SRM VALLIAMMAI ENGINEERING COLLEGE

(An Autonomous Institution)

S.R.M. Nagar, Kattankulathur -603203.

DEPARTMENT OF CYBER SECURITY

CS3567- WEB AND FULL STACK DEVELOPMENT LABORATORY

LAB MANUAL

Regulation 2023

III YEAR / V SEMESTER

Prepared By,
Mr. E.RAJKUMAR., AP (O.G) /CYS

SYLLABUS

CS3567 WEB AND FULL STACK DEVELOPMENT LABORATORY L T P C

0 0 3 1.5

COURSE OBJECTIVES:

- To develop full stack applications with clear understanding of user interface, business logic and data storage.
- To design and develop user interface screens for a given scenario
- To develop the functionalities as web components as per the requirements
- To implement the database according to the functional requirements
- To integrate the user interface with the functionalities and data storage.

LIST OF EXPERIMENTS

The Instructor can choose the technology stack to develop the following full stack experiments based on the Full Stack Web Development Theory Course.

1. Develop a portfolio website for yourself which gives details about yourself for a potential recruiter.
2. Create a web application to manage the TO-DO list of users, where users can login and manage their to-do items
3. Create a simple micro blogging application (like twitter) that allows people to post their content which can be viewed by people who follow them.
4. Create a food delivery website where users can order food from a particular restaurant listed in the website.
5. Develop a classifieds web application to buy and sell used products.
6. Develop a leave management system for an organization where users can apply different types of leaves such as casual leave and medical leave. They also can view the available number of days.
7. Develop a simple dashboard for project management where the statuses of various tasks are available. New tasks can be added and the status of existing tasks can be changed among Pending, In Progress or Completed.
8. Develop an online survey application where a collection of questions is available and users are asked to answer any random 5 questions

TOTAL: 45 PERIODS

COURSE OUTCOMES:

- Design full stack applications with clear understanding of user interface, business logic and data storage.
- Design and develop user interface screens

- Implement the functional requirements using appropriate Tool
- Design and develop database based on the requirements
- Integrate all the necessary components of the application

LIST OF EQUIPMENTS FOR A BATCH OF 30 STUDENTS:

1. Angular JS, React.js /Node.js/ Equivalent, any IDE like IntelliJ/ Eclipse / Visual Studio Code/ Equivalent, Web Server
2. MySQL, MongoDB

INDEX

EX NO: 1 DEVELOP A PORTFOLIO WEBSITE FOR YOURSELF WHICH GIVES DETAILS ABOUT YOURSELF FOR A POTENTIAL RECRUITER.

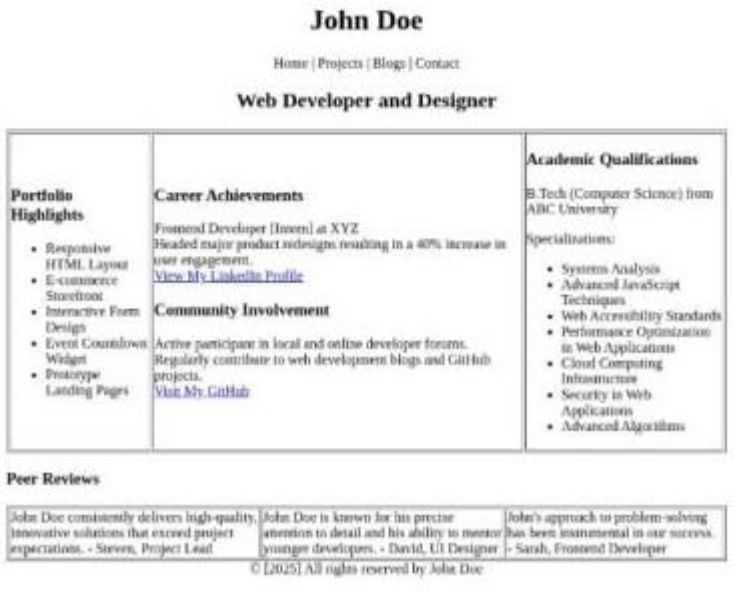
AIM

To design and develop a responsive personal portfolio website using HTML, CSS, and JavaScript, which showcases personal information, skills, projects, and provides a contact form for potential recruiters.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Portfolio of John Doe</title>
</head>
<body>
  <header>
    <h1 style="text-align:center;">John Doe</h1>
    <p style="text-align:center;">Home | Projects | Blogs | Contact</p>
  </header>
  <h2 style="text-align:center;">Web Developer and Designer</h2>
  <table border="1" width="100%">
    <tr>
      <td>
        <h3>Portfolio Highlights</h3>
        <ul>
          <li>Responsive HTML Layout</li>
          <li>E-commerce Storefront</li>
          <li>Interactive Form Design</li>
          <li>Event Countdown Widget</li>
          <li>Prototype Landing Pages</li>
        </ul>
      </td>
      <td>
        <h3>Career Achievements</h3>
        <p>Frontend Developer [Intern] at XYZ<br>Headed major product redesigns resulting in a 40%
increase in user engagement.<br><a href="#">View My LinkedIn Profile</a></p>
        <h3>Community Involvement</h3>
        <p>Active participant in local and online developer forums. Regularly contribute to web
development blogs and GitHub projects.<br><a href="#">Visit My GitHub</a></p>
      </td>
      <td>
        <h3>Academic Qualifications</h3>
        <p>B.Tech (Computer Science) from ABC University</p>
        <p>Specializations:</p>
        <ul>
          <li>Systems Analysis</li>
          <li>Advanced JavaScript Techniques</li>
          <li>Web Accessibility Standards</li>
          <li>Performance Optimization in Web Applications</li>
          <li>Cloud Computing Infrastructure</li>
```

```
                <li>Security in Web Applications</li>
                <li>Advanced Algorithms</li>
            </ul>
        </td>
    </tr>
</table>
<h3>Peer Reviews</h3>
<table border="1" width="100%">
    <tr>
        <td>John Doe consistently delivers high-quality, innovative solutions that exceed project
expectations. - Steven, Project Lead</td>
        <td>John Doe is known for his precise attention to detail and his ability to mentor younger
developers. - David, UI Designer</td>
        <td>John's approach to problem-solving has been instrumental in our success. - Sarah, Frontend
Developer</td>
    </tr>
</table>
<footer style="text-align:center;">
    © [2025] All rights reserved by John Doe
</footer>
</body>
</html>
```

OUTPUT



John Doe

Home | Projects | Blogs | Contact

Web Developer and Designer

Portfolio Highlights • Responsive HTML Layout • E-commerce Storefront • Interactive Form Design • Event Countdown Widget • Prototype Landing Pages	**Career Achievements** Frontend Developer [Intern] at XYZ Headed major product redesigns resulting in a 40% increase in user engagement. View My LinkedIn Profile **Community Involvement** Active participant in local and online developer forums. Regularly contribute to web development blogs and GitHub projects. Visit My GitHub	**Academic Qualifications** B.Tech (Computer Science) from ABC University Specializations: • Systems Analysis • Advanced JavaScript Techniques • Web Accessibility Standards • Performance Optimization in Web Applications • Cloud Computing Infrastructure • Security in Web Applications • Advanced Algorithms

Peer Reviews

John Doe consistently delivers high-quality, innovative solutions that exceed project expectations. - Steven, Project Lead	John Doe is known for his precise attention to detail and his ability to mentor younger developers. - David, UI Designer	John's approach to problem-solving has been instrumental in our success. - Sarah, Frontend Developer

© [2025] All rights reserved by John Doe

Viva Questions:

1. Design Basic layout of Portfolio Website.
2. Give Steps to design Portfolio website.
3. How will you create layout of Portfolio using HTML?
4. How will you Style Portfolio using CSS?
5. How to add responsive based design Portfolio Website?

RESULT

A fully functional, responsive personal portfolio website was successfully developed. The website includes sections for About Me, Skills, Projects, and a Contact form. It is visually appealing, easy to navigate, and serves as an effective digital resume for recruiters and employers.

EX NO: 2 CREATE A WEB APPLICATION TO MANAGE THE TO-DO LIST OF USERS, WHERE USERS CAN LOGIN AND MANAGE THEIR TO-DO ITEMS

AIM

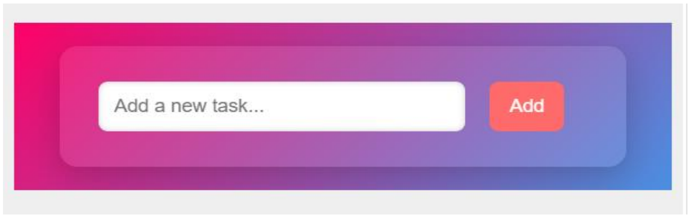
To develop a web application that allows users to securely log in, create, view, update, and delete their personal to-do list items, providing an efficient way to manage daily tasks.

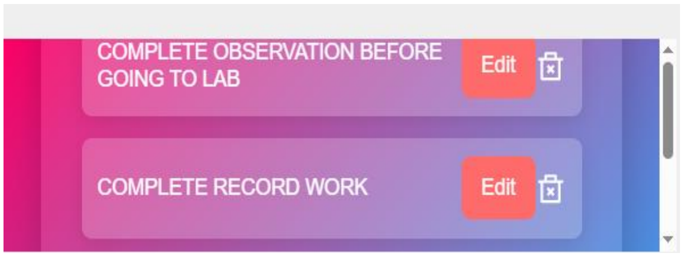
```
<html>
<head>
   <style>
      body {
         font-family: 'Arial', sans-serif;
         background: linear-gradient(135deg, #f06, #4a90e2);
         display: flex;
         justify-content: center;
         align-items: center;
         height: 100vh;
         margin: 0;
         color: #fff;
      }
      .container {
         background: rgba(255, 255, 255, 0.1);
         padding: 2rem;
         border-radius: 15px;
         box-shadow: 0 10px 30px rgba(0, 0, 0, 0.2);
         width: 100%;
         max-width: 400px;
      }
      input[type="text"] {
         width: calc(100% - 100px);
         padding: 0.8rem;
         border-radius: 8px;
         border: none;
         outline: none;
         font-size: 1rem;
         margin-right: 1rem;
         box-shadow: inset 0 0 5px rgba(0, 0, 0, 0.2);
      }
      button {
         padding: 0.8rem 1rem;
         border-radius: 8px;
         border: none;
         background: #ff6b6b;
         color: #fff;
         font-size: 1rem;
         cursor: pointer;
         transition: background 0.3s;
      }
      button:hover {
         background: #ff4757;
      }
```

```
        .todo {
          background: rgba(255, 255, 255, 0.2);
          border-radius: 8px;
          margin-top: 1rem;
          display: flex;
          justify-content: space-between;
          align-items: center;
          padding: 0.8rem;
          box-shadow: 0 5px 15px rgba(0, 0, 0, 0.1);
          transition: transform 0.2s, background 0.3s;
        }
        .todo:hover {
          transform: translateY(-3px);
          background: rgba(255, 255, 255, 0.3);
        }
        .task {
          flex: 1;
          font-size: 1rem;
          cursor: pointer;
        }
        .delete,
        .edit {
          cursor: pointer;
          transition: transform 0.2s, color 0.3s;
        }
        .delete:hover {
          transform: scale(1.1);
          color: #ff4757;
        }
        .edit:hover {
          transform: scale(1.1);
          color: #1e90ff;
        }
        svg {
          fill: currentColor;
        }
    </style>
</head>
<body>
    <div class="container">
        <input type="text" placeholder="Add a new task...">
        <button>Add</button>
        <div id="task-list"></div>
    </div>
    <script>
        let inputs = document.querySelector('input');
        let btn = document.querySelector('button');
        let taskList = document.getElementById('task-list');
        let task = [];
        let localstoragedata = localStorage.getItem("task array");
        if (localstoragedata != null) {
          let ogdata = JSON.parse(localstoragedata);
          task = ogdata;
          maketodo();
        }
        btn.addEventListener("click", function () {
          let query = inputs.value;
          inputs.value = "";
          if (query.trim() === "") {
              alert("no value entered");
              throw new Error("empty input field error");
          }
          let taskObj = {
              id: Date.now(),
              text: query
          }
          task.push(taskObj);
          localStorage.setItem("task array", JSON.stringify(task));
          maketodo();
        });
        function maketodo() {
          taskList.innerHTML = "";
          for (let i = 0; i < task.length; i++) {
              let { id, text } = task[i];
              let element = document.createElement('div');
              element.innerHTML = `
                  <span class="task" contenteditable="false">${text}</span>
                  <button class='edit'>Edit</button>
                  <span class="delete"><svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24" 
width="24" height="24"><path d="M17 6H22V8H20V21C20 21.5523 19.5523 22 19 22H5C4.44772 22 4 
21.5523 4 21V8H2V6H7V3C7 2.44772 7.44772 2 8 2H16C16.5523 2 17 2.44772 17 3V6ZM18 
8H6V20H18V8ZM13.4142 13.9997L15.182 15.7675L13.7678 17.1817L12 15.4139L10.2322 
17.1817L8.81802 15.7675L10.5858 13.9997L8.81802 12.232L10.2322 10.8178L12 12.5855L13.7678 
10.8178L15.182 12.232L13.4142 13.9997ZM9 4V6H15V4H9Z"></path></svg></span>
              `;
              let delbtn = element.querySelector('.delete');
              let editbtn = element.querySelector('.edit');
              let taskText = element.querySelector('.task');
              delbtn.addEventListener("click", function () {
                  let filteredarray = task.filter(function (taskobj) {
                      return taskobj.id != id;
                  });
                  task = filteredarray;
                  localStorage.setItem("task array", JSON.stringify(task));
                  taskList.removeChild(element);
              });
              editbtn.addEventListener("click", function () {
                  if (editbtn.innerText === 'Edit') {
                      taskText.setAttribute('contenteditable', 'true'); // Enable editing
                      taskText.focus(); // Focus on the text to start editing
                      editbtn.innerText = 'Save'; // Change button text to 'Save'
                  } else {
                      taskText.setAttribute('contenteditable', 'false'); // Disable editing
                      let updatedText = taskText.innerText.trim();
                      if (updatedText !== "") {
                          task = task.map(function (taskobj) {
```

```
                  if (taskobj.id === id) {
                      taskobj.text = updatedText;
                  }
                  return taskobj;
              });
              localStorage.setItem("task array", JSON.stringify(task));
          }
          editbtn.innerText = 'Edit'; // Change button text back to 'Edit'
      }
  });
  element.classList.add('todo');
  taskList.appendChild(element);
 }
}
 </script>
</body>
</html>
```

OUTPUT





Viva Questions:

1. What are the key features of To-Do List?

2. What are the step by step implementation of user wish list management?

3. How can the users change the wish to do list management?

4. How can users update the wish list management in future?

5. Can this project be used in marketing applications? If yes how?

RESULT

The to-do list web application was successfully developed with user authentication features. Users can register and log in securely, and manage their to-do items with CRUD (Create, Read, Update, Delete) functionality. The application provides a simple and user-friendly interface to organize tasks efficiently.

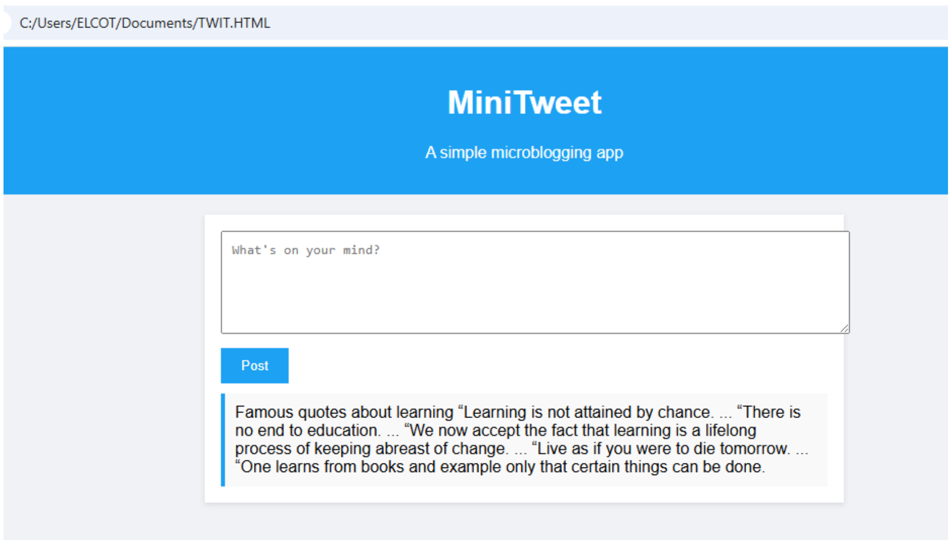
EX NO: 3 CREATE A SIMPLE MICRO BLOGGING APPLICATION (LIKE TWITTER) THAT ALLOWS PEOPLE TO POST THEIR CONTENT WHICH CAN BE VIEWED BY PEOPLE WHO FOLLOW THEM.

AIM

To design and develop a simple microblogging web application (similar to Twitter) that allows users to register, log in, post short content (tweets), follow other users, and view posts from the users they follow.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>MiniTweet - Microblogging App</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background: #f0f2f5;
      margin: 0;
      padding: 0;
    }
    header {
      background: #1DA1F2;
      color: white;
      padding: 1rem;
      text-align: center;
    }
    main {
      max-width: 600px;
      margin: 20px auto;
      padding: 1rem;
      background: white;
      box-shadow: 0 2px 5px rgba(0,0,0,0.1);
    }
    form textarea {
      width: 100%;
      height: 80px;
      padding: 10px;
      margin-bottom: 10px;
    }
    button {
      padding: 10px 20px;
      background: #1DA1F2;
      border: none;
      color: white;
      cursor: pointer;
    }
    .post {
      background: #f9f9f9;
      padding: 10px;
      margin-top: 10px;
      border-left: 4px solid #1DA1F2;
    }
    </style>
  </head>
  <body>
    <header>
      <h1>MiniTweet</h1>
      <p>A simple microblogging app</p>
    </header>

    <main>
      <form id="postForm">
        <textarea id="content" placeholder="What's on your mind?"></textarea>
        <button type="submit">Post</button>
      </form>
      <div id="feed"></div>
    </main>

    <script>
      const form = document.getElementById('postForm');
      const feed = document.getElementById('feed');

      form.addEventListener('submit', function(e) {
        e.preventDefault();
        const content = document.getElementById('content').value;
        if (content.trim() === '') return;

        const post = document.createElement('div');
        post.className = 'post';
        post.textContent = content;

        feed.prepend(post);
        document.getElementById('content').value = '';
      });
    </script>
  </body>
</html>
```

OUTPUT


C:/Users/ELCOT/Documents/TWIT.HTML
MiniTweet
A simple microblogging app
What's on your mind?
Post

Famous quotes about learning “Learning is not attained by chance. ... “There is no end to education. ... “We now accept the fact that learning is a lifelong process of keeping abreast of change. ... “Live as if you were to die tomorrow. ... “One learns from books and example only that certain things can be done.

Viva Questions:

1. What is microblogging?
2. How would you design the database for this application?
3. How would you handle user relationships (following/followers)?
4. How would you efficiently retrieve posts for a user's feed?
5. What technologies would you use for the front-end and back end?

RESULT

A functional microblogging application was successfully developed. It enables users to create accounts, post short messages, follow or unfollow other users, and view a personalized feed of posts from followed users. The application effectively demonstrates core social media features with an intuitive user interface.

EX NO: 4 CREATE A FOOD DELIVERY WEBSITE WHERE USERS CAN ORDER FOOD FROM A PARTICULAR RESTAURANT LISTED IN THE WEBSITE.

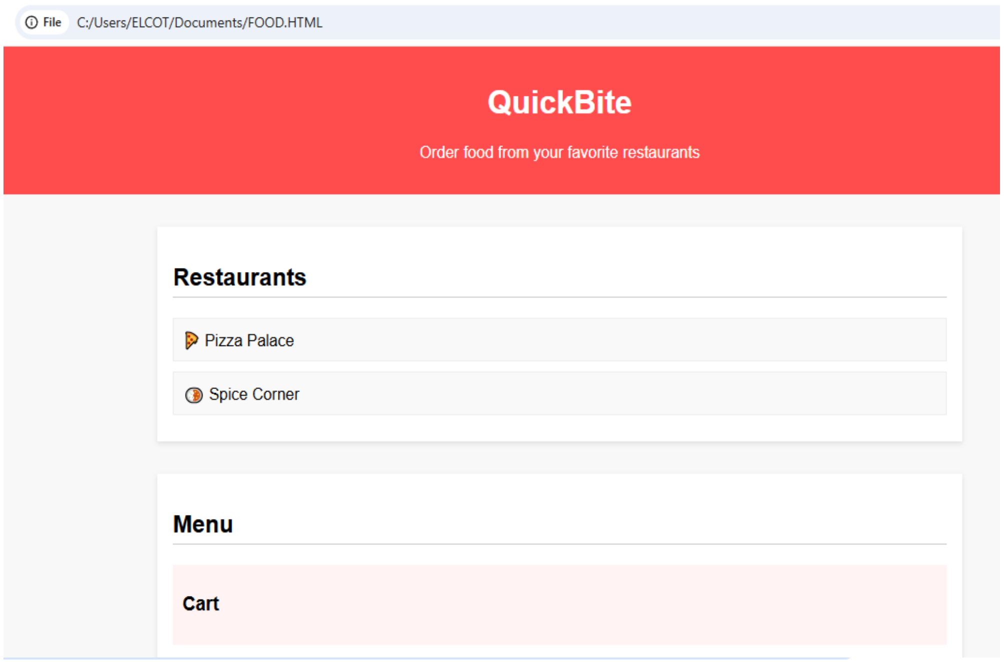
AIM

To develop a food delivery website that allows users to browse a list of available restaurants, view their menus, and place orders for food online, ensuring a smooth and user-friendly ordering experience.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>QuickBite - Food Delivery</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 0;
      background: #f8f8f8;
    }
    header {
      background: #ff4d4d;
      color: white;
      text-align: center;
      padding: 1rem;
    }
    .restaurant-list, .menu {
      max-width: 800px;
      margin: 2rem auto;
      padding: 1rem;
      background: white;
      box-shadow: 0 2px 5px rgba(0,0,0,0.1);
    }
    h2 {
      border-bottom: 1px solid #ccc;
      padding-bottom: 5px;
    }
    .restaurant, .item {
      margin: 10px 0;
      padding: 10px;
      border: 1px solid #eee;
      background: #f9f9f9;
    }
    button {
      background: #ff4d4d;
      color: white;
      border: none;
      padding: 5px 10px;
      cursor: pointer;
    }
    .cart {
      margin-top: 20px;
      background: #fff3f3;
      padding: 10px;
    }
  </style>
</head>
<body>
  <header>
    <h1>QuickBite</h1>
    <p>Order food from your favorite restaurants</p>
  </header>

  <div class="restaurant-list">
    <h2>Restaurants</h2>
    <div class="restaurant" onclick="showMenu('Pizza Palace')">🍕 Pizza Palace</div>
    <div class="restaurant" onclick="showMenu('Spice Corner')">🍛 Spice Corner</div>
  </div>

  <div class="menu" id="menu">
    <h2 id="restaurant-name">Menu</h2>
    <div id="menu-items"></div>
    <div class="cart">
      <h3>Cart</h3>
      <ul id="cart"></ul>
    </div>
  </div>

  <script>
    const menus = {
      "Pizza Palace": ["Margherita Pizza", "Pepperoni Pizza", "Veggie Delight"],
      "Spice Corner": ["Chicken Biryani", "Paneer Butter Masala", "Garlic Naan"]
    };

    const showMenu = (restaurant) => {
      document.getElementById('restaurant-name').textContent = restaurant;
      const menuItemsDiv = document.getElementById('menu-items');
      menuItemsDiv.innerHTML = '';
      menus[restaurant].forEach(item => {
        const div = document.createElement('div');
        div.className = 'item';
        div.innerHTML = `${item} <button onclick="addToCart('${item}')">Add to Cart</button>`;
        menuItemsDiv.appendChild(div);
      });
    };

    const cart = [];
    const addToCart = (item) => {
      cart.push(item);
      const cartList = document.getElementById('cart');
      const li = document.createElement('li');
      li.textContent = item;
      cartList.appendChild(li);
    };
  </script>
</body>
```

```
</html>
```

OUTPUT



File C:/Users/ELCOT/Documents/FOOD.HTML

QuickBite

Order food from your favorite restaurants

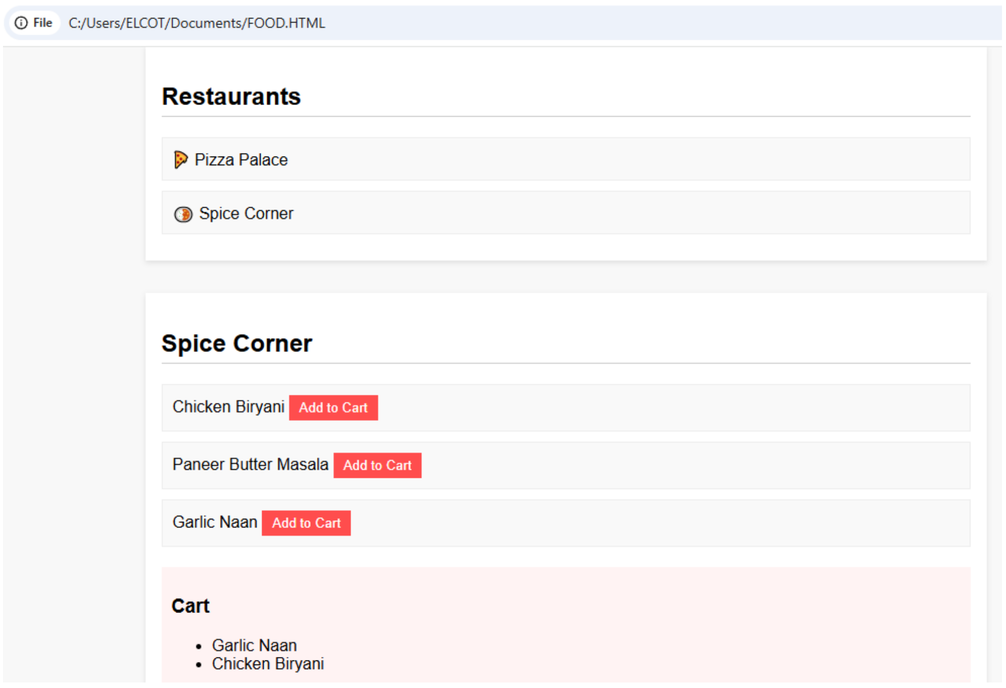
Restaurants

Pizza Palace

Spice Corner

Menu

Cart



File C:/Users/ELCOT/Documents/FOOD.HTML

Restaurants

Pizza Palace

Spice Corner

Spice Corner

Chicken Biryani Add to Cart

Paneer Butter Masala Add to Cart

Garlic Naan Add to Cart

Cart

- Garlic Naan
- Chicken Biryani

Viva Questions:

1. What navigation apps can you add for the delivery services?
2. How to fix timing issues in delivery applications?
3. How shall you display shopping cart feature on a website?
4. How shall you implement search bar for finding restaurants and dishes?
5. How shall you add buttons for adding items to the cart?

RESULT

The food delivery website was successfully developed with features including restaurant listings, dynamic menu display, cart functionality, and order placement. Users can seamlessly browse restaurants, select food items, and place orders, providing a convenient and efficient online food ordering experience.

EX NO: 5 DEVELOP A CLASSIFIEDS WEB APPLICATION TO BUY AND SELL USED PRODUCTS.

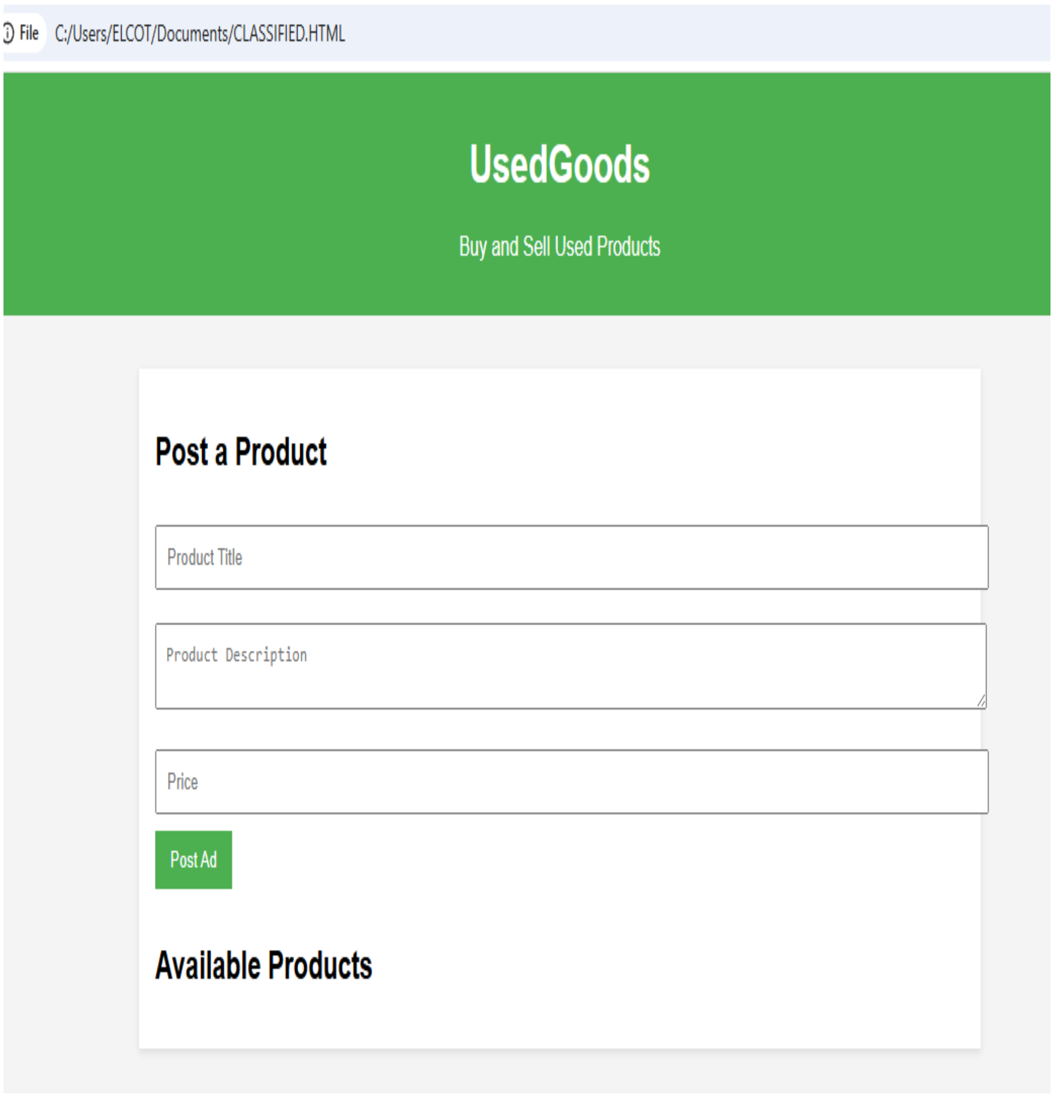
AIM

To design and develop a classifieds web application that enables users to post advertisements for used products they want to sell, and allows other users to browse, search, and contact sellers to buy listed items.

SOURCE CODE:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>UsedGoods - Buy & Sell</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 0;
      background: #f4f4f4;
    }
    header {
      background: #4CAF50;
      color: white;
      padding: 1rem;
      text-align: center;
    }
    .container {
      max-width: 800px;
      margin: 2rem auto;
      background: white;
      padding: 1rem;
      box-shadow: 0 2px 5px rgba(0,0,0,0.1);
    }
    form {
      margin-bottom: 2rem;
    }
    input, textarea {
      width: 100%;
      margin: 10px 0;
      padding: 10px;
    }
    button {
      background: #4CAF50;
      color: white;
      padding: 10px 15px;
      border: none;
      cursor: pointer;
    }
    .listing {
      border: 1px solid #ddd;
      margin-bottom: 1rem;
      padding: 1rem;
      background: #fafafa;
    }
  </style>
</head>
<body>
  <header>
    <h1>UsedGoods</h1>
    <p>Buy and Sell Used Products</p>
  </header>

  <div class="container">
    <h2>Post a Product</h2>
    <form id="postForm">
      <input type="text" id="title" placeholder="Product Title" required>
      <textarea id="description" placeholder="Product Description" required></textarea>
      <input type="text" id="price" placeholder="Price" required>
      <button type="submit">Post Ad</button>
    </form>

    <h2>Available Products</h2>
    <div id="listings"></div>
  </div>

  <script>
    const form = document.getElementById('postForm');
    const listings = document.getElementById('listings');

    form.addEventListener('submit', function(e) {
      e.preventDefault();

      const title = document.getElementById('title').value;
      const description = document.getElementById('description').value;
      const price = document.getElementById('price').value;

      const div = document.createElement('div');
      div.className = 'listing';
      div.innerHTML = `<h3>${title}</h3><p>${description}</p><strong>Price: ₹${price}</strong>`;

      listings.prepend(div);

      form.reset();
    });
  </script>
</body>
</html>
```

