate and understand their BMI based on their height and weight inputs.

App.js :

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

function App() {
  const [height, setHeight] = useState('');
  const [weight, setWeight] = useState('');
  const [bmi, setBMI] = useState(null);

  const calculateBMI = () => {
    if (height && weight) {
      const heightInMeters = height / 100;
      const bmiValue = (weight / (heightInMeters * heightInMeters)).toFixed(2);
      setBMI(bmiValue);
    }
  };

  const getBMICategory = () => {
    if (bmi < 16) return 'Severe Thinness';
    if (bmi >= 16 && bmi < 17) return 'Moderate Thinness';
    if (bmi >= 17 && bmi < 18.5) return 'Mild Thinness';
    if (bmi >= 18.5 && bmi < 25) return 'Normal';
    if (bmi >= 25 && bmi < 30) return 'Overweight';
    if (bmi >= 30 && bmi < 35) return 'Obese Class I';
    if (bmi >= 35 && bmi < 40) return 'Obese Class II';
    if (bmi >= 40) return 'Obese Class III';
    return '';
  };

  const getMeterColor = () => {
    if (bmi < 16) return '#3498db'; // Blue for severe thinness
    if (bmi < 17) return '#9b59b6'; // Purple for moderate thinness
    if (bmi < 18.5) return '#2ecc71'; // Green for mild thinness
    if (bmi < 25) return '#f1c40f'; // Yellow for normal weight
```

```
  if (bmi < 30) return '#e67e22'; // Orange for overweight
  if (bmi < 35) return '#e74c3c'; // Red for obese class I
  if (bmi < 40) return '#c0392b'; // Dark red for obese class II
  return '#96281b';
 };

 return (
  <div className="App">
   <h1 style={{ borderBottom: '2px solid #FFA500', paddingBottom: '10px', marginBottom: '20px', color: 'green' }}>
    BMI Calculator
   </h1>
   <div className="container">
    <div className="form-container">
     <label>
      Height (cm):
      <input type="number" value={height} onChange={(e) => setHeight(e.target.value)} />
     </label>
     <label>
      weight (kg):
      <input type="number" value={weight} onChange={(e) => setWeight(e.target.value)} />
     </label>
     <button
      onClick={calculateBMI}
      onMouseOver={(e) => (e.target.style.background = 'orange')}
      onMouseOut={(e) => (e.target.style.background = 'darkblue')}>
      check BMI
     </button>
     {bmi !== null && (
      <div className="result-container">
       <div className="meter-container">
        <div className="meter" style={{ width: `${bmi * 3}px`, backgroundColor: getMeterColor() }}></div>
       </div>
       <p>
        <strong>Your BMI is : {bmi}</strong>
       </p>
       <p>
        <strong>Category: {getBMICategory()}</strong>
       </p>
      </div>
     )}
    </div>
    <div className="table-container">
     <h4>BMI Categories</h4>
     <table className="category-table">
      <thead>
       <tr>
        <th>Category</th>
        <th>BMI Range (kg/m^2)</th>
       </tr>
      </thead>
      <tbody>
       <tr>
        <td>Severe Thinness</td>
        <td>&lt;16</td>
       </tr>
       <tr>
        <td>Moderate Thinness</td>
        <td>16-17</td>
       </tr>
```

```
            <tr>
              <td>Mild Thinness</td>
              <td>17-18.5</td>
            </tr>
            <tr>
              <td>Normal</td>
              <td>18.5-25</td>
            </tr>
            <tr>
              <td>Overweight</td>
              <td>25-30</td>
            </tr>
            <tr>
              <td>Obese Class I</td>
              <td>30-35</td>
            </tr>
            <tr>
              <td>Obese class II</td>
              <td>35-40</td>
            </tr>
            <tr>
              <td>Obese Class III</td>
              <td>&gt; 40</td>
            </tr>
          </tbody>
        </table>
      </div>
    </div>
    <div className="watermark">
      <p>G VIJAY KUMAR</p>
    </div>
  </div>
 );
}

export default App;
```

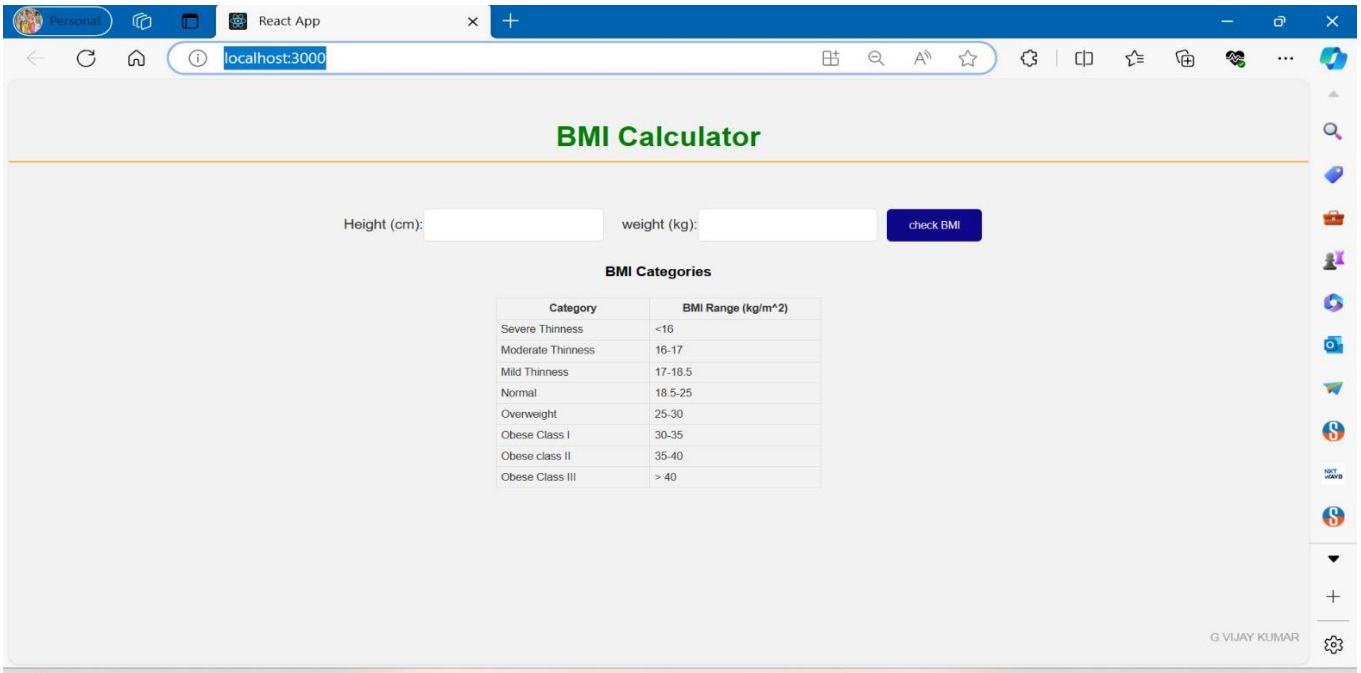
App.css :

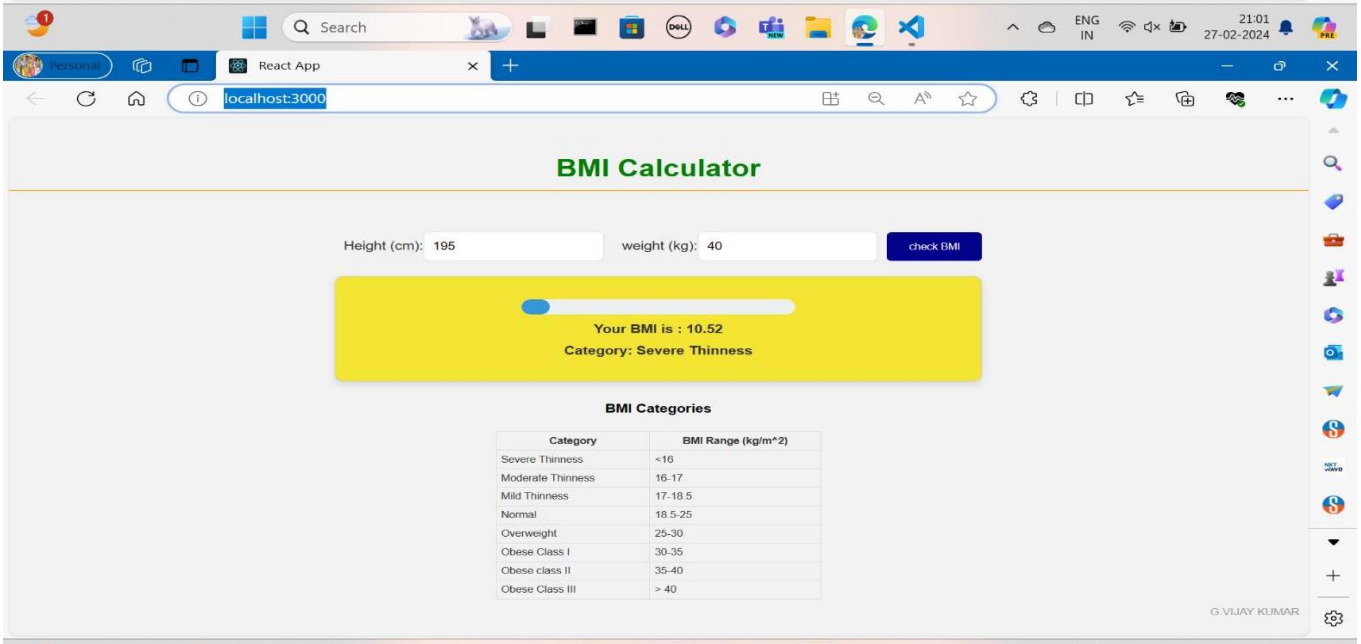
```
.App {
  text-align: center;
  margin-top: 50px;
}
body {
  margin: 0;
  padding: 0;
  font-family: "Arial",sans-serif;
  background-color: #f2f2f2;
}
h1 {
  color: #333;
}
.category-container{
  margin-bottom: 20px;
}
.category-table {
  border-collapse: collapse;
  width: 50%;
  margin: 0 auto;
```

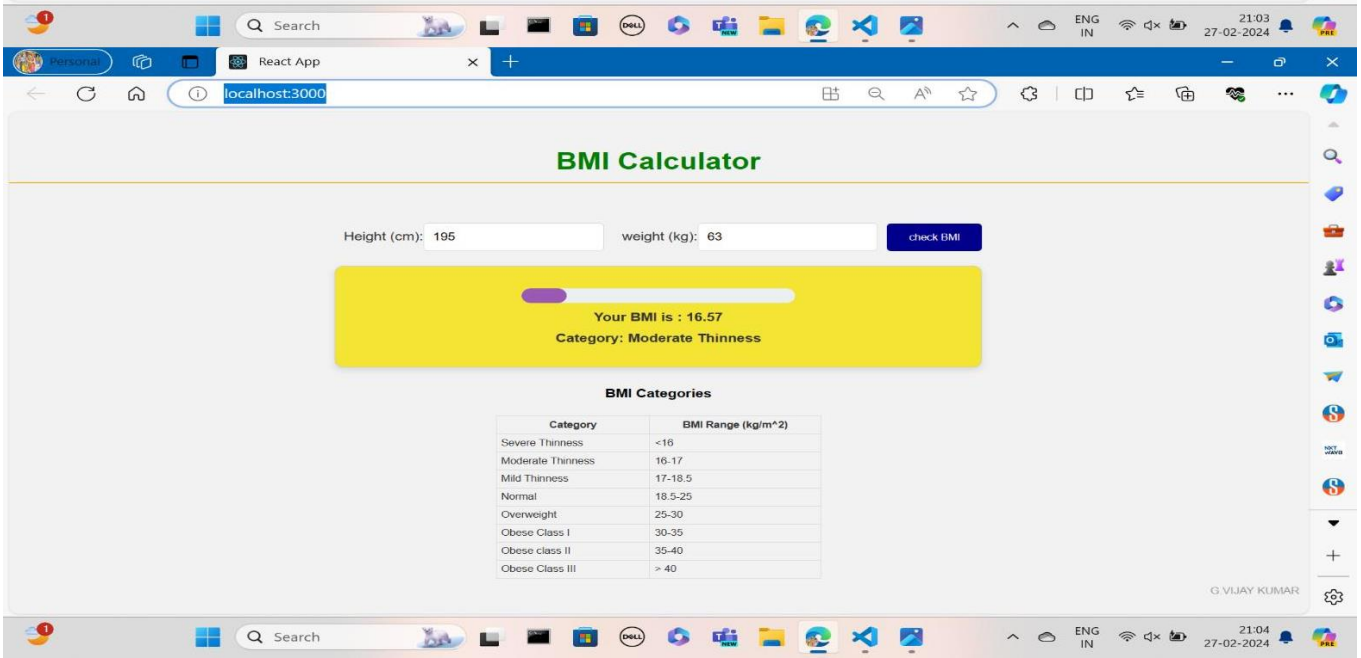
```
}
.category-table th {
  border: 1px solid #ddd;
  padding: 5px;
  text-align: center;
  font-size: 0.75em;
  color: #333;
}
.category-table td {
  border: 1px solid #ddd;
  padding: 5px;
  text-align: left;
  font-size: 0.75em;
  color: #333;
}
.container {
  display: flex;
  flex-direction: column;
  align-items: center;
}
.form-container {
  margin-top: 20px;
}
.table-container {
  width: 50%;
  margin-top: 5px;
}
label {
  margin: 10px;
  font-size: 1em;
  color: #333;
}
input {
  padding: 10px;
  font-size: 0.9em;
  border: 1px solid #ddd;
  border-radius: 5px;
}
button {
  margin-top: 15px;
  padding: 12px 24px;
  font-size: 0.75em;
  cursor: pointer;
  background-color: #0e0789;
  color: #fff;
  border: none;
  border-radius: 5px;
}
.result-container {
  margin-top: 20px;
  padding: 20px;
  border: 1px solid #ccc;
  border-radius: 10px;
  background-color: #f4e433;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}
.meter-container {
  width: 300px;
  height: 20px;
```

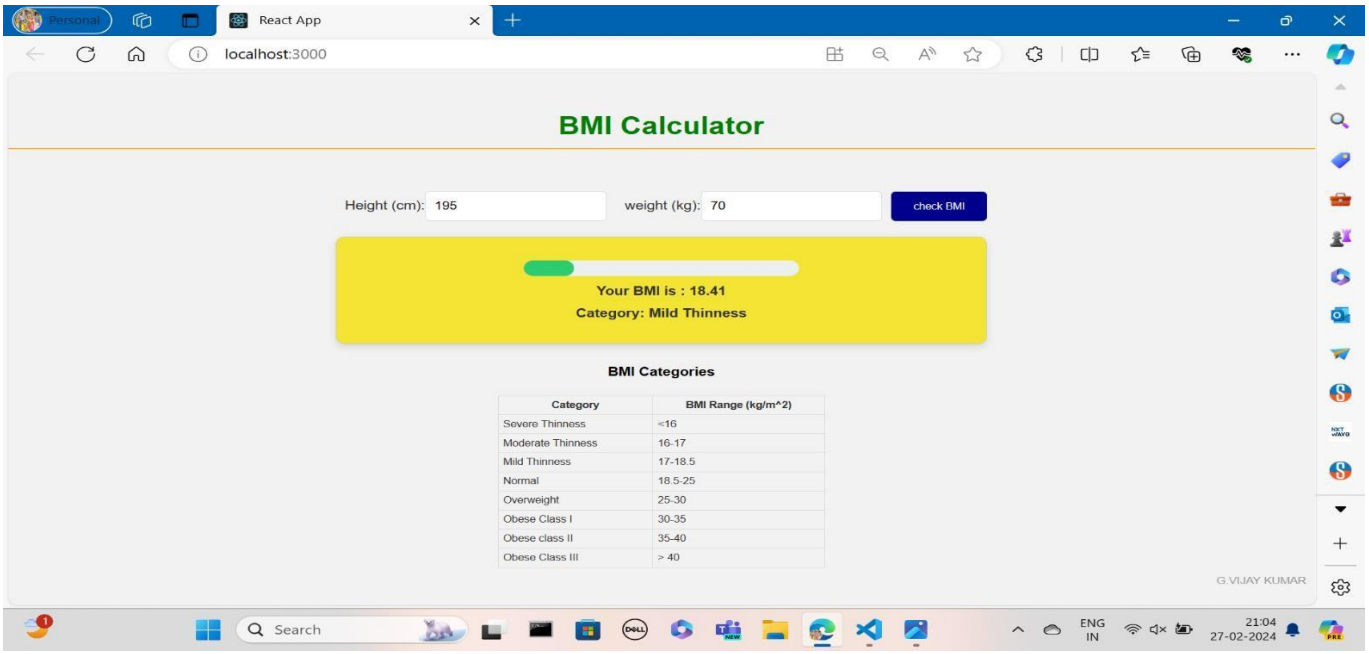
```
  background-color: #ecf0f1;

