Exp. No. : Date :	PORTFOLIO WEBSITE

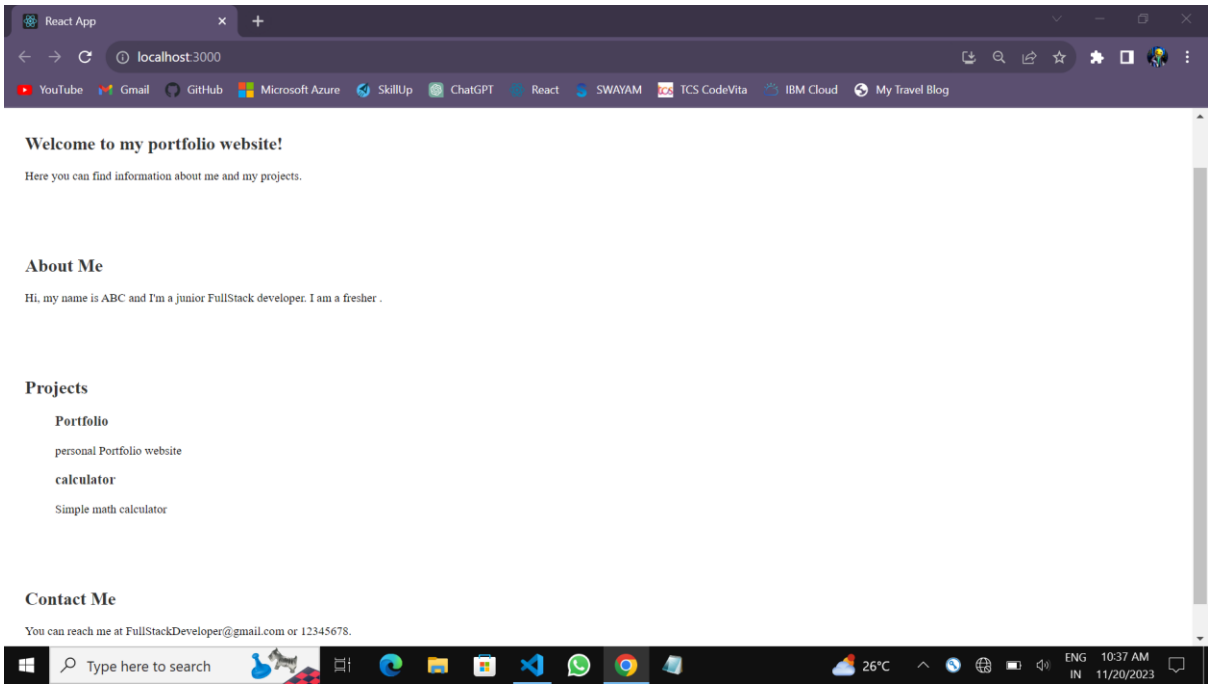
AIM:

Create a React portfolio website to showcase personal and professional information, projects, and contact details with a user-friendly navigation menu.

PROCEDURE:

1. Define a React component named App to structure the webpage.
2. Use <nav> to create a navigation menu with links to Home, About, Projects, and Contact sections.
3. Create sections with corresponding id attributes for Home, About, Projects, and Contact.
4. Populate each section with relevant content, including a welcoming message, personal details, project lists, and contact information.
5. Implement smooth scrolling functionality for navigation links to their respective sections.
6. Style the webpage for a visually appealing and cohesive presentation
   import React from 'react';

OUTPUT:


React App
localhost:3000
YouTube
Gmail
GitHub
Microsoft Azure
SkillUp
ChatGPT
React
SWAYAM
TCS CodeVita
IBM Cloud
My Travel Blog
Welcome to my portfolio website!
Here you can find information about me and my projects.
About Me
Hi, my name is ABC and I'm a junior FullStack developer. I am a fresher .
Projects
Portfolio
personal Portfolio website
calculator
Simple math calculator
Contact Me
You can reach me at FullStackDeveloper@gmail.com or 12345678.
Type here to search
26°C
ENG
IN
10:37 AM
11/20/2023

PROGRAM:

```
function App() {
  return (
    <div>
    <nav>
      <ul>
        <li><a href="#home">Home</a></li>
        <li><a href="#about">About</a></li>
        <li><a href="#projects">Projects</a></li>
        <li><a href="#contact">Contact</a></li>
      </ul>
    </nav>
    <section id="home">
      <h1>Welcome to my portfolio website!</h1>
      <p>Here you can find information about me and my projects.</p
</section>
    <section id="about">
      <h2>About Me</h2>
      <p>Hi, my name is [Your Name] and I'm a
        [Your Profession]. I have [Number of Years]
        years of experience in [Your Field].</p>
    </section>
    <section id="projects">
      <h2>Projects</h2>
      <ul>
        <li>

<h3>Project 1</h3>
          <p>Description of Project 1</p>
        </li>
        <li>
          <h3>Project 2</h3>
          <p>Description of Project 2</p>
        </li>
        <li>
          <h3>Project 3</h3>
```

```
            <p>Description of Project 3</p>
          </li>
        </ul>
      </section>
      <section id="contact">
        <h2>Contact Me</h2>
        <p>You can reach me at [Your Email Address]
          or [Your Phone Number].</p>
      </section>
    </div>);
  }
```

RESULT:

Exp. No. : Date :	TODO LIST

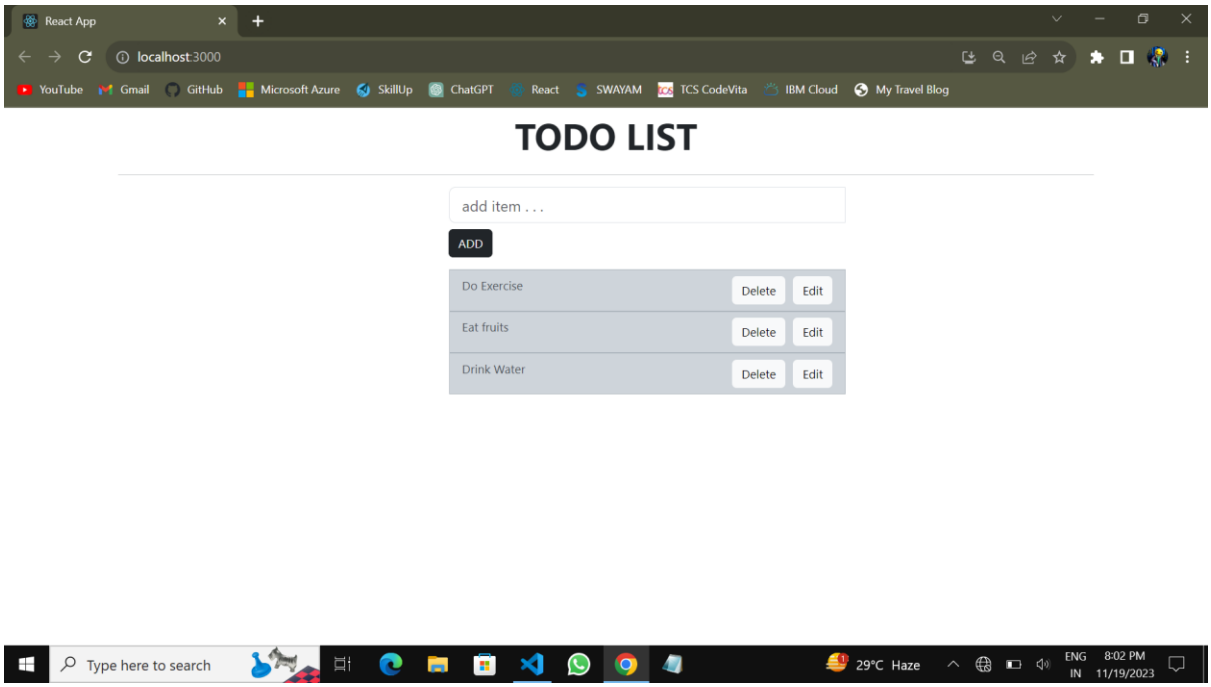
AIM:

Create a React-based todo list application with features for adding, editing, and deleting items.

