**Introduction to Node.js**

**Objective:**

* Understand the basic structure of a Node.js application.
* Set up a Node.js environment and run a simple server.

**Requirements:**

* Install Node.js and npm.

**Introduction:**

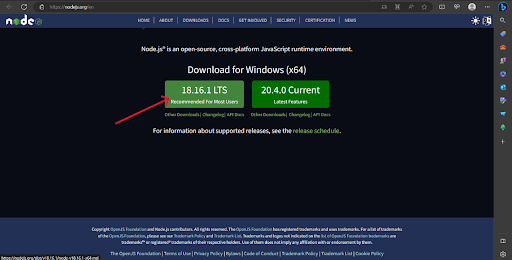
Node.js is an open-source, cross-platform [JavaScript](https://www.geeksforgeeks.org/javascript/) runtime built on [Chrome’s V8 JavaScript engine](https://www.geeksforgeeks.org/explain-v8-engine-in-node-js/). It allows the creation of scalable Web servers without threading and networking tools using JavaScript and a collection of “modules” that handle various core functionalities. It can make console-based and web-based node.js applications.

[Node.js](https://mindmajix.com/node-js/what-is-node-js" \t "_blank" \o "Introduction to Node.js) is basically an open-source runtime environment for JavaScript code that runs outside a web browser. Ryan Dahl designed it in 2009 to allow developers to create JavaScript server-side programs. Node.js runs JavaScript on servers using the V8 JavaScript engine, originally built by Google for the Chrome browser.

**Installation of Node JS on Windows**

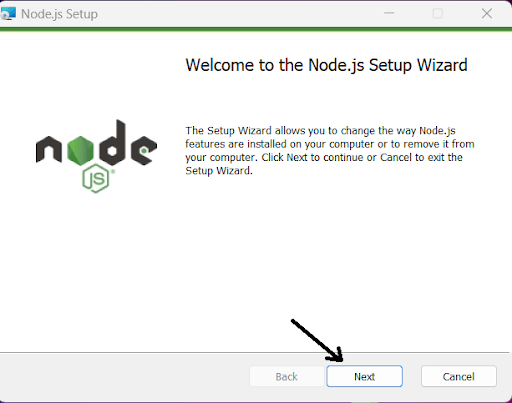
**Below are the steps to install NodeJS on your system:**

**Step 1:** To install NodeJS visit [Node.js (nodejs.org)](https://nodejs.org/en" \t "_blank) and download the LTS (Long Term Support) version.

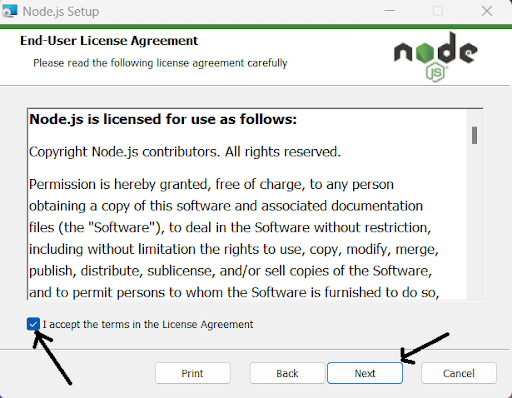


**Step 2:** Once downloaded, open the ‘*.msi’ file*. If the system prompts for, ‘*Do you want to allow this app to make changes to your device?*’ click ‘*Yes.*’

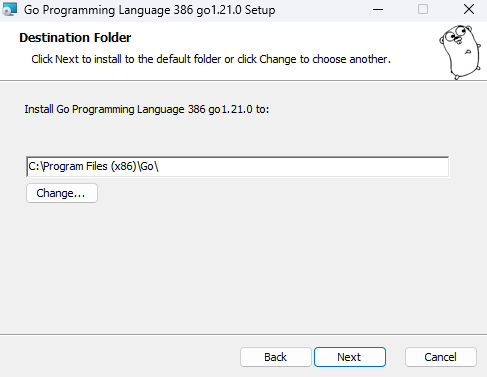
**Step 3:**After clicking ‘*Yes*,’ you will see the below prompt. Click ‘*Next*.’

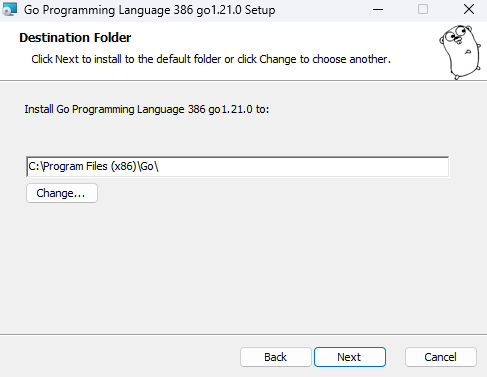


**Step 4:** Mark the checkbox Accept the agreement and click ‘*Next*.’

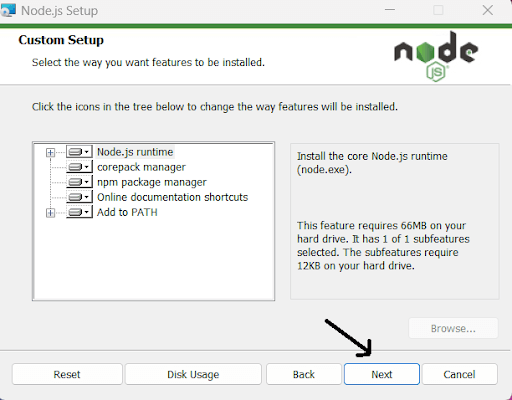


**Step 5:** Confirm the destination folder and click ‘*Next*.’ It is preferred to keep the settings default.

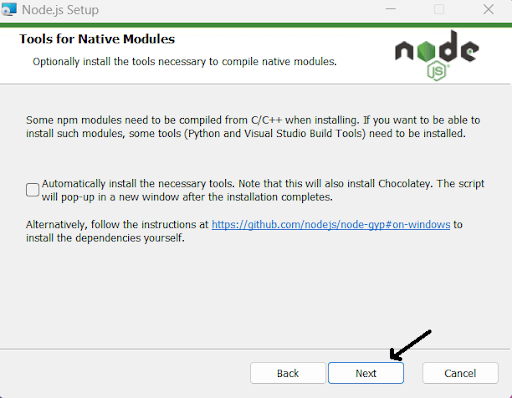




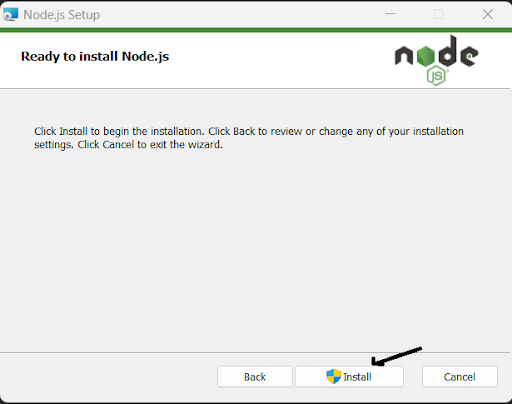
**Step 6:** Click on ‘*Next*’.



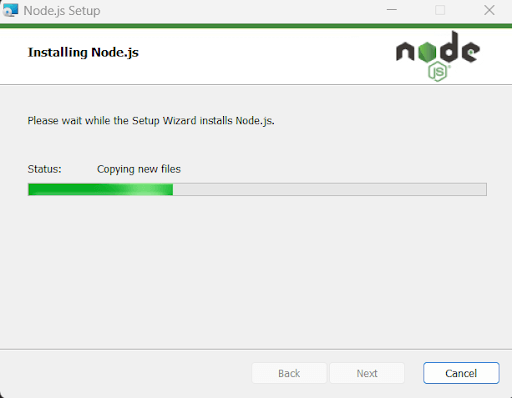
**Step 7:** If you want to install additional tools to be installed, mark the checkbox and click ‘*Next*.’ Else, you can click on ‘*Next*.’ (It is recommended to keep the box *unchecked* and click *Next*).



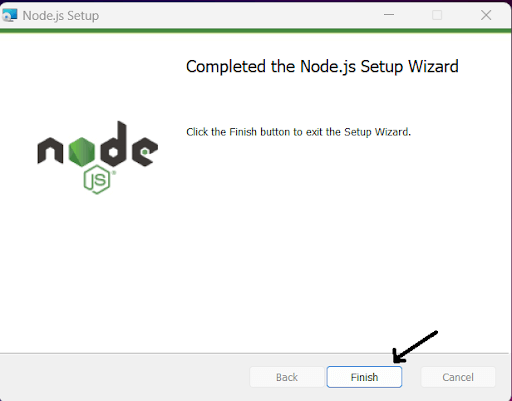
**Step 8:**  Now our NodeJS is ready to install, click ‘*Install*.’



**Step 9:** Going through the installation process.



**Step 10:** After completing the NodeJS installation, click ‘*Finish*.



**Step 11:** Type ‘*node -v*’ on the command prompt to check if the installation is successful. As npm **(Node Package Manager)** also comes with NodeJS, we don’t need to install it explicitly. To check if it’s installed, Type ‘*npm -v*.’

Command: >node -v

Output: v18.16.x

Command: >npm -v

Output: 9.5.x



**How To Write and Run Your First Program in Node.js**

Create a Node.js file named "myfirst.js", and add the following code:

myfirst.js

var http = require('http');

http.createServer(function (req, res) {  
  res.writeHead(200, {'Content-Type': 'text/html'});  
  res.end('Hello World!');  
}).listen(8080);

**Output:** 'Hello World!

Start your command line interface, write node myfirst.js and hit enter:

**C:\Users\hp\OneDrive\Desktop\nodejs>node myfirst.js**

Now, your computer works as a server!

If anyone tries to access your computer on port 8080, they will get a "Hello World!" message in return!

Start your internet browser, and type in the address: [http://localhost:8080](http://localhost:8080/" \t "_blank)

**Output:** **'Hello World!**

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

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

**DESCRIPTION:** HTML, CSS, and JavaScript are essential components for creating a functional responsive web application. A condensed example of a shopping cart with registration, login, catalogue, and cart pages is provided.

**Project Structure:**

1.**index.html:** Main HTML file containing the structure of the web application.

**2. Styles.css:** CSS file for styling the web pages.

**3.Registration Page (registration.html):**

* **Form Elements**: User registration form with fields like name, email, password, and address.

**4. Login Page (login.html)**

* **Form Elements**: Login form with email and password input fields.

**5.Shopping Cart Page (cart.html)**

* Cart Details: Display products added to the cart with quantity, price, and total cost.

**index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<link rel="stylesheet" href="./style.css">

<title>Home - FBS</title>

</head>

<body>

<div class="wrapper">

<div class="container">

<header>

<table width="100%" align="center" cellpadding="0" cellspacing="2">

<tr>

<th width="20%"><img src="fbs.png" alt="FBS LOGO" width="130" height="100"/></th>

<th colspan=4>

<h1 style="color:white;">FBS - WORLD BEST ONLINE EBOOKS WEBSITE</h1>

</th>

</tr>

</table>

</header>

<nav>

<table width="100%" align="center" cellpadding="0" cellspacing="2">

<tbody align="center" style="font-weight:bold;font-size:18px;"">

<tr>

<td width="20%"><hr><a href="index.html">Home</a><hr></td>

<td width="20%"><hr><a href="login.html">Login</a><hr></td>

<td width="20%"><hr><a href="registration.html">Registration</a><hr></td>

<td width="20%"><hr><a href="cart.html" >Cart</a><hr></td>

</tr>

</tbody>

</table>

</nav>

</div>

<div class="container1">

<div class="sidebar1"></div>

<div class="container2">

<main>

<center>

<h2>Welcome to FBS e-Book's Website</h2>

<p>Shopping at <font size=5>FBS</font> can be both <font size=5>fun</font>and <font size=5>savings</font>.</br>Shop with us in this special <fontsize=5>discount</font> season and save upto <font size=5>90%</font> on all your purchases.</br></p>

<br/><br/><br/><br/><br/><br/><br/><br/>

</main>

</div>

<div class="sidebar2"></div>

</div>

<footer><font color="white">(C) 2024 All rights reserved by FBS ebooks</font></footer>

</div>

</body>

</html>

**login.html**

<!DOCTYPE html>

<html lang="en">

<head>

<link rel="stylesheet" href="./style.css">

<title>Login - FBS</title>

</head>

<body>

<div class="wrapper">

<div class="container">

<header>

<table width="100%" align="center" cellpadding="0" cellspacing="2">

<tr>

<th width="20%"><img src="fbs.png" alt="FBS LOGO" width="130" height="100"/></th>

<th colspan=4>

<h1 style="color:white;">FBS - WORLD BEST ONLINE EBOOKS WEBSITE</h1>

</th>

</tr>

</table>

</header>

<nav>

<table width="100%" align="center" cellpadding="0" cellspacing="2">

<tbody align="center" style="font-weight:bold;font-size:18px;"">

<tr>

<td width="20%"><hr><a href="index.html">Home</a><hr></td>

<td width="20%"><hr><a href="login.html">Login</a><hr></td>

<td width="20%"><hr><a href="registration.html">Registration</a><hr></td>

<td width="20%"><hr><a href="cart.html" >Cart</a><hr></td>

</tr>

</tbody>

</table>

</nav>

</div>

<div class="container1">

<div class="sidebar1"></div>

<div class="container2">

<main>

<center><br>

<h3> Login Details</h3> <br/>

<form name="f1">

<table width="100%" align="center" >

<tr>

<td> User Name : </td>

<td> <input type="text" name="username"></td>

</tr>

<tr><td><br></td></tr>

<tr>

<td> Password : </td>

<td> <input type="password" name="password"></td>

</tr>

<tr><td><br></td></tr>

<tr><td></td>

<td><input type="submit" value="SUBMIT">

<input type="reset" value="RESET"></td>

</tr>

</table>

</form>

</center>

</main>

</div>

<div class="sidebar2"></div>

</div>

<footer><font color="white">(C) 2024 All rights reserved by FBS ebooks</font></footer>

</div>

</body>

</html>

**registration.html**

<!DOCTYPE html>

<html lang="en">

<head>

<link rel="stylesheet" href="./style.css">

<title>Registration - FBS</title>

</head>

<body>

<div class="wrapper">

<div class="container">

<header>

<table width="100%" align="center" cellpadding="0" cellspacing="2">

<tr>

<th width="20%"><img src="fbs.png" alt="FBS LOGO" width="130" height="100"/></th>

<th colspan=4>

<h1 style="color:white;">FBS - WORLD BEST ONLINE EBOOKS WEBSITE</h1>

</th>

</tr>

</table>

</header>

<nav>

<table width="100%" align="center" cellpadding="0" cellspacing="2">

<tbody align="center" style="font-weight:bold;font-size:18px;"">

<tr>

<td width="20%"><hr><a href="index.html">Home</a><hr></td>

<td width="20%"><hr><a href="login.html">Login</a><hr></td>

<td width="20%"><hr><a href="registration.html">Registration</a><hr></td>

<td width="20%"><hr><a href="cart.html" >Cart</a><hr></td>

</tr>

</tbody>

</table>

</nav>

</div>

<div class="container1">

<div class="sidebar1"></div>

<div class="container2">

<main>

<center><br>

<h3>Registration Form </h3>

<br/>

<form name="f1">

<table cellpadding="1" align="center" >

<tr><td> Name:\*</td>

<td><input type="text" name="username"></td></tr>

<tr><td>Password:\*</td>

<td><input type="password" name="password"></td></tr>

<tr><td>Email ID:\*</td>

<td><input type="text" name="email"></td></tr>

<tr><td>Phone Number:\*</td>

<td><input type="text" name="phno"></td></tr>

<tr><td valign="top">Gender:\*</td>

<td><input type="radio" name="radio" value="1">Male &nbsp;&nbsp;

<input type="radio" name="radio" value="2">Female</td></tr>

<tr> <td valign="top">Language Known:\*</td>

<td> <input type="checkbox" name="checkbox" value="English">English<br/>

<input type="checkbox" name="checkbox" value="Telugu">Telugu<br>

<input type="checkbox" name="checkbox" value="Hindi">Hindi<br>

<input type="checkbox" name="checkbox" value="Tamil">Tamil

</td></tr>

<tr> <td valign="top">Address:\*</td>

<td><textarea name="address"></textarea></td>

<tr><td></td><td><input type="submit" value="submit" hspace="10">

<input type="reset" value="reset"></td></tr>

<tr> <td colspan=2 >\*<font color="#FF0000">fields are mandatory</font>

</td>

</tr>

</table>

</form>

</center>

</main>

</div>

<div class="sidebar2"></div>

</div>

<footer><font color="white">(C) 2024 All rights reserved by FBS ebooks</font></footer>

</div>

</body>

</html>

**cart.html**

<!DOCTYPE html>

<html lang="en">

<head>

<link rel="stylesheet" href="./style.css">

<title>Cart - FBS</title>

</head>

<body>

<div class="wrapper">

<div class="container">

<header>

<table width="100%" align="center" cellpadding="0" cellspacing="2">

<tr>

<th width="20%"><img src="fbs.png" alt="FBS LOGO" width="130" height="100"/></th>

<th colspan=4>

<h1 style="color:white;">FBS - WORLD BEST ONLINE EBOOKS WEBSITE</h1>

</th>

</tr>

</table>

</header>

<nav>

<table width="100%" align="center" cellpadding="0" cellspacing="2">

<tbody align="center" style="font-weight:bold;font-size:18px;"">

<tr>

<td width="20%"><hr><a href="index.html">Home</a><hr></td>

<td width="20%"><hr><a href="login.html">Login</a><hr></td>

<td width="20%"><hr><a href="registration.html">Registration</a><hr></td>

<td width="20%"><hr><a href="cart.html" >Cart</a><hr></td>

</tr>

</tbody>

</table>

</nav>

</div>

<div class="container1">

<div class="sidebar1"></div>

<div class="container2">

<main>

<center>

<h3>Cart</h3>

<table width="100%" align="center" >

<tbody>

<tr>

<th width="40%"><hr>BookName<hr></th>

<th width="20%"><hr>Price<hr></th>

<th width="20%"><hr>Quantity<hr></th>

<th width="20%"><hr>Amount<hr></th> </tr>

</tbody>

<tbody align=center>

<tr> <td>Java Programming </td>

<td>Rs. 2300/-</td>

<td>2</td>

<td>Rs. 4600/-</td></tr>

<tr><td>Web Technologies</td>

<td>Rs. 3000/-</td>

<td>1</td>

<td>Rs. 3000/-</td></tr>

<tr><td></td>

<td><hr><font color="#996600">Total Amount:</font><hr></td>

<td><hr>3<hr></td>

<td><hr>Rs. 7600/-<hr></td> </tr>

</tbody>

</table>

</center>

</main>

</div>

<div class="sidebar2"></div>

</div>

<footer><font color="white">(C) 2024 All rights reserved by FBS ebooks</font></footer>

</div>

</body>

</html>

**style.css**

body{

font-family: monospace;

}

main {

background-color: #efefef;

color: #330000;

margin-left: 10px;

height: 60vh;

}

header, footer {

background-color: #000d57;

color: #fff;

padding: 1rem;

height: 50px;

}

header, nav{

margin-bottom: 10px;

flex-basis: 50%;

}

footer{

margin-top: 10px;

}

nav {

background-color: #fff;

color: #000;

padding: 1rem;

height: 20px;

}

.sidebar1, .sidebar2 {

flex-basis: 10%;

background-color: #fff;

color: #000;

}

.sidebar2{

margin-left: 10px;

}

.container1{

display: flex;

}

.container2 {

display: flex;

flex-direction: column;

flex: 1;

}

header, nav, main, .sidebar1, .sidebar2, footer{

display: flex;

align-items: center;

justify-content: center;

border-radius: 10px;

}

.wrapper {

display: flex;

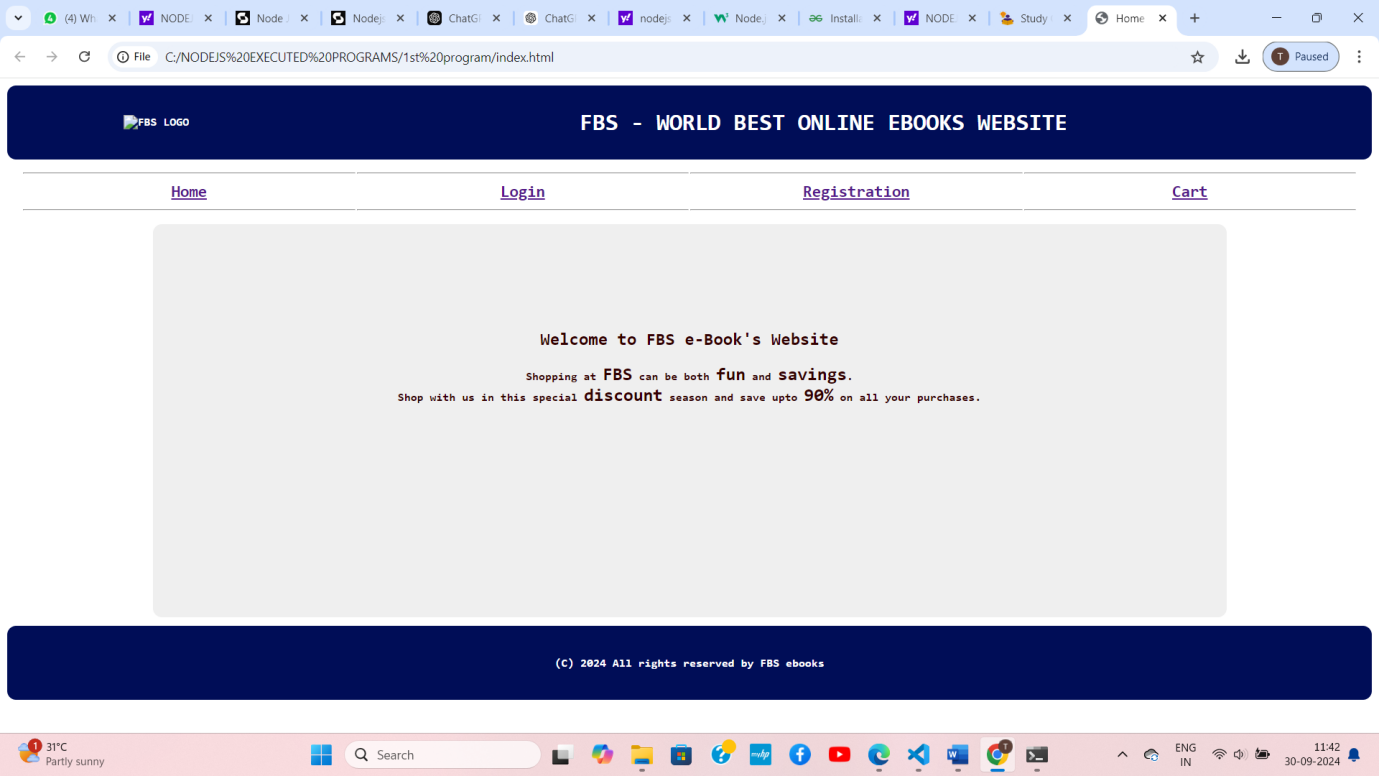
flex-direction: column;

font-weight: 600;

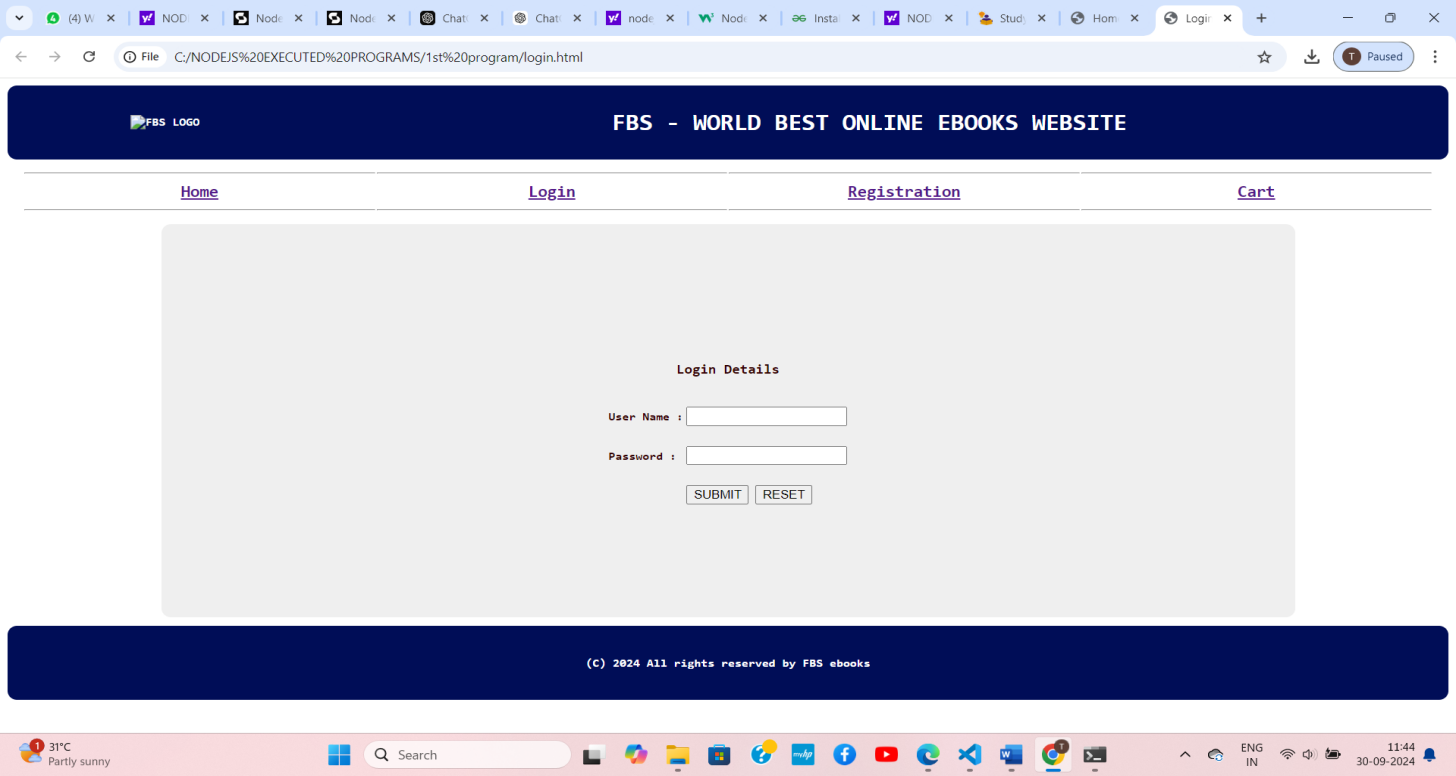
}

**Output :**

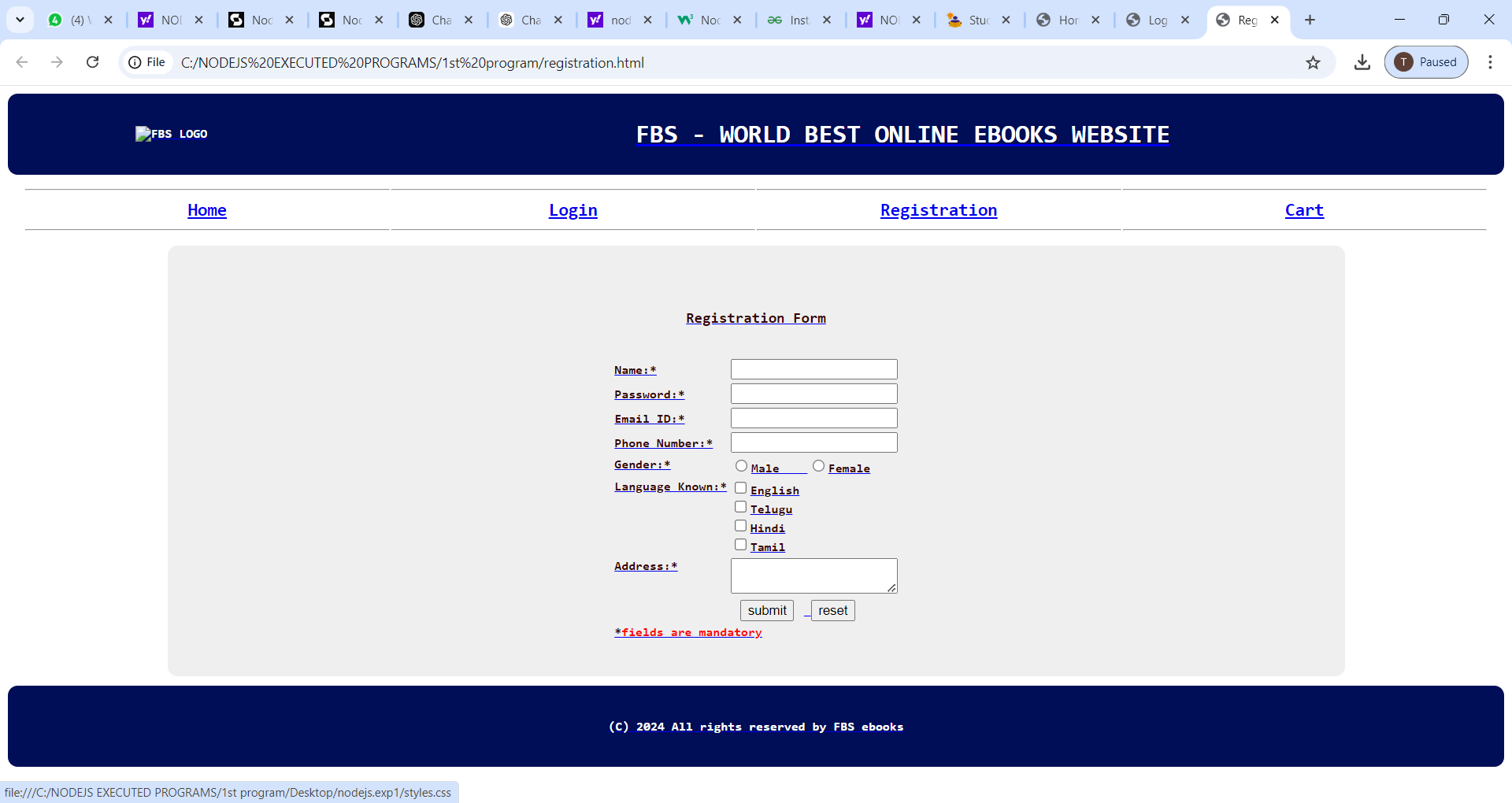
**index.html**



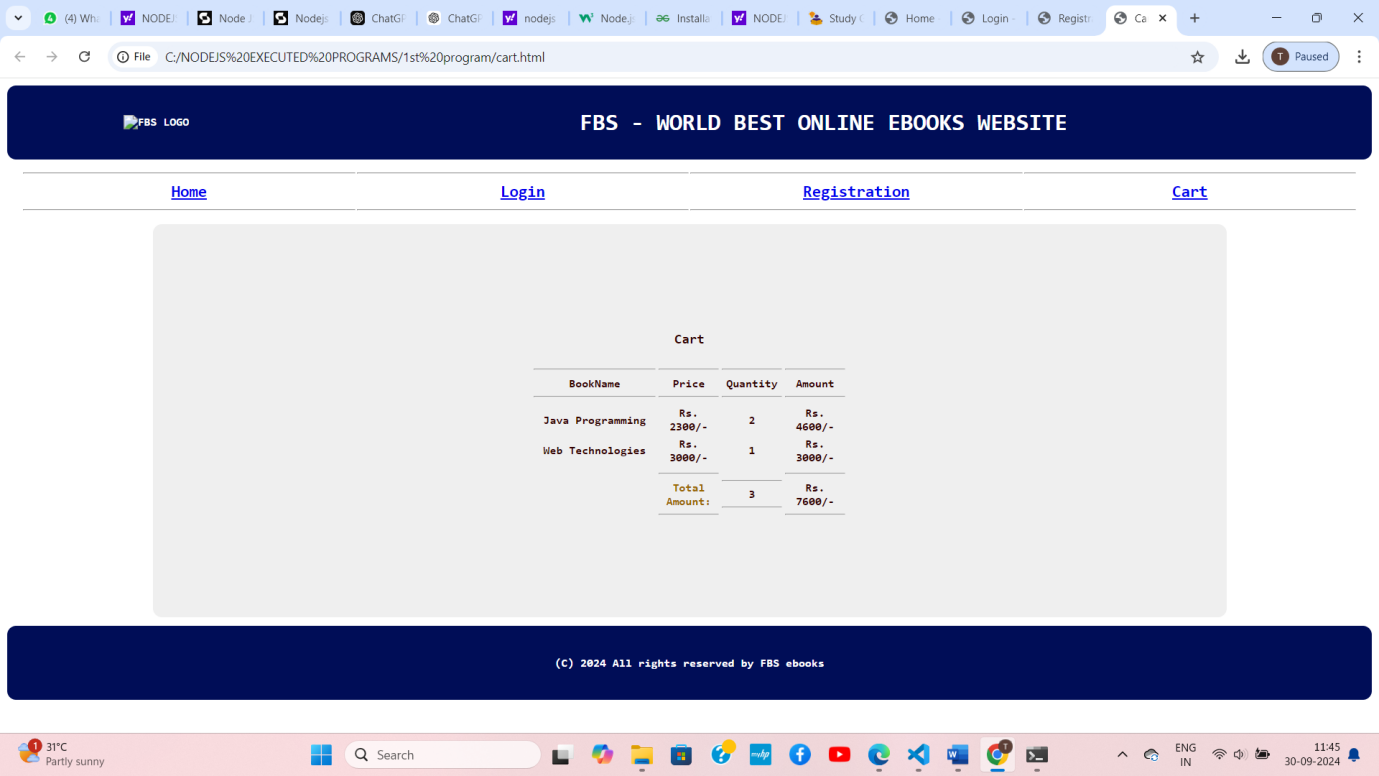
**login.html**



**registration.html**

****

**cart.html**



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

**AIM:** Use JavaScript for doing client – side validation of the pages implemented in

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

**DESCRIPTION:** To perform client-side validation using JavaScript, you can add scripts to validate user inputs on the registration page.

**registrationJS.html**

<html>

<head>

<title> Welcome to NNRG e-Book's website</title>

<script language="javascript">

function validate() {

// username validation

var uname = f1.username.value;

if (uname.length<=0)

{

alert("Please Enter UserName");

f1.username.focus();

return false;

}

if (uname.length < 8)

{

alert("Please enter UserName not less than 8");

f1.username.focus();

return false;

}

//password validation

var pwd = f1.password.value;

if (pwd.length<=0)

{

alert("Please Enter password");

f1.password.focus();

return false;

}

if (pwd.length < 6)

{

alert("Please enter Password not less than 6");

f1.password.focus();

return false;

}

// email validation

var email = f1.email.value;

if (email.length<=0)

{

alert("Please Enter email");

f1.email.focus();

return false;

}

else {

let eflag=false;

for(i=0;i<email.length;i++) {

if(email.charAt(i)=="@")

{

eflag=true;

}

}

if(!(eflag))

{

alert("Please enter a valid Email ID");

f1.email.focus();

return false;

}

}

// phone number validation

var phno = f1.phno.value;

if (phno.length<=0)

{

alert("Please Enter Phone Number");

f1.phno.focus();

return false;

}

if (isNaN(phno))

{

alert("Please Enter Valid Phone Number");

f1.phno.focus();

return false;

}

if (phno.length != 10)

{

alert("Please Enter Valid Phone Number");

f1.phno.focus();

return false;

}

// gender validation

let flag=false;

for(i=0;i<f1.gen.length;i++)

if(f1.gen[i].checked)

flag=true;

if(!(flag))

{

alert("Please choose a Gender");

return false;

}

// Language validation

flag=false;

for(i=0;i<f1.lang.length;i++)

if(f1.lang[i].checked)

flag=true;

if(!(flag))

{

alert("Please select at least one of the Language options.");

return false;

}

// address validation

var addr = f1.address.value;

if (addr.length<=0)

{

alert("Please Enter address");

f1.address.focus();

return false;

}

// to display Success message

alert("Registration Successful");

}

</script>

</head>

<body>

<center><br>

<h3>Registration Form </h3>

<br/>

<form name="f1">

<table cellpadding="1" align="center" >

<tr><td> User Name:\*</td>

<td><input type="text" name="username"></td></tr>

<tr><td>Password:\*</td>

<td><input type="password" name="password"></td></tr>

<tr><td>Email ID:\*</td>

<td><input type="text" name="email"></td></tr>

<tr><td>Phone Number:\*</td>

<td><input type="text" name="phno"></td></tr>

<tr><td valign="top">Gender:\*</td>

<td><input type="radio" name="gen" value="Male">Male &nbsp;&nbsp;

<input type="radio" name="gen" value="Female">Female</td></tr>

<tr> <td valign="top">Language Known:\*</td>

<td> <input type="checkbox" name="lang" value="English">English<br/>

<input type="checkbox" name="lang" value="Telugu">Telugu<br>

<input type="checkbox" name="lang" value="Hindi">Hindi<br>

<input type="checkbox" name="lang" value="Tamil">Tamil

</td></tr>

<tr> <td valign="top">Address:\*</td>

<td><textarea name="address"></textarea></td>

<tr><td></td><td><input type="button" value="SUBMIT" hspace="10" onclick="validate()">

<input type="reset" value="RESET"></td></tr>

<tr> <td colspan=2 >\*<font color="#FF0000">fields are mandatory</font>

</td>

</tr>

</table>

</form>

</center>

</body>

</html>

**OUTPUT:**

**Client - Side validation of Registration Page**



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

**AIM:** Create an xml for the bookstore. Validate the same using both DTD and XSD **DESCRIPTION:** Let's create an XML file for a simple bookstore and validate it using both Document Type Definition (DTD) and XML Schema Definition (XSD).

**Explanation:**

1. **XML File (bookstore.xml):** Represents a simple XML structure with a list of books in a bookstore.

2. **DTD** File (bookstore.dtd): Describes the structure of the XML document using Document Type Definition. Specifies that a bookstore must contain one or more book elements. Each book must contain title, author, and price elements.

**3. XSD File (bookstore.xsd):** Describes the structure of the XML document using XML Schema Definition. Defines complex types for bookstore and book. Specifies that a bookstore must contain an unbounded sequence of book elements. Each book must contain title (string), author (string), and price (decimal) elements.

**Validation:** You can validate the XML file using a tool or programming language that supports DTD and XSD validation.

**Bookstore XML File (bookstore.xml):**

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE bookstore[

<!ELEMENT bookstore (book+)>

<!ELEMENT book (title, author, price)>

<!ELEMENT title (#PCDATA)>

<!ELEMENT author (#PCDATA)>

<!ELEMENT price (#PCDATA)>

]>

<bookstore>

<book>

<title>Introduction to XML</title>

<author>John Doe</author>

<price>29.99</price>

</book>

<book>

<title>Programming with XML</title>

<author>Jane Smith</author>

<price>39.99</price>

</book>

</bookstore>

**XSD File (bookstore.xsd):**

<?xml version="1.0" encoding="UTF-8"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"

targetNamespace="http://example.com"

xmlns="http://example.com">

<xs:element name="root">

<xs:complexType>

<xs:sequence>

<xs:element name="bookstore" type="bookstoreType"/>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:complexType name="bookstoreType">

<xs:sequence>

<xs:element name="book" type="bookType" minOccurs="0" 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="price" type="xs:decimal"/>

</xs:sequence>

</xs:complexType>

</xs:schema>

**OUTPUT:**

To Check the Validity :

Go to the below link,

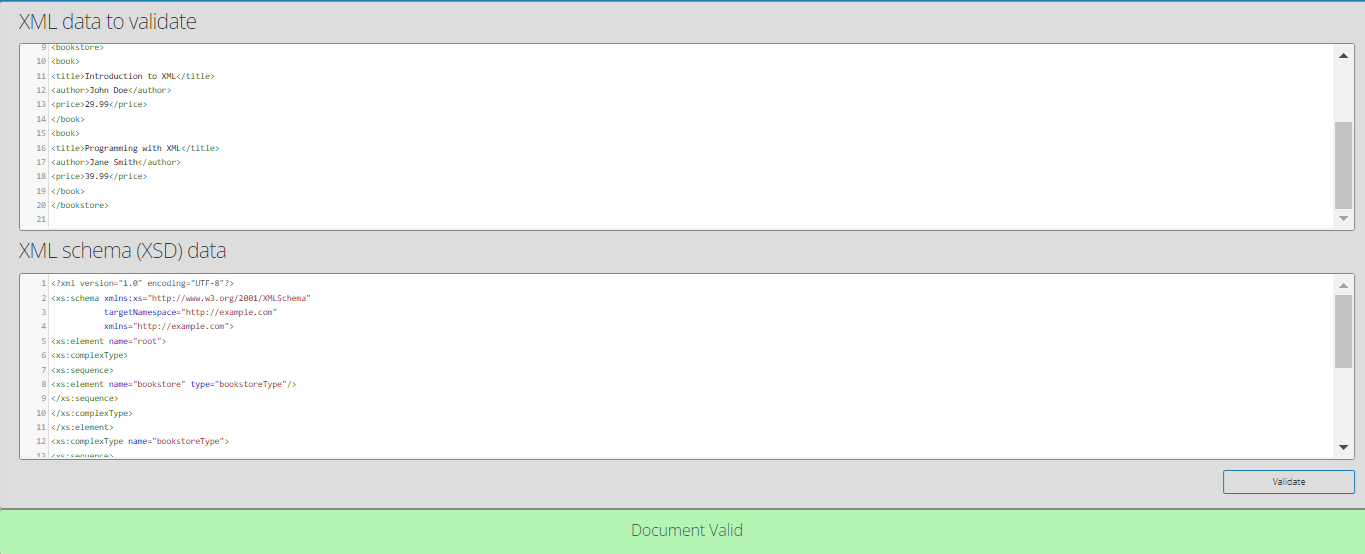
https://www.liquid-technologies.com/online-xsd-validator

Place the XML code in the XML Validate.

Place the XSD code in the XML Schema Data.

Then click the validate Button.

Then it will show the Document as Valid.

****

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

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

**DESCRIPTION:** Let's create a simple custom server using the http module in Node.js and then explore the os, path, and events.

**Explanation:**

* **Open Terminal or Command Prompt:**

Open a terminal or command prompt in the directory where you saved your **server.js** file.

* **Run the Server Script:**

Execute the server script using the Node.js runtime. In the terminal, run:

node server.js

This will start the HTTP server, and you should see the message "Server running at <http://127.0.0.1:3000/>".

* **Access the Server:**

Open your web browser and navigate to http://127.0.0.1:3000/ or http://localhost:3000/. You should see the response "**Hello, World!**".

* **Check OS Information:**

In the same terminal where your server is running, you'll see information about your operating system (OS) type, platform, architecture, CPU cores, etc.

* **Check Current Working Directory:**

The current working directory of the script is printed in the terminal.

* **Check Joined Path:**

The joined path using the path module is printed in the terminal.

* **Check Custom Event:**

The script emits a custom event and listens for it. In the terminal, you should see the message "Custom Event Triggered: { message: 'Hello from custom event!' }".

* **Stop the Server:**

To stop the server, press Ctrl+C in the terminal where the server is running.

**server.js**

// **Step 1**: Import required modules

const http = require('http');

const os = require('os');

const path = require('path');

const { EventEmitter } = require('events');

// **Step 2**: Create an instance of EventEmitter

const eventEmitter = new EventEmitter();

// **Step 3**: Create a simple HTTP server

const server = http.createServer((req, res) => {

res.writeHead(200, { 'Content-Type':'text/plain' });

res.end('Hello, World!\n');

});

// **Step 4**: Define server port and hostname

constport = 3000;

consthostname = '127.0.0.1';

// **Step 5**: Listen for requests on the specified port and hostname

server.listen(constport, consthostname, () => {

console.log('Server running at http://${127.0.0.1}:${3000}/');

});

// **Step 6**: Print OS information

console.log('OS Type:', os.type());

console.log('OS Platform:', os.platform());

console.log('OS Architecture:', os.arch());

console.log('CPU Cores:', os.cpus().length);

// **Step 7**: Print current working directory

console.log('Current Working Directory:', process.cwd());

// **Step 8**: Join paths using the path module

const joinedPath = path.join(\_\_dirname, 'public', 'images');

console.log('Joined Path:', joinedPath);

// **Step 9**: Handle a custom event

eventEmitter.on('customEvent', (data) => {

console.log('Custom Event Triggered:', data);

});

// **Step 10**: Emit a custom event

eventEmitter.emit('customEvent', { message:'Hello from custom event!' });

**OUTPUT:**

C:\ NODEJSPROGRAMS >node server.js

OS Type: Windows\_NT

OS Platform: win32

OS Architecture: x64

CPU Cores: 12

Current Working Directory: C:\NODEJSPROGRAMS

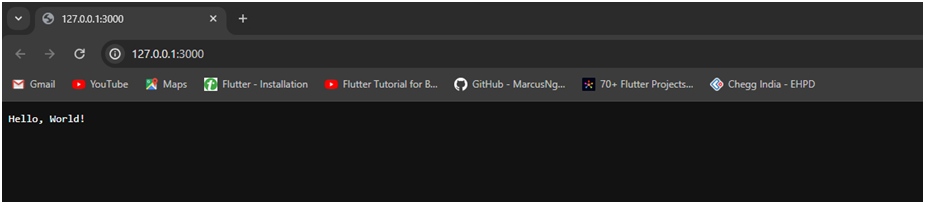
Joined Path: C:\ NODEJS PROGRAMS public\images

Custom Event Triggered: { message: 'Hello from custom event!' }

Server running at **<http://127.0.0.1:3000/>**

**In the Browser:**

**Link: <http://127.0.0.1:3000/>**



**5.Explore the features of ES6 like arrow functions, callbacks, 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.**

**AIM:** Explore the features of ES6 like arrow functions, callbacks, 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.

**DESCRIPTION:** To implement an application for reading weather information from OpenWeatherMap.org and displaying the information in the form of a graph, we can use JavaScript with ES6 features like arrow functions, callbacks, promises, and async/await.

**Explanation of ES6 Features**

* **Arrow Functions:** The fetchWeather function and prepareChartData are implemented as arrow functions for cleaner syntax.
* **Async/Await:** Used in fetchWeather to fetch the weather data asynchronously and await the API response.
* **Callbacks:** The prepareChartData function is a callback that transforms the API response into a format suitable for the chart.
* **Promises:** Axios returns a promise, and we handle it using async/await to keep the code clean.

**Project Structure:**

1. **index.html** – Main HTML file.

2. **script.js** - JavaScript file for handling weather data and graph creation.

3. **styles.css** - CSS file for styling.

4. **node\_modules**/ - Folder for library dependencies.

**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 Dashboard</title>

    <link rel="stylesheet" href="style.css">

</head>

<body>

    <div class="container">

        <h1>Weather Dashboard</h1>

        <select id="citySelect">

            <option value="London">London</option>

            <option value="Mumbai">Mumbai</option>

            <option value="Los Angeles">Los Angeles</option>

            <option value="Paris">Paris</option>

            <option value="Tokyo">Tokyo</option>

            <option value="Hyderabad">Hyderabad</option> <!-- Added Hyderabad -->

        </select>

        <div id="weatherInfo">

            <h2>Weather Information</h2>

            <p id="description"></p>

            <p id="humidity"></p>

        </div>

        <canvas id="weatherChart" width="800" height="400"></canvas>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

    <script src="app.js"></script>

</body>

</html>

**App.js**

const API\_KEY = '73faeaa17fde308d431a5bdf83579e00'; // Replace with your OpenWeatherMap API key

let currentCity = 'London'; // Default city

const fetchWeatherData = async () => {

    const API\_URL = `https://api.openweathermap.org/data/2.5/forecast?q=${currentCity}&units=metric&appid=${API\_KEY}`;

    try {

        const response = await fetch(API\_URL);

        if (!response.ok) {

            throw new Error(`Network response was not ok: ${response.statusText}`);

        }

        const data = await response.json();

        return data;

    } catch (error) {

        console.error('Fetch error:', error);

    }

};

const displayWeatherData = async () => {

    const data = await fetchWeatherData();

    if (!data) return;

    // Extract and format data

    const dates = data.list.map(item => new Date(item.dt\_txt).toLocaleDateString());

    const temperatures = data.list.map(item => item.main.temp);

    const description = data.list[0].weather[0].description; // Weather description

    const humidity = data.list[0].main.humidity; // Humidity

    // Display additional information

  document.getElementById('description').textContent = `Weather Description: ${description}`;

    document.getElementById('humidity').textContent = `Humidity: ${humidity}%`;

    // Create the chart

    const ctx = document.getElementById('weatherChart').getContext('2d');

    new Chart(ctx, {

        type: 'line',

        data: {

            labels: dates,

            datasets: [{

                label: 'Temperature (°C)',

                data: temperatures,

                borderColor: 'rgba(75, 192, 192, 1)',

                backgroundColor: 'rgba(75, 192, 192, 0.2)',

                borderWidth: 1

            }]

        },

        options: {

            responsive: true,

            scales: {

                x: {

                    title: {

                        display: true,

                        text: 'Date'

                    }

                },

                y: {

                    title: {

                        display: true,

                        text: 'Temperature (°C)'

                    }

                }

            }

        }

    });

};

const handleCityChange = () => {

    currentCity = document.getElementById('citySelect').value;

    displayWeatherData();

};

// Initialize the app and set up event listener

document.getElementById('citySelect').addEventListener('change', handleCityChange);

// Display weather data for the default city on page load

displayWeatherData();

**Style.css:**

body {

    font-family: Arial, sans-serif;

    margin: 0;

    padding: 0;

    background-color: #a8dadc;

    display: flex;

    justify-content: center;

    align-items: center;

    height: 100vh;

}

.container {

    text-align: center;

    background: #f1faee;

    padding: 20px;

    border-radius: 8px;

    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

}

h1 {

    margin-bottom: 20px;

}

#weatherInfo {

    margin-bottom: 20px;

}

select {

    margin-bottom: 20px;

    border-radius: 0.5rem;

    border:none;

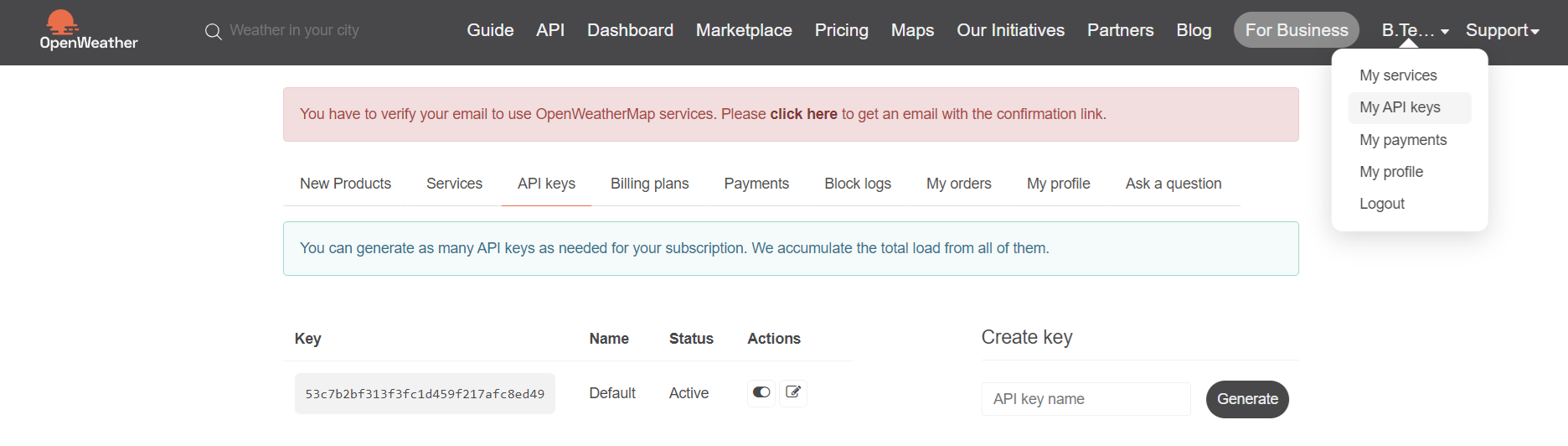
    outline:none;

    background-color:#f1faee ;

}

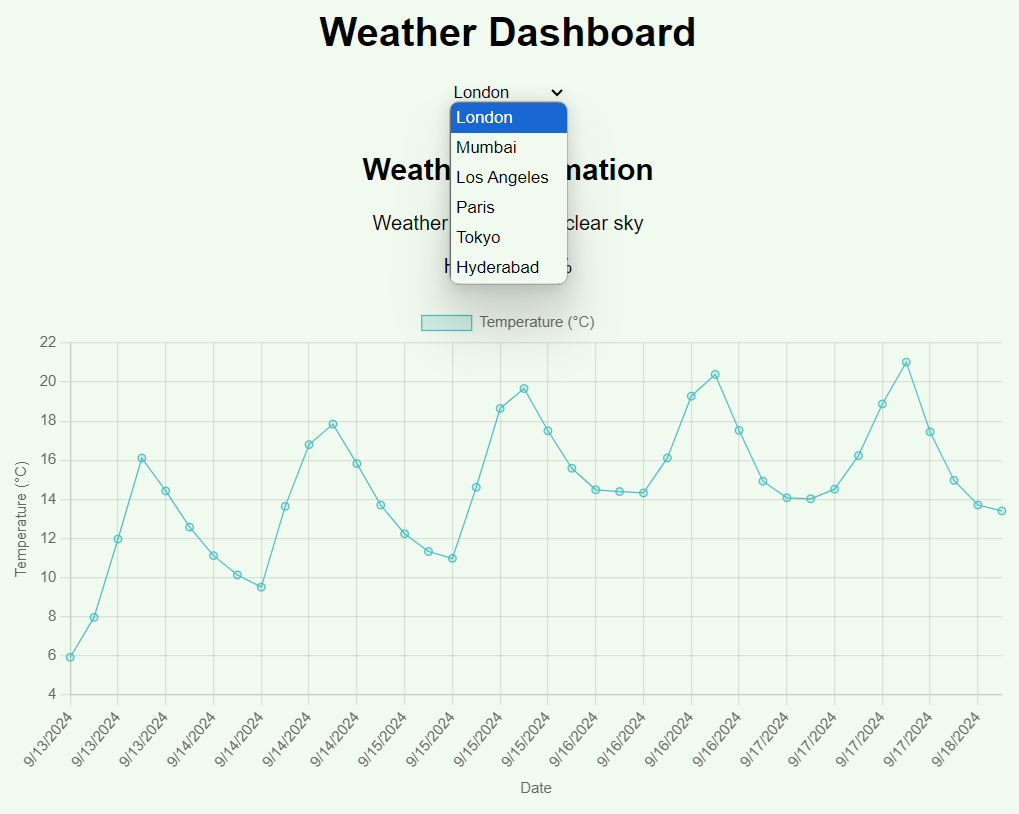
**Create an OpenWeatherMap Account and Generate API Key**

1. Visit the OpenWeatherMap website (https://openweathermap.org/) and click on "Sign Up" or "Log In" to create an account or log into your existing account.
2. Once logged in, navigate to your account dashboard.
3. From the dashboard, locate my API Keys section and click on "Create Key" or "API Keys" to generate a new API key.
4. Provide a name for your API key (e.g., "WeatherApp") and click on the "Generate" or "Create" button.
5. Your API key will be generated and displayed on the screen. Make sure to copy it as we will need it later.

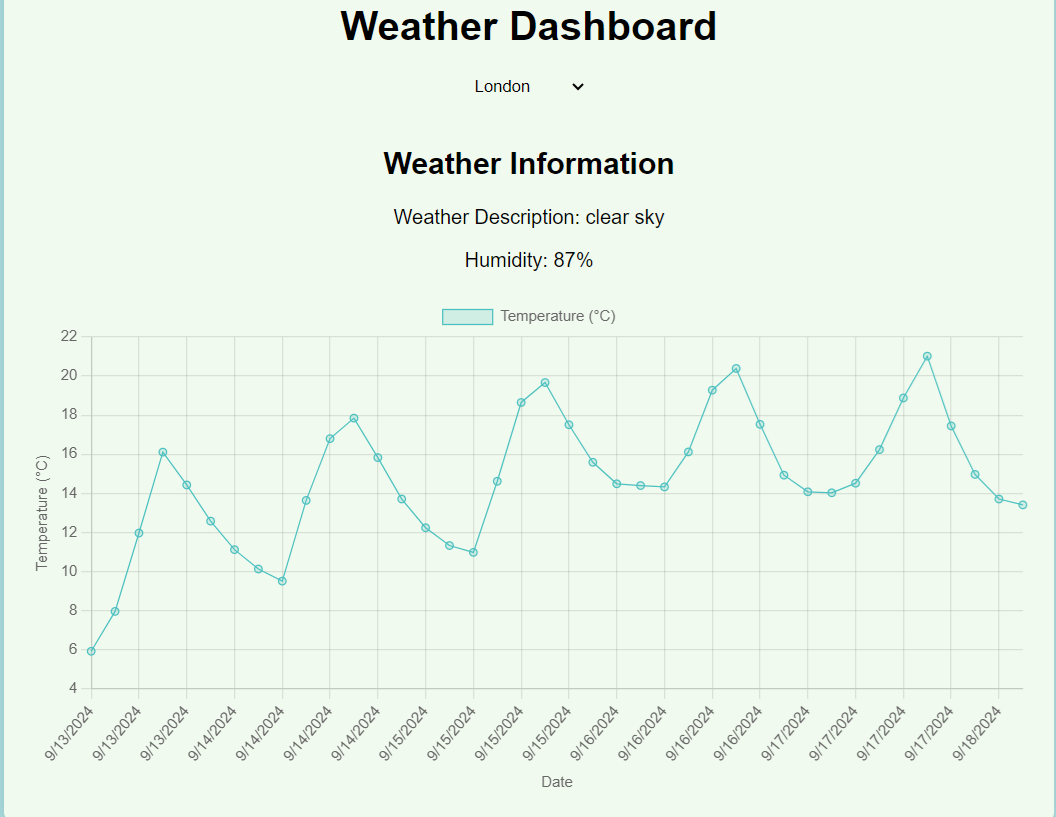


Locate API key

Initially It look Like:



Then, by entering the city then click update chart then



**6.Develop an express [web application](https://studyglance.in/labprograms/nodejsdisplay.php?url1=nodejs/exp7.php&title=Develop%20an%20express%20web%20application%20that%20can%20interact%20with%20REST%20API%20to%20perform%20CRUD%20operations%20on%20student%20data.%20(Use%20Postman)) that can interact with REST [API](https://studyglance.in/labprograms/nodejsdisplay.php?url1=nodejs/exp7.php&title=Develop%20an%20express%20web%20application%20that%20can%20interact%20with%20REST%20API%20to%20perform%20CRUD%20operations%20on%20student%20data.%20(Use%20Postman)) to perform CRUD operations on student data. (Use Postman).**

**AIM:** To develop an express [web application](https://studyglance.in/labprograms/nodejsdisplay.php?url1=nodejs/exp7.php&title=Develop%20an%20express%20web%20application%20that%20can%20interact%20with%20REST%20API%20to%20perform%20CRUD%20operations%20on%20student%20data.%20(Use%20Postman)) that can interact with REST [API](https://studyglance.in/labprograms/nodejsdisplay.php?url1=nodejs/exp7.php&title=Develop%20an%20express%20web%20application%20that%20can%20interact%20with%20REST%20API%20to%20perform%20CRUD%20operations%20on%20student%20data.%20(Use%20Postman)) to perform CRUD operations on student data. (Use Postman).

**Description:** To develop an Express web application that interacts with a REST API to perform CRUD (Create, Read, Update, Delete) operations on student data, we'll follow these steps:

* 1. **Set Up the Project:**

Firstly, we need to create a new folder and open the folder in the command prompt and enter a command as below:

**npm init -y**

Then, install the necessary packages for our Express application:

**npm install express sqlite3**

Then create file named as the app.js and db.js.

**db.js**

const sqlite3 = require('sqlite3').verbose();

// Function to initialize the database schema

function initializeDatabase() {

const db = new sqlite3.Database('./mydatabase.db', (err) => {

if (err) {

console.error(err.message);

} else {

console.log('Connected to the SQLite database.');

createStudentsTable(db);

}

});

// Close the database connection when the Node process exits

process.on('exit', () => {

db.close((err) => {

if (err) {

console.error(err.message);

} else {

console.log('Disconnected from the SQLite database.');

}

});

});

return db; // Return the db instance

}

// Function to create the 'students' table if it doesn't exist

function createStudentsTable(db) {

const createTableQuery = `

CREATE TABLE IF NOT EXISTS students (

id INTEGER PRIMARY KEY AUTOINCREMENT,

name TEXT,

age INTEGER,

grade TEXT

);

`;

db.run(createTableQuery, (err) => {

if (err) {

console.error(err.message);

} else {

console.log('The students table has been created or already exists.');

}

});

}

module.exports = { initializeDatabase };

**app.js**

const express = require('express');

const { initializeDatabase } = require('./db');

const app = express();

const port = 3000;

// Initialize the database and get the db instance

const db = initializeDatabase();

// Middleware to parse request body as JSON

app.use(express.json());

app.get('/', (req, res) => {

res.send('Welcome to the Student API');

});

// Get all students

app.get('/students', (req, res) => {

db.all('SELECT \* FROM students', [], (err, rows) => {

if (err) {

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

}

res.json(rows);

});

});

// Get a single student by ID

app.get('/students/:id', (req, res) => {

const id = req.params.id;

db.get('SELECT \* FROM students WHERE id = ?', [id], (err, row) => {

if (err) {

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

}

if (!row) {

return res.status(404).json({ error: 'Student not found' });

}

res.json(row);

});

});

// Create a new student

app.post('/students', (req, res) => {

const { name, age, grade } = req.body;

db.run('INSERT INTO students (name, age, grade) VALUES (?, ?, ?)', [name, age, grade], function (err) {

if (err) {

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

}

res.status(201).json({ id: this.lastID });

});

});

// Update a student

app.put('/students/:id', (req, res) => {

const id = req.params.id;

const { name, age, grade } = req.body;

db.run('UPDATE students SET name = ?, age = ?, grade = ? WHERE id = ?', [name, age, grade, id], function (err) {

if (err) {

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

}

if (this.changes === 0) {

return res.status(404).json({ error: 'Student not found' });

}

res.json({ updatedID: id });

});

});

// Delete a student

app.delete('/students/:id', (req, res) => {

const id = req.params.id;

db.run('DELETE FROM students WHERE id = ?', [id], function (err) {

if (err) {

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

}

if (this.changes === 0) {

return res.status(404).json({ error: 'Student not found' });

}

res.json({ deletedID: id });

});

});

app.listen(port, () => {

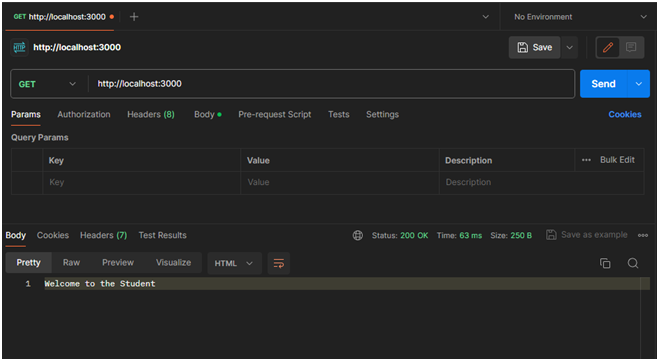
console.log(`Server running at http://localhost:${port}`);

});

**OUTPUT:**

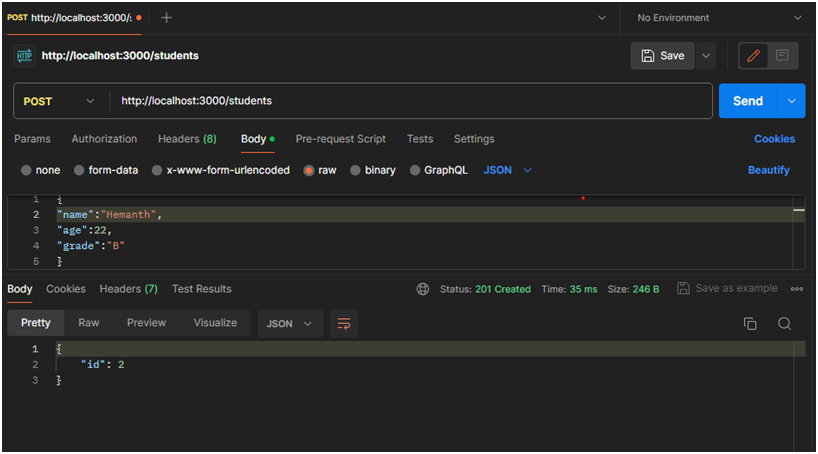
**GET:**

* Open Postman.
* Set the request type to GET.
* Enter the URL: *<http://localhost:3000/students>*.



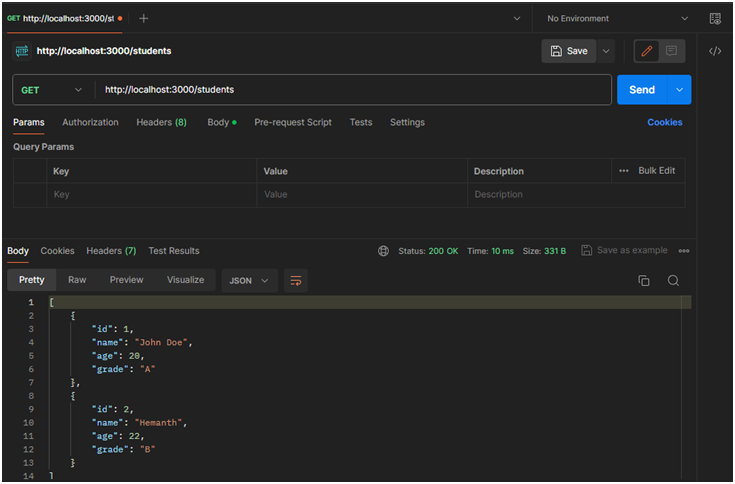
**POST: Create a New Student**

* Open Postman.
* Set the request type to POST.
* Enter the URL: *http://localhost:3000/students.*
* Go to the "Body" tab.
* Select raw and set the body to JSON format.



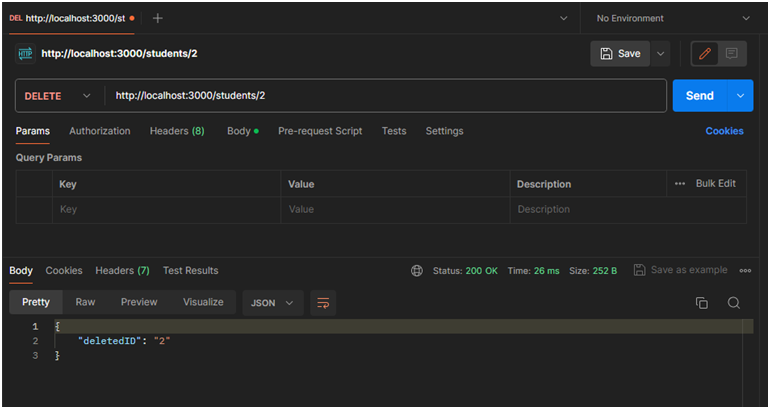
**GET: #all Students**

* Set the request type to GET.
* Enter the URL: *http://localhost:3000/students.*
* Click on the "Send" button
* You should receive a response with details of all students in your SQLite database.



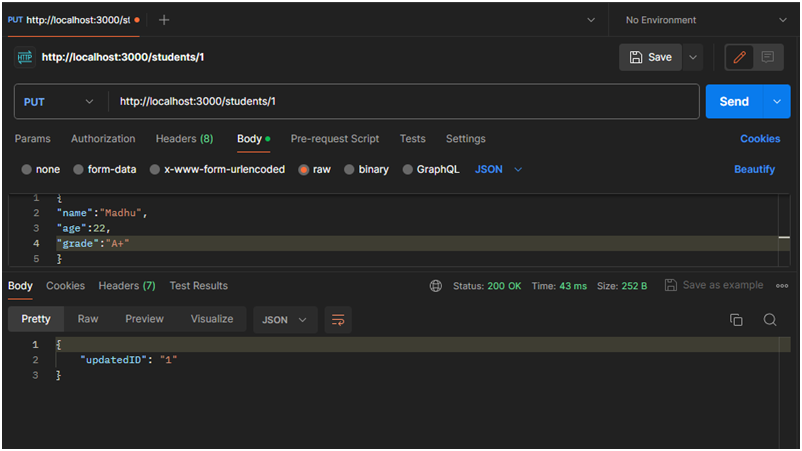
**DELETE:**

* Set the request type to DELETE.
* Enter the URL for the student you want to delete (replace: id with an actual student ID): *http://localhost:3000/students/:id*
* Place instead of ID which replace with number that is ID to be deleted.
* Then click send



**PUT:**

* Set the request type to PUT.
* Enter the URL for the student you want to delete (replace: id with an actual student ID): *http://localhost:3000/students/:id*
* Go to the "Body" tab.
* Select raw and set the body to JSON format



**7.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 using chart.js.**

**Aim:** To 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 using chart.js, follow these steps:

**Step1.Set up React project**

**Open a terminal and create a new React app**

* Run npx create-react-app weather-app

**Navigate to the project directory:**

* cd weather-app

**Install required dependencies:**

* npm install axios chart.js react-chartjs-2

**Step 2.Get OpenWeatherMap API key**

Sign up at OpenWeatherMap and generate an API key**.**

**Step 3.** **Create folder structure**

**weather-app/**

**├── src/**

**│ ├── components**

**│ ├──** Weather.js

**│ ├── services/**

**│ │ └──** weatherService.js

**│ ├── App.js**

**Step 4.Create Components**

* In src/components/Weather.js:

**Weather.js**

import React, { useState } from 'react';

import { fetchWeather } from '../services/weatherService'; // Adjust the path

import { Line } from 'react-chartjs-2';

import { Chart, registerables } from 'chart.js';

// Register the required components for Chart.js

Chart.register(...registerables);

const Weather = () => {

const [city, setCity] = useState('');

const [weather, setWeather] = useState(null);

const [error, setError] = useState('');

const getWeather = async () => {

try {

const data = await fetchWeather(city);

setWeather(data);

setError('');

} catch (error) {

setError('Error fetching weather data');

setWeather(null);

}

};

const temperatureData = {

labels: ['Today', 'Tomorrow', 'Day 3', 'Day 4', 'Day 5'], // Replace with actual dates

datasets: [

{

label: 'Temperature (°C)',

data: [weather ? weather.main.temp : 0, 22, 21, 24, 23], // Replace with actual data if available

borderColor: 'rgba(75, 192, 192, 1)',

backgroundColor: 'rgba(75, 192, 192, 0.2)',

fill: true,

},

],

};

return (

<div className="container mt-5">

<h1 className="text-center">Weather App</h1>

<input

type="text"

className="form-control mb-3"

placeholder="Enter city name"

value={city}

onChange={(e) => setCity(e.target.value)}

/>

<button className="btn btn-primary" onClick={getWeather}>Get Weather</button>

{error && <div className="text-danger">{error}</div>}

{weather && (

<div className="mt-4">

<h2>{weather.name}</h2>

<p>Temperature: {weather.main.temp}°C</p>

<Line data={temperatureData} />

</div>

)}

</div>

);

};

export default Weather;

**Step 5. Create Weather Service In src/services/weatherService.js, create the weather API calls:**

**services/weatherService.js:**

**weatherService.js**

import axios from 'axios';

const API\_KEY = 'f1819733a3466d68cf09a867930346b6'; // Replace with your actual API key

const BASE\_URL = 'https://api.openweathermap.org/data/2.5/weather';

export const fetchWeather = async (city) => {

const response = await axios.get(`${BASE\_URL}?q=${city}&appid=${API\_KEY}&units=metric`);

return response.data;

};

**Step 6.Update App.js to Include the Components In src/App.js**

**src/App.js:**

**App.js**

import React from 'react';

import Weather from './components/Weather';

const App = () => {

return (

<div>

<Weather />

</div>

);

};

export default App;

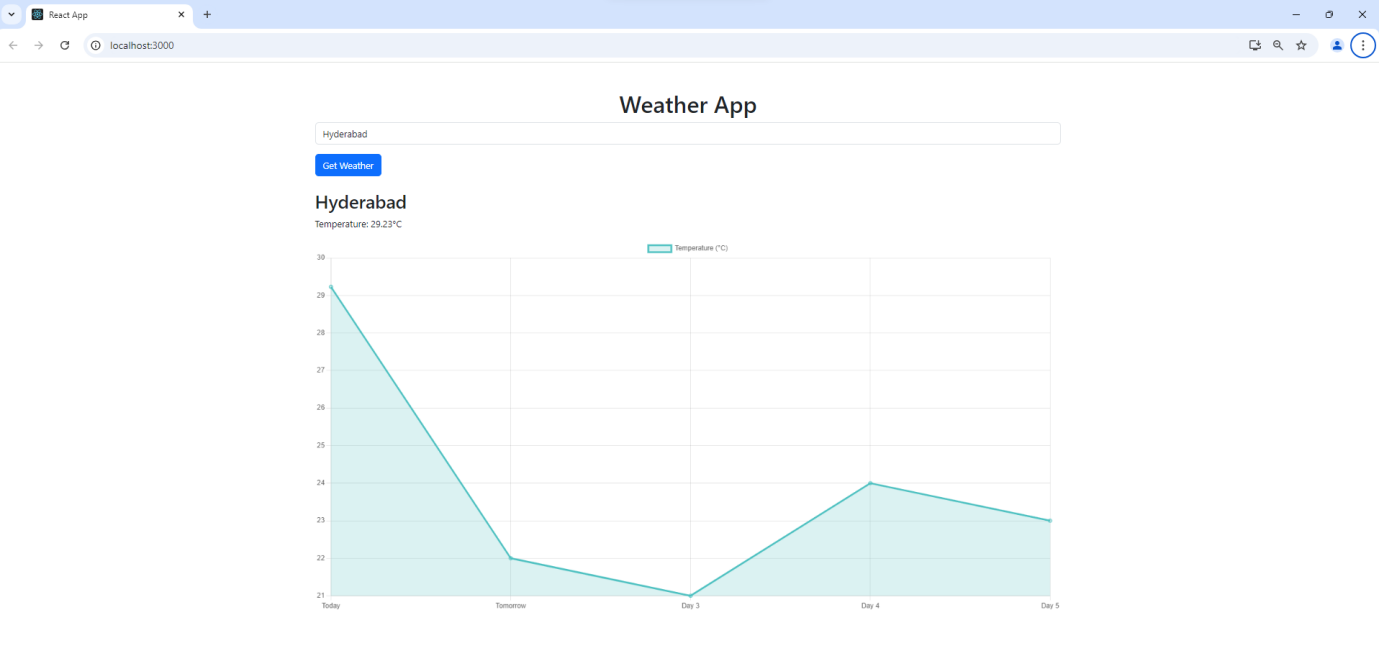
Run the Application

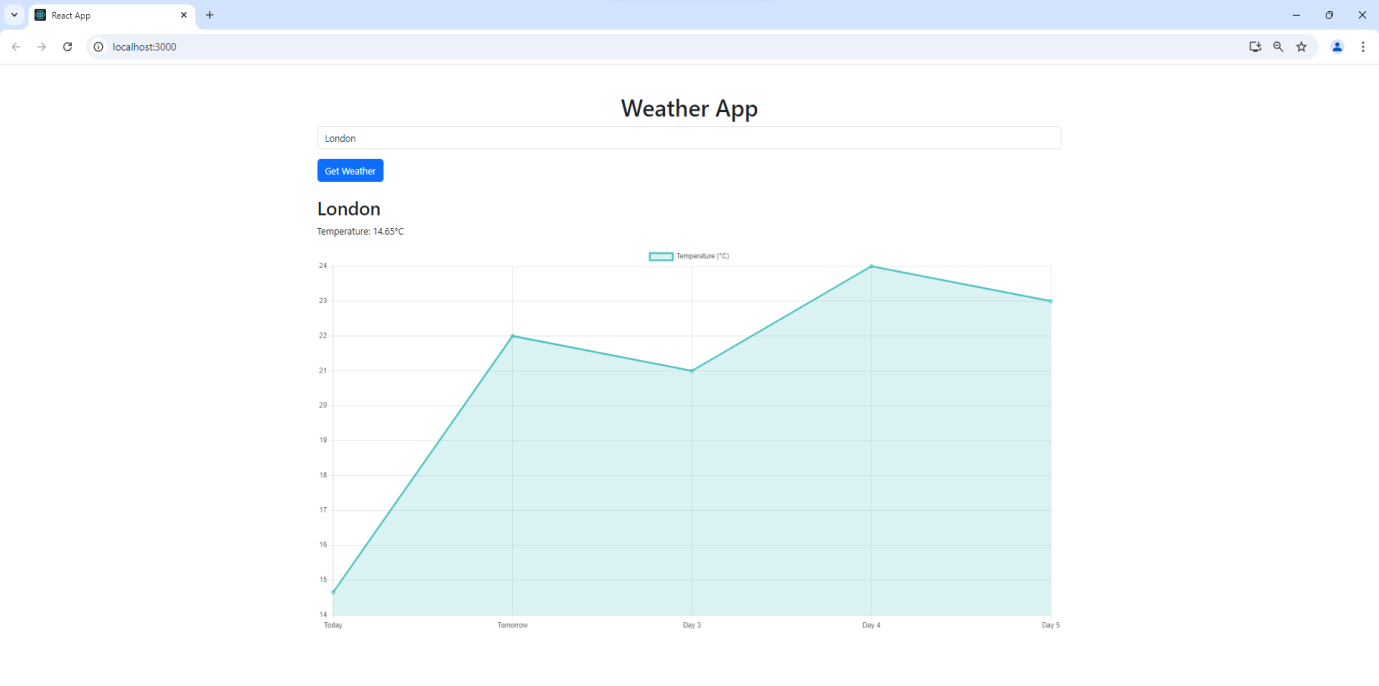
* In the terminal, run:

**npm start**

React app will now be running locally at http://localhost:3000, showing both current and historical weather information.







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

**Step 1: Open a Terminal/Command Prompt**: Depending on your operating system:

**Step 2: set npm global directory:**

**Create a folder for the experiment:** C:\Users\hp\OneDrive\Desktop\LAB8

Run the below command in cmd:

* **cd C:\Users\hp\OneDrive\Desktop**
* **C:\Users\hp\OneDrive\Desktop>npm config set prefix C:\Users\hp\OneDrive\Desktop\LAB8**
* C:\Users\hp\OneDrive\Desktop>cd LAB8

**Step 3: Create a New React Application**

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

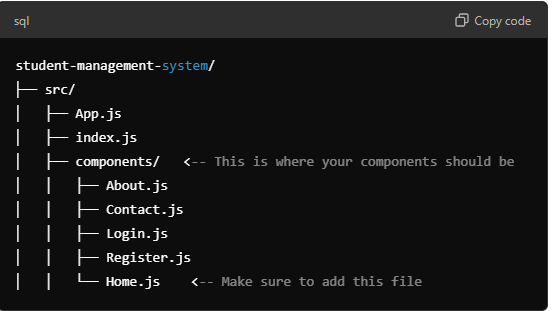
**Step 4: Install React Router**

In the terminal, make sure you are still in the student-management-system directory, then execute:

* npm install react-router-dom

**Step 5: Edit the Source Code**

1. **Open Your Code Editor**: Open your project folder (student-management-system) in a code editor like **Visual Studio Code**, **Atom**, or any text editor you prefer.
2. **Create Component Files**: Inside the src folder, create a new folder named components, and within that folder, create the following files:
   * About.js
   * Contact.js
   * Login.js
   * Register.js
   * Home.js



**Step 6: Edit App.js for Routing**

In the src folder, open App.js and replace the default code with the routing code provided earlier. Ensure you import your components correctly.

**Home.js**

import React from 'react';

const Home = () => {

  return (

    <div>

      <h1>Welcome to Student Management System</h1>

      <p>Manage student details and more!</p>

    </div>

  );

};

export default Home;

**About.js:**

import React from 'react';

const About = () => {

return (

<div>

<h1>About Us</h1>

<p>This student management system is developed to help manage student data efficiently.</p>

</div>

);

};

export default About;

**Contact.js**:

import React from 'react';

const Contact = () => {

return (

<div>

<h1>Contact Us</h1>

<p>Email: support@studentmanagement.com</p>

<p>Phone: +123-456-7890</p>

</div>

);

};

export default Contact;

**Login.js**:

import React, { useState } from 'react';

const Login = () => {

const [credentials, setCredentials] = useState({

email: '',

password: ''

});

const handleChange = (e) => {

setCredentials({

...credentials,

[e.target.name]: e.target.value

});

};

const handleSubmit = (e) => {

e.preventDefault();

alert('Login Successful!');

};

return (

<div>

<h1>Login</h1>

<form onSubmit={handleSubmit}>

<label>Email: </label>

<input

type="email"

name="email"

value={credentials.email}

onChange={handleChange}

required

/>

<br />

<label>Password: </label>

<input

type="password"

name="password"

value={credentials.password}

onChange={handleChange}

required

/>

<br />

<button type="submit">Login</button>

</form>

</div>

);

};

exportdefault Login;

**Register.js**:

import React, { useState } from 'react';

const Register = () => {

const [formData, setFormData] = useState({

username: '',

email: '',

password: ''

});

const handleChange = (e) => {

setFormData({

...formData,

[e.target.name]: e.target.value

});

};

const handleSubmit = (e) => {

e.preventDefault();

alert('Registration Successful!');

};

return (

<div>

<h1>Register</h1>

<form onSubmit={handleSubmit}>

<label>Username: </label>

<input

type="text"

name="username"

value={formData.username}

onChange={handleChange}

required

/>

<br />

<label>Email: </label>

<input

type="email"

name="email"

value={formData.email}

onChange={handleChange}

required

/>

<br />

<label>Password: </label>

<input

type="password"

name="password"

value={formData.password}

onChange={handleChange}

required

/>

<br />

<button type="submit">Register</button>

</form>

</div>

);

};

export default Register;

these .js files should be saved in the new “components” folder🡪 refer the above diagram for creating .js files

**update the app.js file by below code:**

**app.js:**

import React from 'react';

import { BrowserRouter as Router, Route, Routes, Link } from 'react-router-dom';

import Home from './components/Home';

import Register from './components/Register';

import Login from './components/Login';

import Contact from './components/Contact';

import About from './components/About';

function App() {

return (

<Router>

<div>

<nav>

<ul>

<li><Link to="/">Home</Link></li>

<li><Link to="/register">Register</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>

<Routes>

<Route path="/" element={<Home />} />

<Route path="/register" element={<Register />} />

<Route path="/login" element={<Login />} />

<Route path="/contact" element={<Contact />} />

<Route path="/about" element={<About />} />

</Routes>

</div>

</Router>

);

}

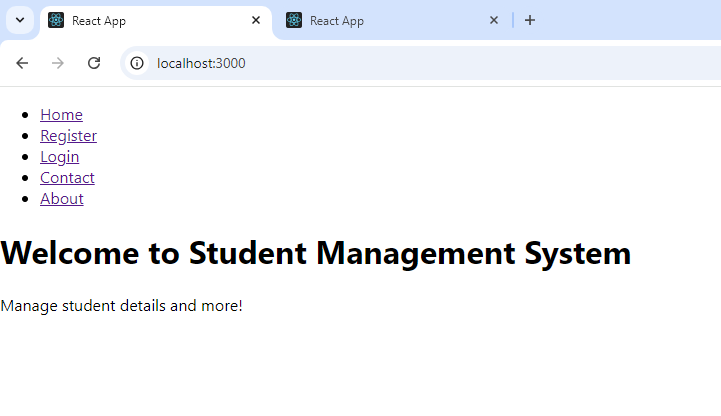
export default App;

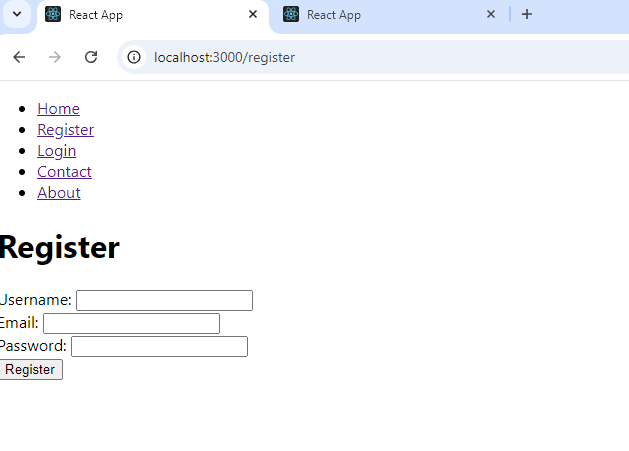
**Step 7: Run Your Application**

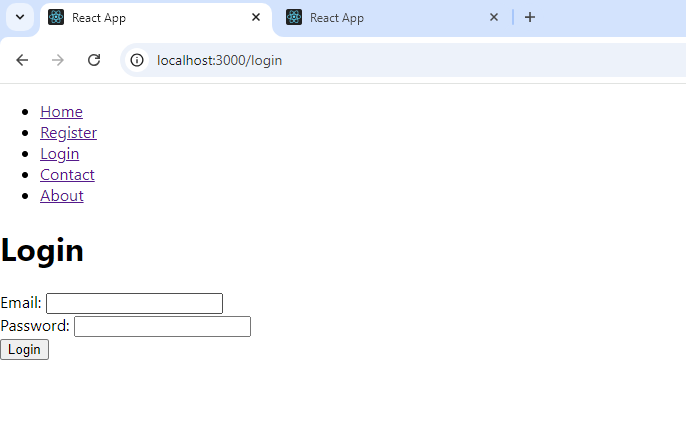
1. **Start the Development Server**: In your terminal, make sure you are still in the student-management-system directory, then execute:

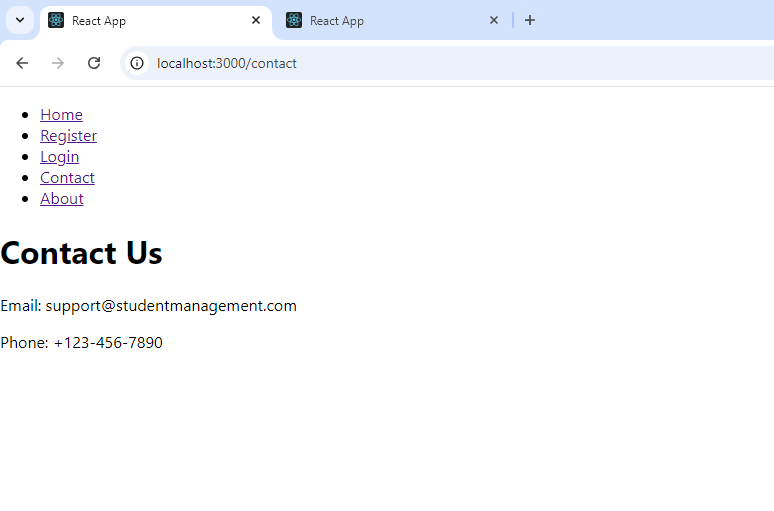
* **npm start**

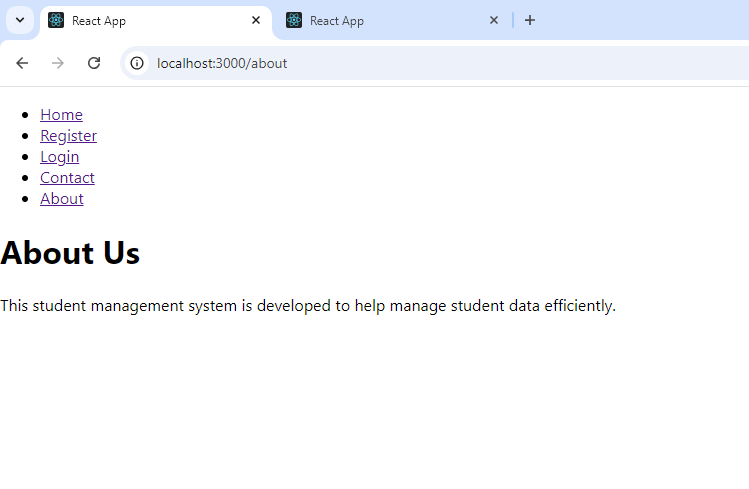
**OUTPUT:**

****

****

****



****

09.Develop a Java Stand-alone application that connects with the database MySQL and perform CRUD operations on the database tables.

In Java applications, connection to MySQL Databases is provided using JDBC(Java Database Connectivity) API’s and execute SQL queries and statements.

**Step 1: Download MySQL Connector (.jar file)**

* mysql\_connector.jar file helps in establishing the connection to MySQL Database using JDBC driver class (jdbc:mysql://localhost:3306/)

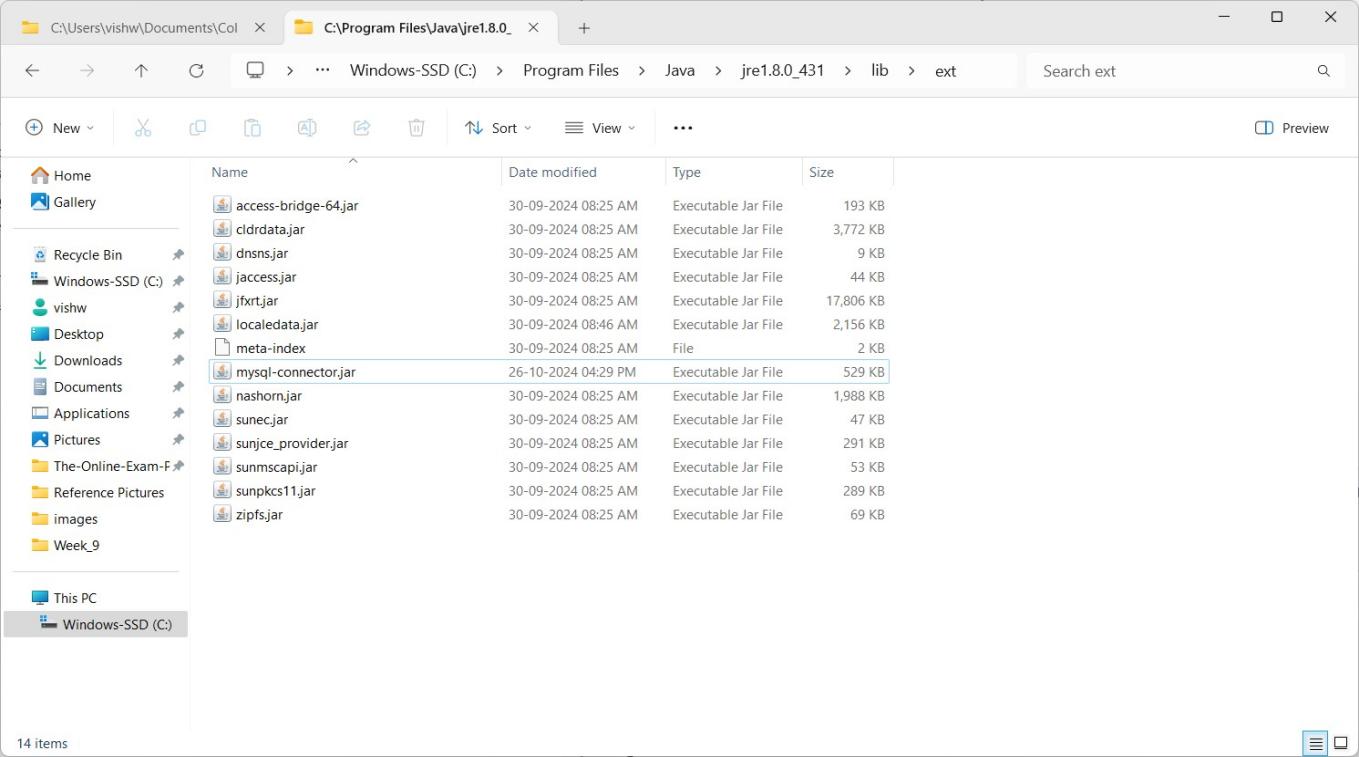
Download the jar file from here:

<https://static.javatpoint.com/src/jdbc/mysql-connector.jar>

**Step 2: Place the .jar file in JRE folder**

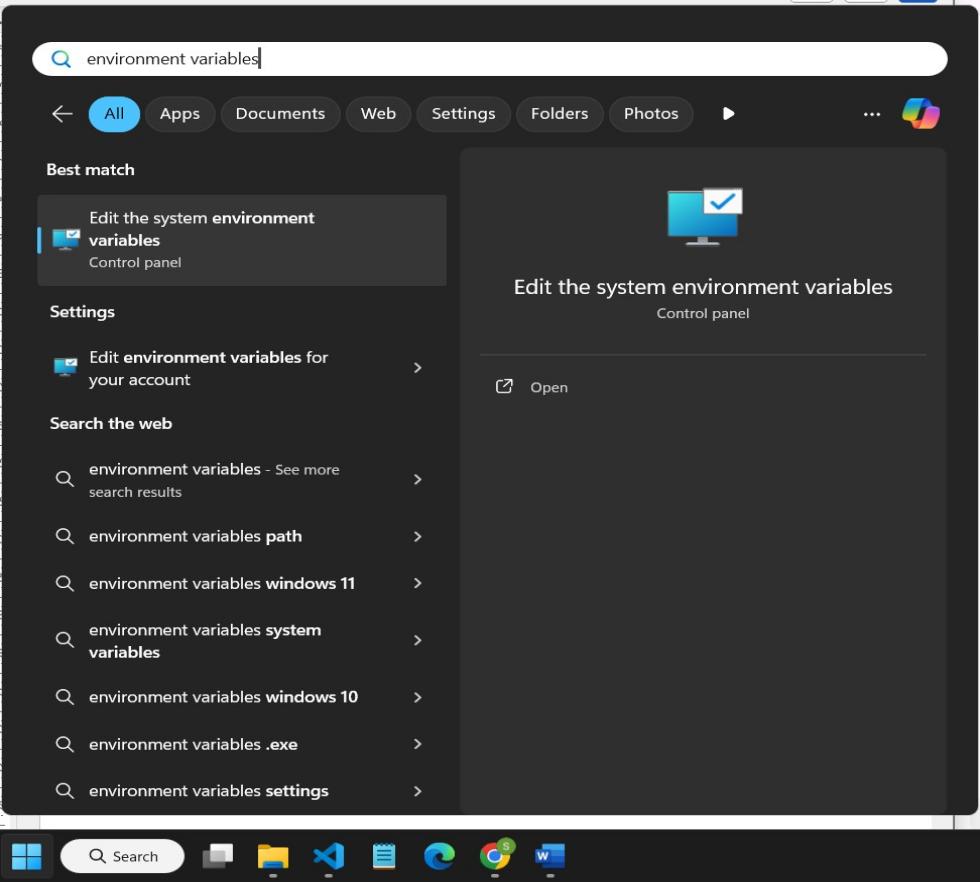
* Move the .jar file to Path:

“C:\Program Files\Java\jre1.8.0\_431\lib\ext\”

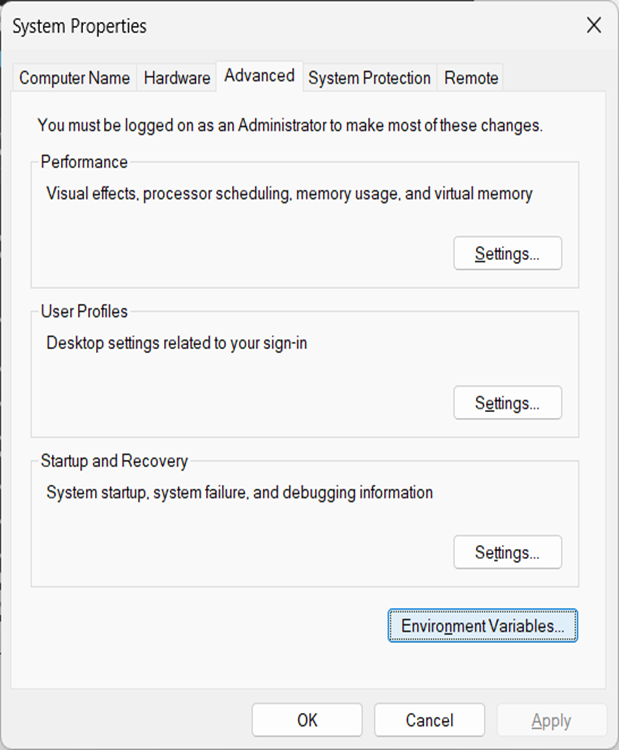


**Step 3:** Setting the Environment Variables

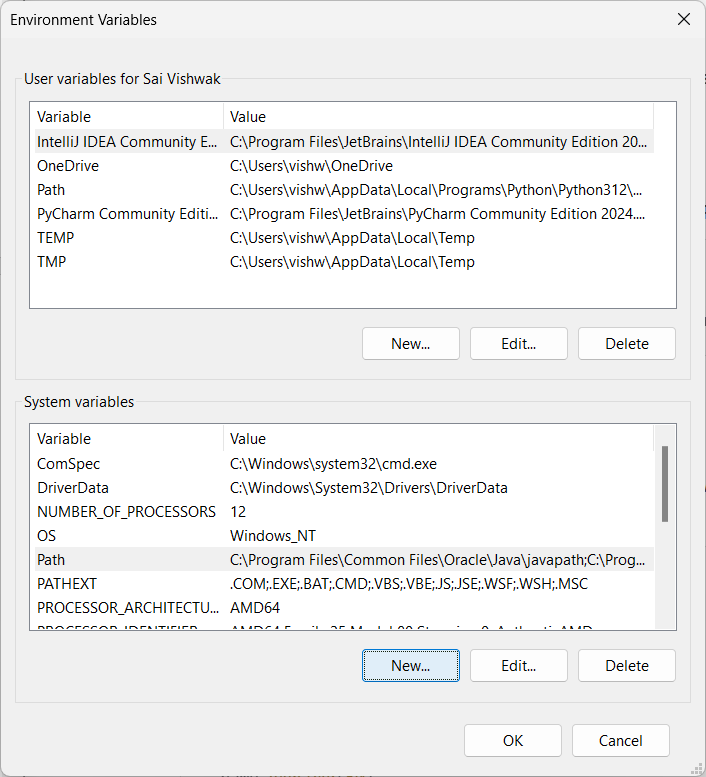
* In Windows Search Box, search for ‘Environment Variables’ and click on ‘Edit the environment variables’



* Click on “Environment Variables” in the “System properties” dialog box.

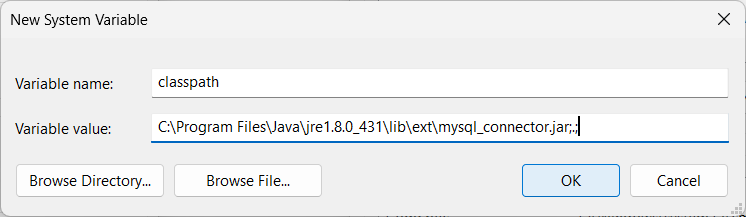


* Click on “New…” button under “System variables”



* Create a new system variable with variable name “classpath” and variable value as PATH to the .jar file placed in jre\lib\ext folder

C:\Program Files\Java\jre1.8.0\_431\lib\ext\mysql\_connector.jar;.;



# **Step 4:** Open XAMPP control panel and turn on MySQL server.

**Step 5:** Create java files to perform CRUD operations on MySQL database.

**1.mysql\_con.java**

import java.sql.\*;

public class mysql\_con {

public static void main(String[] args) {

// Database credentials

String url = "jdbc:mysql://localhost:3306/";

String user = "root"; // Replace with your MySQL username String password = ""; // Replace with your MySQL password

Connection con = null; Statement stmt = null;

try {

// Load the MySQL JDBC driver (use `com.mysql.cj.jdbc.Driver` for MySQL Connector/J 8.0+)

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

// Step 1: Connect to MySQL server

con = DriverManager.getConnection(url, user, password); stmt = con.createStatement();

// Step 2: Create a new database

String createDB = "CREATE DATABASE IF NOT EXISTS mydb"; stmt.executeUpdate(createDB);

System.out.println("Database 'mydb' created successfully.");

// Step 3: Connect to the new database

con = DriverManager.getConnection(url + "mydb", user, password); stmt = con.createStatement();

// Step 4: Create a new table in the database

String createTable = "CREATE TABLE IF NOT EXISTS emp (" +

"id INT(3) PRIMARY KEY, " + "name VARCHAR(10), " + "age INT(2))";

stmt.executeUpdate(createTable); System.out.println("Table 'emp' created successfully.");

} catch (ClassNotFoundException e) {

System.out.println("MySQL JDBC Driver not found. Please add it to your library path.");

e.printStackTrace();

} catch (SQLException e) {

System.out.println("SQL Error: " + e.getMessage()); e.printStackTrace();

} finally {

// Close the resources try {

if (stmt != null) stmt.close(); if (con != null) con.close();

} catch (SQLException e) { e.printStackTrace();

}

}

}

}

**Output**:

Database 'my\_db' created successfully. Table 'emp' created successfully.

**2.insert\_db.java**

import java.sql.\*;

import java.util.Scanner;

public class insert\_db {

public static void main(String[] args) {

// Database connection details

String url = "jdbc:mysql://localhost:3306/mydb";

String user = "root";

// Replace with your MySQL username String

password = ""; // Replace with yourMySQL password

Connection con = null; PreparedStatement pstmt = null;

Scanner scanner = new Scanner(System.in);

try {

// Load the MySQL JDBC Driver Class.forName("com.mysql.jdbc.Driver");

// Establish a connection to the database

con = DriverManager.getConnection(url, user, password);

// Prompt the user for employee details System.out.print("Enter Employee ID: ");

int id = scanner.nextInt();

scanner.nextLine(); // Clear the newline character

System.out.print("Enter Employee Name (max 10 characters): ");

String name = scanner.nextLine();

System.out.print("Enter Employee Age: ");

int age = scanner.nextInt();

// Insert SQL command

String sql = "INSERT INTO emp (id, name, age) VALUES (?, ?, ?)";

// Create a PreparedStatement to execute the SQL command pstmt = con.prepareStatement(sql);

pstmt.setInt(1, id); pstmt.setString(2, name); pstmt.setInt(3, age);

// Execute the insertion

int rowsAffected = pstmt.executeUpdate(); if (rowsAffected > 0) {

System.out.println("Record inserted successfully.");

} else {

System.out.println("Failed to insert the record.");

}

} catch (ClassNotFoundException e) {

System.out.println("MySQL JDBC Driver not found. Please add it to your library

path.");

e.printStackTrace();

} catch (SQLException e) {

System.out.println("SQL Error: " + e.getMessage()); e.printStackTrace();

} finally {

// Close the resources try {

if (pstmt != null) pstmt.close(); if (con != null) con.close(); scanner.close();

} catch (SQLException e) { e.printStackTrace();

}

}

}

}

**Output:**

Enter Employee ID: 101

Enter Employee Name (max 10 characters): Alice

Enter Employee Age: 25

Record inserted successfully.

**3.display\_db.java**

import java.sql.\*;

public class display\_db {

public static void main(String[] args) {

// Database connection details

String url = "jdbc:mysql://localhost:3306/mydb";

String user = "root"; // Replace with your MySQL username String password = ""; // Replace with your MySQL password

Connectioncon=null;

Statementstmt =null;

try {

// Load the MySQL JDBC Driver Class.forName("com.mysql.jdbc.Driver");

// Establish a connection to the database

con = DriverManager.getConnection(url, user, password);

// Create a statement to execute SQL queries stmt = con.createStatement();

String sql = "SELECT \* FROM emp";

// Execute the query and get the result set

ResultSet rs = stmt.executeQuery(sql);

// Display the records

System.out.println("ID Name Age");

System.out.println(" ");

while (rs.next()) {

int id = rs.getInt("id");

String name = rs.getString("name");

int age = rs.getInt("age");

System.out.printf("%-4d %-10s %d%n", id, name, age);

}

} catch (ClassNotFoundException e) {

System.out.println("MySQL JDBC Driver not found. Please add it to your library

path.");

e.printStackTrace();

} catch (SQLException e) {

System.out.println("SQL Error: " + e.getMessage()); e.printStackTrace();

} finally {

// Close the resources try {

if (stmt != null) stmt.close(); if (con != null) con.close();

} catch (SQLException e) { e.printStackTrace();

}

}

}

}

|  |  |  |
| --- | --- | --- |
| **Output**:  ID | Name | Age |
| 101 | Alice | 25 |
| 102 | Bob | 30 |
| 103 | Charlie | 28 |

**4.update\_db.java**

import java.sql.\*;

import java.util.Scanner;

public class update\_db {

public static void main(String[] args) {

// Database connection details

String url = "jdbc:mysql://localhost:3306/mydb";

String user = "root"; // Replace with your MySQL username String password = ""; // Replace with your MySQL password

Connection con = null; PreparedStatement pstmt =null;

Scanner scanner = new Scanner(System.in); try {

// Load the MySQL JDBC Driver Class.forName("com.mysql.jdbc.Driver");

// Establish a connection to the database

con = DriverManager.getConnection(url, user, password);

// Prompt the user for the employee ID to update System.out.print("Enter Employee ID to update: "); int id = scanner.nextInt();

scanner.nextLine(); // Clear the newline character

// Prompt for new name and age

System.out.print("Enter new Employee Name (max 10 characters): "); String name = scanner.nextLine();

System.out.print("Enter new Employee Age: "); int age = scanner.nextInt();

// Update SQL command

String sql = "UPDATE emp SET name = ?, age = ? WHERE id = ?";

// Create a PreparedStatement to execute the SQL command pstmt = con.prepareStatement(sql);

pstmt.setString(1, name); pstmt.setInt(2, age); pstmt.setInt(3, id);

// Execute the update

int rowsAffected = pstmt.executeUpdate(); if (rowsAffected > 0) {

System.out.println("Record updated successfully.");

} else {

System.out.println("No record found with ID: " + id);

}

} catch (ClassNotFoundException e) {

System.out.println("MySQL JDBC Driver not found. Please add it to your library

path.");

e.printStackTrace();

} catch (SQLException e) {

System.out.println("SQL Error: " + e.getMessage()); e.printStackTrace();

} finally {

// Close the resources try {

if (pstmt != null) pstmt.close(); if (con != null) con.close(); scanner.close();

} catch (SQLException e) { e.printStackTrace();

}

}

}

}

Output:

Enter the ID of the employee you want to update: 101 Enter new Name (max 10 characters): Alicia

Enter new Age: 26

Record updated successfully.

**5.delete\_db.java**

import java.sql.\*;

import java.util.Scanner;

public class delete\_db {

public static void main(String[] args) {

// Database connection details

String url = "jdbc:mysql://localhost:3306/mydb";

String user = "root"; // Replace with your MySQL username String password = ""; // Replace with your MySQL password

Connection con = null; PreparedStatement pstmt = null;

Scanner scanner = new Scanner(System.in);

try {

// Load the MySQL JDBC Driver Class.forName("com.mysql.jdbc.Driver");

// Establish a connection to the database

con = DriverManager.getConnection(url, user, password);

// Prompt the user for the employee ID to delete System.out.print("Enter Employee ID to delete: "); int id = scanner.nextInt();

// Delete SQL command

String sql = "DELETE FROM emp WHERE id = ?";

// Create a PreparedStatement to execute the SQL command

pstmt = con.prepareStatement(sql);

pstmt.setInt(1, id);

// Execute the deletion

int rowsAffected = pstmt.executeUpdate(); if (rowsAffected > 0) {

System.out.println("Record deleted successfully.");

} else {

System.out.println("No record found with ID: " + id);

}

} catch (ClassNotFoundException e) {

System.out.println("MySQL JDBC Driver not found. Please add it to your library

path.");

e.printStackTrace();

} catch (SQLException e) {

System.out.println("SQL Error: " + e.getMessage()); e.printStackTrace();

} finally {

// Close the resources try {

if (pstmt != null) pstmt.close(); if (con != null) con.close(); scanner.close();

} catch (SQLException e) { e.printStackTrace();

}

}

}

}

**Output:**

Enter the ID of the employee you want to delete: 102

Record deleted successfully.

**10.For the above experiment-06 application create authorized end points using JWT (JSON Web Token).**

**Aim:**

* To develop an Express web application that performs CRUD operations (Create, Read, Update, Delete) on student data through a REST API, and to secure the endpoints using JSON Web Token (JWT) for authorization. This ensures that only authorized users can interact with the API.
* To set up MySQL with Express and adjust the project to perform CRUD operations with a MySQL database.

**Prerequisites:**

* **Node.js** and **npm**
* **MySQL Database**: Set up a MySQL database with a students table.
* **Postman**: Used to test the API.

**Project Setup:**

1. **Initialize a new Node.js project:**

mkdir student-api

cd student-api

npm init -y

npm install express mysql2 jsonwebtoken bcryptjs dotenv

* **express**: A Node.js framework for building web applications and APIs.
* **mysql2**: A MySQL client for Node.js, compatible with Sequelize.
* **jsonwebtoken**: A library to create and verify JSON Web Tokens (JWT) for authentication.
* **bcryptjs**: A JavaScript-only version of bcrypt for hashing passwords.
* **dotenv**: A library to load environment variables from a .env file.

**Folder Structure:**

**student-api/**

**├── config/**

**│**  **└──** db.js

**├── models/**

**│** └── Student.js

└── User.js

**├── routes/**

**│** ├── auth.js

│ └── students.js

**├── .env**

**├── server.js**

**Step 1: Environment Configuration**

**Create a .env file:Top of Form**

PORT=5000

DB\_HOST=localhost

DB\_USER=root

DB\_PASSWORD=

DB\_NAME=cet\_lab

JWT\_SECRET=d8c2f40f54740f0828286c49cdea55161f357e75944d3ec07136952def67a0fe

**Bottom of Form**

**Step 2: Database Configuration (config/db.js)**

**Set up the MySQL connection in config/db.js:**

// config/db.js

const { Sequelize } = require('sequelize');

const sequelize = new Sequelize(

process.env.DB\_NAME,

process.env.DB\_USER,

process.env.DB\_PASSWORD,

{

host: process.env.DB\_HOST,

dialect: 'mysql',

}

);

module.exports = sequelize;

**Step 3: Define Models for CRUD Operations (models/studentModel.js)**

// models/Student.js

const { DataTypes } = require('sequelize');

const sequelize = require('../config/db');

const Student = sequelize.define('Student', {

name: {

type: DataTypes.STRING,

allowNull: false,

},

age: {

type: DataTypes.INTEGER,

allowNull: false,

},

grade: {

type: DataTypes.STRING,

allowNull: false,

},

});

module.exports = Student;

**Step 4 : Sync Database**: Modify **server.js** to sync the database and establish the MySQL connection.

require('dotenv').config();

const express = require('express');

const sequelize = require('./config/db');

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

const studentRoutes = require('./routes/students');

const app = express();

const PORT = process.env.PORT || 5000;

app.use(express.json());

// Sync Sequelize with MySQL

sequelize.sync()

.then(() => console.log('MySQL Database connected and synced'))

.catch((err) => console.error('Error connecting to MySQL:', err));

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

app.use('/api/students', studentRoutes);

app.listen(PORT, () => {

console.log(`Server running on port ${PORT}`);

});

**Step 5:** Set Up Routes (routes/student.js and routes/auth.js)

// routes/students.js

const express = require('express');

const router = express.Router();

const Student = require('../models/Student');

const jwt = require('jsonwebtoken');

const authenticateToken = (req, res, next) => {

const token = req.header('Authorization')?.split(' ')[1];

if (!token) return res.status(401).json({ message: 'Access Denied' });

try {

const verified = jwt.verify(token, process.env.JWT\_SECRET);

req.user = verified;

next();

} catch (err) {

res.status(400).json({ message: 'Invalid Token' });

}

};

// Create a new student

router.post('/', authenticateToken, async (req, res) => {

try {

const { name, age, grade } = req.body;

const student = await Student.create({ name, age, grade });

res.json(student);

} catch (err) {

res.status(400).json({ message: err.message });

}

});

// Get all students

router.get('/', authenticateToken, async (req, res) => {

try {

const students = await Student.findAll();

res.json(students);

} catch (err) {

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

}

});

// Get a student by ID

router.get('/:id', authenticateToken, async (req, res) => {

try {

const student = await Student.findByPk(req.params.id);

if (!student) return res.status(404).json({ message: 'Student not found' });

res.json(student);

} catch (err) {

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

}

});

// Update a student by ID

router.put('/:id', authenticateToken, async (req, res) => {

try {

const { name, age, grade } = req.body;

const [updated] = await Student.update(

{ name, age, grade },

{ where: { id: req.params.id } }

);

if (!updated) return res.status(404).json({ message: 'Student not found' });

res.json({ message: 'Student updated successfully' });

} catch (err) {

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

}

});

// Delete a student by ID

router.delete('/:id', authenticateToken, async (req, res) => {

try {

const deleted = await Student.destroy({ where: { id: req.params.id } });

if (!deleted) return res.status(404).json({ message: 'Student not found' });

res.json({ message: 'Student deleted successfully' });

} catch (err) {

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

}

});

module.exports = router;

**Implement Authentication Logic in authRoutes**

**// routes/auth.js**

const express = require('express');

const router = express.Router();

const bcrypt = require('bcrypt');

const jwt = require('jsonwebtoken');

const User = require('../models/User'); // Define a User model similar to Student

// Register a new user

router.post('/register', async (req, res) => {

try {

const { username, password } = req.body;

const hashedPassword = await bcrypt.hash(password, 10);

const user = await User.create({ username, password: hashedPassword });

res.json({ message: 'User registered successfully' });

} catch (err) {

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

}

});

// Login user

router.post('/login', async (req, res) => {

try {

const { username, password } = req.body;

const user = await User.findOne({ where: { username } });

if (!user || !await bcrypt.compare(password, user.password)) {

return res.status(400).json({ message: 'Invalid credentials' });

}

const token = jwt.sign({ id: user.id }, process.env.JWT\_SECRET, { expiresIn: '1h' });

res.json({ token });

} catch (err) {

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

}

});

module.exports = router;

**6.Start the Server**:

* In your terminal, navigate to the project’s root directory.
* In XAMP, Start the MYSQL and Apache server, then enter in cmd as

node server.js

**7.. Steps to Test Endpoints in Postman**

**1. Open Postman**

* Launch the Postman application on your computer.

**2. Create a New Collection (Optional)**

* You can create a collection to organize your requests. Click on the “Collections” tab on the left sidebar, then click “New Collection” and give it a name (e.g., “Student API”).

**3. Register a New User (POST /api/auth/register)**

* **Create Request**:
  + Click on “New” > “Request”.
  + Name the request (e.g., "Register User") and save it in your collection.
* **Set Method and URL**:
  + Set the request method to POST.
  + Enter the URL: http://localhost:5000/api/auth/register.
* **Add Body**:
  + Go to the **Body** tab.
  + Select raw and choose JSON from the dropdown.
  + Enter the following JSON data:

{

"username": "testuser",

"password": "testpassword"

}

* **Send Request**:
  + Click the “Send” button.
  + You should receive a response indicating that the user was registered successfully.

**4. Login User to Obtain JWT Token (POST /api/auth/login)**

* **Create Login Request**:
  + Click on “New” > “Request”.
  + Name the request (e.g., "Login User") and save it.
* **Set Method and URL**:
  + Set the method to POST.
  + Enter the URL: http://localhost:5000/api/auth/login.
* **Add Body**:
  + Go to the **Body** tab.
  + Select raw and choose JSON.
  + Enter the following JSON data:

{

"username": "testuser",

"password": "testpassword"

}

* **Send Request**:
  + Click the “Send” button.
  + You should receive a response with a JWT token:

{

"token": "your\_jwt\_token\_here"

}

* + Copy this token; you will use it for the next steps.

**5. Create a New Student (POST /api/students)**

* **Create Student Request**:
  + Click on “New” > “Request”.
  + Name the request (e.g., "Create Student") and save it.
* **Set Method and URL**:
  + Set the method to POST.
  + Enter the URL: http://localhost:5000/api/students.
* **Add Authorization**:
  + Go to the **Authorization** tab.
  + Choose Bearer Token from the dropdown.
  + Paste the JWT token you copied earlier into the token field.
* **Add Body**:
  + Go to the **Body** tab.
  + Select raw and choose JSON.
  + Enter the following JSON data for the student:

{

"name": "John Doe",

"age": 20,

"grade": "A"

}

* **Send Request**:
  + Click the “Send” button.
  + You should receive a response with the created student's details.

**6. Get All Students (GET /api/students)**

* **Create Get Students Request**:
  + Click on “New” > “Request”.
  + Name the request (e.g., "Get All Students") and save it.
* **Set Method and URL**:
  + Set the method to GET.
  + Enter the URL: http://localhost:5000/api/students.
* **Add Authorization**:
  + Go to the **Authorization** tab and use the same JWT token as before.
* **Send Request**:
  + Click the “Send” button.
  + You should receive a list of all students in the response.

**7. Get a Student by ID (GET /api/students/:id)**

* **Create Get Student Request**:
  + Click on “New” > “Request”.
  + Name the request (e.g., "Get Student by ID") and save it.
* **Set Method and URL**:
  + Set the method to GET.
  + Enter the URL: http://localhost:5000/api/students/1 (replace 1 with the actual ID of a student).
* **Add Authorization**:
  + Use the JWT token again in the **Authorization** tab.
* **Send Request**:
  + Click the “Send” button.
  + You should receive the details of the specified student.

**8. Update a Student by ID (PUT /api/students/:id)**

* **Create Update Student Request**:
  + Click on “New” > “Request”.
  + Name the request (e.g., "Update Student") and save it.
* **Set Method and URL**:
  + Set the method to PUT.
  + Enter the URL: http://localhost:5000/api/students/1 (replace 1 with the actual student ID).
* **Add Authorization**:
  + Use the JWT token in the **Authorization** tab.
* **Add Body**:
  + Go to the **Body** tab.
  + Select raw and choose JSON.
  + Enter updated student details:

{

"name": "ramu",

"age": 21,

"grade": "B"

}

* **Send Request**:
  + Click the “Send” button.
  + You should receive a success message indicating the student was updated.

**9. Delete a Student by ID (DELETE /api/students/:id)**

* **Create Delete Student Request**:
  + Click on “New” > “Request”.
  + Name the request (e.g., "Delete Student") and save it.
* **Set Method and URL**:
  + Set the method to DELETE.
  + Enter the URL: http://localhost:5000/api/students/1 (replace 1 with the actual student ID).
* **Add Authorization**:
  + Use the JWT token in the **Authorization** tab.
* **Send Request**:
  + Click the “Send” button.
  + You should receive a confirmation message indicating the student was deleted successfully.

**11. Make a responsive web application for shopping cart using Bootstrap framework.**

**Aim**

Develop a responsive shopping cart web application featuring product catalog, shopping cart summary, and checkout functionality for electronics products.

**Description**

The application showcases various electronic products, allowing users to add items to their cart and view a summary of selected products with the total price. The design uses HTML, CSS, Bootstrap for styling and layout, and JavaScript for cart functionality.

**Tools**

* **HTML5**: For structuring the pages.
* **Bootstrap**: For responsive design and styling.
* **JavaScript**: For handling cart operations (add to cart, calculate total).

**Execution Steps**

1. **Set up HTML and Bootstrap**: Include Bootstrap for styling, create navbar and container sections.
2. **Create Product Cards**: Use Bootstrap’s card component to display product images, names, and prices.
3. **Add JavaScript for Cart Operations**: Implement addToCart and updateCart functions to handle item additions and total calculation.

**Shoppingcart.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Shopping Cart - Electronics</title>

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">

</head>

<body>

<!-- Navbar -->

<nav class="navbar navbar-expand-lg navbar-light bg-light">

<a class="navbar-brand" href="#">ElectroShop</a>

<button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarNav">

<ul class="navbar-nav">

<li class="nav-item active"><a class="nav-link" href="#">Home</a></li>

<li class="nav-item"><a class="nav-link" href="#">Products</a></li>

<li class="nav-item"><a class="nav-link" href="#">About</a></li>

<li class="nav-item"><a class="nav-link" href="#">Contact</a></li>

</ul>

</div>

</nav>

<div class="container mt-5">

<h1 class="text-center mb-4">Shopping Cart</h1>

<h2 class="text-center mb-4">Featured Electronics</h2>

<div class="row">

<div class="col-md-9">

<div class="row" id="product-list"></div>

</div>

<div class="col-md-3">

<h2>Cart Summary</h2>

<ul class="list-group" id="cart-items"></ul>

<h4 class="mt-3">Total: <span id="total">₹0.00</span></h4>

<button class="btn btn-success mt-3">Checkout</button>

</div>

</div>

</div>

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>

<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.5.2/dist/umd/popper.min.js"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

<script>

const products = [

{ id: 1, name: "Laptop", price: 74999, img: "https://cdn.thewirecutter.com/wp-content/media/2024/07/editing-laptop-2048px-233661.jpg?auto=webp&quality=75&width=1024" },

{ id: 2, name: "Smartphone", price: 49999, img: "https://kddi-h.assetsadobe3.com/is/image/content/dam/au-com/common/graph/app/pixel8a\_set.jpg?scl=1" },

{ id: 3, name: "Tablet", price: 29999, img: "https://cdn.thewirecutter.com/wp-content/media/2024/06/besttablets-2048px-9875.jpg" },

{ id: 4, name: "Headphones", price: 6999, img: "https://images.philips.com/is/image/philipsconsumer/60a04af525fe4e7589acb0c90075d2d7?$pnglarge$&wid=1250" },

{ id: 5, name: "Smartwatch", price: 8999, img: "https://gourban.in/cdn/shop/files/WAve.png?v=1705564263&width=2048" },

{ id: 6, name: "Camera", price: 54999, img: "https://images.herzindagi.info/image/2024/Jul/Camera-For-Photography-2.jpg" },

{ id: 7, name: "Bluetooth Speaker", price: 2999, img: "https://m.media-amazon.com/images/I/41n8sieHmNL.\_SX300\_SY300\_QL70\_FMwebp\_.jpg" },

{ id: 8, name: "Monitor", price: 19999, img: "https://media.takealot.com/covers\_images/0603d1d13051459489fa3d2d20c2a83f/s-fb.file" },

{ id: 9, name: "Gaming Console", price: 39999, img: "https://media-ik.croma.com/prod/https://media.croma.com/image/upload/v1712137916/Croma%20Assets/Gaming/Gaming%20Consoles/Images/305987\_tlfvzh.png?tr=w-600" }

];

const productList = document.getElementById('product-list');

products.forEach(({ id, name, price, img }) => {

productList.insertAdjacentHTML('beforeend', `

<div class="col-md-4 mb-4">

<div class="card" style="height: 400px;">

<img src="${img}" class="card-img-top" alt="${name}" style="height: 200px; object-fit: contain;">

<div class="card-body d-flex flex-column">

<h5 class="card-title">${name}</h5>

<p class="card-text">₹${price}</p>

<button class="btn btn-primary mt-auto" onclick="addToCart('${name}', ${price})">Add to Cart</button>

</div>

</div>

</div>

`);

});

let cart = [];

function addToCart(name, price) {

cart.push({ name, price });

updateCart();

}

function updateCart() {

const cartItems = document.getElementById('cart-items');

cartItems.innerHTML = cart.map(item => `<li class="list-group-item d-flex justify-content-between">${item.name}<span>₹${item.price}</span></li>`).join('');

const total = cart.reduce((sum, item) => sum + item.price, 0);

document.getElementById('total').innerText = `₹${total}`;

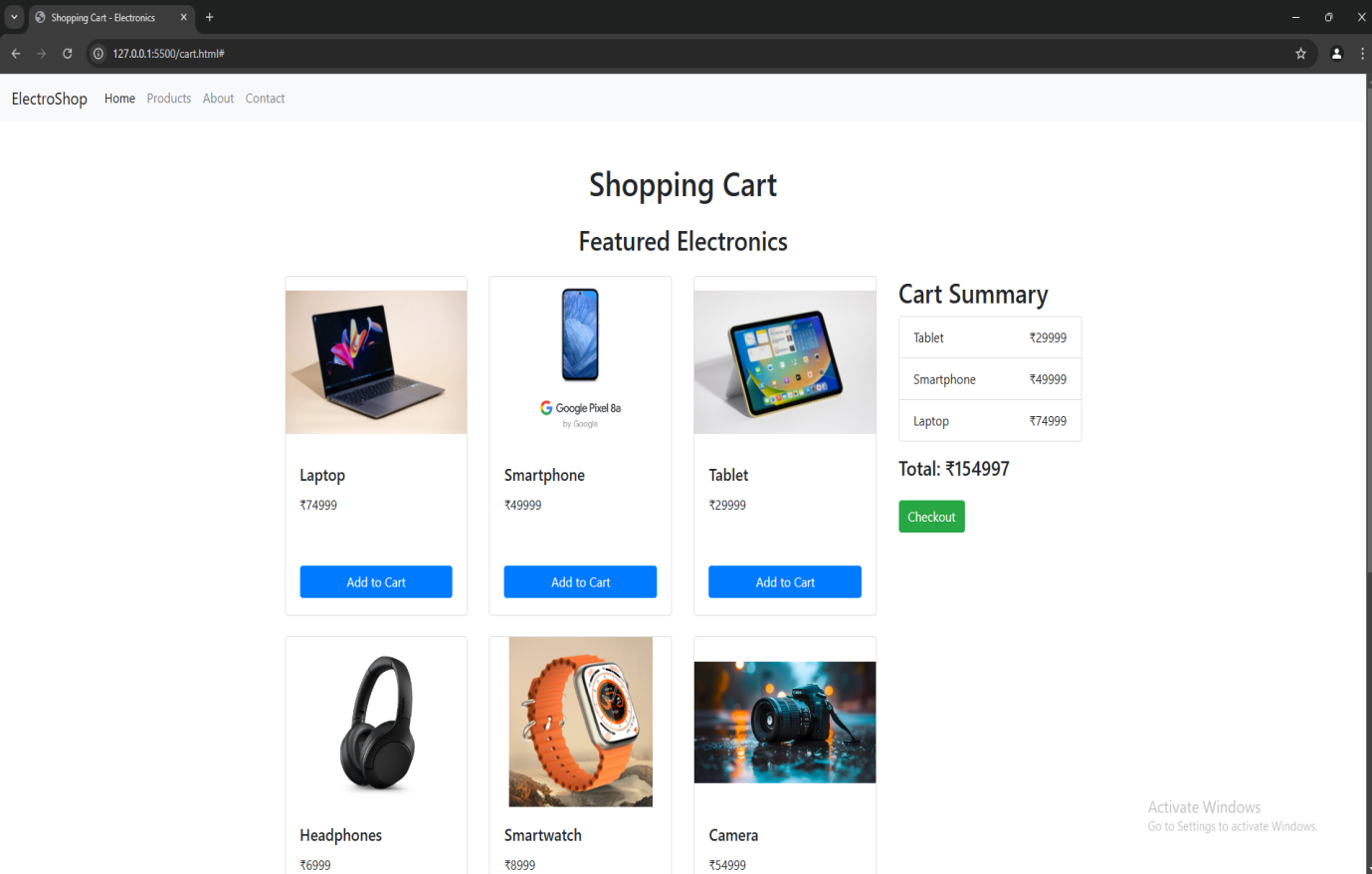
}

</script>

</body>

</html>

**OUTPUT:**



**12.Create a TODO application in react with necessary components.**

**Aim: Create a simple React TODO app with basic functionality (add, delete tasks).**

**Description (Steps):**

1. **Setup the React App:**
   * **Create a new React app using Create React App:**

**npx create-react-app todo-app**

**cd todo-app**

1. **Install Bootstrap:**
   * **Install Bootstrap for styling:**

**npm install bootstrap**

* + **Import Bootstrap in src/index.js:**

**import 'bootstrap/dist/css/bootstrap.min.css';**

**todo-app/src/App.js:**

**import React, { useState } from 'react';**

**const App = () => {**

**const [task, setTask] = useState('');**

**const [tasks, setTasks] = useState([]);**

**const addTask = () => {**

**if (task) setTasks([...tasks, task]);**

**setTask('');**

**};**

**const deleteTask = (index) => {**

**setTasks(tasks.filter((\_, i) => i !== index));**

**};**

**return (**

**<div className="container py-4">**

**<h1 className="text-center mb-4">TODO App</h1>**

**<div className="d-flex mb-3">**

**<input**

**type="text"**

**className="form-control mr-2"**

**value={task}**

**onChange={(e) => setTask(e.target.value)}**

**placeholder="Enter a new task"**

**/>**

**<button className="btn btn-primary" onClick={addTask}>Add</button>**

**</div>**

**<ul className="list-group">**

**{tasks.map((task, index) => (**

**<li key={index} className="list-group-item d-flex justify-content-between align-items-center">**

**{task}**

**<button className="btn btn-danger btn-sm" onClick={() => deleteTask(index)}>Delete</button>**

**</li>**

**))}**

**</ul>**

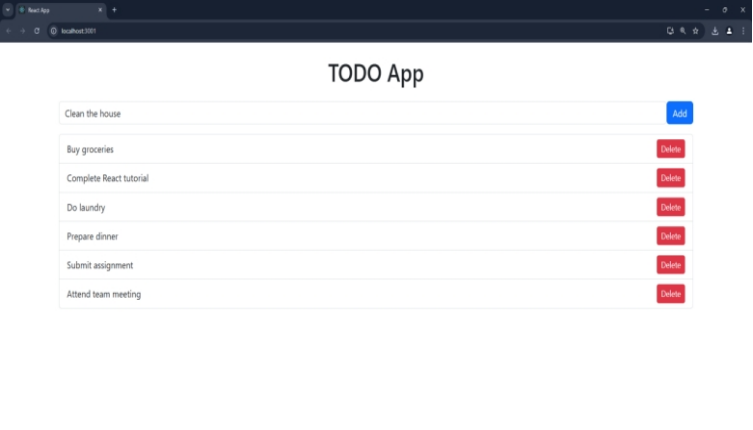
**</div>**

**);**

**};**

**export default App;**

**OUTPUT:**

****

**13.Design a controller with servlet that provides the interaction with application developed in experiment 1 and the database created in experiment9.**

**Aim:** To design a servlet controller that interacts with the front-end application (from **Experiment 1**) and the database (from **Experiment 5**) for seamless integration. The servlet will handle requests like user registration and database interactions.

**Step-by-Step Procedure**

**1. Setup Environment**

**Ensure you have:**

* **Java Development Kit (JDK) installed.**
* **Apache Tomcat configured.**
* **XAMPP running MySQL for database connectivity.**
* **The JDBC driver (mysql-connector-java.jar) added to the WEB-INF/lib folder or Tomcat's lib directory.**

**2.Directory Structure**

**registrationapplication/**

**├── registration.html**

**├── register.jsp**

**├── WEB-INF**

**├── web.xml**

**3. Database Setup**

**Use the database created in Experiment9. For example, a register table:**

**create table register(name varchar(10),password varchar(10),Phonenumber int(10),EmailID varchar(10));**

**4. Front-End Interaction (HTML Form)**

HTML form from Experiment 1 to capture user details:

**registration.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Validation</title>

<style>

body {

background-color: magenta;

font-family: Arial, sans-serif;

color: white;

}

h1 {

text-align: center;

}

table {

margin: auto;

border-spacing: 10px;

}

td {

padding: 5px;

}

input[type="text"], input[type="password"] {

width: 100%;

padding: 5px;

}

input[type="submit"], input[type="reset"] {

padding: 5px 10px;

cursor: pointer;

}

</style>

</head>

<body>

<form action="register.jsp" method="post">

<h1>Registration Form</h1>

<table>

<tr>

<td><label for="t1">Name:</label></td>

<td><input type="text" id="t1" name="t1" minlength="6" required></td>

</tr>

<tr>

<td><label for="t2">Password:</label></td>

<td><input type="password" id="t2" name="t2" required></td>

</tr>

<tr>

<td><label for="t3">Phone Number:</label></td>

<td><input type="text" id="t3" name="t3" pattern="\d{10}" title="Enter a valid 10-digit phone number" required></td>

</tr>

<tr>

<td><label for="t4">E-mail ID:</label></td>

<td><input type="email" id="t4" name="t4" required></td>

</tr>

<tr>

<td><input type="submit" value="Submit"></td>

<td><input type="reset" value="Cancel"></td>

</tr>

</table>

</form>

</body>

</html>

JSP code based on the provided HTML form:

**register.jsp**

<%@ page language="java" import="java.sql.\*" %>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Registration</title>

<style>

body {

font-family: Arial, sans-serif;

background-color: #f4f4f9;

color: #333;

padding: 20px;

}

h3 {

color: green;

}

.error {

color: red;

}

</style>

</head>

<body>

<%

String v1 = request.getParameter("t1"); // Name

String v2 = request.getParameter("t2"); // Password

String v3 = request.getParameter("t3"); // Phone number

String v4 = request.getParameter("t4"); // E-mail ID

Connection cn = null;

PreparedStatement pst = null;

try {

// Load the MySQL driver

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

// Establish the connection

cn = DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb", "root", "");

// SQL query using PreparedStatement to avoid SQL injection

String query = "INSERT INTO register (name, password, phonenumber, emailid) VALUES (?, ?, ?, ?)";

pst = cn.prepareStatement(query);

// Set parameters for the query

pst.setString(1, v1);

pst.setString(2, v2);

pst.setString(3, v3); // Assuming v3 is a string, adjust accordingly if numeric

pst.setString(4, v4);

// Execute the query

int rowsInserted = pst.executeUpdate();

// Feedback to the user

if (rowsInserted > 0) {

out.println("<h3>Registration Successful</h3>");

} else {

out.println("<h3 class='error'>Registration Failed</h3>");

}

} catch (NumberFormatException e) {

out.println("<h3 class='error'>Invalid phone number format: " + e.getMessage() + "</h3>");

} catch (SQLException e) {

out.println("<h3 class='error'>Database error: " + e.getMessage() + "</h3>");

} catch (Exception e) {

out.println("<h3 class='error'>Unexpected error: " + e.getMessage() + "</h3>");

} finally {

// Close resources

try {

if (pst != null) pst.close();

if (cn != null) cn.close();

} catch (SQLException e) {

out.println("<h3 class='error'>Error closing resources: " + e.getMessage() + "</h3>");

}

}

%>

</body>

</html>

**web.xml Configuration**

Configure the servlet in the web.xml file.

**web.xml**

<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://java.sun.com/xml/ns/javaee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee

http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"

version="3.0">

<!-- Metadata about the application -->

<display-name>Registration Application</display-name>

<!-- Servlet mapping for specific servlets (optional) -->

<servlet>

<servlet-name>RegisterServlet</servlet-name>

<jsp-file>/register.jsp</jsp-file>

</servlet>

<servlet-mapping>

<servlet-name>RegisterServlet</servlet-name>

<url-pattern>/register</url-pattern>

</servlet-mapping>

</web-app>

**5. Deployment Steps**

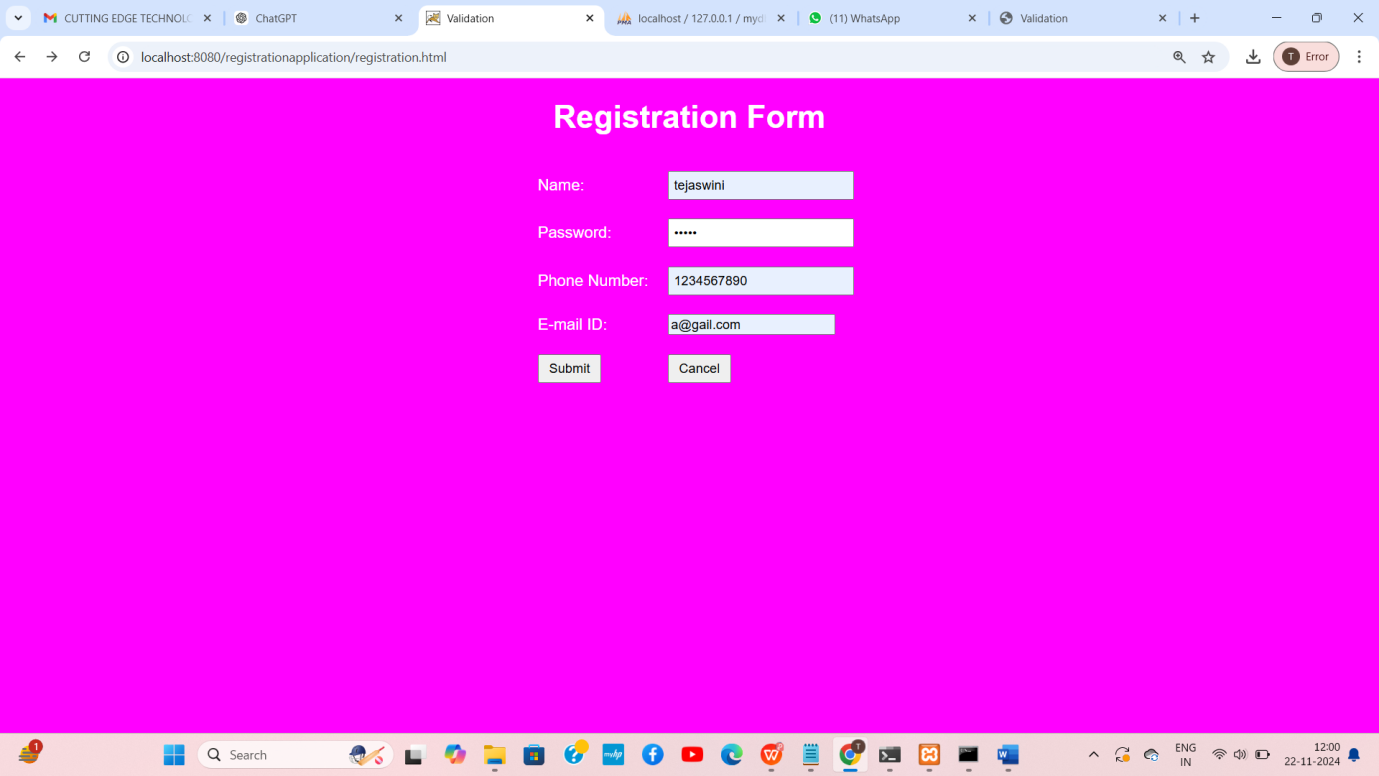
1. Place the web.xml file in the WEB-INF folder.
2. Start the Tomcat server/xamp(start the services-apache, mysql, tomcat)
3. Access the application in the browser:

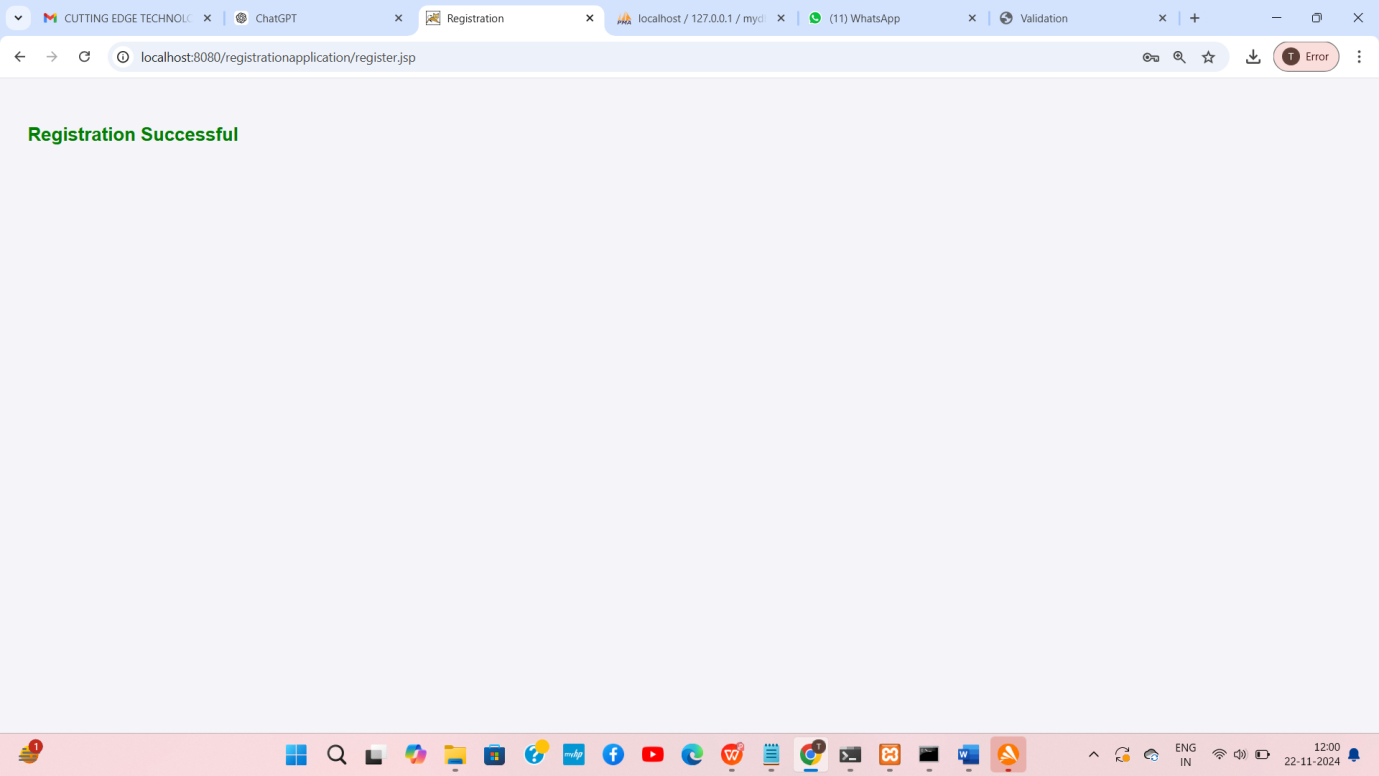
**URL:**

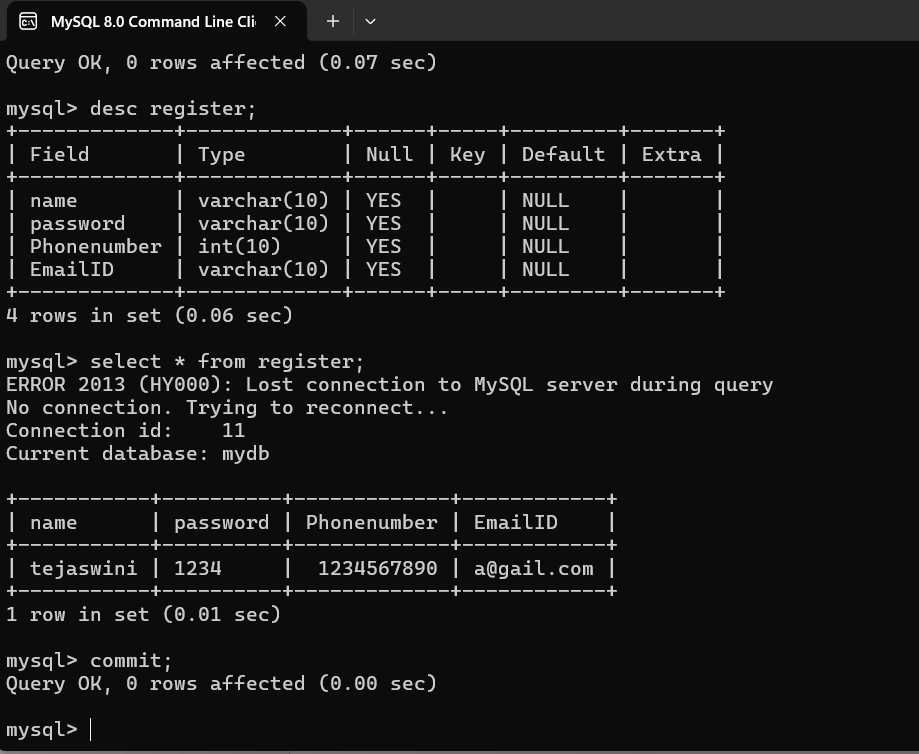
**http://localhost:8080/ registrationapplication/registration.html**

**6. Testing the Application**

1. Open the registration form.
2. Fill in the details and click **submit**.
3. Verify the message displayed after submission.
4. Check the register table in the database for the inserted record.









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

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

**DESCRIPTION:** Session tracking mechanisms are crucial for maintaining the state of a user's interactions with a web application. Two common methods for session tracking are

**Cookies and HTTP Sessions.**

**1. Cookies:** Cookies are small data pieces stored on a user's device by a web browser, used to maintain user-specific information between the client and the server.

2**. HTTP Session:** An HTTP session is a way to store information on the server side between requests from the same client. Each client gets a unique session ID, which is used to retrieve session data.

**Procedure:**

**Step 1: Install Prerequisites**

Ensure Python is installed on your machine. Then, install Flask using pip in cmd:

**pip install flask**

**Step 2: Set Up Project Directory**

**Create a project folder structure like this:**

**session tracking/**

**├── app.py**

**├── templates/**

**│ ├── index.html**

**│ ├── set\_language.html**

**│ ├── index\_session.html**

**Step 3: Code Setup**

**File: app.py**

Copy and paste the Python code into the app.py file:

from flask import Flask, request, render\_template, make\_response, session

app = Flask(\_\_name\_\_)

app.secret\_key = 'super\_secret\_key'  # Set a secret key for session management

# Cookie Example Routes

@app.route('/')

def index():

    # Check if the language cookie is set

    user\_language = request.cookies.get('user\_language')7

    return render\_template('index.html', user\_language=user\_language)

@app.route('/set\_language/<language>')

def set\_language(language):

    # Set the language preference in a cookie

    response = make\_response(render\_template('set\_language.html'))

    response.set\_cookie('user\_language', language)

    return response

# HTTP Session Example Route

@app.route('/session\_example')

def session\_example():

    # Increment the visit count in the session

    session['visit\_count'] = session.get('visit\_count', 0) + 1

    return render\_template('index\_session.html', visit\_count=session['visit\_count'])

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)

**Templates**

**File: templates/index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Cookie Example</title>

</head>

<body>

<h1>Welcome to the website!</h1>

{% if user\_language %}

<p>Your preferred language is: {{ user\_language }}</p>

{% else %}

<p>Your language preference is not set.</p>

{% endif %}

<p><a href="/set\_language/en">Set language to English</a></p>

<p><a href="/set\_language/es">Set language to Spanish</a></p>

</body>

</html>

**File: templates/set\_language.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Set Language</title>

</head>

<body>

<h2>Language set successfully!</h2>

<p><a href="/">Go back to the home page</a></p>

</body>

</html>

**File: templates/index\_session.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Session Example</title>

</head>

<body>

<h1>Welcome to the website!</h1>

<p>This is your visit number: {{ visit\_count }}</p>

</body>

</html>

**Step 4: Run the Application**

1. Open a terminal in the project directory and run:

**python app.py**

1. You should see output like this:

**\* Running on http://127.0.0.1:5000 (Press CTRL+C to quit)**

**Step 5: Access the Application**

**Cookies Example**

1. Open your browser and navigate to:

http://127.0.0.1:5000/

1. Initially, the page will say:  
   *"Your language preference is not set."*
2. Click "Set language to English" or "Set language to Spanish." The preference will be stored in a cookie.
3. Reload the page, and it will display the selected language preference.

**HTTP Session Example**

1. Open your browser and navigate to:

http://127.0.0.1:5000/session\_example

1. The page will display the visit count for the user. Reloading the page will increment the count.

**Output:**

* **For Cookies Example:**
  + First visit:  
    *"Your language preference is not set."*
  + After setting language:  
    *"Your preferred language is: English."*
* **For HTTP Session Example:**
  + First visit:  
    *"This is your visit number: 1."*
  + Reloading:  
    *"This is your visit number: 2."*