INDEX

INDEX

S.No.	Date	Name of the Experiment	Page No.	Marks	Remarks
3		3a) Write a JavaScript program to find the area of a circle using radius (var and let - reassign and observe the difference with var and let) and PI (const) 3b) Write JavaScript code to display the movie details such as movie name, starring, language, and ratings. Initialize the variables with values of appropriate types. Use template literals wherever necessary. 3c) Write JavaScript code to book movie tickets online and calculate the total price, considering the number of tickets and price per ticket as Rs. 150. Also, apply a festive season discount of 10% and calculate the discounted amount. 3d) Write a JavaScript code to make online booking of theatre tickets and calculate the total price based on the below conditions (use conditional statements): • If seats to be booked are not more than 2, the cost per ticket remains $9. • If seats are 5 or more, booking is not allowed. • If seats to be booked are more than 2 but less than 5, based on the number of seats booked, do the following: Calculate total cost by applying discounts of 5, 7, 9, 11 percent, and so on for customer 1,2,3 and 4. The output should be Ticket for customer 1 gets 5% discount, cost is:$8.54 Ticket for customer 2 gets 7% discount, cost is:$8.37 Ticket for customer 3 gets 9% discount, cost is:$8.19 Ticket for customer 4 gets 11% discount, cost is:$8.01 For 4 tickets, you need to pay:$33.12 instead of $36 3e) Repeat the experiment 3d using loops	**16-26**		
4		4a) Write a JavaScript code to do online booking of theatre tickets and calculate the total price based on the below conditions: 1. If seats to be booked are not more than 2, the cost per ticket remains $9. 2. If seats are 6 or more, booking is not allowed. 3. If seats to be booked are more than 2 but less than 5, based on the number of seats booked, do the following:	**27-39**		

		○ Calculate total cost by applying a discount of 5, 7, 9, 11 percent, and so on for customer 1,2,3 till 5. ○ Try the code with different values for the number of seats. Write the following custom functions to implement given requirements: 1. calculateCost(seats): Calculate and display the total cost to be paid by the customer for the tickets they have bought. calculateDiscount(seats): Calculate discount on the tickets bought by the customer. Implement using arrow functions 4b) Create an Employee class extending from a base class Person. 4c) Repeat Experiment 4a that include Event handling 4d) When the user clicks on the given link, they should see an empty cone, a different heading, and a different message and a different background color. When the user clicks again, they should see a re-filled cone, a different heading, a different message, and a different color in the background.			
5		5a) Create an array of objects having movie details. The object should include the movie name, starring, language, and ratings. Render the details of movies on the page using the array. 5b) Simulating a periodic stock price change and displaying on the console. 5c) Create a file login.js with a User class. Create a validate method with username and password as arguments. If the username and password are equal it will return "Login Successful" else will return "Unauthorized access". Create an index.html file with textboxes username and password and a submit button. Add a script tag in HTML to include index.js file. Create an index.js file which imports login module and invokes validate method of User class. On submit of the button in HTML the validate method of the User class should be invoked. Implement the validate method to send the username and password details entered by the user and capture the return value to display in the alert.	**40-45**		

<table>
<tr>
<td>6</td>
<td></td>
<td>6a) Verify how to execute different functions successfully in the Node.js platform.

6b) Create a web server in Node.js

6c) Write a Node.js module to show the workflow of Modularization of Node application.

6d) Write a program to show the workflow of restarting a Node application.

6e) Create a text file src.txt and add the following data to it. Mongo, Express, Angular, Node.</td>
<td>46-50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>7a) Implement routing for the AdventureTrails application as per the below requirement. Write the necessary code in the routes/route.js file.

<table>
<tr><th>Sl. No.</th><th>Route path</th><th>Success response</th><th>Error response</th></tr>
<tr><td>1</td><td>/packages</td><td>Status code: 200
A JSON object with property:
• message: "You can now get the requested packages for your request".</td><td>Status code: 404
A JSON object with property:
• status: "fail"
• message: the error message</td></tr>
<tr><td>2</td><td>/bookpackage</td><td>Status code: 201
A JSON object with property:
• message: "New booking added for the POST request"</td><td>Status code: 404
A JSON object with property:
• status: "fail"
• message: the error message</td></tr>
</table>

7b) In myNotes application: Illustrating the usage of middlewares by performing
(i) handling POST submissions. (ii) displaying customized error messages. (iii) perform logging.

7c &7d) Connecting to MongoDB with Mongoose, Validation Types and Defaults. Write a Mongoose schema to connect with MongoDB. Write a program to wrap the Schema into a Model object.</td>
<td>51-58</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td><u>8a & 8b)</u> Write a program to perform various CRUD (Create-Read-Update-Delete) operations using Mongoose library functions. Include APIs (API should fetch the details, API should update the details, API should delete the details)

8c) Write a program to explain session management using cookies.</td>
<td>59-71</td>
<td></td>
<td></td>
</tr>
</table>

		8d) Write a program to explain session management using sessions 8e) Implement security features using Helmet in the application			
9		9a) On the page, display the price of the mobile-based in three different colors. Instead of using the number in our code, represent them by string values like GoldPlatinum, PinkGold, SilverTitanium. 9b) Define an arrow function inside the event handler to filter the product array with the selected product object using the productId received by the function. 9c) Consider that developer needs to declare a function - getMobileByManufacturer which accepts a string as an input parameter and returns the list of mobiles. 9d) Consider that developer needs to declare a manufacturer's array holding 4 objects with id and price as a parameter. The developer needs to implement an arrow function - myfunction to populate the id parameter of manufacturers array whose price is greater than or equal to 200 dollars. 9e) Consider that developer needs to declare a function - getMobileByManufacturer with two parameters namely manufacturer and id, where manufacturer value should pass as Samsung and the id parameter should be optional while invoking the function. If the id is passed as 101 then this function should return Moto mobile list. If manufacturer parameter is either Samsung/Apple then this function should return the respective mobile list, similar to making Samsung as default Manufacturer.	**72-81**		
10		10a) Consider that developer needs to implement business logic for adding multiple Product values into a cart variable which is type of string array. 10b) Declare an interface named - Product with two properties like productId and productName with a number and string datatype. The developer needs to implement logic to populate the Product details using this interface.	**82-85**		

		10c) Declare an interface named - Product with two properties like productId and productName with the number and string datatype and need to implement logic to populate the Product details. **Use Duck Typing** 10d) Declare an interface with function type and access its value.			
11		11a) Declare a productList interface which extends properties from two other declared interfaces like Category, Product as well as implementation to create a variable of this interface type. 11b) Create objects of the Product class and place them into the productlist array. 11c) Declare a class named - Product with the below-mentioned declarations: (i) productId as number property (ii) Constructor to initialize this value (iii) getProductId method to return the message "Product id is <>". 11d) create a Product class with 4 properties namely productId, productName, productPrice, productCategory with private, public, static, and protected access modifiers and accessing them through Gadget class and its methods.	**86-90**	**46**	
12		12a) Create a Product class with properties namely productId and methods to setProductId() and getProductId() 12b) Create a namespace called ProductUtility and place the Product class definition in it. Import the Product class inside productlist file and use it. 12c) Design a TypeScript module for a Mobile Cart application that includes a function to determine the total cost of a product using the quantity and price values and assign it to a totalPrice variable. 12d) Create a generic array and function to sort numbers as well as string values	**91-95**		

Experiment 1:

Aim:

Create a website for Online shopping store say IEKart

Include the Metadata element in Homepage.html for providing description as "IEKart's is an online shopping website that sells goods in retail. This company deals with various categories like Electronics, Clothing, Accessories etc.

Enhance the Homepage.html of IEKart's Shopping Application by adding appropriate sectioning elements.

Make use of appropriate grouping elements such as list items to "About Us" page of IEKart's Shopping Application.

Link "Login", "SignUp" and "Track order" to "Login.html", "SignUp.html" and "Track.html" page respectively.

Add the © symbol in the Home page footer of IEKart's Shopping application.

Add the global attributes such as contenteditable, spellcheck, id etc. to enhance the Signup Page functionality of IEKart's Shopping application.

Program:

Homepage.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <meta name="description" content="IEKart's is an online shopping website that sells goods in retail. This company deals with various categories like Electronics, Clothing, Accessories etc.">
   <title>IEKart - Online Shopping</title>
   <link rel="stylesheet" href="styles.css">
</head>
<body>
   <header>
      <h1>Welcome to IEKart</h1>
   </header>
   <nav>
      <a href="homepage.html">Home</a>
      <a href="Aboutus.html">About Us</a>
      <a href="Login.html">Login</a>
      <a href="SignUp.html">Sign Up</a>
      <a href="Track.html">Track Order</a>
   </nav>
   <main class="content">
      <div class="sidebar">
         <h3>Electronics</h3>
```

```
        <ul>
          <li>Mobile Phones</li>
          <li>Laptops</li>
          <li>Televisions</li>
          <li>Headphones</li>
          <li>Cameras</li>
        </ul>
      </div>

      <div class="maincontent">
        <section>
          <h2>Shop the Best Categories</h2>
                    <p>Explore our wide range of categories including <span class="highlight">Electronics</span>, <span class="highlight">Clothing</span>, and <span class="highlight">Accessories</span>.</p>
        </section>
        <section>
          <h2>Why Choose IEKart?</h2>
          <ul>
            <li>Wide variety of products</li>
            <li>Affordable prices</li>
            <li>Fast delivery</li>
            <li>Secure payment options</li>
          </ul>
        </section>
      </div>
    </main>
    <footer>
      <a href="#">About Us</a> |
      <a href="/aboutus.html">Privacy Policy</a> |
      <a href="#">Contact Us</a> |
      <a href="#">FAQ</a> |
      <a href="#">Terms & Conditions</a>

      <p>&copy; Copyright 2024 | IEKart Electronics</p>
        <p>Like and Connect with us <a href="#">FB</a> <a href="#">Twitter</a> <a href="#">g+</a></p>
    </footer>
</body>
</html>
```

Aboutus.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <meta name="description" content="IEKart's is an online shopping website that sells goods in retail. This company deals with various categories like Electronics, Clothing, Accessories etc.">
    <title>IEKart - Online Shopping</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <header>
        <h1>Welcome to IEKart</h1>
    </header>
    <nav>
        <a href="homepage.html">Home</a>
        <a href="Aboutus.html">About Us</a>
        <a href="Login.html">Login</a>
        <a href="SignUp.html">Sign Up</a>
        <a href="Track.html">Track Order</a>
    </nav>
    <main class="content">
      <div class="maincontent">
        <section >
          <h1>About Us</h1>
          <p>
            IEKart's very own one-stop solution for all your shopping needs . We are committed to quality, innovation, customer satisfaction.
          </p>

          <h2>Our Vision</h2>
          <p>
            To be the industry leader by revolutionizing the industry, setting new standards.
          </p>

          <h2>Our Mission</h2>
          <p>
            To setting new standards by delivering exceptional products, providing unparalleled customer service.
          </p>
        </section>

        <section>
          <h3>Our Products:</h3>
          <ul>
            <li>Electronics</li>
            <li>Clothing</li>
            <li>Furniture</li>
            <li>Decorative Items</li>

          </ul>
        </section>
      </div>
  </main>
```

```
  <footer>
    <a href="#">About Us</a> |
    <a href="/aboutus.html">Privacy Policy</a> |
    <a href="#">Contact Us</a> |
    <a href="#">FAQ</a> |
    <a href="#">Terms & Conditions</a>

    <p>&copy; Copyright 2024 | IEKart Electronics</p>
      <p>Like and Connect with us <a href="#">FB</a> <a href="#">Twitter</a> <a href="#">g+</a></p>
  </footer>
</body>
</html>
```

Login.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <meta name="description" content="IEKart's is an online shopping website that sells goods in retail. This company deals with various categories like Electronics, Clothing, Accessories etc.">
  <title>IEKart - Online Shopping</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <header>
    <h1>Welcome to IEKart</h1>
  </header>
  <nav>
    <a href="homepage.html">Home</a>
    <a href="AboutUs.html">About Us</a>
    <a href="Login.html">Login</a>
    <a href="SignUp.html">Sign Up</a>
    <a href="Track.html">Track Order</a>
  </nav>
  <h2>Login to Your Account</h2>
 <form action="/login" method="post"> <label for="username">Username:</label>
  <input type="text" id="username" name="username" required><br><br>

  <label for="password">Password:</label>
  <input type="password" id="password" name="password" required><br><br>

  <input type="submit" value="Login">
 </form>
 <p>Don't have an account? <a href="SignUp.html">Sign Up</a></p>
```

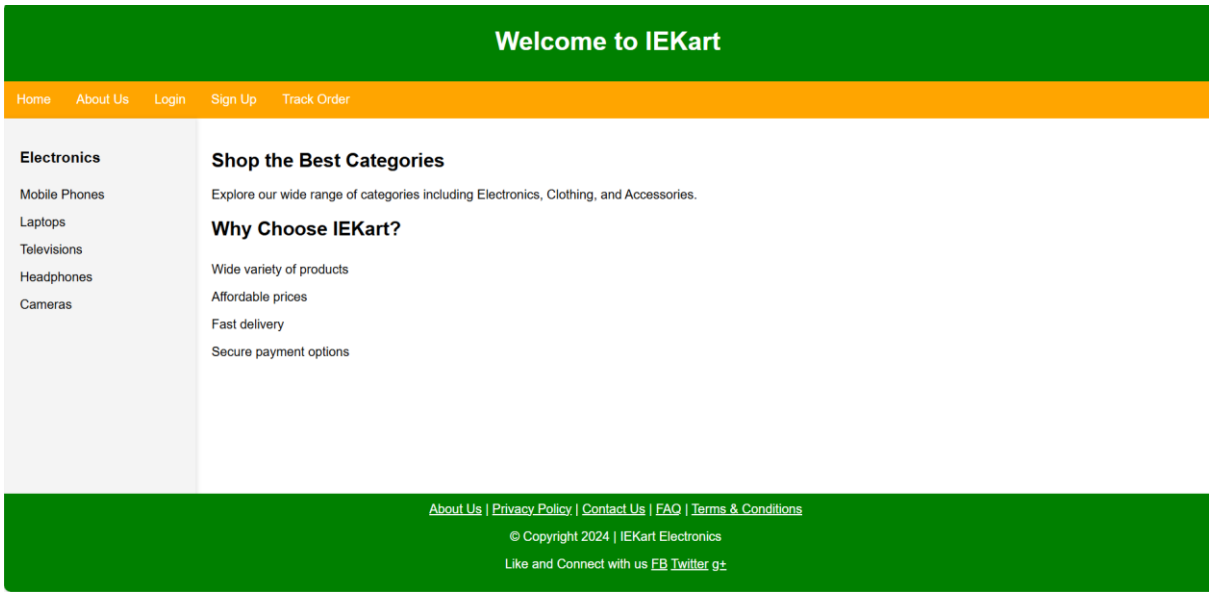
```
    <footer>
        <a href="#">About Us</a> |
        <a href="/aboutus.html">Privacy Policy</a> |
        <a href="#">Contact Us</a> |
        <a href="#">FAQ</a> |
        <a href="#">Terms & Conditions</a>

        <p>&copy; Copyright 2024 | IEKart Electronics</p>
          <p>Like and Connect with us <a href="#">FB</a> <a href="#">Twitter</a> <a
href="#">g+</a></p>
    </footer>
</body>
</html>
```

Styles.css

```
body {
    margin: 0;
    font-family: Arial, sans-serif;
    display: flex; /* Use flexbox for the entire body */
    flex-direction: column; /* Set main axis to vertical */
}

header {
    background-color: green;
    padding: 10px;
    text-align: center;
    color: white;
}

nav {
    background-color: orange;

}

nav a {
    float: left;
    display:block;
    color: white;
    text-align: center;
    padding: 14px 16px;
    text-decoration: none;
}

.content {
    display: flex; /* Use flexbox for main content */
    flex-grow: 1; /* Allow main content to grow to fill remaining space */
}
.sidebar {
    width: 200px;
```

```
    background-color: #f4f4f4;
    padding: 20px;
    box-shadow: 2px 0 5px rgba(0,0,0,0.1);
    height: calc(100vh - 140px); /* Adjust this value as needed */
}
.maincontent {
    flex: 1; /* Allow main content to grow to fill remaining space */
    padding: 20px;
}

ul{
    list-style-type: none;
    padding: 0;
}

ul li{
    padding: 8px 0;
}
footer {
    background-color: green;
    color: white;
    text-align: center;
    padding: 10px;
    position: fixed;
    bottom: 0;
    width: 100%;
}

footer a{
    color:white;
}
```

Output:



Experiment 2:

Aim:

Enhance the details page of IEKart's Shopping application by adding a table element to display the available mobile/any inventories.

Mobile Inventories Available at IEKart's Shopping!

Product Name	Product Details	
	Price	Description
Asus Zenfone	11599	An Economical Phone by Asus
Redmi Note2	8599	An Economical Phone by Xiaomi
Moto G Turbo	10599	An Economical Phone by Moto

Using the form elements create Signup page for IEKart's Shopping application.

Enhance Signup page functionality of IEKart's Shopping application by adding attributes to input elements.

Add media content in a frame using audio, video, iframe elements to the Home page of IEKart's Shopping application

Program:

Details.html

```
<!DOCTYPE html>
<html>
   <head>
      <title>IEKart's Shopping</title>
      <meta name="keywords" content="Online, Shopping,Electronics,Books,Cloths" />
       <meta name="description" content="IEKart's is an online shopping website that sells goods in retail. This company deals
         with various categories like Electronics, Clothing, Accessories etc.">
      <link rel="stylesheet" href="styles.css">
   </head>
   <body>
      <header>
         <h1>Welcome to IEKart</h1>
      </header>
      <nav>
         <a href="homepage.html">Home</a>
         <a href="Aboutus.html">About Us</a>
         <a href="Login.html">Login</a>
         <a href="SignUp.html">Sign Up</a>
         <a href="Track.html">Track Order</a>
      </nav>
      <main class="content">
         <div class="sidebar">
```

```
            <h3>Electronics</h3>
            <ul>
              <li>Mobile Phones</li>
              <li>Laptops</li>
              <li>Televisions</li>
              <li>Headphones</li>
              <li>Cameras</li>
            </ul>
          </div>

          <div class="maincontent">
            <section>
              <!-- Design the table here -->
              <table border=""1px solid red">
              <caption>Mobile Inventories Available at IEKart's Shopping!</caption>
              <thead>
                 <tr>
                    <th rowspan="2">Product Name</th>
                    <th colspan="2">Product Details</th>
                 </tr>
                <tr>
                   <th>Price</th>
                   <th>Description</th>
                </tr>
              </thead>
              <tbody>
                <tr>
                    <td>Asus Zenfone</td>
                    <td>11599</td>
                    <td>An Economical Phone by Asus</td>
                </tr>
                <tr>
                   <td>Redmi Note2</td>
                   <td>8599</td>
                   <td>An Economical Phone by Xiaomi</td>
                </tr>

                <tr>
                   <td>Moto G Turbo</td>
                   <td>10599</td>
                   <td>An Economical Phone by Moto</td>
                </tr>
              </tbody>
              </table>
            </section>
          </div>
        </main>
        <footer>
          <a href="#">About Us</a> |
          <a href="/aboutus.html">Privacy Policy</a> |
```

```
        <a href="#">Contact Us</a> |
        <a href="#">FAQ</a> |
        <a href="#">Terms & Conditions</a>

        <p>&copy; Copyright 2024 | IEKart Electronics</p>
          <p>Like and Connect with us <a href="#">FB</a> <a href="#">Twitter</a> <a
href="#">g+</a></p>
      </footer>

  </body>
 </html>
```

Signup.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="keywords" content="Online, Shopping">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>IECART's Shopping</title>
   <style>
      body {
         font-family: Arial, sans-serif;
      }
      form {
         max-width: 500px;
         margin: auto;
         padding: 20px;
         border: 1px solid #ddd;
         border-radius: 5px;
         box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
      }
      label {
         display: block;
         margin-bottom: 5px;
         font-weight: bold;
      }
      input, select, textarea {
         width: 100%;
         padding: 8px;
         margin-bottom: 15px;
         border: 1px solid #ddd;
         border-radius: 3px;
      }
      .radio-group, .checkbox-group {
         display: flex;
         justify-content: space-between;
         margin-bottom: 15px;
      }
```

```
      .radio-group label, .checkbox-group label {
         display: inline-block;
         margin-right: 10px;
      }
      button {
         padding: 10px 15px;
         color: #fff;
         background-color: #007BFF;
         border: none;
         border-radius: 3px;
         cursor: pointer;
      }
      button[type="reset"] {
         background-color: #6c757d;
      }
      button:hover {
         opacity: 0.9;
      }
      footer {
         text-align: center;
         margin-top: 20px;
         font-size: 0.9em;
      }
      footer nav a {
         margin: 0 5px;
         text-decoration: none;
      }
   </style>

</head>
<body>
   <header>
      <h1 style="text-align: center;">Sign Up!</h1>
   </header>
   <main>
      <form action="/success" method="get">
         <label for="username">Username:</label>
           <input type="text" id="username" placeholder="Enter your username" required
spellcheck="true" contenteditable>

         <label for="email_id">Email ID:</label>
           <input type="email" id="email_id" placeholder="Enter your email ID" required
spellcheck="true">

         <label for="password">Password:</label>
             <input type="password" id="password" placeholder="Enter your password"
required>

         <label>Gender:</label>
         <div class="radio-group">
```

```
            <label for="male"><input type="radio" id="male" name="gender" value="M">
Male</label>
            <label for="female"><input type="radio" id="female" name="gender" value="F">
Female</label>
        </div>

        <label for="dob">DOB:</label>
        <input type="date" id="dob" required>

        <label for="phone_no">Phone number:</label>
            <input type="tel" id="phone_no" placeholder="Enter your contact number"
pattern="\+?\d{10,15}" required spellcheck="false">

        <label for="country">Country:</label>
        <select id="country" placeholder="Select your country">
          <option value="India">India</option>
          <option value="USA">USA</option>
          <option value="UK">UK</option>
          <option value="Canada">Canada</option>
          <option value="Belgium">Belgium</option>
          <option value="France">France</option>
        </select>

        <label>Languages Known:</label>
        <div class="checkbox-group">
            <label for="english"><input type="checkbox" name="language" id="english"
value="English" checked> English</label>
              <label for="hindi"><input type="checkbox" name="language" id="hindi"
value="Hindi"> Hindi</label>
             <label for="french"><input type="checkbox" name="language" id="french"
value="French"> French</label>
        </div>

        <label for="pic">Profile pic:</label>
        <input type="file" id="pic" required>

        <label for="yourself" dir="ltr">About yourself:</label>
            <textarea id="yourself" placeholder="Tell us about yourself" contenteditable
spellcheck="true"></textarea>

        <div>
          <button type="submit">Register</button>
          <button type="reset">Reset</button>
        </div>
      </form>
    </main>
    <footer>
      <nav>About Us | Privacy Policy | Contact Us | FAQ | Terms & Conditions</nav>
      <p>&copy; 2018 | IEKart Electronics</p>
    </footer>
```

```
    <aside>
        <p>Like and Connect with us:</p>
        <a href="#">FB</a> | <a href="#">Twitter</a> | <a href="#">G+</a>
    </aside>
</body>
</html>
```

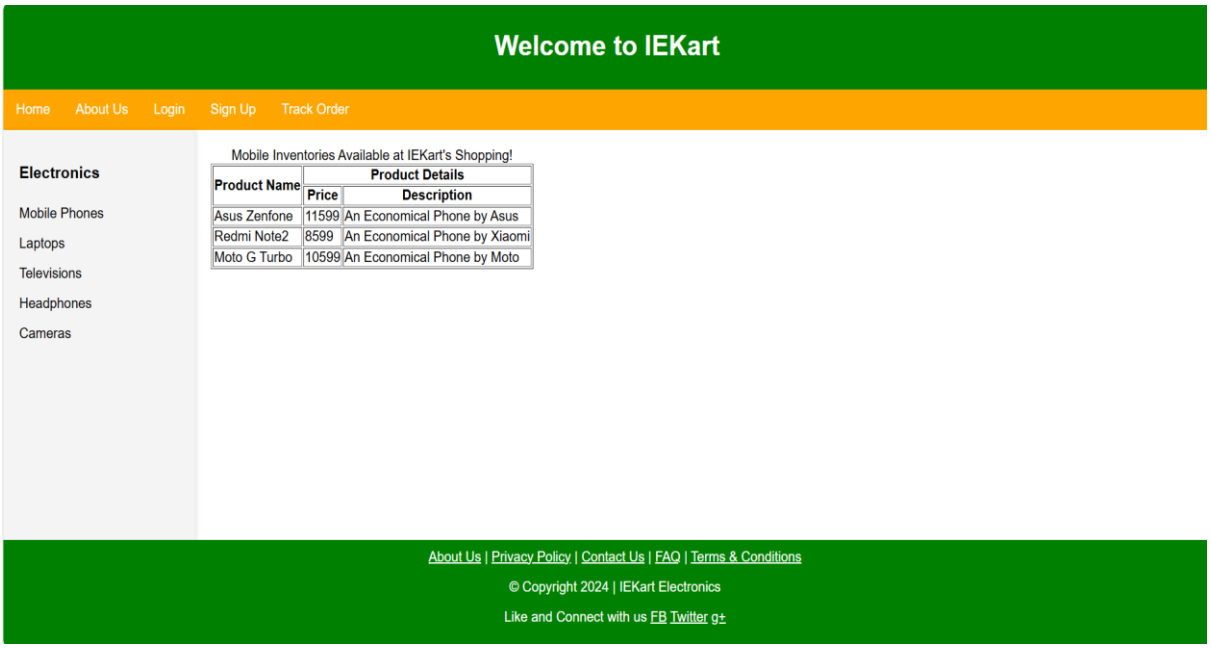
Homepageextension.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
     <meta name="description" content="IEKart's is an online shopping website that sells goods in retail. This company deals with various categories like Electronics, Clothing, Accessories etc.">
    <title>IEKart - Online Shopping</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <header>
        <h1>Welcome to IEKart</h1>
    </header>
    <nav>
        <a href="homepage.html">Home</a>
        <a href="Aboutus.html">About Us</a>
        <a href="Login.html">Login</a>
        <a href="SignUp.html">Sign Up</a>
        <a href="Track.html">Track Order</a>
    </nav>
    <main class="content">
        <div class="sidebar">
            <h3>Electronics</h3>
            <ul>
                <li>Mobile Phones</li>
                <li>Laptops</li>
                <li>Televisions</li>
                <li>Headphones</li>
                <li>Cameras</li>
            </ul>
        </div>

        <div class="maincontent">
            <section>
                <h2>Shop the Best Categories</h2>
```

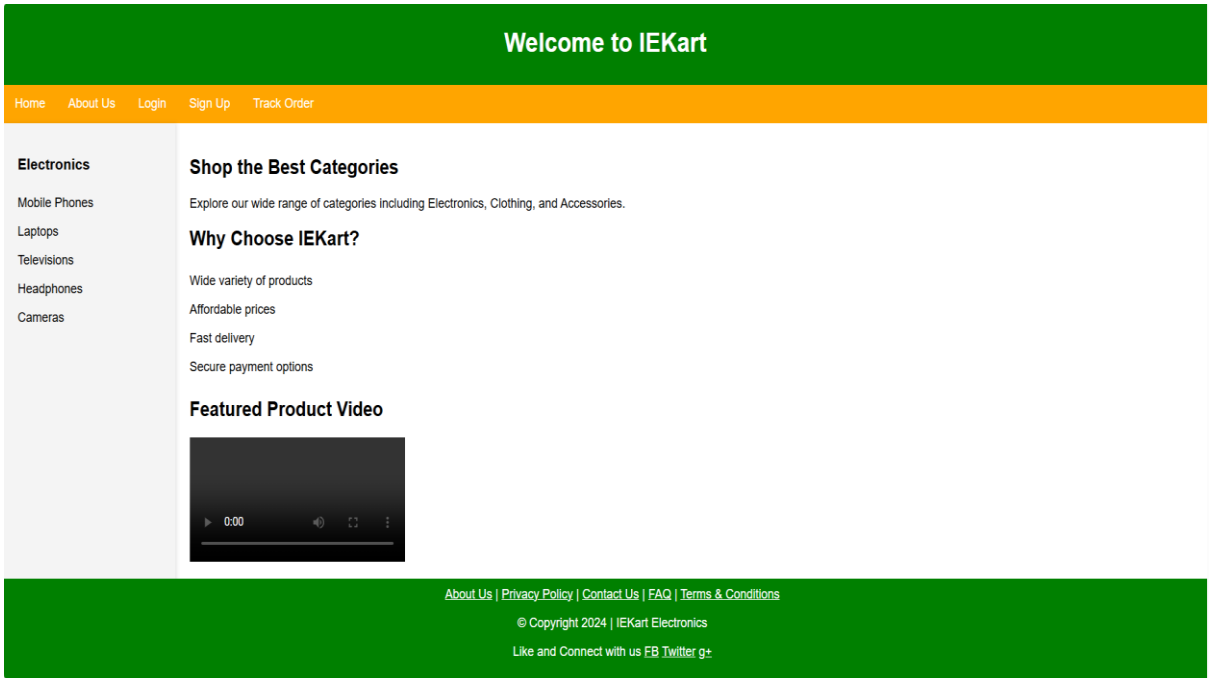
```
                    <p>Explore our wide range of categories including <span class="highlight">Electronics</span>, <span class="highlight">Clothing</span>, and <span class="highlight">Accessories</span>.</p>
        </section>
        <section>
            <h2>Why Choose IEKart?</h2>
            <ul>
                <li>Wide variety of products</li>
                <li>Affordable prices</li>
                <li>Fast delivery</li>
                <li>Secure payment options</li>
            </ul>
        </section>
        <section>
            <h2>Featured Product Video</h2>
            <video controls>
              <source src="path/to/your/product_video.mp4" type="video/mp4">
              Your browser does not support the video element.
            </video>
          </section>
      </div>
    </main>


    <footer>
        <a href="#">About Us</a> |
        <a href="/aboutus.html">Privacy Policy</a> |
        <a href="#">Contact Us</a> |
        <a href="#">FAQ</a> |
        <a href="#">Terms & Conditions</a>

        <p>&copy; Copyright 2024 | IEKart Electronics</p>

</body>
</html>
```

Output:

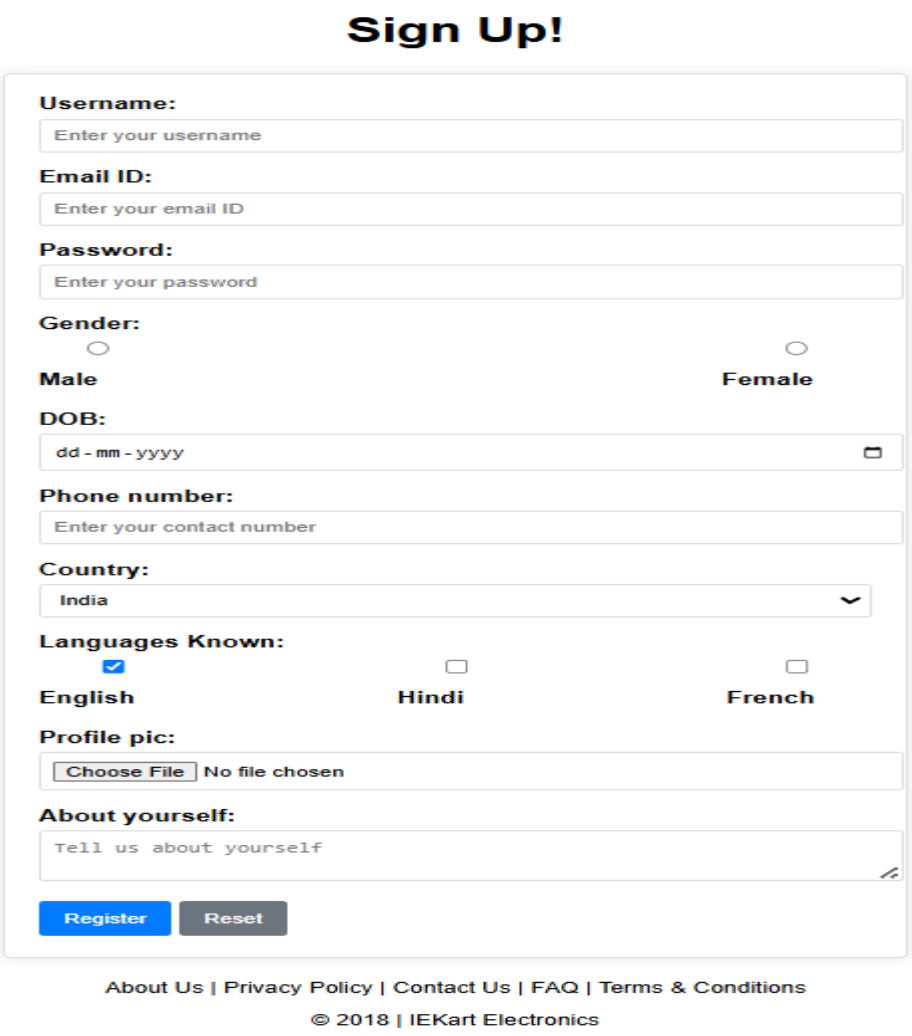
Details page



Homepagextension output:



Signup Page


Sign Up!
Username:
Enter your username
Email ID:
Enter your email ID
Password:
Enter your password
Gender:
Male
Female
DOB:
dd - mm - yyyy
Phone number:
Enter your contact number
Country:
India
Languages Known:
English
Hindi
French
Profile pic:
Choose File
No file chosen
About yourself:
Tell us about yourself
Register
Reset
About Us | Privacy Policy | Contact Us | FAQ | Terms & Conditions
© 2018 | IEKart Electronics

Experiment 3:

Aim:

a) Write a JavaScript program to find the area of a circle using radius (var and let - reassign and observe the difference with var and let) and PI (const)

b) Write JavaScript code to display the movie details such as movie name, starring, language, and ratings. Initialize the variables with values of appropriate types. Use template literals wherever necessary.

c) Write JavaScript code to book movie tickets online and calculate the total price, considering the number of tickets and price per ticket as Rs. 150. Also, apply a festive season discount of 10% and calculate the discounted amount.

d) Write a JavaScript code to make online booking of theatre tickets and calculate the total price based on the below conditions (use conditional statements):

- If seats to be booked are not more than 2, the cost per ticket remains $9.
- If seats are 5 or more, booking is not allowed.
- If seats to be booked are more than 2 but less than 5, based on the number of seats booked, do the following:
  Calculate total cost by applying discounts of 5, 7, 9, 11 percent, and so on for customer 1,2,3 and 4.

The output should be

Ticket for customer 1 gets 5% discount, cost is:$8.54

Ticket for customer 2 gets 7% discount, cost is:$8.37

Ticket for customer 3 gets 9% discount, cost is:$8.19

Ticket for customer 4 gets 11% discount, cost is:$8.01

For 4 tickets, you need to pay:$33.12 instead of $36

e. Repeat the experiment 3d using loops

Program:

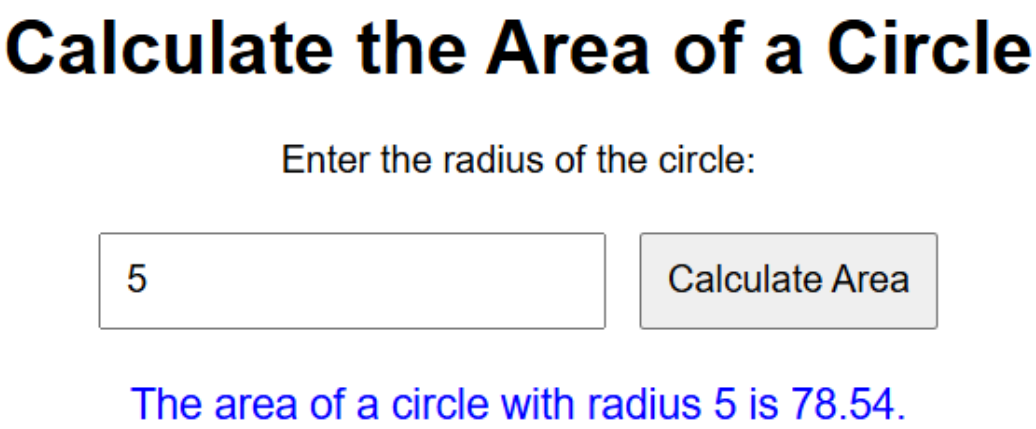
Exp3a)

```
/*
Write a JavaScript program to find the area of a circle using radius (var and let -
reassign and observe the difference with var and let) and PI (const) */

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

```
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Circle Area Calculator</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            text-align: center;
            margin-top: 50px;
        }
        input, button {
            padding: 10px;
            margin: 5px;
            font-size: 16px;
        }
    </style>
</head>
<body>
    <h1>Calculate the Area of a Circle</h1>
    <p>Enter the radius of the circle:</p>
    <input type="number" id="radius" placeholder="Enter radius" >
    <button onclick="calculateAndDisplayArea()">Calculate Area</button>
    <p id="result" style="font-size: 18px; color: blue;"></p>

    <script>
        // Declare the constant PI
        const PI = 3.14159;

        // Function to calculate the area of a circle
        function calculateArea(radius) {
            if (radius < 0) {
                return "Radius cannot be negative.";
            }
            return PI * radius * radius;
        }

        // Function to get input, calculate, and display the area
        function calculateAndDisplayArea() {
            let radiusInput = document.getElementById("radius").value;
            let radius = parseFloat(radiusInput);

            if (isNaN(radius)) {
                document.getElementById("result").textContent = "Please enter a valid number
for radius.";
                return;
            }

            let area = calculateArea(radius);
            if (typeof area === "string") {
                document.getElementById("result").textContent = area;
            } else {
```

```
            document.getElementById("result").textContent = `The area of a circle with radius ${radius} is ${area.toFixed(2)}.`;
        }
    }
  </script>
</body>
</html>
```

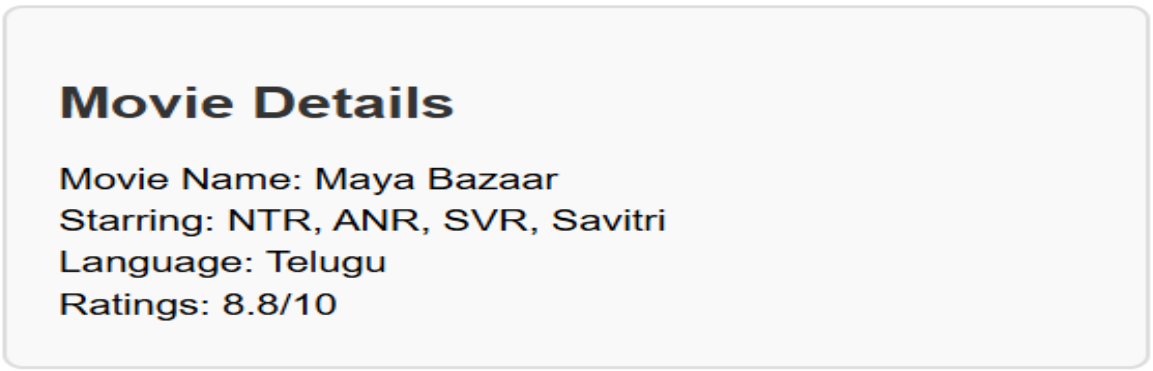
Output:



Exp3b)

```
<!--
Write JavaScript code to display the movie details such as movie name, starring,
language, and ratings. Initialize the variables with values of appropriate types. Use
template literals wherever necessary.
-->

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Movie Details</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 20px;
    }
    .movie-details {
      padding: 20px;
      border: 2px solid #ddd;
      border-radius: 8px;
      background-color: #f9f9f9;
      max-width: 400px;
```

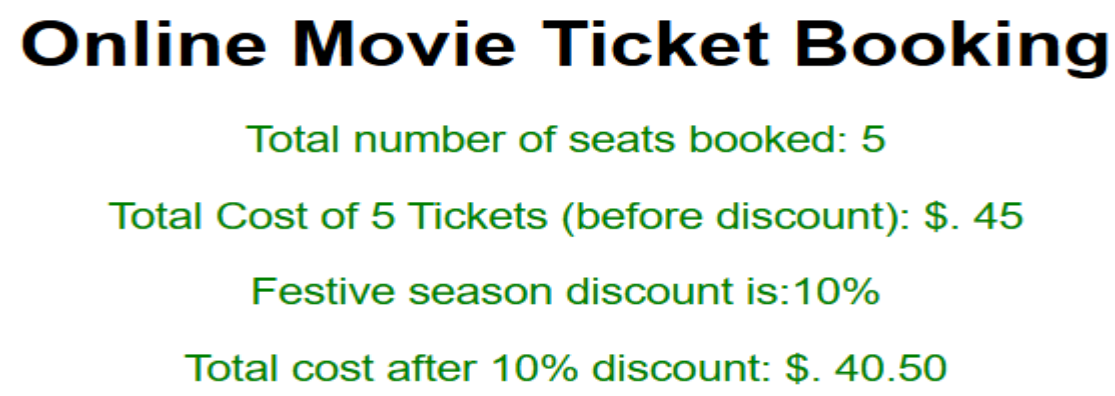
```
        margin: auto;
    }
    h2 {
        color: #333;
    }
    p {
        margin: 5px 0;
    }
  </style>
</head>
<body>
  <div class="movie-details">
    <h2>Movie Details</h2>
    <p id="movieName"></p>
    <p id="starring"></p>
    <p id="language"></p>
    <p id="ratings"></p>
  </div>

  <script>
    // Initialize variables with movie details
    const movieName = "Maya Bazaar";
    const starring = ["NTR", "ANR", "SVR", "Savitri"];
    const language = "Telugu";
    const ratings = 8.8;

    // Use template literals to create the movie details
            document.getElementById("movieName").textContent = `Movie Name: ${movieName}`;
    document.getElementById("starring").textContent = `Starring: ${starring.join(", ")}`;
    document.getElementById("language").textContent = `Language: ${language}`;
    document.getElementById("ratings").textContent = `Ratings: ${ratings}/10`;
  </script>
</body>
</html>
```



Exp3c)

```
<!--Write JavaScript code to do online booking of theatre tickets and calculate the total price, considering the price per ticket as $9. Also, apply a festive season discount of 10% and calculate the discounted amount. -->

<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Movie Ticket Booking</title>
   <style>
      body {
         font-family: Arial, sans-serif;
         text-align: center;
         margin-top: 50px;
      }
      input, button {
         padding: 10px;
         margin: 10px;
         font-size: 16px;
      }
      .result {
         margin-top: 20px;
         font-size: 18px;
         color: green;
      }

      .hidden {
         display: none;
      }
   </style>
</head>
<body>
   <h1>Online Movie Ticket Booking</h1>
   <div id="form">
   <p>Price per ticket: &dollar;9</p>
   <label for="tickets">Enter the number of tickets:</label>
   <input type="number" id="tickets" placeholder="Number of tickets" min="1">
   <button onclick="calculateTotal()">Book Tickets</button>
   </div>
   <div class="result">
      <p id="totaltickets"></p>
      <p id="totalPrice"></p>
      <p id="discount"></p>
      <p id="discountedPrice"></p>
   </div>

   <script>
```

```
        // Function to calculate total price and discounted price
        function calculateTotal() {
            const pricePerTicket = 9; // Price of one ticket
            const discountRate = 0.10; // 10% discount

            // Get the number of tickets from user input
            const tickets = parseInt(document.getElementById("tickets").value);

            // Validate the input
            if (isNaN(tickets) || tickets <= 0) {
              document.getElementById("totalPrice").textContent = "Please enter a valid number
of tickets.";
                document.getElementById("discountedPrice").textContent = "";
                return;
            }

            // Calculate the total price and discounted price
            const totalPrice = tickets * pricePerTicket;
            const discount = totalPrice * discountRate;
            const discountedPrice = totalPrice - discount;

            document.getElementById("form").classList.add("hidden");

            // Display the results
                document.getElementById("totaltickets").textContent = `Total number of seats
booked: ${tickets}`;
                document.getElementById("totalPrice").textContent = `Total Cost of ${tickets}
Tickets (before discount): $. ${totalPrice}`;
                 document.getElementById("discount").textContent = `Festive season discount
is:10%`;
              document.getElementById("discountedPrice").textContent = `Total cost after 10%
discount: $. ${discountedPrice.toFixed(2)}`;
        }
    </script>
</body>
</html>
```



Exp3d)

```
<!DOCTYPE html>
<html>
<head>
<title>Theatre Ticket Booking</title>
<style>
body {
  font-family: sans-serif;
}
.container {
  width: 300px;
  margin: 0 auto;
  padding: 20px;
  border: 1px solid #ccc;
}
label {
  display: block;
  margin-bottom: 10px;
}
input[type="number"] {
  width: 100%;
  padding: 5px;
  margin-bottom: 15px;
  box-sizing: border-box;
}
button {
  background-color: #4CAF50;
  color: white;
  padding: 10px 20px;
  border: none;
  border-radius: 3px;
  cursor: pointer;
}
</style>
</head>
<body>

<div class="container">
  <h2>Theatre Ticket Booking</h2>
  <label for="seats">Number of Seats:</label>
  <input type="number" id="seats" min="1" max="4">
  <button onclick="calculateCost()">Book Tickets</button>
  <p id="result"></p>
</div>

<script>
const baseTicketPrice = 9;

function calculateCost() {
```

```
  const numSeats = parseInt(document.getElementById("seats").value);

  if (numSeats <= 2) {
    const totalPrice = numSeats * baseTicketPrice;
     document.getElementById("result").innerHTML = `Total cost for ${numSeats} seats: $${totalPrice.toFixed(2)}`;
  } else if (numSeats >= 5) {
    document.getElementById("result").innerHTML = "Booking not allowed for more than 4 seats.";
  } else {
    let output = "";
    for (let customer = 1; customer <= numSeats; customer++) {
      const discountPercentage = (customer - 1) * 2 + 5;
      const discountedPrice = baseTicketPrice * (1 - discountPercentage / 100);
     output += `Ticket for customer ${customer} gets ${discountPercentage}% discount, cost is: $${discountedPrice.toFixed(2)}<br>`;
    }
    output += `For ${numSeats} tickets, you need to pay: $${(numSeats * baseTicketPrice * (1 - ((numSeats - 1) * 2 + 5) / 100)).toFixed(2)} instead of $${numSeats * baseTicketPrice}`;
    document.getElementById("result").innerHTML = output;
  }
}
</script>

</body>
</html>
```

Exp3e)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Theatre Ticket Booking</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            text-align: center;
            margin-top: 50px;
        }
        input, button {
            padding: 10px;
            margin: 10px;
            font-size: 16px;
        }
        .result {
            margin-top: 20px;
            font-size: 18px;
            color: green;
            text-align: center;
            display: inline-block;
            white-space: pre-line; /* Allows newlines to display as line breaks */
        }
        .error {
            color: red;
        }

        .hidden {
            display: none;
        }
    </style>
</head>
<body>
    <h1>Theatre Ticket Booking</h1>
    <div id="form">
        <p>Price per ticket: $9</p>
        <label for="seats">Enter the number of seats to book:</label>
        <input type="number" id="seats" placeholder="Number of seats" min="1">
        <button onclick="bookTickets()">Book Tickets</button>
    </div>
    <div class="result">
        <p id="breakdown"></p>
        <p id="totalPrice"></p>
        <p id="errorMessage" class="error"></p>
    </div>
```

```
<script>
    function bookTickets() {
        const costPerTicket = 9; // Base cost per ticket
        const discountRates = [5, 7, 9, 11]; // Discounts for customer1, customer2, etc.
        const seats = parseInt(document.getElementById("seats").value);

        // Clear previous messages
        document.getElementById("breakdown").textContent = "";
        document.getElementById("totalPrice").textContent = "";
        document.getElementById("errorMessage").textContent = "";

        document.getElementById("form").classList.add("hidden");

        // Validate input
        if (isNaN(seats) || seats <= 0) {
            document.getElementById("errorMessage").textContent = "Please enter a valid number of seats.";
            document.getElementById("form").classList.remove("hidden");
            return;
        }

        // Booking is not allowed for 5 or more seats
        if (seats >= 5) {
          document.getElementById("errorMessage").textContent = "Booking is not allowed for 5 or more seats.";
            document.getElementById("form").classList.remove("hidden");
            return;
        }

        // Handle cases with a loop
        let totalCost = 0;
        let breakdown = "";

        for (let i = 0; i < seats; i++) {
            const discount = costPerTicket * discountRates[i] / 100;
            const price = costPerTicket - discount;
            breakdown += `Ticket for customer ${i + 1} gets ${discountRates[i]}% discount, cost is: $${price.toFixed(2)}\n`;
            totalCost += price;
        }

        const originalCost = seats * costPerTicket; // Original cost without discount
        const savings = originalCost - totalCost; // Total savings

        // Update the page with results
        document.getElementById("breakdown").textContent = breakdown;
       document.getElementById("totalPrice").textContent = `For ${seats} tickets, you need to pay: $${totalCost.toFixed(2)} instead of $${originalCost.toFixed(2)} (You save $${savings.toFixed(2)})`;
    }
```

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

Theatre Ticket Booking

Ticket for customer 1 gets 5% discount, cost is: $8.55
Ticket for customer 2 gets 7% discount, cost is: $8.37
Ticket for customer 3 gets 9% discount, cost is: $8.19
Ticket for customer 4 gets 11% discount, cost is: $8.01

For 4 tickets, you need to pay: $33.12 instead of $36.00 (You save $2.88)

Experiment 4:

Exp4a:

Aim: Write a JavaScript code to do online booking of theatre tickets and calculate the total price based on the below conditions:

4. If seats to be booked are not more than 2, the cost per ticket remains $9.
5. If seats are 6 or more, booking is not allowed.
6. If seats to be booked are more than 2 but less than 5, based on the number of seats booked, do the following:
    - Calculate total cost by applying a discount of 5, 7, 9, 11 percent, and so on for customer 1,2,3 till 5.
    - Try the code with different values for the number of seats.

Write the following custom functions to implement given requirements:

2. calculateCost(seats): Calculate and display the total cost to be paid by the customer for the tickets they have bought.
3. calculateDiscount(seats): Calculate discount on the tickets bought by the customer. Implement using arrow functions.

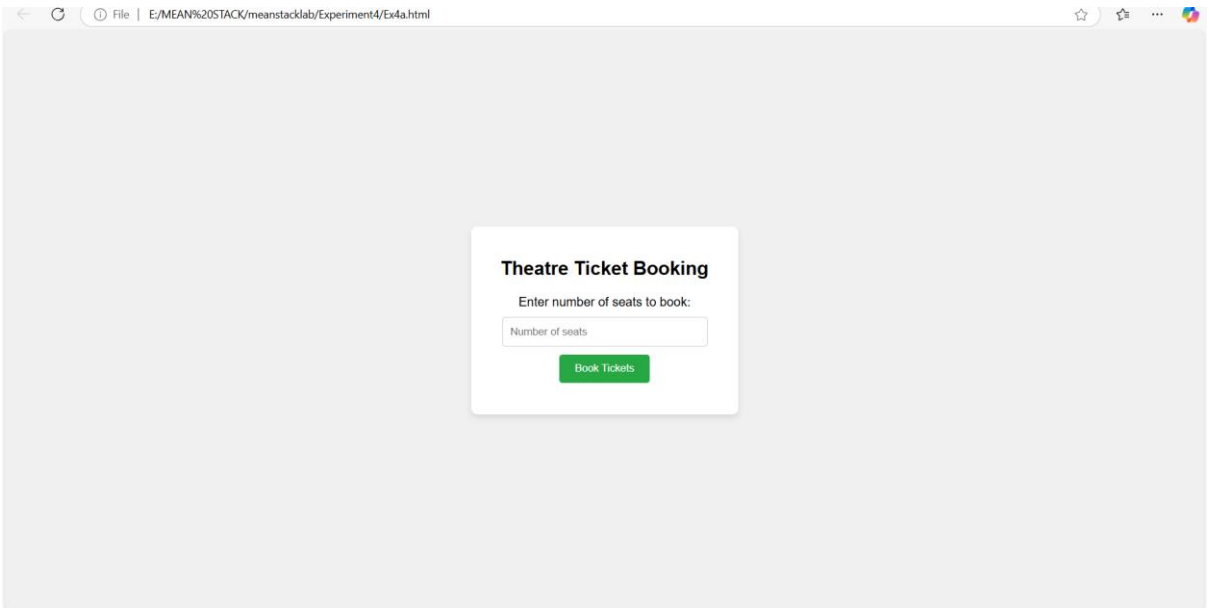
Program:

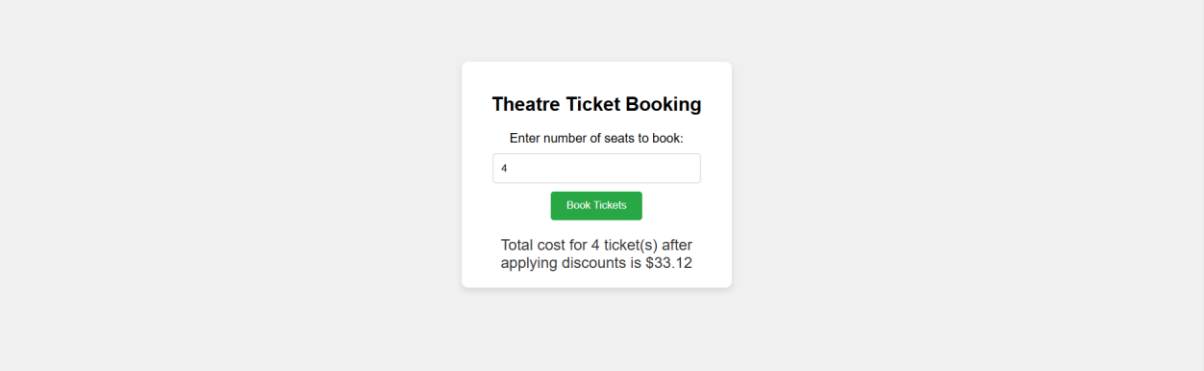
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Theatre Ticket Booking</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 0;
      padding: 0;
      display: flex;
      flex-direction: column;
      align-items: center;
      justify-content: center;
      height: 100vh;
      background-color: #f0f0f0;
    }
    .booking-form {
      background: #fff;
      padding: 20px;
      border-radius: 8px;
      box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
      text-align: center;
      width: 300px;
```

```
    }
    .booking-form input {
      width: 80%;
      padding: 10px;
      margin: 10px 0;
      border: 1px solid #ccc;
      border-radius: 4px;
    }
    .booking-form button {
      padding: 10px 20px;
      background-color: #28a745;
      color: white;
      border: none;
      border-radius: 4px;
      cursor: pointer;
    }
    .booking-form button:hover {
      background-color: #218838;
    }
    .result {
      margin-top: 20px;
      font-size: 1.2em;
      color: #333;
    }
  </style>
</head>
<body>
  <div class="booking-form">
    <h2>Theatre Ticket Booking</h2>
    <label for="seats">Enter number of seats to book:</label>
    <input type="number" id="seats" placeholder="Number of seats" min="1" max="4">
    <button onclick="bookTickets()">Book Tickets</button>
    <div class="result" id="result"></div>
  </div>

  <script>
    // Function to calculate discount
    const calculateDiscount = (seats) => {
      const ticketPrice = 9;
      const discounts = [5, 7, 9, 11]; // Discounts for 1st, 2nd, 3rd, and 4th tickets

      let totalDiscount = 0;
      for (let i = 0; i < seats; i++) {
        const discount = discounts[i] || 0; // Apply discount for each ticket
        totalDiscount += ticketPrice * (discount / 100);
      }
      return totalDiscount;
    };

    // Function to calculate total cost
```

```
    const calculateCost = (seats) => {
      const ticketPrice = 9;
      if (seats > 4) return "Only 4 tickets are available. Booking not allowed for more than 4 seats.";

      const discount = calculateDiscount(seats);
      const totalPrice = seats * ticketPrice;
      const finalPrice = totalPrice - discount;

      return `Total cost for ${seats} ticket(s) after applying discounts is $${finalPrice.toFixed(2)}`;
    };

    // Main function to handle booking
    const bookTickets = () => {
      const seatsInput = document.getElementById("seats");
      const resultDiv = document.getElementById("result");
      const seats = parseInt(seatsInput.value, 10);

      if (!seats || seats < 1) {
        resultDiv.textContent = "Please enter a valid number of seats.";
        return;
      }

      const message = calculateCost(seats);
      resultDiv.textContent = message;
    };
  </script>
</body>
</html>
```

Output:





Exp4b)

Aim: Create an Employee class extending from a base class Person.

Approach to the solution:

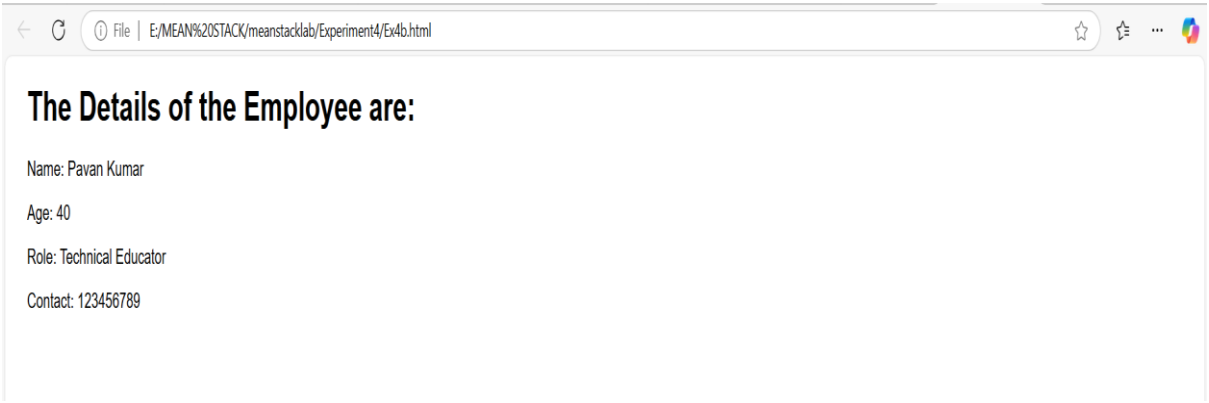
- Create a class Person with name and age as attributes
- Add a constructor to initialize the values
- Create a class Employee extending Person with additional attributes role and contact
- The constructor of the Employee to accept the name, age, role and contact where name and age are initialized through a call to super to invoke the base class constructor
- Add a method getDetails() to display all the details of Employee.

Program:

```
<!DOCTYPE html>
<html>
<head>
  <title>Employee Details</title>
  <style>
    body {
      font-family: sans-serif;
    }
    .container {
      margin: 20px;
    }
  </style>
</head>
<body>

  <div class="container">
    <h1>The Details of the Employee are:</h1>
```

```
    <div id="employeeDetails"></div>
  </div>

  <script>
    class Person {
      constructor(name, age) {
        this.name = name;
        this.age = age;
      }
    }

    class Employee extends Person {
      constructor(name, age, role, contact) {
        super(name, age); // Initialize name and age from Person
        this.role = role;
        this.contact = contact;
      }

     getDetails() {
        return `
          <p>Name: ${this.name}</p>
          <p>Age: ${this.age}</p>
          <p>Role: ${this.role}</p>
          <p>Contact: ${this.contact}</p>
        `;
      }
    }

    // Create an instance of the Employee class
       const employee1 = new Employee("Pavan Kumar", 40, "Technical Educator",
"123456789");

    // Display employee details in the HTML
    const employeeDetailsDiv = document.getElementById("employeeDetails");
    employeeDetailsDiv.innerHTML = employee1.getDetails();

  </script>

</body>
</html>
```

Output:



Exp4c:

Aim: Repeat Experiment 4a that include Event handling

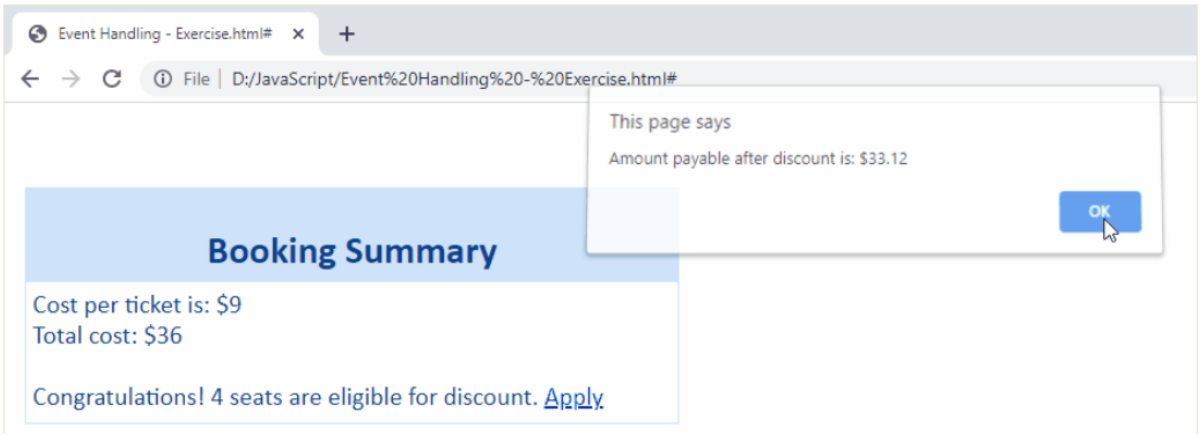
Write a JavaScript code to make online booking of theatre tickets and calculate the total price based on the below conditions:

1. If seats to be booked are not more than 2, the cost per ticket remains $9.
2. If seats are 6 or more, booking is not allowed.
3. If seats to be booked are more than 2 but less than 5, based on the number of seats booked, do the following:
    - Calculate total cost by applying a discount of 5, 7, 9, 11 percent, and so on for customer 1,2,3 till 5.
    - Try the code with different values for the number of seats.

Write the following custom functions to implement given requirements:

- calculateCost(seats): Calculate and display the total cost to be paid by the customer for the tickets he has bought.
- calculateDiscount(seats): Calculate discount on the tickets bought by the customer. Invoke this function only when the user clicks on a link.

Expected Output:



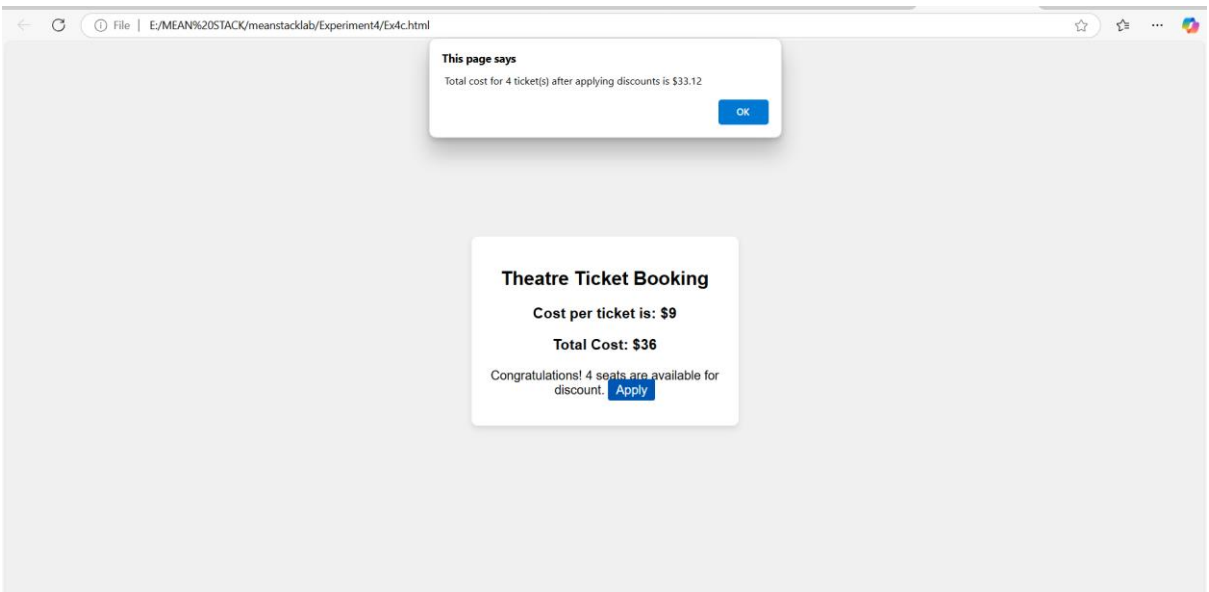
Program:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Theatre Ticket Booking</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 0;
      padding: 0;
      display: flex;
      flex-direction: column;
      align-items: center;
      justify-content: center;
      height: 100vh;
      background-color: #f0f0f0;
    }
    .booking-form {
      background: #fff;
      padding: 20px;
      border-radius: 8px;
      box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
      text-align: center;
      width: 300px;
    }
    .booking-form input {
      width: 80%;
      padding: 10px;
```

```
    margin: 10px 0;
    border: 1px solid #ccc;
    border-radius: 4px;
  }
  .booking-form button {
    padding: 10px 20px;
    background-color: #28a745;
    color: white;
    border: none;
    border-radius: 4px;
    cursor: pointer;
  }
  .booking-form button:hover {
    background-color: #218838;
  }
  .result {
    margin-top: 20px;
    font-size: 1.2em;
    color: #333;
  }
  .booking-form a {
        text-decoration: none; /* Remove default underline */
        color: #007bff; /* Example: Blue link color */
        padding: 5px 10px; /* Optional: Add some padding for better appearance */
        border-radius: 3px; /* Optional: Add rounded corners */
      }

      .booking-form a:hover {
        background-color: #0056b3; /* Example: Darker blue on hover */
        color: #fff; /* White text on hover */
      }
 </style>
</head>
<body>
  <div class="booking-form">
    <h2>Theatre Ticket Booking</h2>
    <h3> Cost per ticket is: $9</h3>
    <h3>Total Cost: $36</h3>
            <p>Congratulations! 4 seats are available for discount. <a
id="applyLink">Apply</a></p>
  </div>

  <script>
    const ticketPrice = 9;
    const discounts = [5, 7, 9, 11]; // Discounts for 1st, 2nd, 3rd, and 4th tickets

    const calculateDiscount = (seats) => {
      let totalDiscount = 0;
      for (let i = 0; i < seats; i++) {
        const discount = discounts[i] || 0;
```

```
            totalDiscount += ticketPrice * (discount / 100);
        }
        return totalDiscount;
    };

    const calculateCost = (seats) => {
        if (seats > 4) return "Only 4 tickets are available. Booking not allowed for more than 4 seats.";

        const discount = calculateDiscount(seats);
        const totalPrice = seats * ticketPrice;
        const finalPrice = totalPrice - discount;

        return `Total cost for ${seats} ticket(s) after applying discounts is $${finalPrice.toFixed(2)}`;
    };

    const applyLink = document.getElementById('applyLink');
    applyLink.addEventListener('click', () => {
        const seats = 4; // Assuming 4 seats are booked
        const discountedCost = calculateCost(seats);
        alert(` ${discountedCost}`);
    });
  </script>
</body>
</html>
```

Output:



4d) Consider the given starter code and style the page as shown in images:

When the user clicks on the given link, they should see an empty cone, a different heading, and a different message and a different background color.

When the user clicks again, they should see a re-filled cone, a different heading, a different message, and a different color in the background.

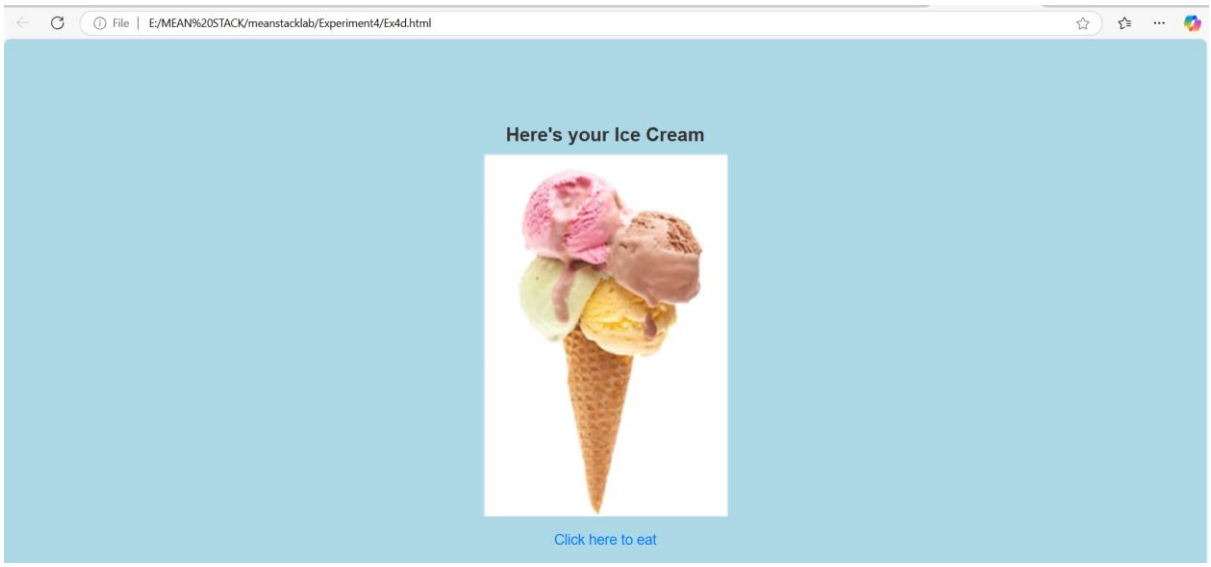
Program:

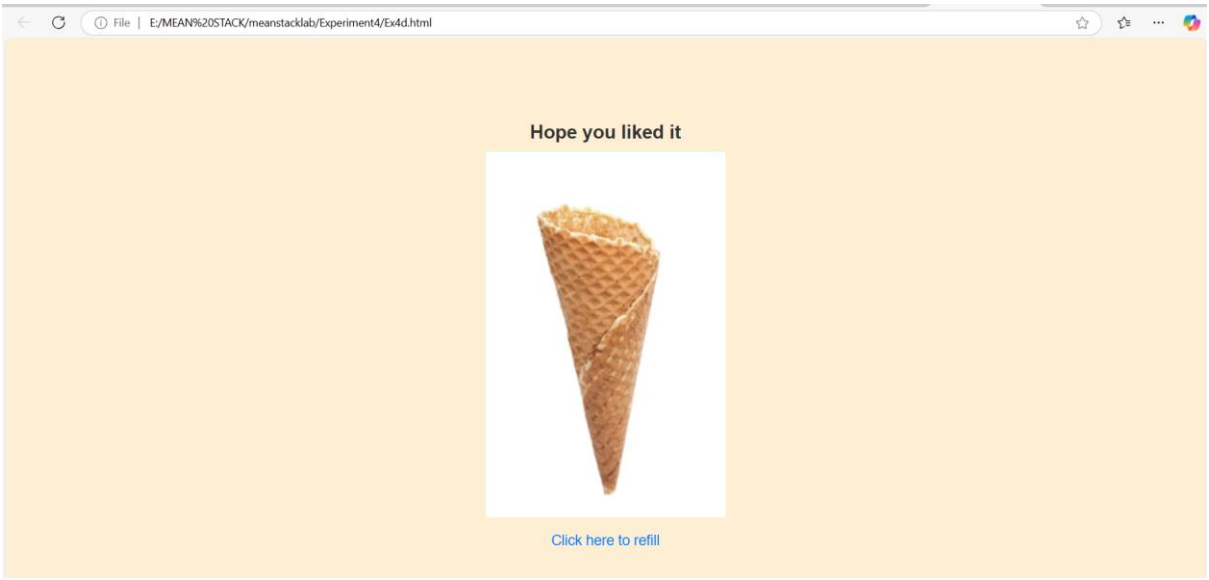
```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Interactive Ice Cream</title>
   <style>
      body {
         display: flex;
         flex-direction: column;
         align-items: center;
         justify-content: center;
         height: 100vh;
         margin: 0;
         background-color: #FFFFFF; /* Default background */
```

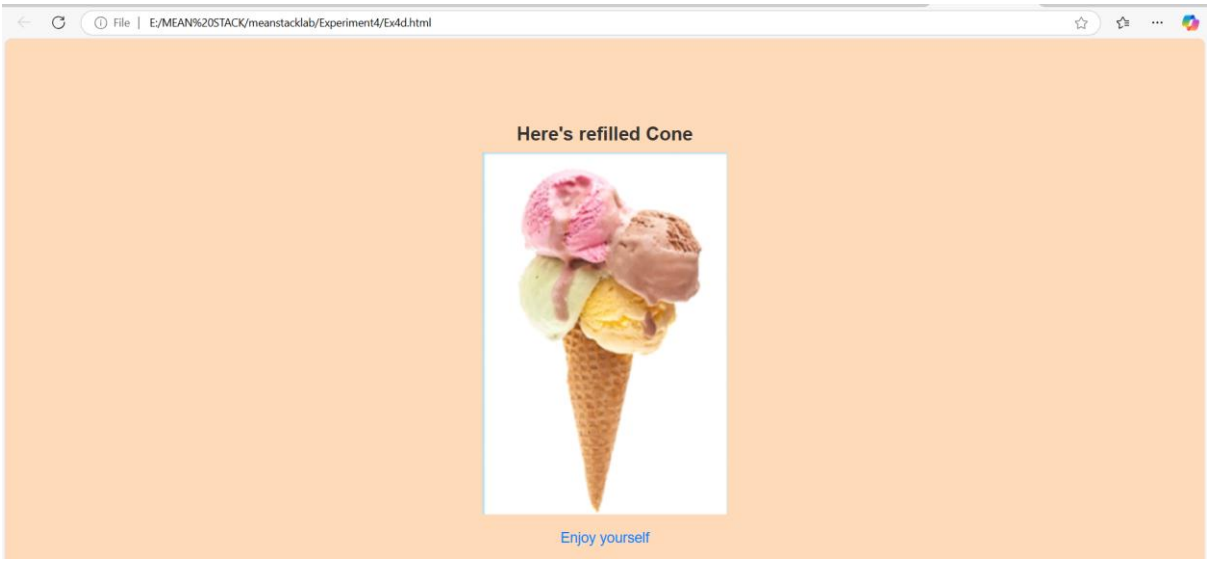
```
        font-family: Arial, sans-serif;
        text-align: center;
      }

      .image {
        margin-bottom: 20px;

      }

      h1 {
        font-size: 24px;
        margin-bottom: 10px;
        color: #333;
      }

      a {
        text-decoration: none;
        color: #007BFF;
        font-size: 18px;
        cursor: pointer;
      }

      a:hover {
        text-decoration: underline;
      }
    </style>
</head>
<body>
    <h1 id="heading">Here's your Ice Cream</h1>
     <img id="iceCreamImage" class="image" src="./images/img1.png"  alt="Empty Ice Cream Cone" />
    <a id="link" onclick="handleClick()">Click here to eat</a>

    <script>
      let state = 0; // 0: empty cone, 1: eaten cone, 2: refilled cone

      function handleClick() {
        const heading = document.getElementById('heading');
        const iceCreamImage = document.getElementById('iceCreamImage');
        const link = document.getElementById('link');

        if (state === 0) {
          // Show eaten cone
          heading.textContent = "Hope you liked it";
          iceCreamImage.src = "./images/img2.png";
          iceCreamImage.alt = "Eaten Ice Cream Cone";
          link.textContent = "Click here to refill";
          document.body.style.backgroundColor = "#FFEFD5"; // Light peach background
          state = 1;
        } else if (state === 1) {
```

```
            // Show refilled cone
            heading.textContent = "Here's refilled Cone";
            iceCreamImage.src = "./images/img1.png";
            iceCreamImage.alt = "Refilled Ice Cream Cone";
            link.textContent = "Enjoy yourself";
            document.body.style.backgroundColor = "#FFDAB9"; // Peach background
            state = 2;
        } else {
            // Reset to empty cone
            heading.textContent = "Click to Eat";
            iceCreamImage.src = "./images/img3.png";
            iceCreamImage.alt = "Empty Ice Cream Cone";
            link.textContent = "Click here to eat";
            document.body.style.backgroundColor = "#FFFFFF"; // White background
            state = 0;
        }
    }
  </script>

</body>
</html>
```

Output:




File | E:/MEAN%20STACK/meanstacklab/Experiment4/Ex4d.html
Hope you liked it
Click here to refill


File | E:/MEAN%20STACK/meanstacklab/Experiment4/Ex4d.html
Here's refilled Cone
Enjoy yourself

Experiment 5:

Aim: a) Create an array of objects having movie details. The object should include the movie name, starring, language, and ratings. Render the details of movies on the page using the array.

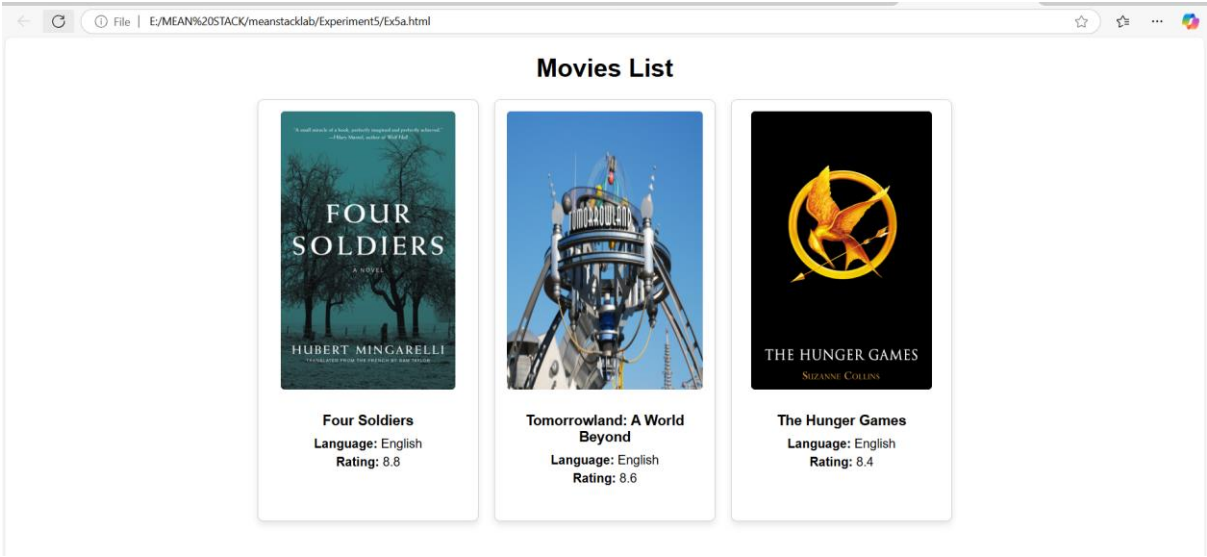
Program:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Movies List</title>
   <style>
      body {
         font-family: Arial, sans-serif;
         margin: 20px;
      }

      h1 {
         text-align: center;
      }

      .movie-container {
         display: flex;
         flex-wrap: wrap;
         justify-content: center;
         gap: 20px;
         margin-top: 20px;
         height:50%;
      }

      .movie-card {
         border: 1px solid #ccc;
         border-radius: 8px;
         padding: 15px;
         width: 250px;
         box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
         text-align: center;
      }

      .movie-card img {
         max-width: 100%;
         height: 70%;
         border-radius: 5px;
         margin-bottom: 10px;
      }

      .movie-card h2 {
```

```
            font-size: 18px;
            margin-bottom: 10px;
        }

        .movie-card p {
            margin: 5px 0;
        }
    </style>
</head>
<body>
    <h1>Movies List</h1>
    <div class="movie-container" id="movieContainer"></div>

    <script>
        // Array of movie objects
        const movies = [
            {
                name: "Four Soldiers",
                language: "English",
                rating: 8.8,
                image: "./images/FourSoldiers.jpg"
            },
            {
                name: "Tomorrowland: A World Beyond",
                language: "English",
                rating: 8.6,
                image: "./images/Tomorrowland.jpg"
            },
            {
                name: "The Hunger Games",
                language: "English",
                rating: 8.4,
                image: "./images/HungerGames.jpg"
            }
        ];

        // Get the container to render movie details
        const movieContainer = document.getElementById("movieContainer");

        // Render movies dynamically
        movies.forEach(movie => {
            // Create a card for each movie
            const movieCard = document.createElement("div");
            movieCard.className = "movie-card";

            // Add movie details
            movieCard.innerHTML = `
                <img src="${movie.image}" alt="${movie.name} Poster">
                <h2>${movie.name}</h2>
                <p><strong>Language:</strong> ${movie.language}</p>
```

```
            <p><strong>Rating:</strong> ${movie.rating}</p>
        `;

        // Append card to the container
        movieContainer.appendChild(movieCard);
      });
    </script>
</body>
</html>
```

Output:

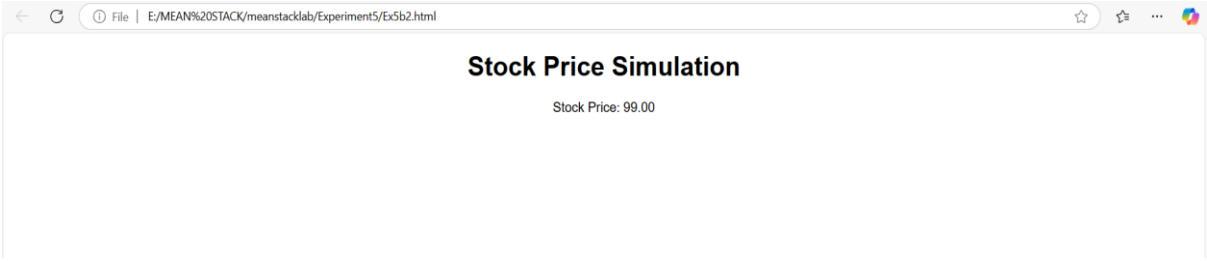


5b) **Aim:** Simulating a periodic stock price change and displaying on the console.

```
<!DOCTYPE html>
<html>
<head>
<title>Stock Price Simulation</title>
<style>
body {
  font-family: sans-serif;
  text-align: center;
}
</style>
</head>
<body>
<h1>Stock Price Simulation</h1>
<div id="price-display"></div>

<script>
async function simulateStockPrice() {
  let currentPrice = 100;
  const displayElement = document.getElementById('price-display');
```

```
    for (let i = 0; i < 5; i++) {
      const priceChange = Math.floor(Math.random() * 3) - 1;
      currentPrice += priceChange;
      displayElement.textContent = `Stock Price: ${currentPrice.toFixed(2)}`;
      await new Promise(resolve => setTimeout(resolve, 3000)); // Pause for 3 seconds
    }

    console.log("Simulation complete.");
  }

  simulateStockPrice();
  </script>

  </body>
  </html>
```

Output:



5c) **Aim:** Create a file login.js with a User class. Create a validate method with username and password as arguments. If the username and password are equal it will return "Login Successful" else will return "Unauthorized access".

Create an index.html file with textboxes username and password and a submit button.
Add a script tag in HTML to include index.js file.
Create an index.js file which imports login module and invokes validate method of User class.
On submit of the button in HTML the validate method of the User class should be invoked.
Implement the validate method to send the username and password details entered by the user and capture the return value to display in the alert.

Program:

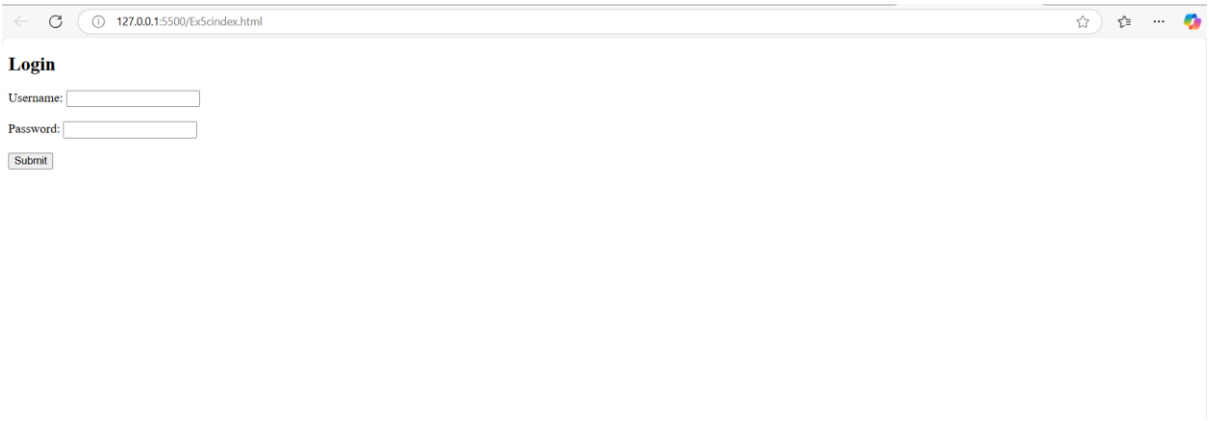
```
//index.html file
    <!DOCTYPE html>
    <html lang="en">
    <head>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
      <title>Login</title>
    </head>
    <body>

      <h2>Login</h2>
```

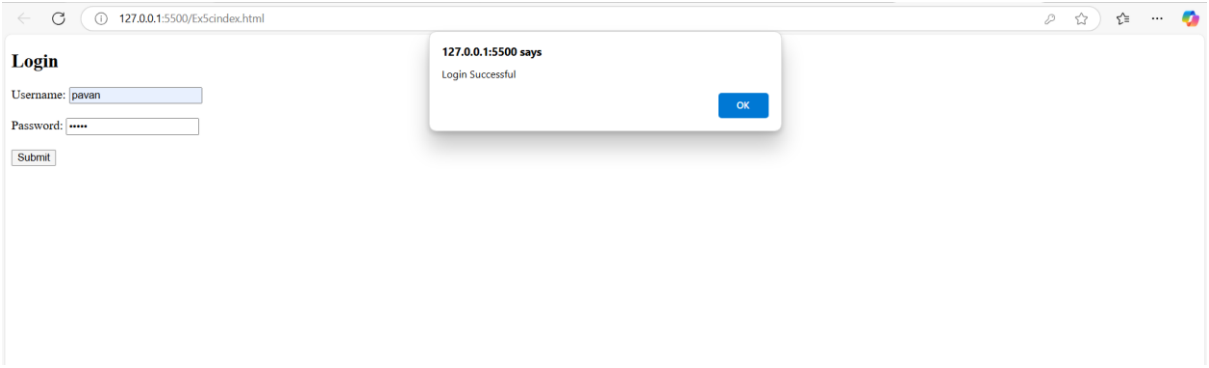
```
    <form id="loginForm">
      <label for="username">Username:</label>
      <input type="text" id="username" name="username" required><br><br>

      <label for="password">Password:</label>
      <input type="password" id="password" name="password" required><br><br>

      <button type="submit">Submit</button>
    </form>

    <script type="module" src="index.js"></script>
  </body>
  </html>
```

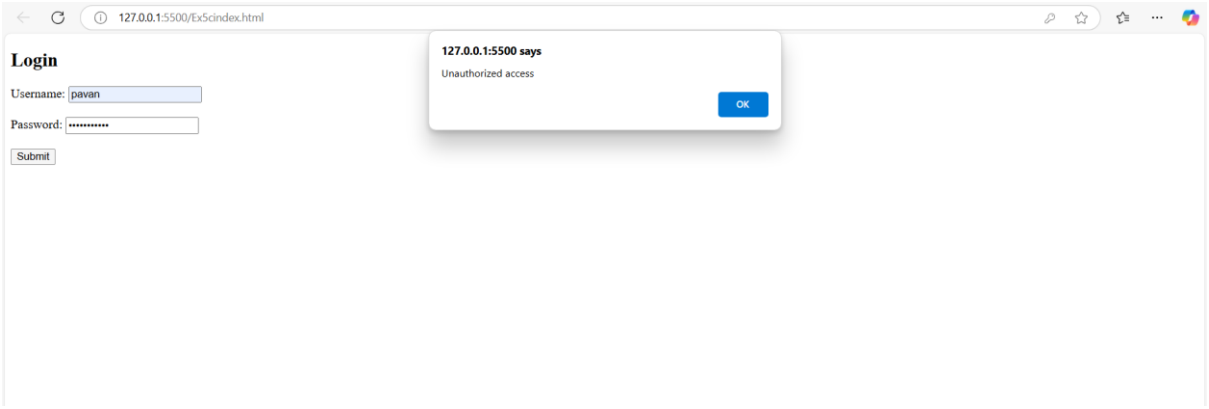
```
//index.js file
import User from './login.js';

const ln=document.getElementById('loginForm');

function handlesubmit(event) {
  event.preventDefault();

  const username = document.getElementById('username').value;
  const password = document.getElementById('password').value;

  const user = new User();
  const result = user.validate(username, password);

  alert(result);
}

ln.addEventListener('submit', handlesubmit );
```

```
login.js
class User {
    validate(username, password) {
      if (username === password) {
        return "Login Successful";
      } else {
        return "Unauthorized access";
      }
    }
  }

  export default User;
```

Open index.html page in LiveServer. Then we will get the output as


127.0.0.1:5500/Ex5cindex.html
Login
Username:
Password:
Submit


127.0.0.1:5500/Ex5cindex.html
Login
Username: pavan
Password:
Submit
127.0.0.1:5500 says
Login Successful
OK


127.0.0.1:5500/Ex5cindex.html
Login
Username: pavan
Password:
Submit
127.0.0.1:5500 says
Unauthorized access
OK

Experiment 6:

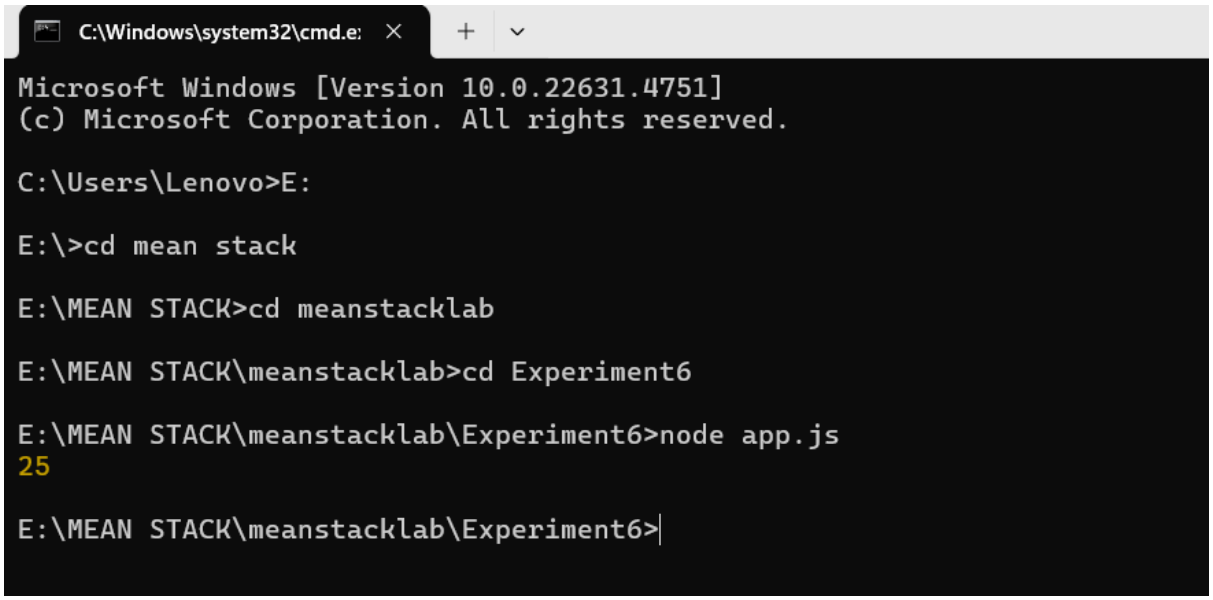
6a) **Aim:** Verify how to execute different functions successfully in the Node.js platform.

```
//app.js
function operateOnNumber(num, operation) {
   return operation(num);
}

function square(x) {
   return x * x;
}

const result = operateOnNumber(5, square);
console.log(result); // Output: 25
```

Navigate to the folder where app.js is located and execute the command

node app.js.

Then the output is



6b) **Aim:** Create a web server in Node.js

Program:

```
//server.js
import { createServer } from 'http';

// Create the server using an arrow function
const server = createServer((req, res) => {
   res.writeHead(200, { 'Content-Type': 'text/plain' });
```

```
    res.write("Hello World! I have created my first server with ES6!");
    res.end();
});

const PORT = 3000;

// Listen on the specified port
server.listen(PORT, () => {
    console.log(`Server is running on http://localhost:${PORT}`);
});

// Error handling: add an event listener for unhandled errors
server.on('error', (err) => {
  console.error('An error occurred:', err);
});
```

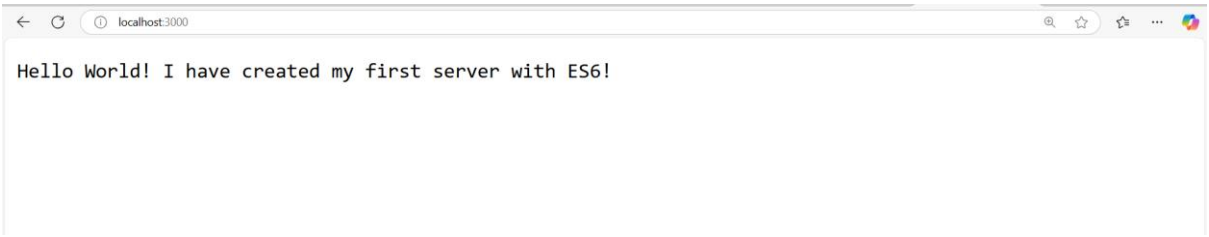
Save the file and start the server using the **node** command.

node server.js.

When the file executes successfully, the following output can be observed in the console.

```
PS E:\MEAN STACK\meanstacklab\Experiment6> node server.js
Server is running on http://localhost:3000
```

Open your browser to visit http://localhost:3000, and you'll see the message



6c) **Aim:** Write a Node.js module to show the workflow of Modularization of Node application.

Program:

Create a DBModule.js file as shown below

```
//DBModule.js
export function authenticateUser(username, password) {
    if (username === "admin" && password === "admin") {
      return "Valid User";
```

```
  } else return "Invalid User";
}
```

Create httpserver.js file and use the autherticateUser function of DBModule.js file

```
//httpserver.js
import { createServer } from "http";
import { authenticateUser } from "./DBModule.js";

const server = createServer(async (request, response) => {
  try {
    const result = authenticateUser("admin", "admin");
    response.writeHead(200, { "Content-Type": "text/html" });
    response.end(`<html><body><h1>${result}</h1></body></html>`);
  } catch (error) {
    console.error("Authentication error:", error);
    response.writeHead(500, { "Content-Type": "text/plain" });
    response.end("Internal Server Error");
  }
  console.log("Request received");
});

server.listen(3000);
console.log("Server is running at port 3000");
```

Save the file and start the server using the **node** command.

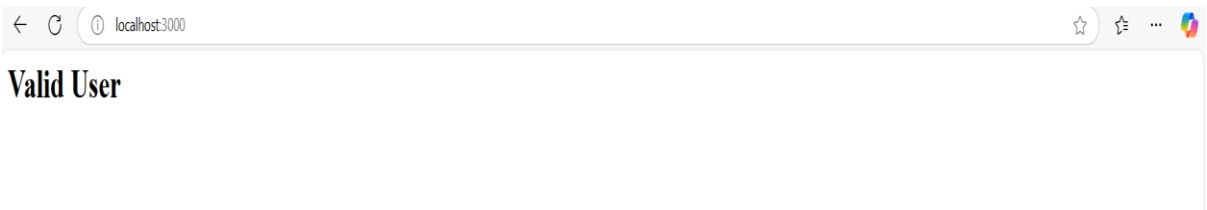
node httpserver.js.

When the file executes successfully, the following output can be observed in the console.

Output:

```
PS E:\MEAN STACK\meanstacklab\Experiment6> node httpserver.js
Server is running at port 3000
```

Open your browser to visit http://localhost:3000, and you'll see the message



6d) **Aim:** Write a program to show the workflow of restarting a Node application.

Program:

While working on a Node.js application and there are no changes in code after the application is started, we will be required to restart the Node process for changes to reflect.

In order to restart the server and to watch for any code changes automatically, we can use the Nodemon tool.

Nodemon

Nodemon is a command-line utility that can be executed from the terminal. It provides a different way to start a Node.js application. It watches the application and whenever any change is detected, it restarts the application.

To install it in the application, run the below command.

```
npm install -g nodemon
```

To illustrate usage of nodemon, create a file app.js as shown below.'

```
//app.js
function operateOnNumber(num, operation) {
   return operation(num);
}

function square(x) {
   return x * x;
}

const result = operateOnNumber(5, square);
console.log(result); // Output: 25
```

Now to start the application, instead of node app.js command, use the command

```
nodemon app.js
```

Thus, the 'nodemon' starts the application in watch mode and restarts the application automatically when any change is detected.

6e) **Aim:** Create a text file src.txt and add the following data to it. Mongo, Express, Angular, Node.

Program:

```
// Create a text file src.txt and add the following data to it. Mongo, Express, Angular,
//Node.

import { writeFile } from 'fs';

// Data to be written to the file
const data = 'Mongo, Express, Angular,Node.';

// Writing data to 'src.txt'
writeFile('src.txt', data, (err) => {
    if (err) {
        console.error('Error writing to file:', err);
    } else {
        console.log('File "src.txt" has been created and data has been added.');
    }
});
```

Output:

```
PS E:\MEAN STACK\meanstacklab\Experiment6> node ex6e.js
File "src1.txt" has been created and data has been added.
```

Experiment 7:

7a) **Aim:** Implement routing for the **AdventureTrails** application as per the below requirement. Write the necessary code in the routes/route.js file.

Sl. No.	Route path	Success response	Error response
1	/packages	Status code: 200 A JSON object with property: • message: "You can now get the requested packages for your request".	Status code: 404 A JSON object with property: • status: "fail" • message: the error message
2	/bookpackage	Status code: 201 A JSON object with property: • message: "New booking added for the POST request"	Status code: 404 A JSON object with property: • status: "fail" • message: the error message

Program:

//route1.js

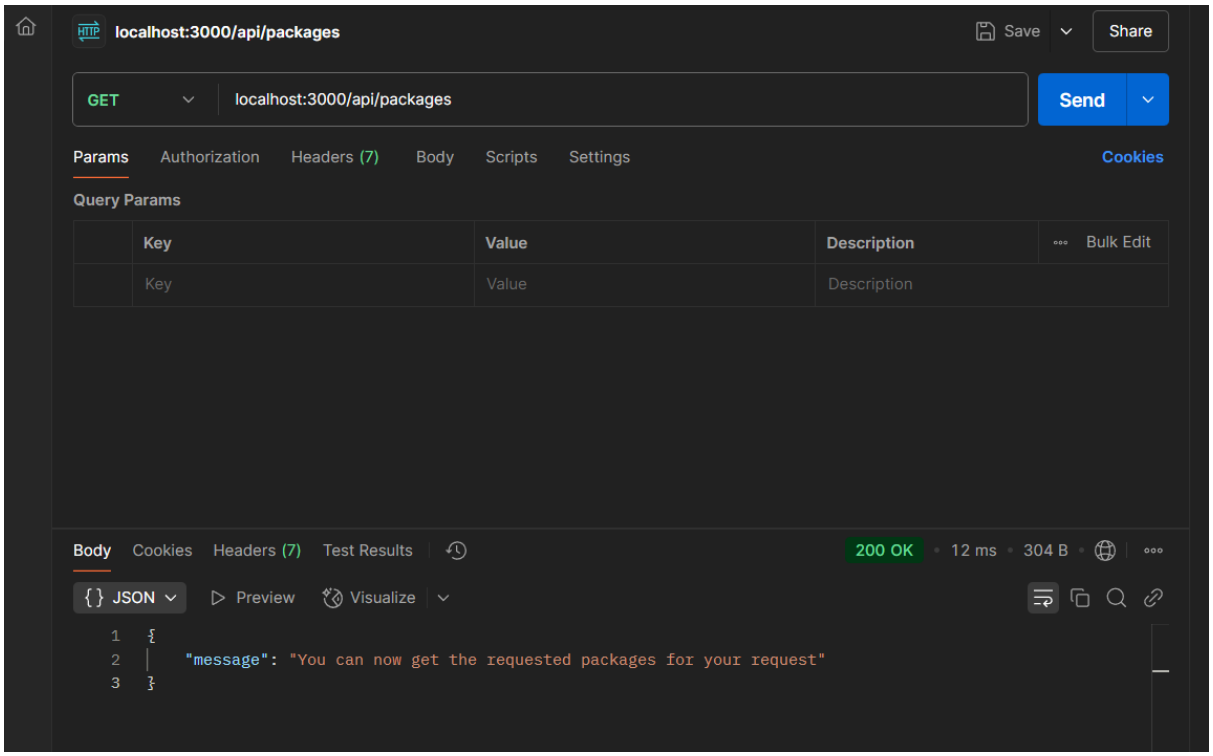
```
import express from 'express';

const router = express.Router();

// Route for "/packages"
router.get('/packages', (req, res) => {
    try {
        res.status(200).json({
            message: "You can now get the requested packages for your request"
        });
    } catch (error) {
        res.status(404).json({
            status: "fail",
            message: error.message || "An error occurred while fetching packages"
        });
    }
});

// Route for "/bookpackage"
router.post('/bookpackage', (req, res) => {
    try {
        res.status(201).json({
```

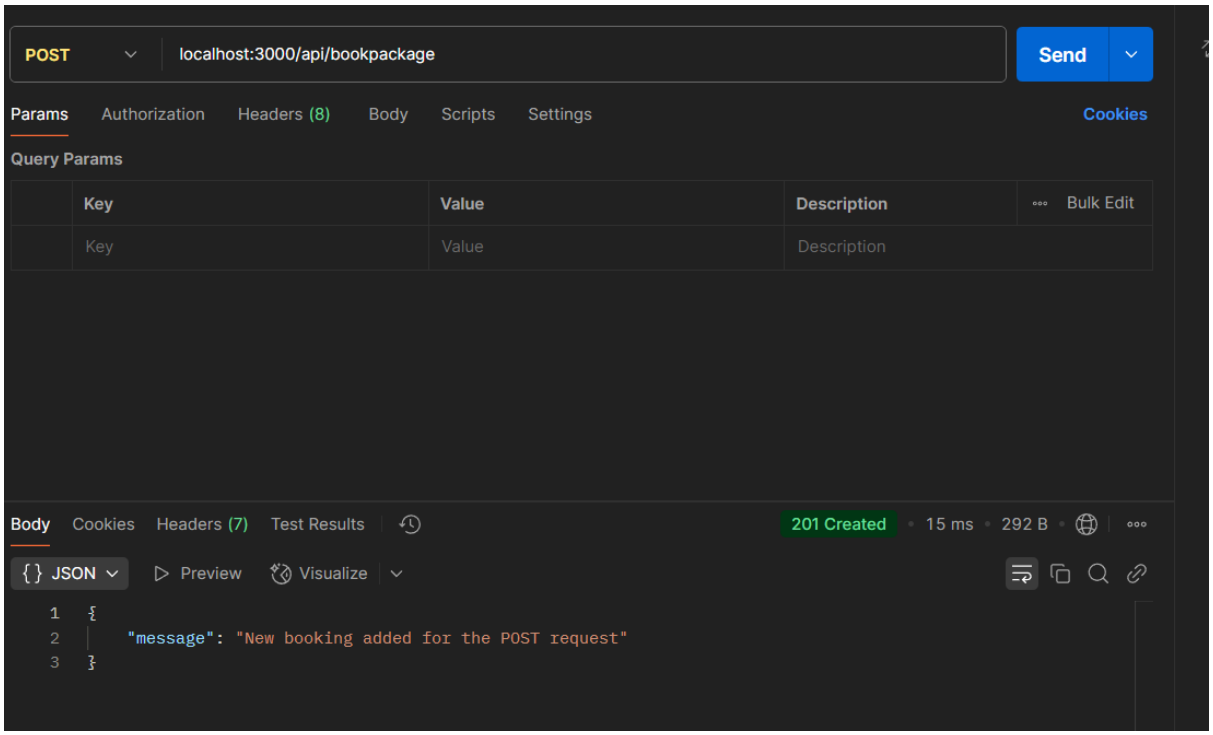
```
            message: "New booking added for the POST request"
        });
    } catch (error) {
        res.status(404).json({
            status: "fail",
            message: error.message || "An error occurred while fetching packages"
        });
    }
});

export default router;
```

```
//server.js
import express from 'express';
import packageRoutes from './route1.js';

const app = express();
const PORT = 3000;

// Middleware to use the package routes
app.use('/api', packageRoutes);

// Handle unknown routes
app.use((req, res) => {
    res.status(404).json({
        status: "fail",
        message: "Route not found"
    });
});

app.listen(PORT, () => {
    console.log(`Server running on http://localhost:${PORT}`);
});
```

Now execute the command node server.js and check the output in postman as follows.

Output:


localhost:3000/api/packages
Save
Share
GET
localhost:3000/api/packages
Send
Params
Authorization
Headers (7)
Body
Scripts
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Key
Value
Description
Body
Cookies
Headers (7)
Test Results
200 OK
12 ms
304 B
{} JSON
Preview
Visualize
{
"message": "You can now get the requested packages for your request"
}


POST
localhost:3000/api/bookpackage
Send
Params
Authorization
Headers (8)
Body
Scripts
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Key
Value
Description
Body
Cookies
Headers (7)
Test Results
201 Created
15 ms
292 B
{} JSON
Preview
Visualize
{
"message": "New booking added for the POST request"
}

7b) **Aim:** In myNotes application: Illustrating the usage of middlewares by performing

(i) handling POST submissions. (ii) displaying customized error messages. (iii) perform logging.

Program:

```
//mynotes.js
import express from 'express';

const app = express();
const PORT = 3000;
let notes = []; // Temporary storage for notes

// Built-in Middleware: Parse JSON Requests
app.use(express.json());

// Logging Middleware: Logs each request
app.use((req, res, next) => {
    console.log(`[${new Date().toISOString()}] ${req.method} ${req.url}`);
    next(); // Move to the next middleware
});

// Validation Middleware: Ensure POST request has title & content
const validateNote = (req, res, next) => {
    const { title, content } = req.body;

    if (!title || !content) {
        const error = new Error("Title and Content are required fields.");
        error.status = 400;
        return next(error); // Pass to error handler
    }

    next(); // Move to the next handler
};

// POST /api/notes - Add a new note
app.post('/api/notes', validateNote, (req, res) => {
    const { title, content } = req.body;
    const newNote = { id: notes.length + 1, title, content };
    notes.push(newNote);

    res.status(201).json({
        message: "Note added successfully",
        note: newNote
    });
});

//GET /api/notes - Retrieve all notes
```

```
app.get('/api/notes', (req, res) => {
    res.status(200).json({ notes });
});

// Error-Handling Middleware: Customized error responses
app.use((err, req, res, next) => {
    console.error(`Error: ${err.message}`);
    res.status(err.status || 500).json({
        status: "fail",
        message: err.message || "Internal Server Error"
    });
});

// Start the server
app.listen(PORT, () => {
    console.log(`Server is running on http://localhost:${PORT}`);
});
```

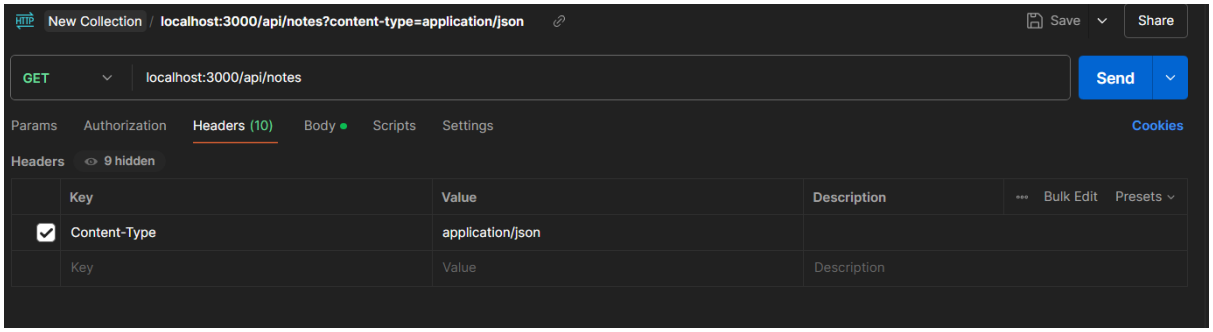
Now execute the above code using node mynotes.js and can check the output in postman.(In the postman, select **"Body" → "raw" → JSON format** and send the request with the following JSON body:

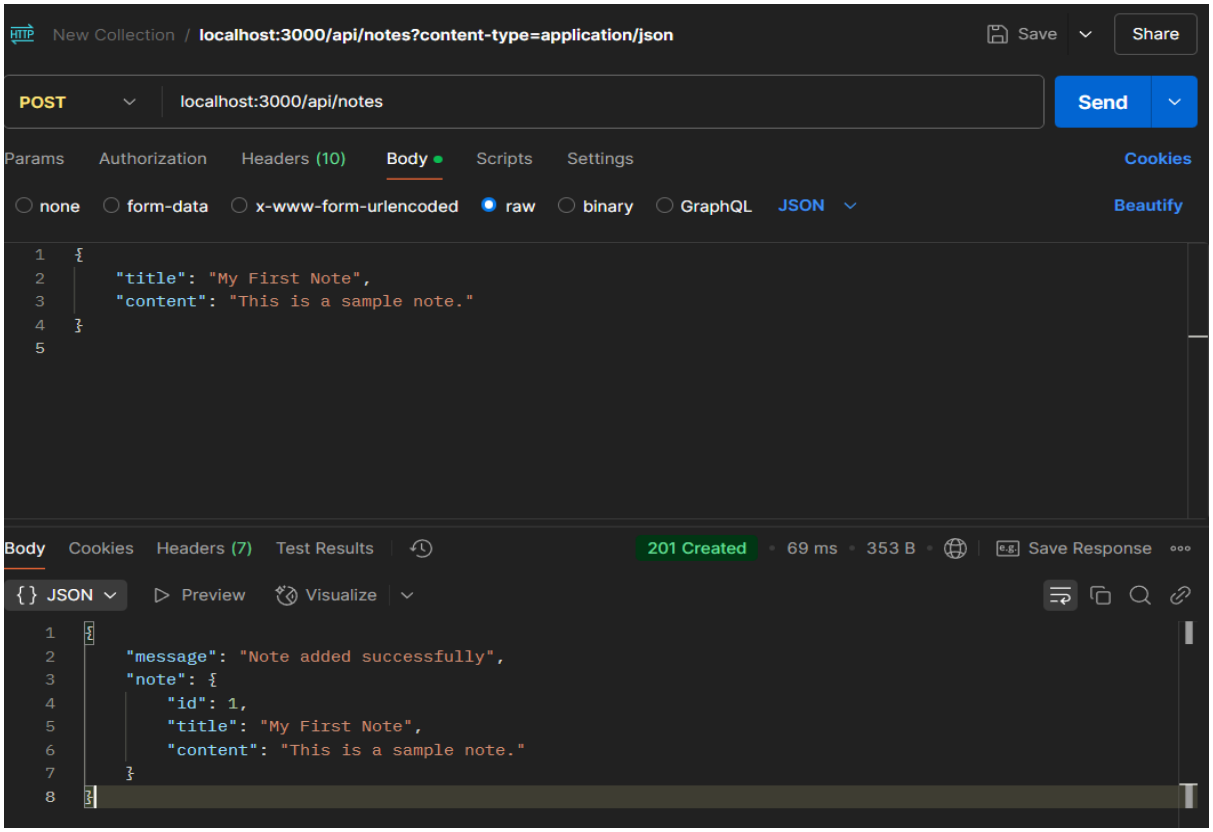
```
{
    "title": "My First Note",
    "content": "This is a sample note."
}
```

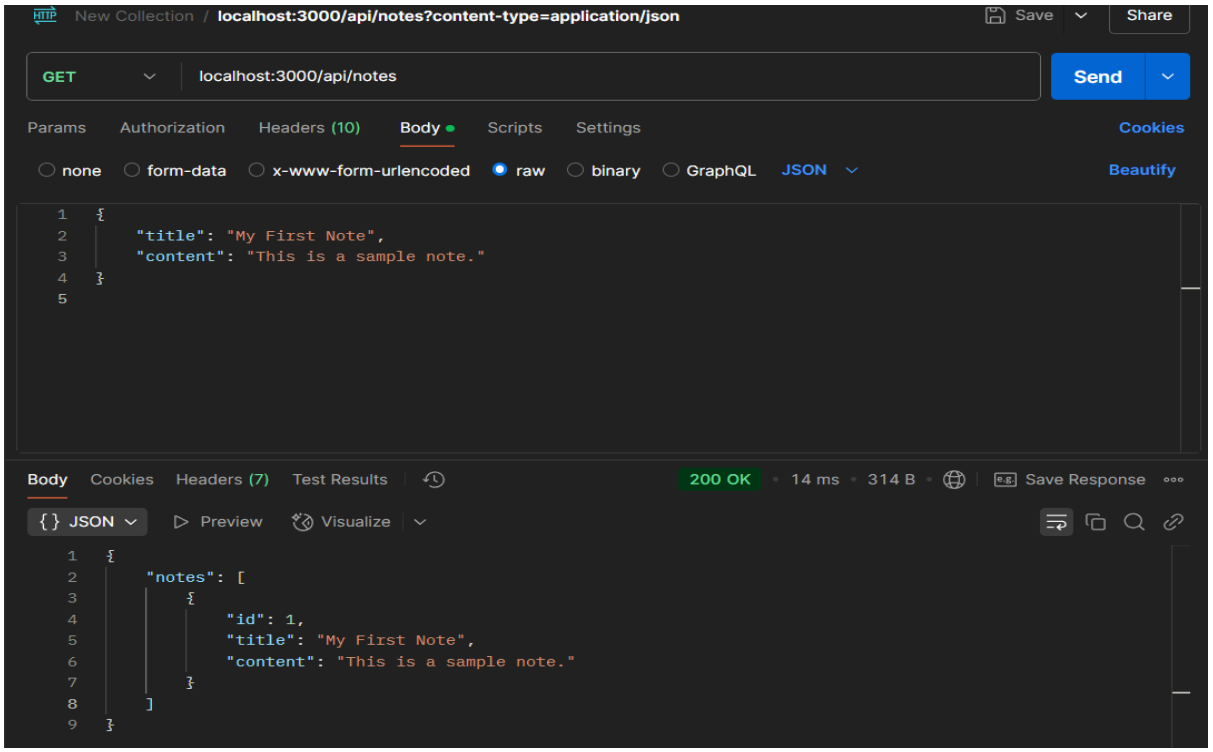
Also set the headers as

```
Key: Content-Type
Value: application/json
```

Output:




New Collection / localhost:3000/api/notes?content-type=application/json
Save
Share
POST
localhost:3000/api/notes
Send
Params
Authorization
Headers (10)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"title": "My First Note",
"content": "This is a sample note."
}
Body
Cookies
Headers (7)
Test Results
201 Created
69 ms
353 B
Save Response
JSON
Preview
Visualize
{
"message": "Note added successfully",
"note": {
"id": 1,
"title": "My First Note",
"content": "This is a sample note."
}
}


New Collection / localhost:3000/api/notes?content-type=application/json
Save
Share
GET
localhost:3000/api/notes
Send
Params
Authorization
Headers (10)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"title": "My First Note",
"content": "This is a sample note."
}
Body
Cookies
Headers (7)
Test Results
200 OK
14 ms
314 B
Save Response
JSON
Preview
Visualize
{
"notes": [
{
"id": 1,
"title": "My First Note",
"content": "This is a sample note."
}
]
}

Exp 7c &7d : **Aim:** Connecting to MongoDB with Mongoose, Validation Types and Defaults. Write a Mongoose schema to connect with MongoDB. Write a program to wrap the Schema into a Model object.

Program:

First install mongoose using npm install mongoose and then create the following files

//db.js file – Handles mongodb connection

```
import mongoose from "mongoose";

const connectDB = async () => {
    try {
        await mongoose.connect("mongodb://127.0.0.1:27017/userDB");
        console.log("Connected to MongoDB");
    } catch (err) {
        console.error("Could not connect to MongoDB:", err);
        process.exit(1); // Exit process if connection fails
    }
};

export default connectDB;
```

//userModel.js -Define mongoose schema

```
import mongoose from "mongoose";

const userSchema = new mongoose.Schema({
    name: { type: String, required: true },
    email: { type: String, required: true, unique: true },
    age: { type: Number, min: 18, max: 60 },
    isActive: { type: Boolean, default: true },
    createdAt: { type: Date, default: Date.now }
});

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

export default User;
```

//app.js

```
import connectDB from "./db.js";
import User from "./userModel.js";

await connectDB(); // Ensure database is connected

const createUser = async () => {
    try {
        const user = new User({
            name: "pavan",
```

```
        email: "pavankumar@example.com",
        age: 40
      });

      const result = await user.save();
      console.log("User saved:", result);
    } catch (err) {
      console.error("Error saving user:", err.message);
    }
};

createUser();
```

Run the command node app.js to see the following output.

Output:

```
PS E:\MEAN STACK\meanstacklab\Exp7express\Ex7c> node app.js
Connected to MongoDB
User saved: {
  name: 'pavan',
  email: 'pavankumar@example.com',
  age: 40,
  isActive: true,
  _id: new ObjectId('67e043d2cf55fae2a7c541fb'),
  createdAt: 2025-03-23T17:24:34.224Z,
  __v: 0
}
```

Experiment 8:

Ex 8a &8b)

Aim: Write a program to perform various CRUD (Create-Read-Update-Delete) operations using Mongoose library functions. Include APIs (API should fetch the details, API should update the details, API should delete the details)

Program:

After Installing node.js, Create a folder for Express project and navigate into it. Initialize your project and create a package.json file. This can be done using following commands

```
mkdir mongoose-crud
cd mongoose-crud
npm init -y
```

Now Install the required packages

```
npm install express mongoose dotenv cors body-parser
```

structure the project as follows:

```
mongoose-crud/
├── models/
│   └── userModel.js
├── routes/
│   └── userRoutes.js
├── controllers/
│   └── userController.js
├── server.js
└── .env
```

Now Perform the following steps.

Step1: Create .env File

```
MONGO_URI=mongodb://localhost:27017/mongoose_crud
PORT=5000
```

Step2: create the db.js file

```
const mongoose = require('mongoose');
require('dotenv').config();

const connectDB = async () => {
    try {
        await mongoose.connect(process.env.MONGO_URI, {
            useNewUrlParser: true,
```

```
            useUnifiedTopology: true
        });
        console.log("MongoDB Connected...");
    } catch (error) {
        console.error("Database connection failed", error);
        process.exit(1);
    }
};

module.exports = connectDB;
```

Step3: Create a file userModel.js inside models folder (models/userModel.js)

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

const userSchema = new mongoose.Schema({
    name: String,
    email: String,
    age: Number
}, { timestamps: true });

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

module.exports = User;
```

Step4: Create userController.js inside controllers folder (controllers/userController.js)

```
const User = require('../models/userModel');

// Create a new user
exports.createUser = async (req, res) => {
    try {
        const user = new User(req.body);
        await user.save();
        res.status(201).json(user);
    } catch (error) {
        res.status(400).json({ error: error.message });
    }
};

// Get all users
exports.getUsers = async (req, res) => {
    try {
        const users = await User.find();
        res.json(users);
    } catch (error) {
```

```
        res.status(500).json({ error: error.message });
    }
};

// Get user by ID
exports.getUserById = 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) {
        res.status(500).json({ error: error.message });
    }
};

// Update user
exports.updateUser = async (req, res) => {
    try {
        const user = await User.findByIdAndUpdate(req.params.id, req.body, { new: true });
        if (!user) return res.status(404).json({ message: "User not found" });
        res.json(user);
    } catch (error) {
        res.status(500).json({ error: error.message });
    }
};

// Delete user
exports.deleteUser = async (req, res) => {
    try {
        const user = await User.findByIdAndDelete(req.params.id);
        if (!user) return res.status(404).json({ message: "User not found" });
        res.json({ message: "User deleted successfully" });
    } catch (error) {
        res.status(500).json({ error: error.message });
    }
};
```

Step5: Create userRoutes.js inside routes folder(routes/userRoutes.js)

```
const express = require('express');
const { createUser, getUsers, getUserById, updateUser, deleteUser } =
require('../controllers/userController');

const router = express.Router();

router.post('/users', createUser);
router.get('/users', getUsers);
router.get('/users/:id', getUserById);
router.put('/users/:id', updateUser);
router.delete('/users/:id', deleteUser);
```

```
module.exports = router;
```

Step6: Create server.js

```
const express = require('express');
const connectDB = require('./db');
const dotenv = require('dotenv');
const cors = require('cors');
const bodyParser = require('body-parser');
const userRoutes = require('./routes/userRoutes');

dotenv.config();
connectDB();

const app = express();

app.use(cors());
app.use(bodyParser.json());
app.use('/api', userRoutes);

const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

Now start the server by using the command node server.js and see the output. To see the output and to test API we will use postman.

To Create a User (POST), Open **Postman,** Select **POST and** Enter the URL:

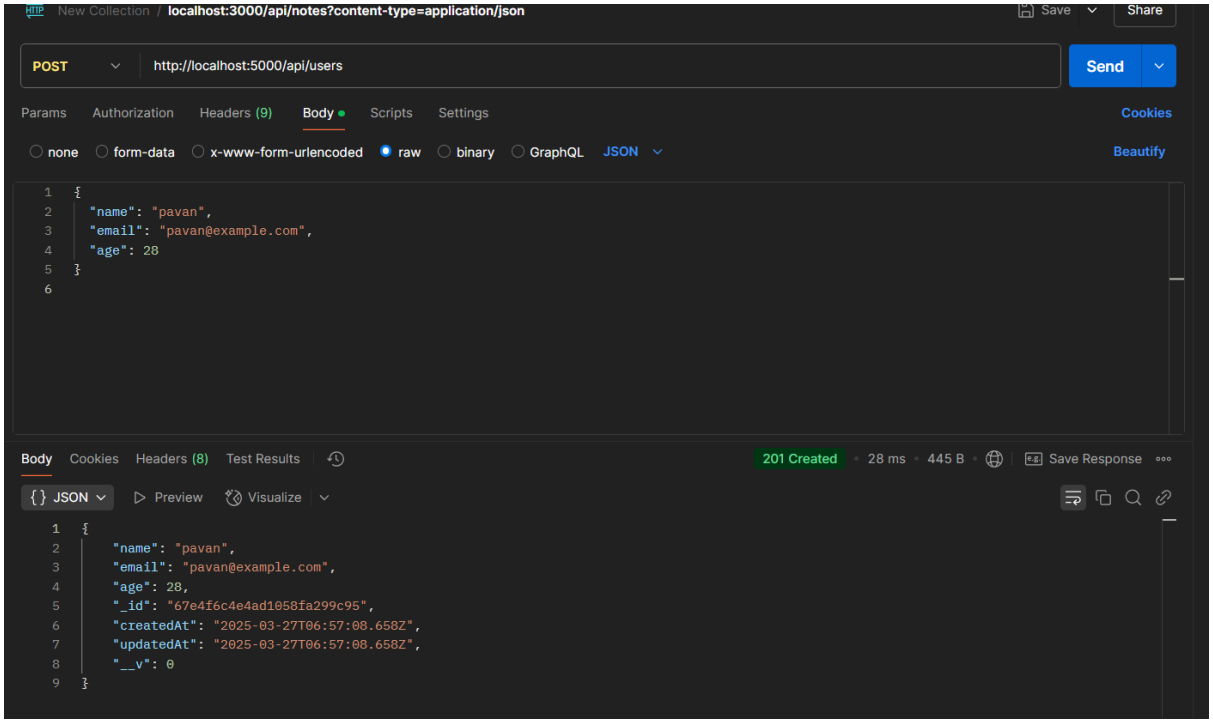
http://localhost:5000/api/users

Now Go to the **Body** tab → Select **raw** → Choose **JSON and** Enter this JSON data:

```
{
  "name": "Pavan",
  "email": "Pavan@example.com",
  "age": 25
}
```

Click **Send**.You should see a **201 Created** response with the new user data.

Output:

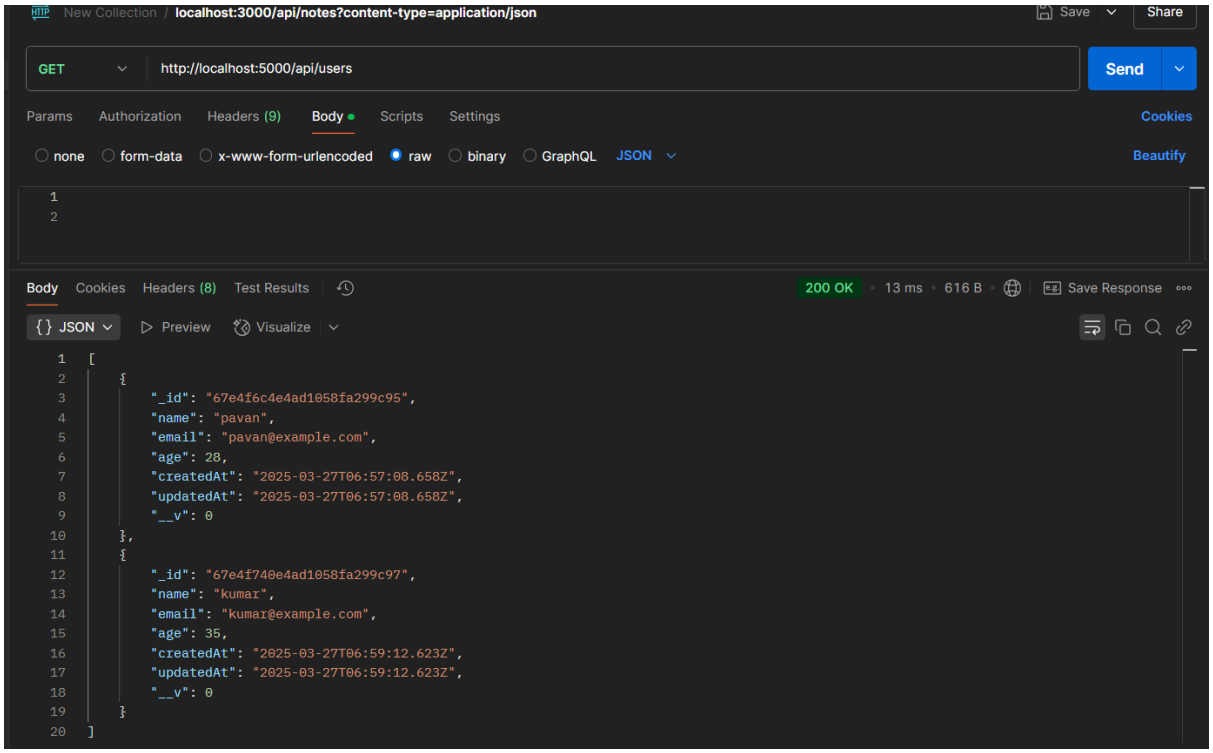


Insert as many users as you can.

To Get All Users , Select **GET and** Enter the URL:

http://localhost:5000/api/users

Click **Send**. You should see a **200 OK** response with a list of users.



To Get a User by ID , Select **GET** and Enter the URL

http://localhost:5000/api/users/{userId}

(replace {userId} with an actual ID from the database). Click **Send**. You should see the user details.

To Update a User , Select **PUT** Enter the URL

http://localhost:5000/api/users/{userId}

(replace {userId} with an actual user ID). In the postman, go to the **Body** tab → Select **raw** → Choose **JSON**. Enter updated user details.

```
{
  "name": "Vinod",
  "email": "vinod@example.com",
  "age": 28
}
```

click **Send**. You should see the updated user details.

To Delete a User , Select **DELETE and** Enter the URL

http://localhost:5000/api/users/{userId}

(replace {userId} with an actual ID):

click **Send**. You should see a success message.

Exp8c: **Aim:** Write a program to explain session management using cookies.