  border-radius: 10px;
  margin: 10px auto;
  overflow: hidden;
}
.meter {
  height: 100%;
  border-radius: 10px;
  transition: width 0.5s ease;
}
.result-container p {
  margin: 10px 0;
  font-size: 1em;
  color: #333;
}
.watermark {
  position: fixed;
  bottom: 10px;
  right: 10px;
  color: rgba(0, 0, 0, 0.4);
  font-size: 0.8em;
}
```

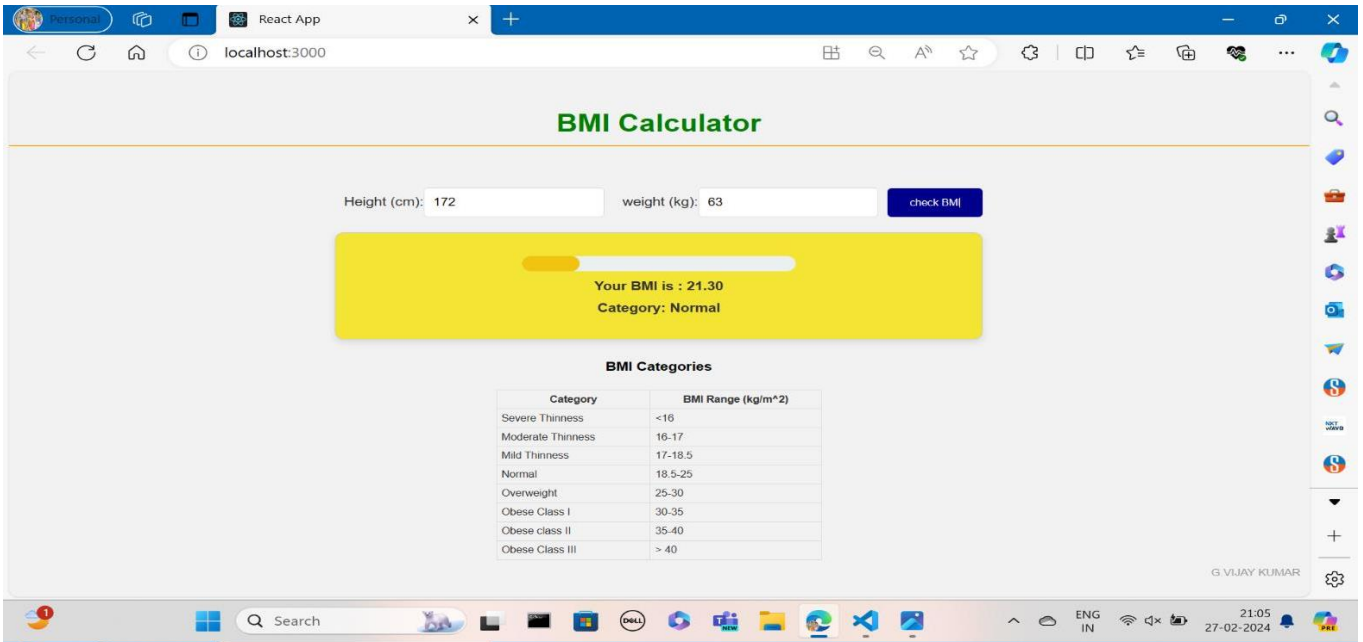
OUTPUT:

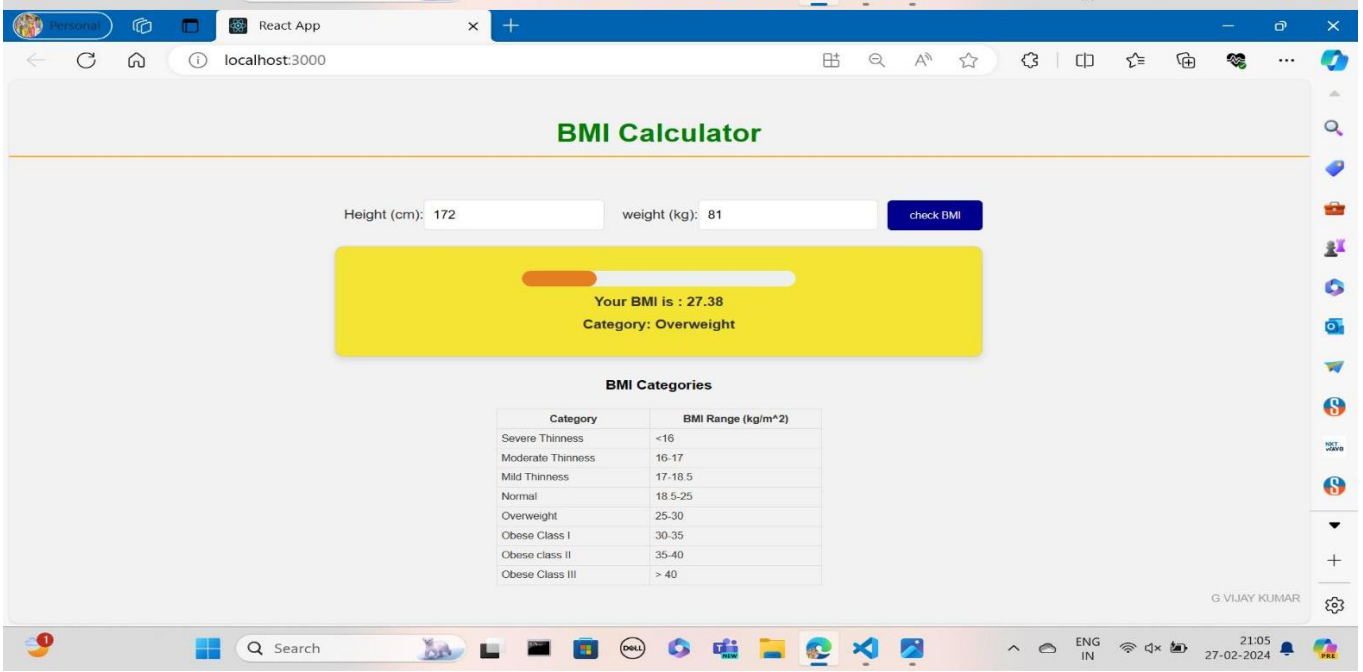

React App
localhost:3000
BMI Calculator
Height (cm):
weight (kg):
check BMI
BMI Categories
Category
BMI Range (kg/m^2)
Severe Thinness
<16
Moderate Thinness
16-17
Mild Thinness
17-18.5
Normal
18.5-25
Overweight
25-30
Obese Class I
30-35
Obese class II
35-40
Obese Class III
> 40
G VIJAY KUMAR

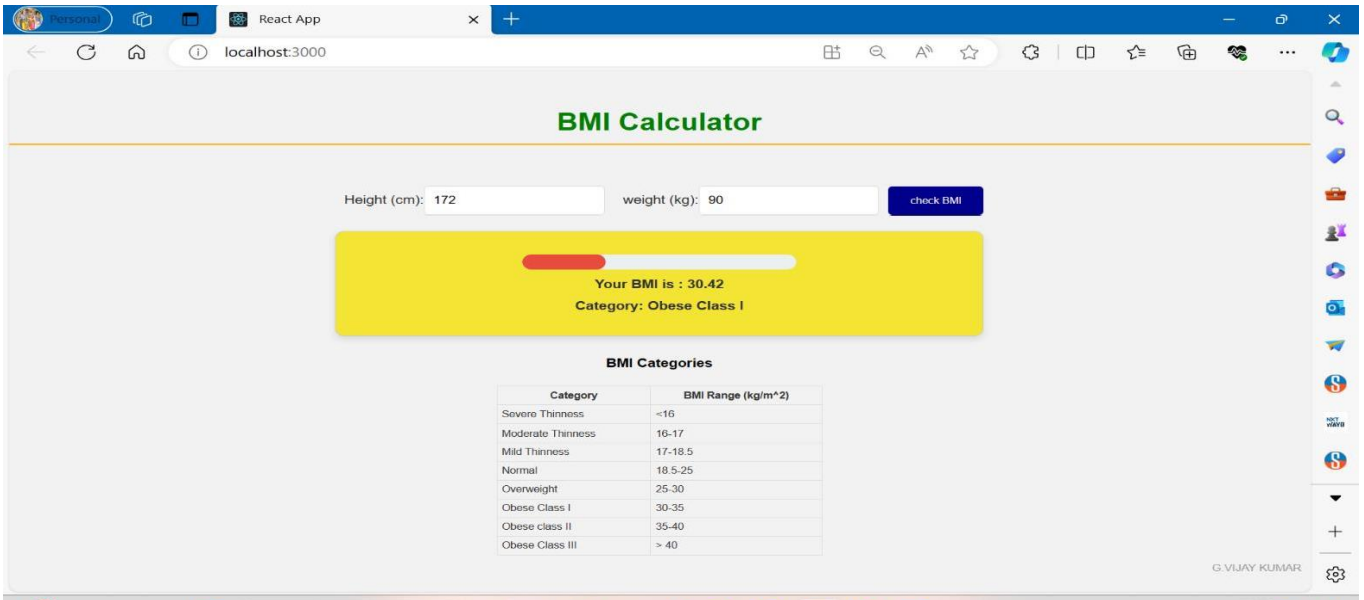

React App
localhost:3000
BMI Calculator
Height (cm): 195
weight (kg): 40
check BMI
Your BMI is : 10.52
Category: Severe Thinness
BMI Categories
Category
BMI Range (kg/m^2)
Severe Thinness
<16
Moderate Thinness
16-17
Mild Thinness
17-18.5
Normal
18.5-25
Overweight
25-30
Obese Class I
30-35
Obese class II
35-40
Obese Class III
> 40
G VIJAY KUMAR

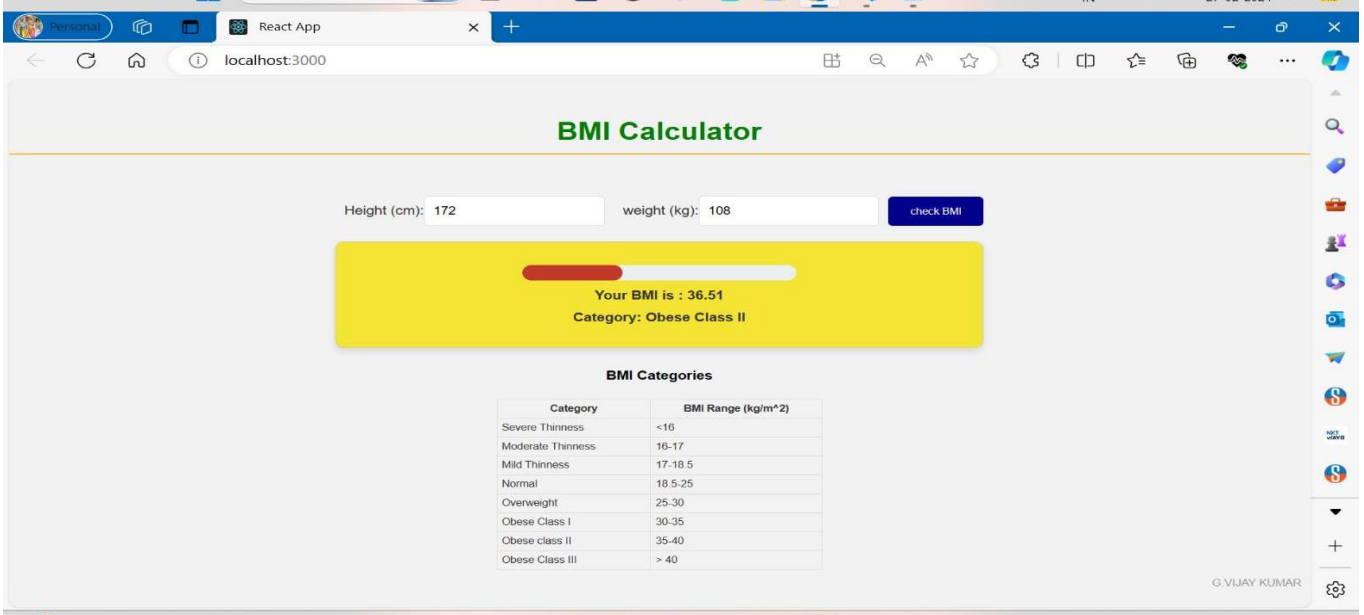

React App
localhost:3000
BMI Calculator
Height (cm): 195
weight (kg): 63
check BMI
Your BMI is : 16.57
Category: Moderate Thinness
BMI Categories
Category
BMI Range (kg/m^2)
Severe Thinness
<16
Moderate Thinness
16-17
Mild Thinness
17-18.5
Normal
18.5-25
Overweight
25-30
Obese Class I
30-35
Obese class II
35-40
Obese Class III
> 40
G VIJAY KUMAR

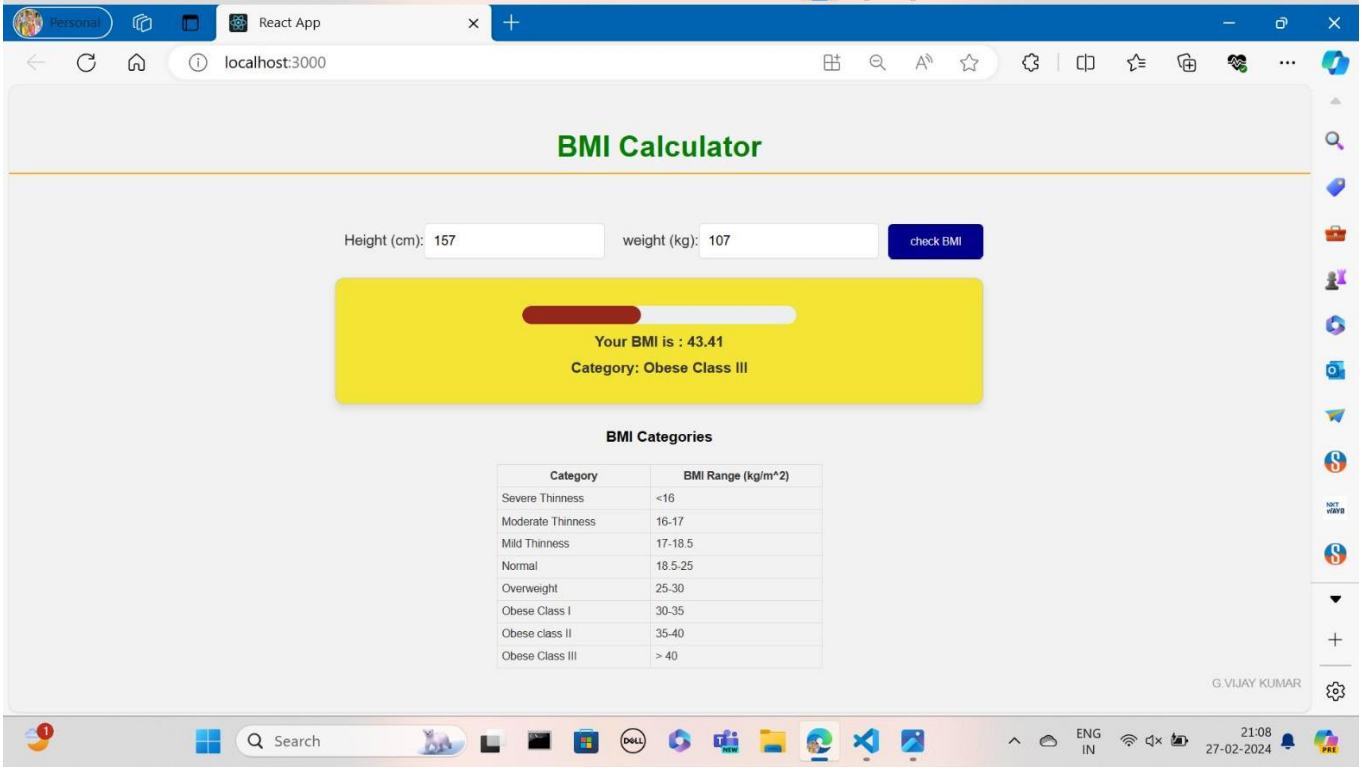

React App
localhost:3000
BMI Calculator
Height (cm): 195
weight (kg): 70
check BMI
Your BMI is : 18.41
Category: Mild Thinness
BMI Categories
Category | BMI Range (kg/m^2)
Severe Thinness | <16
Moderate Thinness | 16-17
Mild Thinness | 17-18.5
Normal | 18.5-25
Overweight | 25-30
Obese Class I | 30-35
Obese class II | 35-40
Obese Class III | > 40
G.VIJAY KUMAR
Search
ENG IN
21:04
27-02-2024


React App
localhost:3000
BMI Calculator
Height (cm): 172
weight (kg): 63
check BMI
Your BMI is : 21.30
Category: Normal
BMI Categories
Category | BMI Range (kg/m^2)
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
G.VIJAY KUMAR
Search
ENG IN
21:05
27-02-2024


React App
localhost:3000
BMI Calculator
Height (cm): 172
weight (kg): 81
check BMI
Your BMI is : 27.38
Category: Overweight
BMI Categories
Category | BMI Range (kg/m^2)
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
G.VIJAY KUMAR
Search
ENG IN
21:05
27-02-2024


React App
localhost:3000
BMI Calculator
Height (cm): 172
weight (kg): 90
check BMI
Your BMI is : 30.42
Category: Obese Class I
BMI Categories
Category | BMI Range (kg/m^2)
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
G.VIJAY KUMAR
Search
ENG IN
21:05
27-02-2024


React App
localhost:3000
BMI Calculator
Height (cm): 172
weight (kg): 108
check BMI
Your BMI is : 36.51
Category: Obese Class II
BMI Categories
Category | BMI Range (kg/m^2)
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
G.VIJAY KUMAR
Search
ENG IN
21:06
27-02-2024


React App
localhost:3000
BMI Calculator
Height (cm): 157
weight (kg): 107
check BMI
Your BMI is : 43.41
Category: Obese Class III
BMI Categories
Category | BMI Range (kg/m^2)
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
G.VIJAY KUMAR
Search
ENG IN
21:08
27-02-2024

EXPERIMENT NO: 13

AIM:

Display a selected set of images in tabular format using React JS.

DESCRIPTION:

Steps to Display Images in Tabular Format in ReactJS :

Initialize Project: Create a new React project using Create React App or any other method.

Component Structure: Plan the component structure for displaying images in a table format. You'll need a main component to hold the state and logic, and child components to render the table, table rows, and table data.

State Management: Use React state to manage the selected set of images. Initialize an empty array in the state to hold the image data.

Fetch Images: Implement a function to fetch the selected set of images from an API or local data source. Update the state with the fetched image data.

Table Rendering: Create a table structure using HTML <table>, <thead>, <tbody>, <tr>, and <td> elements to display the images in a tabular format.

Image Rendering: Inside the table data cells (<td>), render the images using <img> elements. You can set the src attribute of the <img> element to the image URL from the state.

Styling: Apply CSS or a styling library to design the table, images, and overall layout. Ensure that the images are displayed uniformly within the table cells.

Error Handling: Implement error handling to manage cases where the image data fetching fails or the data is invalid. Display error messages or fallback content when necessary.

Loading State: Add a loading state to indicate that the images are being fetched. Display a loader or spinner while the images are being loaded.

Pagination: If dealing with a large set of images, consider implementing pagination to display a limited number of images per page, enhancing the user experience and performance.

Example Components:

App Component: Main component that holds the state, fetches the images, and renders the table component**.**

Table Component: A reusable component to render the table structure and iterate over the image data to render table rows and cells**.**

Image Component: A reusable component to render individual images inside table cells.

By following these steps and principles, you can create a ReactJS application that displays a selected set of images in a tabular format, providing a structured and organized way to view and interact with the images.

App.js :

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

const ImageTable = ({ selectedImages }) => {
 return (
  <div className="table-container">
   <table className="styled-table">
   <thead>
   <tr>
 <th style={{ border: '1px solid #3824ee', padding: '8px', textAlign: 'center', backgroundColor: 'lightblue' }}>Image</th>
 <th style={{ border: '1px solid #3824ee', padding: '8px', textAlign: 'center', backgroundColor: 'lightblue' }}>Name</th>
</tr>

   </thead>
   <tbody>
    {selectedImages.map((image, index) => (
     <tr key={index}>
      <td>
       <img src={image.src} alt={image.alt} style={{ maxWidth: '100px', maxHeight: '100px' }} />
      </td>
      <td>{image.alt}</td>
     </tr>
    ))}
   </tbody>
   </table>
  </div>
 );
};

const App = () => {
 const [selectedImages, setSelectedImages] = useState([

 ]);

 const handleFileUpload = (event) => {
  const files = event.target.files;

  for (let i = 0; i < files.length; i++) {
   const reader = new FileReader();

   reader.onload = (e) => {
    const imageName = files[i].name; // Get the image name
    const newImage = {
     src: e.target.result,
     alt: imageName, // Use the image name as alt text
    };

    setSelectedImages([...selectedImages, newImage]);
   };

   reader.readAsDataURL(files[i]);
  }
 };

 return (
  <div>
   <div>
```

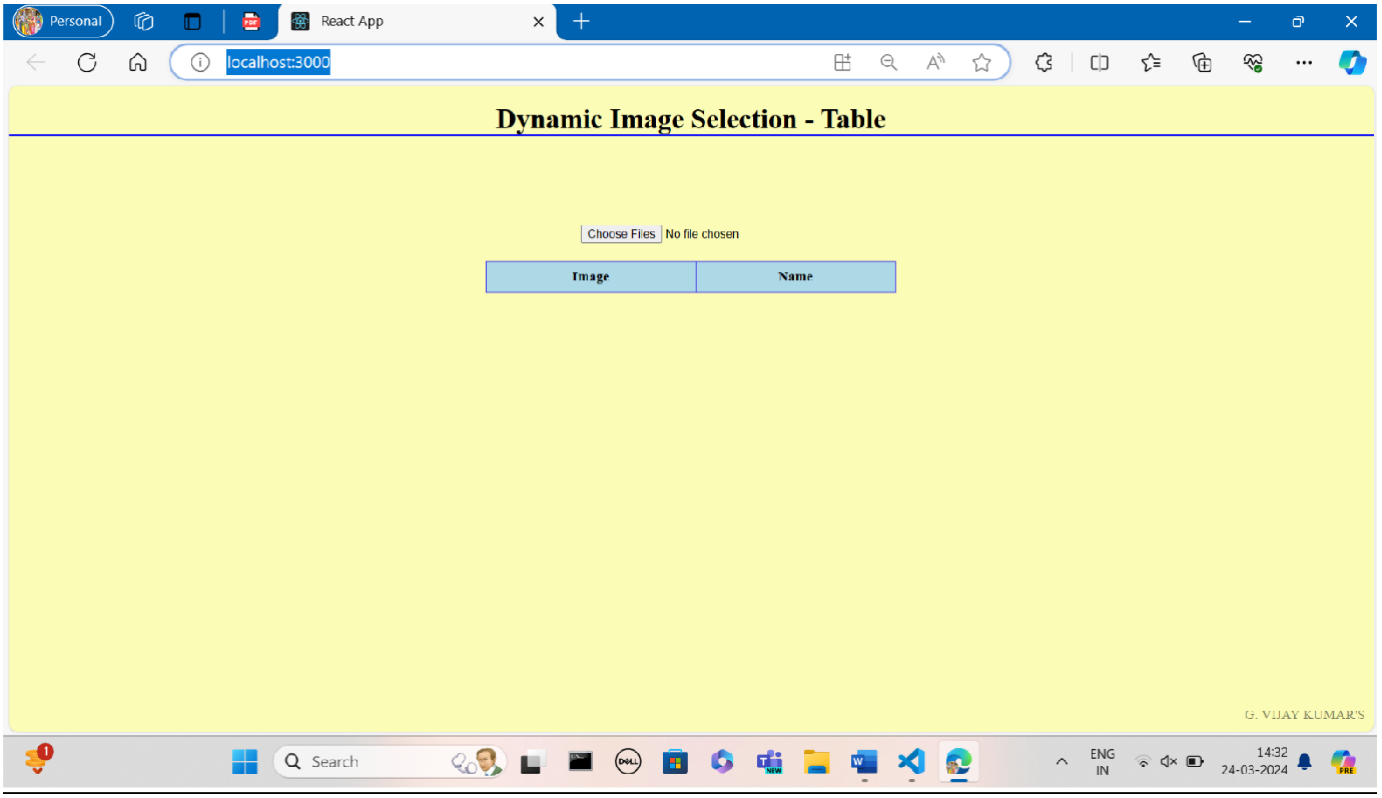
```
      <h1 style={{ borderBottom: '2px solid blue', paddingTop: '20px', marginBottom: '100px' }}>
        Dynamic Image Selection - Table
      </h1>
      </div>
      <div className="upload-container">
        <input type="file" onChange={handleFileUpload} multiple />
        <ImageTable selectedImages={selectedImages} />
      </div>
      <div className="watermark">G. VIJAY KUMAR'S</div>
    </div>
  );
};

export default App;
```

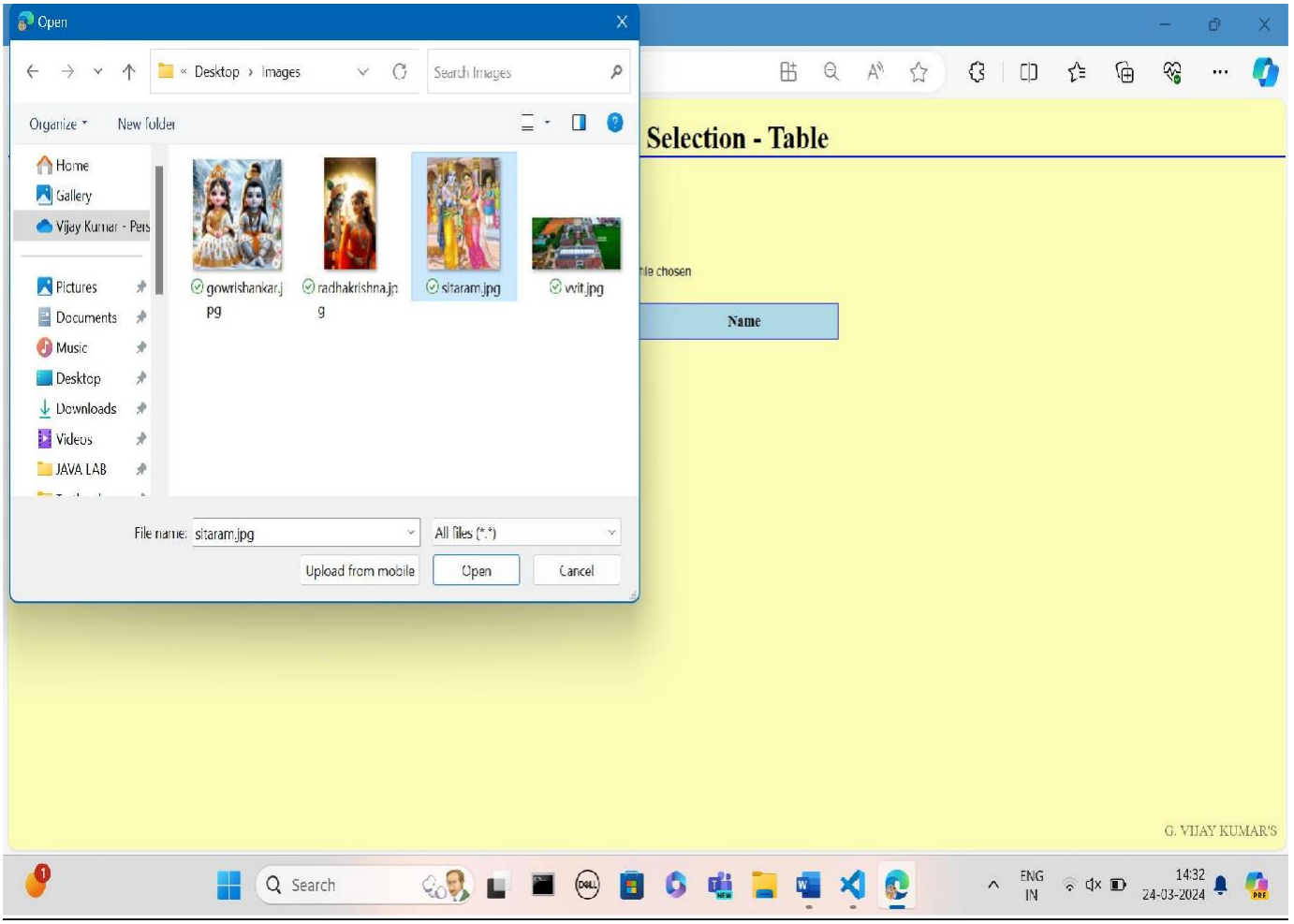
App.css :

