GLOBAL

INSTITUTE OF ENGINEERING AND TECHNOLOGY

(Approved by AICTE & Affiliated to Anna University)

257/1, Bangalore – Chennai High Way, Melvisharam, Ranipet – 632 509

DEPARTMENT OF INFORMATION AND TECHNOLOGY

IT3511-FULL STACK WEB DEVELOPMENT LABORATORY

ACADEMIC YEAR (2025-2026)

NAME : _____

REG NO. : _____

YEAR/SEM : III/V

GLOBAL
INSTITUTE OF ENGINEERING AND TECHNOLOGY
(Approved by AICTE & Affiliated to Anna University)
257/1, Bangalore – Chennai High Way, Melvisharam, Ranipet – 632 509

BONAFIDE CERTIFICATE

Name : .....

Course : B.Tech-IT

Year / Semester: III /V

No :

Certified that this is a bonafide record of work done by the above student in the **IT3511-FULL STACK WEB DEVELOPMENT LABORATORY** during the period 2025 to 2026.

Staff-in-Charge **Head of the Department**

Submitted for the Practical examination held on_____at Global Institute Of Engineering and Technology, Melvisharam, Ranipet-632509.

Internal Examiner **External Examiner**

INDEX

Exp. No	Date	Name of the Experiment	Page No	Marks	Sign

Exp. No	Date	Name of the Experiment	Page No	Marks	Sign

1. Develop a portfolio website for yourself which gives details about yourself for a potential recruiter.

Aim

To create a comprehensive and visually appealing online portfolio that provides potential recruiters with a detailed overview of my skills, experiences, projects, and achievements, effectively show casing my suitability for their organization.

Steps to Implement:

1. Create a new HTML file and paste the provided HTML code into it.
2. Create a "css" folder and place the "styles.css" file in it.
3. Create an "images" folder and place the image files mentioned in the HTML code.
4. Include the links to external libraries (Animate.css and Ionicons) in the <head> section.
5. Customize the content, images, and links according to your preferences.
6. Open the HTML file in a web browser to see the webpage in action.

Components Description:

HTML Structure:

- The HTML document starts with a <!DOCTYPE html> declaration, specifying the document type.
- The <html> element encloses the entire HTML content of the page.
- The <head> section contains metadata about the page, such as character encoding, viewport settings, and links to external resources like stylesheets and icons.
- The <body> section contains the main content of the webpage.
- Navigation (nav):
    - The navigation bar (<nav>) contains the site's logo (heading) and a list of navigation links (<ul>) using an unordered list.
    - Each navigation link is represented by a list item (<li>) containing an anchor (<a>) element.
    - The burger icon button (<button>) is used to toggle the navigation menu on smaller screens.
    - The JavaScript code handles the toggle functionality of the navigation menu when the burger icon is clicked.
- Hero Section:
    - The "hero" section (<section>) contains an image of a person and a "bio" ( ) with a brief description.
    - The bio includes a title (<h2>) and a paragraph (<p>) with text describing the person's role and background.
- More About Section:
    - The "more-about" section provides additional details about the person's involvement in organizations.
    - It includes a title (<h2>) and several paragraphs (<p>) with relevant information.
- Skills Section:
    - The "skills" section displays the person's top skills using icons (<img> elements).
    - Skills are divided into two groups using separate containers with the class first-set and second-set.
    - The icons are animated using the animate.css library with the class animate__pulse.
- Projects Section:
    - The "projects" section showcases recent projects.
    - Each project is contained within a with the class project-container project-card.
    - Each project includes an image (<img>), a title (<h3>), and a description (<p>).
- Contact Section:
    - The "contact" section provides a form for users to get in touch.

- The form includes fields for name, email, and a message (<input> and <textarea> elements).
- The form uses the "formspree.io" service to handle form submissions.

- Social Buttons and Scroll to Top:
  - Social media icons are displayed in a fixed column on the right side of the page (with class socials).
  - A scroll-to-top button is positioned at the bottom right corner of the page.

CSS Styling:

- The provided CSS code defines the styling for various elements on the page.
- CSS variables are used to store reusable values such as colors, fonts, and spacing.
- Media queries are used to make the page responsive for different screen sizes.

JavaScript Interactivity:

- The JavaScript code handles the toggling of the navigation menu on smaller screens when the burger icon is clicked.
- When a navigation link is clicked, the navigation menu closes automatically.
- The scroll-to-top button is implemented using JavaScript to smoothly scroll to the top of the page when clicked.

HTML Code:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <!--CSS Styles -->
    <link rel="stylesheet" href="E:\Nelson\FSWD\Exp_1\css\styles.css" />
    <!-- Favicons -->
    <link
      rel="apple-touch-icon"
      sizes="180x180"
      href="assets/icons/apple-touch-icon.png"
    />
    <link
      rel="icon"
      type="image/png"
      sizes="32x32"
      href="assets/icons/favicon-32x32.png"
    />
    <!-- Animate CSS CDN -->
    <link
      rel="stylesheet"
      href="https://cdnjs.cloudflare.com/ajax/libs/animate.css/4.1.1/animate.min.css"
    />
    <title>Jane Doe | Web Developer</title>
  </head>
  <body>
    <nav>
      <h1>NELSON SELVIN</h1>
      <ul class="navigation">
        <li><a href="#about" class="nav-link">About</a></li>
        <li><a href="#skills" class="nav-link">Skills</a></li>
```

```
        <li><a href="#projects" class="nav-link">Projects</a></li>
        <li><a href="#contact" class="nav-link">Contact</a></li>
      </ul>
      <button class="burger-menu" id="burger-menu">
        <ion-icon class="bars" name="menu-outline"></ion-icon>
      </button>
</nav>
<section class="hero" id="about">
      <img
        src="E:\Nelson\FSWD\Exp_1\images\Teacher.jpg"
        alt="jane-doe"
        loading="lazy"
        class="hero-img"
      />
      <div class="bio animate__animated animate__shakeX">
        <h2 class="bio-title">About Me</h2>
        <p class="bio-text">
            Nelson S Working as an Assistant Professor in IT Department, V.S.B Engineering College, Karur. I am working here from 2018 onwards.
        </p>
      </div>
</section>
    <section class="more-about">
      <h2>More About Me</h2>
      <p>
I am working here from 2018 onwards and I am the department coordinator for the accreditation related work.
      </p>
    </section>
    <section class="skills" id="skills">
      <h2 class="skill-header">My Top Skills</h2>
      <div class="skills-wrapper">
        <div class="first-set animate__animated animate__pulse">
          <img
            src="E:\Nelson\FSWD\Exp_1\images\c.jpg"
            alt=""
            loading="lazy"
            class="icon icon-card"
          />
          <img
            src="E:\Nelson\FSWD\Exp_1\images\java.jpg"
            alt=""
            loading="lazy"
            class="icon icon-card"
          />
          <img
            src="E:\Nelson\FSWD\Exp_1\images\python.jpg"
            alt=""
            loading="lazy"
            class="icon icon-card"
          />
        </div>
        <div class="second-set animate__animated animate__pulse">
          <img src="E:\Nelson\FSWD\Exp_1\images\php.jpg"
```

```
          alt=""
          loading="lazy"
          class="icon icon-card"
         />
         <img src="E:\Nelson\FSWD\Exp_1\images\html.jpg"
          alt=""
          loading="lazy"
          class="icon icon-card"
         />
         <img src="E:\Nelson\FSWD\Exp_1\images\sql.jpg"
          alt=""
          loading="lazy"
          class="icon icon-card"
         />
        </div>
      </div>
    </section>
<section class="projects" id="projects">
     <h2 class="projects-title">Some of my Recent Projects</h2>
     <div class="projects-container">
      <div class="project-container project-card">
       <img src="E:\Nelson\FSWD\Exp_1\images\paper1.jpg"
        alt="expense-tracker"
        loading="lazy"
        class="project-pic"
       />
       <h3 class="project-title">K-MEANS Clustering</h3>
       <p class="project-details">
          Implementing the K-Means algorithm for safeguarding sensitive attributes offers robust data protection through clustering. By grouping similar data points, K- Means minimizes the risk of exposing sensitive information, enhancing privacy in various applications!
       </p>
      </div>
      <div class="project-container project-card">
       <img
        src="E:\Nelson\FSWD\Exp_1\images\paper2.jpg"
        alt="netflic-clone"
        loading="lazy"
        class="project-pic"
       />
       <h3 class="project-title">Face Recognition using Laplacian Faces</h3>
       <p class="project-details">
          Leveraging Laplacian Faces for face recognition results in an advanced technique that extracts distinctive facial features.
       </p>
       </div>
    </div>
   </section>
<section class="contact" id="contact">
     <h2>Get In Touch With Me</h2>
     <div class="contact-form-container">
      <div class="contact-form">
       <form action="https://formspree.io/f/xyylngw" method="POST">
        <div class="form-control">
```

```
            <label for="name">Name</label>
            <input
              type="text"
              id="name"
              name="sender-name"
              placeholder="Enter Your Name"
              class="input-field"
              required
            />
          </div>
          <div class="form-control">
            <label for="email">Email</label>
            <input
              type="email"
              id="email"
              name="sender-email"
              placeholder="Enter Your Email"
              class="input-field"
              required
            />
          </div>
          <div class="form-control">
            <label for="message">Message</label>
            <textarea
              id="message"
              cols="60"
              rows="10"
              placeholder="Enter Your Message"
              name="message"
              class="input-field"
              required
            >
</textarea>
          </div>
          <input
            type="submit"
            value="Submit"
            id="submit-btn"
            class="submit-btn"
          />
        </form>
      </div>
    </div>
  </section>
  <!-- Website scripts -->
  <script src="assets/js/app.js"></script>
  <!-- Ion icons scripts -->
  <script
    type="module"
    src="https://unpkg.com/ionicons@5.5.2/dist/ionicons/ionicons.esm.js"
  ></script>
  <script
    nomodule
    src="https://unpkg.com/ionicons@5.5.2/dist/ionicons/ionicons.js"
```

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

CSS Code:

```
@import
url("https://fonts.googleapis.com/css2?
family=Roboto:ital,wght@0,400;0,900;
1,700&display=swap");
/* Variables */
:root {
 --font-family: "Roboto", sans-serf;
 --normal-font: 400;
 --bold-font: 700;
 --bolder-font: 900;
 --bg-color: #fcfcfc;
 --primary-color: #4756df;
 --secondary-color: #ff7235;
 --primary-shadow: #8b8eaf;
 --secondary-shadow: #a17a69;
 --bottom-margin: 0.5rem;
 --bottom-margin-2: 1rem;
 --line-height: 1.7rem;
 --transition: 0.3s;
}
/* Variables end */
html {
 scroll-behavior: smooth;
}
/* CSS Resets */
* {
 margin: 0;
 padding: 0;
 box-sizing: border-box;
 /* border: 2px solid red; */
}
ul {
 list-style-type: none;
}
a {
 text-decoration: none;
 color: var(--primary-color);
}
a:hover {
 color: var(--secondary-color);
}
body {
 font-family: var(--font-family);
}
section {
 /* max-width: 1100px;
 margin: auto; */
}
/* CSS Resets end */
/* Navbar */
nav {
 position: sticky;
 top: 0;
 left: 0;
 z-index: 1;
 display: flex;
 align-items: center;
 justify-content: space-between;
 padding: 1.5rem 3.5rem;
 background-color: var(--bg-color);
  box-shadow: 0 3px 5px rgba(0, 0, 0,
0.1);
}
nav h1 {
 color: var(--primary-color);
}
nav a {
 color: var(--primary-color);
 transition: var(--transition);
}
nav a:hover {
 color: var(--secondary-color);
     border-bottom: 2px solid var(--
secondary-color);
}
nav ul {
 display: flex;
 gap: 1.9rem;
}
nav ul li {
 font-weight: var(--bold-font);
}
.burger-menu {
 color: var(--primary-color);
 font-size: 2rem;
 border: 0;
 background-color: transparent;
 cursor: pointer;
 display: none;
}
/* Navbar ends */
/* Hero section */
.hero {
 display: flex;
 align-items: center;
 justify-content: center;
```

```
  gap: 2.5rem;
  max-width: 68.75rem;
  margin: auto;
 }
 .hero img {
  height: 37.5rem;
  width: 37.5rem;
 }
 .bio {
  width: 25rem;
  padding: 0.625rem;
  border-radius: 6px;
   box-shadow: 0px 2px 15px 2px var(--
primary-shadow);
 }
 .bio h1 {
  margin-bottom:  var(--bottom-margin);
 }
 .bio p {
  line-height: var(--line-height);
  padding: 0.3rem 0;
 }
 /* Hero section ends */
 /* More about */
 .more-about {
  background-color: var(--bg-color);
  padding: 1rem 6rem;
 }
 .more-about h2 {
  margin-bottom: var(--bottom-margin);
  text-align: center;
 }
 .more-about p {
  line-height: var(--line-height);
  padding: 0.4rem;
 }
 /* More about ends */
 /* Skills section */
 .skills {
  max-width: 68.75rem;
  margin: auto;
  text-align: center;
  margin-top: 2.5rem;
 }
 .skill-header {
  margin-bottom: 1rem;
 }
 .skills-wrapper img {
  padding: 1.25rem;
 }
 .icon {
  width: 11.875rem;
  height: 11.25rem;
 }
 /* Skills section ends */
 /* Projects section */
 .projects {
  background-color: var(--bg-color);
  padding: 32px 0;
  margin-top: 2rem;
 }
 .project-pic {
  width: 65%;
  height: 60%;
 }
 .projects-container {
  display: flex;
  align-items: center;
  justify-content: center;
 }
 .projects-title {
  text-align: center;
  margin-bottom: 1rem;
 }
 .project-container  {
  text-align:  center;
  width: 21.875rem;
  padding: 1rem;
 }
 .project-container p {
  padding: 0.4rem;
 }
 .project-title {
  margin-bottom:  var(--bottom-margin);
 }
 .project-details  {
  margin-bottom:  var(--bottom-margin);
 }
 /* Projects section ends */
 /* Contacts section */
 .contact {
  margin-top: 2rem;
 }
 .contact h2 {
  text-align: center;
    margin-bottom: var(--bottom-margin-
2);
 }
 .contact-form-container {
  max-width: 40.75rem;
  margin: 0 auto;
  padding: 0.938rem;
  border-radius: 5px;
      box-shadow:  0  3px  10px  var(--
secondary-shadow);
 }
 .contact-form-container label {
  line-height: 2.7em;
```

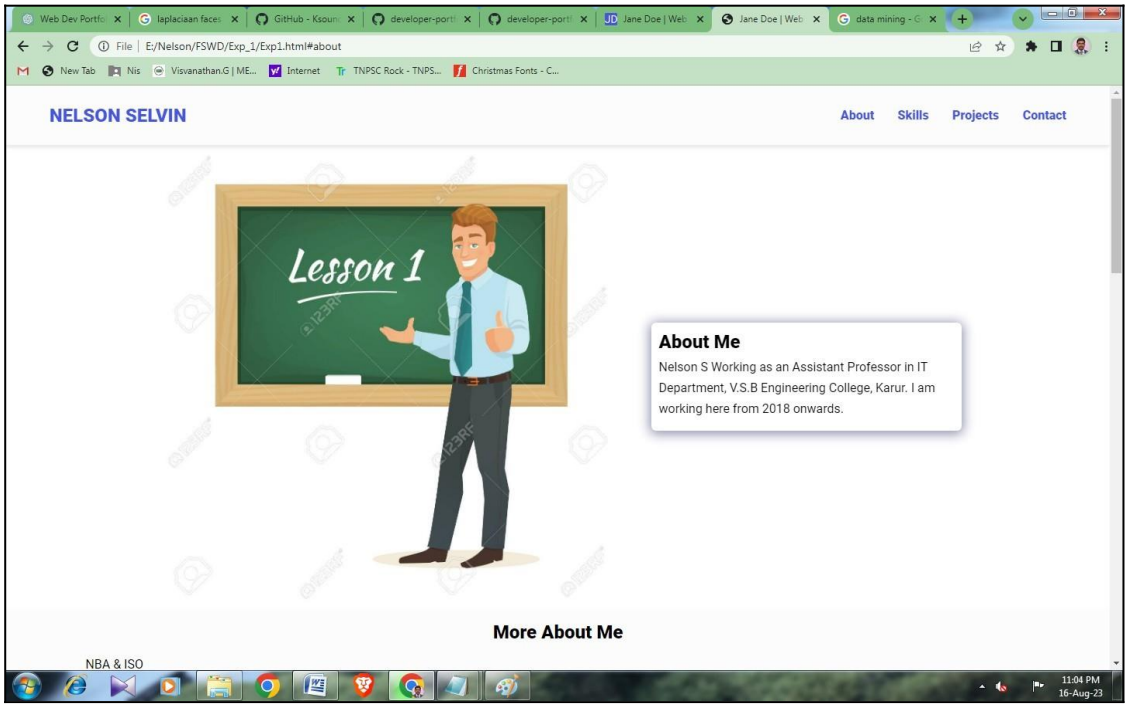
```
  font-weight: var(--bold-font);
  color: var(--primary-color);
}
.contact-form-container textarea {
  min-height: 6.25rem;
  font-size: 14px;
}
.contact-form-container .input-field {
  width: 100%;
  padding-top: 10px;
  padding-bottom: 10px;
  border-radius: 5px;
  border: none;
     border: 2px outset var(--primary-
color);
  font-size: 0.875rem;
  outline: none;
}
.input-field::placeholder {
  padding: 0.5rem;
  color: var(--primary-color);
}
.submit-btn {
  width: 100%;
  padding: 10px;
  margin: 10px 0;
  background-color: #fff;
  border: 2px solid var(--primary-color);
  border-radius: 5px;
  font-size: 1rem;
  font-weight: var(--bold-font);
  transition: var(--transition);
}
.submit-btn:hover {
       background-color:    var(--primary-
color);
  border: 2px solid var(--primary-color);
  cursor: pointer;
}
/* Contacts section ends */
/* Social buttons */
.socials {
  display: flex;
  flex-direction: column;
  position: fixed;
  right: 1%;
  bottom: 50%;
}
.socicon {
  width: 2rem;
  height: 2rem;
}
/* Social button ends */
/* Scroll to top button */
.scroll-up {
  position: fixed;
  right: 0.5%;
  bottom: 3%;
  cursor: pointer;
}
.up-arrow {
  width: 3rem;
  height: 3rem;
}
/* Scroll to top button ends */
/* Footer section */
footer {
  background-color: var(--bg-color);
  padding: 1.25rem;
  text-align: center;
  margin: 2rem 0 0;
}
/* Footer section ends */
/* General (utilities) */
.icon-card {
  background-color: #fff;
  border-radius: 11px;
     box-shadow: 0 3px 10px var(--
secondary-shadow);
  padding: 20px;
  margin: 10px;
}
.project-card {
  background-color: #fff;
  border-radius: 11px;
     box-shadow: 0 3px 10px var(--
primary-shadow);
  padding: 20px;
  margin: 10px;
}
/* Media queries for responsiveness */
@media screen and (max-width: 720px)
{
  nav {
    padding: 1.5rem 1rem;
  }
  nav ul {
    position: fixed;
    background-color: var(--bg-color);
    flex-direction: column;
    top: 86px;
    left: 10%;
    width: 80%;
    text-align: center;
    transform: translateX(120%);
    transition: transform 0.5s ease-in;
  }
  nav ul li {
```

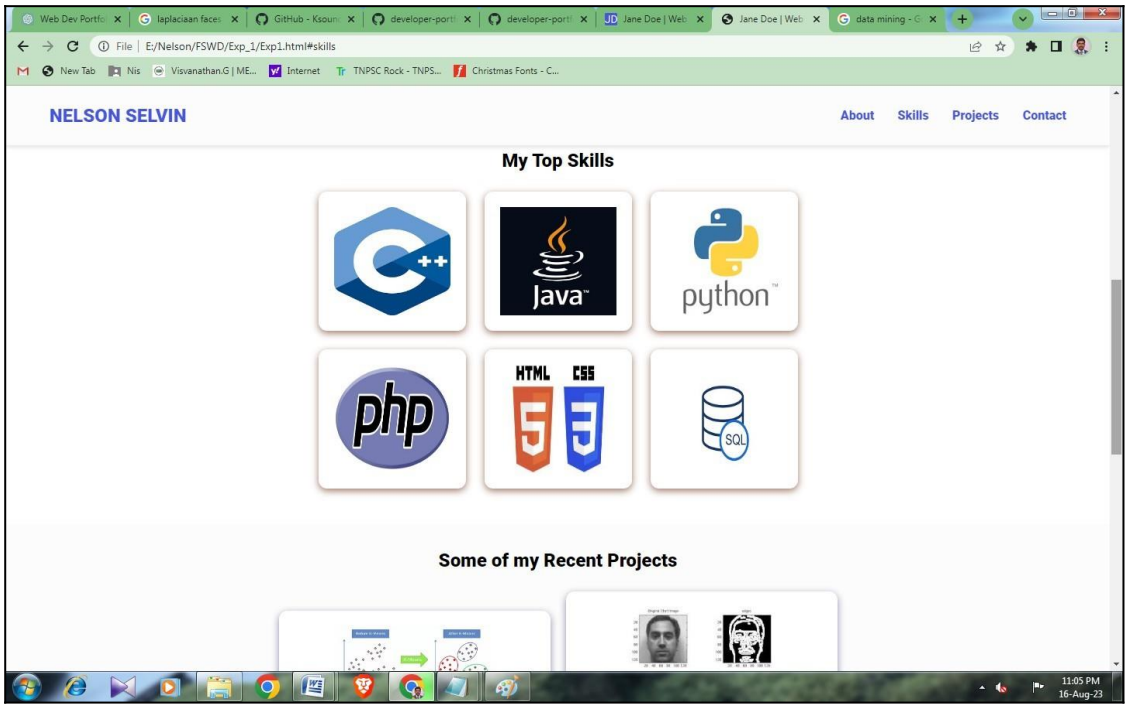
```
  margin: 8px;
 }
 .burger-menu {
  display: block;
 }
 nav ul.show {
  transform: translateX(0);
 }
 .hero {
  margin-top: -4rem;
  flex-direction: column;
  gap: 0;
 }
 .hero img {
  height: 37.5rem;
  width: 30rem;
 }
 .bio {
  margin-top: -7rem;
  width: 20.5rem;
 }
 .more-about {
  margin-top: 2rem;
  padding: 1rem 3.5rem;
 }
 .more-about h2 {
  text-align: center;
 }
 .more-about p {
  text-align: justify;
 }
 .icon {
  width: 5.875rem;
  height: 5.25rem;
 }
 .projects-container {
  flex-direction: column;
 }
 .project-container {
  width: 20.875rem;
 }
 .contact-form-container {
  max-width: 23.75rem;
 }
}
@media screen and (max-width: 420px)
{
 .hero img {
  height: 37.5rem;
  width: 23rem;
 }
 .bio {
  width: 18.3rem;
 }
 .project-container {
  width: 17.875rem;
 }
 .contact-form-container {
  max-width: 17.75rem;
 }}
```

JS Code:

```
// Nav hamburgerburger selections
const burger = document.querySelector("#burger-menu");
const ul = document.querySelector("nav ul");
const nav = document.querySelector("nav");
// Scroll to top selection
const scrollUp = document.querySelector("#scroll-up");
// Select nav links
const navLink = document.querySelectorAll(".nav-link");
// Hamburger menu function
burger.addEventListener("click", () => {
ul.classList.toggle("show");});
// Close hamburger menu when a link is clicked
navLink.forEach((link) =>
link.addEventListener("click", () => {
   ul.classList.remove("show");
  }));
// scroll to top functionality
scrollUp.addEventListener("click", () => {
window.scrollTo({
   top: 0,
   left: 0,
   behavior: "smooth",
   }};
});
```

Output:





Result

A professional portfolio website has been designed and developed successfully.

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 create a web application to manage the TO-DO list of users, where users can login and manage their to-do items.

Steps to Implement:

1. Set Up Your New React Project
   - Create a new project: npx create-react-app todo-list-app
2. Create Components
   - Inside the src folder, create components for the TO-DO list and TO-DO items.
3. Implement TO-DO List Component
   - Edit the TO-DO list component to manage TO-DO list data and actions.
4. Implement TO-DO List Component UI
   - Edit the TO-DO list component to display the list of items and input for adding new items.
5. Implement TO-DO Item Component
   - Create a component for individual TO-DO items.
   - Define props and actions (e.g., marking as completed, deleting).
6. Implement TO-DO Item Component UI
   - Edit the TO-DO item component to display an individual item with appropriate actions.
7. Manage State
   - Use React state to manage the list of TO-DO items and their properties.
8. Add Functionality
   - Implement functions to add new items, mark items as completed, and potentially delete items.
9. Styling
   - Add CSS or use a styling library to enhance the appearance of your components.
10. Run the Application
    - Start your React app using:

      npm start

Components Description:

App

- Import React
- Import Components
- Create the App Component
- Export the App Component
- Usage and Rendering

TodoList

- State Initialization:
  - const [todos, setTodos] = useState([]);
  - const [newTodo, setNewTodo] = useState("");
  - The todos state holds an array of to-do items, and newTodo state holds the value of the new to-do item being entered by the user.
- Adding a New To-Do Item:

- When the user types a new to-do item in the input field and clicks the "Add" button, handleAddTodo function is triggered.
- If the trimmed value of newTodo is not empty, a new to-do item is created and added to the todos state array.
- The newTodo state is then reset to an empty string to clear the input field.

• Deleting a To-Do Item:
- Each to-do item in the list has a "Delete" button.
- Clicking the "Delete" button triggers the handleDeleteTodo function with the index of the clicked item.
- The function creates a copy of the current todos array, removes the item at the specified index, and updates the todos state with the new array.

• Toggling Completion Status:
- Each to-do item includes a checkbox to mark the item as complete or incomplete.
- Clicking the checkbox triggers the handleToggleTodo function with the index of the clicked item.
- The function toggles the checked property of the corresponding item in the todos array and updates the state.

• Rendering To-Do Items:
- The todos array is mapped over to render each individual to-do item.
- For each item, a list element (<li>) is generated.
- Each list item contains a checkbox (for completion status) and the text of the to-do item.
- The text is styled with a line-through decoration if the item is checked (completed).
- A "Delete" button is also provided for each to-do item.

• User Interaction:
- The input field allows users to enter a new to-do item.
- The "Add" button triggers the addition of the new to-do item to the list.
- The checkbox allows users to mark to-do items as complete or incomplete.
- The "Delete" button removes the corresponding to-do item from the list.

• Component Rendering:
- The component renders a title, input field, "Add" button, and a list of to-do items.

• Component Export:
- The TodoList component is exported as the default export of the module.

index

• Import React and Dependencies
• Import ReactDOM
• Import Styles and Components
• Render the Application
- Use ReactDOM.createRoot to create a root instance for rendering.
- The root instance is connected to the DOM element with the id of 'root'.
- Within the root, the App component is rendered within a <React.StrictMode> component. StrictMode is used for highlighting potential problems in the application during development.

App.js:

```
import React from "react";
import TodoList from "./TodoList";
const App = () => {
  return (
    <div>
      <TodoList />
    </div>
```

```
  );};
export default App;
```

TodoList.js:

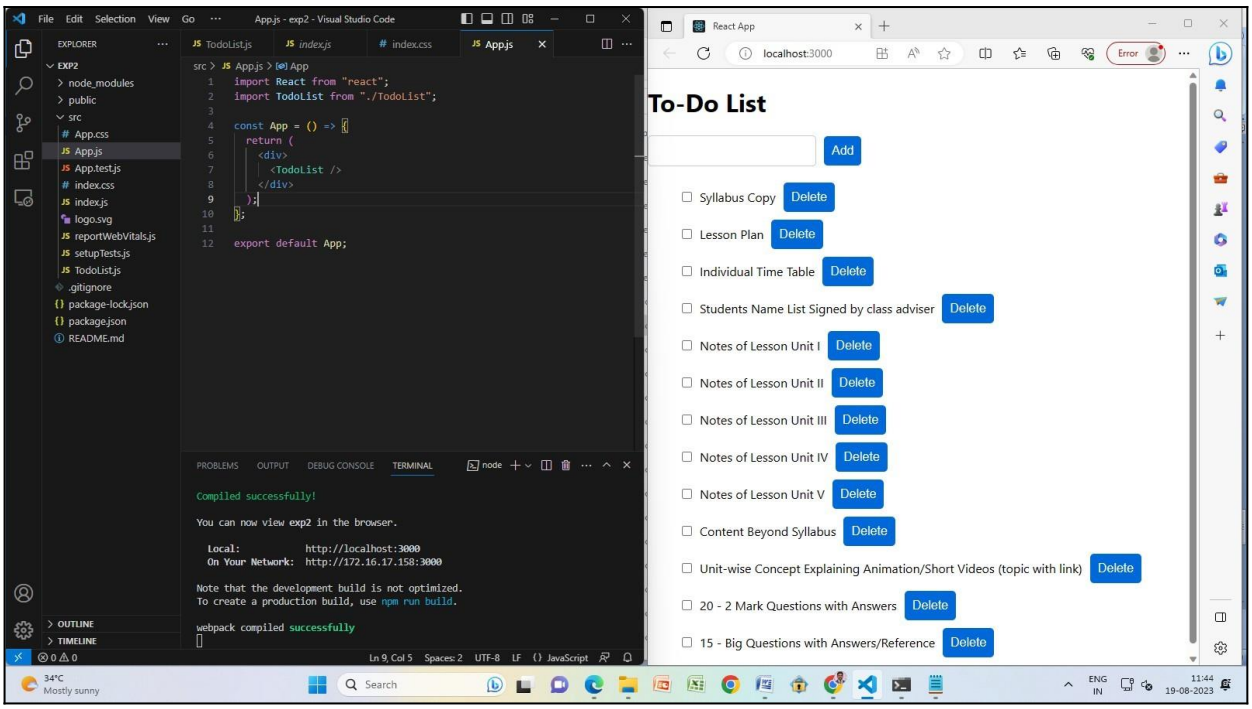
```
import React, { useState } from "react";
const TodoList = () => {
  const [todos, setTodos] = useState([]);
  const [newTodo, setNewTodo] = useState("");
  const handleAddTodo = () => {
    if (newTodo.trim() !== "") {
      setTodos([...todos, { text: newTodo.trim(), checked: false }]);
      setNewTodo("");
    }
  };
  const handleDeleteTodo = (index) => {
    const newTodos = [...todos];
    newTodos.splice(index, 1);
    setTodos(newTodos);
  };
  const handleToggleTodo = (index) => {
    const newTodos = [...todos];
    newTodos[index].checked = !newTodos[index].checked;
    setTodos(newTodos);
  };
  return (
    <div>
      <h1>To-Do List</h1>
      <input
        type="text"
        value={newTodo}
        onChange={(e) => setNewTodo(e.target.value)}
      />
      <button onClick={handleAddTodo}>Add</button>
      <ul>
        {todos.map((todo, index) => (
          <li
            key={index}
            style={{
            display: "flex",
            }}
          >
            <div style={{ display: "flex", alignItems: "center" }}>
              <input
                type="checkbox"
                checked={todo.checked}
                onChange={() => handleToggleTodo(index)}
              />
              <span
                style={{
                  marginRight: "10px",
                  textDecoration: todo.checked ? "line-through" : "none",
                }}
              >
                {todo.text}
              </span>
```

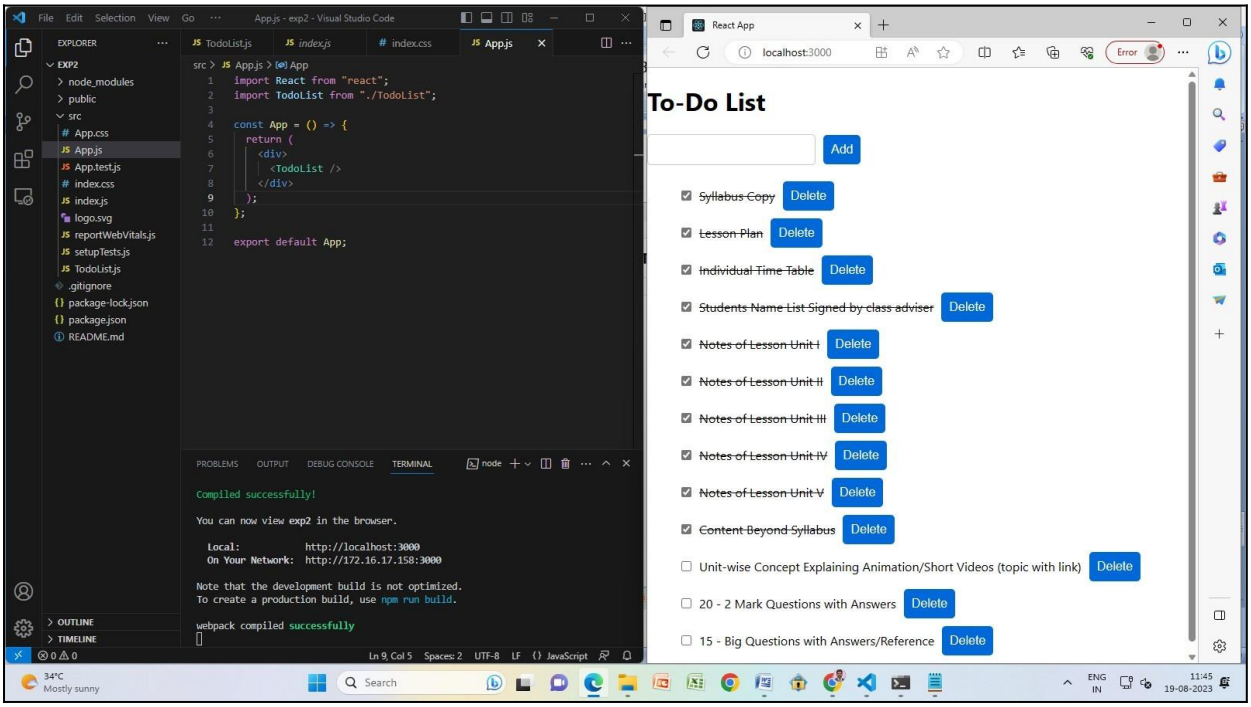
```
            </div>
            <button
              style={{ marginTop: "5px", marginBottom: "5px" }}
              onClick={() => handleDeleteTodo(index)}
            >
              Delete
            </button>
          </li>
        ))}
      </ul>
    </div>
  );
};
export default TodoList;
```

index.js:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
reportWebVitals();
```

Output:





Result

Thus the web application has been created 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.

Aim

To develop a simple micro blogging web application that enabling user registration, content posting.

Steps to Implement:

Steps to create the application:

Step 1: Create React Project

```
npx create-react-app blog
```

Step 2: Change your directory and enter your main folder MY-APP as

```
cd blog
```

Step 3: Install the required modules using the command

```
npm i bootstrap
npm i react-bootstrap
```

Step 4: The dependencies in package.json will look like:

package.json

```
"dependencies": {
   "@testing-library/jest-dom": "^5.16.5",
   "@testing-library/react": "^13.4.0",
   "@testing-library/user-event": "^13.5.0",
   "bootstrap": "^5.3.0",
   "react": "^18.2.0",
   "react-bootstrap": "^2.7.4",
   "react-dom": "^18.2.0",
   "react-scripts": "5.0.1",
   "web-vitals": "^2.1.4"
}
```

Step 5: Create a folder call components and create the files BlogNav.js, Posts.js, Post1.js, Post2.js,

Post3.js, Post4.js

Step 6: Open the terminal and run the project using the command.

```
npm start
```

Components Description:

- **App.js:** This file imports all the components and displays
- **BlogNav.js:** This file is used to create the navigation bar
- **Posts.js:** This file renders the posts
- **Post1.js, Post2.js, Post3.js, Post4.js:** These files contain the content of the blog

App.js

```
import React from "react";
import "./App.css";
import Posts from "./components/Posts";
import Navbar from "./components/BlogNav"
const App = () => {
   return (
      <div className="main-container" style={{backgroundColor: "aliceblue"}}>
         <Navbar />
         <Posts />
      </div>
   );
};
```

```
export default App;
```

Posts.js

```
import React from "react";
import Post1 from "./Post1";
import Post2 from "./Post2";
import Post3 from "./Post3";
import Post4 from "./Post4";
import { Container, Row, Col, Card } from 'react-bootstrap';
const Posts = () => {
    return (
            <Container>
                    <Row className="justify-content-between">
                            <Col md={8} className="mb-4 mt-4">
                                    <Post1 />
                            </Col>
                            <Col md={2} className="mt-4 float-right">
                                    <Card>
                                            <Card.Body>
                                                    <Card.Title>Recent Posts</Card.Title>
                                                    <ul className="list-unstyled">
                                                            <li><a href="#">JavaScript</a></li>
                                                            <li><a                    href="#">Data
Structure</a></li>
                                                            <li><a href="#">Algorithm</a></li>
                                                            <li><a              href="#">Computer
Network</a></li>
                                                    </ul>
                                            </Card.Body>
                                    </Card>
                            </Col>
                            <Col md={8} className="mb-4">
                                    <Post2 />
                            </Col>
                            <Col md={8} className="mb-4">
                                    <Post3 />
                            </Col>
                            <Col md={8} className="mb-4">
                                    <Post4 />
                            </Col>
                    </Row>
            </Container>
    );
};
export default Posts;
```

BlogNav.js

```
import React from "react";
import 'bootstrap/dist/css/bootstrap.css';
import { Navbar, Nav, Form, FormControl } from 'react-bootstrap';
const BlogNav = () => {
    return (
            <div>
                    <Navbar style={{
                            backgroundColor:"#A3C1D4"
```

```
                }}>
                        <img
                src=' D:\nel\blog\Nelson_Pic.png'
                height='30'
                alt="
                loading='lazy'
                />
                        <Navbar.Brand href="#home" style={{color:"white",
marginLeft:"10px"}}>Nelson’s Blog</Navbar.Brand>
                        <Navbar.Toggle />
                        <Navbar.Collapse id="basic-navbar-nav" className="d-flex
justify-content-end">
                                <Nav>
                                        <Nav.Link href="#home" style={{color:"white"}}>
                                                JavaScript
                                        </Nav.Link>
                                        <Nav.Link href="#about" style={{color:"white"}}>
                                                Data Structure
                                        </Nav.Link>
                                        <Nav.Link href="#services"
style={{color:"white"}}>
                                                Algorithm
                                        </Nav.Link>
                                        <Nav.Link href="#contact"
style={{color:"white"}}>
                                                Computer Network
                                        </Nav.Link>
                                </Nav>
                                <Form inline>
                                        <FormControl type="text" placeholder="Search"
className="ml-auto" />
                                </Form>
                        </Navbar.Collapse>
                </Navbar>
        </div>
    )
}
export default BlogNav;
```

Post1.js

```
import { Card } from "react-bootstrap";
const Post1 = () => {
    return (
        <Card>
                <Card.Img
                        variant="top"
                        src=

    "https://media.geeksforgeeks.org/wp-content/cdn-uploads/20230305183140/
Javascript.jpg"
                        width={20}
                        height={250}
                />
                <Card.Body>
```

```
                        <Card.Title>JAVASCRIPT</Card.Title>
                        <Card.Text>
                                JavaScript is the world most popular
                                lightweight, interpreted compiled programming
                                language. It is also known as scripting
                                language for web pages. It is well-known for
                                the development of web pages, many non-browser
                                environments also use it. JavaScript can be
                                used for Client-side developments as well as
                                Server-side developments
                        </Card.Text>
                        <a href="#" className="btn btn-primary">Read More</a>
                </Card.Body>
        </Card>
    );
};
export default Post1;
```