PROCEDURE:

1. State Management:
2. Initialize state in the App component to track user input (userInput) and the list of items (list).
3. Add Item:
4. Implement the addItem method to add a new item to the list if the user input is not empty.
5. Delete Item:
6. Create the deleteItem method to remove an item from the list based on its unique identifier.
7. Edit Item:
8. Implement the editItem method to prompt the user for editing a todo item and update the list accordingly.
9. Render UI:
10. Render the user interface with Bootstrap components for input, adding, displaying, and managing todo list items.

OUTPUT:


React App
localhost:3000
YouTube
Gmail
GitHub
Microsoft Azure
SkillUp
ChatGPT
React
SWAYAM
TCS CodeVita
IBM Cloud
My Travel Blog
TODO LIST
add item . . .
ADD
Do Exercise
Delete
Edit
Eat fruits
Delete
Edit
Drink Water
Delete
Edit
Type here to search
29°C Haze
ENG
IN
8:02 PM
11/19/2023

PROGRAM:

```
// App.js File
import React, { Component } from "react";
import "bootstrap/dist/css/bootstrap.css";
import Container from "react-bootstrap/Container";
import Row from "react-bootstrap/Row";
import Col from "react-bootstrap/Col";
import Button from "react-bootstrap/Button";
import InputGroup from "react-bootstrap/InputGroup";
import FormControl from "react-bootstrap/FormControl";
import ListGroup from "react-bootstrap/ListGroup";
class App extends Component {
constructor(props) {
   super(props);

        // Setting up state
        this.state = {
                userInput: "",
                list: [],
        };
}

// Set a user input value
updateInput(value) {
   this.setState({
                userInput: value,
        });
}

// Add item if user input in not empty
addItem() {
   if (this.state.userInput !== "") {
        const userInput = {
           // Add a random id which is used to delete
           id: Math.random(),

                // Add a user value to list
```

```
                value: this.state.userInput,
                };

                // Update list
                const list = [...this.state.list];
                list.push(userInput);

                // reset state
                this.setState({
                  list,
                  userInput: "",
                });
        }
}
deleteItem(key) {
        const list = [...this.state.list];
        const updateList = list.filter((item)
      => item.id !== key);

        // Update list in state
        this.setState({
                list: updateList,
        });
}
editItem = (index) => {
const todos = [...this.state.list];
const editedTodo = prompt('Edit the todo:');
if (editedTodo !== null && editedTodo.trim() !== '') {
        let updatedTodos = [...todos]
        updatedTodos[index].value= editedTodo
        this.setState({
        list: updatedTodos,
});}}
render() {
        return (
        <Container>
                <Row
```

```
                style={{
                        display: "flex",
                        justifyContent: "center",
                        alignItems: "center",
                        fontSize: "3rem",
                        fontWeight: "bolder",
                        }}
                >
                TODO LIST
                </Row>
                <hr />
        <Row>
        <Col md={{ span: 5, offset: 4 }}>
        <InputGroup className="mb-3">
        <FormControl
              placeholder="add item . . . "
              size="lg"
                 value={this.state.userInput}
              onChange={(item) =>
                 this.updateInput(item.target.value)
                 }
                 aria-label="add something"
                 aria-describedby="basic-addon2"
        />
        <InputGroup>
        <Button
           variant="dark"
           className="mt-2"
           onClick={() => this.addItem()}
        >
        ADD
        </Button>
        </InputGroup>
        </InputGroup>
        </Col></Row><Row>
      <Col md={{ span: 5, offset: 4 }}>
        <ListGroup>
```

```
            {/* map over and print items */}
            {this.state.list.map((item, index) => {
            return (
            <div key = {index} >
            <ListGroup.Item
                    variant="dark"
                    action
                    style={{display:"flex",
                    justifyContent:'space-between'
                    }}>
                    {item.value}
                    <span>
                    <Button style={{marginRight:"10px"}}
                variant = "light"
                    onClick={() => this.deleteItem(item.id)}>
                Delete
                    </Button>
                <Button variant = "light"
                    onClick={() => this.editItem(index)}>
                    Edit
                    </Button>
                    </span>
                    </ListGroup.Item>
                    </div>
    );})}
        </ListGroup>
            </Col>
            </Row>
            </Container>);
    }
```

RESULT:

Exp. No. : Date :	**BLOG APPLICATION**

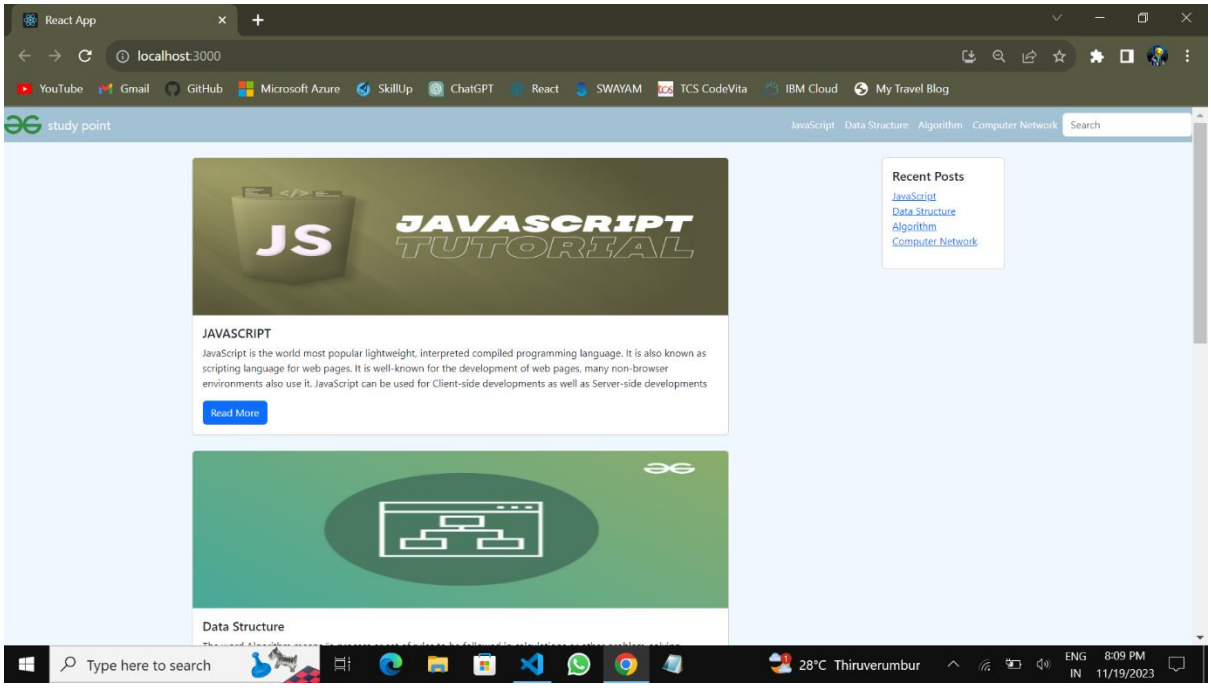
Aim:

Create a React-based blog website featuring posts on JavaScript, Data Structure, Algorithm, and Computer Network, with a responsive design and modular components**.**

Procedure:

1. App Component:
2. Render the main container in App.js with a navigation bar (Navbar) and a list of posts (Posts).
3. Posts Component:
4. Utilize Posts.js to display individual post cards using Bootstrap components and organize them in rows and columns.
5. BlogNav Component:
6. Implement BlogNav.js for the navigation bar, including a logo, site name, and links for different post categories.
7. Post Components:
8. Create individual post components (Post1.js, Post2.js, Post3.js, Post4.js) with images, titles, and content related to specific topics.
9. Styling:
10. Enhance the visual appeal using Bootstrap for responsive design and layout management.
11. Run Application:
12. Execute the React application in a development environment, ensuring dependencies are installed.
13. Navigation and Search:
14. Enable navigation links to scroll to corresponding sections and implement a search form in the navigation bar.
15. Testing:
16. Test the website to ensure proper functionality, responsiveness, and a cohesive display of posts and navigation elements.

OUTPUT:


React App
localhost:3000
YouTube
Gmail
GitHub
Microsoft Azure
SkillUp
ChatGPT
React
SWAYAM
TCS CodeVita
IBM Cloud
My Travel Blog
study point
JavaScript
Data Structure
Algorithm
Computer Network
Search
JS
JAVASCRIPT
TUTORIAL
Recent Posts
JavaScript
Data Structure
Algorithm
Computer Network
JAVASCRIPT
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
Read More
Data Structure
Type here to search
28°C Thiruverumbur
ENG
IN
8:09 PM
11/19/2023

Program:

```
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>);
};
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>
);
};
// 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='https://media.geeksforgeeks.org/gfg-gg-logo.svg'
height='30'
alt="
loading='lazy'
/>
<Navbar.Brand href="#home" style={{color:"white",
marginLeft:"10px"}}>study point</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>
)
}
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>
);
};
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>)}
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>
)
}
```

Result:

Exp. No. : Date :	FOOD DELIVERY APPLICATION

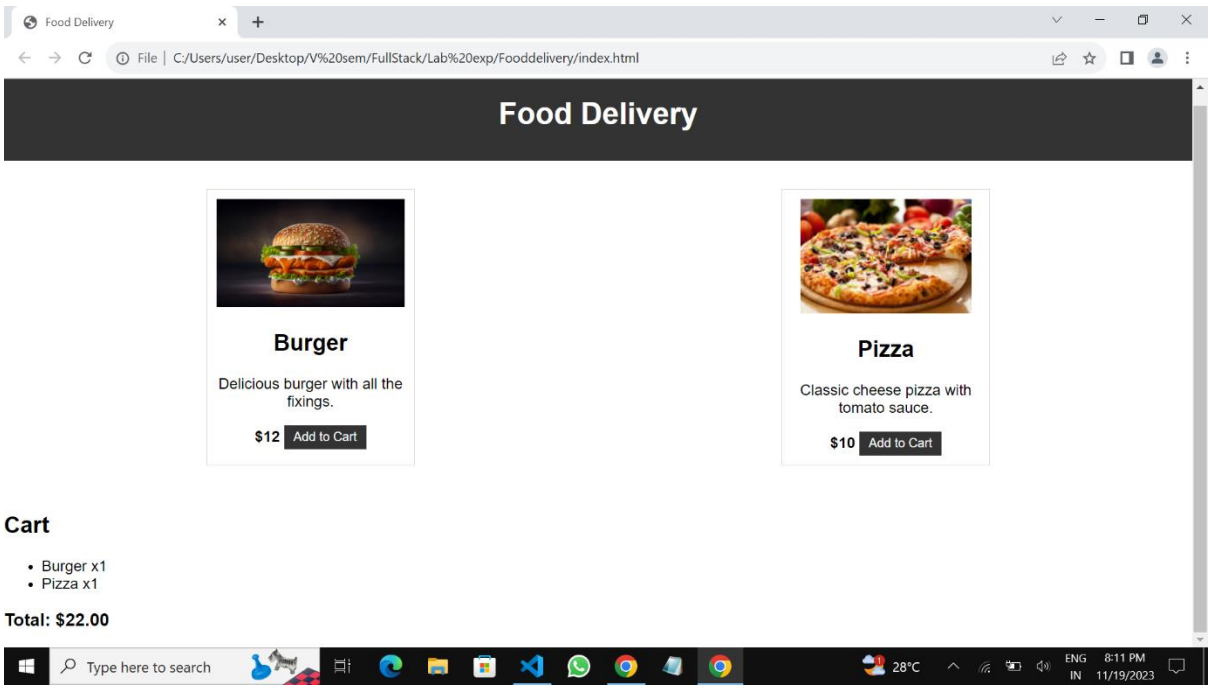
AIM:

Implement a simple food delivery webpage allowing users to add items to the cart, dynamically updating the cart items and total cost.

PROCEDURE:

1. HTML Structure:Design a basic HTML structure with a header, menu displaying food items, and a cart section.
2. CSS Styling:Apply CSS styles to enhance the visual presentation, ensuring a clean and responsive layout.
3. JavaScript Logic:Write JavaScript logic in script.js to handle the cart functionality.
4. Event Listeners:Add event listeners to "Add to Cart" buttons, capturing the selected item's name and price.
5. Update Cart:Update the cart object by incrementing the quantity if the item is already in the cart, or adding it if not.
6. Calculate Total:Keep track of the total cost by updating the total variable with the price of the added item.
7. Display Cart:Dynamically update the cart section in the HTML, displaying the added items and the updated total cost.
8. Responsive Layout:Ensure a responsive design that accommodates various screen sizes, providing a user-friendly experience.

