**Ex No: 1 Date:**

# Static Node.js Server (without Express)

## AIM:

To create a NodeJS server that serves static HTML and CSS files to the user without using Express.

### ALGORITHM:

**Step 1:** Import required core modules: http, fs, and path.

**Step 2:** Create an HTTP server using http.createServer().

**Step 3:** Parse the request URL to determine which file is being requested. **Step 4:** Use path.join() to resolve the full path of the requested file. **Step 5:** Check the file extension and set appropriate content-type headers. **Step 6:** Use fs.readFile() to read the file from the file system.

**Step 7:** Send the file content back in the response.

**Step 8:** Handle 404 errors for missing files.

**Step 9:** Handle 500 errors for server or file read issues.

**Step 10:** Start the server using server.listen(PORT) and test in the browser

**Step 11:** Create a dedicated public directory to organize HTML, CSS, and JS files separately from the server code.

**Step 12:** Set a default file (like index.html) to serve when no specific file is requested (i.e., when the URL is just /).

**Step 13:** Use fs.existsSync() or check fs.stat() to verify the file exists before attempting to read it, for more graceful error handling.

**Step 14:** Ensure MIME types (like .css, .js, .png) are handled dynamically using a content-type map or mime module for scalability.

**Step 15:** Log each request (e.g., method, URL, status) in the console to monitor server activity and aid in debugging.

### DESIGN:

****

1. The program uses modular architecture by importing Node.js core modules like http, fs, and path, avoiding external dependencies for lightweight execution.
2. It maps incoming HTTP requests to corresponding file paths using path.join() and URL parsing, which enables flexible routing of static files.
3. The server identifies file extensions and dynamically sets the appropriate Content-Type headers, ensuring correct rendering of HTML, CSS, JavaScript, and image files in the browser.
4. Static assets are served from a dedicated public directory, maintaining a clear separation between server logic and user interface content.
5. The design incorporates robust error handling for both 404 (File Not Found) and 500

(Internal Server Error), enhancing user experience and simplifying debugging.

### IMPLEMENTATION:

Index.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

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

<title>Student Login</title>

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

</head>

<body>

<div class="login-container">

<h2> Student Login</h2>

<form id="loginForm">

<label for="email">Email:</label>

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

<label for="rollno">Roll Number:</label>

<input type="text" name="rollno" id="rollno" required />

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

</form>

</div>

<script src="script.js"></script>

</body>

</html>

Style.css

body {

margin: 0;

font-family: 'Segoe UI', sans-serif;

background: linear-gradient(to right, #83a4d4, #b6fbff); display: flex;

justify-content: center; align-items: center; height: 100vh;

}

.login-container { background: #fff; padding: 40px 30px; border-radius: 10px;

box-shadow: 0 10px 25px rgba(0, 0, 0, 0.1);

width: 100%;

max-width: 400px;

}

.login-container h2 { margin-bottom: 25px; text-align: center; color: #0077cc;

}

form label { display: block;

margin-bottom: 6px; font-weight: bold;

}

form input { width: 100%; padding: 10px;

margin-bottom: 20px; border: 1px solid #ccc; border-radius: 5px;

}

Script.js

document.getElementById('loginForm').addEventListener('submit', function(*event*) {

*event*.preventDefault(); *// prevent actual form submission* const email = document.getElementById('email').value; const rollno = document.getElementById('rollno').value;

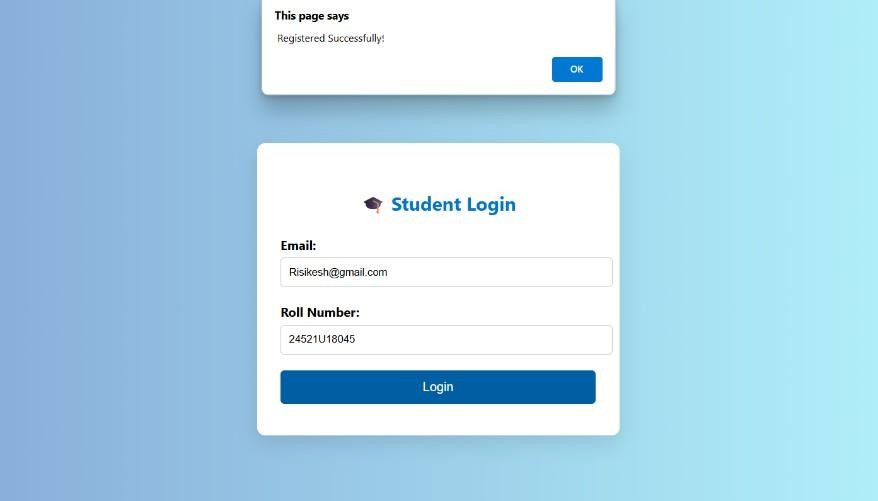
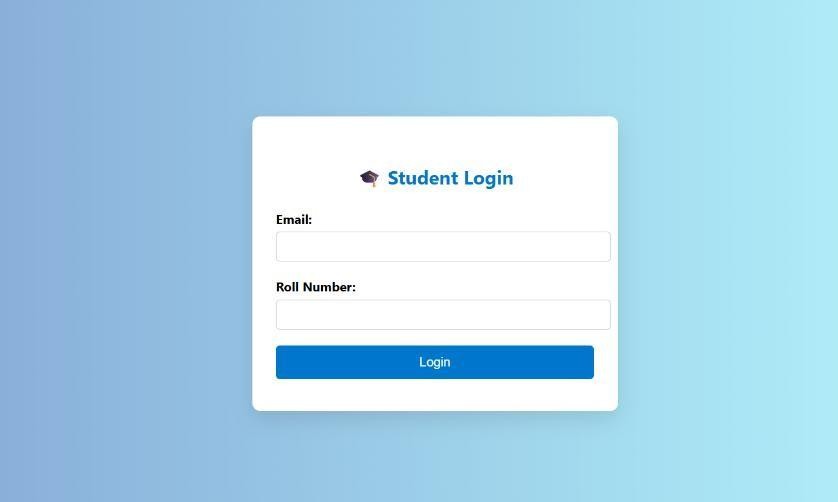
if (email && rollno) {

alert(" Registered Successfully!"); this.reset();

}

});

**OUTPUT:**

****

|  |  |
| --- | --- |
| COE (30): |  |
| OBSERVATION(10) |  |
| RECORD (10): |  |
| VIVA (10): |  |
| TOTAL (60): |  |

## RESULT :

A static Html, Css and JavaScript page is successfully served to the browser using a basic NodeJS HTTP server without using Express framework was executed successfully.

**Ex No: 2 Date:**

# Form Data Handling and Display Using Express.js and Handlebars in Node.js

## AIM:

To create a NodeJS server using Express that stores form data in a JSON file and displays it using Handlebars templating on a separate page.

### ALGORITHM:

**Step 1:** Initialize an Express project and install necessary packages (express, body-parser, express-handlebars, fs).

**Step 2:** Set up middleware to parse form data from req.body.

**Step 3:** Configure Handlebars view engine.

**Step 4:** Create an HTML form to take input from the user.

**Step 5:** Handle POST requests and retrieve form data.

**Step 6:** Store the form data in a .json file using fs.writeFile().

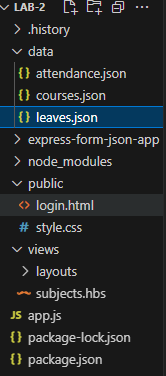
**Step 7:** Redirect the user to a new route after saving.

**Step 8:** Read JSON file data and pass it to Handlebars for rendering.

**Step 9:** Render the view and show the data in formatted layout.

**Step 10:** Start the server and validate the input/output process.

## DESIGN:

****

1. The program uses Express with modular middleware for handling routes and form data.
2. Handlebars is used to separate UI design from server logic.
3. Form input is stored in a .json file for simple data persistence.
4. GET and POST routes manage form submission and result display.

The result page shows total and CGPA dynamically using template rendering.

### SOURCE CODE:

#### Form.handlebar

<!DOCTYPE html>

<html lang="en">

<head><meta charset="UTF-8">

<title>Student Login</title>

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

<body>

<div class="login-container">

<h2>Student Login</h2>

<div class="success-message" id="successMsg">Login successful! Redirecting...</div>

<form id="loginForm" action="/login" method="POST">

<label>User Name:</label>

<input type="text" name="username" required>

<label>Email:</label>

<input type="email" name="email" required>

<label>Roll No:</label>

<input type="text" name="rollno" required>

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

</form> </div>

</body>

</html

#### subject.handlebar

<h2 class="headline">Current Semester Subjects</h2>

<p class="user-info">Logged in as: {{student.username}} ({{student.email}}, Roll No:

{{student.rollno}})</p>

<table class="main-table">

<tr>

<th>Subject</th>

<th>Attendance</th>

<th>Action</th>

</tr>

{{#each subjects}}

<tr>

<td>{{this}}</td>

<td style="text-align: center;">{{attendanceCount ../attendance this}} days</td>

<td style="text-align: center;">

<form action="/mark-attendance" method="POST" style="display:inline;">

<input type="hidden" name="subject" value="{{this}}">

<button type="submit">Mark Attendance</button>

</form>

</td>

</tr>

{{/each}}

</table>

<h3 class="section-headline">Submit Leave Request</h3>

<form action="/submit-leave" method="POST" class="leave-form">

<label for="subject">Subject:</label>

<select name="subject" id="subject" required>

<option value="">Select Subject</option>

{{#each subjects}}

<option value="{{this}}">{{this}}</option>

{{/each}}

</select>

<label for="date">Date:</label>

<input type="date" name="date" id="date" required>

<label for="reason">Reason:</label>

<input type="text" name="reason" id="reason" placeholder="Reason for leave" required>

<button type="submit">Submit Leave</button>

</form>

<h3 class="section-headline">Your Leave Requests</h3>

<table class="main-table">

<tr>

<th>Subject</th>

<th>Date</th>

<th>Reason</th>

</tr>

{{#if leaves.length}}

{{#each leaves}}

<tr>

<td>{{this.subject}}</td>

<td>{{this.date}}</td>

<td>{{this.reason}}</td>

</tr>

{{/each}}

{{else}}

<tr><td colspan="3" style="text-align:center;">No leave requests submitted.</td></tr>

{{/if}}

</table>

<p style="text-align:center;"><a href="/logout">Logout</a></p>

#### Server.js

const express = require('express');

const exphbs = require('express-handlebars'); const bodyParser = require('body-parser'); const fs = require('fs');

const path = require('path');

const app = express(); const PORT = 3000;

const hbs = exphbs.create({ extname: 'hbs',

helpers: {

attendanceCount: function(attendance, subject) {

return (attendance && attendance[subject]) ? attendance[subject].length : 0;

}

}

});

app.engine('hbs', hbs.engine); app.set('view engine', 'hbs');

app.use(bodyParser.urlencoded({ extended: true })); app.use(express.static('public'));

let currentStudent = null; const semesterSubjects = [ "Mathematics",

"Physics", "Chemistry", "Computer Science", "English"

];

const dataDir = path.join( dirname, 'data');

if (!fs.existsSync(dataDir)) fs.mkdirSync(dataDir);

const attendanceFile = path.join(dataDir, 'attendance.json');

if (!fs.existsSync(attendanceFile)) fs.writeFileSync(attendanceFile, '{}');

function getAttendance() {

return JSON.parse(fs.readFileSync(attendanceFile, 'utf-8'));

}

function saveAttendance(att) { fs.writeFileSync(attendanceFile, JSON.stringify(att, null, 2));

}

const leaveFile = path.join(dataDir, 'leaves.json');

if (!fs.existsSync(leaveFile)) fs.writeFileSync(leaveFile, '{}');

function getLeaves() {

return JSON.parse(fs.readFileSync(leaveFile, 'utf-8'));

}

function saveLeaves(leaves) {

fs.writeFileSync(leaveFile, JSON.stringify(leaves, null, 2));

}

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

res.sendFile(path.join( dirname, 'public', 'login.html'));

});

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

const { username, email, rollno } = req.body; currentStudent = { username, email, rollno }; res.redirect('/subjects');

});

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

if (!currentStudent) return res.redirect('/'); const attendance = getAttendance();

const studentAttendance = attendance[currentStudent.email] || {}; const leaves = getLeaves();

const studentLeaves = leaves[currentStudent.email] || []; res.render('subjects', {

student: currentStudent, subjects: semesterSubjects, attendance: studentAttendance, leaves: studentLeaves

});

});

app.post('/mark-attendance', (req, res) => { if (!currentStudent) return res.redirect('/'); const { subject } = req.body;

if (!subject) return res.redirect('/subjects'); const attendance = getAttendance();

if (!attendance[currentStudent.email]) attendance[currentStudent.email] = {};

if (!attendance[currentStudent.email][subject]) attendance[currentStudent.email][subject] = [];

const today = new Date().toISOString().slice(0, 10);

if (!attendance[currentStudent.email][subject].includes(today)) { attendance[currentStudent.email][subject].push(today);

}

saveAttendance(attendance); res.redirect('/subjects');

});

app.post('/submit-leave', (req, res) => {

if (!currentStudent) return res.redirect('/'); const { subject, date, reason } = req.body;

if (!subject || !date || !reason) return res.redirect('/subjects'); const leaves = getLeaves();

if (!leaves[currentStudent.email]) leaves[currentStudent.email] = []; leaves[currentStudent.email].push({ subject, date, reason }); saveLeaves(leaves);

res.redirect('/subjects');

});

app.get('/logout', (req, res) => { currentStudent = null; res.redirect('/');

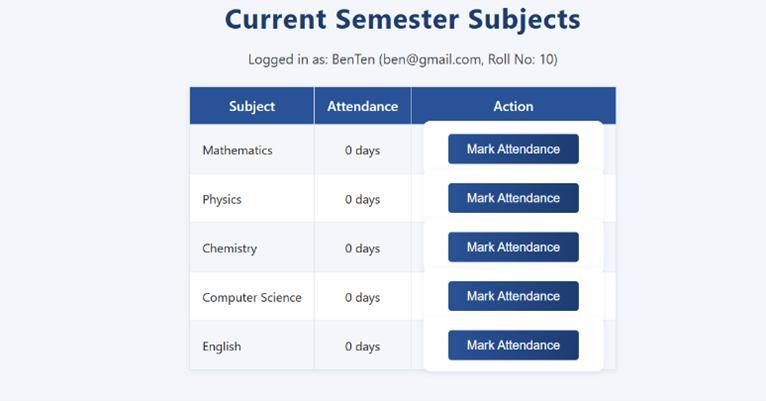
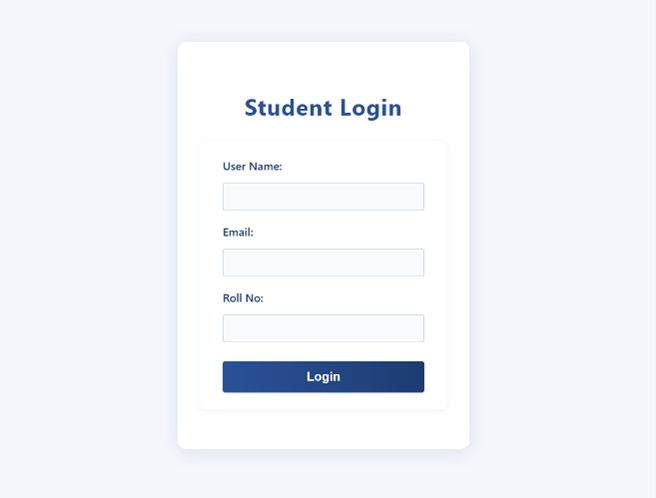
});

app.listen(PORT, () => {

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

});

**OUTPUT:**

****

|  |  |
| --- | --- |
| COE (30): |  |
| OBSERVATION(10) |  |
| RECORD (10): |  |
| VIVA (10): |  |
| TOTAL (60): |  |

## RESULT :

Thus above program form data was successfully saved in a JSON file and displayed using Handlebars on a redirect page after submission was executed successfully.