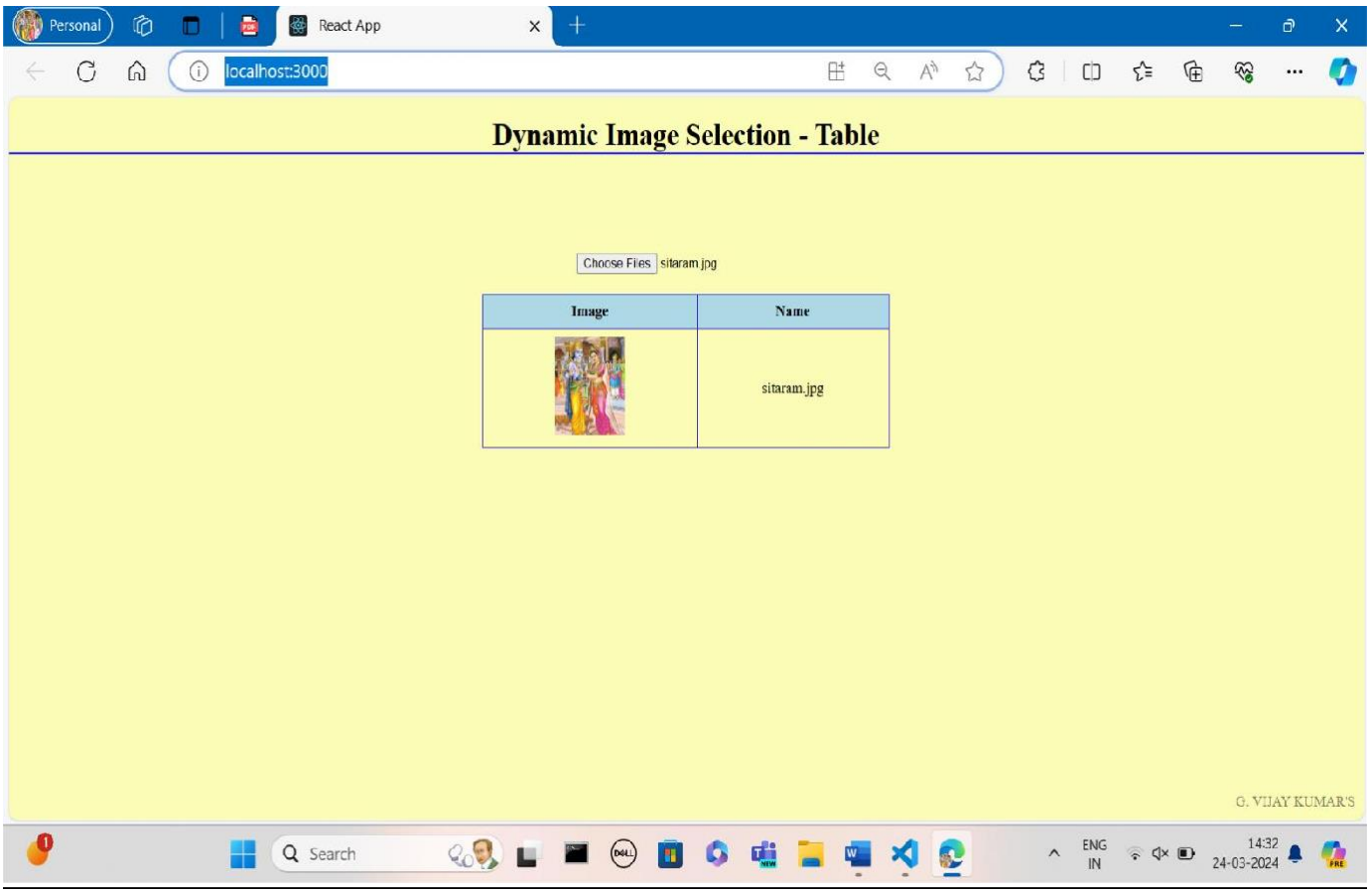
```
body {
  margin: 0;
  background-color: #fbfcb6;
}

code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
    monospace;
}

h1 {
  text-align: center;
  margin-top: 10px;
  width: 100%;
  margin: 0;
}

.app-container {
  text-align: center;
}

.table-container {
  margin-top: 20px;
  max-height: 400px; /* Set a maximum height for the table container */
  overflow-y: auto; /* Add vertical scrollbars if the content overflows */
}

.upload-container {
  margin-top: 10px;
  text-align: center; /* Center the content horizontally */
}

.upload-container input {
  display: inline-block; /* Ensure the input is displayed as a block */
}
.styled-table {
  width: 30%;
  border-collapse: collapse;
  margin: 0 auto;
}

.styled-table th {
```

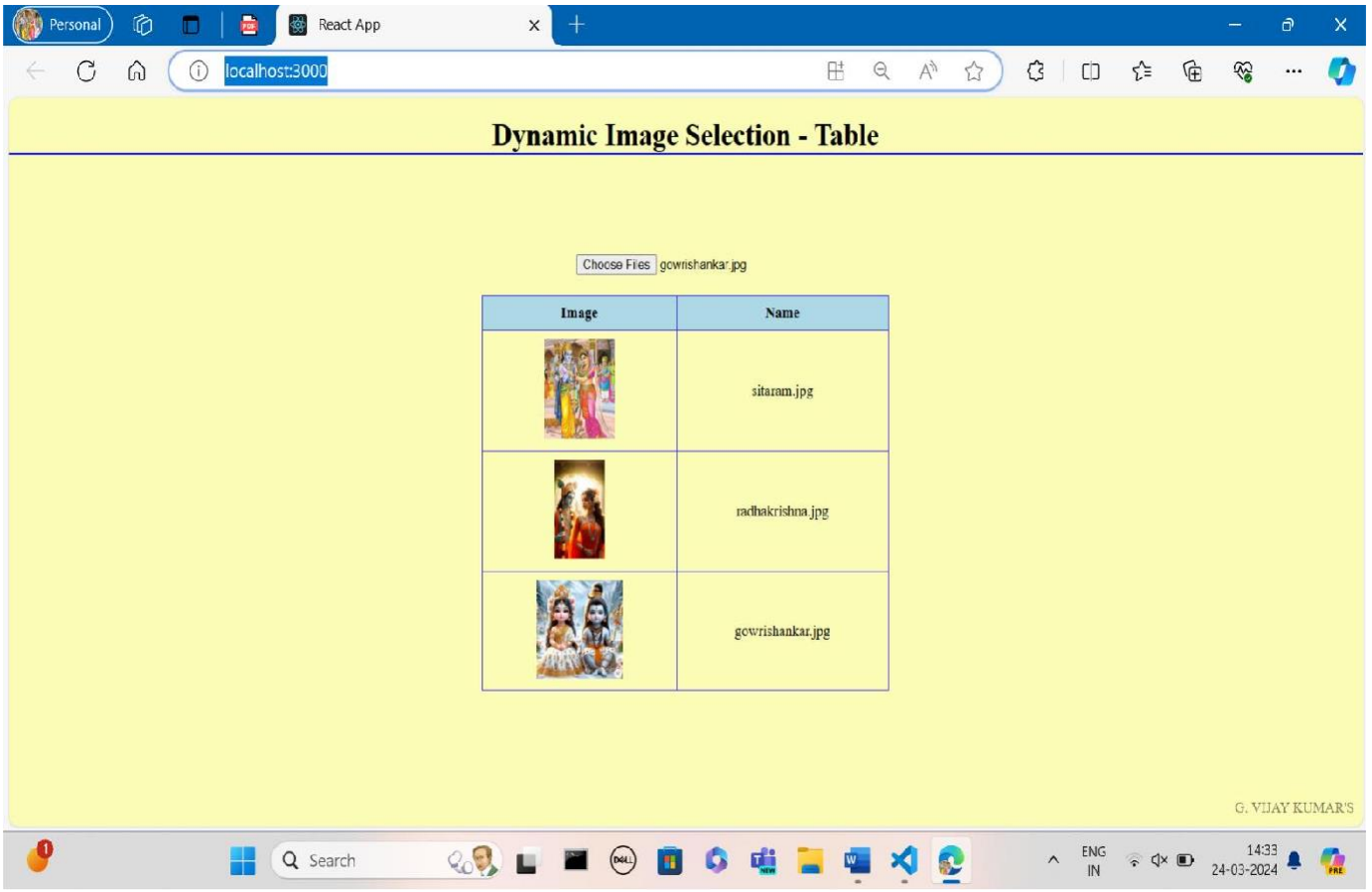
```
  border: 1px solid #3824ee;
  padding: 8px;
  text-align: center; /* Center the text in table cells */
}

.styled-table td {
  border: 1px solid #3824ee;
  padding: 8px;
  text-align: center;
}

.image-cell img {
  width: 100%;
  height: 100%;
  max-width: 150px;
  max-height: 150px;
  object-fit: cover;
}

.watermark {
  position: absolute;
  bottom: 10px;
  right: 10px;
  color: rgba(0, 0, 0, 0.5); /* Adjust the color and opacity of the watermark */
  font-size: 16px;
}
```

OUTPUT:


Personal
React App
localhost:3000
Dynamic Image Selection - Table
Choose Files
No file chosen
Image
Name
G. VIJAY KUMAR'S
Search
ENG
IN
14:32
24-03-2024


Open
Desktop
Images
Search Images
Organize
New folder
Home
Gallery
Pictures
Documents
Music
Desktop
Downloads
Videos
JAVA LAB
gowrishankar.jpg
radhakrishna.jpg
sitaram.jpg
vvit.jpg
File name:
sitaram.jpg
All files (*.*)
Upload from mobile
Open
Cancel
Selection - Table
Name
G. VIJAY KUMAR'S
Search
ENG
IN
14:32
24-03-2024


Personal
React App
localhost:3000
Dynamic Image Selection - Table
Choose Files
sitaram.jpg
Image
Name
sitaram.jpg
G. VIJAY KUMAR'S
Search
ENG
IN
14:32
24-03-2024


Personal
React App
localhost:3000
Dynamic Image Selection - Table
Choose Files
gowrishankar.jpg
Image
Name
sitaram.jpg
radhakrishna.jpg
gowrishankar.jpg
G. VIJAY KUMAR'S
Search
ENG
IN
14:33
24-03-2024

EXPERIMENT NO: 14

AIM:

Implement Upload & down load options on a given file.

DESCRIPTION:

Steps to Implement Upload & Download Options for Files in ReactJS :

Initialize Project: Create a new React project using Create React App or any other method.

Component Structure: Plan the component structure for the file upload and download functionalities. You'll need components for file input, upload button, download button, and handling file operations.

File Upload Input: Implement a file input field (<input type="file">) to allow users to select and upload files from their device.

File Upload Handling: Create a function to handle file uploads. Capture the selected file from the input field, and prepare it for uploading to the server.

Upload Button: Add a button to trigger the file upload process. Attach an event handler to this button to execute the file upload function when clicked.

File Download Handling: Implement a function to handle file downloads. This function should request the file from the server and initiate the download process.

Download Button: Add a button to trigger the file download process. Attach an event handler to this button to execute the file download function when clicked.

Server API: Set up a server-side API (e.g., using Node.js, Express, etc.) to handle file uploads and downloads. Implement endpoints to receive uploaded files, store them, and serve them for download.

File Validation: Implement file validation checks on the client-side to ensure that users upload valid file types and sizes. Display error messages for invalid files.

Styling & UI: Apply CSS or a styling library to design the file upload and download components, buttons, and overall layout. Ensure that the UI is intuitive and user-friendly**.**

Example Components:

App Component: Main component that holds the file upload and download functionalities, and renders the necessary components.

FileInput Component: A reusable component to render the file input field for selecting files.

UploadButton Component: A reusable component to render the upload button and handle file upload events.

DownloadButton Component: A reusable component to render the download button and handle file download events.

FileService: A utility or service module to encapsulate file upload and download logic, making it easier to manage and reuse.

By following these steps and principles, you can create a ReactJS application with functionalities to upload and download files, providing users with a seamless and efficient way to manage file interactions within the application.

App.js :

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

const FileHandlingApp = () => {
  const [file, setFile] = useState(null);
  const [showPreview, setShowPreview] = useState(false);

  const handleFileChange = (event) => {

    const selectedFile = event.target.files[0];
    setFile(selectedFile);
    setShowPreview(false); // Hide preview when a new file is selected
  };

  const handleFileUpload = () => {
    if (file) {
      // You can perform additional actions here (e.g., send the file to a server)
      console.log('File uploaded:', file);
      setShowPreview(true); // Show preview after uploading
    } else {
      console.log('Please select a file first.');
    }
  };

  const handleFileDownload = () => {
    if (file) {
      const blob = new Blob([file], { type: file.type });
      const url = URL.createObjectURL(blob);

      // Create a link element and trigger a click to download the file
      const link = document.createElement('a');
      link.href = url;
      link.download = file.name;
      document.body.appendChild(link);
      link.click();

      // Remove the link from the DOM
      document.body.removeChild(link);
    } else {
      console.log('No file to download.');
    }
  };

  return (
    <div className="container">
      <h1 style={{ borderBottom: '2px solid red', paddingBottom: '10px', marginBottom: '20px', color: 'indigo'}}>File
Handling App</h1>
      <input type="file" onChange={handleFileChange} />
      <br />
      <button onClick={handleFileUpload}>Upload File</button>
      <button onClick={handleFileDownload}>Download File</button>
      {showPreview && (
        <div>
          <h3>File Preview</h3>
          {file.type.startsWith('image/') ? (
```

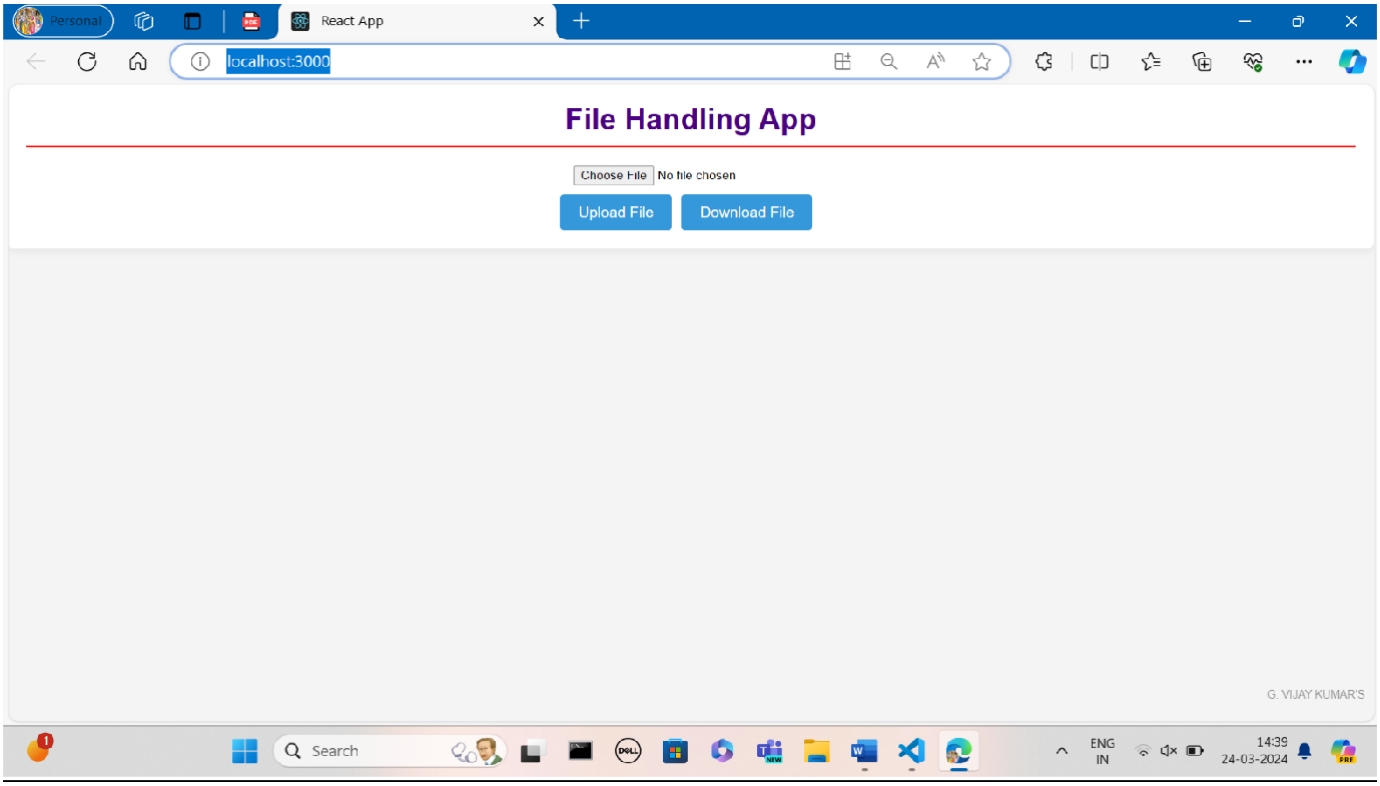
```
            <img
              src={URL.createObjectURL(file)}
              alt="File Preview"
              style={{ maxWidth: '300px', maxHeight: '300px', border: '1px solid #ddd' }}
            />
          ) : (
            <iframe
              title="file-preview"
              style={{ width: '100%', height: '300px', border: '1px solid #ddd' }}
              src={URL.createObjectURL(file)}
            />
          )}
        </div>
      )}
      <div className="watermark">
        <p>G. VIJAY KUMAR'S</p>
      </div>
    </div>
  );
};

export default FileHandlingApp;
```

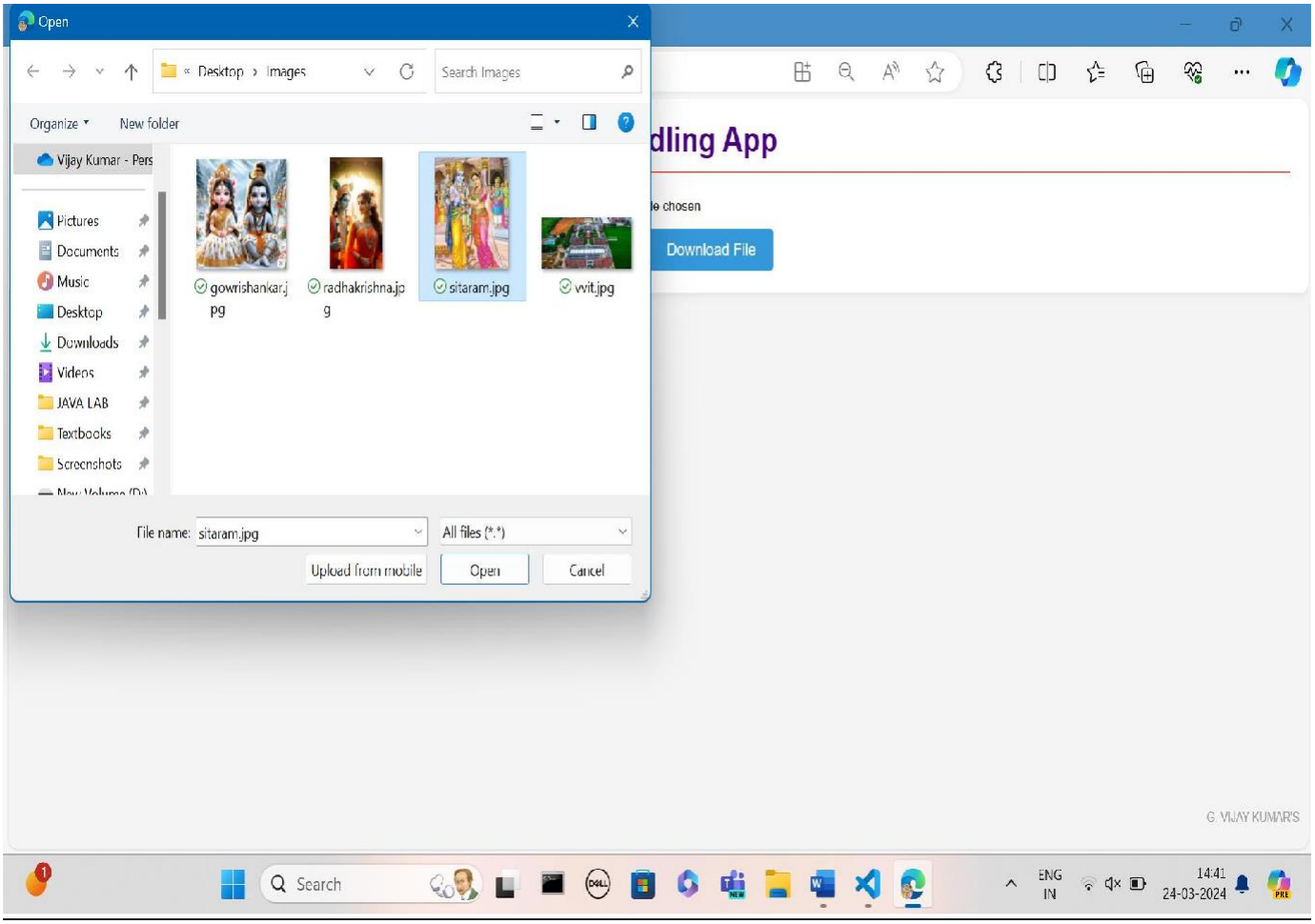
App.css :

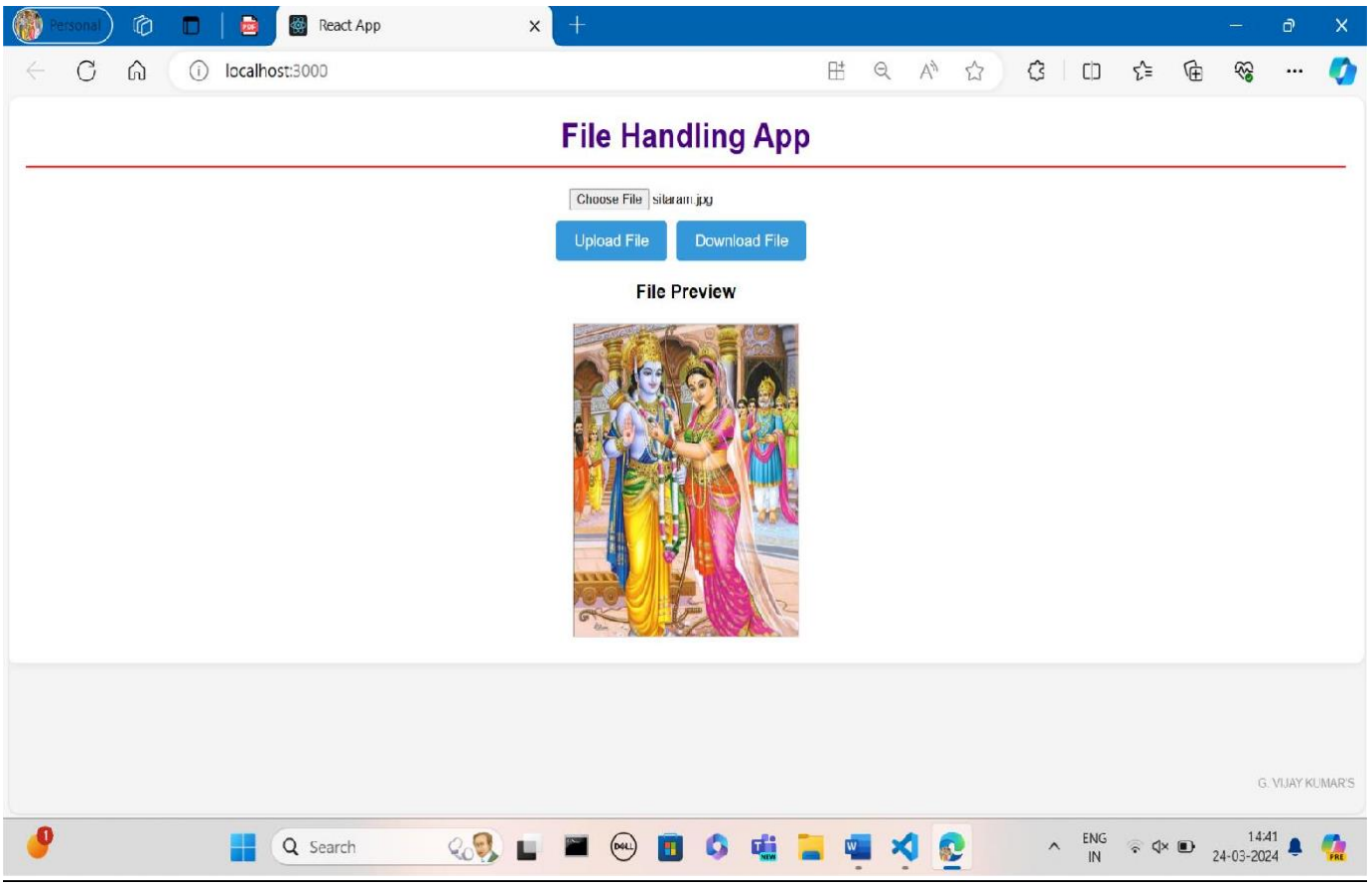
```
body {
  font-family: 'Arial', sans-serif;
  background-color: #f4f4f4;
  margin: 0;
}

