GURU NANAK INSTITUTIONS TECHNICAL CAMPUS (AUTONOMOUS)
SCHOOL OF ENGINEERING & TECHNOLOGY

LABORATORY MANUAL

FULL STACK DEVELOPMENT LAB

B.TECH II Year-II Semester

LAB CODE: 22SD0CS02

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AY: 2024-2025

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LAB MANUAL FOR THE ACADEMIC YEAR 2024-25

LAB : **FULL STACK DEVELOPMENT LAB**

LAB CODE : **22SD0CS02**

YEAR **II**

SEMESTER **II**

STREAM : **CSE**

DOCUMENT NO : **GNITC**

VENUE :

BLOCK :

PREPARED BY :

VERIFIED BY

HOD – SPECIAL BATCH

INDEX

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

PEO 1: Graduates shall have the ability to apply knowledge across the disciplines and in emerging areas of Artificial Intelligence and Machine Learning for higher studies, research, employability, product development and handle the realistic problems.

PEO 2: Graduates shall have good communication skills, possess ethical values, sense Responsibility to serve the society, and protect the environment.

PEO 3: Graduates shall possess academic excellence with innovative insight, managerial skills, leadership qualities, knowledge of contemporary issues and understand the need for lifelong learning for a successful professional career.

PROGRAMME OUTCOMES (POs)

Engineering Graduates will be able to:

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PROGRAMME SPECIFIC OUTCOMES (PSOs)

1. Ability to design, develop, test and debug software applications, evaluate and recognize potential risks and provide innovative solutions.

2. Explore technical knowledge in diverse areas of Information Technology for upliftment of society, successful career, entrepreneurship and higher studies.

VISION OF THE DEPARTMENT

To become a premier Computer Science and Engineering department by imparting high quality education, ethical values, provide creative environment for innovation and global opportunities

MISSION OF THE DEPARTMENT

M1: Nurture young individuals into knowledgeable, skilful and ethical professionals in their pursuit of Information Technology.
M2: Transform the students through soft skills, excellent teaching learning process and sustain high performance by innovations.
M3: Extensive partnerships and collaborations with foreign universities to enrich the knowledge and research.
M4: Develop industry-interaction for innovation and product development to provide good placements.

Awarding the marks for day to day evaluation:

- Total marks for day-to-day evaluation is 15 Marks as per JNTUH.
- These 15 Marks are distributed as:

Record	5 Marks
Experiment setup/program written and execution	5 Marks
Result and Viva-Voce	5 Marks

Allocation of Internal Marks

Total marks for lab internal are 40 Marks as per JNTUH/GNITC.
These 40 Marks are distributed as:

Average of day to day evaluation marks : 15 Marks
Lab Mid exam : 10 Marks

Allocation of External Marks

- Total marks for lab external are 60 Marks as per AUTONOMOUS, JNTUH.
- These 70 Marks are distributed as:

Program Written : 20 Marks
Program Execution and Result : 20 Marks
Viva-Voce : 20 Marks
Record : 10 Marks

Introduction about Lab

There are 60 systems installed in this Lab. Their configurations are as follows:

- Processor : Intel(R) Pentium(R) Dual CPU 2.0GHz
- RAM : 4 GB
- Hard Disk : 500 GB
- Mouse : Optical Mouse

Standard Operating Procedure (SOP)

a) Explanation about the experiment by the concerned faculty using PPT covering the following aspects:

1) Name of the experiment/Aim
2) Software/Hardware required
3) Commands with suitable Options
4) Test Data
5) Valid data sets
6) Limiting value sets
7) Invalid data sets

b) Writing of shell programs by the students

c) Compiling and execution of the program 90 mins.

Write-up in the Observation Book

The students will write the today's experiment in the Observation book as per the following format:

a) Name of the experiment/Aim
b) Software/Hardware required
c) Commands with suitable Options
d) Shell Programs/System call using C-Programs
e) Test Data
    1. Valid data sets
    2. Limiting value sets
    3. Invalid data sets
f) Results for different data sets
g) Viva-Voce Questions and Answers
h) Errors observed (if any) during compilation/execution
i) Signature of the Faculty

Guidelines to the Students

Disciplinary to be maintained by the students

- Students are asked to carry their lab observation book and record book.
- Students must use the equipments with care, any damage caused to the equipment by the student is punishable.
- Students are not allowed to use their cell phones/pendrives/CDs.
- Student need to maintain proper dress code.
- Student are supposed to occupy the systems allotted to them.
- Students are not allowed to make noise in the lab.
- After completion of each experiment student need to update their observation notes and same to be reflected in the record.

- Lab records needs to be submitted after completion of each experiment and get it corrected with the concerned lab faculty.
- If a student is absent for any lab, he/she need to complete the experiment in the free time before attending the next lab.

Steps to perform experiment

- Step1: Students have to write the Date, aim, Software and Hardware requirements for the scheduled experiment in the observation book.
- Step2: Students have to listen and understand the experiment explained by the faculty and note down the important points in the observation book.
- Step3: Students need to write procedure/algorithm in the observation book.
- Step4: Analyze and Develop/implement the logic of the program by the student in respective platform
- Step5: After approval of logic of the experiment by the faculty then the experiment has to be executed on the system.
- Step6: After successful execution, the results have to be recorded in the observation book and shown to the lab in charge faculty..
- Step7: Students need to attend the Viva-Voce on that experiment and write the same in the observation book.
- Step8: Update the completed experiment in the record and submit to the concerned faculty in- charge.

Instructions to maintain the record

- Before staring of the first lab session students must buy the record book and bring the same to the lab.
- Regularly (Weekly) update the record after completion of the experiment and get it corrected with concerned lab in-charge for continuous evaluation.
- In case the record is lost, inform on the same day to the faculty in charge and submit the new record within 2 days for correction.
- If record is not submitted in time or record is not written properly, the record evaluation marks (5M) will be reduced accordingly.

FULL STACK DEVELOPMENT LAB

COURSE OBJECTIVES

- To implement the static web pages using HTML and do client side validation using JavaScript.
- To design and work with databases using Java
- To develop an end to end application using java full stack.
- To introduce Node JS implementation for server side programming.
- To experiment with single page application development using React.

COURSE OUTCOMES

- At the end of the course, the student will be able to,
- Build a custom website with HTML, CSS, and Bootstrap and little JavaScript.
- Demonstrate Advanced features of JavaScript and learn about JDBC
- Develop Server – side implementation using Java technologies like
- Develop the server – side implementation using Node JS.
- Design a Single Page Application using React.

S.NO	LIST OF EXPERIMENTS
1	Build a responsive web application for shopping cart with registration, login, catalog and cart pages using CSS3 features, flex and grid.
2	Make the above web application responsive web application using Bootstrap framework.
3	Use JavaScript for doing client – side validation of the pages implemented in experiment 1 and experiment 2.
4	Explore the features of ES6 like arrow functions, call backs, promises, async/await. Implement an application for reading the weather information from openweathermap.org and display the information in the form of a graph on the web page.
5	Develop a java standalone application that connects with the database (Oracle / mySql) and perform the CRUD operation on the database tables.
6	Create an xml for the bookstore. Validate the same using both DTD and XSD.
7	Design a controller with servlet that provides the interaction with application developed in experiment 1 and the database created in experiment 5.
8	Maintaining the transactional history of any user is very important. Explore the various session tracking mechanism (Cookies, HTTP Session).
9	Create a custom server using http module and explore the other modules of Node JS like OS, path, event.

10	Develop an express web application that can interact with REST API to perform CRUD operations on student data. (Use Postman).
11	For the above application create authorized end points using JWT (JSON Web Token).
12	Create a react application for the student management system having registration, login, contact, about pages and implement routing to navigate through these pages.
13	Create a service in react that fetches the weather information from openweathermap.org and the display the current and historical weather information using graphical representation usingchart.js.
14	Create a TODO application in react with necessary components and deploy it into github.

REFERENCE BOOK:

- Jon Duckett, Beginning HTML, XHTML, CSS, and JavaScript, Wrox Publications, 2010
- Bryan Basham, Kathy Sierra and Bert Bates, Head First Servlets and JSP,
- O'Reilly Media, 2ndEdition, 2008.
- Vasan Subramanian, Pro MERN Stack, Full Stack Web App Development with Mongo, Express, React, and Node, 2nd Edition, A Press.

LAB EXPERIMENTS

1. **Build a responsive web application for shopping cart with registration, login, catalog and cart pages using CSS3 features, flex and grid.**

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>Shopping Cart</title>
</head>
<body>
  <header>
    <h1>Shopping Cart</h1>
    <nav>
      <ul>
        <li><a href="#catalog">Catalog</a></li>
        <li><a href="#cart">Cart</a></li>
        <li><a href="#login">Login</a></li>
        <li><a href="#register">Register</a></li>
      </ul>
    </nav>
  </header>

  <main>
```

```
    <section id="catalog">
      <h2>Catalog</h2>
      <div class="product">Product 1</div>
      <div class="product">Product 2</div>
      <div class="product">Product 3</div>
    </section>

    <section id="cart">
      <h2>Shopping Cart</h2>
      <!-- Cart contents go here -->
    </section>

    <section id="login">
      <h2>Login</h2>
      <!-- Login form goes here -->
    </section>

    <section id="register">
      <h2>Register</h2>
      <!-- Registration form goes here -->
    </section>
  </main>

  <footer>
    <p>&copy; 2024 Shopping Cart App</p>
  </footer>
</body>
</html>
```

styles.css:

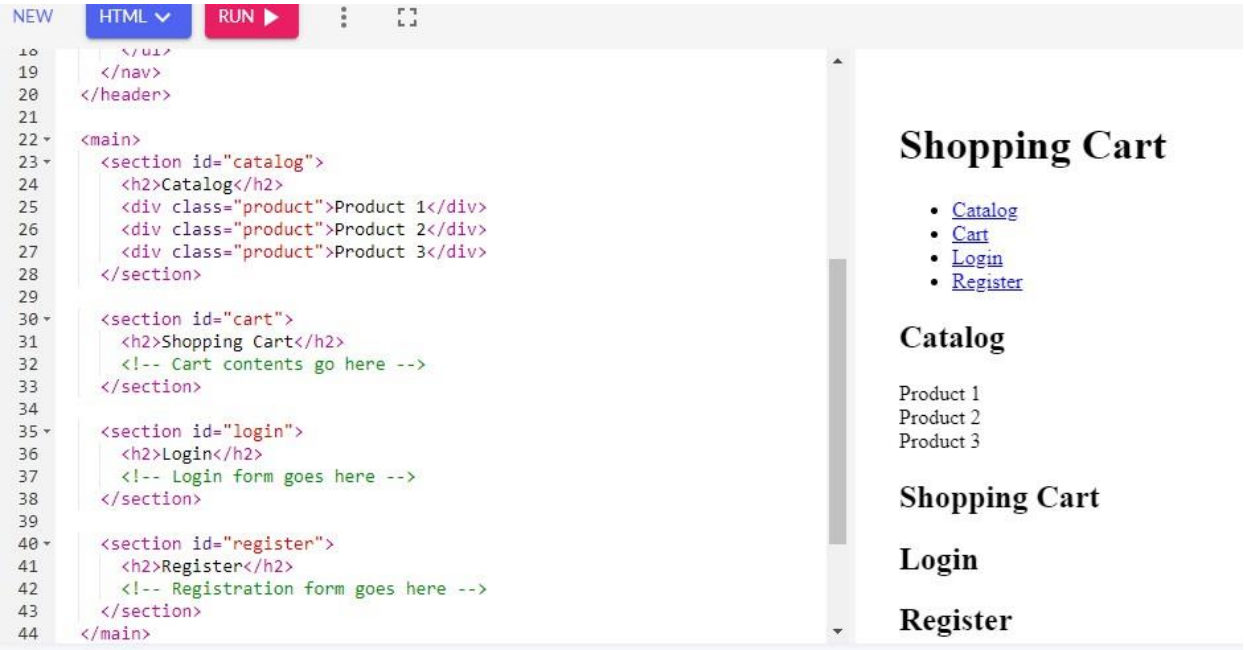
```
body {
  font-family: 'Arial', sans-serif;
  margin: 0;
  padding: 0;
}

header {
  background-color: #333;
  color: #fff;
  padding: 1em;
  text-align: center;
}

nav ul {
  list-style: none;
  padding: 0;
  display: flex;
  justify-content: space-around;
}

nav a {
  text-decoration: none;
  color: #fff;
  font-weight: bold;
}

main {
```

```
  padding: 1em;
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
  gap: 20px;
}

section {
  margin-bottom: 2em;
  padding: 1em;
  border: 1px solid #ddd;
  border-radius: 8px;
}

footer {
  background-color: #333;
  color: #fff;
  text-align: center;
  padding: 1em;
  position: fixed;
  bottom: 0;
  width: 100%;
}

.product {
  border: 1px solid #ddd;
  padding: 1em;
  border-radius: 8px;
  text-align: center;
```

}

/* Add additional styles for catalog, cart, login, and register sections */

OUTPUT:



2. **Make the above web application responsive web application using Bootstrap framework.**

PROGRAM:

index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <!-- Bootstrap CSS CDN -->
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">
  <title>Shopping Cart</title>
</head>
<body>
  <!-- Your existing HTML content goes here -->
</body>
<!-- Bootstrap JS and Popper.js CDN -->
<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd/popper.min.js"
></script>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.min.js"></scrip
t>
</html>
```

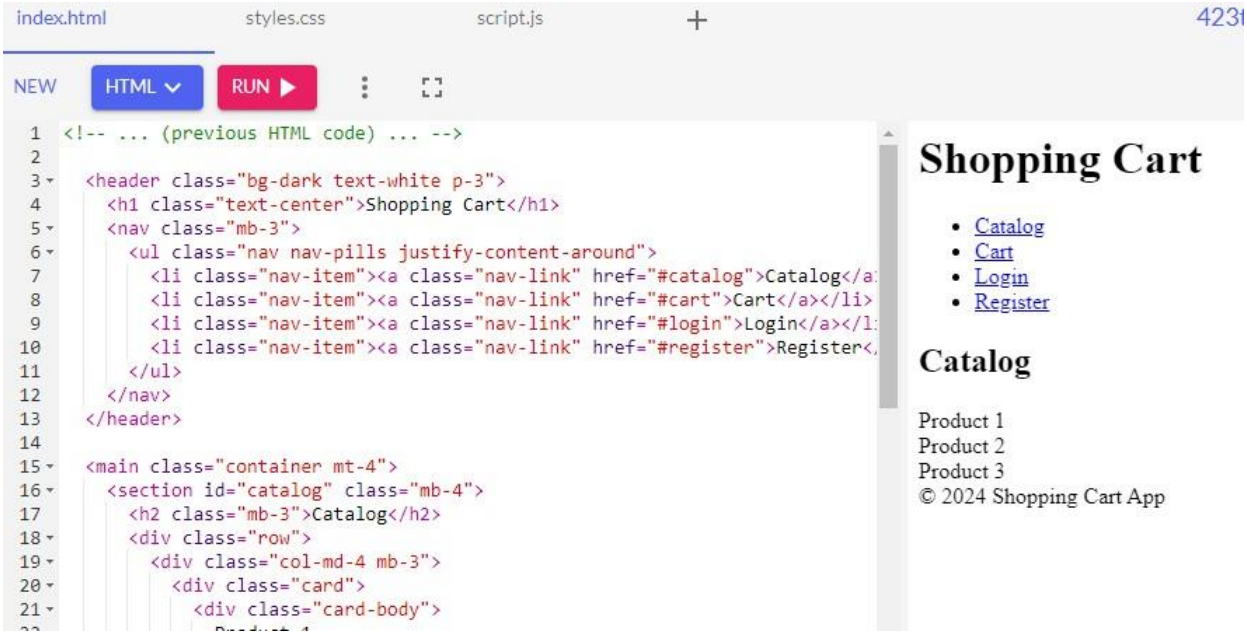
index.html:

```
<!-- ... (previous HTML code) ... -->

  <header class="bg-dark text-white p-3">
    <h1 class="text-center">Shopping Cart</h1>
    <nav class="mb-3">
      <ul class="nav nav-pills justify-content-around">
        <li class="nav-item"><a class="nav-link" href="#catalog">Catalog</a></li>
        <li class="nav-item"><a class="nav-link" href="#cart">Cart</a></li>
        <li class="nav-item"><a class="nav-link" href="#login">Login</a></li>
        <li class="nav-item"><a class="nav-link" href="#register">Register</a></li>
      </ul>
    </nav>
  </header>

  <main class="container mt-4">
    <section id="catalog" class="mb-4">
      <h2 class="mb-3">Catalog</h2>
      <div class="row">
        <div class="col-md-4 mb-3">
          <div class="card">
            <div class="card-body">
              Product 1
            </div>
          </div>
        </div>
```

```
        <div class="col-md-4 mb-3">
          <div class="card">
            <div class="card-body">
              Product 2
            </div>
          </div>
        </div>
        <div class="col-md-4 mb-3">
          <div class="card">
            <div class="card-body">
              Product 3
            </div>
          </div>
        </div>
      </div>
    </section>

    <!-- ... (other sections) ... -->

  </main>

  <footer class="bg-dark text-white text-center p-3 fixed-bottom">
    &copy; 2024 Shopping Cart App
  </footer>

<!-- ... (Bootstrap JS and Popper.js CDN) ... -->
```

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

OUTPUT:



3. Use JavaScript for doing client – side validation of the pages implemented in experiment 1 and experiment 2.

PROGRAM:

index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <!-- ... (previous head content) ... -->
  <script src="validate.js" defer></script>
</head>
<body>
  <!-- ... (previous body content) ... -->
  <section id="login">
    <h2>Login</h2>
    <form id="loginForm" onsubmit="return validateLoginForm()">
      <label for="username">Username:</label>
      <input type="text" id="username" name="username" required>
      <br>
      <label for="password">Password:</label>
      <input type="password" id="password" name="password" required>
      <br>
      <input type="submit" value="Login">
    </form>
  </section>

  <section id="register">
```

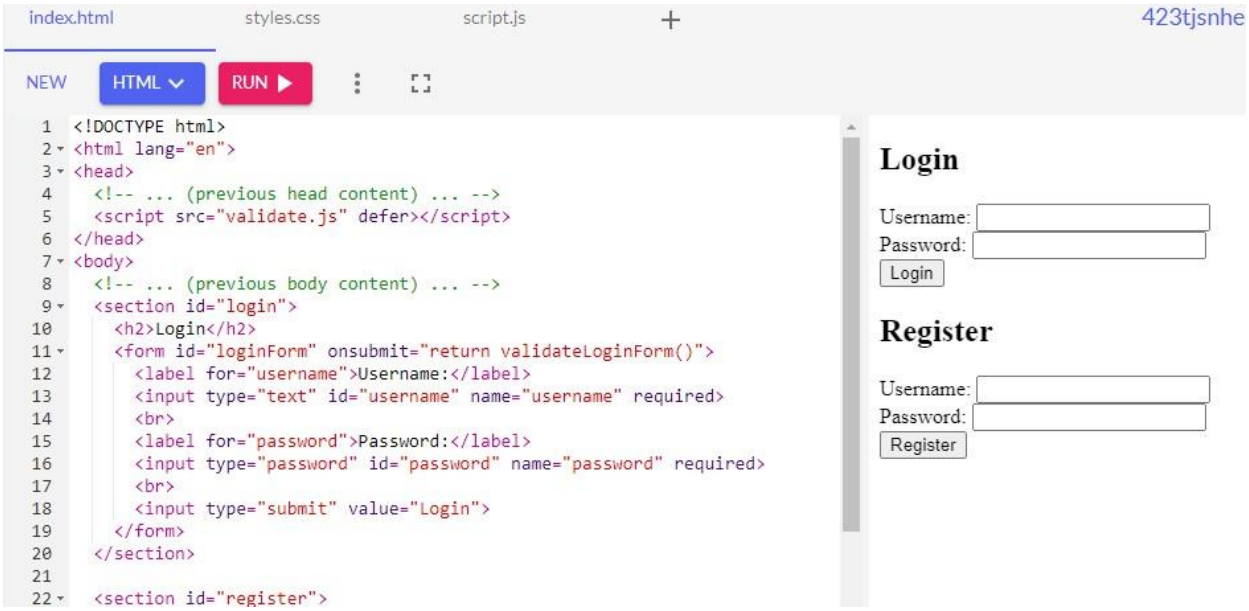
```
    <h2>Register</h2>
    <form id="registerForm" onsubmit="return validateRegisterForm()">
      <label for="newUsername">Username:</label>
      <input type="text" id="newUsername" name="newUsername" required>
      <br>
      <label for="newPassword">Password:</label>
      <input type="password" id="newPassword" name="newPassword" required>
      <br>
      <input type="submit" value="Register">
    </form>
  </section>
  <!-- ... (remaining body content) ... -->
</body>
</html>
```

validate.js:

```
function validateLoginForm() {
  var username = document.getElementById('username').value;
  var password = document.getElementById('password').value;

  if (username === '' || password === '') {
    alert('Please fill in all fields for login.');
    return false;
  }

  // Add more validation as needed
```

```
  return true;
}

function validateRegisterForm() {
  var newUsername = document.getElementById('newUsername').value;
  var newPassword = document.getElementById('newPassword').value;

  if (newUsername === '' || newPassword === '') {
    alert('Please fill in all fields for registration.');
    return false;
  }

  // Add more validation as needed

  return true;
}
```

OUTPUT:



4. **Explore the features of ES6 like arrow functions, call backs, promises, async/await. Implement an application for reading the weather information from openweathermap.org and display the information in the form of a graph on the web page.**

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>Weather Graph</title>
  <script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
  <script src="app.js" defer></script>
</head>
<body>
  <div>
    <h1>Weather Graph</h1>
    <canvas id="weatherChart" width="800" height="400"></canvas>
  </div>
</body>
</html>
```

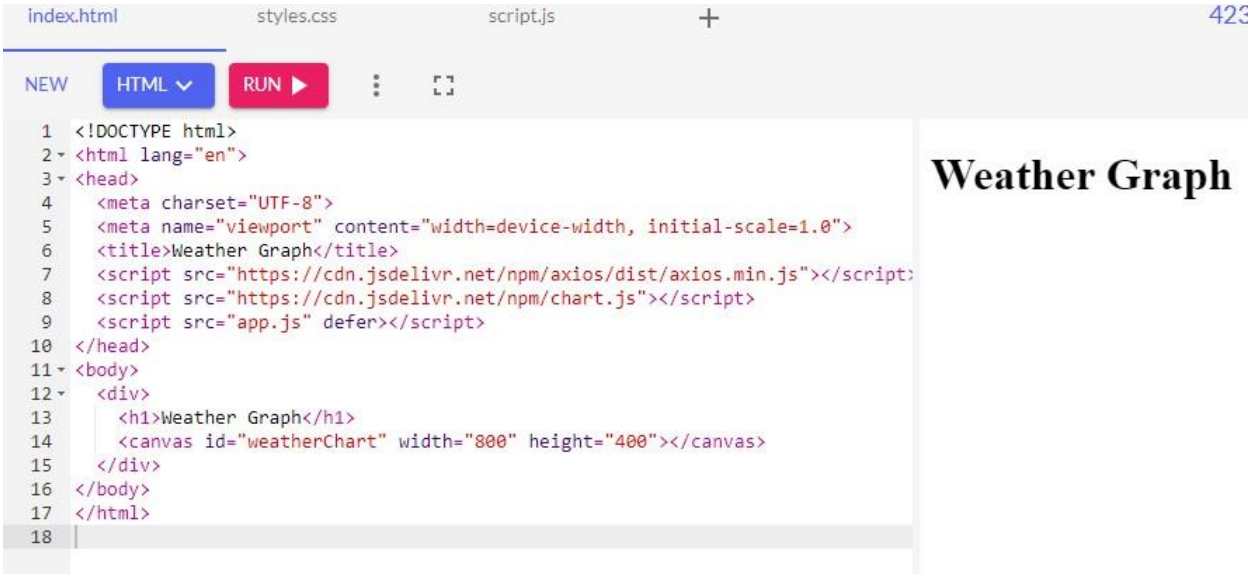
app.js:

```
// Replace 'YOUR_API_KEY' with your OpenWeatherMap API key
const apiKey = 'YOUR_API_KEY';
const city = 'London'; // Change to the desired city

const apiUrl = `https://api.openweathermap.org/data/2.5/forecast?q=${city}&appid=${apiKey}`;

// Fetch weather data from OpenWeatherMap API
const getWeatherData = async () => {
  try {
    const response = await axios.get(apiUrl);
    return response.data.list.map(item => ({
      date: new Date(item.dt * 1000), // Convert timestamp to Date object
      temperature: item.main.temp - 273.15, // Convert temperature from Kelvin to Celsius
    }));
  } catch (error) {
    console.error('Error fetching weather data:', error);
    throw error;
  }
};

// Create a line chart using Chart.js
const createChart = (weatherData) => {
  const ctx = document.getElementById('weatherChart').getContext('2d');
```

```
  new Chart(ctx, {
    type: 'line',
    data: {
      labels: weatherData.map(item => item.date.toLocaleDateString()),
      datasets: [{
        label: 'Temperature (°C)',
        data: weatherData.map(item => item.temperature.toFixed(2)),
        borderColor: 'rgba(75, 192, 192, 1)',
        borderWidth: 2,
        fill: false,
      }],
    },
    options: {
      scales: {
      x: {
          type: 'linear',
          position: 'bottom',
        },
        y: {
          beginAtZero: true,
        },
      },
    },
  });
};
```

```
// Main function to fetch data and create chart
const main = async () => {
  try {
    const weatherData = await getWeatherData();
    createChart(weatherData);
  } catch (error) {
    console.error('An error occurred:', error);
  }
};

// Run the main function
main();
```

OUTPUT:



5. **Develop a java standalone application that connects with the database (Oracle / mySql) and perform the CRUD operation on the database tables.**

PROGRAM:

```
import java.sql.*;

public class DatabaseCRUDExample {

    // JDBC URL, username, and password of MySQL server

    private static final String JDBC_URL =
"jdbc:mysql://localhost:3306/your_database";

    private static final String USERNAME = "your_username";

    private static final String PASSWORD = "your_password";

    public static void main(String[] args) {

        try {

            // Load the JDBC driver

            Class.forName("com.mysql.cj.jdbc.Driver");

            // Establish a connection

            Connection connection = DriverManager.getConnection(JDBC_URL,
USERNAME, PASSWORD);

            // Perform CRUD operations

            createRecord(connection, "John Doe", 25);

            readRecords(connection);

            updateRecord(connection, 1, "Updated Name", 30);

            deleteRecord(connection, 2);


            // Close the connection

            connection.close();

        } catch (ClassNotFoundException | SQLException e) {
```

```
            e.printStackTrace();
        }
    }
    private static void createRecord(Connection connection, String name, int age)
throws SQLException {
        String insertQuery = "INSERT INTO users (name, age) VALUES (?, ?)";
        try (PreparedStatement preparedStatement =
connection.prepareStatement(insertQuery))  {
            preparedStatement.setString(1, name);
            preparedStatement.setInt(2, age);
            preparedStatement.executeUpdate();
            System.out.println("Record created successfully.");
        }
    }
    private static void readRecords(Connection connection) throws SQLException {
        String selectQuery = "SELECT * FROM users";
        try (Statement statement = connection.createStatement();
            ResultSet resultSet = statement.executeQuery(selectQuery)) {
            System.out.println("User Records:");
            while (resultSet.next()) {
                int id = resultSet.getInt("id");
                String name = resultSet.getString("name");
                int age = resultSet.getInt("age");
                System.out.println("ID: " + id + ", Name: " + name + ", Age: " + age);
            }
        }
    }
```

```
    private static void updateRecord(Connection connection, int id, String name, int
age) throws SQLException {
        String updateQuery = "UPDATE users SET name = ?, age = ? WHERE id =
?";
        try (PreparedStatement preparedStatement =
connection.prepareStatement(updateQuery)) {
            preparedStatement.setString(1, name);
            preparedStatement.setInt(2, age);
            preparedStatement.setInt(3, id);
            preparedStatement.executeUpdate();
            System.out.println("Record updated successfully.");
        }
    }

    private static void deleteRecord(Connection connection, int id) throws
SQLException {
        String deleteQuery = "DELETE FROM users WHERE id = ?";
        try (PreparedStatement preparedStatement =
connection.prepareStatement(deleteQuery))  {
            preparedStatement.setInt(1, id);
            preparedStatement.executeUpdate();
            System.out.println("Record deleted successfully.");
        }
    }
}
```

OUTPUT:

6. **Create an xml for the bookstore. Validate the same using both DTD and XSD.**

PROGRAM:

bookstore.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE bookstore SYSTEM "bookstore.dtd">
<bookstore>
  <book>
    <title>Harry Potter and the Sorcerer's Stone</title>
    <author>J.K. Rowling</author>
    <genre>Fantasy</genre>
    <price>19.99</price>
  </book>
  <book>
    <title>To Kill a Mockingbird</title>
    <author>Harper Lee</author>
    <genre>Classic</genre>
    <price>14.99</price>
  </book>
  <!-- Add more books as needed -->
</bookstore>
```

bookstore.dtd:

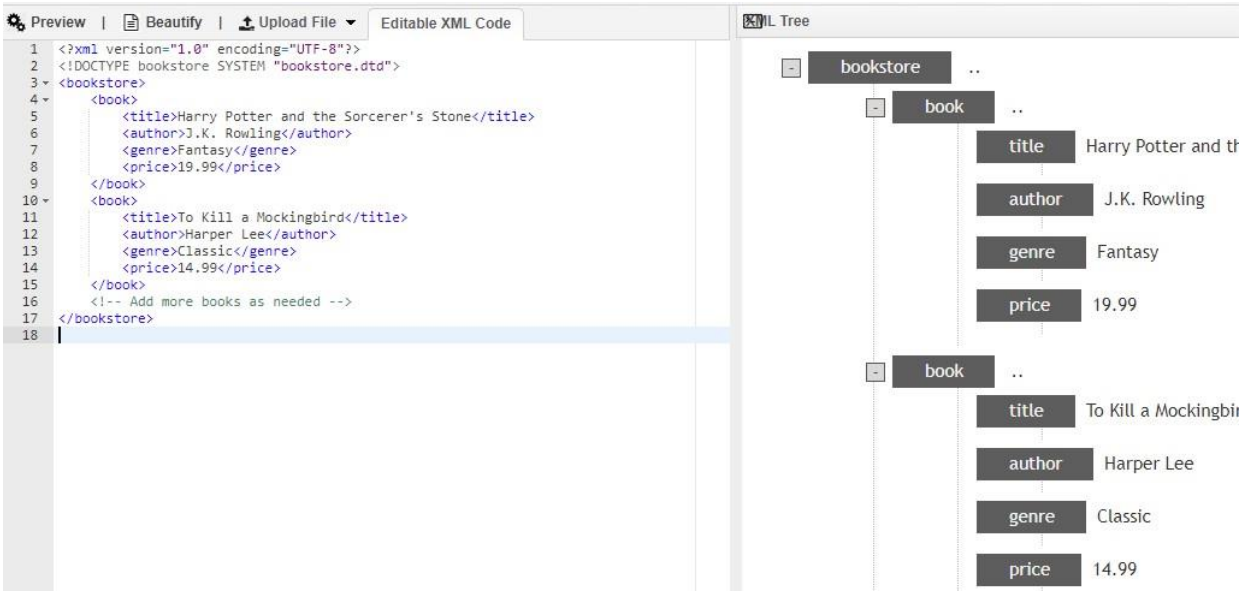
```
<!ELEMENT bookstore (book+)>
<!ELEMENT book (title, author, genre, price)>
<!ELEMENT title (#PCDATA)>
```

```
<!ELEMENT author (#PCDATA)>
<!ELEMENT genre (#PCDATA)>
<!ELEMENT price (#PCDATA)>
```

bookstore.xsd:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="bookstore" type="BookstoreType"/>
    <xs:complexType name="BookstoreType">
        <xs:sequence>
            <xs:element name="book" type="BookType" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="BookType">
        <xs:sequence>
            <xs:element name="title" type="xs:string"/>
            <xs:element name="author" type="xs:string"/>
            <xs:element name="genre" type="xs:string"/>
            <xs:element name="price" type="xs:decimal"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

OUTPUT:


Preview | Beautify | Upload File
Editable XML Code
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE bookstore SYSTEM "bookstore.dtd">
3 <bookstore>
4 <book>
5 <title>Harry Potter and the Sorcerer's Stone</title>
6 <author>J.K. Rowling</author>
7 <genre>Fantasy</genre>
8 <price>19.99</price>
9 </book>
10 <book>
11 <title>To Kill a Mockingbird</title>
12 <author>Harper Lee</author>
13 <genre>Classic</genre>
14 <price>14.99</price>
15 </book>
16 <!-- Add more books as needed -->
17 </bookstore>
18
ML Tree
bookstore ..
book ..
title Harry Potter and th
author J.K. Rowling
genre Fantasy
price 19.99
book ..
title To Kill a Mockingbi
author Harper Lee
genre Classic
price 14.99

7. Design a controller with servlet that provides the interaction with application developed in experiment 1 and the database created in experiment 5.

PROGRAM:

Experiment1Servlet.java:

```
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/experiment1")
public class Experiment1Servlet extends HttpServlet {
    private static final String JDBC_URL =
"jdbc:mysql://localhost:3306/your_database";
    private static final String USERNAME = "your_username";
    private static final String PASSWORD = "your_password";

    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws IOException {
```

```
        PrintWriter out = response.getWriter();

        try {
            // Load the JDBC driver
            Class.forName("com.mysql.cj.jdbc.Driver");

            // Establish a connection
            Connection connection = DriverManager.getConnection(JDBC_URL,
USERNAME, PASSWORD);

            // Perform a sample database operation
            int totalUsers = getTotalUsers(connection);

            // Display the result
            out.println("<html><body>");
            out.println("<h2>Total Users in the Database: " + totalUsers + "</h2>");
            out.println("</body></html>");

            // Close the connection
            connection.close();
        } catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
            out.println("Error: " + e.getMessage());
        }
    }
```

```
    private int getTotalUsers(Connection connection) throws SQLException {
        String query = "SELECT COUNT(*) FROM users";
        try (PreparedStatement preparedStatement =
connection.prepareStatement(query);
             ResultSet resultSet = preparedStatement.executeQuery()) {
            if (resultSet.next()) {
                return resultSet.getInt(1);
            }
        }
        return 0;
    }
}
```

OUTPUT:

8. **Maintaining the transactional history of any user is very important. Explore the various session tracking mechanism (Cookies, HTTP Session).**

Cookies:

How it works:

Cookies are small pieces of data stored on the client-side. When a user visits a website, the server sends cookies to the client, and the client stores them locally. On subsequent requests, the client sends the stored cookies back to the server.

Usage for Session Tracking:

A unique session identifier can be stored in a cookie.

Cookies can store user preferences, language settings, or any other data.

Pros:

Simple to implement.

Lightweight and efficient.

Cons:

Limited storage space (usually around 4KB per domain).

Vulnerable to security threats (e.g., cross-site scripting attacks).

HTTP Session:

How it works:

The server creates a unique session for each user, and a session identifier is generated. This identifier is sent to the client, usually via a cookie or URL rewriting. The server uses the identifier to associate subsequent requests with the correct session data.

Usage for Session Tracking:

Session attributes can be stored on the server and associated with a specific user's session.

Commonly used for authentication and authorization.

Pros:

No size limitation on data stored in the session.

More secure than cookies.

Cons:

Requires server resources to manage and store sessions.

May impact performance if not managed properly.

URL Rewriting:

How it works:

Session ID is appended to URLs, allowing the server to associate requests with specific sessions. This is an alternative method when cookies are disabled.

Usage for Session Tracking:

Can be used in combination with cookies or independently.

Pros:

Works even if cookies are disabled.

Cons:

URLs may become lengthy and less user-friendly.

Vulnerable to security threats.

Hidden Form Fields:

How it works:

Hidden form fields in HTML forms carry session information. This is often used for session tracking in web applications.

Usage for Session Tracking:

Values are included in HTML forms and submitted with each request.

Pros:

Works without cookies.

Cons:

Data is visible in the HTML source code.

Limited storage space.

9. Create a custom server using http module and explore the other modules of Node JS like OS, path, event.

Program:

Custom Server using http Module:

```
const http = require('http');
const server = http.createServer((req, res) => {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello, custom server!');
});
const PORT = 3000;
server.listen(PORT, () => {
  console.log(`Server running at http://localhost:${PORT}/`);
});
```

OS Module:

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

console.log('OS Platform:', os.platform());
console.log('OS Architecture:', os.arch());
console.log('Total Memory:', os.totalmem() / (1024 * 1024) + ' MB');
console.log('Free Memory:', os.freemem() / (1024 * 1024) + ' MB');
console.log('CPU Cores:', os.cpus().length);
```

Path Module:

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

const filePath = '/path/to/some/file.txt';

console.log('File Name:', path.basename(filePath));
console.log('Directory Name:', path.dirname(filePath));
console.log('File Extension:', path.extname(filePath));
console.log('Normalized Path:', path.normalize(filePath));
```

Event Module:

```
const EventEmitter = require('events');
class MyEmitter extends EventEmitter {}
const myEmitter = new MyEmitter();
myEmitter.on('event', () => {
console.log('Event emitted!');
});
myEmitter.emit('event');
```

OUTPUT:

10. Develop an express web application that can interact with REST API to perform CRUD operations on student data. (Use Postman).

Program:

Initialize the Project:

```
mkdir express-rest-api
cd express-rest-api
npm init -y
npm install express body-parser
```

Create the Express App:

```
const express = require('express');
const bodyParser = require('body-parser');
const app = express();
const PORT = 3000;
// Middleware
app.use(bodyParser.json());
// Routes
const students = [
  { id: 1, name: 'John Doe', age: 20 },
  { id: 2, name: 'Jane Smith', age: 22 },
];
app.get('/students', (req, res) => {
  res.json(students);
});
app.get('/students/:id', (req, res) => {
  const student = students.find(s => s.id === parseInt(req.params.id));
```

```
  if (!student) return res.status(404).json({ message: 'Student not found' });
  res.json(student);
});
app.post('/students', (req, res) => {
  const newStudent = req.body;
  newStudent.id = students.length + 1;
  students.push(newStudent);
  res.status(201).json(newStudent);
});
app.put('/students/:id', (req, res) => {
  const student = students.find(s => s.id === parseInt(req.params.id));
  if (!student) return res.status(404).json({ message: 'Student not found' });
  student.name = req.body.name || student.name;
  student.age = req.body.age || student.age;
  res.json(student);
});
app.delete('/students/:id', (req, res) => {
  const index = students.findIndex(s => s.id === parseInt(req.params.id));
  if (index === -1) return res.status(404).json({ message: 'Student not found' });
  students.splice(index, 1);
  res.json({ message: 'Student deleted successfully' });
});
// Start the server
app.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});
```

Run the Express App:

node index.js

Test with Postman:

Open Postman and test the CRUD operations on the student data:

GET All Students:

URL: http://localhost:3000/students

Method: GET

GET a Specific Student:

URL: http://localhost:3000/students/1 (replace 1 with an existing student ID)

Method: GET

POST (Create) a Student:

URL: http://localhost:3000/students

Method: POST

Body: JSON with student data

PUT (Update) a Student:

URL: http://localhost:3000/students/1 (replace 1 with an existing student ID)

Method: PUT

Body: JSON with updated student data

DELETE a Student:

URL: http://localhost:3000/students/1 (replace 1 with an existing student ID)

Method: DELETE

OUTPUT:

11. For the above application create authorized end points using JWT (JSON Web Token)

PROGRAM:

Install the jsonwebtoken library:

npm install jsonwebtoken

Modify the index.js file:

```
const express = require('express');
const bodyParser = require('body-parser');
const jwt = require('jsonwebtoken');
const app = express();
const PORT = 3000;
const SECRET_KEY = 'your_secret_key'; // Replace with a strong secret key
// Middleware
app.use(bodyParser.json());
// Authentication middleware
const authenticateJWT = (req, res, next) => {
  const token = req.header('Authorization');
  if (!token) return res.status(401).json({ message: 'Unauthorized: Missing token' });
  jwt.verify(token, SECRET_KEY, (err, user) => {
    if (err) return res.status(403).json({ message: 'Forbidden: Invalid token' });
    req.user = user;
    next();
  });
};
```

```
// Routes
const students = [
  { id: 1, name: 'John Doe', age: 20 },
  { id: 2, name: 'Jane Smith', age: 22 },
];
app.post('/login', (req, res) => {
  const { username, password } = req.body;
  // You should validate the username and password against your user database
  if (username === 'user' && password === 'password') {
    const token = jwt.sign({ username }, SECRET_KEY);
    res.json({ token });
  } else {
    res.status(401).json({ message: 'Invalid credentials' });
  }
});
// Protected route
app.get('/students', authenticateJWT, (req, res) => {
  res.json(students);
});
// ... (other routes are protected similarly)
// Start the server
app.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});
```

Test with Postman:

Login to Get Token:

URL: http://localhost:3000/login

Method: POST

Body: JSON with username and password (e.g., { "username": "user", "password": "password" })

Copy the token from the response.

Use Token to Access Protected Routes:

For protected routes (e.g., http://localhost:3000/students), include the token in the Authorization header with the Bearer scheme.

OUTPUT:

12. Create a react application for the student management system having registration, login, contact, about pages and implement routing to navigate through these pages.

PROGRAM:

Create a new React App:

```
npx create-react-app student-management-system
cd student-management-system
```

Install React Router:

```
npm install react-router-dom
```

Create Components: Registration.js:

```
import React from 'react';
const Registration = () => {
  return <div>Registration Page</div>;
};
export default Registration;
```

Login.js:

```
import React from 'react';
const Login = () => {
  return <div>Login Page</div>;
};
export default Login;
```

Contact.js:

```
import React from 'react';
const Contact = () => {
  return <div>Contact Page</div>;
};
export default Contact;
```

About.js:

```
import React from 'react';
const About = () => {
  return <div>About Page</div>;
};
export default About;
```

Create a Router Component:

Router.js:

```
import React from 'react';
import { BrowserRouter as Router, Route, Link, Switch } from 'react-router-dom';
import Registration from './Registration';
import Login from './Login';
import Contact from './Contact';
import About from './About';
const AppRouter = () => {
return (
    <Router>
      <div>
```

```
<nav>
  <ul>
    <li>
      <Link to="/registration">Registration</Link>
    </li>
    <li>
      <Link to="/login">Login</Link>
    </li>
    <li>
      <Link to="/contact">Contact</Link>
    </li>
    <li>
      <Link to="/about">About</Link>
    </li>
  </ul>
</nav>
<Switch>
  <Route path="/registration">
    <Registration />
  </Route>
  <Route path="/login">
    <Login />
  </Route>
  <Route path="/contact">
    <Contact />
  </Route>
```

```
          <Route path="/about">
            <About />
          </Route>
        </Switch>
      </div>
    </Router>
  );
};
export default AppRouter;
```

Use the Router Component in App.js:

```
import React from 'react';
import AppRouter from './Router';
function App() {
  return (
    <div className="App">
      <h1>Student Management System</h1>
      <AppRouter />
    </div>
  );
}
export default App;
```

Run the Application:

```
npm start
```

13. Create a service in react that fetches the weather information from openweathermap.org and the display the current and historical weather information using graphical representation usingchart.js

PROGRAM:

Create a new React App:

npx create-react-app weather-app

cd weather-app

Install Dependencies:

npm install axios react-chartjs-2 chart.js

Create a WeatherService Component:

```
import axios from 'axios'

const API_KEY = 'YOUR_API_KEY'; // Replace with your OpenWeatherMap API key

const API_BASE_URL = 'https://api.openweathermap.org/data/2.5';

const getWeatherData = async (city) => {
  try {
    const response = await axios.get(`${API_BASE_URL}/weather?q=${city}&appid=${API_KEY}`);
    return response.data;
  } catch (error) {
    console.error('Error fetching current weather:', error);
    throw error;
  }
};
```

```
const getHistoricalWeatherData = async (city, days) => {
  const endDate = new Date().toISOString().split('T')[0]; // Today's date
  const startDate = new Date(new Date().setDate(new Date().getDate() - days)).toISOString().split('T')[0]; // Date 'days' ago
  try {
    const response = await axios.get(
`${API_BASE_URL}/onecall/timemachine?lat=${city.lat}&lon=${city.lon}&dt=${Math.round(new Date(startDate).getTime() / 1000)}&appid=${API_KEY}`
    );
    return response.data;
  } catch (error) {
    console.error('Error fetching historical weather:', error);
    throw error;
  }
};
export { getWeatherData, getHistoricalWeatherData };
```

Create a WeatherChart Component:

WeatherChart.js:

```
import React, { useState, useEffect } from 'react';
import { Line } from 'react-chartjs-2';
import { getWeatherData, getHistoricalWeatherData } from './WeatherService';
const WeatherChart = ({ city }) => {
  const [currentWeather, setCurrentWeather] = useState(null);
  const [historicalWeather, setHistoricalWeather] = useState(null);
```

```
  useEffect(() => {
    const fetchData = async () => {
      try {
        const currentWeatherData = await getWeatherData(city);
        setCurrentWeather(currentWeatherData);
        const historicalWeatherData = await getHistoricalWeatherData(city, 7); // Fetch historical data for the last 7 days
        setHistoricalWeather(historicalWeatherData);
      } catch (error) {
        // Handle error
      }
    };
    fetchData();
  }, [city]);
  const chartData = {
    labels: historicalWeather?.hourly.map((hour) => new Date(hour.dt * 1000).toLocaleTimeString([], { hour: 'numeric' })),
    datasets: [
      {
        label: 'Temperature (°C)',
        data: historicalWeather?.hourly.map((hour) => hour.temp - 273.15),
        borderColor: 'rgba(75, 192, 192, 1)',
        borderWidth: 2,
        fill: false,
      },
    ],
```

```
  };
  return (
    <div>
      <h2>Weather Chart for {city.name}</h2>
      <Line data={chartData} />
    </div>
  );
};
export default WeatherChart;
```

Integrate WeatherChart in App.js:

App.js:

```
import React from 'react';
import WeatherChart from './WeatherChart';
const App = () => {
  const city = { name: 'London', lat: 51.509865, lon: -0.118092 }; // Change to the desired city
  return (
    <div className="App">
      <h1>Weather App</h1>
      <WeatherChart city={city} />
    </div>
  );
};
export default App;
```

Run the Application:

npm start

OUTPUT:

14. Create a TODO application in react with necessary components and deploy it into github.

PROGRAM:

Create a New React App:

npx create-react-app react-todo

cd react-todo

Create TODO Components:

TodoList.js:

```
import React from 'react';
import TodoItem from './TodoItem';
const TodoList = ({ todos, onDelete }) => {
  return (
    <ul>
      {todos.map((todo) => (
        <TodoItem key={todo.id} todo={todo} onDelete={onDelete} />
      ))}
    </ul>
  );
};
export default TodoList;
```

TodoItem.js:

```
import React from 'react';
const TodoItem = ({ todo, onDelete }) => {
```

```
  return (
    <li>
      {todo.text} <button onClick={() => onDelete(todo.id)}>Delete</button>
    </li>
  );
};
export default TodoItem;
```

AddTodo.js:

```
import React, { useState } from 'react';
const AddTodo = ({ onAdd }) => {
const [text, setText] = useState('');
const handleAdd = () => {
    if (text.trim() !== '') {
      onAdd({ id: Date.now(), text });
      setText('');
    }
  };
  return (
    <div>
      <input type="text" value={text} onChange={(e) =>
setText(e.target.value)} />
      <button onClick={handleAdd}>Add Todo</button>
    </div>
  );
};
```

```
export default AddTodo;
```

Modify App.js:

```
import React, { useState } from 'react';
import TodoList from './TodoList';
import AddTodo from './AddTodo';
const App = () => {
  const [todos, setTodos] = useState([]);
  const handleAddTodo = (newTodo) => {
  setTodos([...todos, newTodo]);
  };
  const handleDeleteTodo = (id) => {
    setTodos(todos.filter((todo) => todo.id !== id));
  };
  return (
    <div className="App">
      <h1>TODO App</h1>
      <AddTodo onAdd={handleAddTodo} />
      <TodoList todos={todos} onDelete={handleDeleteTodo} />
    </div>
  );
};

export default App;
```

Deploy to GitHub:

git init

git add .

git commit -m "Initial commit"

Deploy Using GitHub Pages:

npm install gh-pages --save-dev

"homepage": "https://your-username.github.io/your-repo-name",

"scripts": {

  "predeploy": "npm run build",

  "deploy": "gh-pages -d build",

  // ... other scripts

}

Deploy your app:

npm run deploy