OUTPUT



File C:/Users/ELCOT/Documents/CLASSIFIED.HTML

UsedGoods

Buy and Sell Used Products

Post a Product

Product Title

Product Description

Price

Post Ad

Available Products

Viva Questions:

1. How can ad-posting be done in this website to make it better?
2. How can search functionality be implemented in this website to search for items as per customer wishlist?
3. How can payment integration be done in this website for marketing business products?
4. How can users be notified new messages and responses to ads?
5. How would you manage backend development of databases for storing specific features of recently introduced products?

RESULT

The classifieds web application was successfully developed with key features such as user registration/login, posting ads with product images and descriptions, search and filter options, and contact functionality. The application provides an effective platform for users to buy and sell used products online.

EX NO: 6 DEVELOP A LEAVE MANAGEMENT SYSTEM FOR AN ORGANIZATION WHERE USERS CAN APPLY DIFFERENT TYPES OF LEAVES SUCH AS CASUAL LEAVE AND MEDICAL LEAVE. THEY ALSO CAN VIEW THE AVAILABLE NUMBER OF DAYS.

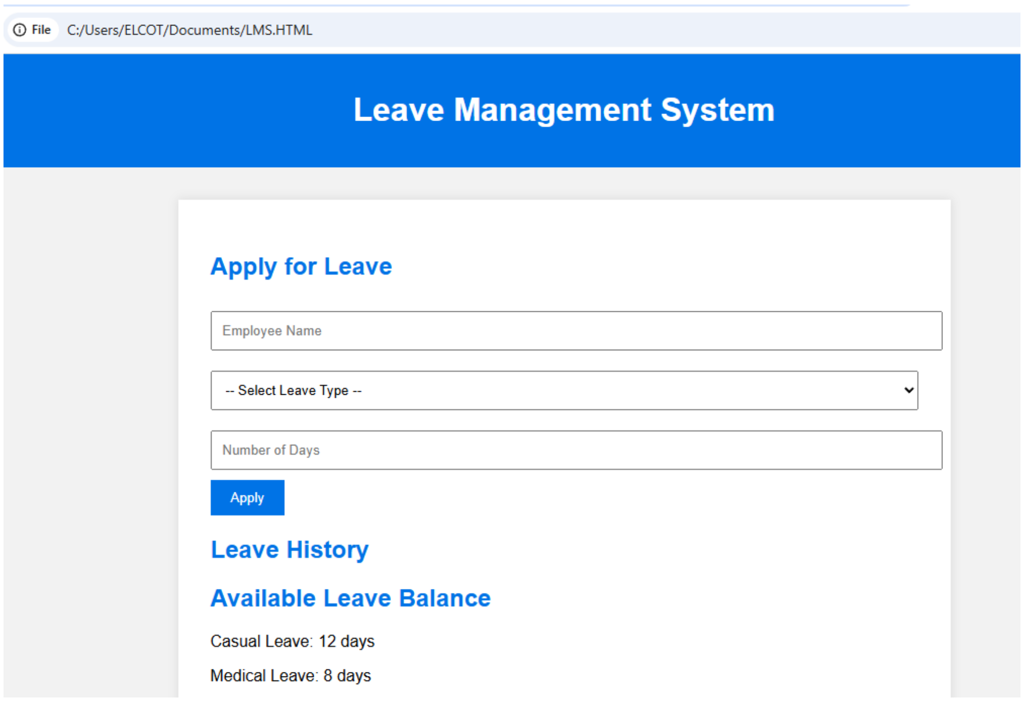
AIM

To develop a Leave Management System for an organization that enables employees to apply for different types of leaves (such as casual leave, medical leave, etc.), track their leave history, and view the number of available leave days, while also allowing administrators to manage and approve leave requests.

SOURCE CODE:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Leave Manager</title>
  <style>
    body {
      font-family: sans-serif;
      background-color: #f2f2f2;
      margin: 0;
      padding: 0;
    }
    header {
      background: #0073e6;
      color: white;
      text-align: center;
      padding: 1rem;
    }
    .container {
      max-width: 700px;
      margin: 2rem auto;
      background: white;
      padding: 2rem;
      box-shadow: 0 0 10px rgba(0,0,0,0.1);
    }
    h2 {
      color: #0073e6;
    }
    input, select {
      width: 100%;
      padding: 10px;
      margin: 10px 0;
    }
    button {
      background: #0073e6;
      color: white;
      padding: 10px 20px;
      border: none;
      cursor: pointer;
    }
    .leave-list {
      margin-top: 20px;
    }
    .leave-entry {
      background: #f9f9f9;
      padding: 10px;
      margin-bottom: 10px;
      border-left: 4px solid #0073e6;
    }
  </style>
</head>
<body>
  <header>
    <h1>Leave Management System</h1>
  </header>

  <div class="container">
    <h2>Apply for Leave</h2>
    <form id="leaveForm">
      <input type="text" id="employeeName" placeholder="Employee Name" required>
      <select id="leaveType" required>
        <option value="">-- Select Leave Type --</option>
        <option value="Casual Leave">Casual Leave</option>
        <option value="Medical Leave">Medical Leave</option>
      </select>
      <input type="number" id="leaveDays" placeholder="Number of Days" min="1" required>
      <button type="submit">Apply</button>
    </form>

    <div class="leave-list">
      <h2>Leave History</h2>
      <div id="history"></div>
    </div>

    <div class="leave-balance">
      <h2>Available Leave Balance</h2>
      <p>Casual Leave: <span id="casualBalance">12</span> days</p>
      <p>Medical Leave: <span id="medicalBalance">8</span> days</p>
    </div>
  </div>

  <script>
    let casualLeave = 12;
    let medicalLeave = 8;

    const form = document.getElementById('leaveForm');
    const history = document.getElementById('history');
    const casualBalance = document.getElementById('casualBalance');
    const medicalBalance = document.getElementById('medicalBalance');

    form.addEventListener('submit', function(e) {
      e.preventDefault();

      const name = document.getElementById('employeeName').value;
      const type = document.getElementById('leaveType').value;
      const days = parseInt(document.getElementById('leaveDays').value);

      if (type === 'Casual Leave' && days > casualLeave) {
        alert('Insufficient Casual Leave balance');
        return;
      }
      if (type === 'Medical Leave' && days > medicalLeave) {
        alert('Insufficient Medical Leave balance');
        return;
      }

      const entry = document.createElement('div');
      entry.className = 'leave-entry';
      entry.innerHTML = `<strong>${name}</strong> applied for <strong>${days} day(s)</strong> of
<em>${type}</em>`;
      history.prepend(entry);

      if (type === 'Casual Leave') {
        casualLeave -= days;
        casualBalance.textContent = casualLeave;
      } else if (type === 'Medical Leave') {
        medicalLeave -= days;
        medicalBalance.textContent = medicalLeave;
      }

      form.reset();
    });
  </script>
</body>
</html>
```

OUTPUT


File C:/Users/ELCOT/Documents/LMS.HTML

Leave Management System

Apply for Leave

Employee Name

-- Select Leave Type --

Number of Days

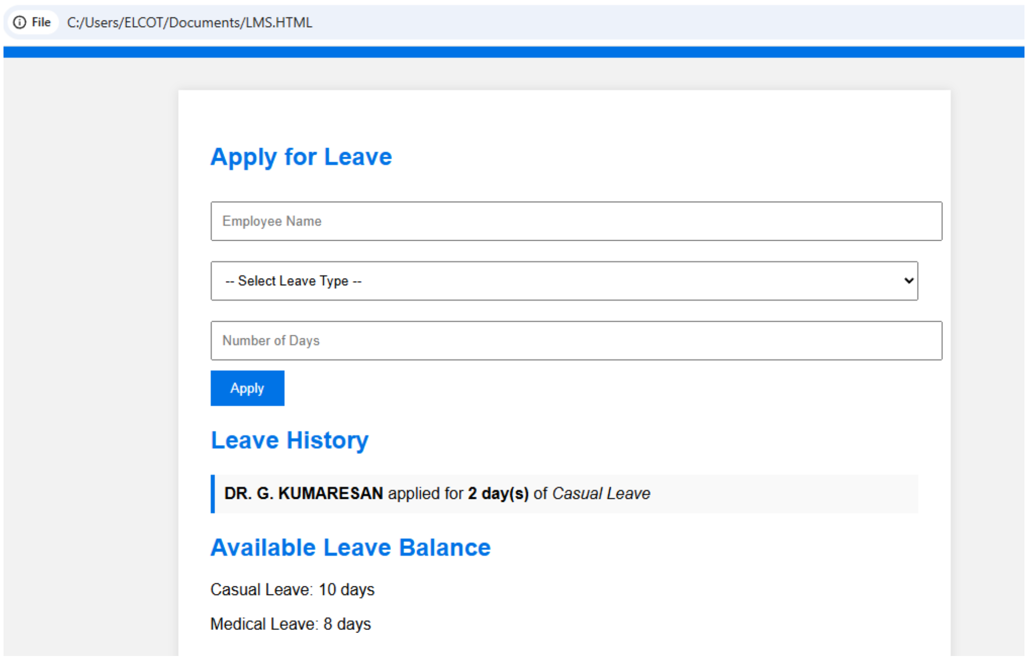
Apply

Leave History

Available Leave Balance

Casual Leave: 12 days

Medical Leave: 8 days


File C:/Users/ELCOT/Documents/LMS.HTML

Apply for Leave

Employee Name

-- Select Leave Type --

Number of Days

Apply

Leave History

DR. G. KUMARESAN applied for **2 day(s)** of *Casual Leave*

Available Leave Balance

Casual Leave: 10 days

Medical Leave: 8 days

Viva Questions:

1. How does the system handle situations where an employee is unable to submit a request themselves?
2. How does the system handle situations where an employee's leave request conflicts with business needs?
3. What are the consequences of unauthorized or excessive absence?
4. Describe a situation where you had to handle a complex leave request.
5. How is the system maintained and updated?

RESULT

The Leave Management System was successfully developed with functionalities for user login, leave application, leave type categorization, real-time leave balance display, and admin-level leave approval or rejection. The system enhances transparency and streamlines the leave management process within the organization.

EX NO: 7 DEVELOP A SIMPLE DASHBOARD FOR PROJECT MANAGEMENT WHERE THE STATUSES OF VARIOUS TASKS ARE AVAILABLE. NEW TASKS CAN BE ADDED AND THE STATUS OF EXISTING TASKS CAN BE CHANGED AMONG PENDING, IN PROGRESS OR COMPLETED.

AIM

To design and develop a simple project management dashboard that allows users to add new tasks, view the status of various existing tasks, and update their status among "Pending", "In Progress", or "Completed", providing an efficient way to manage and track project activities.

Simple web-based dashboard for project management using **HTML, CSS, and JavaScript**. It allows you to:

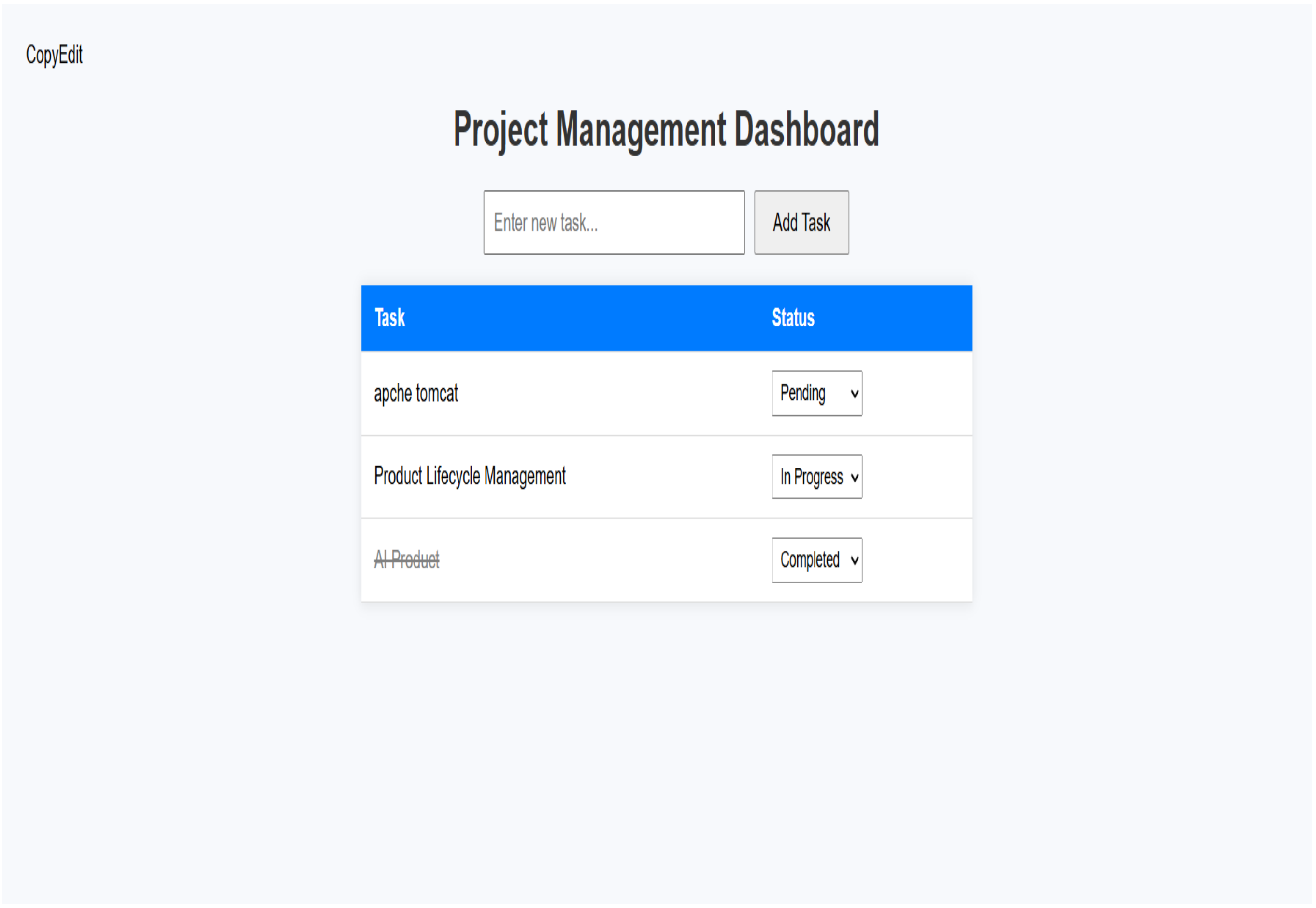
- View tasks with their statuses
- Add new tasks
- Change status of existing tasks between **Pending**, **In Progress**, and **Completed**

Code:

html
CopyEdit

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width, initial-scale=1" />
<title>Simple Project Management Dashboard</title>
<style>
  body {
    font-family: Arial, sans-serif;
    margin: 30px;
    background-color: #f7f9fc;
  }
  h1 {
    text-align: center;
    color: #333;
  }
  #task-form {
    display: flex;
    justify-content: center;
    margin-bottom: 20px;
  }
  #task-input {
    width: 300px;
    padding: 10px;
    font-size: 16px;
  }
  #add-task-btn {
    padding: 10px 20px;
    margin-left: 10px;
    font-size: 16px;
    cursor: pointer;
  }
  table {
    width: 100%;
    border-collapse: collapse;
    max-width: 700px;
    margin: auto;
    background: white;
    box-shadow: 0 0 10px rgba(0,0,0,0.1);
  }
  th, td {
    padding: 12px 15px;
    border-bottom: 1px solid #ddd;
    text-align: left;
  }
  th {
    background-color: #007bff;
    color: white;
  }
  select {
    padding: 5px;
    font-size: 14px;
  }
  tr.completed td {
    text-decoration: line-through;
    color: gray;
  }
</style>
</head>
<body>

<h1>Project Management Dashboard</h1>

<div id="task-form">
  <input type="text" id="task-input" placeholder="Enter new task..." />
  <button id="add-task-btn">Add Task</button>
</div>

<table>
  <thead>
    <tr>
      <th>Task</th>
      <th>Status</th>
    </tr>
  </thead>
  <tbody id="task-table-body">
    <!-- Tasks will appear here -->
  </tbody>
</table>

<script>
  const tasks = [];

  const taskInput = document.getElementById('task-input');
  const addTaskBtn = document.getElementById('add-task-btn');
  const taskTableBody = document.getElementById('task-table-body');

  function renderTasks() {
    taskTableBody.innerHTML = ''; // Clear table body

    tasks.forEach((task, index) => {
      const tr = document.createElement('tr');
      if (task.status === 'Completed') {
        tr.classList.add('completed');
      }

      const tdTask = document.createElement('td');
      tdTask.textContent = task.name;

      const tdStatus = document.createElement('td');
      const select = document.createElement('select');
      ['Pending', 'In Progress', 'Completed'].forEach(status => {
        const option = document.createElement('option');
        option.value = status;
        option.textContent = status;
        if (status === task.status) option.selected = true;
        select.appendChild(option);
      });

      select.addEventListener('change', (e) => {
        tasks[index].status = e.target.value;
        renderTasks();
      });

      tdStatus.appendChild(select);

      tr.appendChild(tdTask);
      tr.appendChild(tdStatus);

      taskTableBody.appendChild(tr);
    });
  }

  addTaskBtn.addEventListener('click', () => {
    const taskName = taskInput.value.trim();
    if (taskName) {
      tasks.push({ name: taskName, status: 'Pending' });
      taskInput.value = '';
      renderTasks();
    } else {
      alert('Please enter a task name.');
    }
  });

  // Optional: allow pressing Enter key to add task
  taskInput.addEventListener('keypress', (e) => {
    if (e.key === 'Enter') {
      addTaskBtn.click();
    }
  });

  // Initial render (empty)
  renderTasks();
</script>

</body>
</html>
```

Output:


CopyEdit

Project Management Dashboard

Task	Status
apche tomcat	Pending
Product Lifecycle Management	In Progress
~~AI Product~~	Completed

How to use it:

1. Save this code as dashboard.html.
2. Open it in any modern web browser.
3. Add new tasks using the input box and "Add Task" button.
4. Change the status from the dropdown menu for each task.
5. Completed tasks will show with strikethrough style.

Viva Questions:

1. How do you handle accuracy in survey in this system?
2. Can you explain the role of surveying software in your work?
3. How to stay updated with latest surveying technologies?
4. Can you describe a time when you solve complex surveying problem?
5. What are the common sources of errors in surveying that needs to be avoided?

RESULT

The project management dashboard was successfully developed with functionality to create tasks, view task lists, and update their status dynamically. The dashboard provides a clear visual representation of task progress, helping teams stay organized and manage projects effectively.

EX NO: 8 DEVELOP AN ONLINE SURVEY APPLICATION WHERE A COLLECTION OF QUESTIONS IS AVAILABLE AND USERS ARE ASKED TO ANSWER ANY RANDOM 5 QUESTIONS

AIM

To develop an online survey application that presents users with a collection of questions and randomly selects 5 questions for each user to answer, thereby collecting valuable user responses for analysis.

Here's a simple plan and example code to develop an **online survey application** where:

- There's a collection of questions.
- Each user gets asked **5 random questions** from that collection.
- User answers are collected and saved.

Basic Features:

1. **Question bank** — a list of questions stored in backend.
2. **Random selection** — serve 5 random questions per user session.
3. **Answer submission** — collect user answers.
4. (Optional) Store answers somewhere (database or in-memory for demo).

Tech stack:

- Frontend: React or plain HTML+JS
- Backend: Node.js + Express (or any backend)
- For simplicity, I'll give you a **Node.js + Express + EJS template** example here with in-memory storage.

Simple Example (Node.js + Express)

Step 1: Setup Express app

```
bash
CopyEdit
npm init -y
npm install express ejs body-parser
```

Step 2: Create app.js

```
js
CopyEdit
const express = require('express');
const bodyParser = require('body-parser');
const app = express();

app.set('view engine', 'ejs');
app.use(bodyParser.urlencoded({ extended: false }));

// Question bank
const questions = [
  "What is your favorite color?",
  "How often do you exercise?",
  "What is your dream job?",
  "Which country do you want to visit next?",
  "Do you prefer coffee or tea?",
  "What's your favorite movie genre?",
  "How many hours do you sleep?",
  "What's your hobby?",
  "Are you a morning person or night owl?",
  "Do you like working remotely?"
];

// Simple in-memory answers store (demo only)
const answers = [];

// Route to show random 5 questions
app.get('/', (req, res) => {
  // Shuffle and pick 5
  const shuffled = questions.sort(() => 0.5 - Math.random());
  const selected = shuffled.slice(0, 5);

  res.render('survey', { questions: selected });
});

// Handle form submission
app.post('/submit', (req, res) => {
  const userAnswers = req.body;  // answers keyed by question index or text
  answers.push(userAnswers);

  res.send('Thanks for completing the survey!');
});

// Start server
app.listen(3000, () => {
  console.log('Survey app listening on http://localhost:3000');
});
```

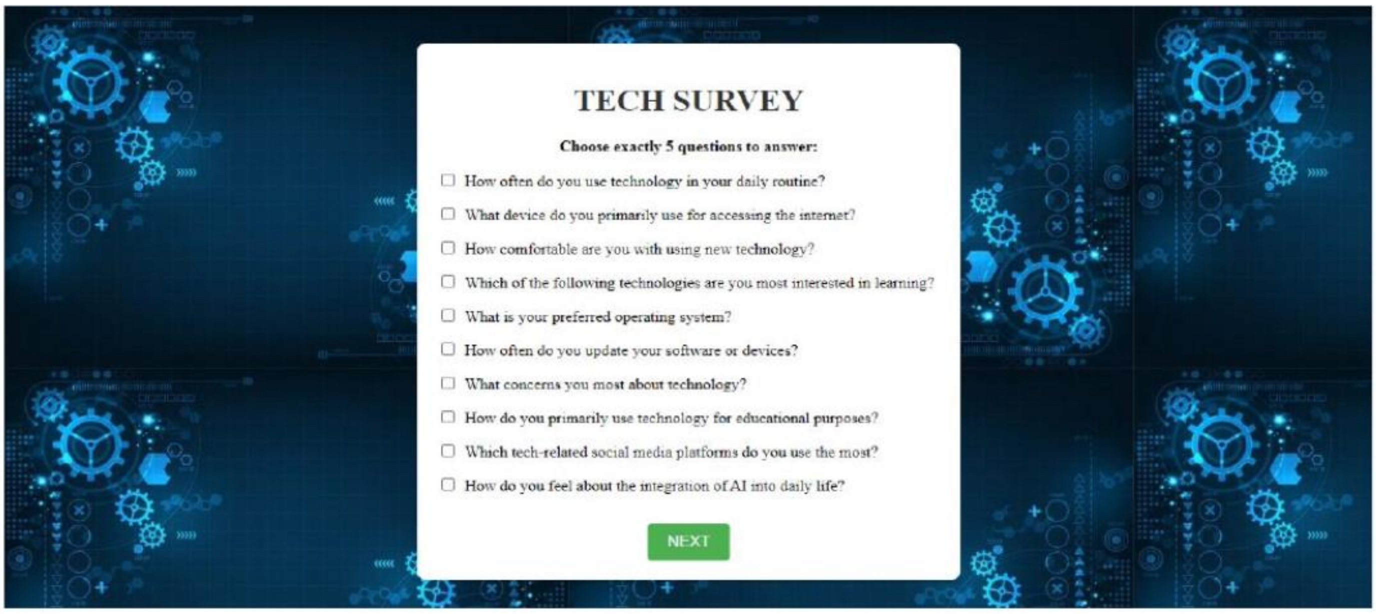
Step 3: Create view views/survey.ejs

html
CopyEdit
```
<!DOCTYPE html>
<html>
<head>
  <title>Online Survey</title>
</head>
<body>
  <h1>Please answer the following questions:</h1>
  <form action="/submit" method="POST">
    <% questions.forEach((q, i) => { %>
      <p>
        <label><%= q %></label><br/>
        <input type="text" name="q<%= i %>" required />
      </p>
    <% }) %>
    <button type="submit">Submit</button>
  </form>
</body>
</html>
```

How to run:

- Save the files.
- Run node app.js.
- Visit http://localhost:3000
- You will see 5 random questions to answer.
- Submit answers and see thank you message.

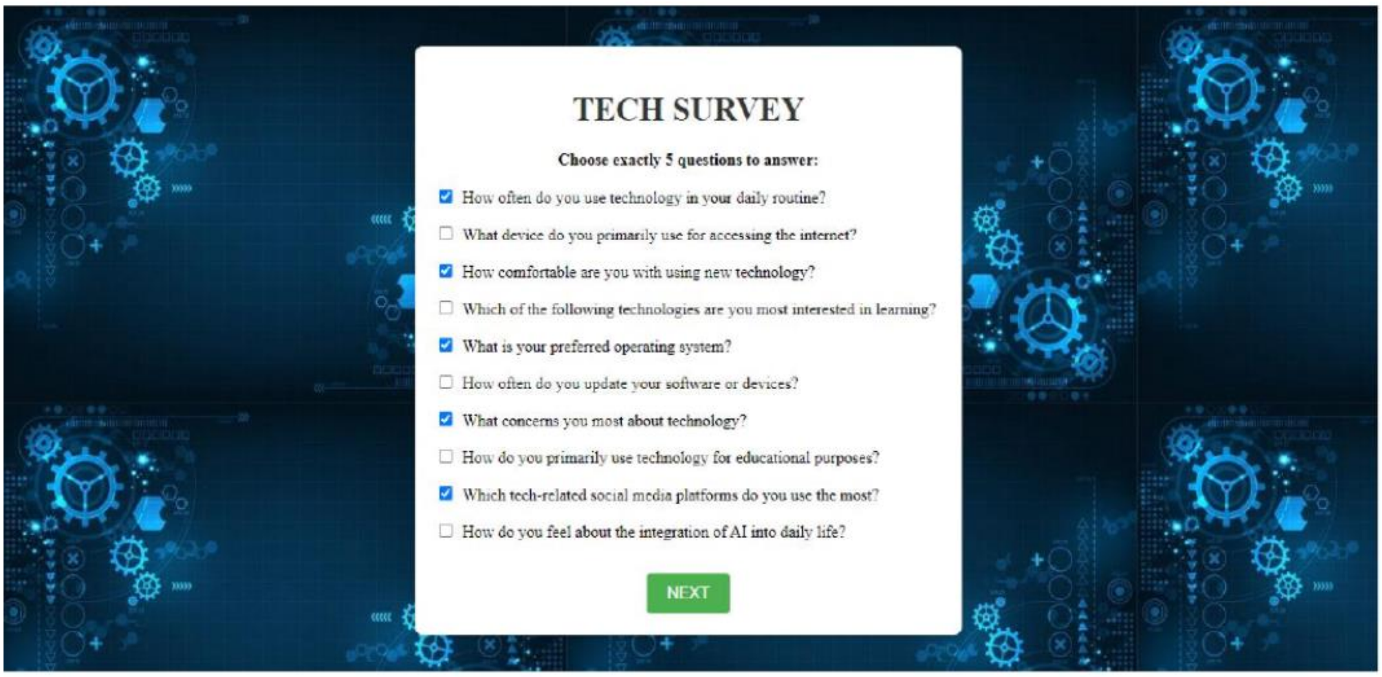
Output:



TECH SURVEY

Choose exactly 5 questions to answer:

- [ ] How often do you use technology in your daily routine?
- [ ] What device do you primarily use for accessing the internet?
- [ ] How comfortable are you with using new technology?
- [ ] Which of the following technologies are you most interested in learning?
- [ ] What is your preferred operating system?
- [ ] How often do you update your software or devices?
- [ ] What concerns you most about technology?
- [ ] How do you primarily use technology for educational purposes?
- [ ] Which tech-related social media platforms do you use the most?
- [ ] How do you feel about the integration of AI into daily life?



TECH SURVEY

Choose exactly 5 questions to answer:

- [x] How often do you use technology in your daily routine?
- [ ] What device do you primarily use for accessing the internet?
- [x] How comfortable are you with using new technology?
- [ ] Which of the following technologies are you most interested in learning?
- [x] What is your preferred operating system?
- [ ] How often do you update your software or devices?
- [x] What concerns you most about technology?
- [ ] How do you primarily use technology for educational purposes?
- [x] Which tech-related social media platforms do you use the most?
- [ ] How do you feel about the integration of AI into daily life?

Enhancements you can add:

- Store answers in a real database.
- Add user authentication.
- Add different types of questions (radio, checkbox).
- Show a summary of answers.
- Prevent repeated questions per user session.
- Add UI styling.