.container {
  text-align: center;
  padding: 20px;
  border-radius: 8px;
  background-color: #fff;
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

h1 {
  margin-top: 0;
  border-bottom: 2px solid #db349b;
  padding-bottom: 10px;
}

input[type="file"] {
  margin-bottom: 10px;
}
button {
  margin-right: 10px; /* Add space between buttons */
  background-color: #3498db;
  color: #fff;
  padding: 10px 20px;
  font-size: 16px;
  cursor: pointer;
  border: none;
  border-radius: 4px;
}
```

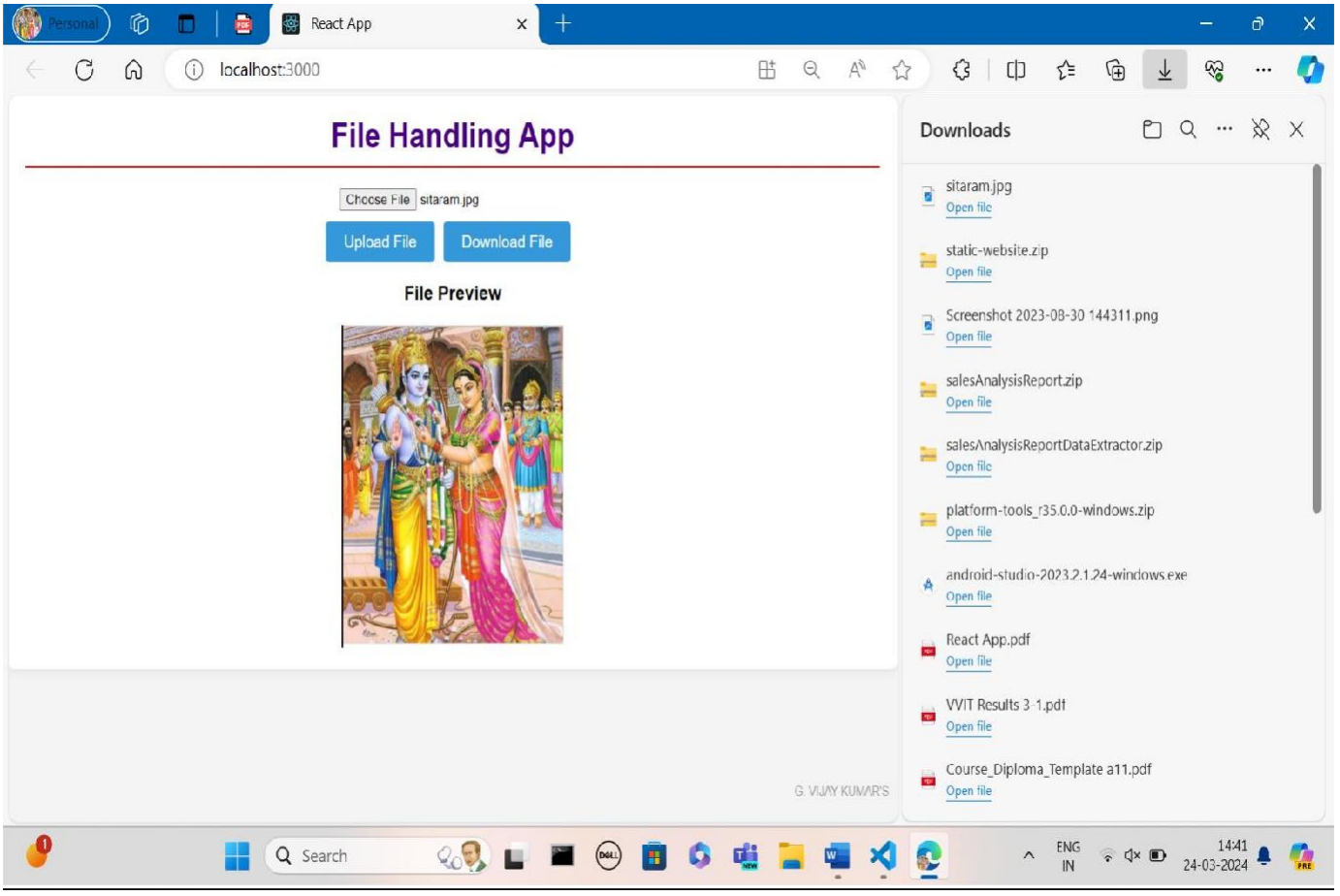
```
button:hover {
  background-color: #297fb8;
}

.watermark {
  position: fixed;
  bottom: 10px;
  right: 10px;
  color: rgba(0, 0, 0, 0.4);
  font-size: 12px;
}
```

OUTPUT:


React App
localhost:3000
File Handling App
Choose File
No file chosen
Upload File
Download File
G. VIJAY KUMAR'S
Search
ENG
IN
14:39
24-03-2024


Open
Desktop › Images
Search Images
Organize
New folder
Vijay Kumar - Pers
Pictures
Documents
Music
Desktop
Downloads
Videos
JAVA LAB
Textbooks
Screenshots
gowrishankar.jpg
radhakrishna.jpg
sitaram.jpg
vvit.jpg
File name: sitaram.jpg
All files (*.*)
Upload from mobile
Open
Cancel
dling App
e chosen
Download File
G. VIJAY KUMAR'S
Search
ENG
IN
14:41
24-03-2024


React App
localhost:3000
File Handling App
Choose File sitaram.jpg
Upload File
Download File
File Preview
G. VIJAY KUMAR'S
Search
ENG
IN
14:41
24-03-2024


React App
localhost:3000
File Handling App
Choose File sitaram.jpg
Upload File
Download File
File Preview
G. VIJAY KUMAR'S
Downloads
sitaram.jpg
Open file
static-website.zip
Open file
Screenshot 2023-08-30 144311.png
Open file
salesAnalysisReport.zip
Open file
salesAnalysisReportDataExtractor.zip
Open file
platform-tools_r35.0.0-windows.zip
Open file
android-studio-2023.2.1.24-windows.exe
Open file
React App.pdf
Open file
VVIT Results 3-1.pdf
Open file
Course_Diploma_Template a11.pdf
Open file
Search
ENG
IN
14:41
24-03-2024

EXPERIMENT NO: 15

AIM:

Create a React application to view EMI calculator.

DESCRIPTION:

Basic View :

$$E = P \times r \times \frac{(1+r)^n}{(1+r)^n - 1}$$

Where,

E is the EMI
P is the principal amount
r is the monthly rate of interest
n is the number of months

Creating an EMI (Equated Monthly Installment) Calculator in ReactJS involves designing a user interface to input loan details such as principal amount, interest rate, and loan tenure. The calculator will then compute the EMI based on these inputs and display the result to the user. Here's a detailed explanation of how to achieve this:

Steps to Create EMI Calculator in ReactJS

1. **Initialize Project**: Create a new React project using Create React App or any other method.
2. **Component Structure**: Plan the component structure for the EMI Calculator. You'll need components for input fields, a button to calculate EMI, and an area to display the result.
3. **State Management**: Use React state to manage the principal amount, interest rate, loan tenure, and computed EMI within the component.
4. **Input Fields**: Create controlled input fields for the principal amount, interest rate, and loan tenure where users can enter their values.
5. **Calculate EMI**: Implement a function to calculate EMI using the formula given;
   Where:
   - `P` = Principal amount
   - `R` = Monthly interest rate (annual rate / 12 / 100)
   - `N` = Loan tenure in months
6. **Button & Event Handling**: Add a button that triggers the EMI calculation when clicked. Implement an event handler to perform the calculation and update the state with the result.

7. **Display Result**: Create a section to display the calculated EMI result to the user.
8. **Styling**: Apply CSS or a styling library to design the EMI Calculator interface, making it user-friendly and visually appealing.
9. **Validation**: Implement validation checks to ensure that users enter valid input values for the principal amount, interest rate, and loan tenure. Display error messages for invalid inputs.
10. **Additional Features**: Enhance the calculator by adding features like currency formatting for input and output values, sliders for selecting interest rates or tenure, and amortization schedules to show payment breakdown over time.

Example Components:

- **App Component**: Main component that holds the state, logic for EMI calculation, and renders the input fields, button, and result display.
- **Input Component**: A reusable component for text inputs, used for principal amount, interest rate, and loan tenure.
- **Button Component**: A reusable button component to trigger the EMI calculation.
- **Result Component**: A component to display the calculated EMI result to the user.

App.js :

```
import React, { useState } from 'react';
import { Container, Form, Button, Card, Row, Col, Table } from 'react-bootstrap';
import 'bootstrap/dist/css/bootstrap.min.css';
import './App.css';
import { saveAs } from 'file-saver';

function App() {
  const [principal, setPrincipal] = useState('');
  const [interestRate, setInterestRate] = useState('');
  const [loanTerm, setLoanTerm] = useState('');
  const [emi, setEmi] = useState([]);
  const [totalPayment, setTotalPayment] = useState(0);
  const [monthlyPayment, setMonthlyPayment] = useState(0);

  const calculateEmi = () => {
    let p = parseFloat(principal);
    const r = parseFloat(interestRate) / 100 / 12;
    const n = parseFloat(loanTerm) * 12;

    const emiValue = (p * r * Math.pow(1 + r, n)) / (Math.pow(1 + r, n) - 1);
    setMonthlyPayment(emiValue.toFixed(2));

    const totalAmount = emiValue * n;
    setTotalPayment(totalAmount.toFixed(2));

    // Use setEmi to update the state variable
    setEmi(Array.from({ length: n }, (_, i) => {
      const interestPayment = p * r;
      const principalPayment = emiValue - interestPayment;
      p -= principalPayment;

      return {
        month: i + 1,
        principal: principalPayment.toFixed(2),
        interest: interestPayment.toFixed(2),
        balance: p.toFixed(2),
```

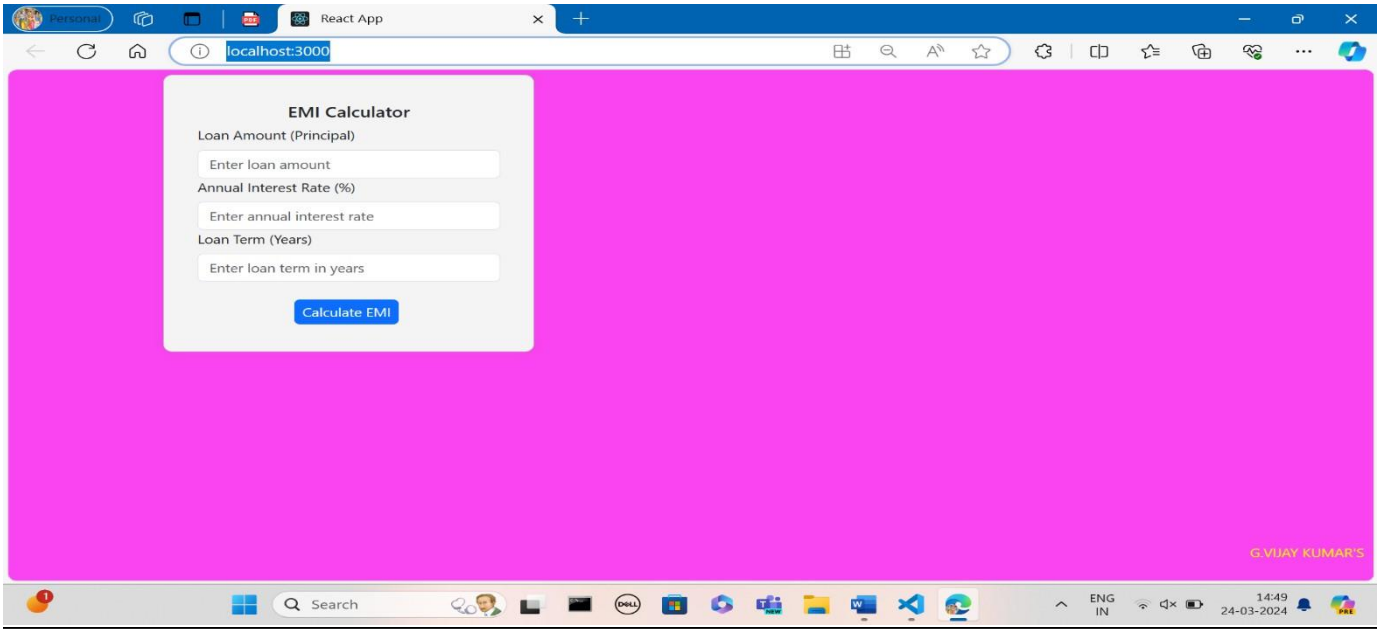
```
    };
  }));
};

const downloadEmiStatement = () => {
  const emiData = emi.map(entry => `${entry.month},${entry.principal},${entry.interest},${entry.balance}`).join('\n');
  const blob = new Blob([`Month,Principal,Interest,Loan Outstanding\n${emiData}`], { type: 'text/csv;charset=utf-8' });
  saveAs(blob, 'emi_statement.csv');
};

return (
  <div>
  <Container className="mt-2">
    <Card className="calculator-card">
      <Card.Body>
        <Card.Title className="text-center">EMI Calculator</Card.Title>
        <Form>
          <Form.Group>
            <Form.Label>Loan Amount (Principal)</Form.Label>
            <Form.Control
              type="number"
              placeholder="Enter loan amount"
              value={principal}
              onChange={(e) => setPrincipal(e.target.value)}
            />
          </Form.Group>
          <Form.Group>
            <Form.Label>Annual Interest Rate (%)</Form.Label>
            <Form.Control
              type="number"
              placeholder="Enter annual interest rate"
              value={interestRate}
              onChange={(e) => setInterestRate(e.target.value)}
            />
          </Form.Group>
          <Form.Group>
            <Form.Label>Loan Term (Years)</Form.Label>
            <Form.Control
              type="number"
              placeholder="Enter loan term in years"
              value={loanTerm}
              onChange={(e) => setLoanTerm(e.target.value)}
            />
          </Form.Group>
          <div className="d-flex justify-content-center mt-4">
            <Button variant="primary" className="small-button" onClick={calculateEmi} style={{ marginRight: '5px' }}>
              Calculate EMI
            </Button>
            {emi.length > 0 && (
              <Button variant="success" className="small-button" onClick={downloadEmiStatement}>
                Download EMI Statement
              </Button>
            )}
          </div>
        </Form>
        {totalPayment !== 0 && monthlyPayment !== 0 && (
          <div className="mt-4">
            <Row>
            <Col xs={6} className="text-right">
```

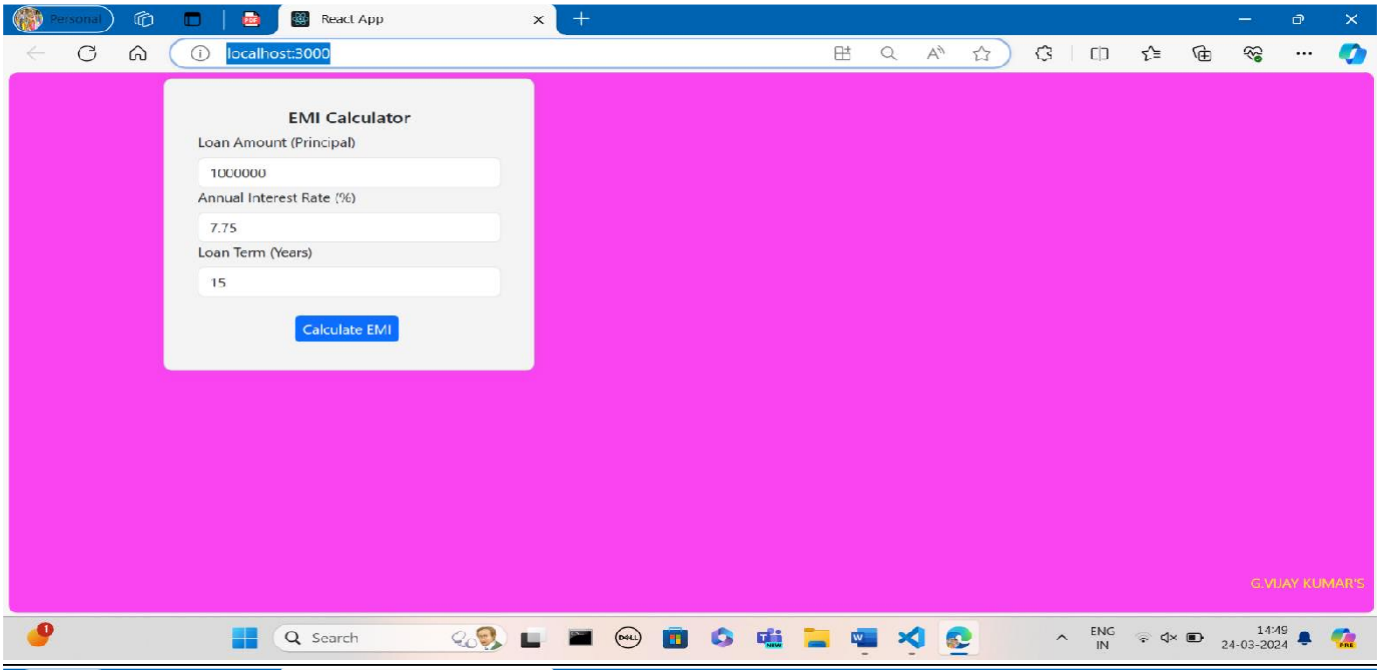
```
            <h6 style={{ color: 'darkblue' }}>Total Amount:</h6>
            <h6 style={{ color: 'darkblue' }}>Principal:</h6>
            <h6 style={{ color: 'darkblue' }}>Interest:</h6>
            <h6 style={{ color: 'red' }}>EMI:</h6>
           </Col>
           <Col xs={6} className="text-left">
            <h6 style={{ color: 'darkblue' }}>₹ {(+totalPayment).toLocaleString('en-IN', { maximumFractionDigits: 2 })} /-
</h6>
            <h6 style={{ color: 'darkblue' }}>₹ {(+principal).toLocaleString('en-IN', { maximumFractionDigits: 2 })} /-</h6>
            <h6 style={{ color: 'darkblue' }}>₹ {((+totalPayment) - (+principal)).toLocaleString('en-IN', {
maximumFractionDigits: 2 })} /-</h6>
            <h6 style={{ color: 'red' }}>₹ {(+monthlyPayment).toLocaleString('en-IN', { maximumFractionDigits: 2 })} /-
<br></br>(per Month)</h6>
           </Col>
           </Row>
          </div>
        )}
      </Card.Body>
    </Card>
    </Container>

    {emi.length > 0 && (
      <div className="mt-4" style={{ overflowY: 'auto', maxHeight: '570px' }}>
      <h5 className="text-center">EMI Statement</h5>
        <Table striped bordered hover size="sm" className="text-center table-sm-font-size">
        <thead>
        <tr>
          <th>Month</th>
          <th>Principal</th>
          <th>Interest</th>
          <th>Loan Outstanding</th>
        </tr>
        </thead>
      <tbody>
        {emi.map((entry) => (
          <tr key={entry.month}>
            <td>{entry.month}</td>
            <td>₹ {Math.floor(entry.principal)}</td>
            <td>₹ {Math.floor(entry.interest)}</td>
            <td>₹ {Math.floor(entry.balance)}</td>
          </tr>
        ))}
      </tbody>
      </Table>
    </div>
   )}