Post2.js

```
import { Card } from "react-bootstrap";
const Post2 = () => {
    return (
        <Card>
                <Card.Img
                        variant="top"
                        src=
"https://media.geeksforgeeks.org/img-practice/banner/coa-gate-2022-thumbnail.png"
                        width={20}
                        height={250}
                />
                <Card.Body>
                        <Card.Title>Data Structure</Card.Title>
                        <Card.Text>
                                The word Algorithm means “a process
                                or set of rules to be followed in calculations
                                or other problem-solving operations”. Therefore
                                Algorithm refers to a set of rules/instructions
                                that step-by-step define how a work is to be
                                executed upon in order to get the expected
                                results.
                        </Card.Text>
                        <a href="#" className="btn btn-primary">Read More</a>
                </Card.Body>
        </Card>
    )
}
export default Post2;
```

Post3.js

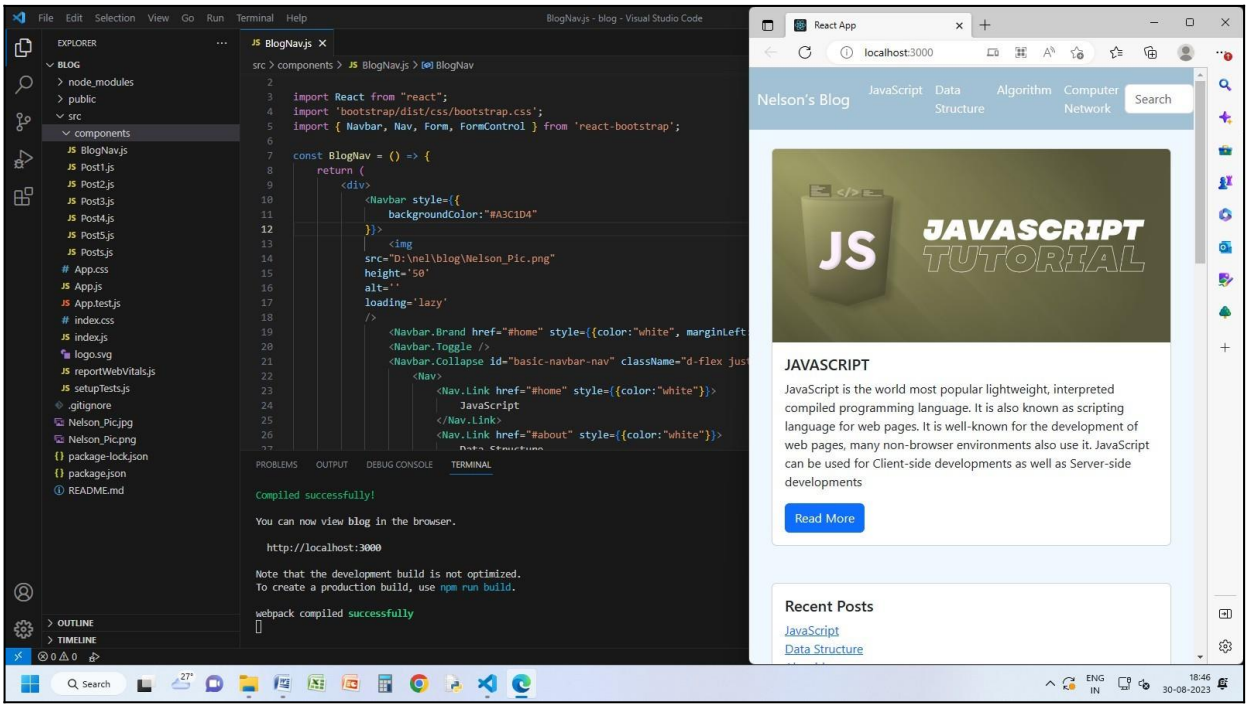
```
import { Card } from "react-bootstrap";
const Post3 = () => {
    return (
        <Card>
                <Card.Img
```

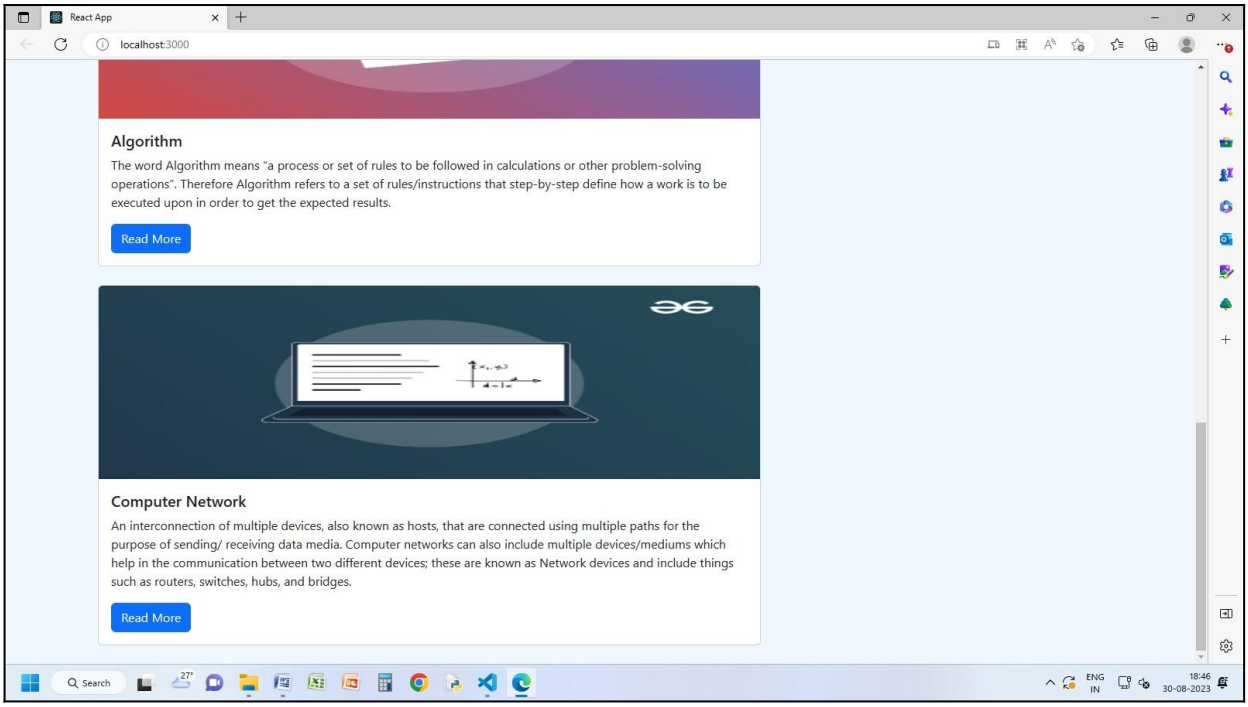
```
                    variant="top"
                    src=
"https://media.geeksforgeeks.org/img-practice/banner/google-test-series-thumbnail.png"
                    width={20}
                    height={250}
                />
                <Card.Body>
                    <Card.Title>Algorithm</Card.Title>
                    <Card.Text>
                        The word Algorithm means “a process
                        or set of rules to be followed in calculations
                        or other problem-solving operations”. Therefore
                        Algorithm refers to a set of rules/instructions
                        that step-by-step define how a work is to be
                        executed upon in order to get the expected
                        results.
                    </Card.Text>
                    <a href="#" className="btn btn-primary">Read More</a>
                </Card.Body>
            </Card>
    )
}
export default Post3;
```

Post4.js

```
import { Card } from "react-bootstrap";
const Post4 = () => {
    return (
            <Card>
                <Card.Img
                    variant="top"
                    src=
"https://media.geeksforgeeks.org/img-practice/banner/cp-maths-java-thumbnail.png"
                    width={20}
                    height={250}
                />
                <Card.Body>
                    <Card.Title>Computer Network</Card.Title>
                    <Card.Text>
                        An interconnection of multiple devices,
                        also known as hosts, that are connected using
                        multiple paths for the purpose of sending/
                        receiving data media. Computer networks can
                        also include multiple devices/mediums which
                        help in the communication between two different
                        devices; these are known as Network devices
                        and include things such as routers, switches,
                        hubs, and bridges.
                    </Card.Text>
                    <a href="#" className="btn btn-primary">Read More</a>
                </Card.Body>
            </Card>
    )}
export default Post4;
```

Output:





Result

Thus a simple micro blogging platform has been created successfully.

4. Create a food delivery website where users can order food from a particular restaurant listed in the website.

Aim

To create a user-friendly React-based food delivery website to browse and customize orders.

Steps to Implement:

Steps to create the application:

Step 1: Create a React App

npx create-react-app shopping-cart-app

Step 2: Change your working directory to the newly created app:

cd shopping-cart-app

Step 3: Functionality - Replace the src Directory "App.js" file Contents with the below code.

Step 4: Styling - Replace the src Directory "App.css" file contents with the below code.

Step 5: Running the App

npm start

Components Description:

- **App.js:** It allows the user to view the list of products, add them to the cart, adjust quantities, and see the total cost.
- **App.css:** contains styling rules for a React-based shopping cart application, including the app container, shopping modal, product list, cart, quantity, total cost, and buttons, ensuring a visually appealing and functional user interface.

App.js:

```
import React, { useState } from 'react';
import './App.css';
function App() {
  const [isShoppingOpen, setIsShoppingOpen] = useState(false);
  const [products, setProducts] = useState([
    {
      id: 1,
      name: 'Chat Masala',
      image: '1.png',
      price: 120000,
    },

 {
      id: 2,
      name: 'Tandoori',
      image: '2.PNG',
      price: 120000,
    },

 {
      id: 3,
      name: 'Chicken Salad',
      image: '3.PNG',
      price: 220000,
    },

 {
      id: 4,
```

```
      name: 'Pumpkin Soup',
      image: '4.PNG',
      price: 123000,
    },

  {
      id: 5,
      name: 'Veg Salad',
      image: '5.PNG',
      price: 320000,
    },

 {
      id: 6,
      name: 'Paneer Pizza',
      image: '6.PNG',
      price: 120000,
    },
  ]);
  const [cart, setCart] = useState([]);
  const openShopping = () => {
  setIsShoppingOpen(true);
  };
  const closeShopping = () => {
    setIsShoppingOpen(false);
  };
  const addToCart = (key) => {
    const product = products[key];
    const cartItem = {
      ...product,
      quantity: 1,
    };
    setCart([...cart, cartItem]);
  };
  const changeQuantity = (key, newQuantity) => {
    if (newQuantity === 0) {
      const updatedCart = [...cart];
      updatedCart.splice(key, 1);
      setCart(updatedCart);
    } else {
      const updatedCart = [...cart];
      updatedCart[key].quantity = newQuantity;
      updatedCart[key].price = newQuantity * products[key].price;
      setCart(updatedCart);
    }
  };
  return (
    <div className={`App ${isShoppingOpen ? 'active' : ''}`}>
      <div className="shopping">
        <button onClick={openShopping}>Open Shopping</button>
      </div>
      <div className="closeShopping">
        <button onClick={closeShopping}>Close Shopping</button>
      </div>
```

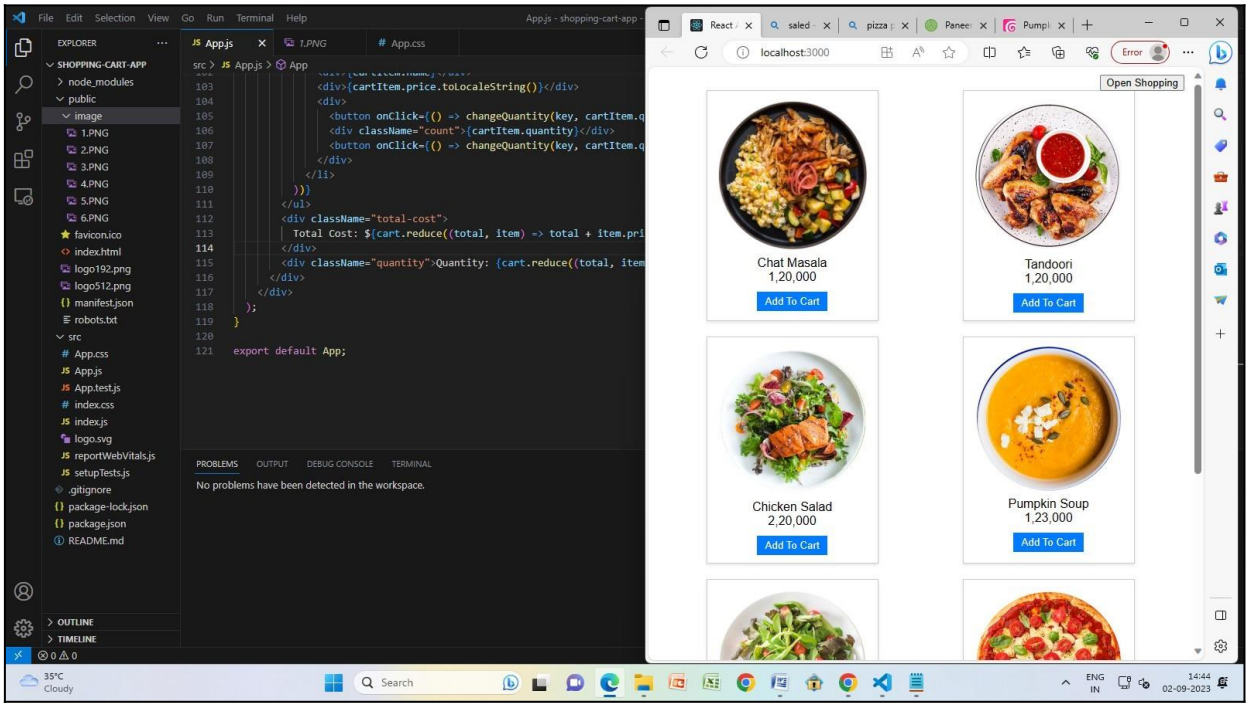
```
      <div className="list">
        {products.map((product, key) => (
          <div className="item" key={key}>
            <img src={`image/${product.image}`} alt={product.name} />
            <div className="title">{product.name}</div>
            <div className="price">{product.price.toLocaleString()}</div>
            <button onClick={() => addToCart(key)}>Add To Cart</button>
          </div>
        ))}
      </div>
      <div className="listCard">
        <ul>
          {cart.map((cartItem, key) => (
            <li key={key}>
              <div>
                <img src={`image/${cartItem.image}`} alt={cartItem.name} />
              </div>
              <div>{cartItem.name}</div>
              <div>{cartItem.price.toLocaleString()}</div>
              <div>
                <button onClick={() => changeQuantity(key, cartItem.quantity - 1)}>-</button>
                <div className="count">{cartItem.quantity}</div>
                <button onClick={() => changeQuantity(key, cartItem.quantity + 1)}>+</button>
              </div>
            </li>
          ))}
        </ul>
        <div className="total-cost">
          Total Cost: ${cart.reduce((total, item) => total + item.price, 0).toLocaleString()}
        </div>
              <div className="quantity">Quantity: {cart.reduce((total, item) => total +
item.quantity, 0)}</div>
      </div>
    </div>
  );
}
export default App;
```

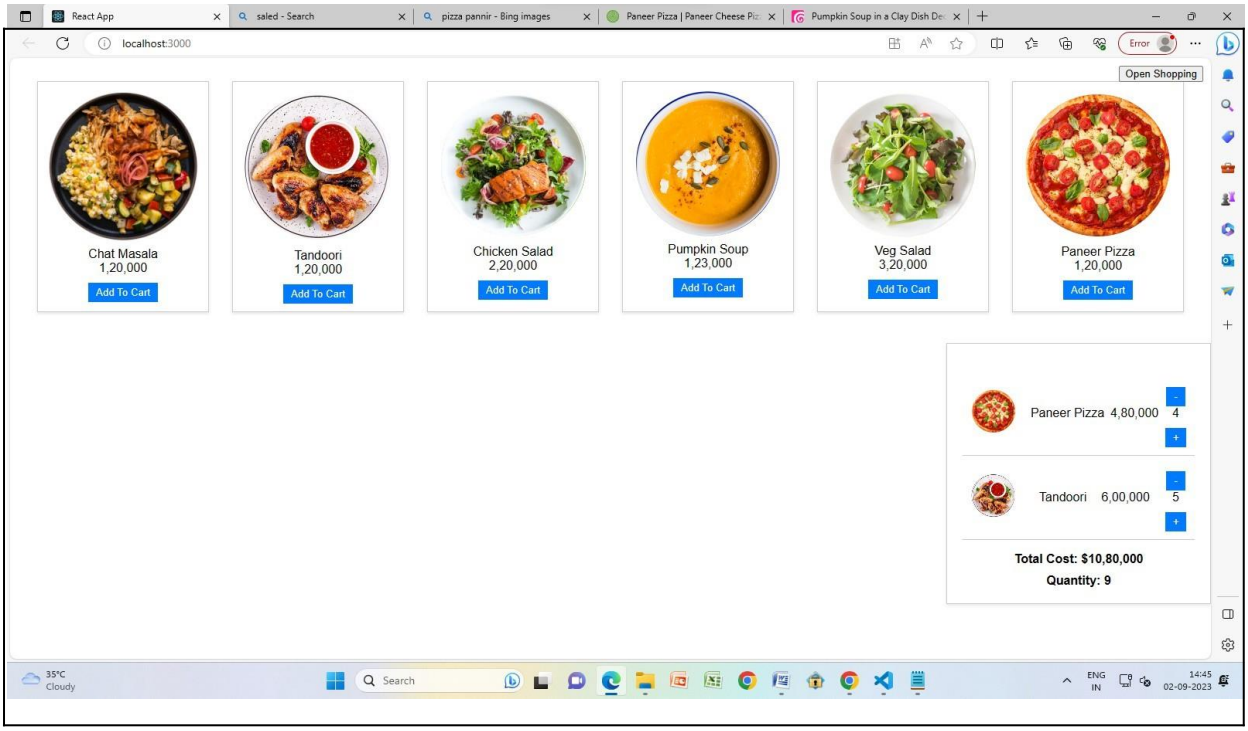
App.css

```
/* App Container */
.App {
  text-align: center;
  font-family: Arial, sans-serif;
}
/* Styling for the Shopping Cart Modal
*/
.shopping {
  position: absolute;
  top: 10px;
  right: 10px;
}
.closeShopping {
  position: absolute;
  top: 10px;
  right: 10px;
  display: none;
}
.shopping.active + .closeShopping {
  display: block;
}
/* Styling for the Product List */
.list {
  display: flex;
  flex-wrap: wrap;
  justify-content: space-around;
  padding: 20px;
}
```

```
.item {
  width:   200px;
  margin:   10px;
  padding: 10px;
  border: 1px solid #ccc;
  background-color: #fff;
  box-shadow: 0px 2px 4px rgba(0, 0, 0,
0.1);
  text-align: center;
}
.item img {
  max-width: 100%;
}
/* Styling for the Cart */
.listCard {
  width: 300px;
  padding: 20px;
  margin-top: 10px;
  border: 1px solid #ccc;
  background-color: #fff;
  box-shadow: 0px 2px 4px rgba(0, 0, 0,
0.1);
  float: right;
}
.listCard ul {
  list-style: none;
  padding: 0;
}
.listCard li {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 10px;
  border-bottom: 1px solid #ccc;
}
.listCard img {
  max-width: 60px;
  max-height: 60px;
  margin-right: 10px;
}
/* Styling for Quantity and Total Cost */
.quantity,
.total-cost {
  margin-top: 10px;
  font-weight: bold;
}
/* Styling for Add To Cart Button */
.item button,
.listCard button {
  background-color: #007bff;
  color: #fff;
  border: none;
  padding: 5px 10px;
  cursor: pointer;
  margin-top: 10px;
}
.item button:hover,
.listCard button:hover {
  background-color: #0056b3;
}
/* Styling for Increment/Decrement
Buttons */
.listCard .count {
  margin: 0 10px;
}
```

OUTPUT:





Result

Thus the food delivery website with menu browsing and order placement using React has been created successfully.

5. Develop a classifieds web application to buy and sell used products.

Aim

To create a classifieds webpage where users can search for products based on keywords and view detailed product listings.

Steps to Implement:

Steps to create the application:
Step 1: Create your project structure according to the requirements and entered the code properly.
Step 2: Install the dependencies (npm install express)
Step 3: Run the Server (node server.js)
Step 4: Open your web browser and navigate to http://localhost:3000 to access your classifieds
webpage.