OUTPUT:


Food Delivery
File | C:/Users/user/Desktop/V%20sem/FullStack/Lab%20exp/Fooddelivery/index.html
Food Delivery
Burger
Delicious burger with all the fixings.
$12
Add to Cart
Pizza
Classic cheese pizza with tomato sauce.
$10
Add to Cart
Cart
Burger x1
Pizza x1
Total: $22.00
Type here to search
28°C
ENG
IN
8:11 PM
11/19/2023

PROGRAM:

#index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="styles.css">
  <title>Food Delivery</title>
</head>
<body>
  <header>
    <h1>Food Delivery</h1>
  </header>
  <div class="menu">
    <div class="item">
      <img src="burger.jpg" alt="Burger">
      <h2>Burger</h2>
      <p>Delicious burger with all the fixings.</p>
      <span class="price">$8.99</span>
      <button class="add-to-cart">Add to Cart</button></div>
    <div class="item">
      <img src="pizza.jpg" alt="Pizza">
      <h2>Pizza</h2>
      <p>Classic cheese pizza with tomato sauce.</p>
      <span class="price">$10.99</span>
      <button class="add-to-cart">Add to Cart</button> </div></div>
  <div class="cart">
    <h2>Cart</h2>
    <ul id="cart-items"></ul>
    <h3>Total: $<span id="cart-total">0.00</span></h3>
  </div>
  <script src="script.js"></script>
</body>
</html>
```

#style.css

```
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
  }
  header {
    background-color: #333;
    color: white;
    text-align: center;
  padding: 10px;
  }
  .menu {
    display: flex;
    justify-content: space-around;
    flex-wrap: wrap;
    padding: 20px;
  }
  .item {
    border: 1px solid #ddd;
    padding: 10px;
    margin: 10px;
    text-align: center;
    width: 200px;}
  .item img {
    max-width: 100%;}
  .price {
    font-weight: bold;}
  .add-to-cart {
    background-color: #333;
    color: white;
    border: none;
    padding: 5px 10px;
    cursor: pointer;
  }
  .cart {
    border: 1px solid #ddd;
    padding: 10px;
```

```
    position: fixed;
    top: 350px;
    right: 40px;
    width: 250px; }
#script.js
const addToCartButtons = document.querySelectorAll('.add-to-cart');
const cartItemsList = document.getElementById('cart-items');
const cartTotal = document.getElementById('cart-total');
let total = 0;
const cart = {};
addToCartButtons.forEach(button => {
  button.addEventListener('click', () => {
    const item = button.parentElement;
    const itemName = item.querySelector('h2').textContent;
    const itemPrice = parseFloat(item.querySelector('.price')
.textContent.slice(1));
      if (cart[itemName]) {
      cart[itemName] += 1;
    } else {
      cart[itemName] = 1; }
    total += itemPrice;
    updateCart();
  });});
function updateCart() {
  cartItemsList.innerHTML = '';
  for (const item in cart) {
    const li = document.createElement('li');
    li.textContent = `${item} x${cart[item]}`;
    cartItemsList.appendChild(li);
  }
  cartTotal.textContent = total.toFixed(2);
}
```

RESULT:

Exp. No. : Date :	CLASSIFIED WEB APPLICATION TO BUY AND SELL OLD PRODUCTS

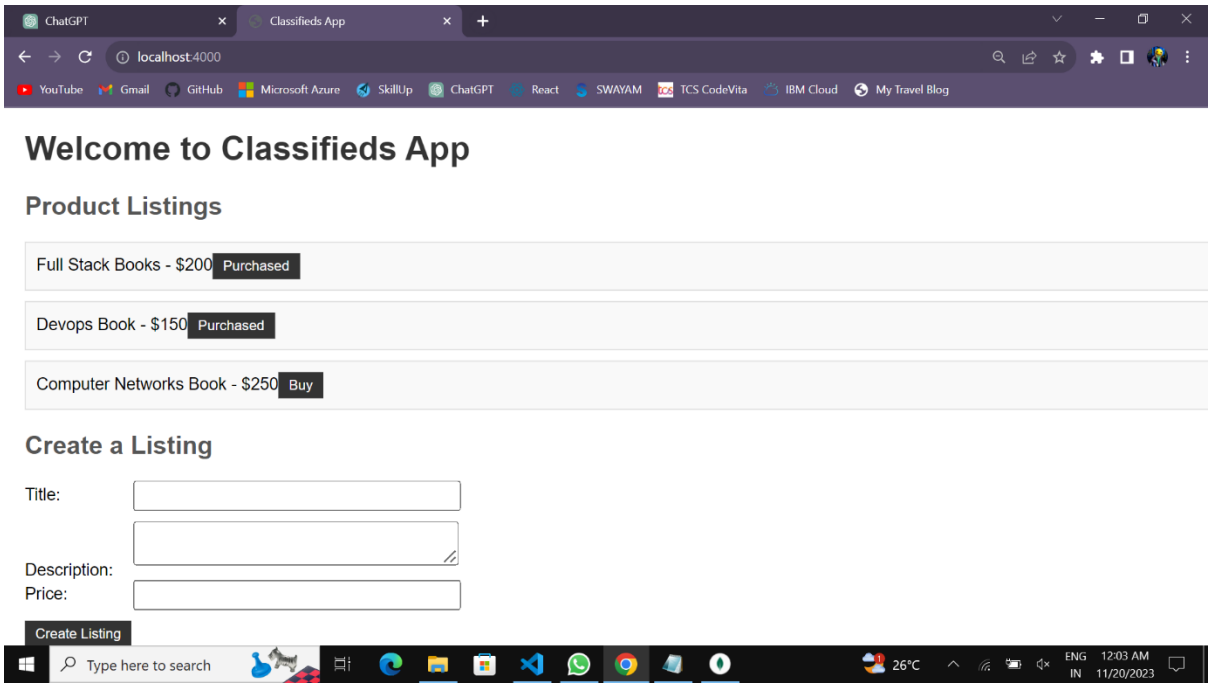
AIM:

Create a Classifieds App enabling users to view product listings, purchase items, and create new listings with MongoDB for data storage.Create a Classifieds App enabling users to view product listings, purchase items, and create new listings with MongoDB for data storage**.**

PROCEDURE:

1. Set up an Express server with MongoDB integration, handling product and user routes.
2. Create MongoDB models for products and users (Product and User models).
3. Implement logic in `app.js` to fetch and display product listings, allowing users to purchase items or create new listings.
4. Enable users to create new product listings via a form in `index.html` and handle the creation process in `app.js`.
5. Implement a function in `app.js` to mark products as "Purchased" and update the UI accordingly.
6. Implement user registration functionality in `userRoutes.js` to allow users to register and store user data in MongoDB.
7. Serve static files, including CSS styles, and style the UI for a visually appealing Classifieds App.

OUTPUT:


ChatGPT
Classifieds App
localhost:4000
YouTube
Gmail
GitHub
Microsoft Azure
SkillUp
ChatGPT
React
SWAYAM
TCS CodeVita
IBM Cloud
My Travel Blog
Welcome to Classifieds App
Product Listings
Full Stack Books - $200 Purchased
Devops Book - $150 Purchased
Computer Networks Book - $250 Buy
Create a Listing
Title:
Description:
Price:
Create Listing
Type here to search
26°C
ENG
IN
12:03 AM
11/20/2023

PROGRAM:

#app.js