   <div className="watermark">
      <p>G VIJAY KUMAR'S </p>
    </div>
  </div>
  );
}

export default App;
```

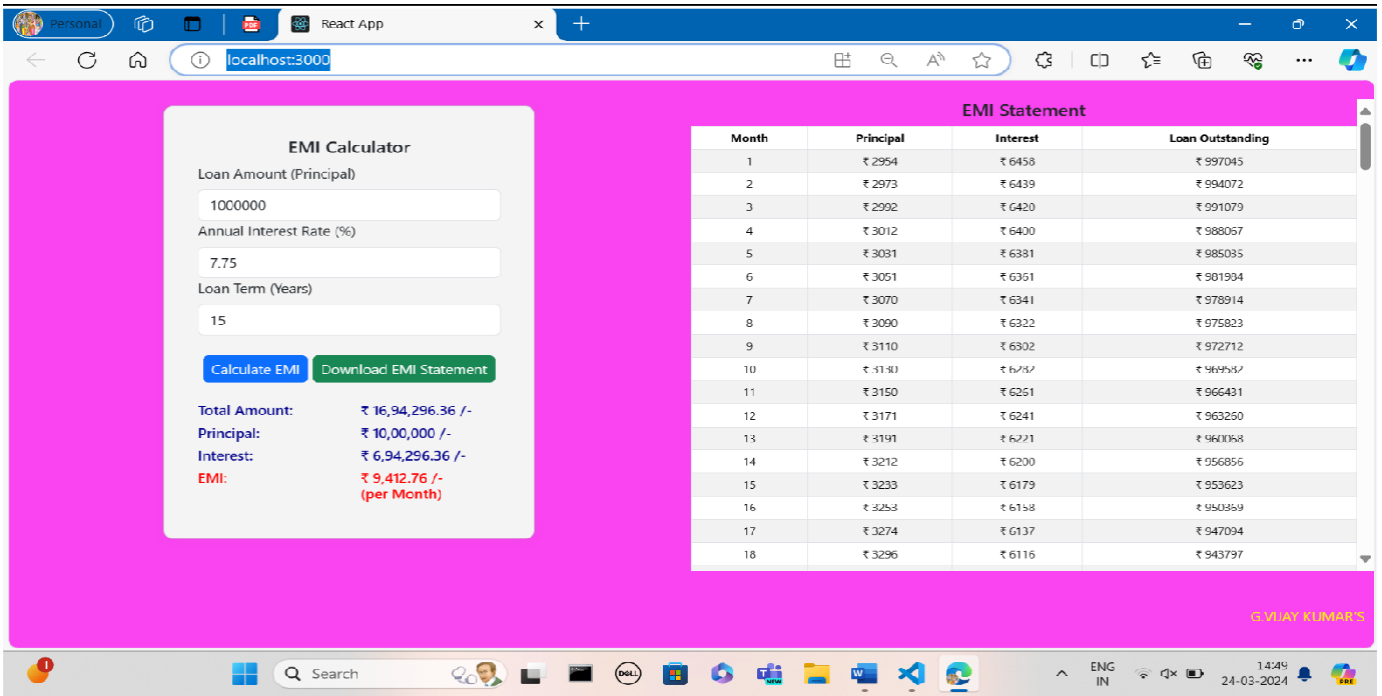
App.css :

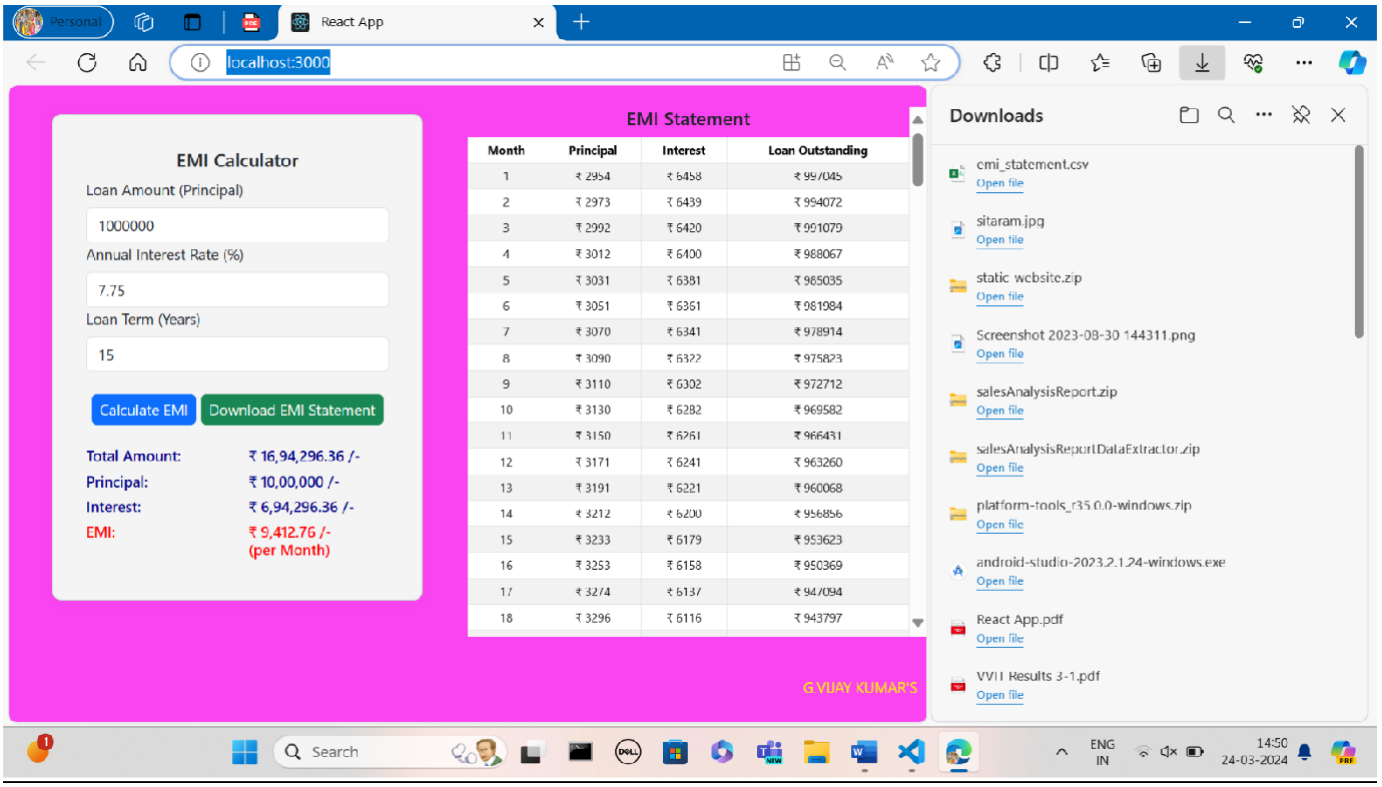
```
.calculator-card{
  max-width:400px;
  margin:auto;
  padding:20px;
  border-radius:10px;
  height:100%;
  background-color: #f4f4f4;
  box-shadow: 0 0 10px rgba(141,156,142,0.1);
}
.result{
  color:#ff0000;
}
.text-center{
  text-align:center;
}
body{
  margin:0;
  padding:0;
  overflow:hidden;
  background-color: #fa43f1;
}
.container{
  float:left;
  width:50%;
}
.table-container{
  float:left;
  width:50%;
  height:100vh;
  overflow-y: auto;
}
.table-sm-font-size{
  font-size:0.8rem !important;
}
.small-button{
  padding: 0.25rem 0.5rem;
}
.small-text{
  font-size:0.8rem;
}
.watermark{
  position:fixed;
  bottom:10px;
  right: 10px;
  color: rgb(250,246,8);
  font-size: 0.8rem;
}
```

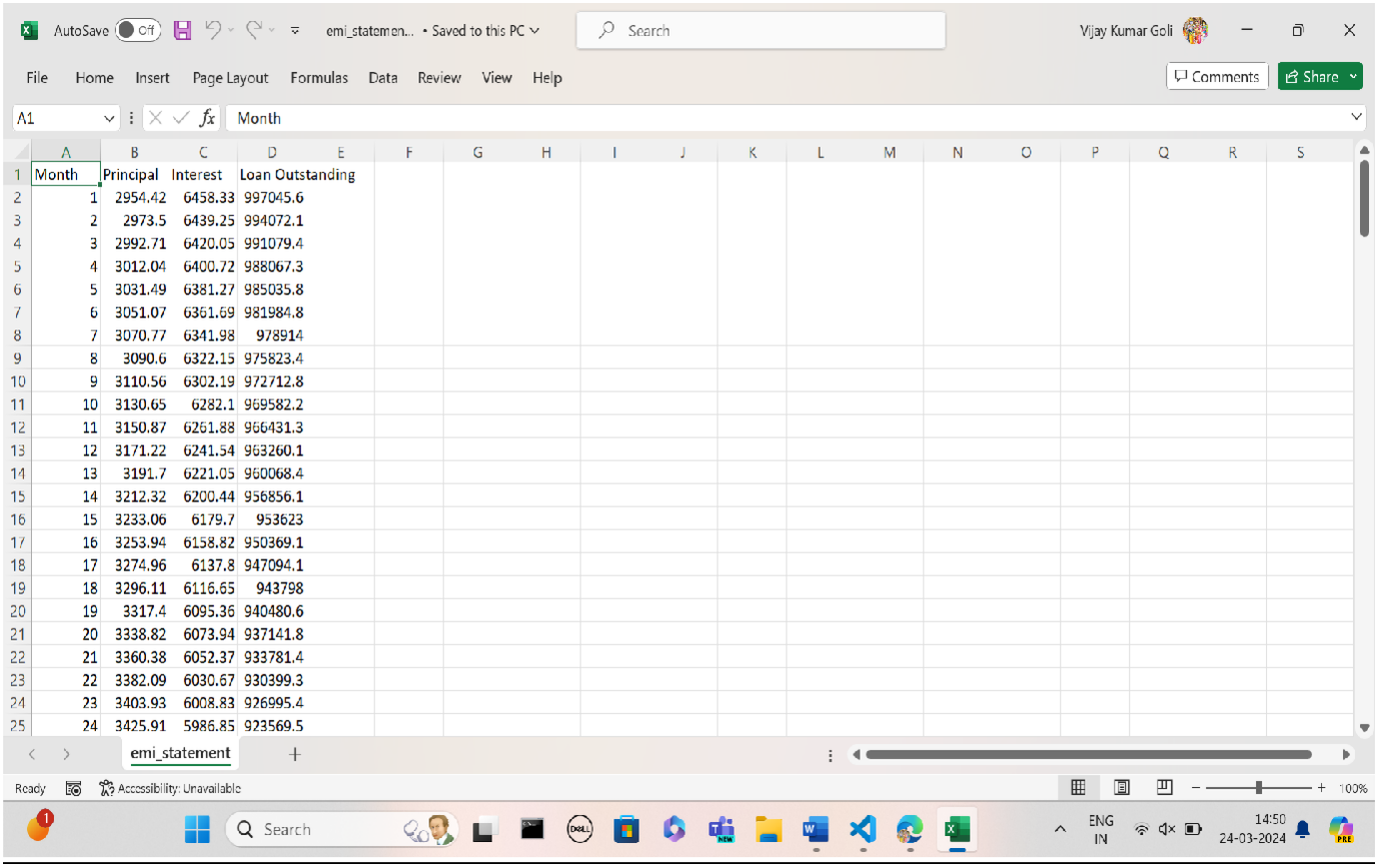
OUTPUT:


React App
localhost:3000
EMI Calculator
Loan Amount (Principal)
Enter loan amount
Annual Interest Rate (%)
Enter annual interest rate
Loan Term (Years)
Enter loan term in years
Calculate EMI
G.VIJAY KUMAR'S


React App
localhost:3000
EMI Calculator
Loan Amount (Principal)
1000000
Annual Interest Rate (%)
7.75
Loan Term (Years)
15
Calculate EMI
G.VIJAY KUMAR'S


React App
localhost:3000
EMI Calculator
Loan Amount (Principal)
1000000
Annual Interest Rate (%)
7.75
Loan Term (Years)
15
Calculate EMI
Download EMI Statement
Total Amount: ₹ 16,94,296.36 /-
Principal: ₹ 10,00,000 /-
Interest: ₹ 6,94,296.36 /-
EMI: ₹ 9,412.76 /- (per Month)
EMI Statement
Month | Principal | Interest | Loan Outstanding
1 | ₹ 2954 | ₹ 6458 | ₹ 997045
2 | ₹ 2973 | ₹ 6439 | ₹ 994072
3 | ₹ 2992 | ₹ 6420 | ₹ 991079
4 | ₹ 3012 | ₹ 6400 | ₹ 988057
5 | ₹ 3031 | ₹ 6381 | ₹ 985035
6 | ₹ 3051 | ₹ 6361 | ₹ 981984
7 | ₹ 3070 | ₹ 6341 | ₹ 978914
8 | ₹ 3090 | ₹ 6322 | ₹ 975823
9 | ₹ 3110 | ₹ 6302 | ₹ 972712
10 | ₹ 3130 | ₹ 6282 | ₹ 969582
11 | ₹ 3150 | ₹ 6251 | ₹ 966431
12 | ₹ 3171 | ₹ 6241 | ₹ 963260
13 | ₹ 3191 | ₹ 6221 | ₹ 960068
14 | ₹ 3212 | ₹ 6200 | ₹ 956856
15 | ₹ 3233 | ₹ 6179 | ₹ 953623
16 | ₹ 3253 | ₹ 6158 | ₹ 950369
17 | ₹ 3274 | ₹ 6137 | ₹ 947094
18 | ₹ 3296 | ₹ 6116 | ₹ 943797
G.VIJAY KUMAR'S


React App
localhost:3000
EMI Calculator
Loan Amount (Principal)
1000000
Annual Interest Rate (%)
7.75
Loan Term (Years)
15
Calculate EMI
Download EMI Statement
Total Amount: ₹ 16,94,296.36 /-
Principal: ₹ 10,00,000 /-
Interest: ₹ 6,94,296.36 /-
EMI: ₹ 9,412.76 /- (per Month)
EMI Statement
Month Principal Interest Loan Outstanding
1 ₹ 2954 ₹ 6458 ₹ 997045
2 ₹ 2973 ₹ 6439 ₹ 994072
3 ₹ 2992 ₹ 6420 ₹ 991079
4 ₹ 3012 ₹ 6400 ₹ 988067
5 ₹ 3031 ₹ 6381 ₹ 985035
6 ₹ 3051 ₹ 6361 ₹ 981984
7 ₹ 3070 ₹ 6341 ₹ 978914
8 ₹ 3090 ₹ 6322 ₹ 975823
9 ₹ 3110 ₹ 6302 ₹ 972712
10 ₹ 3130 ₹ 6282 ₹ 969582
11 ₹ 3150 ₹ 6261 ₹ 966431
12 ₹ 3171 ₹ 6241 ₹ 963260
13 ₹ 3191 ₹ 6221 ₹ 960068
14 ₹ 3212 ₹ 6200 ₹ 956856
15 ₹ 3233 ₹ 6179 ₹ 953623
16 ₹ 3253 ₹ 6158 ₹ 950369
17 ₹ 3274 ₹ 6137 ₹ 947094
18 ₹ 3296 ₹ 6116 ₹ 943797
G VIJAY KUMAR'S
Downloads
cmi_statement.csv
sitaram.jpg
static website.zip
Screenshot 2023-08-30 144311.png
salesAnalysisReport.zip
salesAnalysisReportDataExtractor.zip
platform-tools_r35.0.0-windows.zip
android-studio-2023.2.1.24-windows.exe
React App.pdf
VVIT Results 3-1.pdf
Open file
Search
ENG IN
14:50 24-03-2024


AutoSave Off
emi_statemen... • Saved to this PC
Search
Vijay Kumar Goli
File Home Insert Page Layout Formulas Data Review View Help
Comments
Share
A1
Month
Month Principal Interest Loan Outstanding
1 2954.42 6458.33 997045.6
2 2973.5 6439.25 994072.1
3 2992.71 6420.05 991079.4
4 3012.04 6400.72 988067.3
5 3031.49 6381.27 985035.8
6 3051.07 6361.69 981984.8
7 3070.77 6341.98 978914
8 3090.6 6322.15 975823.4
9 3110.56 6302.19 972712.8
10 3130.65 6282.1 969582.2
11 3150.87 6261.88 966431.3
12 3171.22 6241.54 963260.1
13 3191.7 6221.05 960068.4
14 3212.32 6200.44 956856.1
15 3233.06 6179.7 953623
16 3253.94 6158.82 950369.1
17 3274.96 6137.8 947094.1
18 3296.11 6116.65 943798
19 3317.4 6095.36 940480.6
20 3338.82 6073.94 937141.8
21 3360.38 6052.37 933781.4
22 3382.09 6030.67 930399.3
23 3403.93 6008.83 926995.4
24 3425.91 5986.85 923569.5
emi_statement
Ready
Accessibility: Unavailable
100%
Search
ENG IN
14:50 24-03-2024

EXPERIMENT NO: 16

AIM:

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:

Designing a Hotel Bill screen in ReactJS involves creating a user-friendly interface where users can select multiple items from a dropdown box, enter quantities in a text field, and view the transaction details along with the bill amount in a table. Here's a detailed explanation of how to achieve this:

Steps to Design Hotel Bill Screen in ReactJS

1. **Initialize Project**: Create a new React project using Create React App or any other method.
2. **Component Structure**: Plan the component structure for the Hotel Bill screen. You'll need components for dropdown boxes, text fields, buttons, and a table to display transaction details.
3. **State Management**: Use React state to manage the selected items, quantities, and transaction details within the component.
4. **Dropdown Boxes**: Create dropdown boxes populated with available items (e.g., food items, beverages, etc.) from which users can select multiple items.
5. **Text Fields**: Provide text fields next to each dropdown box where users can enter quantities for the selected items.
6. **Add to Bill Button**: Implement a button next to each dropdown and text field pair to add the selected item and quantity to the bill.
7. **Table for Transactions**: Create a table to display the transaction details, including item names, quantities, prices, and total amounts. Update this table dynamically as users add items to the bill.
8. **Calculate Bill Amount**: Implement a function to calculate the total bill amount based on the selected items, quantities, and their respective prices.
9. **Display Bill Amount**: Create an area to display the total bill amount to the user, updating it whenever the bill details change.
10. **Styling**: Apply CSS or a styling library to design the Hotel Bill screen interface, making it user-friendly, visually appealing, and easy to navigate.

Example Components:

- **App Component**: Main component that holds the state, logic for adding items to the bill, and renders the dropdown boxes, text fields, buttons, table, and bill amount display.
- **Dropdown Component**: A reusable component for dropdown boxes to select items.
- **TextField Component**: A reusable component for text fields to enter quantities.
- **Button Component**: A reusable button component to add items to the bill.
- **Table Component**: A component to display transaction details in a table format.
- **BillAmount Component**: A component to display the total bill amount to the user.

App.js :

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

const HotelBillScreen = () => {
  const [selectedItems, setSelectedItems] = useState([]);
  const [itemQuantity, setItemQuantity] = useState(1);
  const [transactions, setTransactions] = useState([]);
  const [currentDateTime, setCurrentDateTime] = useState(new Date());


  const items = [
    { id: 1, name: 'Idly', cost: 10 },
    { id: 2, name: 'Dosa', cost: 20 },
    { id: 3, name: 'Vada', cost: 20 },
    { id: 4, name: 'Poori', cost: 20 },
    { id: 5, name: 'Upma', cost: 20 },
    { id: 6, name: 'Chapati', cost: 20 },
    { id: 7, name: 'Bonda', cost: 20 },
    { id: 8, name: 'Vada', cost: 20 },
    { id: 9, name: 'Lemon Rice', cost: 30 },
    { id: 10, name: 'Tomato Rice', cost: 30 },
  ];

  const addItem = () => {
    if (selectedItems.length > 0) {
      const newTransaction = {
        id: transactions.length + 1,
        itemName: selectedItems[0].name,
        quantity: itemQuantity,
        costPerItem: selectedItems[0].cost,
      };

      setTransactions([...transactions, newTransaction]);
      setSelectedItems([]);
      setItemQuantity(1);
    } else {
      alert('Please select an item!');
    }
  };

  const calculateTotalCost = () => {
    return transactions.reduce(
      (total, transaction) => total + transaction.quantity * transaction.costPerItem,0);
  };

  const calculateGST = () => {
    const totalCost = calculateTotalCost();
    return (5 / 100) * totalCost;
  };

  const calculateBillAmount = () => {
    const totalCost = calculateTotalCost();
    const gst = calculateGST();
    const billAmount = totalCost + gst;
    return Math.ceil(billAmount);
  };
```

```
  setInterval(() => {
    setCurrentDateTime(new Date());
  }, 1000);

  const currentDate = new Date();
const formattedDate = currentDate.toLocaleDateString('en-US', {
  year: 'numeric',
  month: 'short',
  day: 'numeric',
});

const formattedTime = currentDate.toLocaleTimeString('en-US', {
  hour: 'numeric',
  minute: 'numeric',
  hour12: true,
});

const handlePrint = () => {
  window.print();
};

  return (
    <div
      style={{
        maxWidth: '600px',
        margin: 'auto',
        padding: '20px',
        fontFamily: 'Arial, sans-serif',
      }}
    >
      <h2 style={{ textAlign: 'center', color: '#333' }}>G.Vijay's Hotel Invoice</h2>

      <div style={{ display: 'flex', justifyContent: 'center', alignItems: 'center'}}>
    <h4 style={{ color: '#333' }}>{formattedDate} {formattedTime}</h4>
  </div>

      <hr style={{ border: '1px solid #ccc', margin: '15px 0' }} />

      <div style={{ display: 'flex', justifyContent: 'center', alignItems: 'center', marginBottom: '15px' }}>
  <div style={{ flex: 1, marginRight: '10px' }}>
    <label htmlFor="item" style={{ display: 'block', marginBottom: '5px', color: '#555', fontSize: '14px' }}>
      Select Item:
    </label>
    <select
      id="item"
      style={{ width: '100%', padding: '8px', fontSize: '14px', borderRadius: '4px' }}
      onChange={(e) => {
        const selectedItem = items.find((item) => item.id === parseInt(e.target.value));
        setSelectedItems([selectedItem]);
      }}
      value={selectedItems.length > 0 ? selectedItems[0].id : ''}
    >
      <option value="" disabled>
        Select an item
      </option>
      {items.map((item) => (
        <option key={item.id} value={item.id}>
          {item.name}
```

```
          </option>
        ))}
      </select>
    </div>
    <div style={{ flex: 1, marginRight: '10px' }}>
      <label htmlFor="itemQuantity" style={{ display: 'block', marginBottom: '5px', color: '#555', fontSize: '14px' }}>
        Number of Selected Items:
      </label>
      <input
        type="number"
        id="itemQuantity"
        value={itemQuantity}
        style={{ width: '100%', padding: '8px', fontSize: '14px', borderRadius: '4px' }}
        onChange={(e) => setItemQuantity(parseInt(e.target.value))}
      />
    </div>
    <button
      style={{
        background: 'darkblue',
        color: 'white',
        border: 'none',
        padding: '12px 20px',
        borderRadius: '5px',
        cursor: 'pointer',
        fontSize: '14px',
        transition: 'background 0.3s ease',
        marginLeft: '14px',
      }}
      onClick={addItem}
      onMouseOver={(e) => e.target.style.background = 'orange'}
      onMouseOut={(e) => e.target.style.background = 'darkblue'}
    >
      Add Item
    </button>
  </div>


      <div style={{ marginTop: '20px' }}>
        <hr style={{ border: '1px solid #ccc', margin: '15px 0' }} />
        <h4 style={{ textAlign: 'center', color: '#333' }}>List of Items</h4>
        <table className="category-table" style={{ width: '100%', borderCollapse: 'collapse', marginTop: '10px', }}>
        <thead>
        <tr style={{ background: '#0e0789', color: 'white' }}>
            <th style={{ textAlign: 'center', padding: '8px' }}>S.No.</th>
            <th style={{ textAlign: 'center', padding: '8px' }}>Item Name</th>
            <th style={{ textAlign: 'center', padding: '8px' }}>Item Quantity</th>
            <th style={{ textAlign: 'center', padding: '8px' }}>Cost Per Item</th>
            <th style={{ textAlign: 'center', padding: '8px' }}>Total Cost</th>
          </tr>
        </thead>
        <tbody>
          {transactions.map((transaction, index) => (
            <tr key={index} style={{ borderBottom: '1px solid #ccc' }}>
              <td style={{ textAlign: 'center', padding: '8px' }}>{index + 1}</td>
              <td style={{ textAlign: 'left', padding: '8px' }}>{transaction.itemName}</td>
              <td style={{ textAlign: 'center', padding: '8px' }}>{transaction.quantity}</td>
              <td style={{ textAlign: 'center', padding: '8px' }}>{transaction.costPerItem}</td>
              <td style={{ textAlign: 'right', padding: '8px' }}>
                {transaction.quantity * transaction.costPerItem}
```

