Chapter 9

JavaScript in the Browser vs. Node.js

Example: Dropdown Navigation Menu

JavaScript is added to HTML file using <script> Tag.

HTML File: nav-test.html

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width, initial-scale=1">

<link href="styles.css" rel="stylesheet" type="text/css">

<style>

.navbar {

  overflow: hidden;

  background-color: #333;

  font-family: Arial, Helvetica, sans-serif;

}

.navbar a {

  float: left;

  font-size: 16px;

  color: white;

  text-align: center;

  padding: 14px 16px;

  text-decoration: none;

}

.dropdown {

  float: left;

  overflow: hidden;

}

.dropdown .dropbtn {

  cursor: pointer;

  font-size: 16px;

  border: none;

  outline: none;

  color: white;

  padding: 14px 16px;

  background-color: inherit;

  font-family: inherit;

  margin: 0;

}

.navbar a:hover, .dropdown:hover .dropbtn, .dropbtn:focus {

  background-color: red;

}

.dropdown-content {

  display: none;

  position: absolute;

  background-color: #f9f9f9;

  min-width: 160px;

  box-shadow: 0px 8px 16px 0px rgba(0,0,0,0.2);

  z-index: 1;

}

.dropdown-content a {

  float: none;

  color: black;

  padding: 12px 16px;

  text-decoration: none;

  display: block;

  text-align: left;

}

.dropdown-content a:hover {

  background-color: #ddd;

}

.show {

  display: block;

}

</style>

<header>

    <h1>DES Pune University, Pune !</h1> <br>

<h4>Simple Responsive Webpage</h4>

</header>

</head>

<body style="background-color:white;">

<div class="navbar">

  <a href="#home">Home</a>

  <a href="#news">About</a>

  <a href="#news">Contact</a>

  <div class="dropdown">

  <button class="dropbtn" onclick="myFunction()">Schools ▼

    <i class="fa fa-caret-down"></i>

  </button>

  <div class="dropdown-content" id="myDropdown">

    <a href="#">Engineering and Technology</a>

<a href="#">Science and Mathematics</a>

<a href="#">Humanities and Social Sciences</a>

<a href="#">Business and Management</a>

<a href="#">Design and Art</a>

</div>

  </div>

</div>

<h3>Dropdown Menu inside a Navigation Bar</h3>

<p>Click on the "Dropdown" link to see the dropdown menu.</p>

<section>

    <h2><center>Welcome to Our Website!</center></h2>

</section>

<section>

    <h2>About Us</h2>

    <p><h3>Hello!! Welcome to DES PU, Pune University !!!

      <br> Become a part of the 139 years legacy!!</h3></p>

</section>

<script>

/\* When the user clicks on the button,

toggle between hiding and showing the dropdown content \*/

function myFunction() {

  document.getElementById("myDropdown").classList.toggle("show");

}

// Close the dropdown if the user clicks outside of it

window.onclick = function(e) {

  if (!e.target.matches('.dropbtn')) {

  var myDropdown = document.getElementById("myDropdown");

    if (myDropdown.classList.contains('show')) {

      myDropdown.classList.remove('show');

    }

  }

}

</script>

<footer>

    <p>Copyright © 2024 DES, Pune University</p>

</footer>

</body>

</html>

CSS File: styles.css (For Overall webpage styling effect)

/\* Reset default margin and padding \*/

\* {

    margin: 0;

    padding: 0;

    box-sizing: border-box;

}

/\* Basic styling \*/

body {

    font-family: Arial, sans-serif;

    line-height: 1.6;

    /\*background-color: #f0f0f0;\*/

    margin: 0;

    padding: 0;

}

header {

    background-color: #FFA500;

    color:  #800000;

    text-align: center;

    padding: 1em 0;

}

section {

    background-color:       #F5DEB3;

    margin-bottom: 0.2em;

    padding: 3em;

    flex: 1 1 100%;

    font-family:cambria;

}

section h2 {

    color: #800000;

}

footer {

   /\* background-image: url('DESPU.png');\*/

    text-align: center;

    padding: 0.6em 0;

    background-color:   #4B0082;

    color: #fff;

}

/\* Responsive design \*/

@media (min-width: 768px) {

    main {

        flex-wrap: nowrap;

    }

    section {

        flex: 1 1 calc(50% - 20px);

        margin-right: 16px;

    }

    section:last-child {

        margin-right: 0;

    }

}

@media (min-width: 1024px) {

    section {

        flex: 1 1 calc(33.33% - 20px);

    }

}

1. **Event Handling:**

Adding click functionality to a button.

**HTML File: button-event.html**

<!DOCTYPE html>

<html>

<head>

<button id="btn">Click Me!</button>

<script>

    document.getElementById('btn').addEventListener('click', () => {

        alert('Button clicked!');

    });

</script>

</body>

</html>

**Example: Real-Time Weather Updates**

This example fetches current weather information for a city using the OpenWeatherMap API and dynamically updates the page content.

**HTML File: button-event.html**

<!DOCTYPE html>

<html>

<head>

<meta charset='utf-8'>

<meta http-equiv='X-UA-Compatible' content='IE=edge'>

<title>Dash Board</title>

<meta name='viewport' content='width=device-width,initial-scale=1'>

</head>

<body>

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

<link rel="stylesheet" href=

"https://cdnjs.cloudflare.com/ajax/libs/animate.css/4.1.1/animate.min.css">

<link rel="stylesheet" href=

"https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.1/css/all.min.css">

<link rel="stylesheet" href=

"https://fonts.googleapis.com/css2?family=Montserrat:wght@400;700&display=swap">

</head>

<div class="container">

<div class="weather-card">

<h3>

Weather DashBoard

</h3>

<input type="text" id="city-input"

placeholder="Enter city name">

<button id="city-input-btn" onclick="weatherFn($('#city-input').val())">

Get Weather

</button>

<div id="weather-info" class="animate\_\_animated animate\_\_fadeIn">

<h3 id="city-name"></h3>

<p id="date"></p>

<p id="temperature"></p>

<p id="description"></p>

</div>

</div>

</div>

<script src= "https://code.jquery.com/jquery-3.6.0.min.js">

</script>

<script src= "https://momentjs.com/downloads/moment.min.js">

</script>

<script>

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

const apiKey = 'f00c38e0279b7bc85480c3fe775d518c'; **//Do not forget to replace your API Key**

$(document).ready(function () {

weatherFn('');

});

async function weatherFn(cName) {

const temp = `${url}?q=${cName}&appid=${apiKey}&units=metric`;

try {

const res = await fetch(temp);

const data = await res.json();

if (res.ok) {

weatherShowFn(data);

} else {

alert('City not found Please try again');

}

} catch (error) {

console.error('Error fetching weather data:-', error);

}

}

function weatherShowFn(data) {

$('#city-name').text(data.name);

$('#date').text(moment().format('MMMM Do YYYY, h:mm:ss a'));

$('#temperature').html(`${data.main.temp}°C`);

$('#description').text(data.weather[0].description);

$('#weather-icon').attr('src',`...`);

$('#weather-info').fadeIn();

}

</script>

</body>

</html>

**Example to Illustrate the Differences**

**Browser JavaScript: Fetching and Displaying Data**

**differ.html**

<!DOCTYPE html>

<html>

<head>

<title>Browser JavaScript Example</title>

</head>

<body>

<div id="data"></div>

<script>

fetch('https://dog.ceo/api/breeds/image/random')

.then(response => response.json())

.then(data => {

document.getElementById('data').innerText = JSON.stringify(data, null, 2);

})

.catch(error => {

console.error('Error fetching data:', error);

});

</script>

</body>

</html>

**Node.js: Fetching Data from API**

const https = require('https');

https.get('https://dog.ceo/api/breeds/image/random', (res) => {

let data = '';

res.on('data', (chunk) => {

data += chunk;

});

res.on('end', () => {

console.log(JSON.parse(data));

});

}).on('error', (err) => {

console.error('Error:', err.message);

});

To view the output of a Node.js file, save the code in a file named getdata.js. Ensure that Node.js is installed on your system. (Details about installing Node.js are covered in the section titled “Using JavaScript Modules in Node.js (require, import)“.) To execute the file, open the Terminal window in Visual Studio Code and run the command:

node getdata.js

**Syntax**

* **Importing a Module**

const moduleName = require('module');

* **Exporting a Module**

module.exports = valueToExport;

**Example: CommonJS**

**File: mathOperations.js**

// Define functions to export

function add(a, b) {

return a + b;

}

function subtract(a, b) {

return a - b;

}

// Export functions

module.exports = {

add,

subtract

};

**File: app.js**

// Import the module

const math = require('./mathOperations');

// Use the exported functions

console.log('Addition:', math.add(5, 3)); // Output: 8

console.log('Subtraction:', math.subtract(5, 3)); // Output: 2

**Execution:**

node app.js

**ES Modules (import)**

ES Modules are the modern module system introduced in ECMAScript 2015 (ES6). They use import and export keywords.

**Syntax**

* **Importing a Module**

import moduleName from 'module';

* **Exporting from a Module**

export const functionName = () => { ... };

export default functionName;

**Example: ES Modules**

**File: mathOperations.mjs**

// Define functions to export

export function add(a, b) {

return a + b;

}

export function subtract(a, b) {

return a - b;

}

**File: app.mjs**

// Import the module

import { add, subtract } from './mathOperations.mjs';

// Use the imported functions

console.log('Addition:', add(5, 3)); // Output: 8

console.log('Subtraction:', subtract(5, 3)); // Output: 2

**Execution:**

node --experimental-modules app.mjs

**Using Built-in Node.js Modules**

**Example: Using fs Module**

const fs = require('fs');

// Read a file

fs.readFile('example.txt', 'utf8', (err, data) => {

if (err) throw err;

console.log(data);

});

**Importing the fs Module**

To use the fs module, import it into your Node.js script:

const fs = require('fs');

**Reading Files**

You can read files asynchronously or synchronously using fs.

**Asynchronous File Reading**

fs.readFile('example.txt', 'utf8', (err, data) => {

if (err) {

console.error('Error reading file:', err);

return;

}

console.log('File content:', data);

});

* **Parameters:**
  + 'example.txt': The file to read.
  + 'utf8': Encoding format.
  + callback: A function to handle the data or error.

**Synchronous File Reading**

try {

const data = fs.readFileSync('example.txt', 'utf8');

console.log('File content:', data);

} catch (err) {

console.error('Error reading file:', err);

}

**Writing Files**

Similar to reading, you can write files asynchronously or synchronously.

**Asynchronous File Writing**

fs.writeFile('output.txt', 'Hello, World!', (err) => {

if (err) {

console.error('Error writing file:', err);

return;

}

console.log('File written successfully!');

});

**Synchronous File Writing**

try {

fs.writeFileSync('output.txt', 'Hello, World!');

console.log('File written successfully!');

} catch (err) {

console.error('Error writing file:', err);

}

**Appending to Files**

To add content to an existing file, use fs.appendFile or fs.appendFileSync.

**Asynchronous Appending**

fs.appendFile('output.txt', '\nAppending new content!', (err) => {

if (err) {

console.error('Error appending to file:', err);

return;

}

console.log('Content appended successfully!');

});

**Synchronous Appending**

try {

fs.appendFileSync('output.txt', '\nAppending new content!');

console.log('Content appended successfully!');

} catch (err) {

console.error('Error appending to file:', err);

}

**Deleting Files**

Use fs.unlink to delete files.

**Asynchronous Deletion**

fs.unlink('output.txt', (err) => {

if (err) {

console.error('Error deleting file:', err);

return;

}

console.log('File deleted successfully!');

});

**Synchronous Deletion**

try {

fs.unlinkSync('output.txt');

console.log('File deleted successfully!');

} catch (err) {

console.error('Error deleting file:', err);

}

**Working with Directories**

The fs module also supports directory operations.

**Creating Directories**

fs.mkdir('newDir', (err) => {

if (err) {

console.error('Error creating directory:', err);

return;

}

console.log('Directory created successfully!');

});

**Reading Directories**

fs.readdir('.', (err, files) => {

if (err) {

console.error('Error reading directory:', err);

return;

}

console.log('Directory contents:', files);

});

**Removing Directories**

fs.rmdir('newDir', (err) => {

if (err) {

console.error('Error removing directory:', err);

return;

}

console.log('Directory removed successfully!');

});

**Watching for File Changes**

The fs.watch method allows you to monitor file or directory changes.

fs.watch('example.txt', (eventType, filename) => {

console.log(`Event Type: ${eventType}`);

console.log(`File Changed: ${filename}`);

});

**Error Handling in fs Operations**

Always handle errors while working with the file system to avoid unexpected crashes. Use try...catch blocks for synchronous operations and error callbacks for asynchronous operations.

**Example: A Simple File Operations Script**

const fs = require('fs');

// Writing to a file

fs.writeFile('test.txt', 'Hello, Node.js!', (err) => {

if (err) throw err;

console.log('File created!');

// Reading the file

fs.readFile('test.txt', 'utf8', (err, data) => {

if (err) throw err;

console.log('File content:', data);

// Appending to the file

fs.appendFile('test.txt', '\nAppended content.', (err) => {

if (err) throw err;

console.log('Content appended.');

// Deleting the file

fs.unlink('test.txt', (err) => {

if (err) throw err;

console.log('File deleted.');

});

});

});

});

**Importing the http Module**

To create an HTTP server, start by importing the http module:

const http = require('http');

**Creating the Server**

Use the http.createServer() method to create a server. It accepts a callback function with two parameters:

* req (request): Represents the incoming request from the client.
* res (response): Represents the response that will be sent back to the client.

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

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

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

});

**Listening on a Port**

Once the server is created, use the server.listen() method to make it listen on a specific port:

server.listen(3000, () => {

console.log('Server is running on http://localhost:3000');

});

**Complete Example**

Here’s a full example of a simple HTTP server – httpserver.html:

const http = require('http');

// Create the server

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

// Set response header

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

// Send response

res.end('Welcome to my Node.js server!');

});

// Start the server

server.listen(3000, () => {

console.log('Server is running on http://localhost:3000');

});

**Handling Different Routes**

You can use the req.url property to handle different routes dynamically.

**Example:**

const http = require('http');

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

if (req.url === '/') {

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

res.end('Home Page');

} else if (req.url === '/about') {

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

res.end('About Page');

} else {

res.writeHead(404, { 'Content-Type': 'text/plain' });

res.end('Page Not Found');

}

});

server.listen(3000, () => {

console.log('Server is running on http://localhost:3000');

});

**Serving HTML Content**

To send HTML content, set the Content-Type header to text/html.

**Example:**

const http = require('http');

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

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

res.end('<h1>Welcome to my Node.js Server</h1><p>This is a simple HTML response.</p>');

});

server.listen(3000, () => {

console.log('Server is running on http://localhost:3000');

});

**Serving JSON Data**

To serve JSON data, set the Content-Type header to application/json.

**Example:**

const http = require('http');

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

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

const data = { message: 'Hello, World!', timestamp: Date.now() };

res.end(JSON.stringify(data));

});

server.listen(3000, () => {

console.log('Server is running on http://localhost:3000');

});

**Handling Query Parameters**

You can use the url module to parse query parameters from the request URL.

**Example:**

const http = require('http');

const url = require('url');

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

const parsedUrl = url.parse(req.url, true);

const name = parsedUrl.query.name || 'Guest';

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

res.end(`Hello, ${name}!`);

});

server.listen(3000, () => {

console.log('Server is running on http://localhost:3000');

});

**Stopping the Server**

You can stop the server programmatically using the server.close() method:

server.close(() => {

console.log('Server has been stopped.');

});

**Error Handling**

Use the error event to handle server errors gracefully:

server.on('error', (err) => {

console.error('Server error:', err);

});

**Understanding package.json**

Example:

{

"name": "my-node-app",

"version": "1.0.0",

"description": "A simple Node.js project",

"main": "index.js",

"dependencies": {

"express": "^4.17.1"

},

"devDependencies": {

"nodemon": "^2.0.7"

},

"scripts": {

"start": "node index.js",

"dev": "nodemon index.js"

}

}

**Working with Scripts**

Scripts in package.json allow you to define custom commands:

"scripts": {

"start": "node index.js",

"test": "echo \"No test specified\""

}

Run a script using:

npm run <script-name>

**Example:**

npm start

**Example**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Client-Side Example</title>

<script>

async function fetchData() {

const response = await fetch('https://jsonplaceholder.typicode.com/posts');

const data = await response.json();

const list = document.getElementById('data-list');

data.forEach(post => {

const li = document.createElement('li');

li.textContent = post.title;

list.appendChild(li);

});

}

</script>

</head>

<body>

<h1>Posts</h1>

<ul id="data-list"></ul>

<button onclick="fetchData()">Load Posts</button>

</body>

</html>

**Server-Side JavaScript in Full-Stack Applications**

**Example**

// Import necessary modules

const express = require('express');

const app = express();

const PORT = 3000;

// Middleware to parse JSON

app.use(express.json());

// Sample route

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

res.json([

{ id: 1, title: 'First Post' },

{ id: 2, title: 'Second Post' },

]);

});

// Start the server

app.listen(PORT, () => {

console.log(`Server is running on http://localhost:${PORT}`);

});

**Integration of Front-End and Back-End**

**Example**

**Back-End (Node.js + Express)**

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

res.json([{ id: 1, name: 'Rupali' }, { id: 2, name: 'Mahesh' }]);

});

**Front-End (React)**

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

function App() {

const [users, setUsers] = useState([]);

useEffect(() => {

fetch('/api/users')

.then((response) => response.json())

.then((data) => setUsers(data));

}, []);

return (

<div>

<h1>User List</h1>

<ul>

{users.map((user) => (

<li key={user.id}>{user.name}</li>

))}

</ul>

</div>

);

}

export default App;

**Database Integration**

JavaScript frameworks and libraries facilitate seamless database integration.

**MongoDB with Mongoose**

const mongoose = require('mongoose');

mongoose.connect('mongodb://localhost/mydb', { useNewUrlParser: true, useUnifiedTopology: true });

const User = mongoose.model('User', { name: String });

const user = new User({ name: 'Rupali Chopade' });

user.save().then(() => console.log('User saved'));

**MySQL with Sequelize**

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

const sequelize = new Sequelize('mysql://user:password@localhost:3306/mydb');

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

name: {

type: DataTypes.STRING,

allowNull: false,

},

});

sequelize.sync().then(() => console.log('Database synced'));