Components Description:

index.html

HTML file defining the structure of a classifieds webpage with sections for products and including external CSS and JavaScript files.

styles.css

CSS file containing styles for resetting default styles, defining the page layout, styling the header, navigation menu, search section, and product listings, and making the design responsive.

script.js

JavaScript code that handles the functionality of the webpage, including searching for products based on user input and displaying filtered product listings.

server.js

Node.js server script using Express to serve static files (HTML, CSS, JavaScript, and images) and listening on port 3000 to host the classifieds webpage.

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <link rel="stylesheet" href="styles.css">
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Classifieds - Buy and Sell</title>
</head>
<body>
  <header>
    <h1>Classifieds - Buy and Sell</h1>
  </header>
  <div class="container">
    <section id="search-section">
      <input type="text" id="search-input" placeholder="Search for products">
      <button id="search-button">Search</button>
    </section>
    <section id="products">
      <h2>Featured Products</h2>
```

```
            <ul id="product-list">
                <!-- Product listings will be displayed here -->
            </ul>
        </section>
    </div>
    <script src="script.js"></script>
</body>
</html>
```

styles.css

```
/* Reset some default styles for consistency */
body, h1, h2, h3, p {
    margin: 0;
    padding: 0;
}
body {
    font-family: Arial, sans-serif;
    background-color: #f0f0f0;
    color: #333;
    margin: 0;
    padding: 0;
}
.container {
    max-width: 1200px;
    margin: 0 auto;
    padding: 20px;
}
h1 {
    font-size: 28px;
    margin-bottom: 20px;
}
h2 {
    font-size: 24px;
    margin-bottom: 15px;
}
ul {
    list-style: none;
}
/* Header styles */
header {
    background-color: #3498db;
    color: #fff;
    padding: 10px 0;
}
header h1 {
    font-size: 36px;
    text-align: center;
}
/* Navigation menu styles */
nav {
    text-align: center;
}
nav ul {
    display: inline-block;
    list-style: none;
}
nav ul li {
    display: inline;
    margin-right: 20px;
}
nav a {
    color: #fff;
    text-decoration: none;
    font-weight: bold;
    font-size: 16px;
}
/* Search section styles */
#search-section {
    background-color: #fff;
    padding: 20px;
    border-radius: 5px;
     box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    margin-bottom: 20px;
}
#search-input {
    width: 70%;
    padding: 10px;
    border: 1px solid #ccc;
    border-radius: 3px;
    margin-right: 10px;
}
#search-button {
    padding: 10px 20px;
    background-color: #3498db;
    color: #fff;
    border: none;
    border-radius: 3px;
    cursor: pointer;
    font-weight: bold;
}
/* Product listings styles */
#products {
    background-color: #fff;
    padding: 20px;
```

```
    border-radius: 5px;
     box-shadow: 0 0 10px rgba(0, 0, 0,
0.1);
}
#product-list {
    display: grid;
      grid-template-columns: repeat(auto-
fill, minmax(250px, 1fr));
    gap: 20px;
}
.product {
    border: 1px solid #ccc;
    padding: 10px;
    border-radius: 5px;
    background-color: #fff;
}
.product h3 {
    font-size: 18px;
    margin-bottom: 10px;
}
.product p {
    font-size: 14px;
    margin-bottom: 5px;
}
.product img {
    max-width: 100%;
    height: auto;
    border: 1px solid #ddd;
    border-radius: 3px;
}
/* Responsive design */
@media (max-width: 768px) {
    #search-input {
        width: 60%;
    }
}
```

script.js

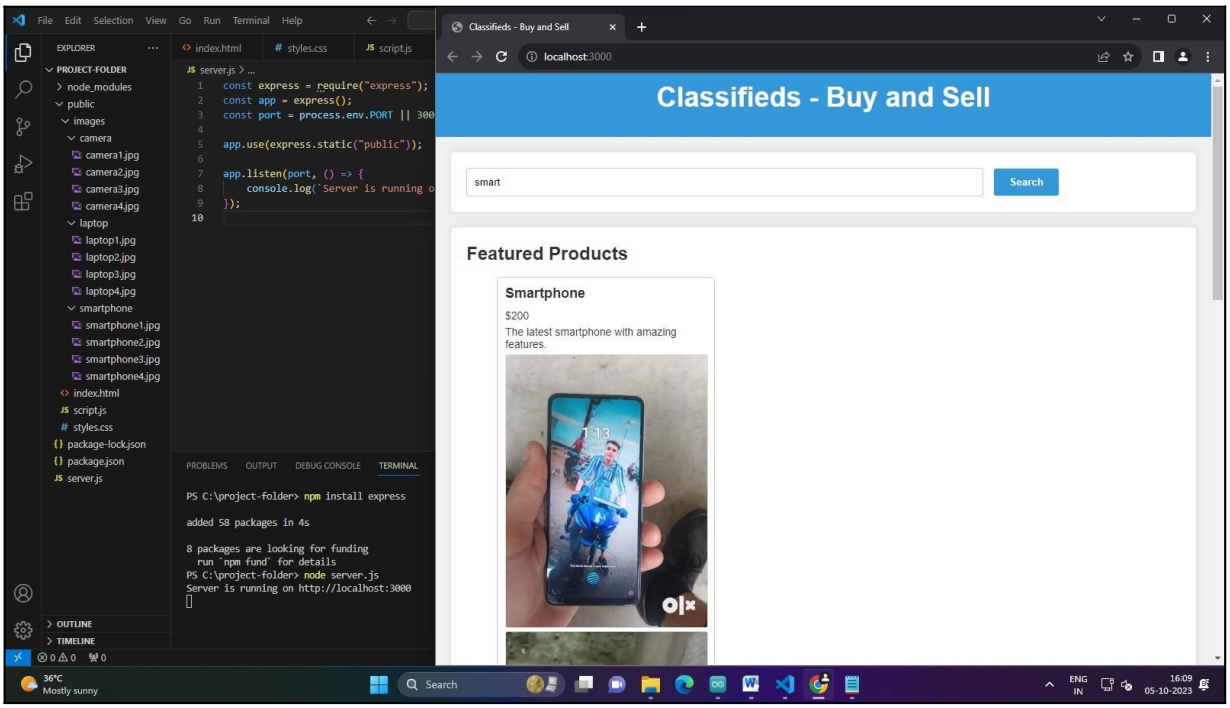
```
document.addEventListener("DOMContentLoaded", () => {
    const searchButton = document.getElementById("search-button");
    const searchInput = document.getElementById("search-input");
    const productList = document.getElementById("product-list");
    searchButton.addEventListener("click", () => {
        const searchTerm = searchInput.value.toLowerCase();
                                    const filteredProducts = products.filter(product =>
product.title.toLowerCase().includes(searchTerm));
        displayProducts(filteredProducts);
    });
    function displayProducts(productArray) {
        productList.innerHTML = "";
        productArray.forEach(product => {
            const listItem = document.createElement("li");
            listItem.classList.add("product");
            listItem.innerHTML = `
                <h3>${product.title}</h3>
                <p>${product.price}</p>
                <p>${product.description}</p>
                <div class="product-images">
                        ${product.images.map(image => `<img src="images/${product.category}/$
{image}" alt="${product.title}">`).join("")}
                </div>
            `;
            productList.appendChild(listItem);
        });
    }
});
// Example data (you can replace this with your data)
const products = [
    {
        id: 1,
        title: "Used Laptop",
        price: "$300",
```

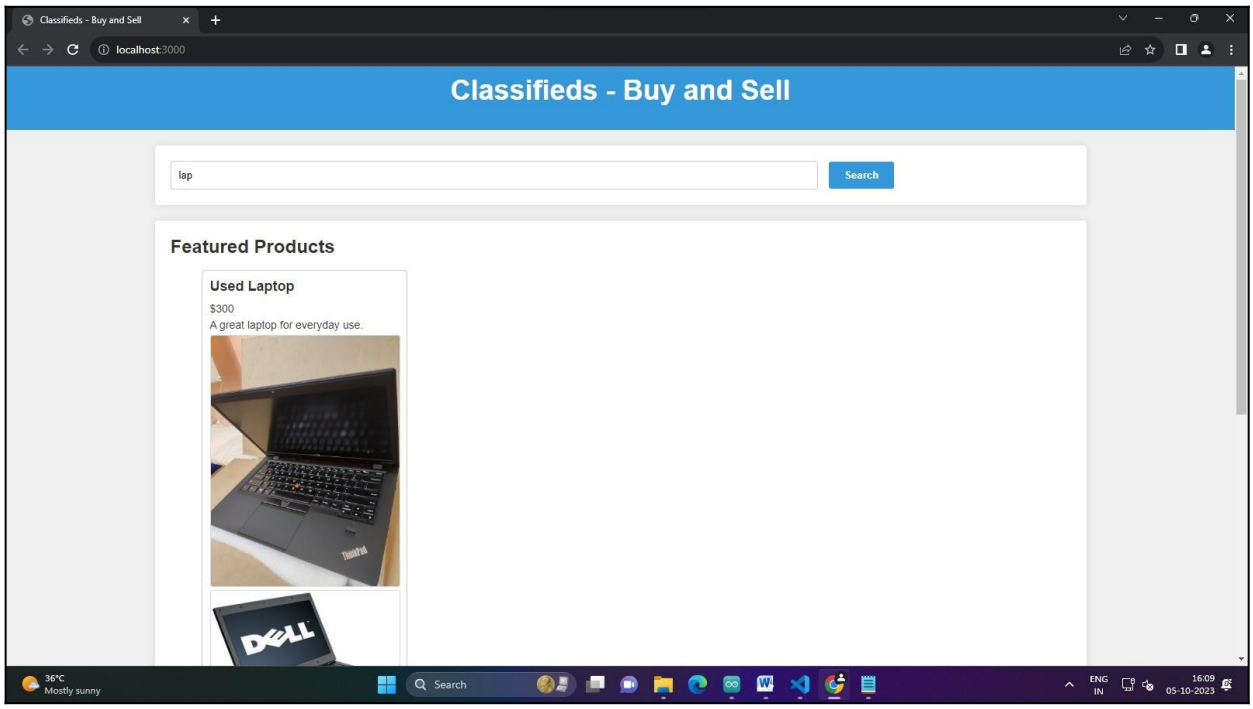
```
    description: "A great laptop for everyday use.",
    category: "laptop",
    images: ["laptop1.jpg", "laptop2.jpg", "laptop3.jpg", "laptop4.jpg"],
  },
  {
    id: 2,
    title: "Smartphone",
    price: "$200",
    description: "The latest smartphone with amazing features.",
    category: "smartphone",
                images: ["smartphone1.jpg", "smartphone2.jpg", "smartphone3.jpg",
"smartphone4.jpg"],
  },
  {
    id: 3,
    title: "Camera",
    price: "$400",
    description: "A high-quality camera for photography enthusiasts.",
    category: "camera",
    images: ["camera1.jpg", "camera2.jpg", "camera3.jpg", "camera4.jpg"],
  },
];
```

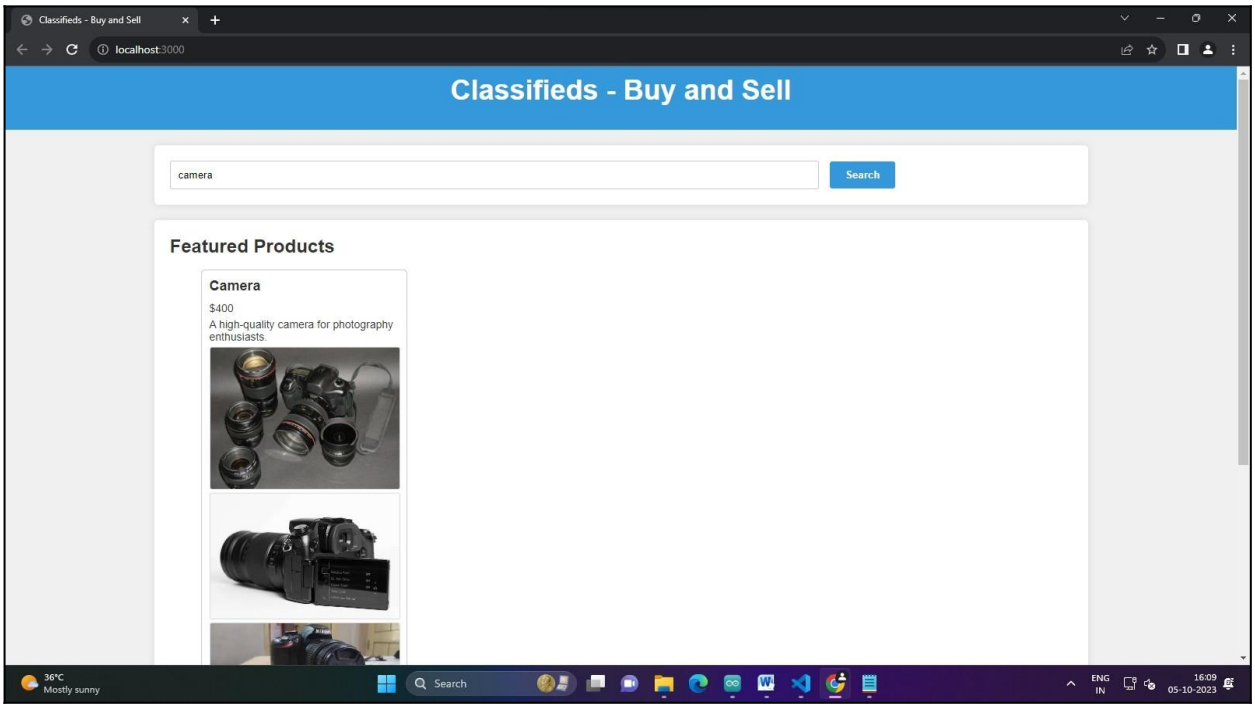
server.js

```
const express = require("express");
const app = express();
const port = process.env.PORT || 3000;
app.use(express.static("public"));
app.listen(port, () => {
  console.log(`Server is running on http://localhost:${port}`);
});
```

Output:







Result

Thus a functional classifieds webpage has been created, that allows users to search for products, with filtered results displayed dynamically.

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 design and implement a user-friendly Leave Management System enabling employees to apply for different types of leaves (casual and medical) and check their leave balances, enhancing organizational efficiency and employee satisfaction.

Steps to Implement:

Steps to create the application:

Step 1: Create a new React application using Create React App and install React Router DOM for handling routing.

Step 2: Inside the frontend/src directory, create a "components" folder. Inside the "components" folder, create Login.js, Signup.js, and Leavemanagement.js files with respective functionalities for user authentication and leave management.

Step 3: Implement the frontend components. Login.js handles user login, Signup.js handles user registration, and Leavemanagement.js handles leave application and balance display.

Step 4: Create a CSS file (Leavemanagement.css) for styling the components.

Step 5: Configure routes in App.js using React Router DOM to navigate between Login, Signup, and Leavemanagement components.

Step 6: Set up the backend:

Create a folder named "backend."

Inside the backend folder, run npm init -y to initialize a Node.js application.

Install required packages: npm install express mongoose body-parser bcrypt cors dotenv

Step 7: Inside the backend folder, create "models" and "routes" folders.

In the "models" folder, create a leave.js file defining the MongoDB schema for user leave data.

In the "routes" folder, create login.js, signup.js, and leave.js files for handling authentication, registration, and leave management routes.

Step 8: Implement the backend routes:

login.js: Handle user login, validate credentials, and return user data.

signup.js: Handle user registration and save user data to the database.

leave.js: Handle leave application, deduct leave days from the user's balance, and update the database.

Step 9: Create an index.js file in the backend folder to set up Express server, connect to MongoDB, and define API routes.

Step 10: Start both frontend and backend:

In the frontend directory, run npm start to start the React development server.

In the backend directory, run node index.js to start the Express server.

Components Description:

Frontend

Login.js

Login.js is a React component enabling user authentication. It captures username and password, sends a login request to a server, and redirects users to a leave management page upon successful login, also offering a signup option.

Signup.js

Signup.js is a React component allowing users to register by providing a username and password. It sends a signup request to a server, displaying a success or error message, and offers a link to the login page.

Leavemanagement.js

Leavemanagement.js is a React component managing leave applications. It displays user information, allows applying for casual or medical leave, updates leave balances, and provides real-time feedback on available leave days.

Leavemanagement.css

Leavemanagement.css is a CSS file defining styles for a leave management application. It includes font imports, sets up a clean layout with centered headings and responsive form elements. The CSS animations slideInLeft and slideInRight create smooth entry effects for the form and balance containers, enhancing the user interface.

App.js

App.js is the main file of a React application, defining routes using the React Router library. It sets up routes for login, signup, and leave management components, allowing seamless navigation between different parts of the application based on the URL path.

index.js

index.js is the entry point of a React application. It uses ReactDOM to render the main App component inside a Router component from React Router. The App component and its defined routes are then rendered in the HTML element with the id 'root', effectively initializing the React application and enabling navigation between different components based on the URL path.

Backend

leave.js

leave.js is a Node.js module defining a Mongoose schema for user data. It specifies the structure of user documents, including fields for username, password, casual leave balance, and medical leave balance. The schema is used to create a Mongoose model named 'User', which is then exported for use in other parts of the application. This schema is typically utilized for database operations, enabling the storage and retrieval of user-related information in a MongoDB database.

Routes

login.js

login.js is a Node.js module implementing user authentication logic for a login endpoint. It verifies incoming username and password, responds with user details if authentication is successful, and returns an error message if authentication fails. The module uses the User model and Express.js for routing.

signup.js

signup.js is a Node.js module that manages user registration for a signup endpoint. It checks if a user with the given username already exists in the database and, if not, creates a new user with default leave balances. It provides a success message upon successful registration and handles errors gracefully. This module utilizes the User model and Express.js for routing.

leave.js

leave.js is a Node.js module serving as a backend API for managing user leave balances. It includes routes for retrieving a user's leave information by ID (GET request) and updating their leave balances based on leave type and duration (PUT request). The module checks if the requested leave balance update is valid and responds with success messages or error messages if limits are exceeded. It utilizes the User model and Express.js for routing.

index.js

index.js is the main file of a Node.js server application. It creates an Express server, establishes a connection to a MongoDB database, configures routes for user authentication and leave management, and listens on a specific port. The server handles incoming requests for user login, registration, and leave management operations, providing essential functionality for the associated frontend application.

Login.js

```
// LEAVEMANAGEMENT\frontend\src\components
import React, { useState } from 'react';
import { useNavigate } from "react-router-dom";
function Login() {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [message, setMessage] = useState('');
  const [objectid, setObjectid] = useState('');
  const navigate=useNavigate();
  function handlesignup(){
    navigate("/signup")
  }
  const handleLogin = async () => {
    try {
      const response = await fetch('http://localhost:3001/login', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({ username, password }),
      });
      if (response.status === 200) {
        const data = await response.json();
        setObjectid(data.userId);
        console.log("User ID : ",data.userId);
        console.log("fjflf balance : ",data.leavebalance);
navigate("/Leavemanagement/",{state:
{id:data.userId,casual:data.casual,medical:data.medical,username:data.username1}});
      } else {
        const data = await response.json();
        setMessage(data.message);
      }
    } catch (error) {
      console.error(error);
    }
  };
  return (
    <div>
```

```
      <h2>Login</h2>
      <p>{objectid}</p>
      <input
        type="text"
        placeholder="Username"
        value={username}
        onChange={(e) => setUsername(e.target.value)}
      />
      <input
        type="password"
        placeholder="Password"
        value={password}
        onChange={(e) => setPassword(e.target.value)}
      />
      <button onClick={handleLogin}>Login</button>
      <p>{message}</p>
      <br>
      </br>
      <h1 onClick={handlesignup}>Signup</h1>
    </div>
  );
}
export default Login;
```

Signup.js

//LEAVEMANAGEMENT\frontend\src\components

```
import React, { useState } from 'react';
import { useNavigate } from "react-router-dom";
function Signup() {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [message, setMessage] = useState('');
  const navigate=useNavigate();
  function handleLogin(){
    navigate("/")
  }
  const handleSignup = async () => {
    try {
      const response = await fetch('http://localhost:3001/signup', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({ username, password }),
      });
      if (response.status === 201) {
        const data = await response.json();
        setMessage(data.message);
      } else {
        const data = await response.json();
        setMessage(data.message);
      }
    } catch (error) {
      console.error(error);
```

```
    }
  };
  return (
    <div>
      <h2>Signup</h2>
      <input
        type="text"
        placeholder="Username"
        value={username}
        onChange={(e) => setUsername(e.target.value)}
      />
      <input
        type="password"
        placeholder="Password"
        value={password}
        onChange={(e) => setPassword(e.target.value)}
      />
      <button onClick={handleSignup}>Signup</button>
      <p>{message}</p>
      <br>
      </br>
      <h1 onClick={handleLogin}>Login</h1>
    </div>
  );
}
export default Signup;
```

Leavemanagement.js

```
// LEAVEMANAGEMENT\frontend\src\components
import React, { useState, useEffect } from "react";
import './Leavemanagement.css'
import { useLocation } from "react-router-dom";
const Leavemanagement = () => {
  const location = useLocation();
  const [users, setUsers] = useState();
  const [leavebalanceData, setLeaveBalanceData] = useState(location.state.casual);
  const [message, setMessage] = useState("Welcome!!!");
  const [medical, setMedical] = useState(location.state.medical);
  const [casual, setCasual] = useState(location.state.casual);
  const [data, setData] = useState([]);
  const [leaveType, setLeaveType] = useState("casual");
  const [leaveDays, setLeaveDays] = useState("");
  const id = location.state.id;
  const name=location.state.username;
  const handleSignup = async () => {
  fetchUsers();
    try {
      const response = await fetch('http://localhost:3001/leave', {
        method: 'PUT',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({ id, leaveType, leaveDays}),
      });
```

```
    if (response.status === 200) {
      const data = await response.json();
      setMessage(data.message);
      setCasual(data.casual);
      setMedical(data.medical);
      if (leaveType == "casual") {
        console.log("casual",data.casual);
        setLeaveBalanceData(data.casual);
      }
      else if (leaveType == "medical") {
        console.log("medical 1",data.medical);
        setLeaveBalanceData(data.medical);
      }
      Change();
      } else {
      const data = await response.json();
      setMessage(data.message);
    }
  } catch (error) {
    console.error(error);
  }
};
const Change=() =>{
  if (leaveType == "casual") {
    console.log("casual",casual);
    setLeaveBalanceData(casual);
  }
  else if (leaveType == "medical") {
    console.log("medical",medical);
    setLeaveBalanceData(medical);
  }
}
const fetchUsers = async () => {
  try {
    const response = await fetch(`http://localhost:3001/leave/${id}`);
    const data = await response.json();
    if (response.ok) {
      if (leaveType == "casual") {
        console.log("casual",data.casual);
        setLeaveBalanceData(data.casual);
      }
      else if (leaveType == "medical") {
        console.log("medical 1",data.medical);
        setLeaveBalanceData(data.medical);
      }
      setData(data);
    } else {
      console.error('Failed to fetch users:', data.message);
    }
  } catch (error) {
    console.error('Error during user fetch:', error.message);
  }
};
const handleLeaveTypeChange = (event) => {
```

```
  setLeaveType(event.target.value);
  if (event.target.value == "casual") {
  setLeaveBalanceData(casual);
    console.log(event.target.value,casual,leavebalanceData);
  }
  else if (event.target.value == "medical") {
    setLeaveBalanceData(medical);
    console.log(event.target.value,medical,leavebalanceData);
  }
};
const handleLeaveDaysChange = (event) => {
  setLeaveDays(event.target.value);
};
const handleSubmit = (event) => {
  event.preventDefault();
  const days = parseInt(leaveDays);
  setLeaveDays("");
  updateleave();
  handleSignup();
};
const updateleave = async () => {
  try {
    const response = await fetch(`http://localhost:3001/leave/${id}`);
    const data = await response.json();
    if (response.ok) {
      setUsers(data.leavebalance);
    } else {
      console.error('Failed to fetch users:', data.message);
    }
  } catch (error) {
    console.error('Error during user fetch:', error.message);
  }
};
/*const reset = () =>{
  setLeaveBalanceData({
    ...leaveBalanceData,
    [leaveType]: leaveBalanceData[leaveType] + 5,
  });
}
<button type="submit" onClick={() => reset()}>Reset</button>
*/
return (
  <div className="container">
    <h1>Leave Management System</h1>
    <div className="form-container">
    <h2>Leave Application</h2>
      <table><tr>
        <td>ID</td><td>: </td><td>{location.state.id}</td></tr>
       <tr> <td>Name</td><td>: </td><td>{name}</td></tr>
        <tr><td>Message</td><td>: </td><td>{message}</td></tr>
      </table>
      <br></br>
      <form onSubmit={handleSubmit}>
        <select value={leaveType} onChange={handleLeaveTypeChange}>
```

```
            <option value="casual">Casual Leave</option>
            <option value="medical">Medical Leave</option>
          </select>
          <input
            type="number"
            min={1}
            max={10}
            value={leaveDays}
            onChange={handleLeaveDaysChange}
            placeholder="No. of days"
            required
          />
          <button type="submit">Apply Leave</button>
        </form>
      </div>
      <div className="balance-container">
        <h2>Leave Balance</h2>
        <div className="balance">
        <p>Available casual leave: {casual}   medical leave: {medical}</p>
        </div>
      </div>
    </div>
  );
//Available {leaveType} Leaves: {leavebalanceData}
};
export default Leavemanagement;
```

Leavemanagement.css

// LEAVEMANAGEMENT\frontend\src\components

```
@import url('https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500&display=swap');
body {
  font-family: 'Roboto', sans-serif;
  margin: 0;
  padding: 0;
  background-color: #f5f5f5;
}
.container {
  width: 80%;
  margin: auto;
}
h1 {
  text-align: center;
  color: #4CAF50;
}
.form-container,
.balance-container {
  background-color: #fff;
  margin-top: 20px;
  padding: 20px;
  border-radius: 5px;
}
form input[type="number"]{
  width: 98%;
```

```
  padding: 10px;
  margin-bottom: 10px;
}
form select,
form button {
width: 100%;
padding: 10px;
  margin-bottom: 10px;
}
form button {
  background-color: #4CAF50;
  color: #fff;
}
.balance-container {
   animation-name: slideInRight;
   animation-duration: .5s;
   animation-fill-mode: both;
}
.form-container {
   animation-name: slideInLeft;
   animation-duration: .5s;
   animation-fill-mode: both;
}
@keyframes slideInLeft {
   from { transform: translateX(-100%); }
   to { transform: translateX(0); }
}
@keyframes slideInRight {
   from { transform: translateX(100%); }
   to { transform: translateX(0); }
}
```

App.js

// LEAVEMANAGEMENT\leavemanagement\src

```
import React from 'react';
import Login from './components/Login';
import Signup from './components/Signup';
import Leavemanagement from './components/Leavemanagement';
import { Route,Routes} from "react-router-dom";
function App() {
  return (
    <div className="App">
      <Routes>
        <Route path='/' element={<Login />} />
      <Route path='/signup' element={<Signup />} />
      <Route path='/Leavemanagement' element={<Leavemanagement />} />
        </Routes>
    </div>
);
}
export default App;
```

index.js

//LEAVEMANAGEMENT\leavemanagement\src

```
import React from 'react';
import ReactDOM from 'react-dom';
import { BrowserRouter as Router } from 'react-router-dom';
import App from './App';
ReactDOM.render(
  <Router>
    <App />
  </Router>,
  document.getElementById('root')
);
```

leave.js

```
// LEAVEMANAGEMENT\backend\models
const mongoose = require('mongoose');
const userSchema = new mongoose.Schema({
  username: {
    type: String,
    required: true,
    unique: true,
  },
  password: {
    type: String,
    required: true,
  },
  casual: {
    type: String,
  },
  medical: {
    type: String,
  },
});
const User = mongoose.model('User', userSchema);
module.exports = User;
```

login.js

```
//LEAVEMANAGEMENT\backend\routes
const express = require('express');
const router = express.Router();
//const bcrypt = require('bcrypt');
const User = require('../models/leave'); // Assuming you have a User model
router.post('/', async (req, res) => {
  const { username, password } = req.body;
  try {
    const user = await User.findOne({ username });
    if (!user) {
      return res.status(401).json({ message: 'Authentication failed' });
    }
    //const isPasswordValid = await bcrypt.compare(password, user.password);
    //if (!isPasswordValid) {
      const userId = user._id;
      const medical=user.medical;
      const casual=user.casual;const username1=user.username;
      if(password!=user.password){
```

```
      return res.status(401).json({ message: 'Authentication failed' });
    }
    // Handle successful login
    res.status(200).json({ message: 'Login successful',userId,casual,medical,username1});
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: 'Server error' });
  }
});
module.exports = router;
```

signup.js

```
// LEAVEMANAGEMENT\backend\routes
const express = require('express');
const router = express.Router();
//const bcrypt = require('bcrypt');
const User = require('../models/leave');
router.post('/', async (req, res) => {
  try {
    const { username, password } = req.body;
    const existingUser = await User.findOne({ username });
    if (existingUser) {
      return res.status(409).json({ message: 'User already exists' });
    }
    const casual="5";
    const medical="5";
    //const hashedPassword = await bcrypt.hash(password, 10);
    const newUser = new User({
      username,
      //password: hashedPassword,
      password,casual,medical,
    });
    await newUser.save();
    res.status(201).json({ message: 'Signup successful' });
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: 'Server error' });
  }
});
module.exports = router;
```

leave.js

```
//LEAVEMANAGEMENT/backend/routes
const express = require('express');
const router = express.Router();
const mongoose = require('mongoose');
//const bcrypt = require('bcrypt');
const User = require('../models/leave');
router.get('/:id', async (req, res) => {
try {
    const user = await User.findById(req.params.id);
    console.log(req.params.id);
    if (!user) {
```

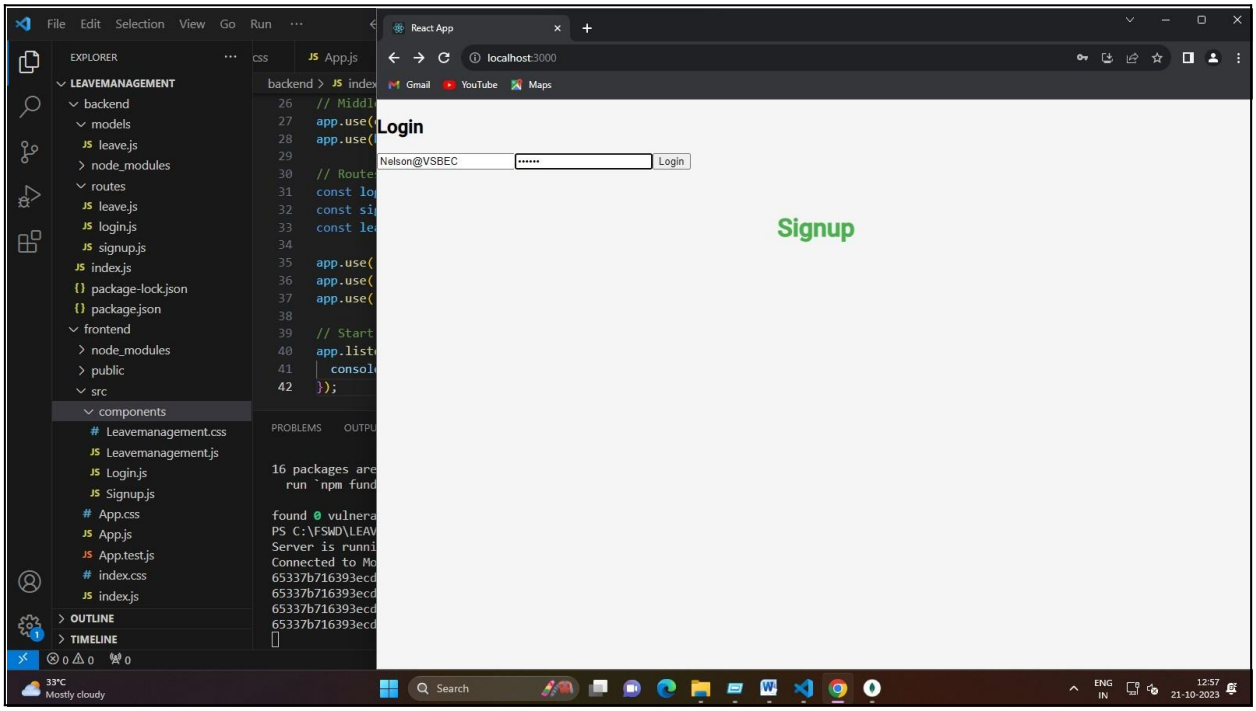
```
      return res.status(404).json({ message: 'User not found' });
    }
    res.status(200).json(user);
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: 'Server error' });
  }
});
router.put('/', async (req, res) => {
  try {
    const { id, leaveType, leaveDays } = req.body;
    const existingUser = await User.findOne({ id });
    const user = await User.findById(id);
    if (existingUser) {
      return res.status(409).json({ message: 'User found' });
    }
    if(leaveType=="medical"){
      const medical= (user.medical)-(leaveDays);
      if( medical>=0 && user.medical > medical ){
      updatedUser= await User.findByIdAndUpdate(
        id,
        { medical},
        { new: true }
      );
      if (!updatedUser) {
        return res.status(404).json({ message: 'User not found' });
      }
                                res.status(200).json({ message: 'Update successful', user:
updatedUser,casual:updatedUser.casual,medical:updatedUser.medical  });
      }
      else{
                                                                  res.json({ message: 'limit
exeeded',casual:updatedUser.casual,medical:updatedUser.medical});
      }
    }
    else if(leaveType=="casual"){
      const casual= (user.casual)-(leaveDays);
      if(casual>=0 && user.casual > casual ){
      updatedUser= await User.findByIdAndUpdate(
         id,
         { casual},
         { new: true }
       );
       if (!updatedUser) {
         return res.status(404).json({ message: 'User not found' });
       }
                                res.status(200).json({ message: 'Update successful', user:
updatedUser,casual:updatedUser.casual,medical:updatedUser.medical});
      }
      else{
                                                                  res.json({ message: 'limit
exeeded' ,casual:updatedUser.casual,medical:updatedUser.medical});
      }
    }
```

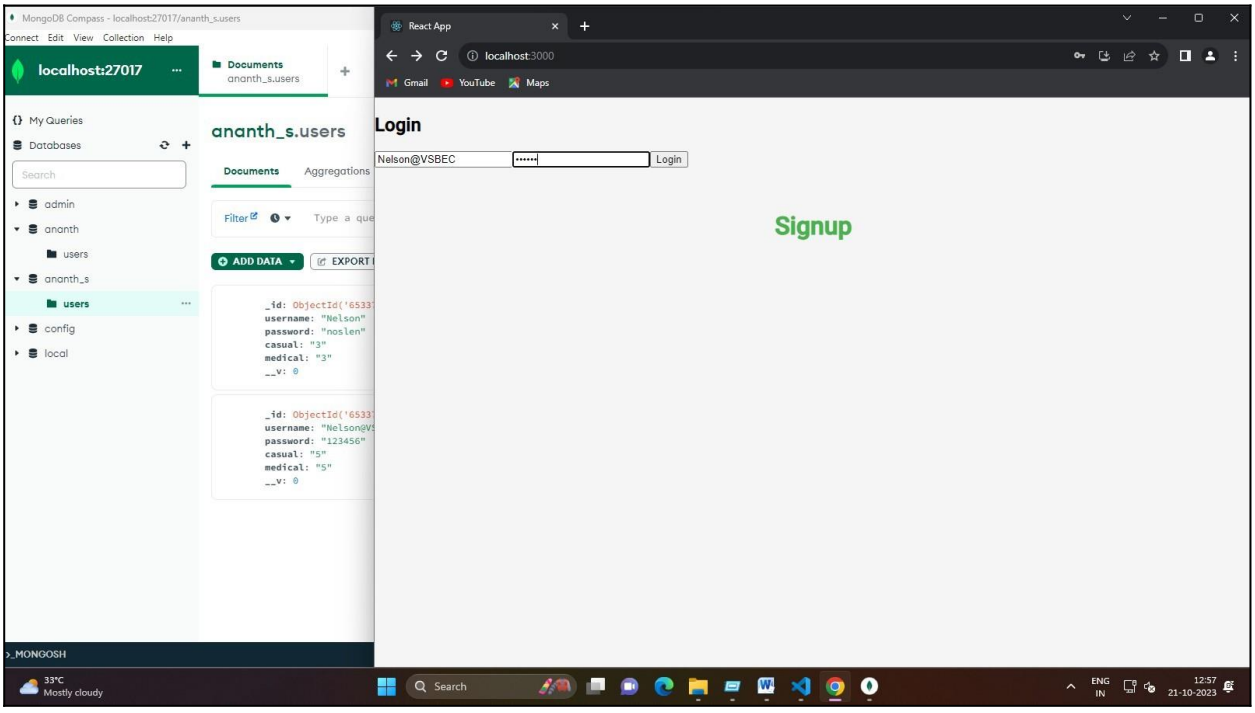
```
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: 'Server error' });
  }
});
module.exports = router;
```

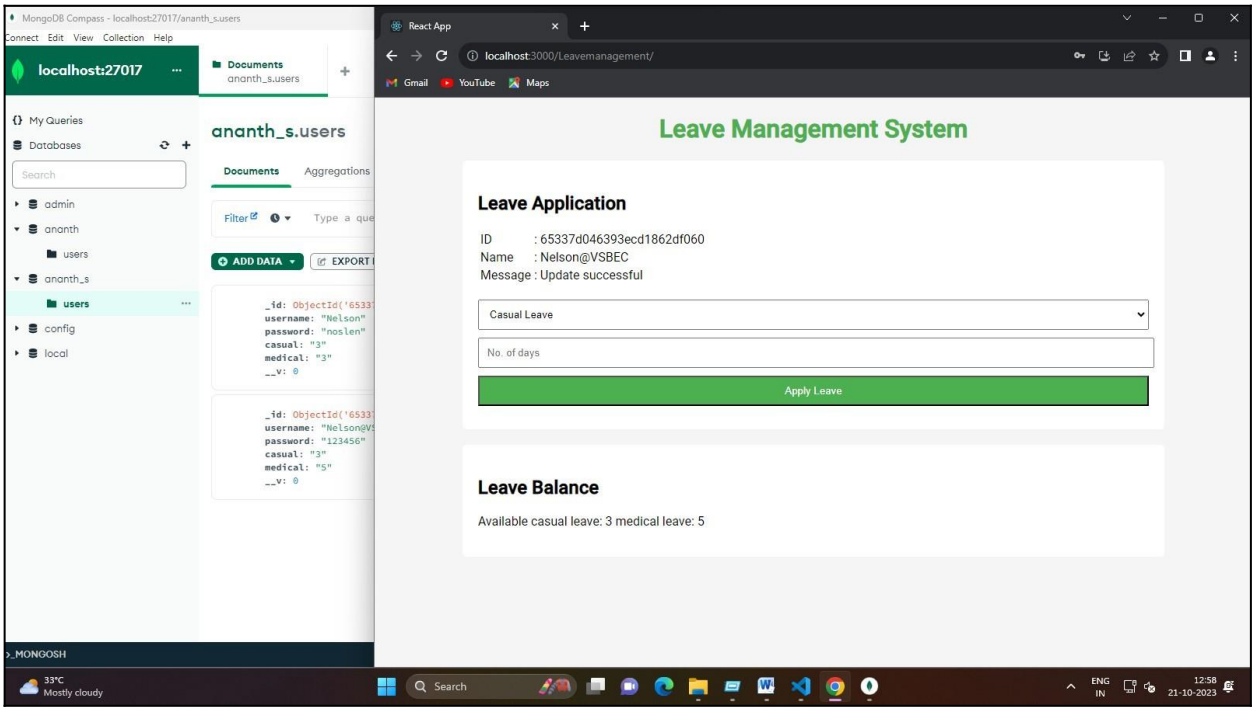
index.js