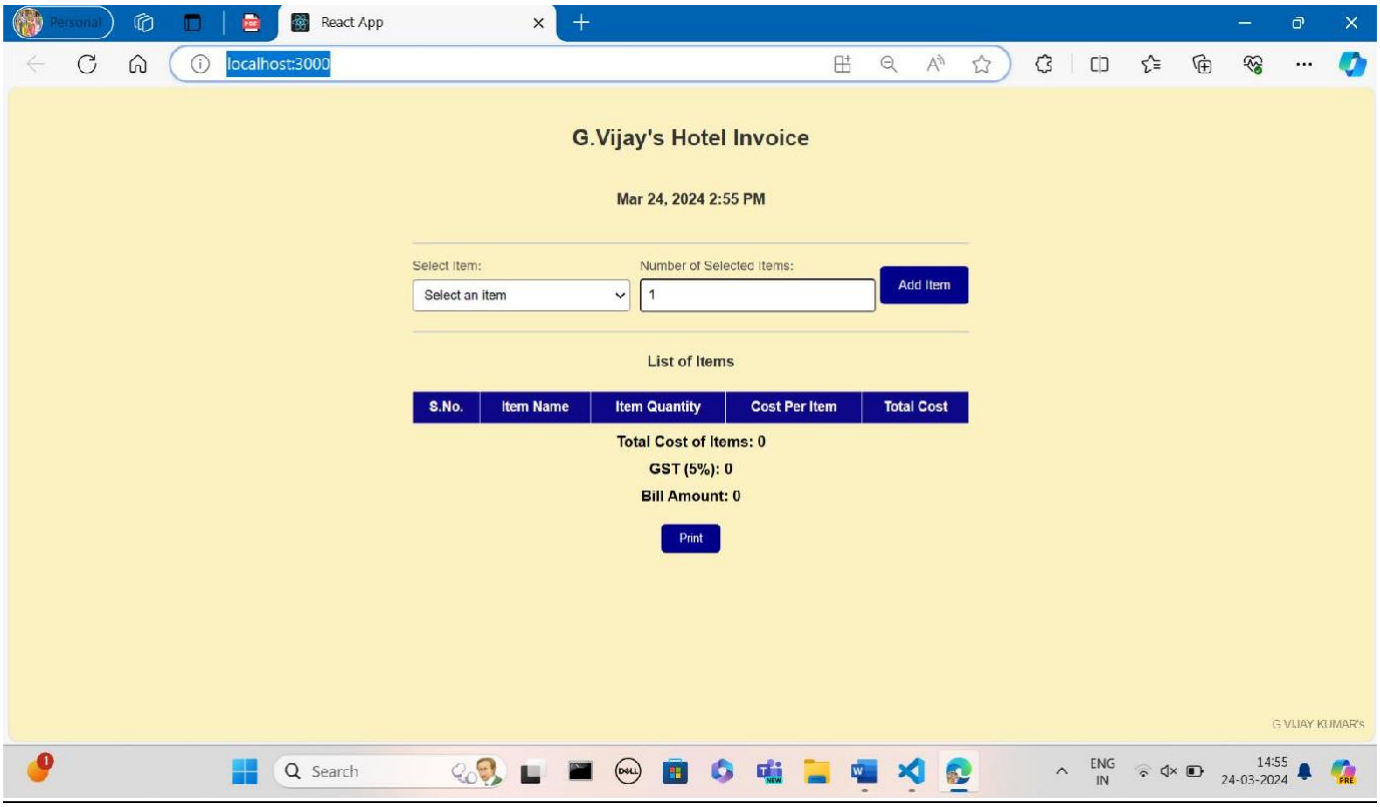
```
              </td>
            </tr>
          ))}
        </tbody>
      </table>
      <div style={{ marginTop: '10px', textAlign: 'center' }}>
        <strong>Total Cost of Items: {calculateTotalCost()}</strong>
      </div>
      <div style={{ marginTop: '10px', textAlign: 'center' }}>
        <strong>GST (5%): {calculateGST()}</strong>
      </div>
      <div style={{ marginTop: '10px', textAlign: 'center' }}>
        <strong>Bill Amount: {calculateBillAmount()}</strong>
      </div>
    </div>
    <div style={{ textAlign: 'center', marginTop: '20px' }}>
      <button
        style={{
          background: 'darkblue',
          color: 'white',
          border: 'none',
          padding: '8px 20px',
          borderRadius: '5px',
          cursor: 'pointer',
          fontSize: '12px',
          transition: 'background 0.3s ease',
        }}
        onClick={handlePrint}
        onMouseOver={(e) => e.target.style.background = 'orange'}
        onMouseOut={(e) => e.target.style.background = 'darkblue'}
      >
        Print
      </button>
    </div>
    <div className="watermark">G.VIJAY KUMAR's</div>
  </div>
 );
};

export default HotelBillScreen;
```

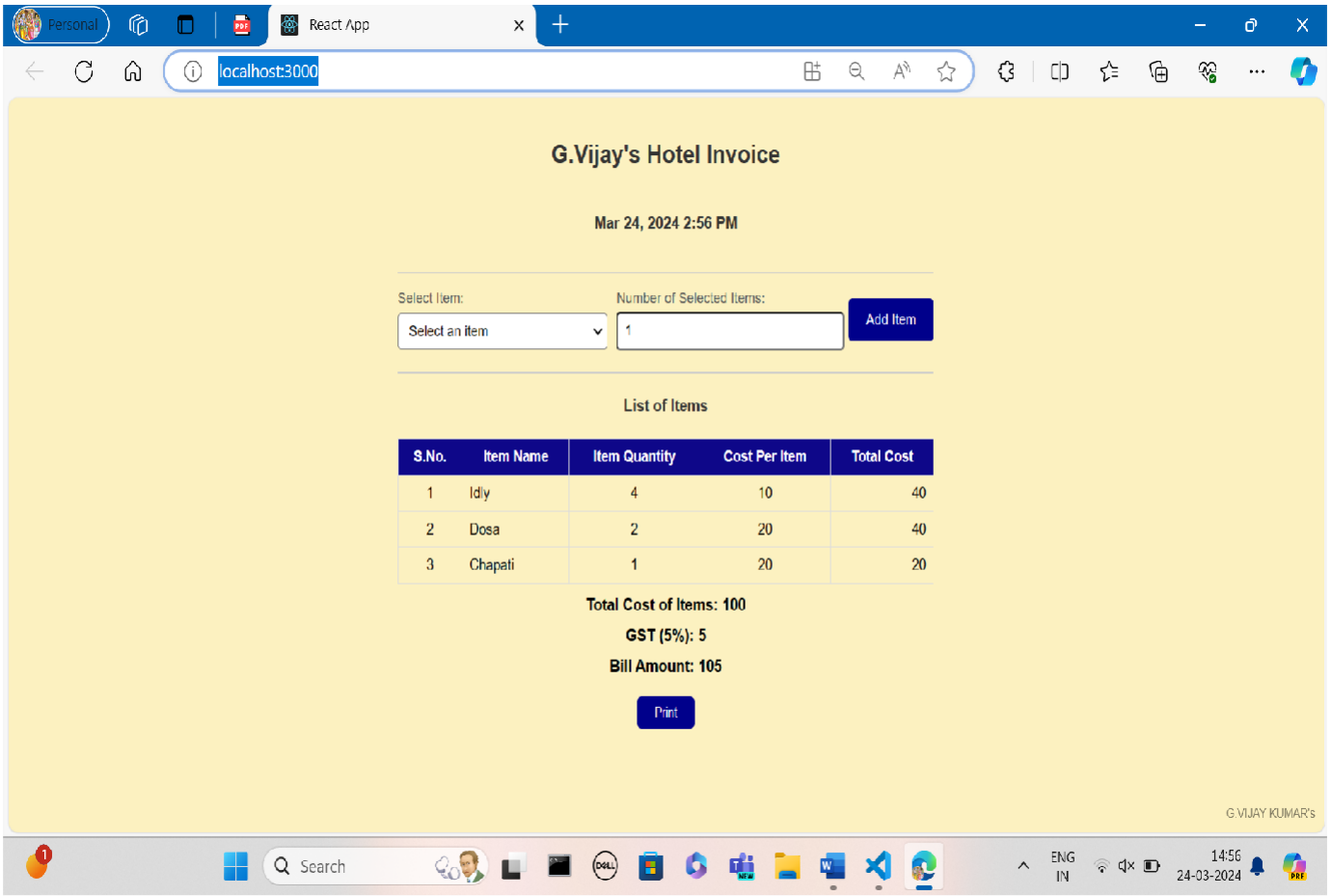
App.css :

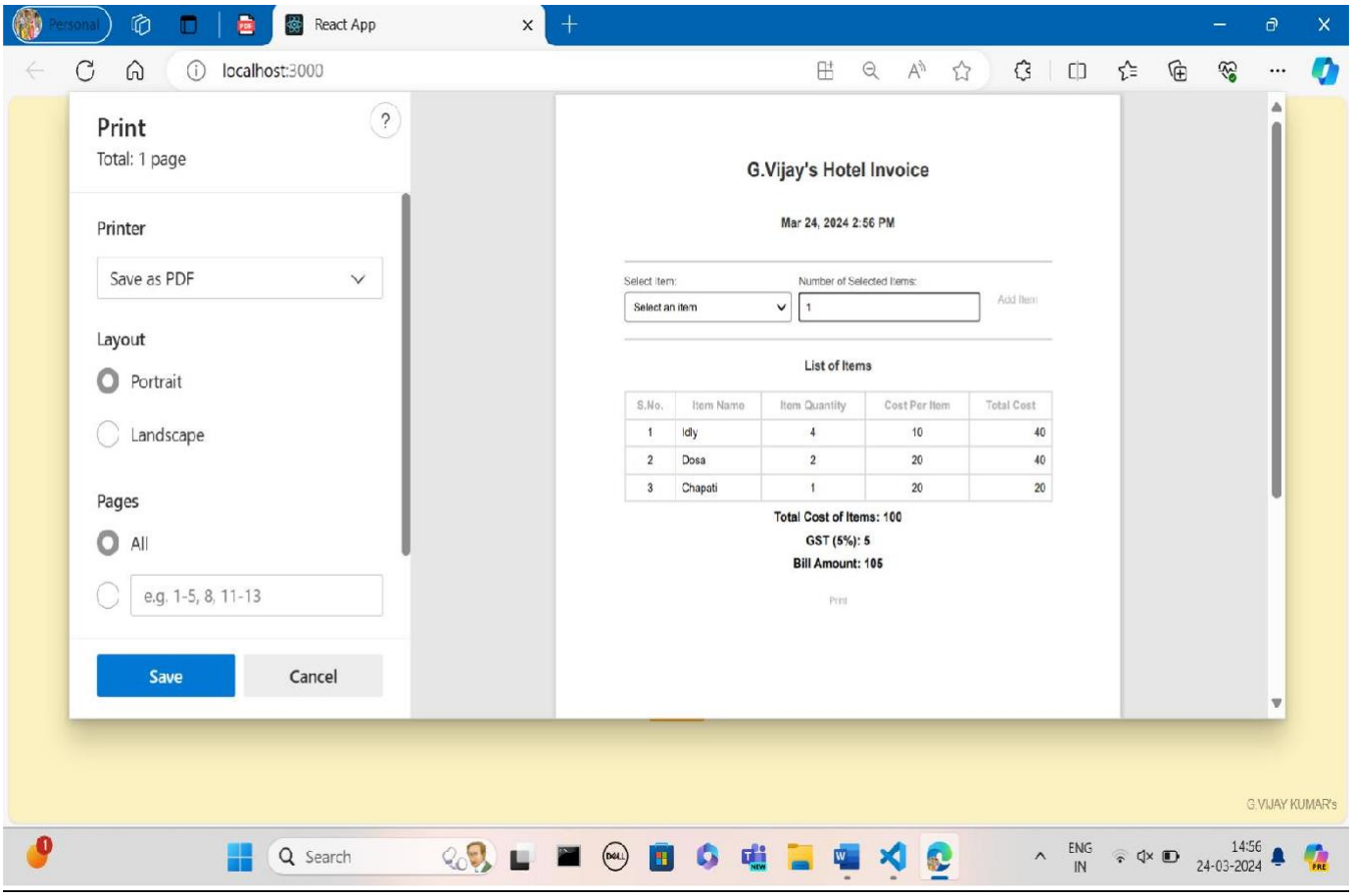
```
.App {
  background-color: blue;
}

body {
  background-color: #fcf1c0;
  margin: 0;
  padding: 0;
  font-family: Arial,sans-serif;
}

.category-table th,.category-table td {
  border: 1px solid #ddd;
  padding: 8px;
  text-align: left;
  font-size: 0.9em;
```

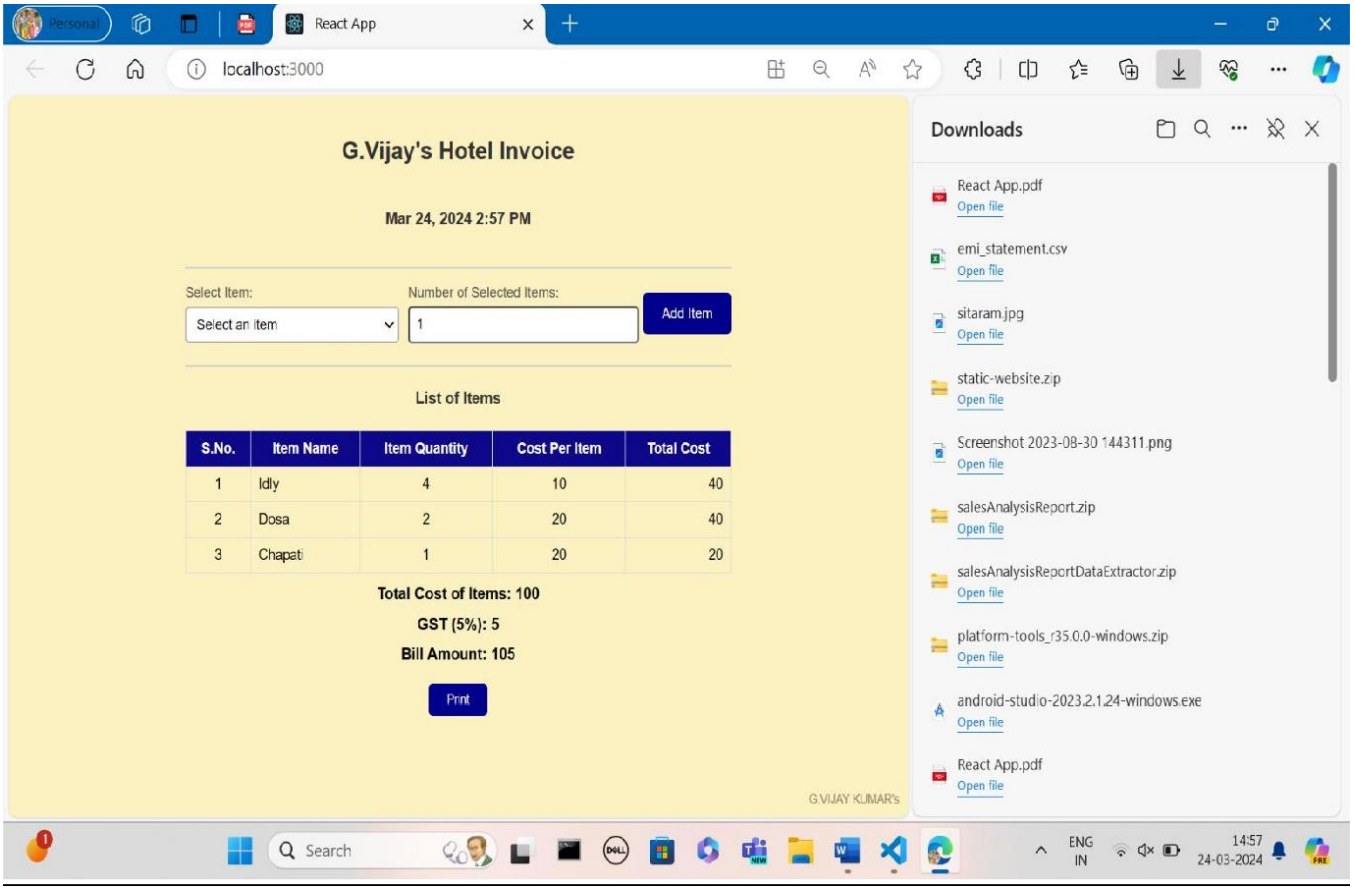
```
}

.App-logo {
  height: 40vmin;
  pointer-events: none;
}

@media (prefers-reduced-motion: no-preference) {
  .App-logo {
    animation: App-logo-spin infinite 20s linear;
  }
}

.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: rgb(247,127,133);
}

.App-link {
  color: #61dafb;
}

@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}

.watermark {
  position: fixed;
  bottom: 10px;
  right:10px;
  color:rgba(0,0,0,0.5);
  font-size: 12px;
}
```

OUTPUT:


React App
localhost:3000
G.Vijay's Hotel Invoice
Mar 24, 2024 2:55 PM
Select Item:
Select an item
Number of Selected Items:
1
Add Item
List of Items
S.No.
Item Name
Item Quantity
Cost Per Item
Total Cost
Total Cost of Items: 0
GST (5%): 0
Bill Amount: 0
Print
G VIJAY KUMAR's
Search
ENG
IN
14:55
24-03-2024


React App
localhost:3000
G.Vijay's Hotel Invoice
Mar 24, 2024 2:56 PM
Select Item:
Select an item
Number of Selected Items:
1
Add Item
List of Items
S.No.
Item Name
Item Quantity
Cost Per Item
Total Cost
1
Idly
4
10
40
2
Dosa
2
20
40
3
Chapati
1
20
20
Total Cost of Items: 100
GST (5%): 5
Bill Amount: 105
Print
G VIJAY KUMAR's
Search
ENG
IN
14:56
24-03-2024


React App
localhost:3000
Print
Total: 1 page
Printer
Save as PDF
Layout
Portrait
Landscape
Pages
All
e.g. 1-5, 8, 11-13
Save
Cancel
G.Vijay's Hotel Invoice
Mar 24, 2024 2:56 PM
Select Item:
Select an item
Number of Selected Items:
1
Add Item
List of Items
S.No. | Item Name | Item Quantity | Cost Per Item | Total Cost
1 | Idly | 4 | 10 | 40
2 | Dosa | 2 | 20 | 40
3 | Chapati | 1 | 20 | 20
Total Cost of Items: 100
GST (5%): 5
Bill Amount: 105
Print
G.VIJAY KUMAR's
Search
ENG
IN
14:56
24-03-2024


React App
localhost:3000
G.Vijay's Hotel Invoice
Mar 24, 2024 2:57 PM
Select Item:
Select an item
Number of Selected Items:
1
Add Item
List of Items
S.No. | Item Name | Item Quantity | Cost Per Item | Total Cost
1 | Idly | 4 | 10 | 40
2 | Dosa | 2 | 20 | 40
3 | Chapati | 1 | 20 | 20
Total Cost of Items: 100
GST (5%): 5
Bill Amount: 105
Print
G.VIJAY KUMAR's
Downloads
React App.pdf
Open file
emi_statement.csv
Open file
sitaram.jpg
Open file
static-website.zip
Open file
Screenshot 2023-08-30 144311.png
Open file
salesAnalysisReport.zip
Open file
salesAnalysisReportDataExtractor.zip
Open file
platform-tools_r35.0.0-windows.zip
Open file
android-studio-2023.2.1.24-windows.exe
Open file
React App.pdf
Open file
Search
ENG
IN
14:57
24-03-2024

EXPERIMENT NO: 17

AIM:

Demonstrate the procedure to create a schema in MongoDB.