Program:

After Installing node.js, Create a folder for Express project and navigate into it. Initialize your project and create a package.json file. This can be done using following commands

```
Mkdir Exp8c
cd Exp8c
npm init -y
```

Now Install the required packages

```
npm install express cookie-parser
```

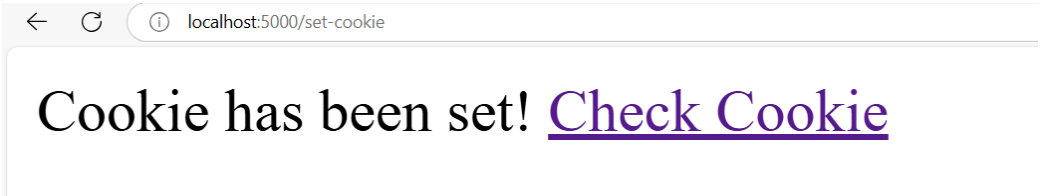
Now create server.js

```
require('dotenv').config();
const express = require('express');
const cookieParser = require('cookie-parser');

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

// Middleware to parse cookies
app.use(cookieParser());

//Set Cookie with Secure Options
app.get('/set-cookie', (req, res) => {
   res.cookie('user', 'Pavan', {
      maxAge: 60000,   // Cookie expires in 60 seconds
      httpOnly: false, // Allow access in browser
      secure: false,   // Set to true if using HTTPS
      sameSite: 'Lax'  // Controls cross-site cookie behavior
   });
   res.send('Cookie has been set! <a href="/get-cookie">Check Cookie</a>');
});

//  Get Cookie
app.get('/get-cookie', (req, res) => {
   const user = req.cookies.user;
   if (user) {
      res.send(` User: ${user} <br> <a href="/delete-cookie">Delete Cookie</a>`);
   } else {
      res.send('No session found! <a href="/set-cookie">Set Cookie</a>');
   }
});

//  Delete Cookie
```

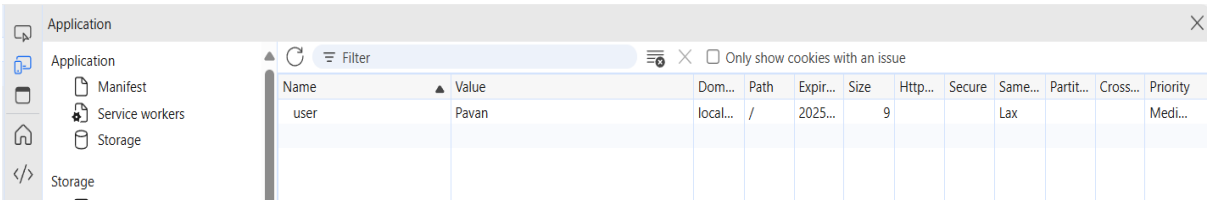
```
app.get('/delete-cookie', (req, res) => {
    res.clearCookie('user');
    res.send('Cookie deleted! <a href="/get-cookie">Check Again</a>');
});

// Start Server
app.listen(PORT, () => console.log(` Server running on http://localhost:${PORT}`));
```

Now start the application by typing node server.js . Open the browser and type the command localhost:5000/set-cookie . You can also see

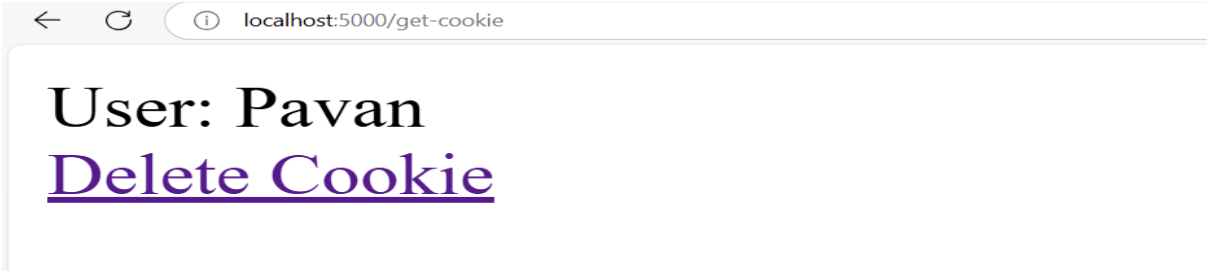
Output:



You can see inside the browser also. Open DevTools and see that cookie has been set.



You can see the cookie by typing localhost:5000/get-cookie



Remember that here cookie was set only for 60 seconds.

Exp8d: Aim : Write a program to explain session management using sessions.

Program:

After Installing node.js, Create a folder for Express project and navigate into it. Initialize your project and create a package.json file. This can be done using following commands

```
Mkdir Exp8d
cd Exp8d
npm init -y
```

Now Install the required packages

```
npm install express express-session cors
```

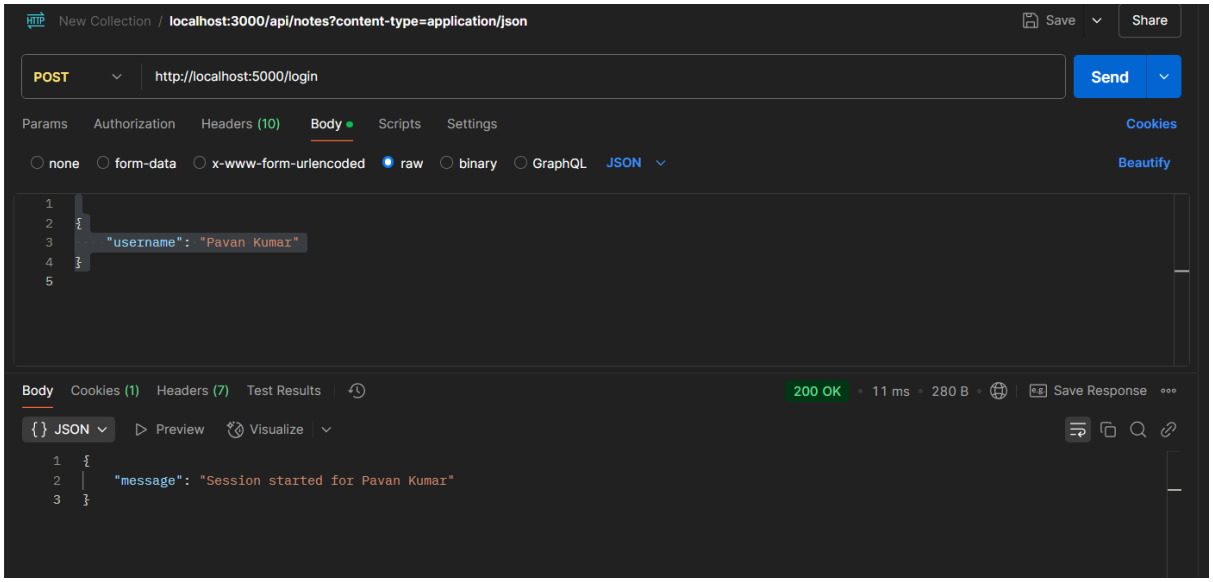
Now create server.js

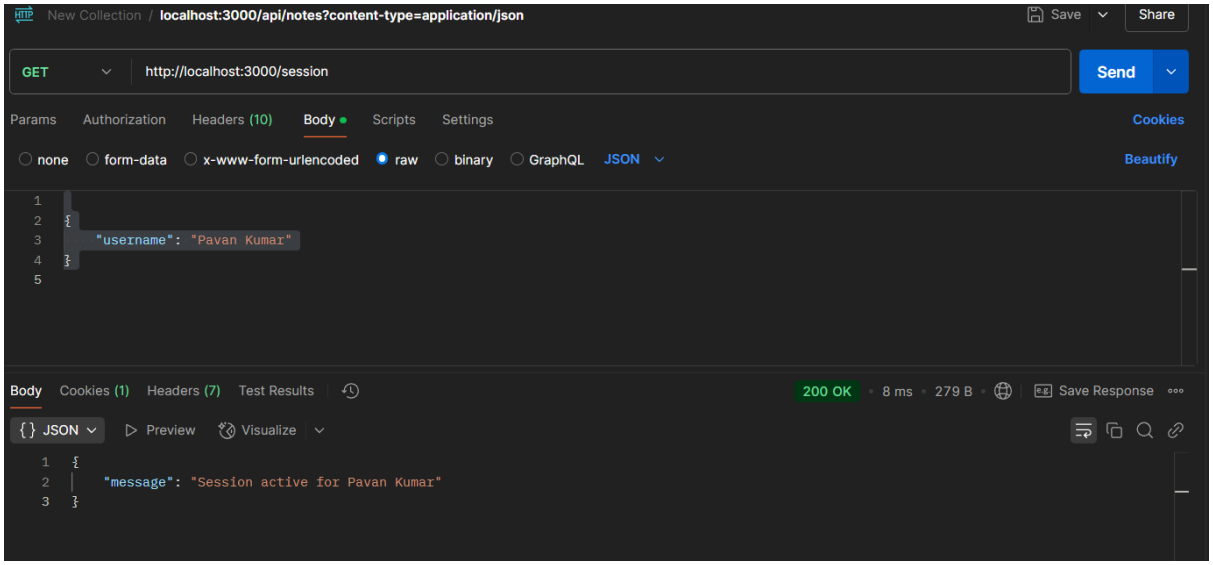
```
const express = require('express');
const session = require('express-session');

const app = express();
const PORT = 5000;

// Middleware
app.use(express.json());  // To handle JSON requests

// Configure Sessions
app.use(session({
    secret: 'mysecretkey',   // Secret key for signing the session ID
    resave: false,           // Prevents session from saving if not modified
    saveUninitialized: true, // Save new sessions
    cookie: { maxAge: 60000 } // Session expires in 1 minute
}));

//  Login (Start Session)
app.post('/login', (req, res) => {
    const { username } = req.body;

    if (!username) {
        return res.status(400).json({ message: "Username is required!" });
    }

    req.session.user = username; // Store username in session
    res.json({ message: `Session started for ${username}` });
});
```

```
// Check Session
app.get('/session', (req, res) => {
    if (!req.session.user) {
        return res.status(401).json({ message: "No active session!" });
    }
    res.json({ message: `Session active for ${req.session.user}` });
});

// Logout (Destroy Session)
app.post('/logout', (req, res) => {
    req.session.destroy(err => {
        if (err) {
            return res.status(500).json({ message: "Logout failed!" });
        }
        res.json({ message: "Session ended successfully!" });
    });
});

// Start Server
app.listen(PORT, () => console.log(` Server running on http://localhost:${PORT}`));
```

Now run the server by using the command node server.js. In the postman , we can check it.

Output:





Exp8e: **Aim:** Implement security features using Helmet in the application

Program:

After Installing node.js, Create a folder for Express project and navigate into it. Initialize your project and create a package.json file. This can be done using following commands

```
Mkdir Exp8e
cd Exp8e
npm init -y
```

Now Install the required packages

```
npm install express helmet
```

Now create app.js file

```
//app.js
const express = require('express');
const routing = require('./route');
const app = express();
app.use('/', routing);
app.listen(3000);
console.log('Server listening in port 3000');
```

Create route.js file

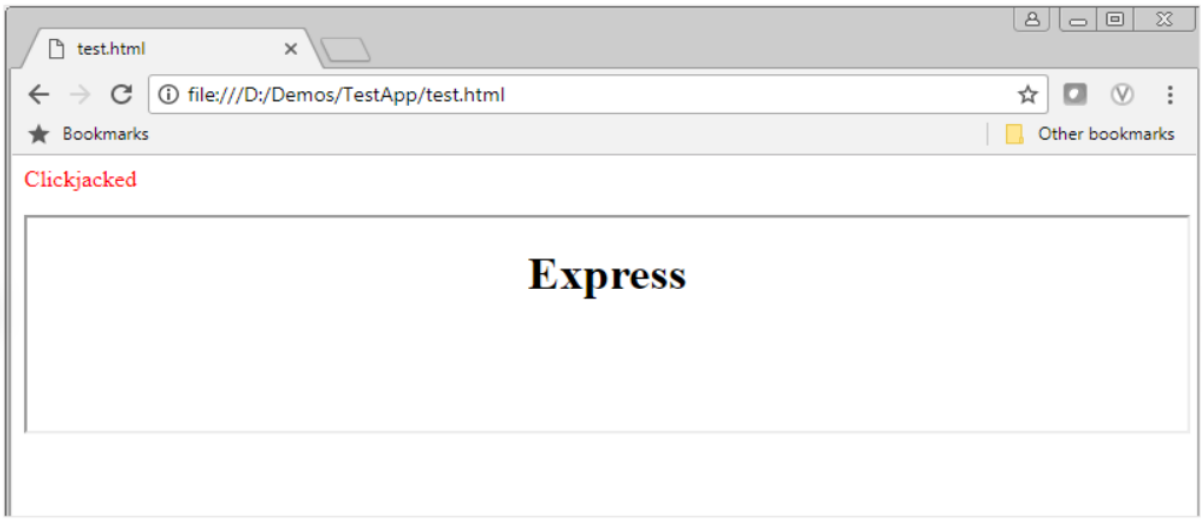
```
//route.js
const express = require('express');
const router = express.Router();
router.get('/', function (req, res) {
  res.send('<h1>Express</h1>');
```

```
});
router.get('/about', function (req, res) {
  res.send('About Us Page');
});
module.exports = router;
```

Create a file test.html inside the folder to demonstrate a clickjacking attack.

```
//test.html file
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <style>
    p {
      color: red;
    }
    iframe {
      width: 100%;
      height: 90%
    }
  </style>
</head>
<body>
  <p>Clickjacked</p>
  <iframe src="http://localhost:3000"></iframe>
</body>
</html>
```

Now start the server by typing node app.js command.



Thus, the page gets loaded in the iframe.

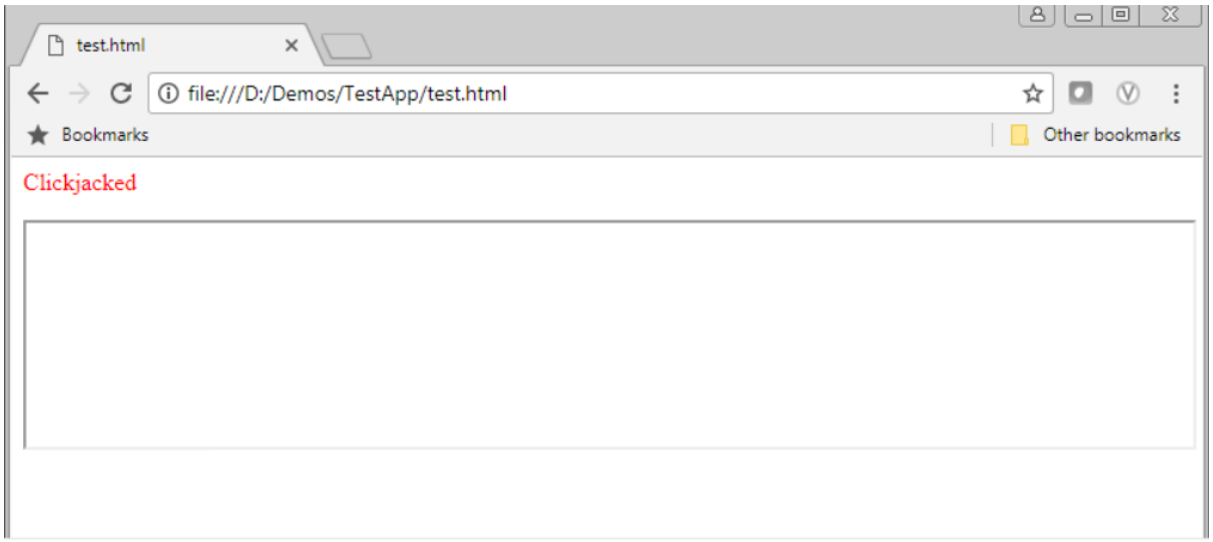
Modify the app.js file by adding the helmet configuration as shown below.

```
//app.js after adding helmet configuaration
const express = require('express');
const helmet = require('helmet');
const routing = require('./route');
const app = express();
app.use(helmet());
app.use('/', routing);
app.listen(3000);
console.log('Server listening in port 3000');
```

Now stop the server and start the server again. Access the URL "http://localhost:3000/" and observe the output.

The helmet then sets an 'X-Frame-Options' header to value "SAMEORIGIN', which instructs the browser to not allow framing from other domains.

After clearing the browser cache, open the page test.html and observe the output.



The page fails to load in the iframe due to the headers set by helmet middleware.

Experiment 9:

Exp9a:

Aim: On the page, display the price of the mobile-based in three different colors. Instead of using the number in our code, represent them by string values like GoldPlatinum, PinkGold, SilverTitanium.

Program:

Create index.html file and script.ts file

```
//index.html file
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Samsung Galaxy Note 7</title>
   <style>
      body {
         font-family: sans-serif;
         display: flex;
         justify-content: center;
         align-items: flex-start;
         padding: 20px;
         background-color: #f4f4f4;
      }

      .product-container {
         background-color: white;
         padding: 20px;
         border-radius: 8px;
         box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
         display: flex;
         gap: 30px;
      }

      .image-section {
         text-align: center;
      }

      .product-image {
         max-width: 200px;
         height: auto;
         margin-bottom: 15px;
      }

      .details-section {
         display: flex;
         flex-direction: column;
      }
```

```
.color-options {
    display: flex;
    gap: 15px;
    margin-bottom: 20px;
}

.color-option-container {
    display: flex;
    flex-direction: column;
    align-items: center;
}

.color-option {
    cursor: pointer;
    border-radius: 50%;
    width: 30px;
    height: 30px;
    border: 2px solid #ccc;
    box-sizing: border-box;
    margin-bottom: 5px;
}

.color-option.selected {
    border-color: #007bff; /* Highlight selected color */
}

.color-option[data-color="Gold Platinum"] {
    background-color: #FFD700; /* Gold color */
}

.color-option[data-color="Pink Gold"] {
    background-color: #FFC0CB; /* Pink color */
}

.color-option[data-color="Silver Titanium"] {
    background-color: #D3D3D3; /* Silver color */
}

.color-name {
    font-size: 0.8em;
    color: #777;
    text-align: center;
    margin-bottom: 2px;
}

.color-price {
    font-size: 0.9em;
    color: #555;
    font-weight: normal; /* Default font weight */
```

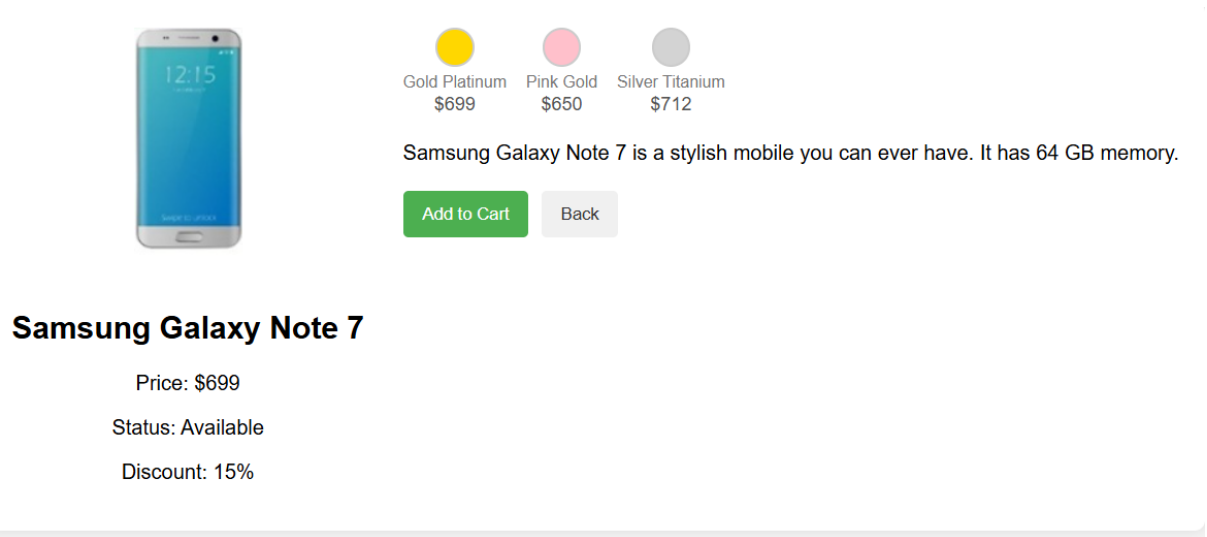
```
            text-align: center; /* Center the price text */
        }

        .color-price.selected-price {
            font-weight: bold; /* Bold for selected price */
        }

        .product-description {
            margin-bottom: 20px;
        }

        .actions {
            display: flex;
            gap: 10px;
        }

        .add-to-cart {
            background-color: #4CAF50;
            color: white;
            padding: 10px 15px;
            border: none;
            border-radius: 4px;
            cursor: pointer;
        }

        .back-button {
            background-color: #f0f0f0;
            color: #333;
            padding: 10px 15px;
            border: none;
            border-radius: 4px;
            cursor: pointer;
        }
    </style>
</head>
<body>
    <div class="product-container">
        <div class="image-section">
            <img class="product-image" src="./mobile1.jpg" alt="Samsung Galaxy Note 7">
            <h2>Samsung Galaxy Note 7</h2>
            <p>Price: $699</p>
            <p>Status: Available</p>
            <p>Discount: 15%</p>
        </div>
        <div class="details-section">
            <div class="color-options">
                <div class="color-option-container">
                    <div class="color-option" data-color="Gold Platinum" data-
price="699"></div>
```

```
            <div class="color-name" data-color-display="Gold Platinum">Gold Platinum</div>
            <div class="color-price" data-color-display="Gold Platinum">$699</div>
          </div>
          <div class="color-option-container">
            <div class="color-option" data-color="Pink Gold" data-price="650"></div>
            <div class="color-name" data-color-display="Pink Gold">Pink Gold</div>
            <div class="color-price" data-color-display="Pink Gold">$650</div>
          </div>
          <div class="color-option-container">
            <div class="color-option" data-color="Silver Titanium" data-price="712"></div>
            <div class="color-name" data-color-display="Silver Titanium">Silver Titanium</div>
            <div class="color-price" data-color-display="Silver Titanium">$712</div>
          </div>
        </div>
        <div class="product-description">
          Samsung Galaxy Note 7 is a stylish mobile you can ever have. It has 64 GB memory.
        </div>
        <div class="actions">
          <button class="add-to-cart">Add to Cart</button>
          <button class="back-button">Back</button>
        </div>
      </div>
    </div>

    <script src="script.ts"></script>
</body>
</html>
```

```
//script.ts file
document.addEventListener('DOMContentLoaded', () => {
  const colorOptions = document.querySelectorAll('.color-option');
  const colorPrices = document.querySelectorAll('.color-price');

  let selectedColor: string | null = null;
  let currentPrice: number | null = null;

  // Set initial selected color and price (default to the first option)
  const firstColorOption = colorOptions[0] as HTMLElement;
  if (firstColorOption) {
      firstColorOption.classList.add('selected');
      selectedColor = firstColorOption.dataset.color || null;
      currentPrice = parseFloat(firstColorOption.dataset.price || '0');
      updateDisplayedPrice(selectedColor);
  }
```

```
  colorOptions.forEach(option => {
    const htmlOption = option as HTMLElement;
    htmlOption.addEventListener('click', () => {
      // Remove selection from previously selected option
      colorOptions.forEach(opt => (opt as HTMLElement).classList.remove('selected'));

      // Add selection to the clicked option
      htmlOption.classList.add('selected');
      selectedColor = htmlOption.dataset.color || null;
      currentPrice = parseFloat(htmlOption.dataset.price || '0');
      updateDisplayedPrice(selectedColor);
    });
  });

  function updateDisplayedPrice(selectedColor: string | null) {
    colorPrices.forEach(priceElement => {
      const htmlPriceElement = priceElement as HTMLElement; // Type cast here
      const colorName = htmlPriceElement.dataset.colorDisplay;
      if (colorName === selectedColor) {
        htmlPriceElement.classList.add('selected-price');
      } else {
        htmlPriceElement.classList.remove('selected-price');
      }
    });
  }

  const backButton = document.querySelector('.back-button');
  if (backButton) {
    backButton.addEventListener('click', () => {
      // In a real application, you would navigate back to the previous page.
      alert('Going back to the previous page (not implemented in this example).');
    });
  }

  const addToCartButton = document.querySelector('.add-to-cart');
  if (addToCartButton) {
    addToCartButton.addEventListener('click', () => {
      if (selectedColor && currentPrice) {
                alert(`Added Samsung Galaxy Note 7 (${selectedColor}) to cart for
$${currentPrice.toFixed(2)}!`);
        // In a real application, you would add the item to the shopping cart.
      } else {
        alert('Please select a color.');
      }
    });
  }
});
```

Upload the image in the same folder and execute the command tsc script.ts file. Then run the index.html to see the output.



Exp9b: Aim:

Define an arrow function inside the event handler to filter the product array with the selected product object using the productId received by the function.

Program:

Create script.ts file as follows.

```
//script.ts
type Product = {
  id: number;
  name: string;
  price: number;
};

const products: Product[] = [
  { id: 1, name: "Laptop", price: 1000 },
  { id: 2, name: "Smartphone", price: 500 },
  { id: 3, name: "Tablet", price: 300 },
];

const handleProductSelection = (productId: number) => {
```

```
  const filteredProduct = products.filter((product) => product.id === productId);
  console.log(filteredProduct);
};

// Example usage
handleProductSelection(2);
```

Now compile the script.ts file into script.js file using tsc script.ts command. Now to see the output you can run the command node script.js.

Output:

```
PS E:\MEAN STACK\meanstacklab\Experiment9\Exp9b> node script.js
[ { id: 2, name: 'Smartphone', price: 500 } ]
```

Exp9c:

Aim: Consider that developer needs to declare a function - getMobileByManufacturer which accepts a string as an input parameter and returns the list of mobiles.

Program:

```
//script.ts file
// declaring a function which accepts string datatype as parameter and returns string array
function getMobileByManufacturer(manufacturer: string): string[] {

    let mobileList: string[];

    if (manufacturer === 'Samsung') {
        mobileList = ['Samsung Galaxy S6 Edge', 'Samsung Galaxy Note 7',
'Samsung Galaxy J7 SM-J700F'];
        return mobileList;
    } else if (manufacturer === 'Apple') {
    mobileList = ['Apple iPhone 5s', 'Apple iPhone 6s ', 'Apple iPhone 7'];
    return mobileList;
    } else {
    mobileList = ['Nokia 105', 'Nokia 230 Dual Sim'];
    return mobileList;
    }
}

// logic to populate the Samsung manufacturer details on console
console.log('The available mobile list: ' + getMobileByManufacturer('Nokia'));
```

Now compile the script.ts file to script.js file using the command tsc script.ts. Now run the script.js file using the command node script.js to see the output.

Output:

```
PS E:\MEAN STACK\meanstacklab\Experiment9\Exp9c> node script.js
The available mobile list: Nokia 105,Nokia 230 Dual Sim
```

Exp9d:

Aim: Consider that developer needs to declare a manufacturer's array holding 4 objects with id and price as a parameter. The developer needs to implement an arrow function - myfunction to populate the id parameter of manufacturers array whose price is greater than or equal to 200 dollars.

Program:

```
//script.ts file
// declaring an Array with 3 objects
const manufacturers = [{ id: 'Samsung', price: 150 },
    { id: 'Microsoft', price: 200 },
    { id: 'Apple', price: 400  },
    { id: 'Micromax', price: 100  }
];

let test:any;

// Arrow function to populate the details of Array whose price is greater than 200
function myFunction() {
test = manufacturers.filter((manufacturer) => manufacturer.price >= 200);
}

// self-invoking an arrow function
myFunction();

console.log('Details of Manufacturer array are : ');

// logic to populate the manufacturer array details based on id value
for (const item of test) {
console.log(item.id);
}
```

Now compile the file script.ts file to script.js using tsc script.ts command. Now execute the file script.js using the command node script.js to see the output.

Output:

```
PS E:\MEAN STACK\meanstacklab\Experiment9\Exp9d> node script.js
Details of Manufacturer array are :
Microsoft
Apple
```

Exp9e:

Aim: Consider that developer needs to declare a function - getMobileByManufacturer with two parameters namely manufacturer and id, where manufacturer value should pass as Samsung and the id parameter should be optional while invoking the function. If the id is passed as 101 then this function should return Moto mobile list. If manufacturer parameter is either Samsung/Apple then this function should return the respective mobile list, similar to making Samsung as default Manufacturer.

Sol:

```
//script.ts file
// declaring a function with optional parameter
function getMobileByManufacturer(manufacturer: string = 'Samsung', id?: number): string[]
{

   let mobileList: string[];

   // logic to be evaluated if id parameter while invoking above declared function
   if (id) {
      if (id === 101) {
         mobileList = ['Moto G Play, 4th Gen', 'Moto Z Play with Style Mod'];
         return mobileList;
      }
   }

   // logic to return mobileList based on manufacturer category
   if (manufacturer === 'Samsung') {
      mobileList = [' Samsung Galaxy S6 Edge', ' Samsung Galaxy Note 7',
         ' Samsung Galaxy J7 SM-J700F'];
      return mobileList;
   } else if (manufacturer === 'Apple') {
      mobileList = [' Apple iPhone 5s', ' Apple iPhone 6s', ' Apple iPhone 7'];
      return mobileList;
   } else {
      mobileList = [' Nokia 105', ' Nokia 230 Dual Sim'];
      return mobileList;
   }

}

// statement to invoke function with optional parameter
console.log('The available mobile list : ' + getMobileByManufacturer('Apple'));

// statement to invoke function with default parameter
console.log('The available mobile list : ' + getMobileByManufacturer(undefined, 101));
```

Now compile the file script.ts file to script.js using tsc script.ts command. Now execute the file script.js using the command node script.js to see the output.

Output:

```
PS E:\MEAN STACK\meanstacklab\Experiment9\Exp9e> node script.js
The available mobile list :  Apple iPhone 5s, Apple iPhone 6s, Apple iPhone 7
The available mobile list : Moto G Play, 4th Gen,Moto Z Play with Style Mod
```

Experiment 10:

Exp10a:

Aim: Consider that developer needs to implement business logic for adding multiple Product values into a cart variable which is type of string array.

Program:

Create a file script.ts

```
//script.ts file
// declaring a empty string array
const cart: string[] = [];

// arrow function logic to push values into cart array
const pushtoCart = (item: string) => { cart.push(item); };

// logic to add items into cart
function addtoCart(...productName: string[]): string[] {

    for (const item of productName) {
        pushtoCart(item);
    }
    return cart;

}

// to populate value on console
console.log('Cart Items are:' + addtoCart(' Moto G Play, 4th Gen', ' Apple iPhone 5s'));
```

Now compile the file script.ts file to script.js using tsc script.ts command. Now execute the file script.js using the command node script.js to see the output.

Output:

```
PS E:\MEAN STACK\meanstacklab\Experiment10\Exp10a> tsc script.ts
PS E:\MEAN STACK\meanstacklab\Experiment10\Exp10a> node script.js
Cart Items are: Moto G Play, 4th Gen, Apple iPhone 5s
```

Exp10b:

Aim:

Declare an interface named - Product with two properties like productId and productName with a number and string datatype. The developer needs to implement logic to populate the Product details using this interface.

Program:

//script.ts file

```
// declaring an interface
interface Product {
    productId: number ;
    productName: string ;
}

// logic to display the Product details with interface object as parameter
function getProductDetails(prodobj: Product): string {
    return 'The product name is : ' + prodobj.productName;
}

// declaring a variable having interface properties
const product1 = {productId: 1001, productName: 'Mobile'};

// declaring variable and invoking Product details function
const productDetails: string = getProductDetails(product1);

// line to populate the created product on console
console.log(productDetails);
```

Now compile the file script.ts file to script.js using tsc script.ts command. Now execute the file script.js using the command node script.js to see the output.

Output:

```
PS E:\MEAN STACK\meanstacklab\Experiment10\Exp10b> tsc script.ts
PS E:\MEAN STACK\meanstacklab\Experiment10\Exp10b> node script.js
The product name is : Mobile
```

Exp10c:

Aim: Declare an interface named - Product with two properties like productId and productName with the number and string datatype and need to implement logic to populate the Product details. **Use Duck Typing**

Program:

Consider the script.ts file

```
//script.ts file

// declaring an interface
interface Product {
    productId: number;
    productName: string;
}

// logic to display the Product details with interface object as parameter
function getProductDetails(prodobj: Product): string {
    return 'The product name is : ' + prodobj.productName;
}

// declaring a variable along with interface properties
const product2 = {productId: 1001, productName: 'Mobile', productCategory: 'Gadget'};

// declaring variable and invoking Product details function
const productDetails: string = getProductDetails(product2);

// line to populate the created product variable on console
console.log(productDetails);
```

Now compile the file script.ts file to script.js using tsc script.ts command. Now execute the file script.js using the command node script.js to see the output.

Output:

```
PS E:\MEAN STACK\meanstacklab\Experiment10\Exp10c> tsc script.ts
PS E:\MEAN STACK\meanstacklab\Experiment10\Exp10c> node script.js
The product name is : Mobile
```

Exp10d:

Aim: Declare an interface with function type and access its value.

Program:

Consider the script.ts file

```
//script.ts file
// Declaring an interface with a function type
interface StringGenerator {
    (name: string, id: number): string;
  }

  // Implementing the function that matches the interface using a normal function
  const createCustomerID: StringGenerator = function(name: string, id: number): string {
    return `The customer name is ${name} and id is ${id}`;
  };

  // Using the interface reference to call the function
  const customerId: string = createCustomerID('Pavan Kumar', 101);

  // Outputting the customer details
  console.log(customerId);
```

Now compile the file script.ts file to script.js using tsc script.ts command. Now execute the file script.js using the command node script.js to see the output.

Output:

```
PS E:\MEAN STACK\meanstacklab\Experiment10\Exp10d> tsc script.ts
PS E:\MEAN STACK\meanstacklab\Experiment10\Exp10d> node script.js
The customer name is Pavan Kumar and id is 101
```

Experiment 11:

Exp11a:

Aim: Declare a productList interface which extends properties from two other declared interfaces like Category, Product as well as implementation to create a variable of this interface type.

Program:

Consider the script.ts file

```
//script.ts file
// Interface for Category
interface Category {
   categoryName: string;
 }

 // Interface for Product
 interface Product {
  productId: number;
  productName: string;
 }

 // Interface for ProductList extending Category and Product
 interface ProductList extends Category, Product {
  quantity: number;
  inStock: boolean;
 }

 // Creating a variable of type ProductList
 const myProduct: ProductList = {
  categoryName: "Electronics",
  productId: 123,
  productName: "Laptop",
  quantity: 10,
  inStock: true,
 };

 // Accessing the properties of the ProductList variable
 console.log ("Category Name:", myProduct. CategoryName);
 console.log ("Product ID:", myProduct. ProductId);
 console.log ("Product Name:", myProduct. ProductName);
 console.log ("Quantity:", myProduct. Quantity);
 console.log ("In Stock:", myProduct. InStock);
```

Now compile the file script.ts file to script.js using tsc script.ts command. Now execute the file script.js using the command node script.js to see the output.

Output:

```
PS E:\MEAN STACK\meanstacklab\Experiment11\Exp11a> tsc script.ts
PS E:\MEAN STACK\meanstacklab\Experiment11\Exp11a> node script.js
Category Name: Electronics
Product ID: 123
Product Name: Laptop
Quantity: 10
In Stock: true
```

Exp11b:

Aim: Create objects of the Product class and place them into the productlist array.

Program:

Consider the script.ts file

```
//script.ts file
// Class for Product
class Product {
    productId: number;
    productName: string;
    categoryName: string;
    quantity: number;
    inStock: boolean;
  }

  // Array to hold Product objects
  const productList: Product[] = [];

  // Create Product objects by directly assigning properties
  const product1: Product = {
    productId: 101,
    productName: "Smartphone",
    categoryName: "Electronics",
    quantity: 50,
    inStock: true,
  };

  const product2: Product = {
    productId: 205,
    productName: "The Great Gatsby",
    categoryName: "Books",
    quantity: 20,
    inStock: true,
  };


  // Place the objects into the productList array
  productList. push(product1);
```

```
productList. push(product2);


// Log the productList array to see the objects
console.log(productList);
```

Now compile the file script.ts file to script.js using tsc script.ts command. Now execute the file script.js using the command node script.js to see the output.

Output:

```
PS E:\MEAN STACK\meanstacklab\Experiment11\Exp11b> tsc script.ts
PS E:\MEAN STACK\meanstacklab\Experiment11\Exp11b> node script.js
[
  {
    productId: 101,
    productName: 'Smartphone',
    categoryName: 'Electronics',
    quantity: 50,
    inStock: true
  },
  {
    productId: 205,
    productName: 'The Great Gatsby',
    categoryName: 'Books',
    quantity: 20,
    inStock: true
  }
]
```

Exp11c:

Aim: Declare a class named - Product with the below-mentioned declarations: (i) productId as number property (ii) Constructor to initialize this value (iii) getProductId method to return the message "Product id is <>".

Program:
Consider the script.ts file

```
//script.ts file
class Product {
   productId: number;

   constructor(id: number) {
     this.productId = id;
   }
```

```
  getProductId (): string {
    return `Product id is ${this. productId}`;
  }
}

// Example usage:
const myProduct = new Product(123);
console.log (myProduct.getProductId()); // Output: Product id is 123
```

Now compile the file script.ts file to script.js using tsc script.ts command. Now execute the file script.js using the command node script.js to see the output.

Output:

```
PS E:\MEAN STACK\meanstacklab\Experiment11\Exp11c> tsc script.ts
PS E:\MEAN STACK\meanstacklab\Experiment11\Exp11c> node script.js
Product id is 123
```

Exp11d:

Aim: create a Product class with 4 properties namely productId, productName, productPrice, productCategory with private, public, static, and protected access modifiers and accessing them through Gadget class and its methods.

Program:

Consider the script.ts file

```
//script.ts
// declaring a Product class with access modifiers
class Product {
    static productPrice = 150;
    private productId: number;
    public productName: string;
    protected productCategory: string;
    // declaration of constructor with 3 parameters
    constructor (productId: number, productName: string, productCategory: string) {
        this. productId = productId;
        this. productName = productName;
        this. productCategory = productCategory;
    }
    // method ot display product id details
    getProductId () {
        console.log ('The Product id is: ' + this. productId);
    }
}

// declaring a Gadget class extending the properties from Product class
class Gadget extends Product {
```

```
    // method to display productCategory property
    getProduct (): void {
        console.log ('Product category is : ' + this. productCategory);
    }
}
// Gadget Class object creation
const g: Gadget = new Gadget(1234, 'Mobile', 'SmartPhone');
// invoking getProduct method with the help of Gadget object
g.getProduct();

// invoking getProductId method with the help of Gadget object
g.getProductId();

// line to populate product name property with Gadget object
console.log ('Product name is : ' + g.productName);

// line to populate static property product price directly with Product class name
console.log ('Product price is: $' + Product.productPrice);
```

Now compile the file script.ts file to script.js using tsc script.ts command. Now execute the file script.js using the command node script.js to see the output.

Output:

```
PS E:\MEAN STACK\meanstacklab\Experiment11\Exp11d> tsc script.ts
PS E:\MEAN STACK\meanstacklab\Experiment11\Exp11d> node script.js
Product category is : SmartPhone
The Product id is: 1234
Product name is : Mobile
Product price is: $150
```

Experiment 12:

Exp12a:

Aim: Create a Product class with properties namely productId and methods to setProductId() and getProductId()

Program:

Consider the script.ts file

```
//script.ts file
class Product {
    private productId: number;

    constructor (id: number) {
      this.productId = id;
    }

    // Method to set the productId
    setProductId(newId: number): void {
      this.productId = newId;
    }

    // Method to get the productId
    getProductId(): number {
      return this.productId;
    }
  }

  const myProduct = new Product(100);
  console.log ("Initial Product ID:", myProduct.getProductId());
  myProduct.setProductId(2000);
  console.log ("Updated Product ID:", myProduct.getProductId());
                        // Output: Updated Product ID: 2000
```

Now compile the file script.ts file to script.js using tsc script.ts command. Now execute the file script.js using the command node script.js to see the output.

Output:

```
PS E:\MEAN STACK\meanstacklab\Experiment12\Exp12a> tsc script.ts
PS E:\MEAN STACK\meanstacklab\Experiment12\Exp12a> node script.js
Initial Product ID: 100
Updated Product ID: 2000
```

Exp12b:

Aim: Create a namespace called ProductUtility and place the Product class definition in it. Import the Product class inside productlist file and use it.

Program:

Create the file product.ts file and create namespace Productutility with class Productclass in it as shown below.

```
//product.ts
namespace ProductUtility {
    export class Productclass {
      public productId: number;
      public productName: string;

      constructor(id: number, name: string) {
        this.productId = id;
        this.productName = name;
      }

    }
  }
```

Now import the Product class from the namespace ProductUtility in another file productList.ts file as shown below.

```
//productList.ts file
/// <reference path= “. /product.ts" />

// Using the namespace and the Product class within it
const productList: ProductUtility.Productclass[] = [];

// Create Product objects using the namespaced class
const product1 = new ProductUtility.Productclass(101, "Laptop");
const product2 = new ProductUtility.Productclass(202, "Book");
const product3 = new ProductUtility.Productclass(303, "T-Shirt");

// Add products to the productList array
productList. push(product1);
productList. push(product2);
productList. push(product3);

// Use the Product objects
productList.forEach(product => {
  console.log (`Product ID: ${product.productId}`);
  console.log (`Product Name: ${product.productName}`);
});
```

Now compile both product.ts and productList.ts into a single JavaScript file named finaloutput.js

tsc --outfile finaloutput.js product.ts productList.ts

now execute the command node finaloutput.js to see the output.

Output:

```
PS E:\MEAN STACK\meanstacklab\Experiment12\Exp12b> tsc --outfile finaloutput.js product.ts productList.ts
PS E:\MEAN STACK\meanstacklab\Experiment12\Exp12b> node finaloutput.js
Product ID: 101
Product Name: Laptop
Product ID: 202
Product Name: Book
Product ID: 303
Product Name: T-Shirt
```

Exp12c:

Aim: Design a TypeScript module for a Mobile Cart application that includes a function to determine the total cost of a product using the quantity and price values and assign it to a totalPrice variable.

Program:

Consider the product.ts file that include a function to determine the total cost of a product.

```
//product.ts file
// Define a class for a Product
class Product {
    name: string;
    price: number;

    constructor(name: string, price: number) {
      this.name = name;
      this.price = price;
    }
  }

  // Function to calculate the total price of a product based on quantity
   function calculateTotalPrice(product: Product, quantity: number): number {
    const totalPrice = product.price * quantity;
    return totalPrice;
  }

  export {Product, calculateTotalPrice};
```

Consider the cart.ts file and import the class and function defined in the product.ts module

```
//cart.ts file
// Import the Product class and the calculateTotalPrice function from the product module
import { Product, calculateTotalPrice } from './product';

// Example usage within the cart module
const item = new Product("Smartphone", 500);
const quantityOrdered = 3;
const totalPriceOfItem = calculateTotalPrice(item, quantityOrdered);
console.log(`Total price for ${quantityOrdered} ${item.name}(s): ${totalPriceOfItem}`);
```

Now compile the file cart.ts file to script.js using tsc cart.ts command. Now execute the file cart.js using the command node cart.js to see the output.

Output:

```
PS E:\MEAN STACK\meanstacklab\Experiment12\Exp12c> tsc cart.ts
PS E:\MEAN STACK\meanstacklab\Experiment12\Exp12c> node cart.js
Total price for 3 Smartphone(s): 1500
```

Exp12d:

Aim: Create a generic array and function to sort numbers as well as string values.

Program:

```
//script.ts file
// Generic array that can hold numbers or strings
const mixedArray: (number | string)[] = [5, "apple", 2, "banana", 8, "cherry"];

// Generic function to sort an array of numbers or strings
function sortArray<T extends number | string>(arr: T[]): T[] {
  return arr.sort();
}

// Sorting the mixed array (TypeScript will infer the type)
const sortedMixedArray = sortArray(mixedArray);
console.log("Sorted Mixed Array:", sortedMixedArray);

// Creating a number array
const numberArray: number[] = [5, 2, 8, 1, 9];
const sortedNumberArray = sortArray(numberArray);
console.log("Sorted Number Array:", sortedNumberArray);

// Creating a string array
const stringArray: string[] = ["apple", "banana", "cherry", "date"];
const sortedStringArray = sortArray(stringArray);
console.log("Sorted String Array:", sortedStringArray);
```

Now compile the file script.ts file to script.js using tsc script.ts command. Now execute the file script.js using the command node script.js to see the output.

Output:

```
PS E:\MEAN STACK\meanstacklab\Experiment12\Exp12d> tsc script.ts
PS E:\MEAN STACK\meanstacklab\Experiment12\Exp12d> node script.js
Sorted Mixed Array: [ 2, 5, 8, 'apple', 'banana', 'cherry' ]
Sorted Number Array: [ 1, 2, 5, 8, 9 ]
Sorted String Array: [ 'apple', 'banana', 'cherry', 'date' ]
```