React frontend plus a backend (Node.js)

- Backend: Node.js + Express
  - Serve the question bank via API
  - Receive and save answers via API
- Frontend: React
  - Fetch 5 random questions from backend
  - Show questions with text inputs
  - Submit answers to backend

Step 1: Backend Setup (Node.js + Express)

1.1 Initialize backend

bash
CopyEdit

```
mkdir survey-backend
cd survey-backend
npm init -y
npm install express cors body-parser
```

1.2 Create index.js

js
CopyEdit

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

const app = express();
app.use(cors());
app.use(bodyParser.json());
```

```
const PORT = 4000;

// Question bank
const questions = [
  "What is your favorite color?",
  "How often do you exercise?",
  "What is your dream job?",
  "Which country do you want to visit next?",
  "Do you prefer coffee or tea?",
  "What's your favorite movie genre?",
  "How many hours do you sleep?",
  "What's your hobby?",
  "Are you a morning person or night owl?",
  "Do you like working remotely?"
];

// In-memory store for answers
const answers = [];

// API to get 5 random questions
app.get('/api/questions', (req, res) => {
  const shuffled = [...questions].sort(() => 0.5 - Math.random());
  const selected = shuffled.slice(0, 5);
  res.json(selected);
});

// API to receive answers
app.post('/api/submit', (req, res) => {
  const userAnswers = req.body;  // expect { question: answer, ... }
  answers.push(userAnswers);
  console.log('Received answers:', userAnswers);
  res.json({ message: 'Thanks for submitting the survey!' });
});

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

1.3 Run backend

bash
CopyEdit
```
node index.js
```

Step 2: Frontend Setup (React)

2.1 Create React app

bash

```
CopyEdit
npx create-react-app survey-frontend
cd survey-frontend
npm start
```

2.2 Replace content of src/App.js with:

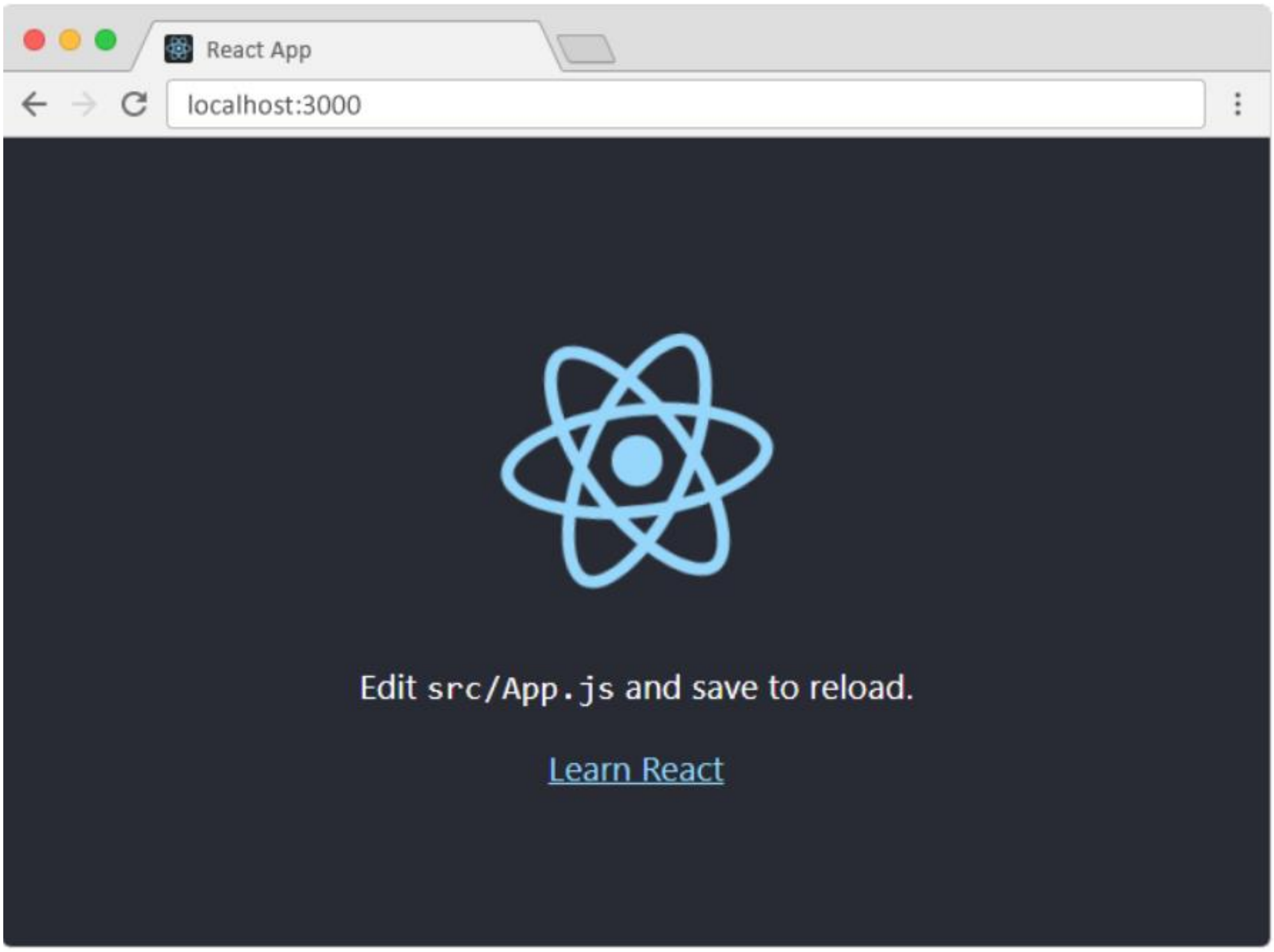
```
jsx
CopyEdit
import React, { useEffect, useState } from 'react';

function App() {
  const [questions, setQuestions] = useState([]);
  const [answers, setAnswers] = useState({});
  const [message, setMessage] = useState('');

  useEffect(() => {
    fetch('http://localhost:4000/api/questions')
      .then(res => res.json())
      .then(data => setQuestions(data))
      .catch(console.error);
  }, []);

  const handleChange = (index, value) => {
    setAnswers(prev => ({ ...prev, [questions[index]]: value }));
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const res = await fetch('http://localhost:4000/api/submit', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(answers)
      });
      const result = await res.json();
      setMessage(result.message);
      setAnswers({});
    } catch (err) {
      setMessage('Error submitting survey.');
    }
  };

  if (message) {
    return <div style={{ padding: 20 }}><h2>{message}</h2></div>;
  }

  return (
    <div style={{ padding: 20, maxWidth: 600, margin: 'auto' }}>
      <h1>Online Survey</h1>
      <form onSubmit={handleSubmit}>
        {questions.map((q, i) => (
          <div key={i} style={{ marginBottom: 15 }}>
            <label>{q}</label><br />
            <input
              type="text"
              required
              value={answers[q] || ''}
              onChange={e => handleChange(i, e.target.value)}
              style={{ width: '100%', padding: 8 }}
            />
          </div>
        ))}
        <button type="submit">Submit Answers</button>
      </form>
    </div>
  );
}

export default App;
```

2.3 Run React frontend

bash
CopyEdit

```
npm start
```

Go to http://localhost:3000 and you should see 5 random questions loaded from backend.



Edit `src/App.js` and save to reload.

Learn React

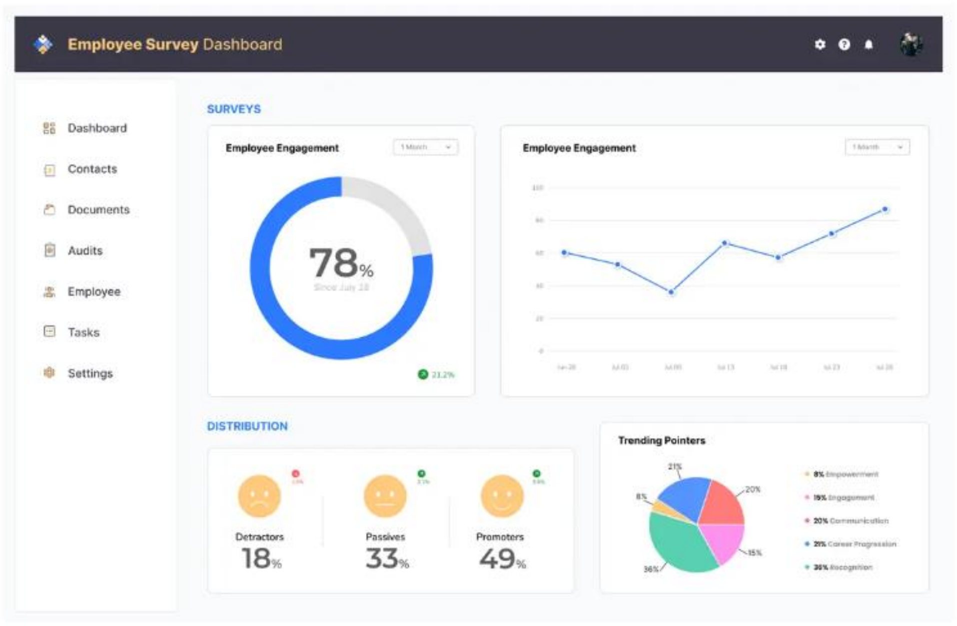
What you get

- Frontend fetches 5 random questions on load
- User answers and submits
- Backend receives and stores answers (in memory)
- Thank you message after submission

Next steps you can add

- Persist answers in a database like MongoDB or SQLite
- Add different question types (radio, checkbox)
- Add user authentication

Display survey results/admin dashboard



Viva Questions:

1. How does the user registration and authentication process work?
2. How are survey responses stored and managed?
3. How does the application handle large volumes of survey responses?
4. Explain the process of generating survey reports.
5. How do you handle different time zones for users and survey responses?

RESULT

The online survey application was successfully developed with a question bank feature, random question generation logic, and a response submission system. Users are presented with 5 random questions each time they access the survey, and their responses are stored for future evaluation and analysis.

ADDITIONAL EXPERIMENT

EXP NO 9 Develop full stack development application for airline reservation.

AIM:

Creating a full-stack airline reservation system involves developing both the **frontend**, **backend**, and **database**, with features such as:

- **User Registration & Login**
- **Flight Search**
- **Seat Selection**
- **Booking Management**
- **Admin Controls for Adding Flights**

Here's an outline and basic implementation stack for the application:

Tech Stack Suggestion

- **Frontend:** React.js (JavaScript)
- **Backend:** Node.js with Express
- **Database:** MongoDB or PostgreSQL
- **Authentication:** JWT (JSON Web Tokens)
- **Styling:** Tailwind CSS / Bootstrap
- **Hosting:** Vercel (frontend), Render or Railway (backend + database)

Project Structure

```
bash
CopyEdit
airline-reservation/
├── client/             # React frontend
├── server/             # Node.js backend
└── README.md
```

Backend (Node.js + Express + MongoDB)

`/server` structure

```
bash
CopyEdit
server/
├── controllers/
│   └── authController.js
│   └── flightController.js
├── models/
│   └── User.js
│   └── Flight.js
│   └── Booking.js
├── routes/
│   └── authRoutes.js
│   └── flightRoutes.js
│   └── bookingRoutes.js
├── server.js
└── .env
```

📦 Key Features

<u>`server.js` (entry point)</u>

```
js
CopyEdit
```

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

const authRoutes = require('./routes/authRoutes');
const flightRoutes = require('./routes/flightRoutes');
const bookingRoutes = require('./routes/bookingRoutes');

const app = express();
app.use(cors());
app.use(express.json());

mongoose.connect(process.env.MONGO_URI, { useNewUrlParser: true, useUnifiedTopology:
true })
    .then(() => console.log("MongoDB connected"))
    .catch(err => console.log(err));

app.use('/api/auth', authRoutes);
app.use('/api/flights', flightRoutes);
app.use('/api/bookings', bookingRoutes);

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

Frontend (React)

`/client` structure

```
bash
CopyEdit
client/
├── src/
│   ├── components/
│   ├── pages/
│   ├── services/
│   └── App.js
├── package.json
└── tailwind.config.js or index.css
```

□ Pages

- `HomePage` – Search flights
- `LoginPage`, `RegisterPage`
- `FlightDetailsPage` – Book ticket
- `BookingsPage` – View your bookings
- `AdminPage` – Add flights

Sample: Search Flights Page

```
jsx
CopyEdit
// SearchFlights.jsx
import React, { useState } from 'react';
import axios from 'axios';

function SearchFlights() {
    const [flights, setFlights] = useState([]);
    const [from, setFrom] = useState('');
    const [to, setTo] = useState('');

    const handleSearch = async () => {
        const res = await
axios.get(`http://localhost:5000/api/flights?from=${from}&to=${to}`);
        setFlights(res.data);
    };

    return (
        <div>
            <input placeholder="From" value={from} onChange={(e) =>
setFrom(e.target.value)} />
            <input placeholder="To" value={to} onChange={(e) => setTo(e.target.value)}
/>
            <button onClick={handleSearch}>Search</button>
            <ul>
                {flights.map(f => (
                    <li key={f._id}>{f.flightNumber} - {f.departureTime}</li>
                ))}
            </ul>
        </div>
    );
}
```

MongoDB Schema (Example)

User Schema

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

const userSchema = new mongoose.Schema({
    name: String,
    email: String,
    password: String,
    isAdmin: { type: Boolean, default: false }
});

module.exports = mongoose.model('User', userSchema);
```

Flight Schema

```
js
CopyEdit
const flightSchema = new mongoose.Schema({
    flightNumber: String,
    from: String,
    to: String,
    departureTime: Date,
    arrivalTime: Date,
    seatsAvailable: Number,
});

module.exports = mongoose.model('Flight', flightSchema);
```

Booking Schema

```
js
CopyEdit
const bookingSchema = new mongoose.Schema({
    userId: mongoose.Schema.Types.ObjectId,
    flightId: mongoose.Schema.Types.ObjectId,
    seatsBooked: Number,
    bookedAt: { type: Date, default: Date.now }
});

module.exports = mongoose.model('Booking', bookingSchema);
```

Authentication (JWT)

- `/api/auth/register`
- `/api/auth/login`
- Protected routes using JWT middleware:

```
js
CopyEdit
const jwt = require('jsonwebtoken');

const authenticate = (req, res, next) => {
    const token = req.header('Authorization')?.split(' ')[1];
    if (!token) return res.sendStatus(401);

    try {
        const decoded = jwt.verify(token, process.env.JWT_SECRET);
        req.user = decoded;
        next();
    } catch {
        res.sendStatus(403);
    }
};
```

Step 1: Backend Setup

Prerequisites

Make sure you have:

- Node.js and npm installed
- MongoDB instance (local or MongoDB Atlas)

Project Initialization

1. Create a folder:

```
bash
CopyEdit
mkdir airline-reservation
cd airline-reservation
mkdir server && cd server
npm init -y
```

2. Install dependencies:

```
bash
CopyEdit
npm install express mongoose dotenv cors jsonwebtoken bcryptjs
npm install --save-dev nodemon
```

3. Create file structure:

```
bash
CopyEdit
touch server.js .env
mkdir models controllers routes middleware
```

.env file

```
env
CopyEdit
PORT=5000
MONGO_URI=your_mongo_connection_string
JWT_SECRET=supersecretjwtkey
```

□ Step 2: User Authentication

User Model (`models/User.js`)

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

const userSchema = new mongoose.Schema({
    name: String,
    email: { type: String, unique: true },
    password: String,
    isAdmin: { type: Boolean, default: false }
});

module.exports = mongoose.model('User', userSchema);
```

Auth Controller (`controllers/authController.js`)

js
CopyEdit
```
const User = require('../models/User');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');

exports.register = async (req, res) => {
    try {
        const { name, email, password } = req.body;
        const hashedPassword = await bcrypt.hash(password, 10);
        const user = new User({ name, email, password: hashedPassword });
        await user.save();
        res.status(201).json({ message: 'User registered' });
    } catch (err) {
        res.status(400).json({ error: 'Email already exists' });
    }
};

exports.login = async (req, res) => {
    try {
        const { email, password } = req.body;
        const user = await User.findOne({ email });
        if (!user) return res.status(404).json({ error: 'User not found' });

        const match = await bcrypt.compare(password, user.password);
        if (!match) return res.status(401).json({ error: 'Invalid password' });

        const token = jwt.sign({ id: user._id, isAdmin: user.isAdmin },
process.env.JWT_SECRET);
        res.json({ token });
    } catch (err) {
        res.status(500).json({ error: 'Login error' });
    }
};
```

Auth Routes (`routes/authRoutes.js`)

js
CopyEdit
```
const express = require('express');
const router = express.Router();
const { register, login } = require('../controllers/authController');

router.post('/register', register);
router.post('/login', login);

module.exports = router;
```

Step 3: Server Setup (`server.js`)

js
CopyEdit

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

const authRoutes = require('./routes/authRoutes');

const app = express();

app.use(cors());
app.use(express.json());

mongoose.connect(process.env.MONGO_URI)
    .then(() => console.log('MongoDB Connected'))
    .catch(err => console.error(err));

app.use('/api/auth', authRoutes);

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

Test User Auth API

Using Postman or any REST client:

- **Register**
  - `POST http://localhost:5000/api/auth/register`
  - JSON Body: `{ "name": "John", "email": "john@example.com", "password": "123456" }`
- **Login**
  - `POST http://localhost:5000/api/auth/login`
  - Response: `{ token: "JWT_TOKEN" }`

Viva Questions:

1. How do you ensure accuracy and attention to detail when managing reservations?

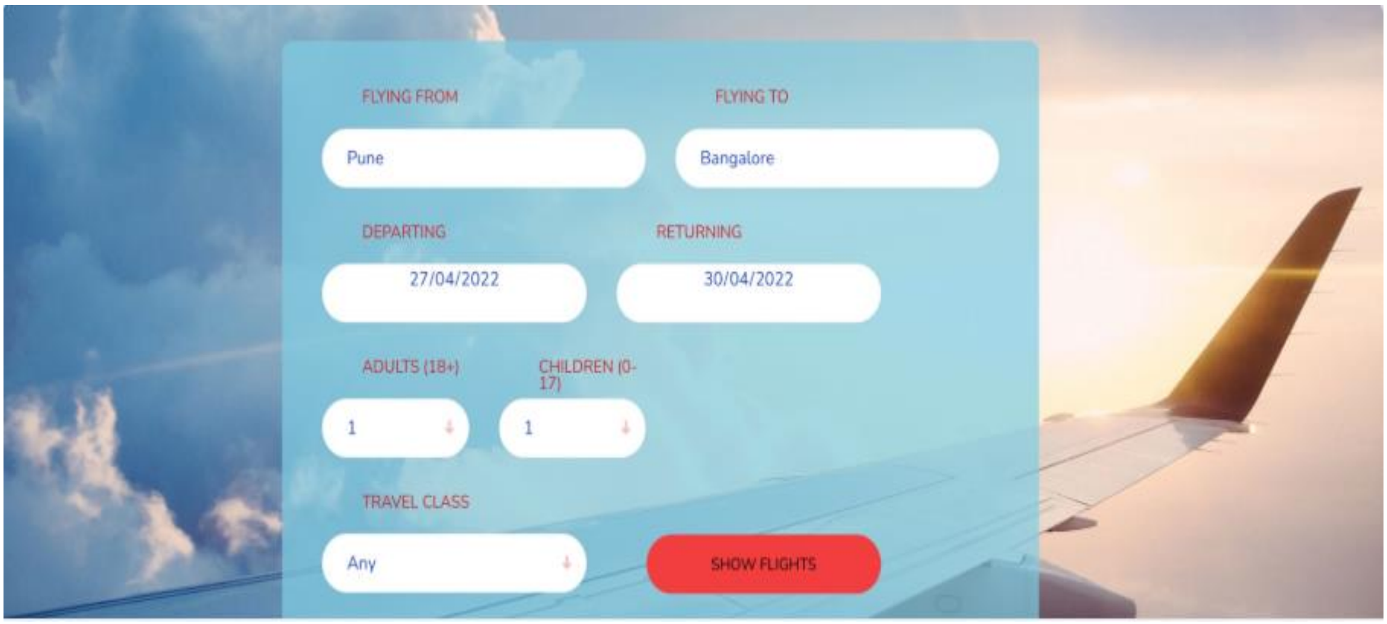
2. What are the steps involved in handling a flight change or cancellation in your application?

3. How do you handle situations in your application when a flight is overbooked?

4. How shall you manage customer flying experience in your system?

5. How will you handle the situation if multiple customers book the tickets in your system at the same time?

Output:



FLYING FROM

Pune

FLYING TO

Bangalore

DEPARTING

27/04/2022

RETURNING

30/04/2022

ADULTS (18+)

1

CHILDREN (0-17)

1

TRAVEL CLASS

Any

RESULT: The airline reservation application using full stack development was successfully developed and provide robust foundation for managing users , bookings and generating reports in fight booking system.