DESCRIPTION:

Creating a schema in MongoDB involves defining the structure of your documents within a collection. Here's a detailed explanation of the procedure to create a schema in MongoDB:

Steps to Create a Schema in MongoDB

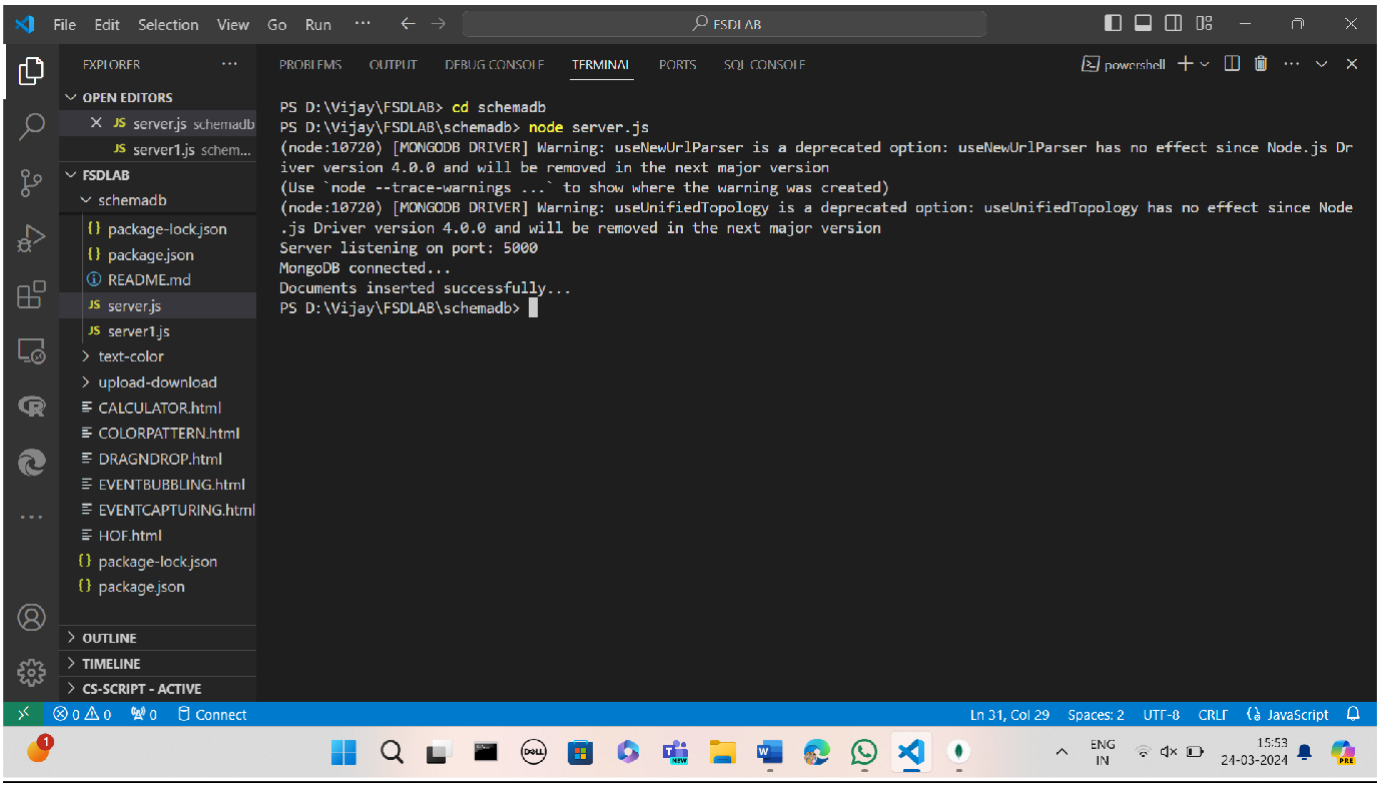
1. **Database Selection**: Connect to your MongoDB instance and select the database where you want to create the schema or collection.
2. **Collection Creation**: Choose a name for your collection and create it using the MongoDB shell or a MongoDB client.
3. **Document Structure**: Define the structure of your documents based on the data you want to store. Identify the fields that each document will contain, their data types, and any validation rules.
4. **Field Definitions**:
    - **Field Names**: Assign descriptive and meaningful names to your fields.
    - **Data Types**: Choose appropriate data types for your fields (e.g., string, number, boolean, date, object, array).
    - **Validation**: Optionally, define validation rules to enforce data integrity and consistency (e.g., required fields, minimum/maximum values, allowed values).
5. **Embedding & Referencing**:
    - **Embedding**: Decide whether to embed related data within a document using arrays or sub-documents (embedded documents).
    - **Referencing**: Alternatively, use references (ObjectIds) to link documents across collections.
6. **Indexes**:
    - **Index Creation**: Identify fields that require indexing for efficient query performance (e.g., frequently queried fields, fields used in sorting or filtering).
    - **Index Types**: Choose appropriate index types (e.g., single field, compound, text, geospatial) based on your query patterns.
7. **Schema Evolution**:
    - **Versioning**: Consider versioning your schema if it's expected to evolve over time to handle changes gracefully.
    - **Migration Strategy**: Plan a migration strategy to update existing documents when schema changes occur.
8. **Documentation & Guidelines**:
    - **Documentation**: Document your schema design, including field descriptions, data types, validation rules, and indexing strategies.
    - **Guidelines**: Establish guidelines and best practices for maintaining and updating the schema to ensure consistency and reliability.
9. **Testing & Validation**:
    - **Data Insertion**: Insert sample data into your collection to test the schema and verify that it meets your requirements.
    - **Query Testing**: Execute test queries to validate the schema's effectiveness in retrieving and manipulating data.
10. **Monitoring & Optimization**:
    - **Monitoring**: Monitor the performance and usage patterns of your collection to identify potential bottlenecks or areas for optimization.
    - **Optimization**: Optimize your schema, indexes, and queries based on real-world usage and performance metrics to ensure optimal performance.

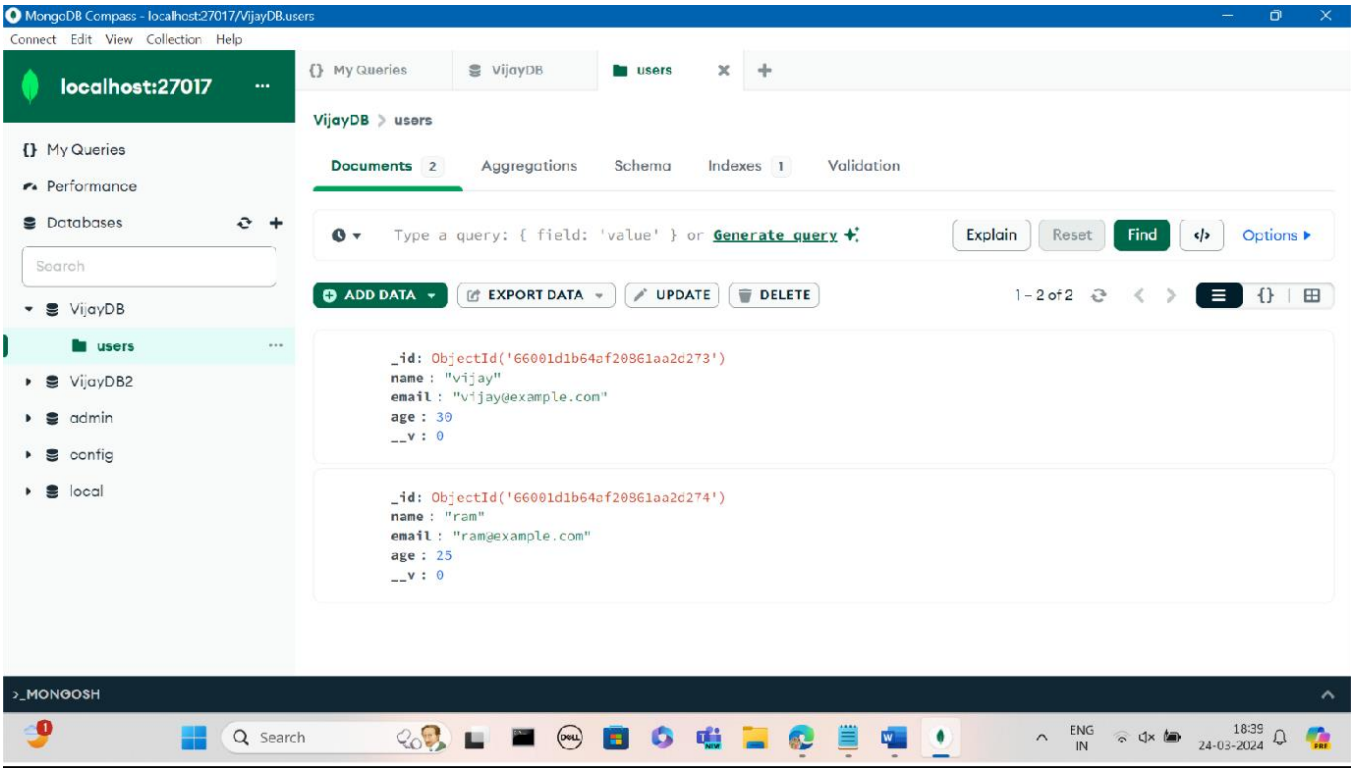
Server.js :

```
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');

const app = express();
const PORT = process.env.PORT || 5000;

// MongoDB connection
mongoose.connect('mongodb://localhost:27017/VijayDB', {
  useNewUrlParser: true,
  useUnifiedTopology: true
})
.then(() => console.log('MongoDB connected...'))
.catch(err => console.log(err));

// Define schema
const userSchema = new mongoose.Schema({
  name: String,
  email: String,
  age: Number
});

const User = mongoose.model('User', userSchema);

// Middleware
app.use(bodyParser.json());

// Insert static rows
const staticUsers = [
  { name: '**Vijay**', email: 'vijay@example.com', age: 20 },
  { name: '**Ram**', email: 'ram@example.com', age: 21 }
];

User.insertMany(staticUsers)
  .then(() => console.log('Documents inserted successfully...'))
  .catch(err => console.error('Error inserting static rows: ', err));

// API routes
app.post('/api/users', async (req, res) => {
  try {
    const { name, email, age } = req.body;
    const newUser = new User({ name, email, age });
    await newUser.save();
    res.status(201).json(newUser);
  } catch (error) {
    console.error('Error adding user:', error);
    res.status(500).json({ message: error.message });
  }
});

// Start server
app.listen(PORT, () => {
  console.log(`Server listening on port: ${PORT}`);
});
```

OUTPUT:

Creating a schema in MongoDB: Connecting to MongoDB





EXPERIMENT NO: 18

AIM:

Demonstrate CRUD operations using MongoDB.

DESCRIPTION:

CRUD operations stand for Create, Read, Update, and Delete, which are the basic operations for managing data in databases. Here's a brief explanation of each CRUD operation in the context of MongoDB:

CRUD Operations in MongoDB

1. **Create (C)**:
   - **Functionality**: The Create operation involves inserting new documents into a collection.
   - **MongoDB Method**: In MongoDB, you can use the `insertOne()` method to insert a single document and the `insertMany()` method to insert multiple documents into a collection.
   - **Example**: Inserting a new user document with fields like `name`, `email`, and `age` into a `users` collection.
2. **Read (R)**:
   - **Functionality**: The Read operation involves retrieving or querying documents from a collection.
   - **MongoDB Method**: MongoDB provides various methods like `find()`, `findOne()`, and `aggregate()` to query documents based on specific criteria, fetch all documents, or perform complex aggregations.
   - **Example**: Fetching all user documents from a `users` collection where `age` is greater than 25.
3. **Update (U)**:
   - **Functionality**: The Update operation involves modifying existing documents in a collection.
   - **MongoDB Method**: MongoDB offers methods like `updateOne()`, `updateMany()`, and `replaceOne()` to update documents based on specific criteria or replace an existing document entirely.
   - **Example**: Updating the `email` field of a user document with a specific `name` in a `users` collection.
4. **Delete (D)**:
   - **Functionality**: The Delete operation involves removing documents from a collection.
   - **MongoDB Method**: MongoDB provides methods like `deleteOne()` and `deleteMany()` to delete documents based on specific criteria or remove multiple documents at once.
   - **Example**: Deleting a user document with a specific `email` from a `users` collection.

Summary:

- **Create**: Inserting new documents into a collection using `insertOne()` or `insertMany()` methods.
- **Read**: Retrieving or querying documents from a collection using `find()`, `findOne()`, or `aggregate()` methods.
- **Update**: Modifying existing documents in a collection using `updateOne()`, `updateMany()`, or `replaceOne()` methods.
- **Delete**: Removing documents from a collection using `deleteOne()` or `deleteMany()` methods.

By understanding and implementing these CRUD operations, you can effectively manage and manipulate data in MongoDB collections, enabling you to build robust and scalable applications with MongoDB as the database backend.

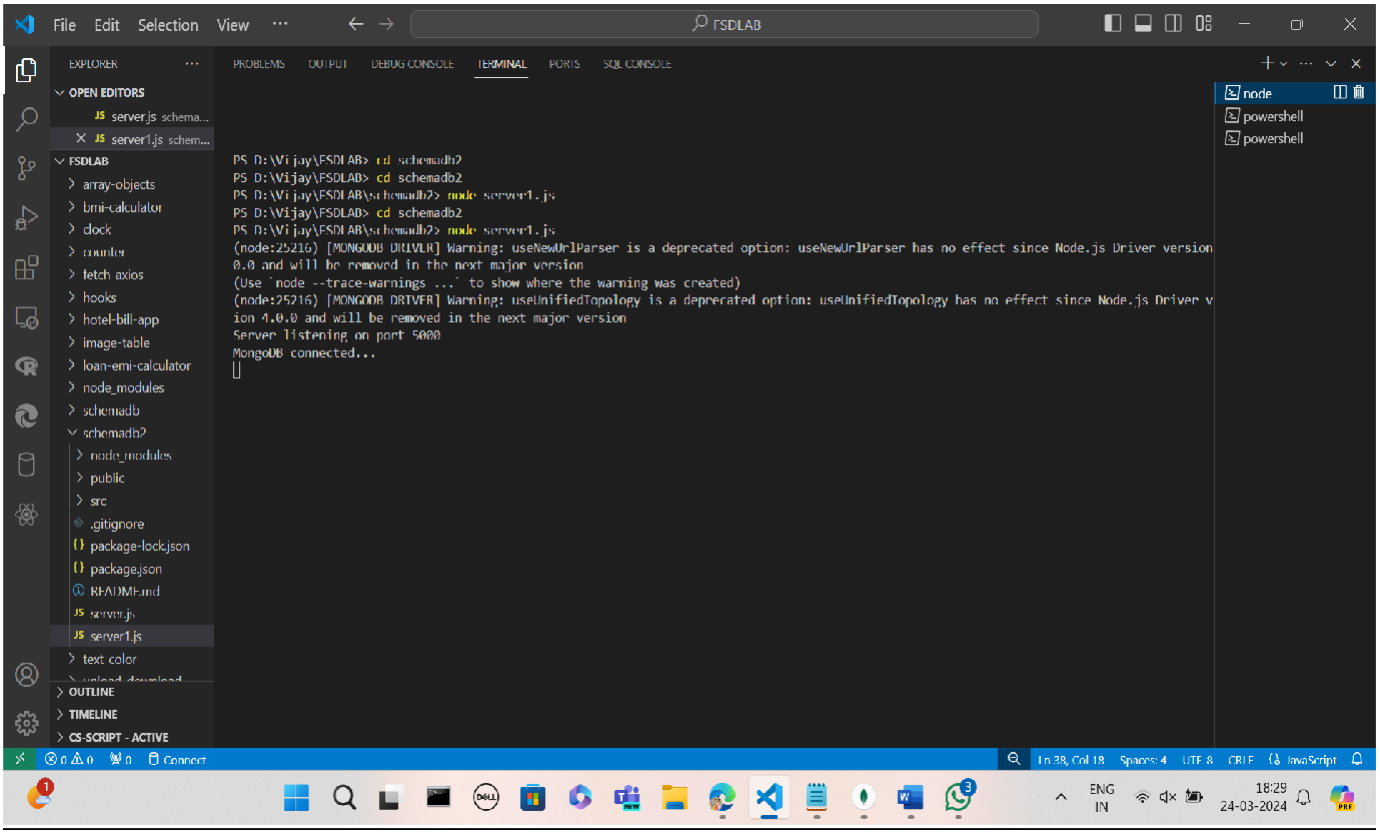
Server1.js :

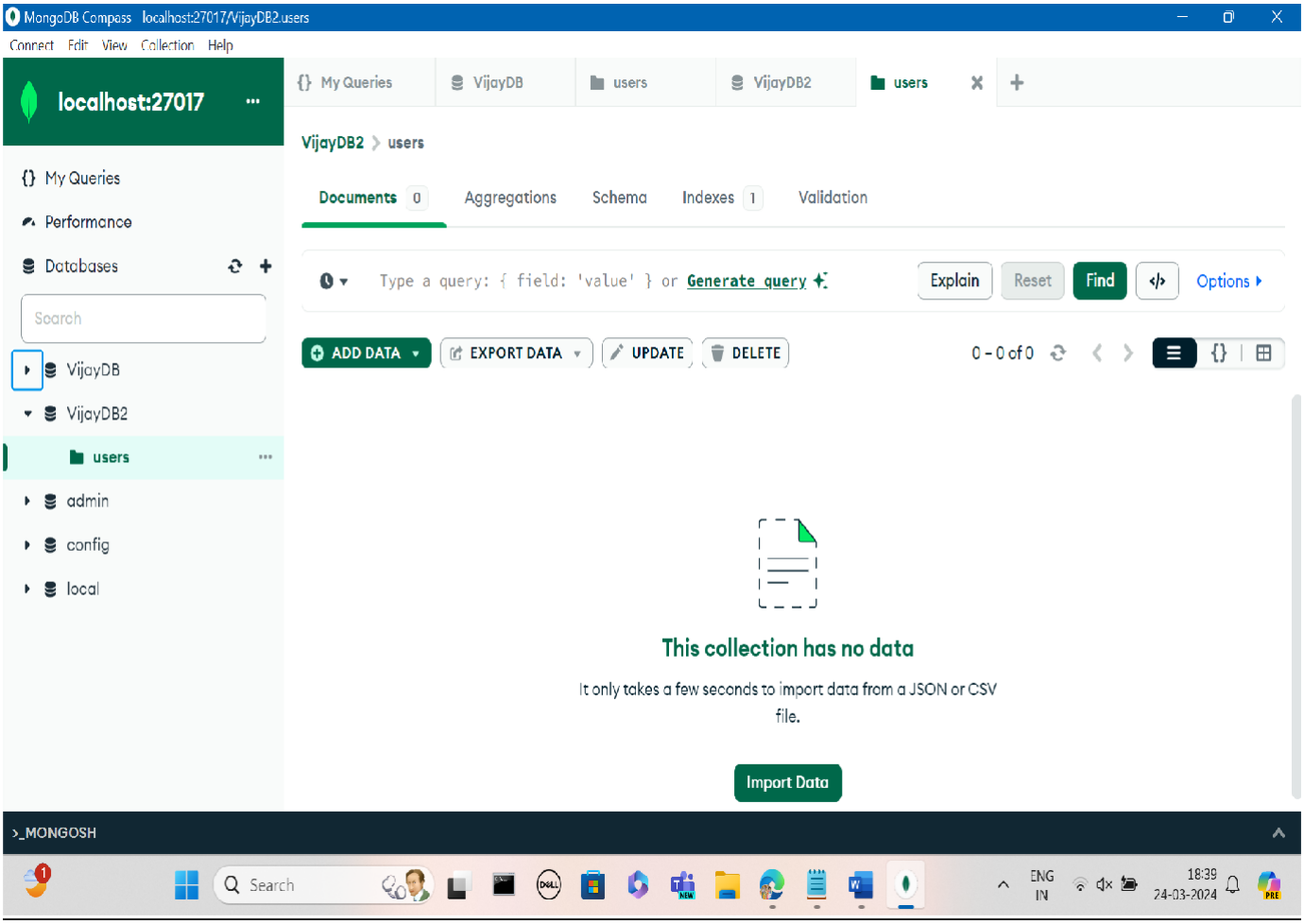
```
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
const { unstable_renderSubtreeIntoContainer } = require('react-dom');

const app = express();
const PORT = process.env.PORT || 5000;

// MongoDB connection
mongoose.connect('mongodb://localhost:27017/Mahesh', {
  useNewUrlParser: true,
  useUnifiedTopology: true
})
.then(() => console.log('MongoDB connected...'))
.catch(err => console.log(err));

// Define schema
const userSchema = new mongoose.Schema({
  name: String,
  email: String,
  age: Number
});

const User = mongoose.model('User', userSchema);

// Middleware
app.use(bodyParser.json());
//create user
app.post('/api/users',async(req,res)=>{
   try{
const {name,email,age} = req.body;
      const newUser = new User({name,email,age});
      await newUser.save();
      res.status(201).json(newUser);
   }
   catch(error){
      console.error('Error adding user:',error);
      res.status(500).json({message:error.message});
   }
});
app.get('/api/users',async(req,res) =>{
   try{
      const users = await User.find();
      res.json(users);
   }
   catch(error){
      console.error('Error fecthing users:',error);
      res.status(500).json({message:error.message});
   }
});
//get a user by ID
app.get('/api/users/:id',async(req,res)=>{
   try{
      const user = await User.findById(req.params.id);
      if(!user){
         return res.status(404).json({message:'User not found'});
      }
```

```
        res.json(user);
    }
    catch(error){
        console.error('Error fecthing users:',error);
        res.status(500).json({message:error.message});
    }
});
//update a user
app.put('/api/users/:id',async(req,res)=>{
    try{
        const {name,email,age} = req.body;
        const updatedUser = await User.findByIdAndUpdate(req.params.id,{name,email,age},{new:true});
        if(!updatedUser){
            return res.status(404).json({message : 'User not found'});
        }
        res.json(updatedUser);
    }
    catch(error){
        console.error('Error updating users:',error);
        res.status(500).json({message:error.message});
    }
});
//Delete a user
app.delete('/api/users/:id', async (req, res) => {
    try {
      const deletedUser = await User.findByIdAndDelete(req.params.id);
      if (!deletedUser) {
        return res.status(404).json({ message: 'User not found' });
      }
      res.json({ message: 'User deleted' });
    } catch (error) {
      console.error('Error deleting user:', error);
      res.status(500).json({ message: error.message });
    }
  });

//start server
app.listen(PORT,()=>{
    console.log(`Server listening on port ${PORT}`);
});
```