```
// LEAVEMANAGEMENT/backend/
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
const cors = require('cors');
const dotenv = require('dotenv');
dotenv.config();
const app = express();
const port = process.env.PORT || 3001;
// MongoDB connection
mongoose.connect("mongodb://0.0.0.0:27017/ananth_s", {
useNewUrlParser: true,
  useUnifiedTopology: true,
});
mongoose.connection.on('connected', () => {
  console.log('Connected to MongoDB');
});
mongoose.connection.on('error', (err) => {
  console.error('MongoDB connection error:', err);
});
// Middleware
app.use(cors());
app.use(bodyParser.json());
// Routes
const loginRouter = require('./routes/login');
const signupRouter = require('./routes/signup');
const leaveRouter = require('./routes/leave');
app.use('/login', loginRouter);
app.use('/signup', signupRouter);
app.use('/leave', leaveRouter);
// Start the server
app.listen(port, () => {
  console.log(`Server is running on port ${port}`);});
```

OUTPUT:


File Edit Selection View Go Run
EXPLORER
LEAVEMANAGEMENT
backend
models
leave.js
node_modules
routes
leave.js
login.js
signup.js
index.js
package-lock.json
package.json
frontend
node_modules
public
src
components
Leavemanagement.css
Leavemanagement.js
Login.js
Signup.js
App.css
App.js
App.test.js
index.css
index.js
OUTLINE
TIMELINE
React App
localhost:3000
Gmail
YouTube
Maps
Login
Nelson@VSBEC
Login
Signup
33°C
Mostly cloudy
Search
ENG
IN
12:57
21-10-2023


MongoDB Compass - localhost:27017/ananth_s.users
Connect Edit View Collection Help
localhost:27017
My Queries
Databases
Search
admin
ananth
users
ananth_s
users
config
local
Documents
ananth_s.users
ananth_s.users
Documents
Aggregations
Filter
ADD DATA
EXPORT
_id: ObjectId('6533
username: "Nelson"
password: "noslen"
casual: "3"
medical: "3"
__v: 0
_id: ObjectId('6533
username: "Nelson@V
password: "123456"
casual: "5"
medical: "5"
__v: 0
>_MONGOSH
React App
localhost:3000
Gmail
YouTube
Maps
Login
Nelson@VSBEC
Login
Signup
33°C
Mostly cloudy
Search
ENG
IN
12:57
21-10-2023



Result

Thus a user-friendly Leave Management System has been designed and implemented successfully.

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, InProgress or Completed.

Aim

To design and implement a user-friendly project management dashboard that allows users to add new tasks and update task statuses among Pending, InProgress, or Completed, thereby facilitating efficient task tracking and management.

Steps to Implement:

Step 1: Setup Frontend and Backend Projects

- Create a new folder named "ProjectDashboard".
- Inside "ProjectDashboard", create folders named "frontend" and "backend".

Step 2: Setup Frontend Project

- Open a terminal and navigate to "ProjectDashboard/frontend".
- Create a new React.js application inside the "frontend" folder:
  - npx create-react-app frontend
- Install required dependencies for React Router and CanvasJS:
  - cd frontend
  - npm i react-router-dom @canvasjs/react-charts

Step 3: Setup Backend Project

- Open a terminal and navigate to "ProjectDashboard/backend".
- Initialize a new Node.js application and install required dependencies:
  - npm init -y
  - npm install express mongoose body-parser cors dotenv

Step 4: Define Frontend Components

- Inside "frontend/src", create a folder named "components".
- Create the following components inside the "components" folder: Homepage.js, Newproject.js, and Project.js.
- Implement the logic for each component based on the provided code.

Step 5: Define Backend Routes and Models

- Inside "backend", create folders named "models" and "routes".
- Define the MongoDB schema for the user in "models/user.js".
- Implement backend routes for handling homepage, new project creation, and project update in "routes/homepage.js", "routes/newproject.js", and "routes/project.js" respectively.

Step 6: Configure Express.js Server

- Create an index.js file inside the "backend" folder.
- Configure Express.js server, connect to MongoDB, and define routes in the index.js file.
- Start the Express server:
  - node index.js

Step 7: Integrate Frontend and Backend

- Inside "frontend/src", configure API endpoints to make requests to the backend server (e.g., http://localhost:3001).
- Integrate frontend components with backend routes by making HTTP requests to the corresponding endpoints.

Step 8: Run the Application

- Open two separate terminals: one for frontend and one for backend.
- Start the React development server for frontend:
  - cd frontend
  - npm start
- Start the Express.js server for backend (in the backend folder):
  - cd backend

- node index.js

Step 9: Access the Application

- Open a web browser and navigate to http://localhost:3000 to access the project management dashboard.

Components Description:

FrontEnd

Homepage.js is a React functional component responsible for rendering the main page of the project management dashboard. It includes features for searching existing projects and creating new projects, allowing users to interact with the application through input fields, buttons, and dynamic navigation.

Newproject.js is a React functional component representing the form for creating a new project within the project management dashboard. It provides input fields for project details such as name, aim, algorithm, program, and output. Users can input information, submit the form, and receive feedback messages regarding the status of the project creation process.

Project.js is a React functional component responsible for displaying and managing project details within the project management dashboard. It allows users to update project information, visualize project data using a pie chart, and provides options to view or hide the chart report based on user interaction. The component communicates with the backend to handle data updates and chart rendering.

App.js is the main component of the project management dashboard frontend. It defines the application routes using React Router, allowing navigation between the homepage, new project creation, and project details display components. It serves as the entry point for rendering different sections of the dashboard based on user interactions and URL paths.

index.js is the entry file of the React application. It initializes the ReactDOM rendering process, wrapping the App component with BrowserRouter to enable routing functionality. The rendered content is injected into the HTML element with the ID 'root', effectively displaying the entire project management dashboard in the web browser.

Backend

user.js is a Mongoose schema file defining the structure of the 'User' model for the MongoDB database. It specifies the properties of a project, including project name, aim, algorithm, program, and output. The schema ensures data consistency and integrity when interacting with the MongoDB database in the backend of the project management system.

homepage.js is an Express route file handling POST requests related to project details retrieval. It queries the MongoDB database for a specific project name, validates the existence of the project, and returns relevant project information, including ID, aim, algorithm, program, and output. If the project doesn't exist, it responds with an error message or a success message with project details if found, ensuring proper communication between the frontend and backend of the project management system.

newproject.js is an Express route file handling POST requests for creating new projects. It checks if the project with the given name already exists in the MongoDB database. If not, it creates a new project entry with provided details including project name, aim, algorithm, program, and output. It responds with a success message if the project is added successfully or an error message in case of server issues, ensuring proper project creation functionality in the backend of the project management system.

project.js is an Express route file handling POST requests for updating existing projects. It searches for a project by its name in the MongoDB database. If found, it updates the

project details including aim, algorithm, program, and output. It responds with a success message upon successful update, or an error message if the project is not found or if there are server issues, ensuring proper project update functionality in the backend of the project management system.

index.js is the main entry point of the backend application for the PROJECTDASHBOARD. It configures an Express server, establishes a connection to MongoDB, sets up middleware for handling JSON data and enabling CORS, defines routes for homepage, new project creation, and project updates, and starts the server on a specified port. This file orchestrates the backend functionality, ensuring proper communication between the frontend and the database, allowing seamless management of projects within the dashboard.

Homepage.js

```
// PROJECTDASHBOARD\frontend\src\components
import React, { useState } from 'react';
import { useNavigate } from "react-router-dom";
function Homepage() {
  const [projectname, setProjectname] = useState('');
  const [message, setMessage] = useState('');
  const [ Objectid,setObjectid] = useState('');
  const navigate=useNavigate();
    function handlenewproject(){
    navigate("/newproject")
  }
  const handleProject = async () => {
    try {
      const response = await fetch('http://localhost:3001/homepage', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({ projectname}),
      });
        if (response.status === 200) {
            const data = await response.json();
        setObjectid(data.userId);
navigate("/project/",{state:
{id:data.userId,projectname:projectname,projectaim:data.projectaim,projectalgorithm:data.p
rojectalgorithm,projectprogram:data.projectprogram,projectoutput:data.projectoutput}});
      } else {
        const data = await response.json();
        setMessage(data.message);
      }
    } catch (error) {
      console.error(error);
    }
  };
  return (<center>
    <div><div>
    <h2>Search Project</h2>
    <input
    type="text"
    placeholder="Project"
```

```
    value={projectname}
    onChange={(e) => setProjectname(e.target.value)}
  />
  <button onClick={handleProject}>Submit</button>
  <p>{message}</p>
  <br></br>
</div>
<div  >
<h1 onClick={handlenewproject}>Create Project</h1>
  </div></div></center>
  );
};
export default Homepage;
```

Newproject.js

// PROJECTDASHBOARD\frontend\src\components

```
import React, { useState } from 'react';
function Newproject() {
  const [projectname, setProjectname] = useState('');
  const [projectaim, setProjectaim] = useState('');
  const [projectalgorithm, setProjectalgorithm] = useState('');
  const [projectprogram, setProjectprogram] = useState('');
  const [projectoutput, setProjectoutput] = useState('');
  const [message, setMessage] = useState('');
  const handleNewproject = async () => {
  try {
     const response = await fetch('http://localhost:3001/newproject', {
       method: 'POST',
       headers: {
         'Content-Type': 'application/json',
       },
                                                                                body:
JSON.stringify({  projectname,projectaim,projectalgorithm,projectprogram,projectoutput  }),
     });
     if (response.status === 201) {
       const data = await response.json();
       setMessage(data.message);
     } else {
       const data = await response.json();
       setMessage(data.message);
     }
   } catch (error) {
     console.error(error);
   }
  };
  return (
   <center> <div>
     <h1>New Project</h1>
     <h4>Project name</h4>
     <input
       type="text"
       placeholder="Name"
       value={projectname}
       onChange={(e) => setProjectname(e.target.value)}
```

```
      />
        <h4>Aim</h4>
        <textarea
        type="text"
        placeholder="Aim"
        value={projectaim}
        onChange={(e) => setProjectaim(e.target.value)} rows={5} cols={100}
      />
      <h4>Algorithm</h4>
      <textarea
      type="text"
      placeholder="Algorithm"
      value={projectalgorithm}
      onChange={(e) => setProjectalgorithm(e.target.value)} rows={10} cols={100}
    />
      <h4>Program</h4>
      <textarea      type="text"
      placeholder="Program"
      value={projectprogram}
      onChange={(e) => setProjectprogram(e.target.value)} rows={10} cols={100} />
         <h4>Output</h4>
      <textarea
      type="text"
      placeholder="Output"
      value={projectoutput}
      onChange={(e) => setProjectoutput(e.target.value)} rows={10} cols={100}
    />
   <h1></h1>
      <button onClick={handleNewproject}>Submit</button>
      <p>{message}</p>
    </div></center>
  );
}
export default Newproject;
```

Project.js

```
// PROJECTDASHBOARD\frontend\src\components
import React, { useState } from 'react';
import CanvasJSReact from '@canvasjs/react-charts';
import { useLocation } from 'react-router-dom';
var CanvasJSChart = CanvasJSReact.CanvasJSChart;
function countWordsInParagraph(paragraph) {
  const words = paragraph.trim().split(/\s+/);
  var a = words.length;
  if (paragraph === "") {
    a = 0;
  }
  return a;
}
function percentage(a, b) {
  return (a / b) * 100;
}
const Project = () => {
  const [message, setMessage] = useState('');
```

```
  const [projectname, setProjectname] = useState('');
  const [projectDetails, setProjectDetails] = useState('');
  const location = useLocation();
  const state = location.state;
  const projectName = state.projectname;
  const projectAim = state.projectaim;
  const projectAlgorithm = state.projectalgorithm;
  const projectProgram = state.projectprogram;
  const projectOutput = state.projectoutput;
  const projectaimcount = countWordsInParagraph(projectAim);
  const projectalgorithmcount = countWordsInParagraph(projectAlgorithm);
  const projectprogramcount = countWordsInParagraph(projectProgram);
  const projectoutputcount = countWordsInParagraph(projectOutput);
     const b = projectaimcount + projectalgorithmcount + projectprogramcount +
projectoutputcount;
  const options = {
    exportEnabled: true,
    animationEnabled: true,
    title: {
      text: projectName.toUpperCase()
    },
    data: [{
      type: "pie",
      startAngle: 75,
      toolTipContent: "<b>{label}</b>: {y}%",
      showInLegend: "true",
      legendText: "{label}",
      indexLabelFontSize: 16,
      indexLabel: "{label} - {y}%",
      dataPoints: [
        { y: percentage(projectaimcount, b), label: "Aim" },
        { y: percentage(projectalgorithmcount, b), label: "Algorithm" },
        { y: percentage(projectprogramcount, b), label: "Program" },
        { y: percentage(projectoutputcount, b), label: "Output" },
      ]
    }]
  };
  const handleInputChange = (fieldName, value) => {
    setProjectDetails({
      ...projectDetails,
      [fieldName]: value,
    });
  };
  const [chartVisible, setChartVisible] = useState(false);
  const [buttonVisible, setbuttonVisible] = useState(true);
  const handlechart = async () => {
    setChartVisible(true);
    setbuttonVisible(false);
  };
  const handleHide = async () => {
    setChartVisible(false);
    setbuttonVisible(true);
  };
  const handleNewproject = async () => {
```

```
    try {
      const updatedProjectDetails = {
        projectname: projectName,
        projectaim: projectDetails.projectAim || projectAim,
        projectalgorithm: projectDetails.projectAlgorithm || projectAlgorithm,
        projectprogram: projectDetails.projectProgram || projectProgram,
        projectoutput: projectDetails.projectOutput || projectOutput,
      };
      const response = await fetch('http://localhost:3001/project', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify(updatedProjectDetails),
      });
      if (response.status === 201) {
        const data = await response.json();
        setMessage(data.message);
      } else {
        const data = await response.json();
        setMessage(data.message);
      }
    } catch (error) {
      console.error(error);
    }
  };
  return (
    <center>
      <div>
        <h4>Project name</h4>
        <input
          type="text"
          placeholder="Name"
          value={projectName}
          onChange={(e) => setProjectname(e.target.value)}
        />
        <h4>Aim</h4>
        <textarea
          type="text"
          placeholder="Aim"
          value={projectDetails.projectAim || projectAim}
          onChange={(e) => handleInputChange('projectAim', e.target.value)}
          rows={5}
          cols={100}
        />
        <h4>Algorithm</h4>
        <textarea
          type="text"
          placeholder="Algorithm"
          value={projectDetails.projectAlgorithm || projectAlgorithm}
          onChange={(e) => handleInputChange('projectAlgorithm', e.target.value)}
          rows={10}
          cols={100}
        />
```

```
        <h4>Program</h4>
        <textarea
          type="text"
          placeholder="Program"
          value={projectDetails.projectProgram || projectProgram}
          onChange={(e) => handleInputChange('projectProgram', e.target.value)}
          rows={10}
          cols={100}
        />
        <h4>Output</h4>
        <textarea
          type="text"
          placeholder="Output"
          value={projectDetails.projectOutput || projectOutput}
          onChange={(e) => handleInputChange('projectOutput', e.target.value)}
          rows={10}
          cols={100}
        />
        <br />
        <button onClick={() => { handleNewproject(); }}>Update</button>
        <p>{message}</p>
        {chartVisible ? (
          <div>
            <CanvasJSChart options={options} />
            <button onClick={handleHide}>Hide Report</button>
          </div>
        ) : null}
        {buttonVisible ? (
          <div>
            <button onClick={handlechart}>View Report</button>
          </div>
        ) : null}
      </div>
    </center>
  );
};
export default Project;
```

App.js

```
// PROJECTDASHBOARD\frontend \src
import React from 'react';
import Project from './components/Project';
import Homepage from './components/Homepage';
import Newproject from './components/Newproject';
import { Route,Routes} from "react-router-dom";
function App() {
  return (
    <div className="App">
      <Routes>
      <Route path='/' element={<Homepage />} />
      <Route path='/newproject' element={<Newproject />} />
      <Route path='/project' element={<Project />} /> {/* Add this route for the Project */}

        </Routes>
```

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

index.js

```
// PROJECTDASHBOARD\frontend \src
import React from 'react';
import ReactDOM from 'react-dom';
import { BrowserRouter as Router } from 'react-router-dom';
import App from './App';
ReactDOM.render(
  <Router>
    <App />
  </Router>,
  document.getElementById('root')
);
```

user.js

```
// PROJECTDASHBOARD\backend\models
const mongoose = require('mongoose');
const userSchema = new mongoose.Schema({
  projectname: { type: String, required: true},
  projectaim: { type: String },
  projectalgorithm: { type: String },
  projectprogram: { type: String },
  projectoutput: { type: String }
});
module.exports = mongoose.model('User', userSchema);
```

PROJECTDASHBOARD\backend\routes

homepage.js

```
const express = require('express');
const router = express.Router();
const UserModel = require('../models/user');
router.post('/', async (req, res) => {
  const { projectname } = req.body;
  try {
    const user = await UserModel.findOne({ projectname });
    if (!user) {
      return res.status(401).json({ message: 'Project name is wrong' });
    }
    const userid=user.id;
    const projectaim=user.projectaim;
    const projectalgorithm=user.projectalgorithm;
    const projectprogram=user.projectprogram;
    const projectoutput=user.projectoutput;
                                        res.status(200).json({ message: 'Project
successful',userid:userid,projectname,projectaim,projectalgorithm,projectprogram,projectout
put});
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: 'Server error' });
  }
});
module.exports = router;
```

newproject.js
PROJECTDASHBOARD\backend\routes

```
const express = require('express');
const router = express.Router();
const User = require('../models/user');
router.post('/', async (req, res) => {
try {
        const { projectname,projectaim,projectalgorithm,projectprogram,projectoutput } =
req.body;
   const existingUser = await User.findOne({ projectname });
   if (existingUser) {
    return res.status(409).json({ message: 'Project already exists' });
   }
   const newUser = new User({
    projectname,projectaim,projectalgorithm,projectprogram,projectoutput,
   });
   await newUser.save();
   res.status(201).json({ message: 'Project added successful' });
  } catch (err) {
   console.error(err);
   res.status(500).json({ message: 'Server error' });
  }
});
module.exports = router;
```

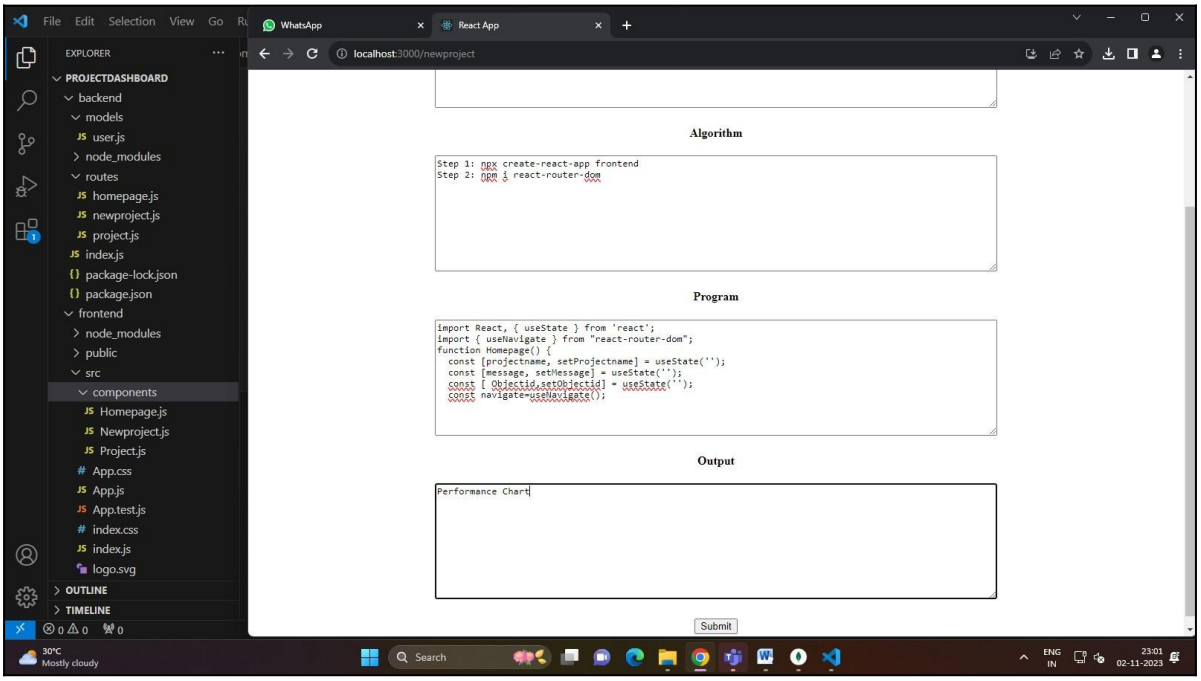
project.js

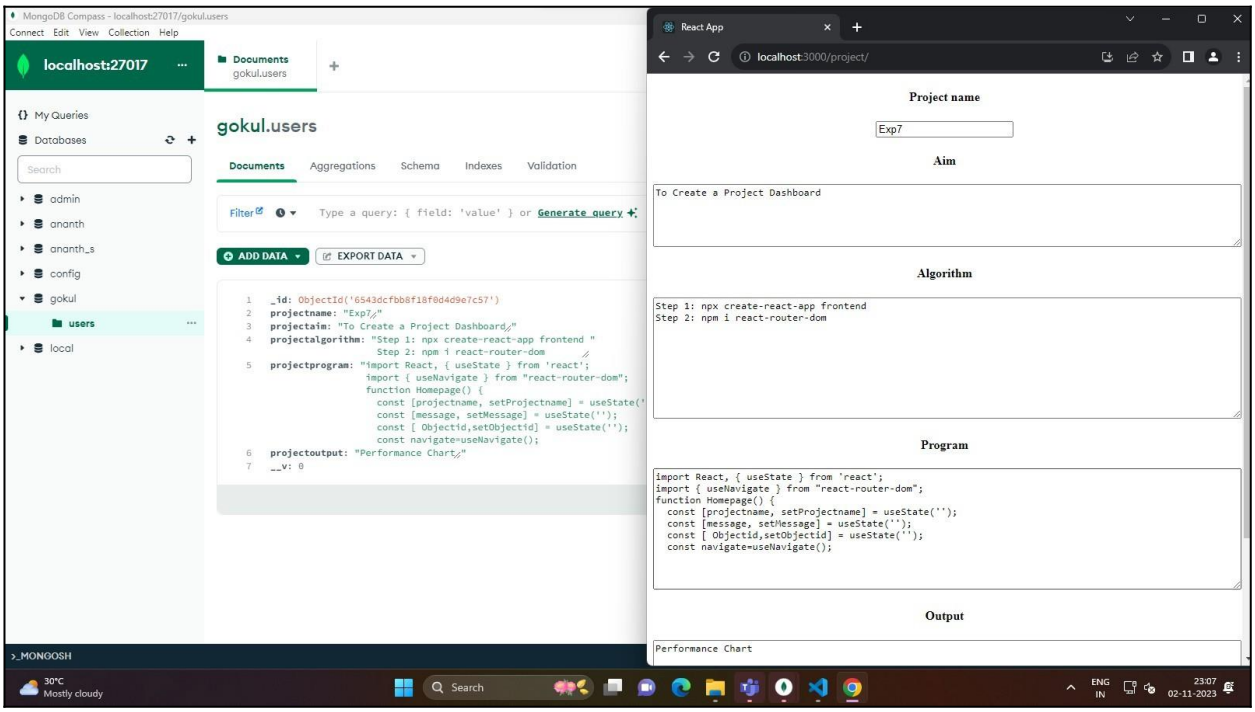
```
// PROJECTDASHBOARD/backend/routes
const express = require('express');
const router = express.Router();
const User = require('../models/user');
router.post('/', async (req, res) => {
try {
      const { projectname, projectaim, projectalgorithm, projectprogram, projectoutput } =
req.body;
   // Find the existing user by projectname
   const existingUser = await User.findOne({ projectname});
   if (!existingUser) {
    return res.status(404).json({ message: 'Project not found' });
   }
   existingUser.projectaim = projectaim;
   existingUser.projectalgorithm = projectalgorithm;
   existingUser.projectprogram = projectprogram;
   existingUser.projectoutput = projectoutput;
   await existingUser.save();
   res.status(200).json({ message: 'Project updated successfully' });
  } catch (err) {
   console.error(err);
   res.status(500).json({ message: 'Server error' });
  }
});
module.exports = router;
```

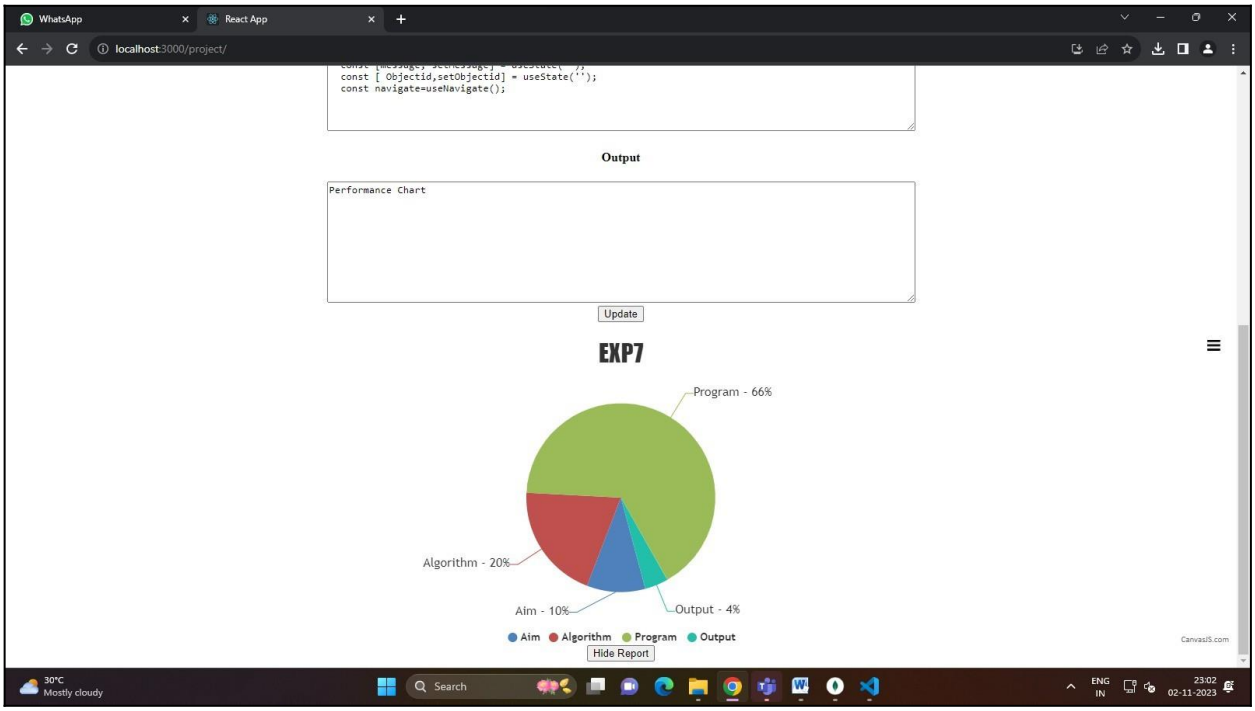
index.js

```
// PROJECTDASHBOARD/backend/
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const dotenv = require('dotenv');
const bodyParser = require('body-parser');
dotenv.config();
const app = express();
const port = process.env.PORT || 3001;
// MongoDB connection
mongoose.connect("mongodb://0.0.0.0:27017/gokul", {
useNewUrlParser: true,
  useUnifiedTopology: true,
});
mongoose.connection.on('connected', () => {
  console.log('Connected to MongoDB');
});
mongoose.connection.on('error', (err) => {
  console.error('MongoDB connection error:', err);
});
// Middleware
app.use(cors());
app.use(bodyParser.json());
// Routes
const homepageRouter = require('./routes/homepage');
const newprojectRouter = require('./routes/newproject');
const projectRouter = require('./routes/project');
// const leaveRouter = require('./routes/leave');
app.use('/homepage', homepageRouter);
app.use('/newproject', newprojectRouter);
app.use('/project', projectRouter);
// app.use('/leave', leaveRouter);
// Start the server
app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

OUTPUT:


Algorithm
Step 1: npx create-react-app frontend
Step 2: npm i react-router-dom
Program
import React, { useState } from 'react';
import { useNavigate } from "react-router-dom";
function Homepage() {
const [projectname, setProjectname] = useState('');
const [message, setMessage] = useState('');
const [Objectid,setObjectid] = useState('');
const navigate=useNavigate();
Output
Performance Chart
Submit


Project name
Exp7
Aim
To Create a Project Dashboard
Algorithm
Step 1: npx create-react-app frontend
Step 2: npm i react-router-dom
Program
Output
Performance Chart



Result

Thus the Project Management Dashboard has been developed for enabling users to seamlessly add tasks and update statuses, enhancing task tracking and management efficiency was successful.

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 create an online survey application with a diverse collection of questions, allowing users to answer random queries, enhancing their engagement and feedback participation.

Steps to Implement:

Step 1: Create a new folder named "SURVEY".

Step 2: Inside the "SURVEY" folder, create a frontend application:

- Open a new terminal.
- Navigate to the "SURVEY" folder (cd SURVEY).
- Run the following commands:
  - mkdir frontend
  - cd frontend
  - npx create-react-app frontend
  - npm i react-router-dom
  - npm install --save recharts

Step 3: Create the frontend components and styles:

- Inside the "SURVEY/frontend/src" folder:
  - Create a folder named "components".
  - Inside the "components" folder, create three files: "Login.js", "Signup.js", and "Survey.js".
  - Create a CSS file for styling named "Survey.css".
  - Copy and paste the respective code for each file.

Step 4: Set up the backend application:

- Inside the "SURVEY" folder, create a backend application:
  - Open a new terminal.
  - Run the following commands:
    - mkdir backend
    - cd backend
    - npm init -y
    - npm install express mongoose body-parser bcrypt cors dotenv
    - mkdir models
    - mkdir routes

Step 5: Create backend routes and models:

- Inside the "SURVEY/backend/models" folder, create a file named "survey.js". Copy and paste the code for the mongoose schema.
- Inside the "SURVEY/backend/routes" folder, create three files: "login.js", "signup.js", and "survey.js". Copy and paste the respective code for each file.

Step 6: Set up the backend server:

- Inside the "SURVEY/backend" folder, create a file named "index.js". Copy and paste the code for setting up the Express server.

Step 7: Run the application:

- Open two separate terminals.
- In the first terminal, navigate to "SURVEY/frontend" and run npm start to start the frontend application.
- In the second terminal, navigate to "SURVEY/backend" and run node index.js to start the backend server.

Step 8: Access the application:

- Open a web browser and go to "http://localhost:3000" to access the frontend application.

Components Description:

FrontEnd

Login.js component handles user authentication by allowing users to enter their username and password. It sends a request to the server for validation, and if successful, it redirects the user to the survey page. Users can also navigate to the signup page by clicking the "Signup" link.

Signup.js component enables users to create new accounts by providing a username, password, and registration number. Upon submission, it sends the user data to the server for registration. Users can also navigate back to the login page by clicking the "Login" link.

Survey.js is a React component that displays a student survey form, allowing users to provide feedback on instructors and class experiences. Users select staff members, answer survey questions, and submit responses, which are then visualized using a bar chart. Additionally, users can review previous survey responses.

Survey.css is a cascading style sheet (CSS) file defining the styles for the survey form interface. It specifies the layout, colors, and animations, enhancing the visual presentation of the survey components. The styles include responsive design, button formatting, and a fade-in animation for smooth user experience.

App.js is the main entry point of the React application. It defines the routes for different components (Login, Signup, and Survey) using the React Router library, allowing users to navigate between login, signup, and survey interfaces within the application.

index.js serves as the entry point for the React application, rendering the main App component within the BrowserRouter provided by React Router. It establishes the connection between the React app and the HTML root element, allowing the application components to be displayed in the specified DOM element.

Backend

survey.js defines a Mongoose schema for user data in a MongoDB database. It specifies the structure of user documents, including fields like username, password, regno, and ratings for different staff members (Manivannan, Nelson, Kosalairaman, MaheshKumar, Gobinath, Umamaheswari). The schema is used to create a Mongoose model called User, which can be manipulated to interact with the MongoDB database.

login.js is an Express.js route module responsible for handling user authentication during login. It receives a POST request with username and password parameters, queries the database for a matching user, and validates the provided password. If authentication succeeds, it sends a JSON response indicating a successful login along with the user's ID, name, and registration number. If authentication fails, it returns an error message and a 401 status code.

signup.js is an Express.js route module responsible for handling user registration. It receives a POST request with username, password, and regno parameters, checks if the user already exists in the database, and creates a new user entry if not. The module hashes the password, stores user information in the database, and returns a JSON response indicating a successful signup or an error message with a 500 status code in case of server issues.

survey.js is an Express.js route module responsible for handling survey-related requests. It includes endpoints for updating survey responses based on staff members, calculating

response sums, and retrieving user survey data by ID. The module communicates with the MongoDB database using Mongoose to perform these operations.

index.js in the SURVEY/backend/ directory sets up an Express.js server that listens for incoming requests on a specified port. It establishes a connection to a MongoDB database, uses middleware for handling JSON data and enabling CORS, and defines routes for login, signup, and survey-related operations, ensuring the backend functionality for the survey application.

FrontEnd

Login.js
//SURVEY\frontend\src\components

```
import React, { useState } from 'react';
import { useNavigate } from "react-router-dom";
function Login() {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [message, setMessage] = useState('');
  const [objectid, setObjectid] = useState('');
  const navigate = useNavigate();
  function handlesignup() {
    navigate("/signup")
  }
  const handleLogin = async () => {
    try {
      const response = await fetch('http://localhost:3001/login', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({ username, password }),
      });
      if (response.status === 200) {
        const data = await response.json();
        setObjectid(data.userId);
        console.log("User ID : ", data.userId);
                navigate("/Survey/", { state: { id: data.userId, name1: data.name, regno1: data.regno } });
      } else {
        const data = await response.json();
        setMessage(data.message);
      }
    } catch (error) {
      console.error(error);
    }
  };
  return (
    <div>
      <h2>Login</h2>
      <p>{objectid}</p>
      <input
        type="text"
        placeholder="Username"
```

```
        value={username}
        onChange={(e) => setUsername(e.target.value)}
      />
      <input
        type="password"
        placeholder="Password"
        value={password}
        onChange={(e) => setPassword(e.target.value)}
      />
      <button onClick={handleLogin}>Login</button>
      <p>{message}</p>

      <br>
      </br>
      <h1 onClick={handlesignup}>Signup</h1>
    </div>
  );
}
export default Login;
```

Signup.js

//SURVEY\frontend\src\components

```
import React, { useState } from 'react';
import { useNavigate } from "react-router-dom";
function Signup() {
  const [username, setUsername] = useState('');
  const [regno, setRegno] = useState('');
  const [password, setPassword] = useState('');
  const [message, setMessage] = useState('');
  const navigate = useNavigate();
  function handleLogin() {
    navigate("/")
  }
  const handleSignup = async () => {
    try {
      const response = await fetch('http://localhost:3001/signup', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({ username, password, regno }),
      });
      if (response.status === 201) {
        const data = await response.json();
        setMessage(data.message);
      } else {
        const data = await response.json();
        setMessage(data.message);
      }
    } catch (error) {
      console.error(error);
    }
  };
```

```
  return (
    <div>
      <h2>Signup</h2>
      <input
        type="text"
        placeholder="Username"
        value={username}
        onChange={(e) => setUsername(e.target.value)}
      />
      <input
        type="password"
        placeholder="Password"
        value={password}
        onChange={(e) => setPassword(e.target.value)}
      />
      <input
        type="text"
        placeholder="Reg No."
        value={regno}
        onChange={(e) => setRegno(e.target.value)}
      />
      <button onClick={handleSignup}>Signup</button>
      <p>{message}</p>
      <br>
      </br>
      <h1 onClick={handleLogin}>Login</h1>
    </div>
  );
}
export default Signup;
```

Survey.js

```
// SURVEY\frontend\src\components
import React, { useState } from 'react';
import { BarChart, Bar, CartesianGrid, XAxis, YAxis } from 'recharts';
import './Survey.css'
import { useLocation } from "react-router-dom";
const Survey = () => {
  const location = useLocation();
  const id1 = location.state.id;
     const staffNames = ["Manivannan", "Nelson", "Kosalairaman", "MaheshKumar",
"Gobinath", "Umamaheswari"];
  const questions = [
    {
      id: 'q1',
      question: '1. The class was very well organized',
    },
    {
      id: 'q2',
      question: '2. The instructor was very knowledgeable about the topic that was taught',
    },
    {
      id: 'q3',
      question: '3. The instructor was very good at communication',
```

```
  },
  {
    id: 'q4',
    question: '4. The instructor was motivating and enthusiastic',
  },
  {
    id: 'q5',
    question: "5. The instructor's methods helped in understanding the topic better",
  },
  {
    id: 'q6',
    question: '6. The grading system was not biased',
  },
  {
    id: 'q7',
    question: '7. The facilities provided were up to your expectations',
  },
  {
    id: 'q8',
    question: '8. The teaching environment in the class helped in better learning',
  },
  {
    id: 'q9',
    question: '9. The hygiene was very well maintained at the facility',
  },
  {
    id: 'q10',
    question: '10. Is the instructor biased between three classes',
  },
];
const choiceValues = {
  'Strongly disagree': 1,
  'Somewhat disagree': 2,
  'Neutral': 3,
  'Somewhat agree': 4,
  'Strongly agree': 5,
};
console.log("id :", id1);
console.log("name:", location.state.name1);
console.log("regm=n: ", location.state.regno1);
const [staffMember, setStaffMember] = useState('');
const [Manivannan, setManivannan] = useState('');
const [Nelson, setNelson] = useState('');
const [Kosalairaman, setKosalairaman] = useState('');
const [MaheshKumar, setMaheshKumar] = useState('');
const [Gobinath, setGobinath] = useState('');
const [Umamaheswari, setUmamaheswari] = useState('');
const [responses, setResponses] = useState({});
const [message, setMessage] = useState('Welcome!!!!');
const handleStaffChange = (e) => {
  setResponses({});
  setStaffMember(e.target.value);
};
const data = [
```

```
  { name: 'Manivannan', students: Manivannan },
  { name: 'Nelson', students: Nelson },
  { name: 'Kosalairaman', students: Kosalairaman },
  { name: 'MaheshKumar', students: MaheshKumar },
  { name: 'Gobinath', students: Gobinath },
  { name: 'Umamaheswari', students: Umamaheswari },
];
const review = async () => {
  try {
    const response = await fetch(`http://localhost:3001/survey/${id1}`);
    const data = await response.json();
    if (response.ok) {
      setManivannan(data.Manivannan);
      setNelson(data.Nelson);
      setKosalairaman(data.Kosalairaman);
      setMaheshKumar(data.MaheshKumar);
      setGobinath(data.Gobinath);
      setUmamaheswari(data.Umamaheswari);
    } else {
      console.error('Failed to fetch users:', data.message);
    }
  } catch (error) {
    console.error('Error during user fetch:', error.message);
  }
};
const handleSubmit1 = async () => {
  if (staffMember === "") {
    alert("Please Choose StaffName !!!!!");
  }
  else {
    try {
      const response = await fetch('http://localhost:3001/survey', {
        method: 'PUT',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({ id1, staffMember, responses }),
      });
      console.log("id = ", id1);
      if (response.status === 200) {
        const data = await response.json();
        setMessage(data.message);
      } else {
        const data = await response.json();
        setMessage(data.message);
      }
    } catch (error) {
      console.error(error);
    }
  }
};
const handleResponseChange = (questionId, choice) => {
  console.log("afjljjflfj");
  const value = choiceValues[choice];
```

```
  setResponses((prevResponses) => ({
    ...prevResponses,
    [questionId]: value,
  }));
};
const handleSubmit = (e) => {
  e.preventDefault();
  console.log('Data to be submitted:', {
  staffMember,
    responses,
  });
};
return (
  <div className="container">
    <h1>Student Survey Form</h1>
    <form onSubmit={handleSubmit}>
      <p>{message}</p>
      <p>Name :{location.state.name1}</p>
      <p>RegNo : {location.state.regno1}</p>
      <div className="student-info">
        <div className="form-group">
          <label htmlFor="staffMember">Select Staff Member:</label>
          <select
            name="staffMember"
            id="staffMember"
            value={staffMember}
            onChange={handleStaffChange}
          >
            <option value="">Select a staff member</option>
            {staffNames.map((name, index) => (
              <option key={index} value={name}>
                {name}
              </option>
            ))}
          </select>
        </div>
      </div>
      {questions.map((question) => (
        <div key={question.id}>
          <p>{question.question}</p>
          <div className="options">
            {Object.keys(choiceValues).map((choice) => (
              <div key={choice}>
                <label>
                  <input
                    type="radio"
                    name={question.id}
                    value={choice}
                    checked={responses[question.id] === choiceValues[choice]}
                    onChange={() => handleResponseChange(question.id, choice)}
                  />
                  {choice}
                </label>
              </div>
```

```
            ))}
          </div>
        </div>
      ))}
      <button type="submit" className="btn btn-primary"
        onClick={handleSubmit1}>
        Submit
      </button><br></br><br></br>
      <button
        type="button"
        className="btn btn-secondary"
        onClick={review}
      >
        Review
      </button>
    </form>
    <br></br>
    <BarChart width={700} height={600} data={data}>
      <Bar dataKey="students" fill="green" />
      <CartesianGrid stroke="#ccc" />
      <XAxis dataKey="name" />
      <YAxis domain={[0, 5]} />
    </BarChart>
  </div>
 );
};
export default Survey;
```

Survey.css

```
// SURVEY\frontend\src\components
.container {
  max-width: 700px;
  margin: 0 auto;
  padding: 20px;
  background-color: #f5f5f5;
  border-radius: 10px;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
  animation: fade-in 1s ease-in-out;
}
@keyframes fade-in {
  from {
    opacity: 0;
  }
  to {
    opacity: 1;
  }
}
.btn {
  background-color: #007bff;
  color: #fff;
  padding: 10px 20px;
  border: none;
  border-radius: 5px;
  cursor: pointer;
```

```
    transition: background-color 0.3s ease;
}
.btn:hover {
    background-color: #0056b3;
}
```

App.js

```
// SURVEY\frontend\src
import React from 'react';
import Login from './components/Login';
import Signup from './components/Signup';
import Survey from './components/Survey';
import { Route, Routes } from "react-router-dom";
function App() {
  return (
    <div className="App">
      <Routes>
        <Route path='/' element={<Login />} />
        <Route path='/signup' element={<Signup />} />
        <Route path='/Survey' element={<Survey />} />
      </Routes>
    </div>

  );
}
export default App;
```

index.js

```
// SURVEY\frontend\src
import React from 'react';
import ReactDOM from 'react-dom';
import { BrowserRouter as Router } from 'react-router-dom';
import App from './App';
ReactDOM.render(
  <Router>
    <App />
  </Router>,
  document.getElementById('root')
);
```

Backend

survey.js

```
//SURVEY\backend\models
const mongoose = require('mongoose');
const userSchema = new mongoose.Schema({
  username: {
    type: String,
    required: true,
  },
  password: {
    type: String,
    required: true,
  },
  regno: {
```

```
    type: String,
  },
  Manivannan: {
    type: String,
  },
  Nelson: {
    type: String,
  },
  Kosalairaman: {
    type: String,
  },
  MaheshKumar: {
    type: String,
  },
  Gobinath: {
    type: String,
  },
  Umamaheswari: {
    type: String,
  },
});
const User = mongoose.model('User', userSchema);
module.exports = User;
```

login.js

SURVEY\backend\routes

```
const express = require('express');
const router = express.Router();
//const bcrypt = require('bcrypt');
const User = require('../models/survey'); // Assuming you have a User model
router.post('/', async (req, res) => {
  const { username, password } = req.body;
  try {
    const user = await User.findOne({ username });
    if (!user) {
      return res.status(401).json({ message: 'Authentication failed' });
    }
    //const isPasswordValid = await bcrypt.compare(password, user.password);
    //if (!isPasswordValid) {
      const userId = user._id;
      const name = user.username;
      const regno = user.regno;
      if(password!=user.password){
      return res.status(401).json({ message: 'Authentication failed' });
    }
    // Handle successful login
    res.status(200).json({ message: 'Login successful',userId,name,regno});
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: 'Server error' });
  }
});
module.exports = router;
```

signup.js

```
// SURVEY\backend\routes
const express = require('express');
const router = express.Router();
//const bcrypt = require('bcrypt');
const User = require('../models/survey');
router.post('/', async (req, res) => {
  try {
    const { username, password,regno } = req.body;
    const existingUser = await User.findOne({ username });
    if (existingUser) {
      return res.status(409).json({ message: 'User already exists' });
    }
    const Manivannan="0";
    const Nelson="0";
    const Kosalairaman="0";
    const MaheshKumar="0";
    const Gobinath="0";
    const Umamaheswari="0";
    console.log(username,
      //password: hashedPassword,
        password,regno,Manivannan,Nelson,Kosalairaman,MaheshKumar,Gobinath,Umamahes
wari);
    //const hashedPassword = await bcrypt.hash(password, 10);
    const newUser = new User({
      username,
      //password: hashedPassword,
        password,regno,Manivannan,Nelson,Kosalairaman,MaheshKumar,Gobinath,Umamahes
wari,
    });
    await newUser.save();
    res.status(201).json({ message: 'Signup successful' });
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: 'Server error' });
  }
});
module.exports = router;
```

survey.js

```
// SURVEY/backend/routes
const express = require('express');
const router = express.Router();
const mongoose = require('mongoose');
const User = require('../models/survey');
router.put('/', async (req, res) => {
  try {
    const { id1, staffMember, responses } = req.body;
    console.log("id = ", id1);
    const existingUser = await User.findById(id1);
    const user = await User.findById(id1);
    if (!existingUser) {
      return res.status(409).json({ message: 'User found' });
    }
```

```
const calculateSumOfResponses = (responses) => {
  let sum = 0;
  for (const questionId in responses) {
    sum += responses[questionId];
  }
  sum = sum / 10;
  console.log("sum", sum);
  return (sum);
}
if (staffMember == "Manivannan") {
  console.log("if staf 1 =", user.Manivannan);
  const Manivannan = calculateSumOfResponses(responses);
  if (staffMember == "Manivannan" && user.Manivannan == "0") {
    console.log("Sum = ", Manivannan);
    updatedUser = await User.findByIdAndUpdate(
      id1,
      { Manivannan },
      { new: true }
    );
    if (!updatedUser) {
      return res.status(404).json({ message: 'User not found' });
    }
    res.status(200).json({ message: 'Update successful', user: updatedUser });
  }
  else {
    console.log("already filed");
    res.json({ message: 'limit exeeded' });
  }
}
else if (staffMember == "Nelson") {
  console.log("if Nelson =", user.Nelson);
  const Nelson = calculateSumOfResponses(responses);
  if (staffMember == "Nelson" && user.Nelson == "0") {
  console.log("Sum = ", Nelson);
    updatedUser = await User.findByIdAndUpdate(
      id1,
      { Nelson },
      { new: true }
    );
    if (!updatedUser) {
      return res.status(404).json({ message: 'User not found' });
    }
    res.status(200).json({ message: 'Update successful', user: updatedUser });
  }
  else {
    console.log("already filed");
    res.json({ message: 'limit exeeded' });
  }
}
else if (staffMember == "Kosalairaman") {
  console.log("if Kosalairaman =", user.Kosalairaman);
  const Kosalairaman = calculateSumOfResponses(responses);
  if (staffMember == "Kosalairaman" && user.Kosalairaman == "0") {
    console.log("Sum = ", Kosalairaman);
```

```
    updatedUser = await User.findByIdAndUpdate(
      id1,
      { Kosalairaman },
      { new: true }
    );
    if (!updatedUser) {
      return res.status(404).json({ message: 'User not found' });
    }
    res.status(200).json({ message: 'Update successful', user: updatedUser });
  }
  else {
    console.log("already filed");
    res.json({ message: 'limit exeeded' });
  }
}
else if (staffMember == "MaheshKumar") {
  console.log("MaheshKumar =", user.MaheshKumar);
  const MaheshKumar = calculateSumOfResponses(responses);
  if (staffMember == "MaheshKumar" && user.MaheshKumar == "0") {
    console.log("Sum = ", MaheshKumar);
    updatedUser = await User.findByIdAndUpdate(
      id1,
      { MaheshKumar },
      { new: true }
    );
    if (!updatedUser) {
      return res.status(404).json({ message: 'User not found' });
    }
    res.status(200).json({ message: 'Update successful', user: updatedUser });
  }
  else {
    console.log("already filed");
    res.json({ message: 'limit exeeded' });
  }
}
else if (staffMember == "Gobinath") {
  console.log("if staf 5 =", user.Gobinath);
  const Gobinath = calculateSumOfResponses(responses);
  if (staffMember == "Gobinath" && user.Gobinath == "0") {
    console.log("Sum = ", Gobinath);
    updatedUser = await User.findByIdAndUpdate(
      id1,
      { Gobinath },
      { new: true }
    );
    if (!updatedUser) {
      return res.status(404).json({ message: 'User not found' });
    }
    res.status(200).json({ message: 'Update successful', user: updatedUser });
  }
  else {
    console.log("already filed");
    res.json({ message: 'limit exeeded' });
  }
```

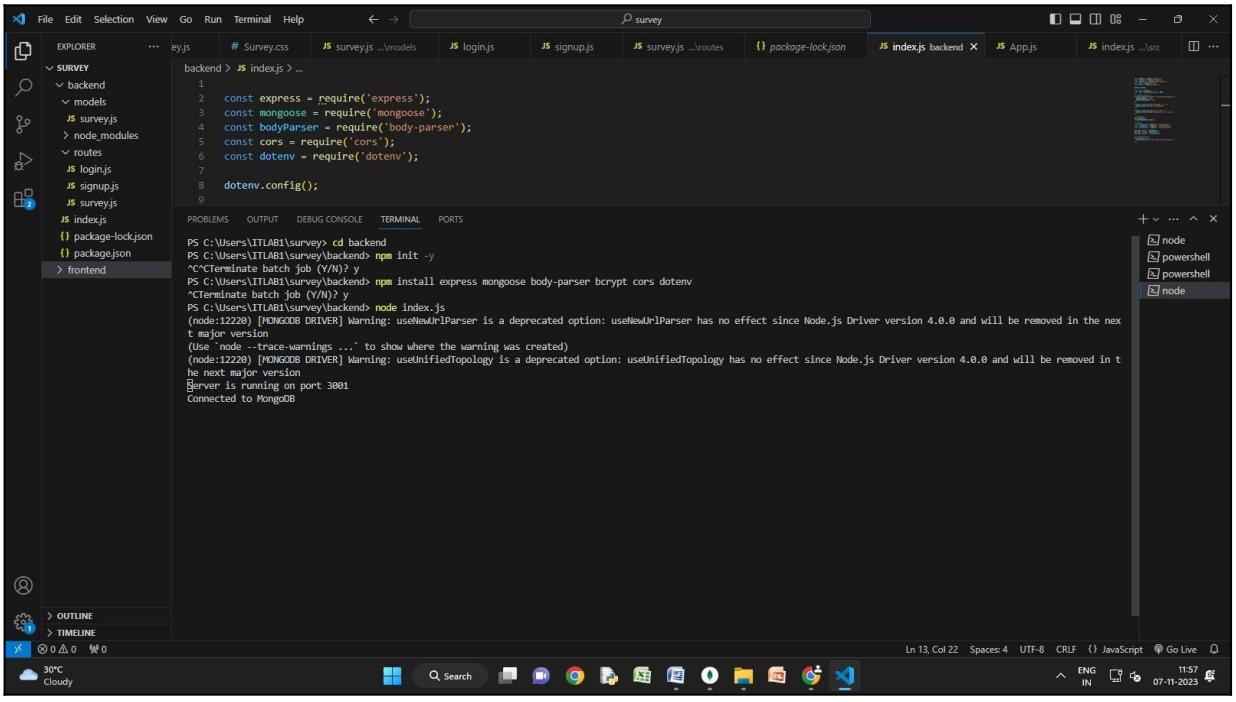
```
    }
    else if (staffMember == "Umamaheswari") {
      console.log("if staf 6 =", user.Umamaheswari);
      const Umamaheswari = calculateSumOfResponses(responses);
      if (staffMember == "Umamaheswari" && user.Umamaheswari == "0") {
        console.log("Sum = ", Umamaheswari);
        updatedUser = await User.findByIdAndUpdate(
          id1,
          { Umamaheswari },
          { new: true }
        );
        if (!updatedUser) {
          return res.status(404).json({ message: 'User not found' });
        }
        res.status(200).json({ message: 'Update successful', user: updatedUser });
      }
      else {
        console.log("already filed");
        res.json({ message: 'limit exeeded' });
      }
    }
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: 'Server error' });
  }
});
router.get('/:id', async (req, res) => {
  try {
    const user = await User.findById(req.params.id);
    console.log(req.params.id);
    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }
    res.status(200).json(user);
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: 'Server error' });
  }
});
module.exports = router;
```

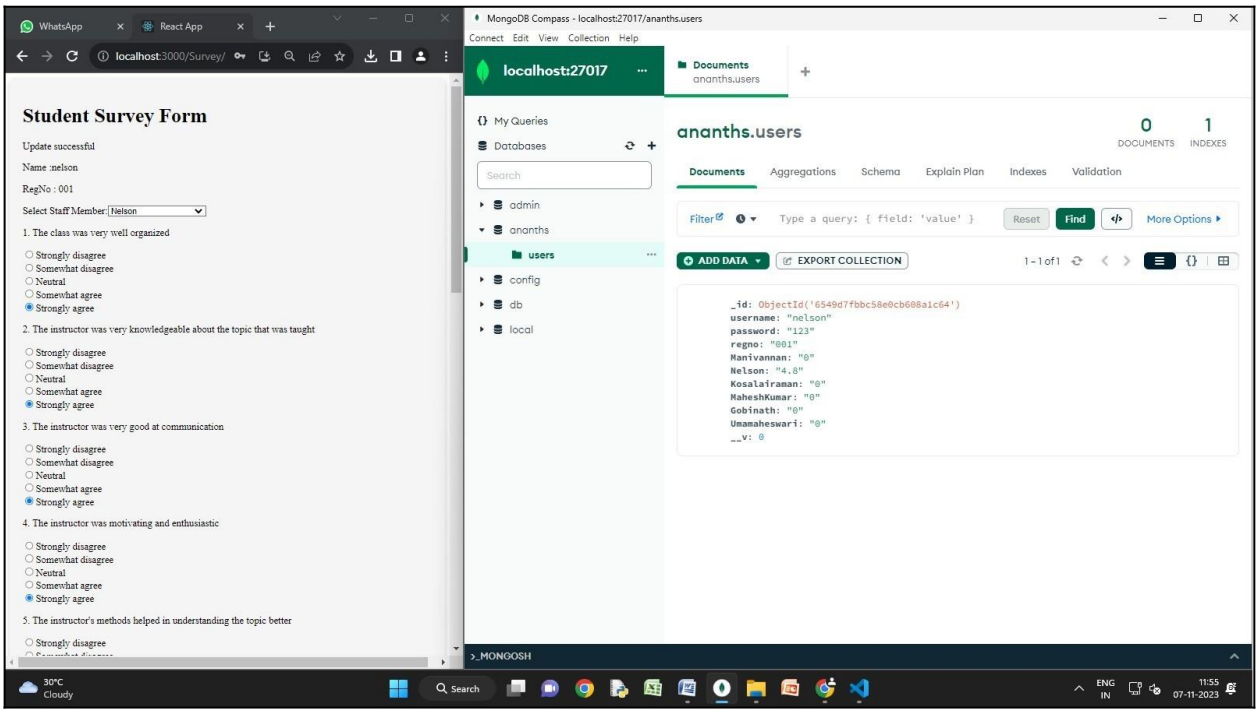
index.js

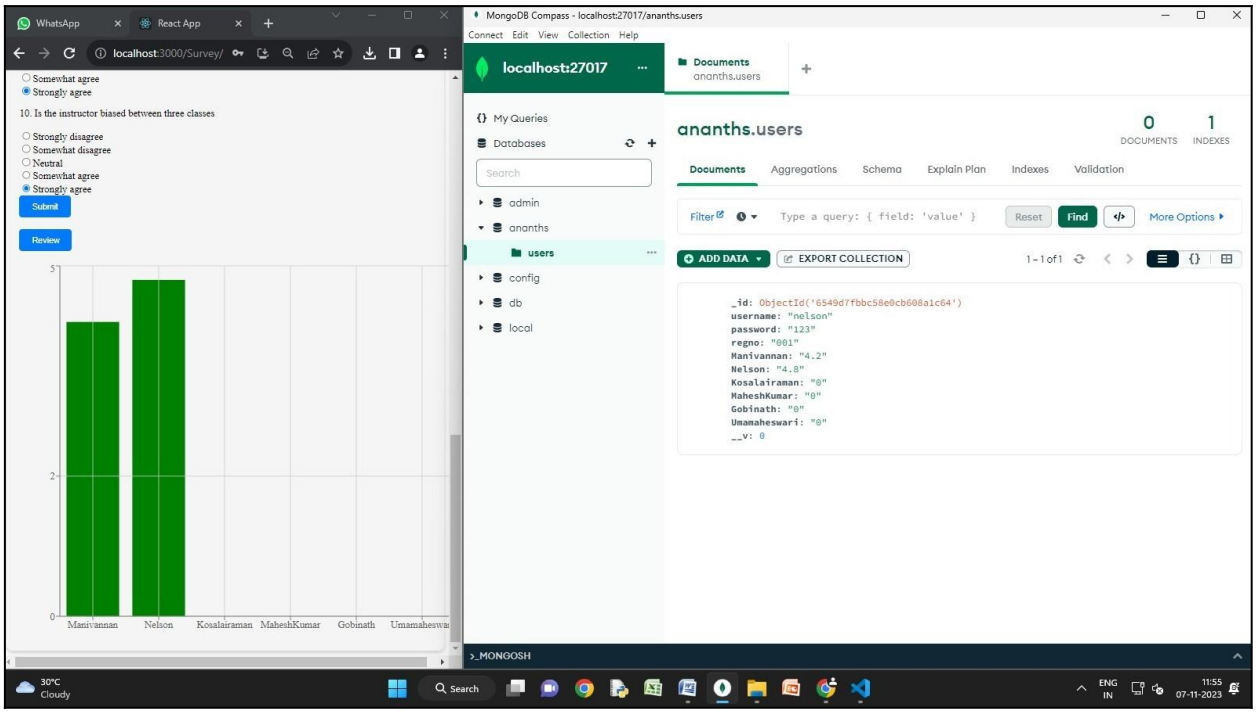
```
// SURVEY/backend/
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
const cors = require('cors');
const dotenv = require('dotenv');
dotenv.config();
const app = express();
const port = process.env.PORT || 3001;
// MongoDB connection
mongoose.connect("mongodb://0.0.0.0:27017/ananth", {
useNewUrlParser: true,
```

```
  useUnifiedTopology: true,
});
mongoose.connection.on('connected', () => {
  console.log('Connected to MongoDB');
});
mongoose.connection.on('error', (err) => {
  console.error('MongoDB connection error:', err);
});
// Middleware
app.use(cors());
app.use(bodyParser.json());
// Routes
const loginRouter = require('./routes/login');
const signupRouter = require('./routes/signup');
const surveyRouter = require('./routes/survey');
app.use('/login', loginRouter);
app.use('/signup', signupRouter);
app.use('/survey', surveyRouter);
// Start the server
app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

Output:







Result

Thus the interactive online survey platform has been developed successfully.