```
document.addEventListener('DOMContentLoaded', () => {
  const productList = document.getElementById('product-list')
  // Function to fetch and display product listings
  const fetchProducts = () => {
    fetch('/api/products')
      .then(response => response.json())
      .then(products => {
        productList.innerHTML = ''; // Clear existing list
        products.forEach(product => {
          const li = document.createElement('li');
          li.textContent = `${product.title} - $${product.price}`;
          // Add a "Buy" button for each product
          const actionButton = document.createElement('button');
          if (product.status === 'purchased') {
            actionButton.textContent = 'Purchased';
            actionButton.disabled = true;
          } else {
            actionButton.textContent = 'Buy';
            actionButton.addEventListener('click', () =>
 buyProduct(product._id, actionButton));
          li.appendChild(actionButton);
          productList.appendChild(li);
        });
      })
      .catch(error => console.error('Error fetching products:',
error));
  };
  // Call the fetchProducts function initially and after
creating a listing
  fetchProducts();

  // Create a new product listing
  const createListingButton = document.getElementById
('create-listing-button');
```

```
  createListingButton.addEventListener('click', () => {
    const title = document.getElementById('title').value;
    const description = document.getElementById('description')
.value;
    const price = document.getElementById('price').value;
    const newData = {
      title: title,
      description: description,
      price: price
    };
 fetch('/api/products', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(newData)
    })
      .then(response => response.json())
      .then(product => {
        console.log('Product created:', product);
        // Clear input fields
        document.getElementById('title').value = '';
        document.getElementById('description').value = '';
        document.getElementById('price').value = '';
      // Fetch and display products after creating a listing
        fetchProducts();
      })
      .catch(error => console.error
('Error creating product:', error));
  });
// Buy a product
  const buyProduct = (productId, actionButton) => {
    fetch(`/api/products/${productId}/buy`, {
      method: 'PUT',
      headers: {
        'Content-Type': 'application/json'
      }
```

```
    })
    .then(response => response.json())
    .then(updatedProduct => {
      console.log('Product purchased:', updatedProduct);
       // Update the button text to "Purchased" and disable it
      actionButton.textContent = 'Purchased';
      actionButton.disabled = true;
       // Optionally, you can add a
 CSS class to style the
"Purchased" button differently
      actionButton.classList.add('purchased-button');
    })
      .catch(error => console.error
('Error purchasing product:',

error));
  };
 // Initial fetch and display of products
  fetchProducts();
});
#index.html
!DOCTYPE html>
<html>
<head>
  <title>Classifieds App</title>
  <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
  <h1>Welcome to Classifieds App</h1>
 <h2>Product Listings</h2>
  <ul id="product-list"></ul>
 <h2>Create a Listing</h2>
  <form id="create-listing-form">
    <label for="title">Title:</label>
    <input type="text" id="title" required><br>
    <label for="description">Description:</label>
    <textarea id="description" required></textarea><br>
```

```
    <label for="price">Price:</label>
    <input type="number" id="price" required><br>
    <button type="button" id="create-listing-button">
Create Listing</button>
  </form>
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script src="app.js"></script>
</body>
</html>
```

Style.css

```
body {
  font-family: Arial, sans-serif;
  margin: 20px;
}
h1 {
  color: #333;
}
h2 {
  color: #555;
  margin-top: 20px;
}
ul {
  list-style-type: none;
  padding: 0;
}
li {
  margin: 10px 0;
  padding: 10px;
  border: 1px solid #ddd;
  background-color: #f9f9f9;
}
label {
  display: inline-block;
  width: 100px;
}
input, textarea {
```

```
  width: 300px;
  padding: 5px;
  margin-bottom: 10px;
}
button {
  background-color: #333;
  color: #fff;
  padding: 5px 10px;
  border: none;
  cursor: pointer;
}
/* Style for the "Purchased" button */
.purchased-button {
  background-color: #ccc;
  color: #888;
  cursor: not-allowed;
}
```

Productuser.js

```
const express = require('express');
const router = express.Router();
const Product = require('../models/Product');
// Create a new product listing
router.post('/', async (req, res) => {
  try {
    const newProduct = await Product.create(req.body);
    res.json(newProduct);
  } catch (err) {
    res.status(400).json({ message: err.message });
  }
});
// Get all product listings
router.get('/', async (req, res) => {
  try {
    const products = await Product.find();
    res.json(products);
  } catch (err) {
    res.status(500).json({ message: err.message });
```

```
  }
});
// Update product status to mark as purchased
router.put('/:productId/buy', async (req, res) => {
  try {
    const productId = req.params.productId;
    // Find the product by ID and update its status or relevant fields
    const updatedProduct = await Product.findByIdAndUpdate(
      productId,
      { $set: { status: 'purchased' } }, // Update
 the status or relevant fields here
      { new: true } // Return the updated product
    );
    if (!updatedProduct) {
      return res.status(404).json({ message: 'Product not found' });
    }
    res.json(updatedProduct);
  } catch (error) {
    console.error('Error buying product:', error);
    res.status(500).json({ message: 'An error occurred
while buying the product' });
  }
});
module.exports = router;
#user route.js
const express = require('express');
const router = express.Router();
const User = require('../models/User');
// User registration
router.post('/register', async (req, res) => {
  try {
    const newUser = await User.create(req.body);
    res.json(newUser);
  } catch (err) {
    res.status(400).json({ message: err.message });
  }
});
```

```
module.exports = router;
#SERVER .JS
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
const cors = require('cors');
const path = require('path'); // Import the 'path' module
const app = express();
// Middleware
app.use(cors());
app.use(bodyParser.json());
// Connect to MongoDB
mongoose.connect('mongodb://127.0.0.1:27017/classifieds', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});
mongoose.connection.on('error', console.error.bind(console,
'MongoDB connection error:'));
app.use(express.static(path.join(__dirname, 'public')));
// Routes
const userRoutes = require('./routes/userRoutes');
const productRoutes = require('./routes/productRoutes');
app.use('/api/users', userRoutes);
app.use('/api/products', productRoutes);
const PORT = process.env.PORT || 4000;
app.listen(4000, () => {
  console.log(`Server is running on port ${PORT}`);
```

RESULT:

Exp. No. : Date :	DEVELOP A LEAVE MANAGEMENT SYSTEM

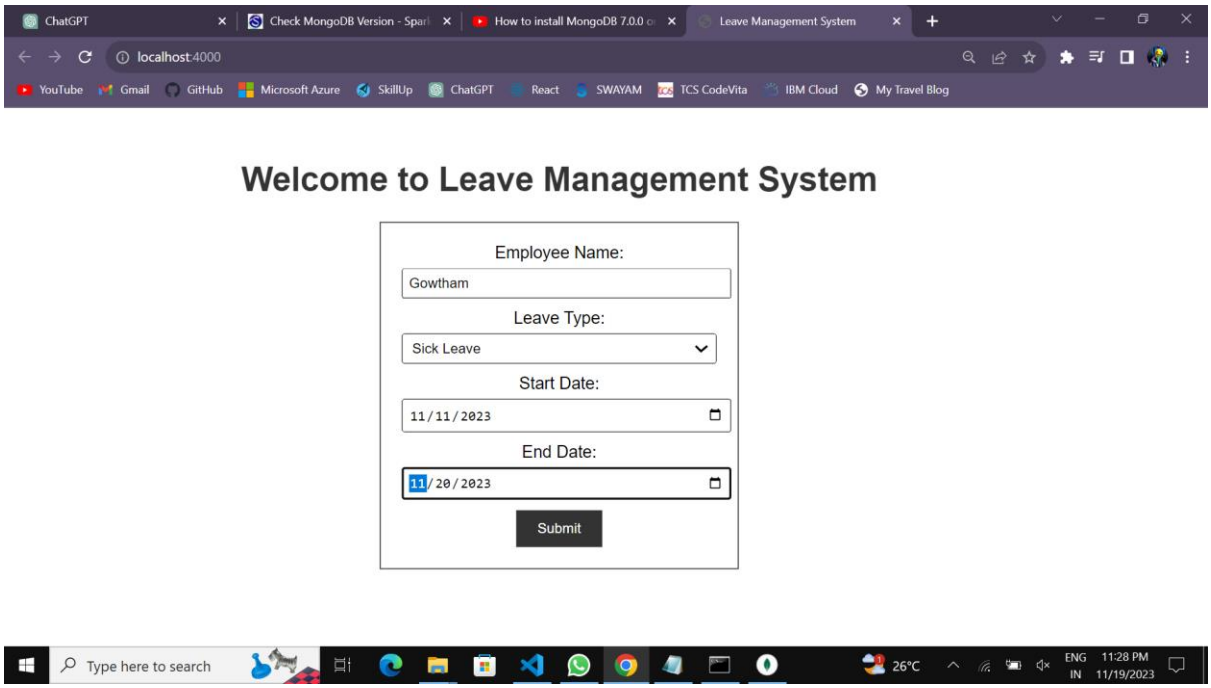
AIM:

Build a Leave Management System allowing employees to submit leave requests and view existing requests, with data stored in MongoDB**.**

PROCEDURE:

1. HTML Form:Create an HTML form (index.html) to input employee details and leave information.
2. CSS Styling:Style the form using CSS (style.css) for a visually appealing layout.
3. Express Server Setup:Set up an Express server (server.js) with middleware for static files and body parsing.
4. MongoDB Connection:Connect to MongoDB, define a schema for leave requests, and create a model (LeaveRequest) using Mongoose.
5. Retrieve Leave Requests:Implement a route to retrieve leave requests from MongoDB and render them on the home page.
6. Leave Request Submission:Create a route to handle form submissions, create a new LeaveRequest instance, and save it to the MongoDB database.
7. Server Initialization:Initialize the server, listening on the specified port (4000), and display a message once the server is running.
8. Submit Action:Enable the form to submit leave requests to the server, triggering the creation and storage of new leave requests in MongoDB**.**

OUTPUT:


ChatGPT
Check MongoDB Version - Spark
How to install MongoDB 7.0.0 on
Leave Management System
localhost:4000
YouTube
Gmail
GitHub
Microsoft Azure
SkillUp
ChatGPT
React
SWAYAM
TCS CodeVita
IBM Cloud
My Travel Blog
Welcome to Leave Management System
Employee Name:
Gowtham
Leave Type:
Sick Leave
Start Date:
11/11/2023
End Date:
11/20/2023
Submit
Type here to search
26°C
ENG
IN
11:28 PM
11/19/2023

PROGRAM:

index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Leave Management System</title>
 <link rel="stylesheet" href="/style.css">
</head>
<body>
 <h1>Welcome to Leave Management System</h1>
 <form action="http://localhost:4000/apply" method="POST">
 <label for="employeeName">Employee Name:</label>
 <input type="text" id="employeeName" name="employeeName" required> <br>
<label for="leaveType">Leave Type:</label>
 <select id="leaveType" name="leaveType" required>
 <option value="vacation">Vacation</option>
 <option value="sick">Sick Leave</option>
 </select><br>
<label for="startDate">Start Date:</label>
 <input type="date" id="startDate" name="startDate" required><br>
<label for="endDate">End Date:</label>
 <input type="date" id="endDate" name="endDate" required> <br>
<button type="submit">Submit</button>
 </form></body>
</html>
```

styles.css:

```
body {
 font-family: Arial, sans-serif;
 text-align: center;
margin: 50px;}
h1 {
 color: #333;}
form {
 margin: 20px auto;
```

```
 width: 300px;
border: 1px solid #333;
 padding: 20px;}
label {
 display: block;
margin-bottom: 5px;}
input[type="text"],
input[type="date"],
select {
 width: 100%;
padding: 5px;
margin-bottom: 10px;
}
button {
 background-color: #333;
color: #fff;
border: none;
padding: 10px 20px;
 cursor: pointer;
}
```

server.js:

```
const express = require('express');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');
//const ejs = require('ejs');
const app = express();
//app.set('view engine', 'ejs');
app.use(express.static('public'));
app.use(bodyParser.urlencoded({ extended:false }));
// Connect to MongoDB (replace with your MongoDB connection string)
mongoose.connect('mongodb://127.0.0.1:27017/leave_management_system', {
 useNewUrlParser: true,
useUnifiedTopology: true,
});
mongoose.connection.on('open', () => {
 // Start your server or perform database operations here
 });
```

```
const leaveRequestSchema = new mongoose.Schema({
 employeeName: String,
 leaveType: String,
 startDate: Date,
 endDate: Date,
});
const LeaveRequest = mongoose.model('LeaveRequest',
leaveRequestSchema);app.get('/', (req, res) => {
 // Retrieve leave requests from MongoDB
LeaveRequest.find({}, (err, leaveRequests) => {
 if (err) {
 console.error(err);
 res.status(500).send('Internal Server Error');
 } else {
 res.render('index', { leaveRequests });
 }
 });
});
app.post('/apply', (req, res) => {
 // Create and save a new leave request
const newLeaveRequest = new LeaveRequest({
 employeeName:req.body.employeeName,
 leaveType:req.body.leaveType,
 startDate:req.body.startDate,
 endDate:req.body.endDate
 });
newLeaveRequest.save();
res.end("Submitted");
 });
const port = process.env.PORT || 4000; // Use the provided port or default to 3000
app.listen(port, () => {
 console.log(`Server is running on http://localhost:${port}`);
});
```

RESULT:

Exp. No. : Date :	**DEVELOP A SIMPLE DASHBOARD FOR PROJECT MANAGEMENT SYSTEM**

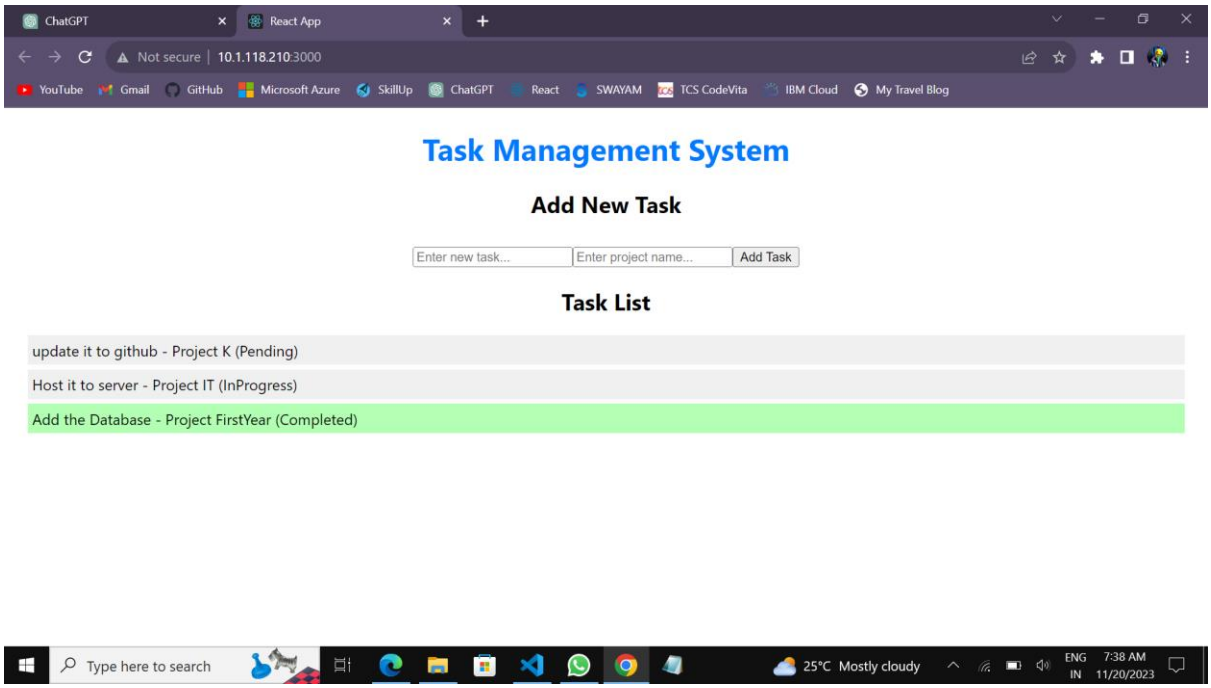
AIM:

Create a task management system where users can add and update the status of tasks through a React front end communicating with an Express back end**.**

PROCEDURE:

1. Express Server Setup:Set up an Express server (server.js) with CORS middleware and JSON body parsing.
2. TaskForm Component:Create a React functional component (TaskForm.js) with a form to add new tasks.
3. TaskList Component:Develop a React component (TaskList.js) to display the list of tasks and allow updating their status.
4. App Component State:Use React useState hook in the App.js component to manage the state of tasks.
5. Add Task Functionality:Implement a function in App.js (addTask) to add new tasks to the state.
6. Update Status Functionality:Create a function (updateStatus) in App.js to update the status of tasks.
7. Task Form Submission:Connect the TaskForm component to the addTask function to handle form submissions.
8. Task Status Update:Connect the TaskList component to the updateStatus function to update task statuses dynamically.

OUTPUT:


ChatGPT
React App
Not secure
10.1.118.210:3000
YouTube
Gmail
GitHub
Microsoft Azure
SkillUp
ChatGPT
React
SWAYAM
TCS CodeVita
IBM Cloud
My Travel Blog
Task Management System
Add New Task
Enter new task...
Enter project name...
Add Task
Task List
update it to github - Project K (Pending)
Host it to server - Project IT (InProgress)
Add the Database - Project FirstYear (Completed)
Type here to search
25°C Mostly cloudy
ENG
IN
7:38 AM
11/20/2023

PROGRAM:

server.js:

```
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const app = express();
app.use(cors());
app.use(bodyParser.json());
app.listen(5000, () => {
 console.log('Server is running on port 5000');
});
```

TaskForm.js:

```
// client/src/components/TaskForm.js
import React, { useState } from 'react';
const TaskForm = ({ addTask }) => {
 const [taskName, setTaskName] = useState('');
 const handleSubmit = (e) => {
 e.preventDefault();
 if (taskName.trim() !== '') {
 addTask(taskName);
 setTaskName(''); // Clear the input field } };
 return (
 <div className="task-form">
 <h2>Add New Task</h2>
 <form onSubmit={handleSubmit}>
 <input type="text"placeholder="Task Name"value={taskName}
onChange={(e) => setTaskName(e.target.value)}/>
 <button type="submit">Add Task</button>
 </form>
 </div> );
};
```

TaskList.js:

```
import React from 'react';
const TaskList = ({ tasks, updateStatus }) => {
 return (
 <div className="task-list">
 <h2>Task List</h2><ul>
```

```
{tasks.map((task, index) => (
<li key={index}>
<span>{task.name}</span><div>
<span>Status: {task.status}</span>
<button onClick={() => updateStatus(index)}>
Update Status
</button></div> </li>
))}
</ul></div>
);
 };
```

App.js:

```
//client/src/App.js
import React, { useState } from 'react';
import TaskForm from './components/TaskForm';
import TaskList from './components/TaskList';
function App() {
 const [tasks, setTasks] = useState([]);
 const addTask = (taskName) => {
 setTasks([...tasks, { name: taskName, status: 'Pending' }]);};
 const updateStatus = (index) => {
 const updatedTasks = [...tasks];
 updatedTasks[index].status =
 updatedTasks[index].status === 'Pending' ? 'InProgress' : 'Completed';
 setTasks(updatedTasks);
 };
 return (
 <div className="App">
 <TaskForm addTask={addTask} />
 <TaskList tasks={tasks} updateStatus={updateStatus} />
 </div>
 );}
```

RESULT:

Exp. No. : Date :	DEVELOP AN ONLINE SURVEY APPLICATION

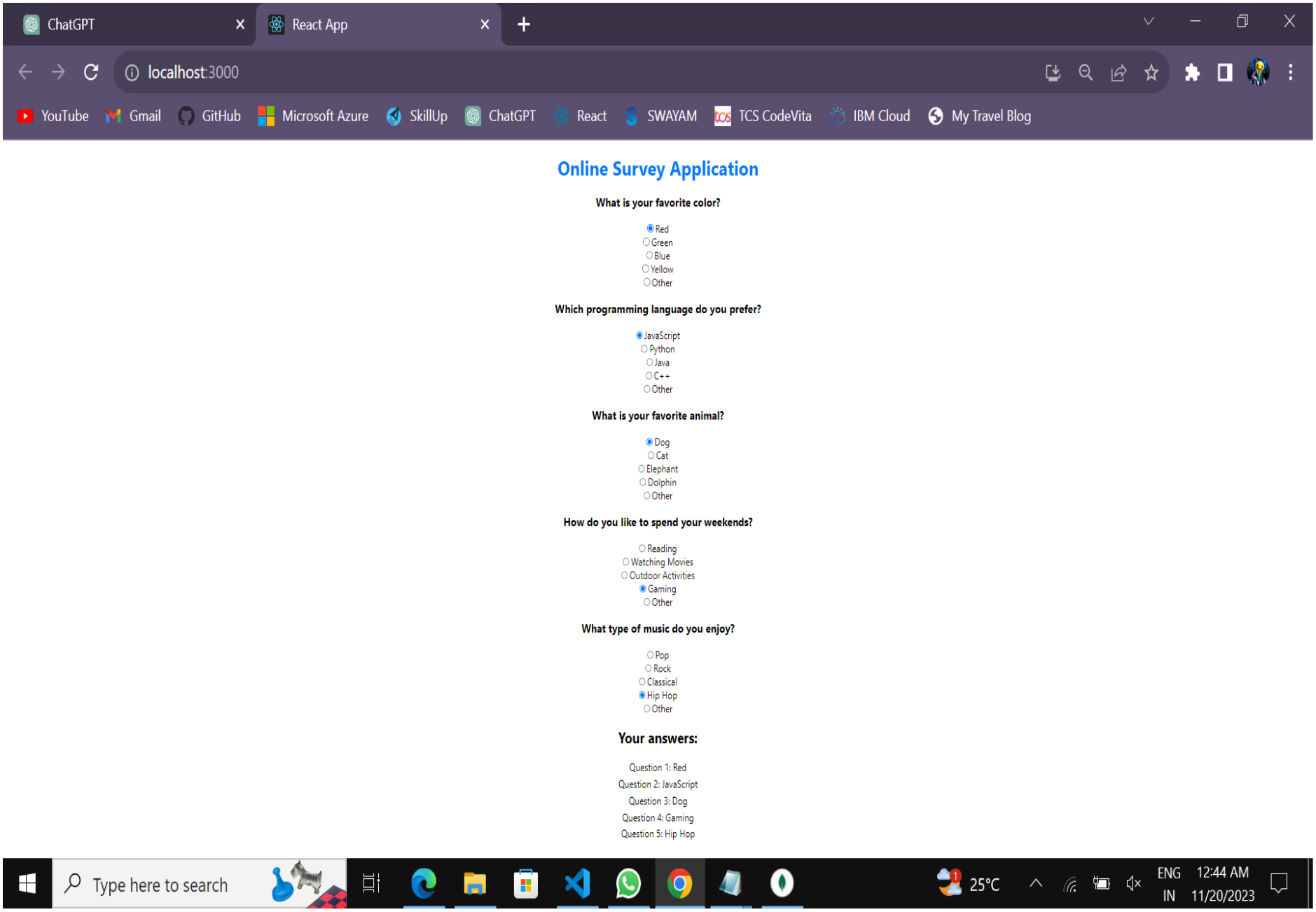
AIM:

Create a dynamic survey application in React where users can answer randomly selected questions, view their responses, and see aggregated survey results.

PROCEDURE:

1. Survey Component Setup: Develop a React component (Survey.js) with state variables for selected questions, survey responses, and result visibility.
2. Question Selection Function: Implement a function (selectRandomQuestions) to randomly select and display 5 questions from the predefined list.
3. Clear Survey Function: Create a function (clearSurvey) to clear input values and reset the selected questions.
4. Submit Survey Function: Implement a function (submitSurvey) to collect and store user responses, clear the survey, and update the responses state.
5. Toggle Results View Function:Develop a function (toggleViewResults) to toggle between survey questions and result views.
6. Render Survey Questions:Render survey questions dynamically based on the selected questions and capture user responses using input fields.
7. Submit Button Functionality:Connect the "Submit" button to gather user responses and trigger the submitSurvey function.
8. Results Component Integration:Integrate a Results component to display survey responses when the "View Results" button is clicked.

OUTPUT:


ChatGPT
React App
localhost:3000
YouTube
Gmail
GitHub
Microsoft Azure
SkillUp
ChatGPT
React
SWAYAM
TCS CodeVita
IBM Cloud
My Travel Blog
Online Survey Application
What is your favorite color?
Red
Green
Blue
Yellow
Other
Which programming language do you prefer?
JavaScript
Python
Java
C++
Other
What is your favorite animal?
Dog
Cat
Elephant
Dolphin
Other
How do you like to spend your weekends?
Reading
Watching Movies
Outdoor Activities
Gaming
Other
What type of music do you enjoy?
Pop
Rock
Classical
Hip Hop
Other
Your answers:
Question 1: Red
Question 2: JavaScript
Question 3: Dog
Question 4: Gaming
Question 5: Hip Hop
Type here to search
25°C
ENG
IN
12:44 AM
11/20/2023

PROGRAM:

Survey.js:

```
import React, { useState, useRef } from 'react';
import Results from './Results';
const questions = [
 'What is your favorite color?',
 'What is your favorite animal?',
 'Where do you live?',
 'What is your age?',
 'What is your gender?',
];
const Survey = () => {
 const [selectedQuestions, setSelectedQuestions] = useState([]);
 const [surveyResponses, setSurveyResponses] = useState([]);
 const [viewResults, setViewResults] = useState(false);
 const inputRefs = useRef([]); // To store references to input elements
 const selectRandomQuestions = () => {
 const randomQuestions = [];
 while (randomQuestions.length < 5) {
 const randomIndex = Math.floor(Math.random() * questions.length);
 if (!randomQuestions.includes(randomIndex)) {
 randomQuestions.push(randomIndex);
 }
 }
 setSelectedQuestions(randomQuestions);
 };
 const clearSurvey = () => {
 inputRefs.current.forEach((inputRef) => {
 inputRef.value = ''; // Clear input values
 });
 };
 const submitSurvey = (responses) => {
 setSurveyResponses([...surveyResponses, responses]);
 clearSurvey();
 };
 const toggleViewResults = () => {
 setViewResults(!viewResults);
```

```
};
return (
<div>
<h1>Survey Questions</h1>
{!viewResults && (
<div>
<button onClick={selectRandomQuestions}>Select Random
Questions</button> <ul>
{selectedQuestions.map((index) => (
<li key={index}>
{questions[index]}
<input type="text" placeholder="Your Answer" ref={(el) =>
(inputRefs.current[index] = el)} />
</li>
))}
</ul>
<button
onClick={() => {
const answers = selectedQuestions.map((index) => inputRefs.
current[index].value);
submitSurvey(answers);
}}
>
Submit
</button>
</div>
)}
<button onClick={toggleViewResults}>
{viewResults ? 'Hide Results' : 'View Results'}
</button>
{viewResults && <Results surveyResponses={surveyResponses} />}
</div>
);
};
export default Survey;
```

Results.js

```
import React from 'react';
```

```
const Results = ({ surveyResponses }) => {
 return (
 <div>
 <h1>Survey Results</h1>
 <ul>
 {surveyResponses.map((response, index) => (
 <li key={index}>
 <strong>Person {index + 1}:</strong>
 <ul>
 {response.map((answer, questionIndex) => (
 <li key={questionIndex}>
 {`Question ${questionIndex + 1}: ${answer}`}
 </li>
 ))}
 </ul>
 </li>
 ))}
 </ul>
 </div>
 );
};
export default Results;
import React from 'react';
import './App.css';
import Survey from './Survey';
function App() {
 return (
 <div className="App">
 <Survey />
 </div>
 );
}
```

RESULT: