

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](#) runtime built on [Chrome's V8 JavaScript engine](#). 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](#) 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

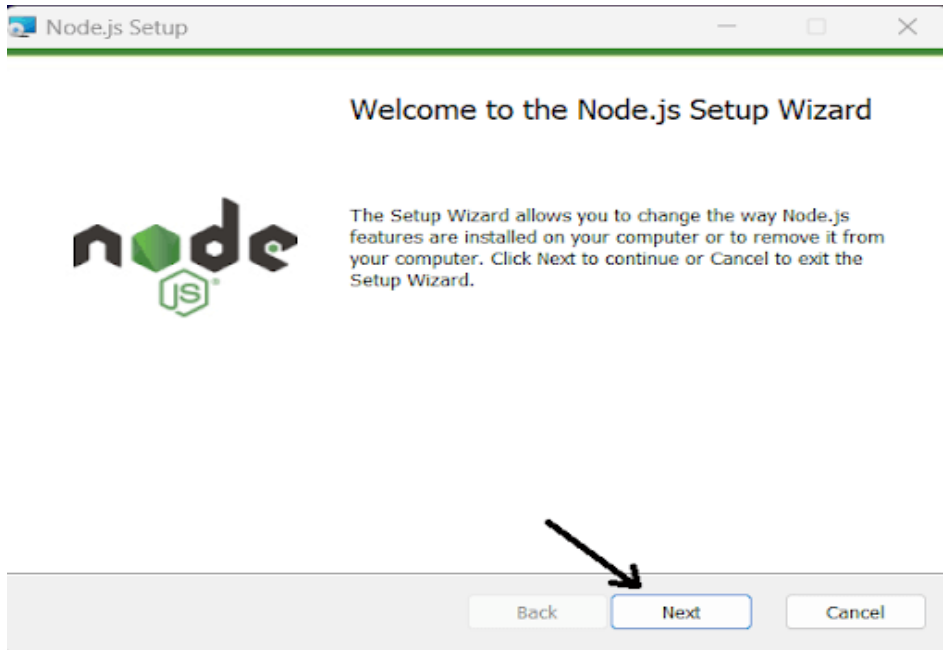
Below are the steps to install NodeJS on your system:

Step 1: To install NodeJS visit [Node.js \(nodejs.org\)](#) and download the LTS (Long Term Support) version.

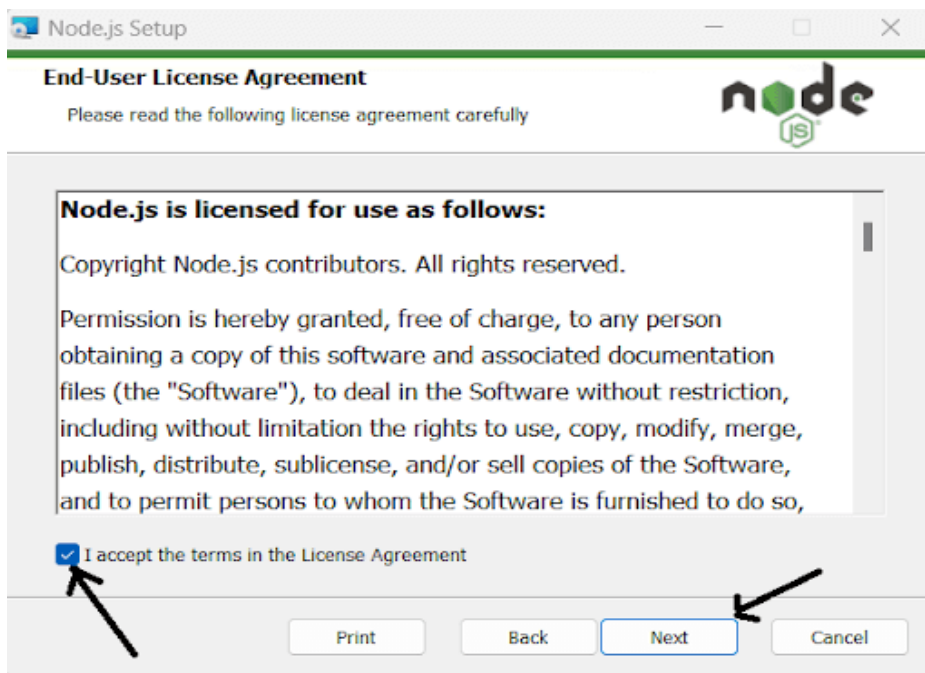


Step 2: Once downloaded, open the ‘.msi’ file. If the system prompts for, ‘Do you want to allow this app to make changes to your device?’ click ‘Yes.’

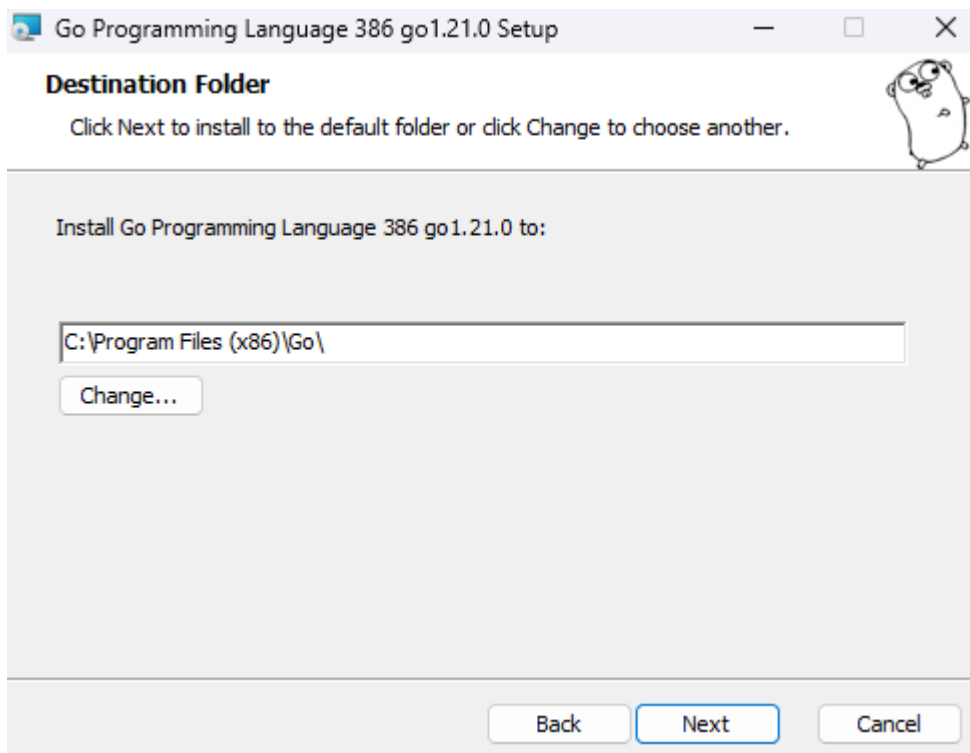
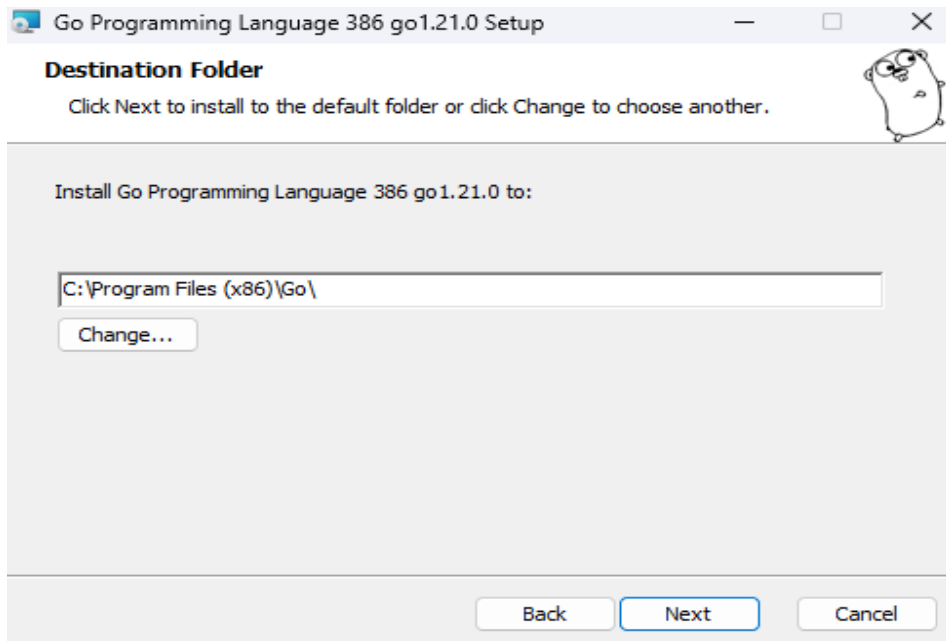
Step 3:After clicking ‘Yes,’ you will see the below prompt. Click ‘Next.’



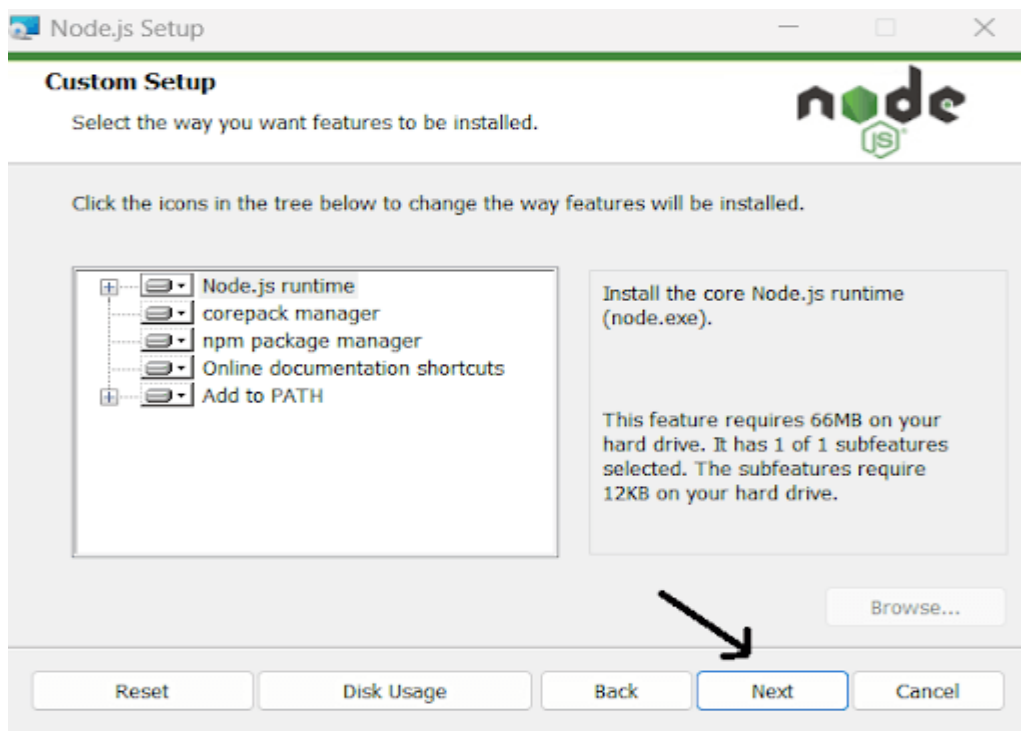
Step 4: Mark the checkbox Accept the agreement and click ‘Next.’



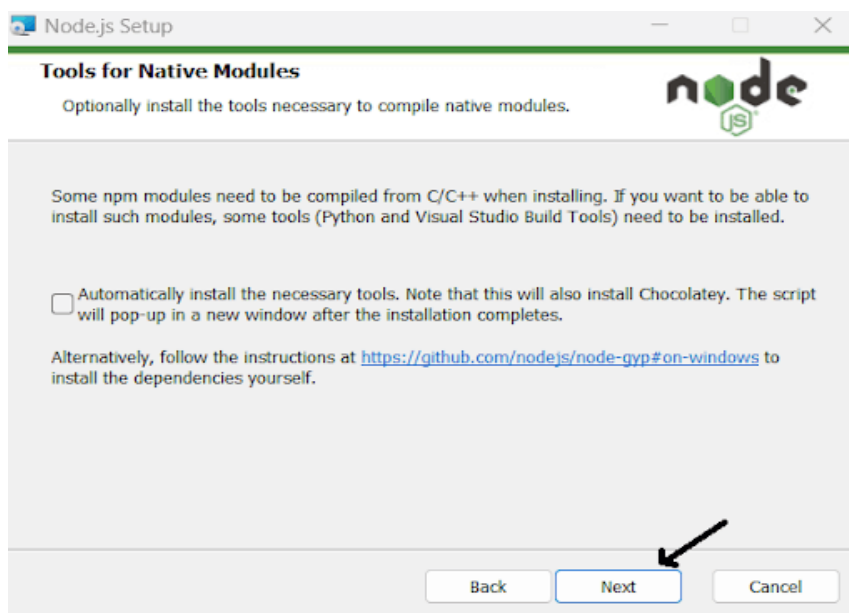
Step 5: Confirm the destination folder and click ‘*Next*.’ It is preferred to keep the settings default.



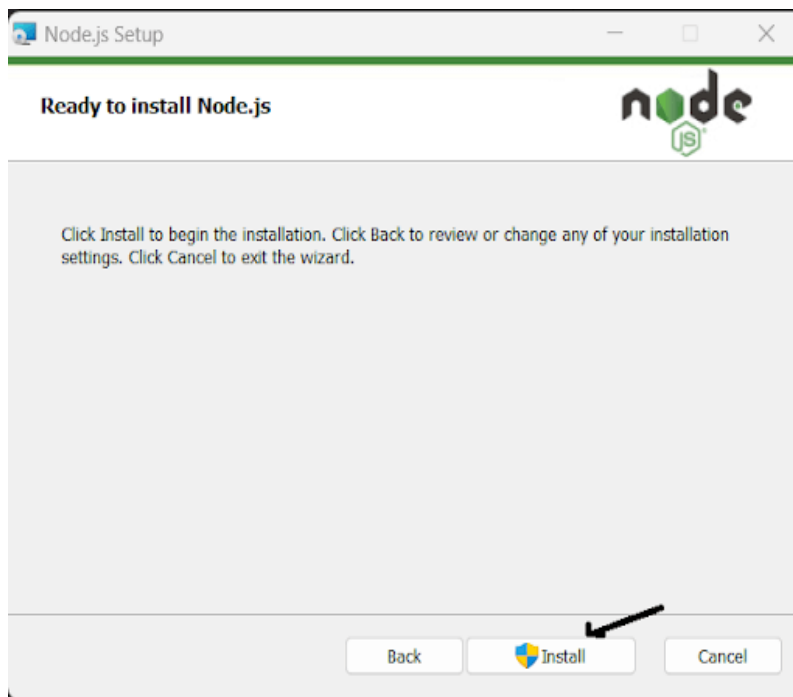
Step 6: Click on 'Next'.



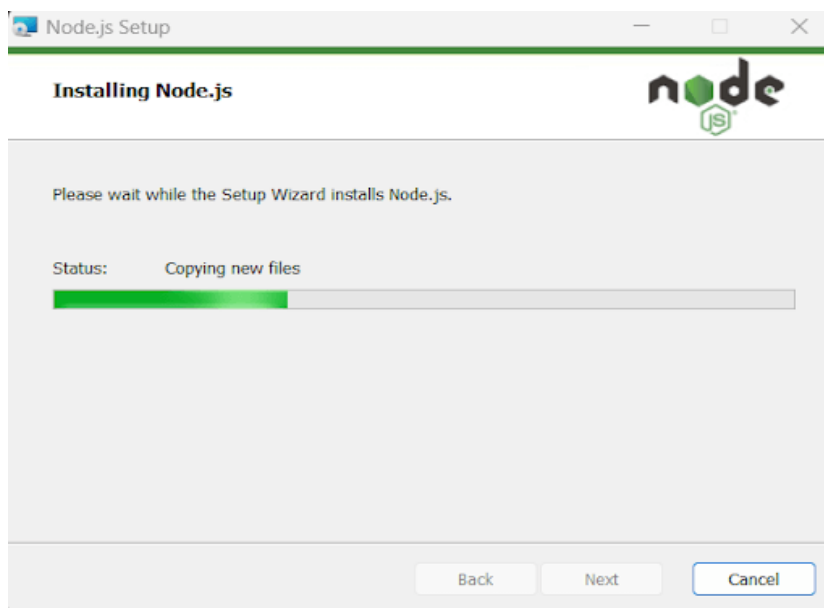
Step 7: If you want to install additional tools to be installed, mark the checkbox and click 'Next.' Else, you can click on 'Next.' (It is recommended to keep the box *unchecked* and click *Next*).



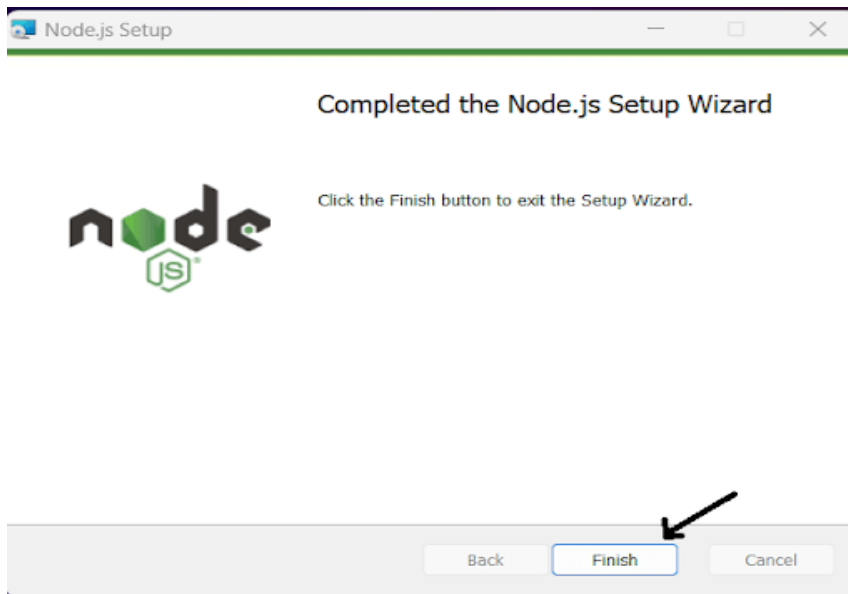
Step 8: Now our NodeJS is ready to install, click '*Install.*'



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

A screenshot of a Windows Command Prompt window. The title bar says 'Command Prompt'. The text inside shows the following commands and outputs:

```
Microsoft Windows [Version 10.0.22621.1848]
(c) Microsoft Corporation. All rights reserved.

C:\Users\DELL>node -v
v18.16.1

C:\Users\DELL>npm -v
9.5.1

C:\Users\DELL>
```

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>

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%"></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/>
            </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%"></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%"></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>
```



```

</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></tr>
              <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;
}

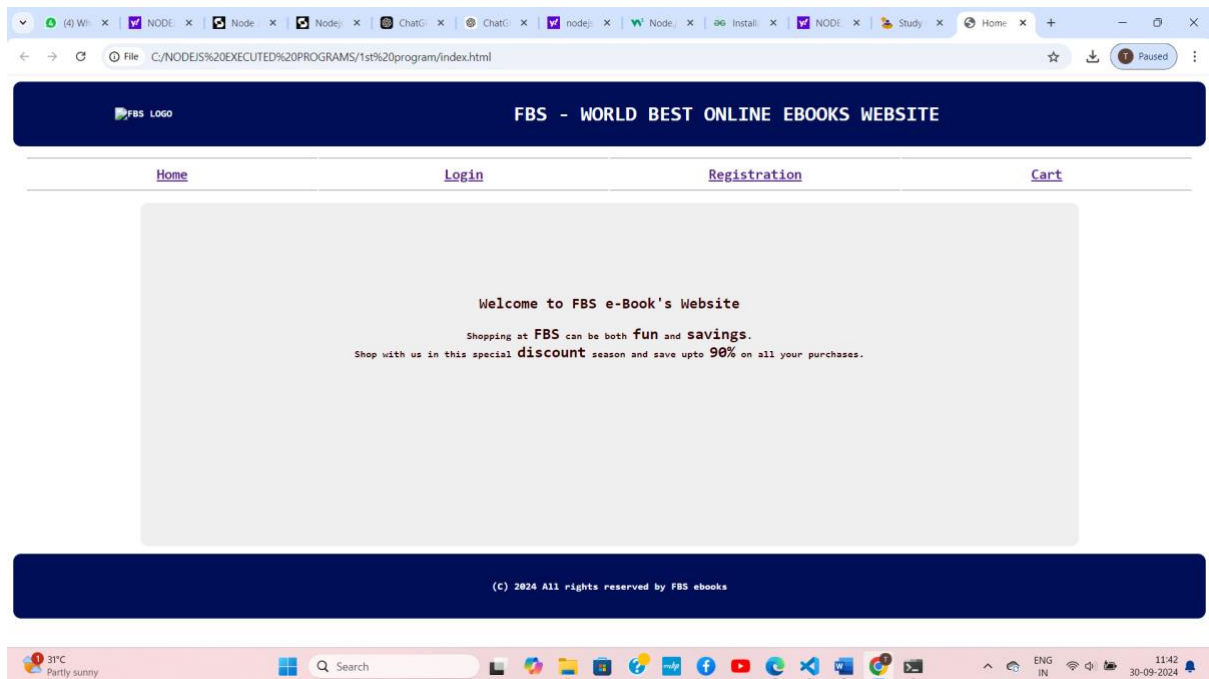
header, nav, main, .sidebar1, .sidebar2, footer{
  display: flex;
  align-items: center;
  justify-content: center;
  border-radius: 10px;
}

.wrapper {
  display: flex;
  flex-direction: column;
  font-weight: 600;
}

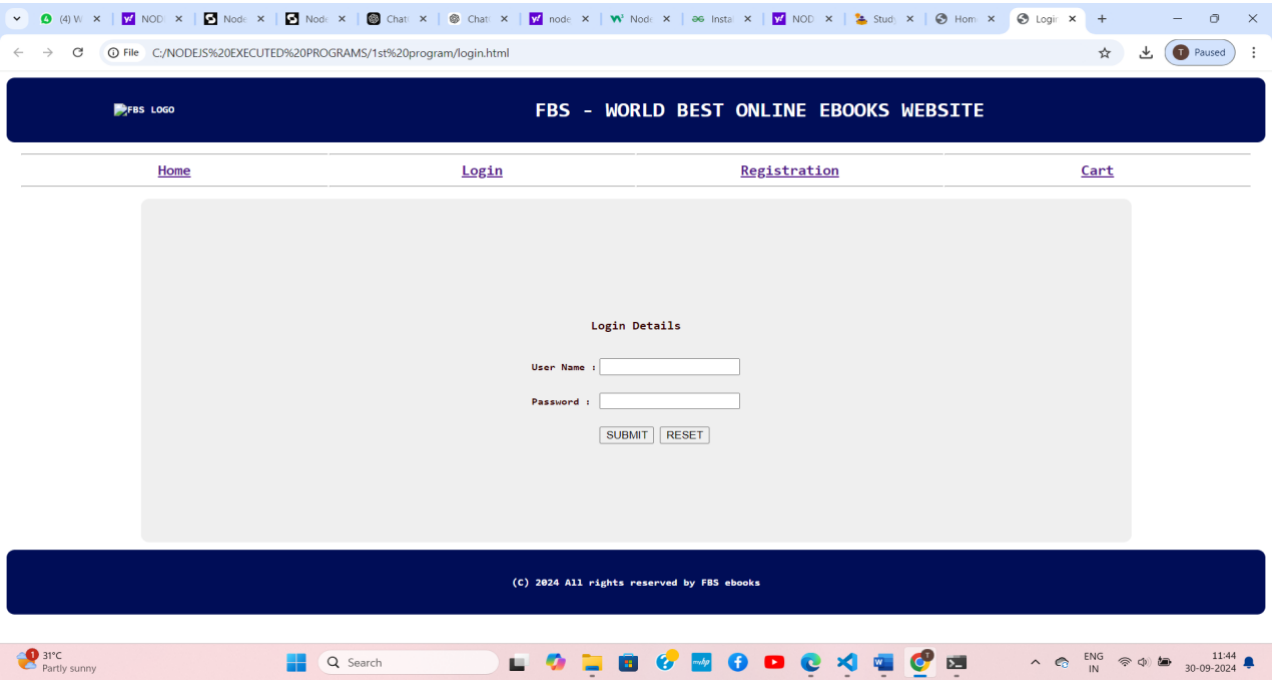
```

Output :

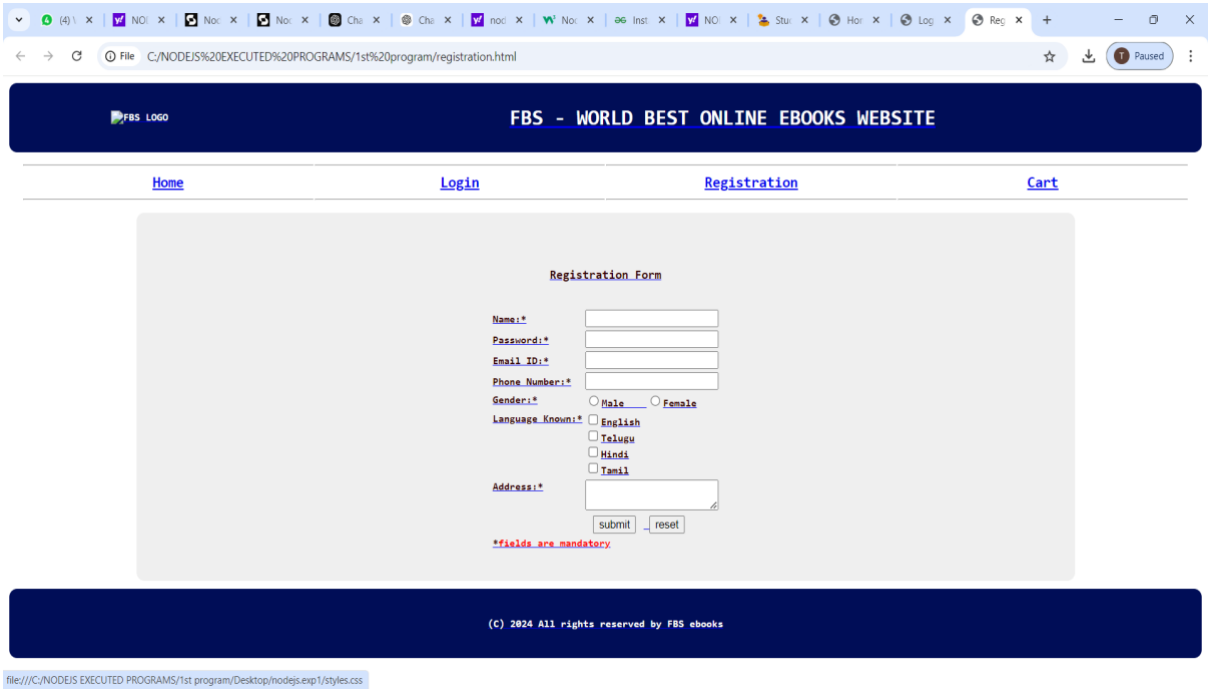
index.html



login.html



registration.html



cart.html

FBS LOGO

FBS - WORLD BEST ONLINE EBOOKS WEBSITE

[Home](#)

[Login](#)

[Registration](#)

[Cart](#)

Cart

BookName	Price	Quantity	Amount
Java Programming	Rs. 2300/-	2	Rs. 4600/-
Web Technologies	Rs. 3000/-	1	Rs. 3000/-
Total Amount:		3	Rs. 7600/-

(C) 2024 All rights reserved by FBS ebooks

31°C
Partly sunny

Search

ENG
IN

11:45
30-09-2024

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();
      }
    }
  </script>
</head>
</html>
```

```

        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");
            fl.email.focus();
            return false;
        }
    }
    // phone number validation
    var phno = fl.phno.value;
    if (phno.length<=0)
    {
        alert("Please Enter Phone Number");
        fl.phno.focus();
        return false;
    }
    if (isNaN(phno))
    {
        alert("Please Enter Valid Phone Number");
        fl.phno.focus();
        return false;
    }
    if (phno.length != 10)
    {
        alert("Please Enter Valid Phone Number");
        fl.phno.focus();
        return false;
    }
    // gender validation
    let flag=false;
    for(i=0;i<fl.gen.length;i++)
        if(fl.gen[i].checked)
            flag=true;
    if(!(flag))
    {
        alert("Please choose a Gender");
        return false;
    }
    // Language validation
    flag=false;
    for(i=0;i<fl.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;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&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

The screenshot shows a web browser window with multiple tabs. The active tab displays a message box that says "This page says Registration Successful" with an "OK" button. Below the message box is a registration form with the following fields:

- Email ID: * (text input with value btejaswini@tkrcet.com)
- Phone Number: * (text input with value 6303385856)
- Gender: * (radio buttons for Male and Female, with Female selected)
- Language Known: * (checkboxes for English, Telugu, Hindi, and Tamil, with English selected)
- Address: * (text input with value Hyderabad)

At the bottom of the form are "SUBMIT" and "RESET" buttons. A red text label below the form states: "*fields are mandatory".

The Windows taskbar at the bottom shows the date and time as 14:27 on 30-09-2024, along with various system icons and a search bar.

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.

XML data to validate

```

9 <bookstore>
10 <book>
11 <title>Introduction to XML</title>
12 <author>John Doe</author>
13 <price>29.99</price>
14 </book>
15 <book>
16 <title>Programming with XML</title>
17 <author>Jane Smith</author>
18 <price>39.99</price>
19 </book>
20 </bookstore>
21

```

XML schema (XSD) data

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3   targetNamespace="http://example.com"
4   xmlns="http://example.com">
5   <xs:element name="root">
6     <xs:complexType>
7       <xs:sequence>
8         <xs:element name="bookstore" type="bookstoreType"/>
9       </xs:sequence>
10    </xs:complexType>
11  </xs:element>
12  <xs:complexType name="bookstoreType">
13    <xs:sequence>

```

Validate

Document 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
const port = 3000;
const hostname = '127.0.0.1';

// Step 5: Listen for requests on the specified port and hostname
server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});

// 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:

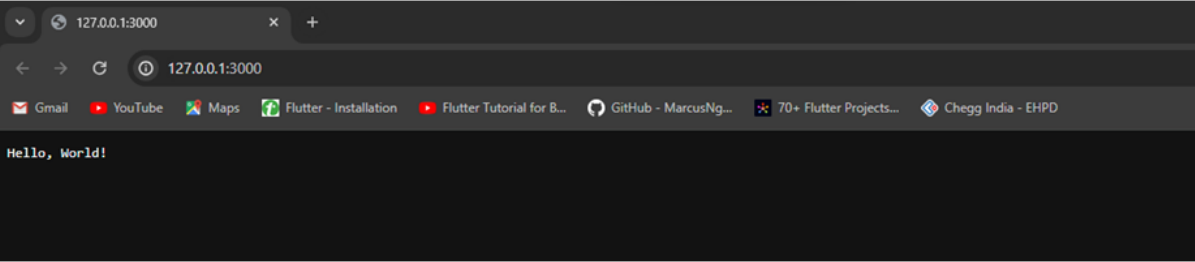
```

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 = '73faea17fde308d431a5bdf83579e00'; // 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.



[Guide](#)
[API](#)
[Dashboard](#)
[Marketplace](#)
[Pricing](#)
[Maps](#)
[Our Initiatives](#)
[Partners](#)
[Blog](#)
[For Business](#)

B.Te...

My services
 My API keys
 My payments
 My profile
 Logout

[Support](#)

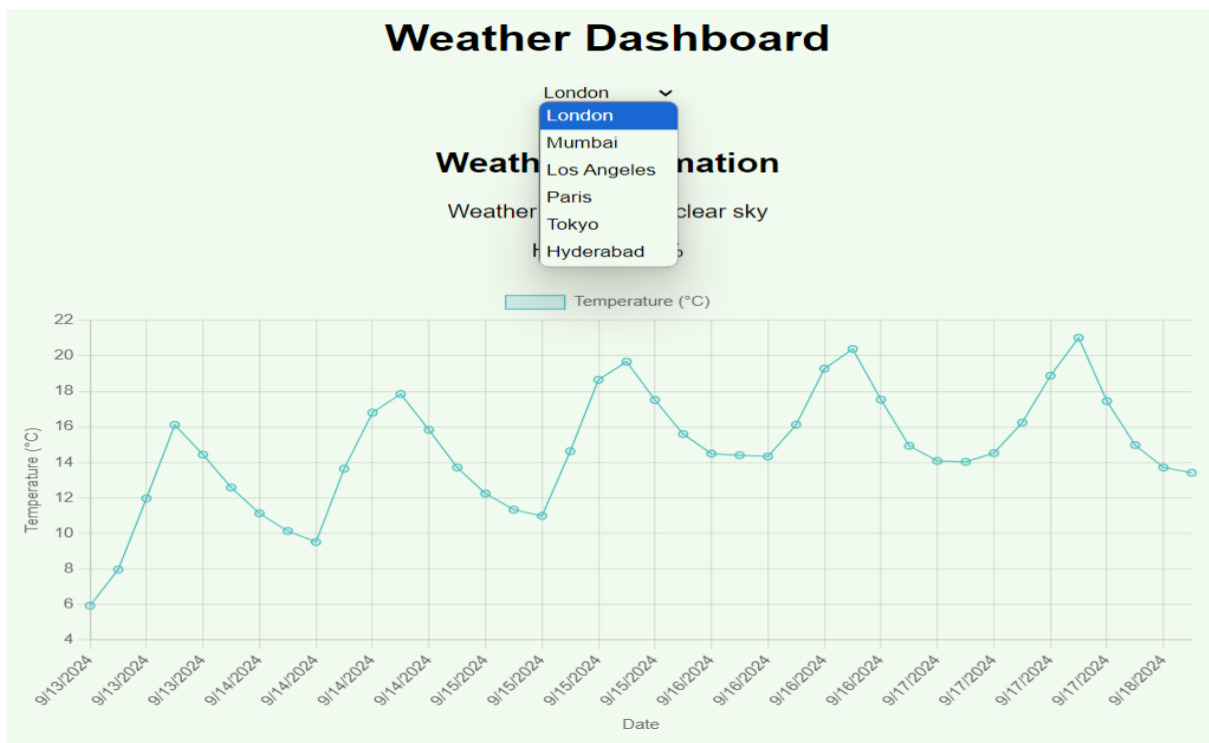
You have to verify your email to use OpenWeatherMap services. Please [click here](#) to get an email with the confirmation link.

[New Products](#)
[Services](#)
[API keys](#)
[Billing plans](#)
[Payments](#)
[Block logs](#)
[My orders](#)
[My profile](#)
[Ask a question](#)

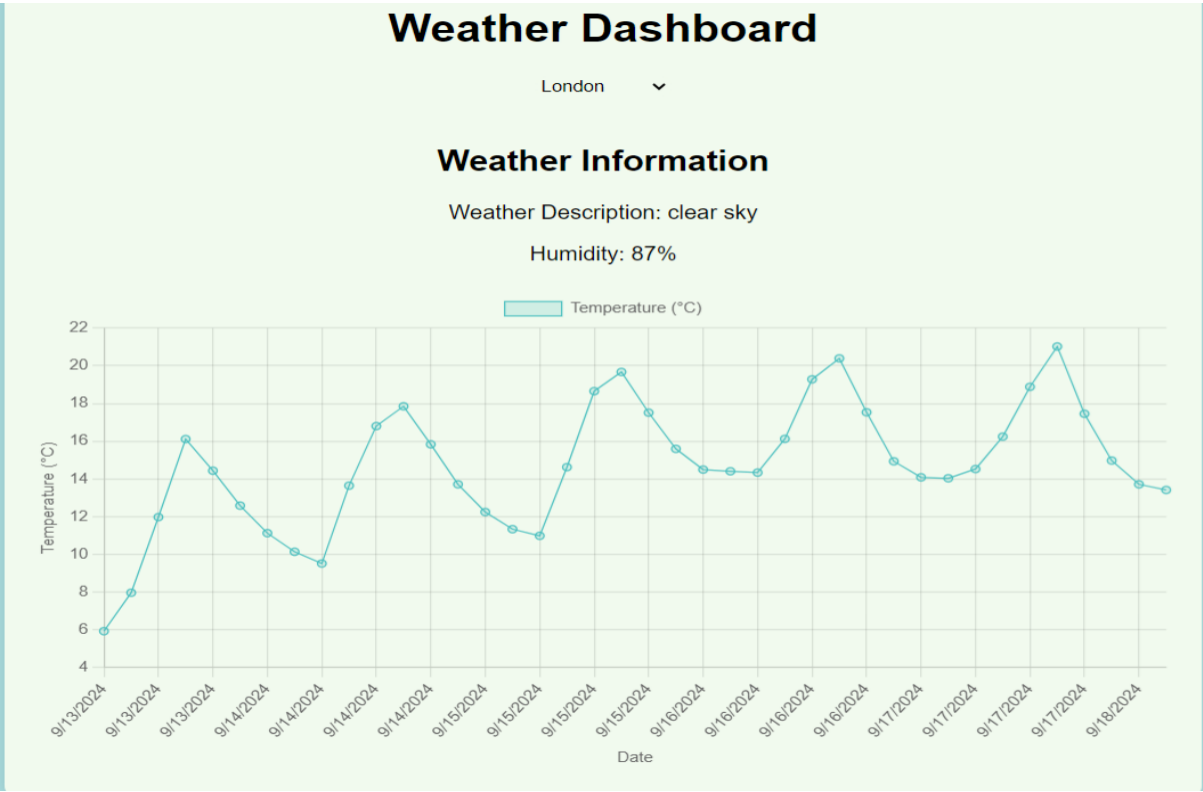
You can generate as many API keys as needed for your subscription. We accumulate the total load from all of them.

Key	Name	Status	Actions	Create key
53c7b2bf313f3fc1d459f217afc8ed49	Default	Active	<div> <div></div> <div></div> </div>	<input type="text" value="API key name"/> <div>Generate</div>

Locate API key
Initially It look Like:



Then, by entering the city then click update chart then



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

AIM: To develop an express web application that can interact with REST API 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.');

```

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 });
  });
});

```

```

});
});

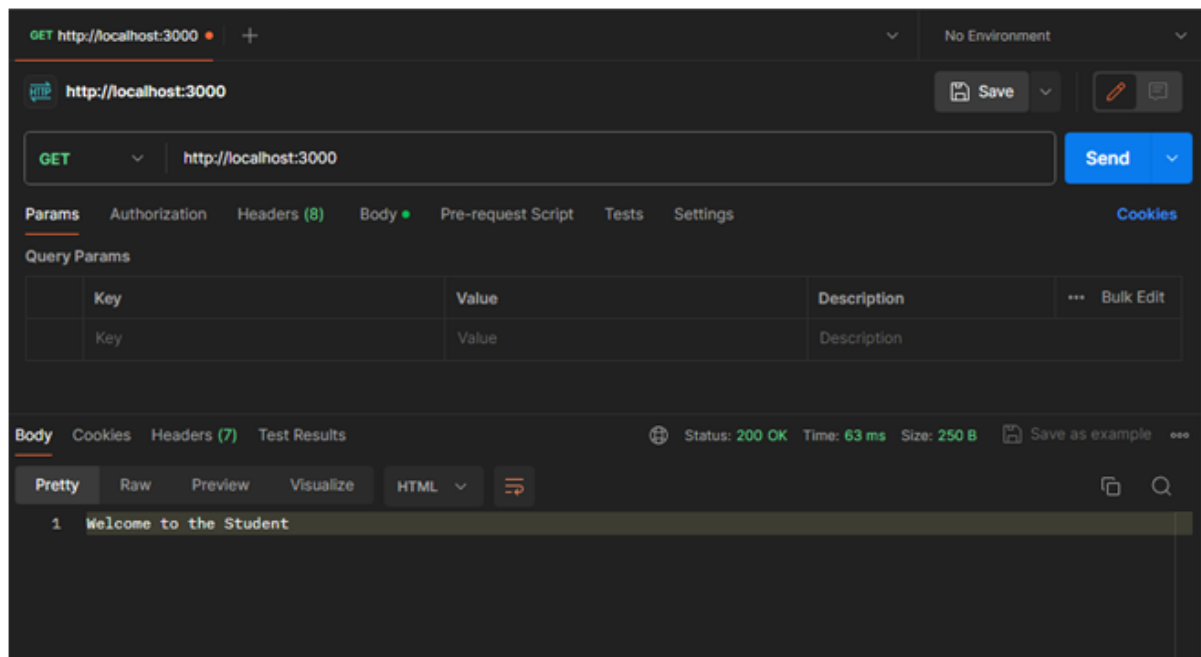
app.listen(port, () => {
  console.log(`Server running at http://localhost:${port}`);
});

```

OUTPUT:

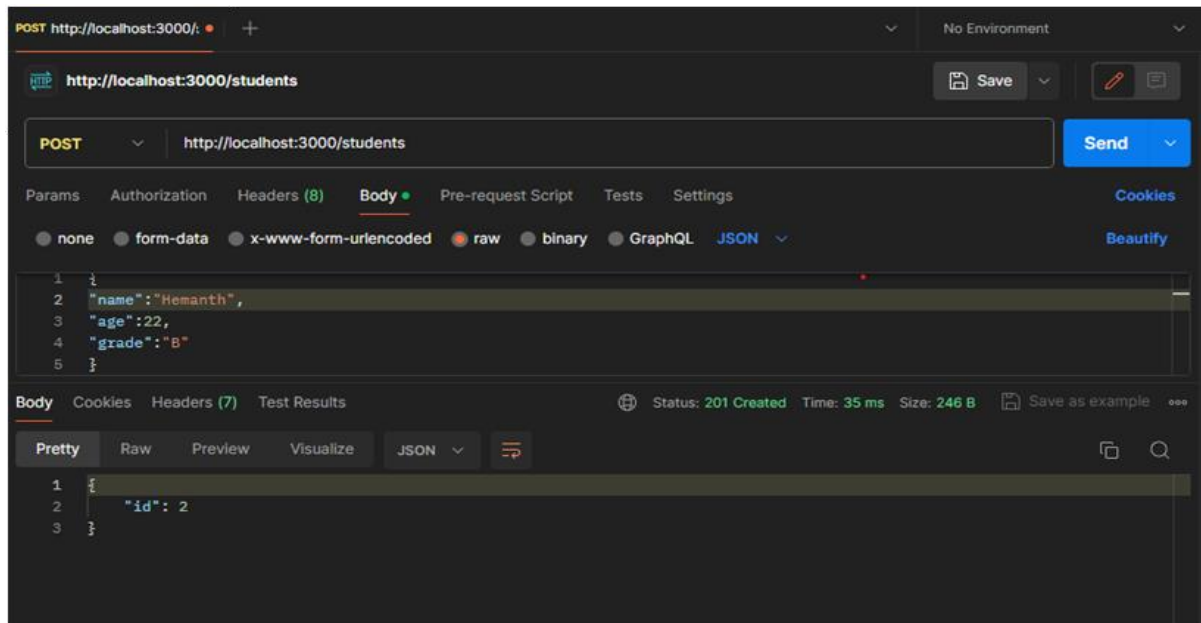
GET:

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



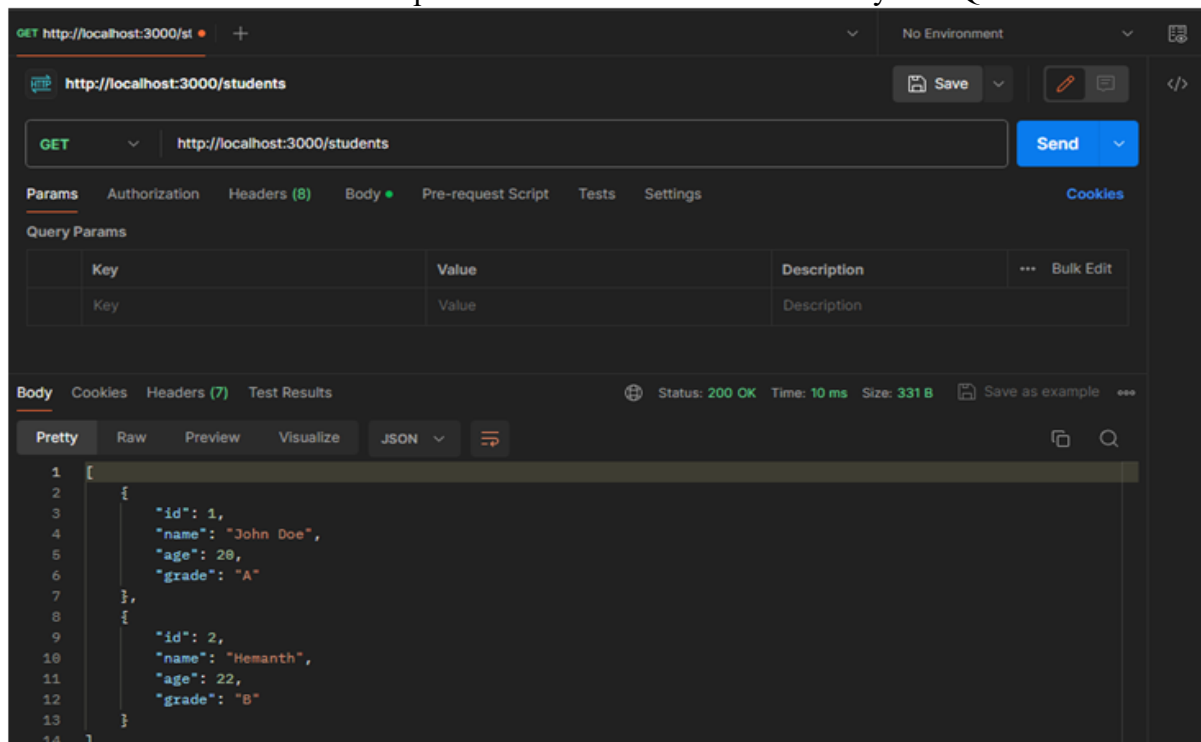
POST: Create a New Student

- 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.



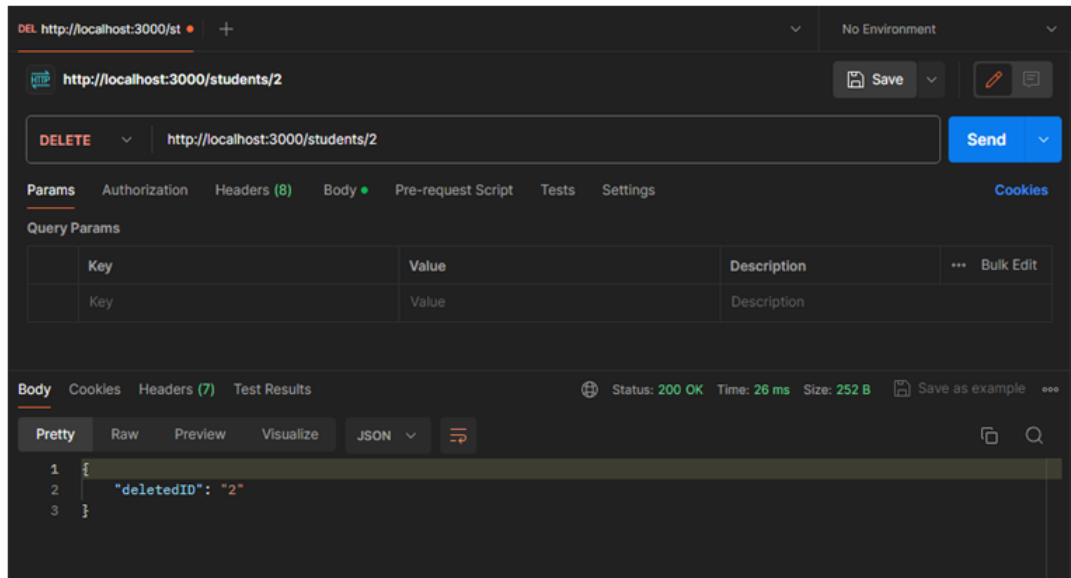
GET: #all Students

- 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.



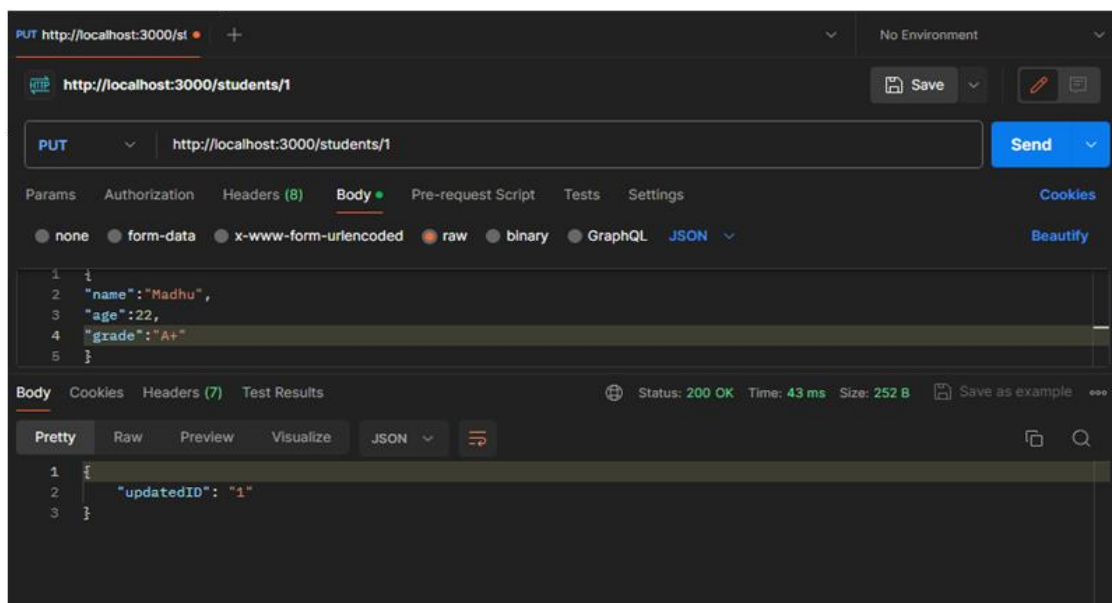
DELETE:

- 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

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.



Activate Windows
Go to Settings to activate Windows.