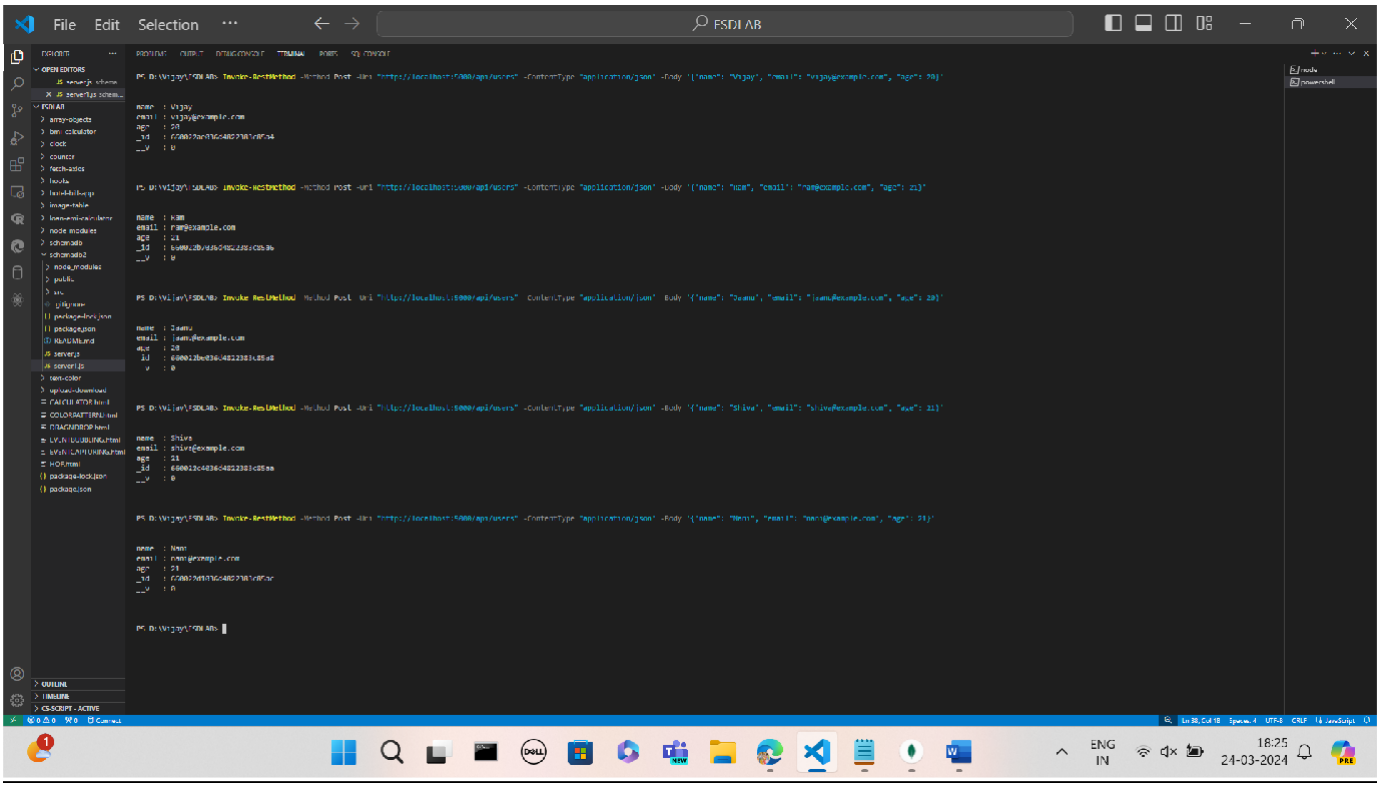
OUTPUT:

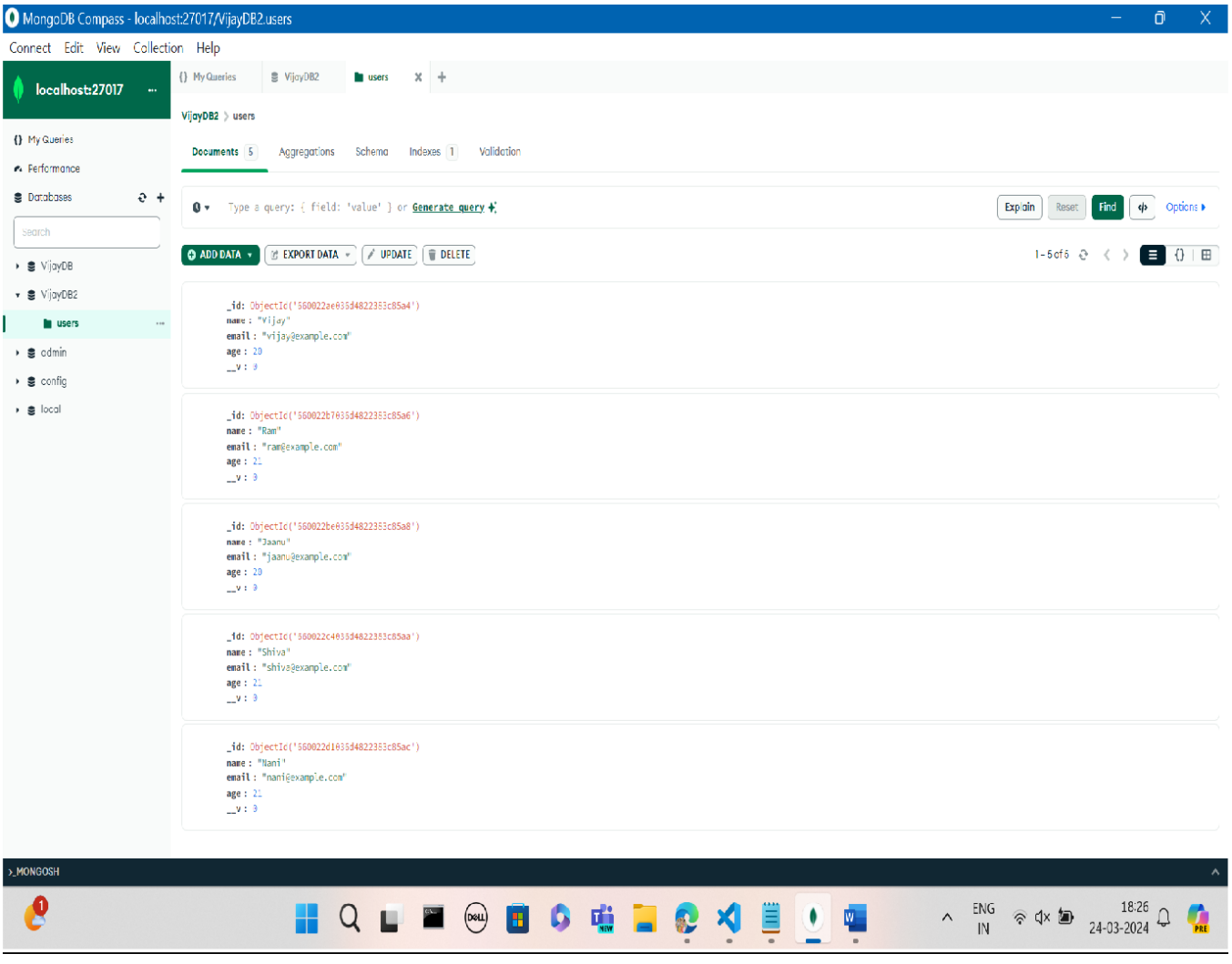
CRUD Operations using MongoDB: Connecting to MongoDB



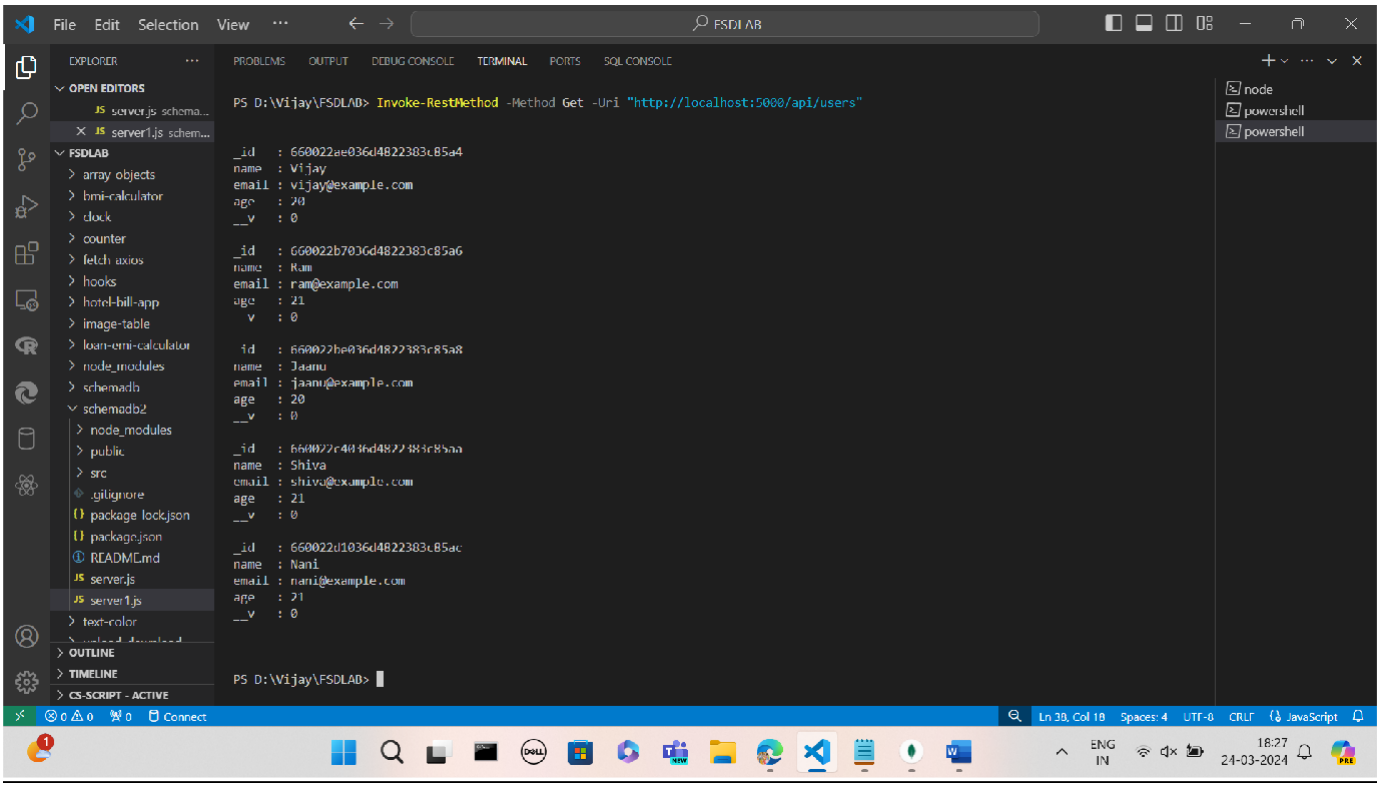


Inserting User Documents into MongoDB:

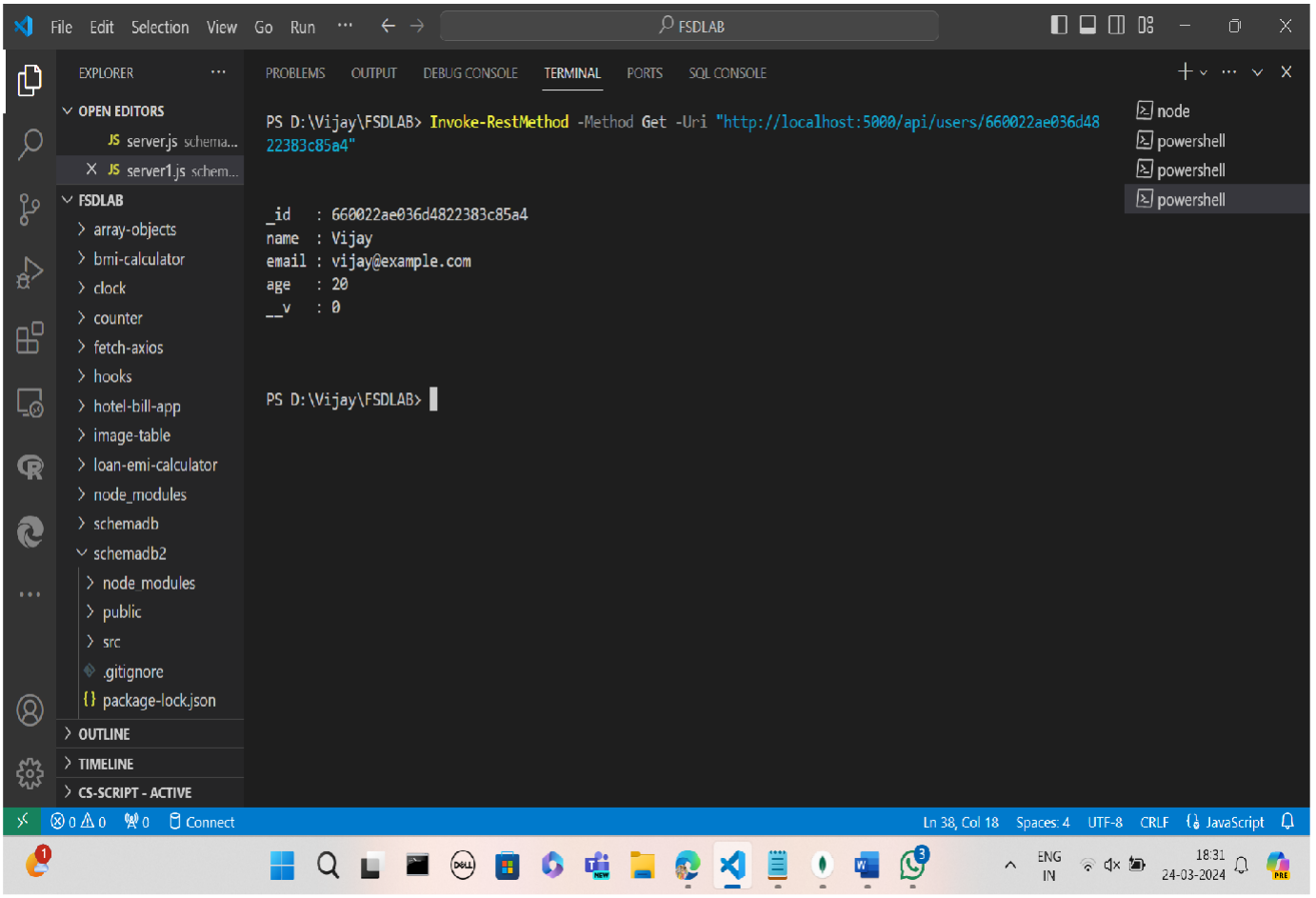




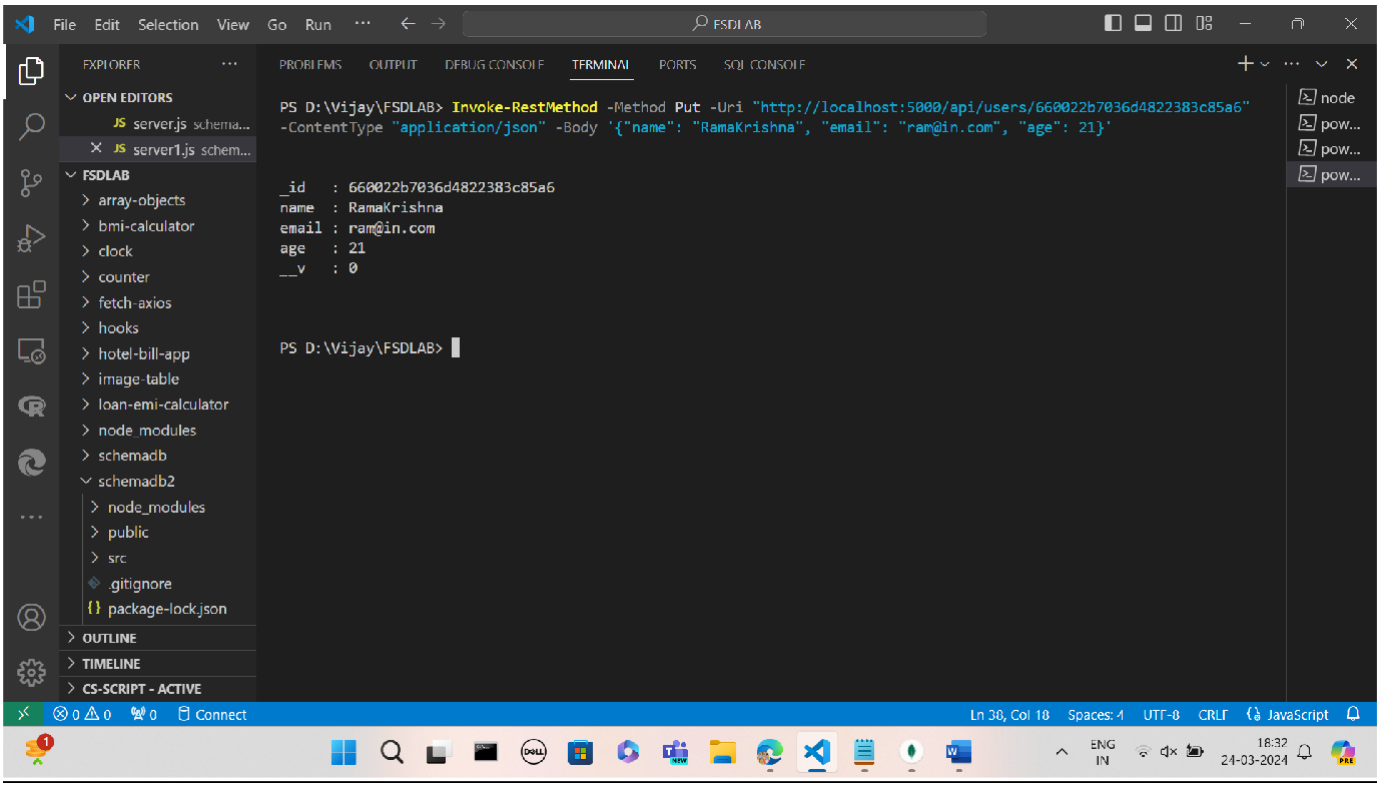
Getting all User Documents:

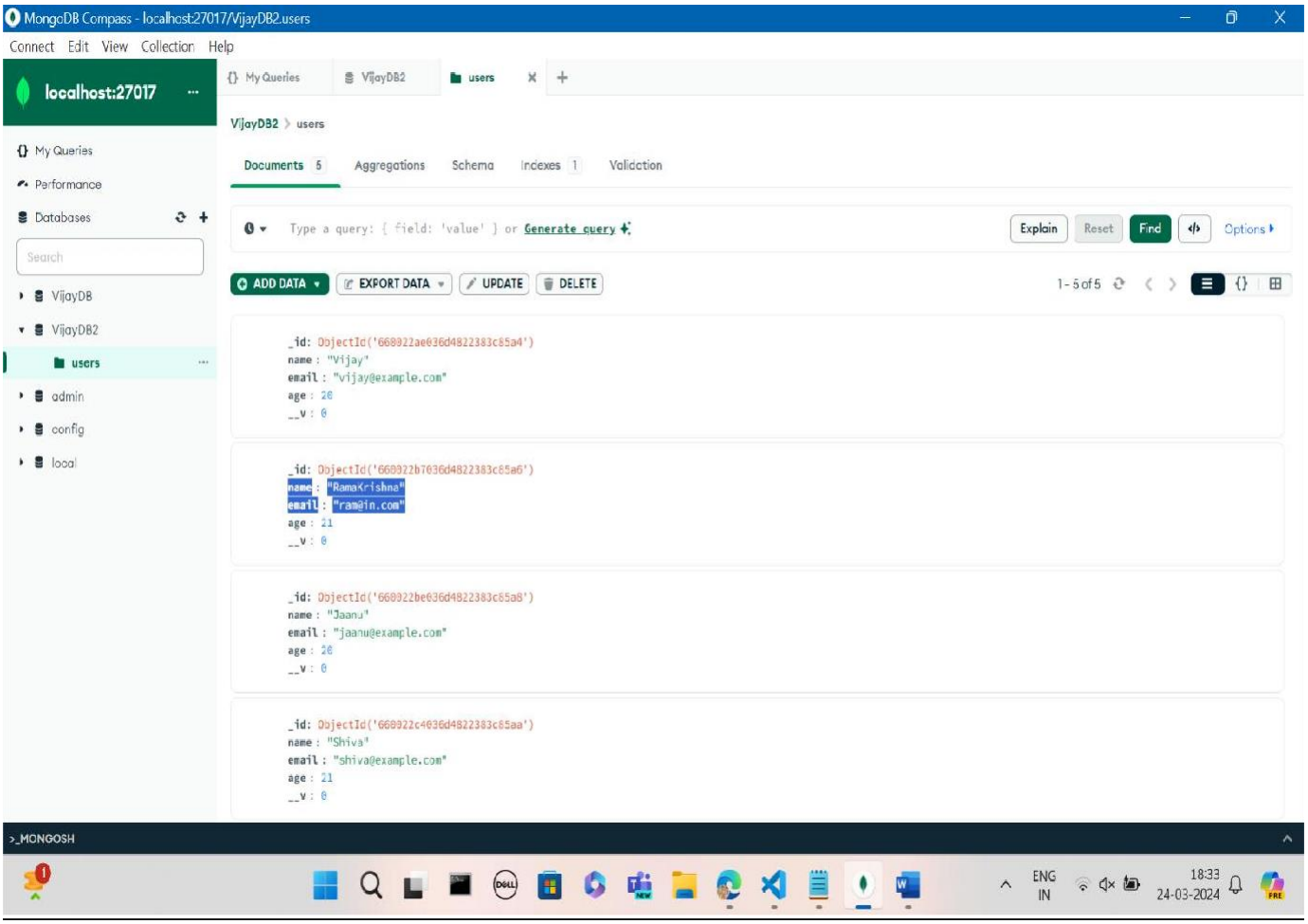


Retrieving a Specific User Document by using ID:



Updating Specific User Document By using ID:





Deleting a Specific User By using ID: