**AWT Lab Projects**

Simple projects:

1. Create a Node Js server that listens to port 6001.

const http=require('http')

const port=6001

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

    res.write('Hello Node From Port Number' + port)

    res.end()

})

server.listen(port,function(error){

    if(error){

        console.log("Something Went Wrong",error)

    }else{

        console.log('Server is listening on port' + port)

    }

})

1. Implement URL parameter routing to display specific content based on the URL.(Eg: ‘/’, ‘/home’, ‘/profile’, etc.,)

var express=require("express");

var app=express();

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

    res.send("List of Users");

});

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

    const userObj={

        id:10,

        name:"ABC",

        lastName:"DEF",

        status:true

    }

    res.send(userObj);

});

// Route for the home page

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

    res.send("Welcome to the home page");

});

// Route for the profile page

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

    res.send("This is the profile page");

});

// Route for a dynamic user page using a route parameter

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

    const userId = req.params.id;

    res.send(`Displaying content for user with ID: ${userId}`);

});

app.listen(4000, () => {

    console.log('Server is running on port 4000');

});

1. Create a route that returns a JSON response containing your name and email.

const express = require('express');

const app = express();

const PORT = 3000;

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

    const myInfo = {

        name: 'myName',

        email: 'myName.email@example.com'

    };

    res.json(myInfo);

});

app.listen(PORT, () => {

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

});

1. Implement a route that accepts POST requests and logs the request data to the console.

Indes.js:  
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>POST Request Logger</title>

</head>

<body>

    <h1>POST Request Logger</h1>

    <form id="postForm">

        <label for="requestData">Enter Data:</label>

        <input type="text" id="requestData" name="requestData" required>

        <button type="submit">Send POST Request</button>

    </form>

    <script>

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

            event.preventDefault();

            // Get the data from the input field

            const requestData = document.getElementById('requestData').value;

            // Make a POST request to the server

            fetch('http://localhost:3006/', {

                method: 'POST',

                headers: {

                'Content-Type': 'application/json',

                },

                body: JSON.stringify({ data: requestData }),

            })

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

            .then(data => {

                console.log('Server response:', data);

            })

            .catch(error => {

                console.error('Error sending POST request:', error);

            });

        });

    </script>

</body>

</html>

Index.js:  
  
const express = require('express');

const bodyParser = require('body-parser');

const app = express();

const port = 3006;

// Middleware to parse JSON data in the request body

app.use(bodyParser.json());

// Route for handling POST requests

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

    // Log the request data to the console

    console.log('Received POST request:', req.body);

    // Respond with a simple message

    res.json({ message: 'Request logged successfully!' });

});

// Serve static files (for the client-side code)

app.use(express.static('public'));

// Start the server

app.listen(port, () => {

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

});

1. Set up a static file server to serve HTML, CSS, and JavaScript files.

Server.js:  
const http = require('http');

const fs = require('fs');

const path = require('path');

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

    // Extract the file path from the request URL

    let filePath = '.' + req.url;

    // If the request is for the root, serve an index.html file

    if (filePath === './') {

        filePath = '/index.html';

    }

    // Specify the absolute path to the directory containing your static files

    const staticFilesDirectory = 'C:/Users/harip/OneDrive/Desktop/AWT/small projects/5/Staticfiles';

    // Construct the absolute path to the file

    const absoluteFilePath = path.join(staticFilesDirectory, filePath);

    // Determine the content type based on the file extension

    const extname = String(path.extname(filePath)).toLowerCase();

    const contentType = {

        '.html': 'text/html',

        '.css': 'text/css',

        '.js': 'text/javascript',

    }[extname] || 'application/octet-stream';

    // Read the file and serve it

    fs.readFile(absoluteFilePath, (err, content) => {

        if (err) {

            if (err.code === 'ENOENT') {

                // File not found

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

                res.end('404 Not Found');

            } else {

                // Server error

                res.writeHead(500, { 'Content-Type': 'text/html' });

                res.end('500 Internal Server Error');

            }

        } else {

            // Successful response

            res.writeHead(200, { 'Content-Type': contentType });

            res.end(content, 'utf-8');

        }

    });

});

// Set the port number for the server to listen on

const port = 3000;

// Start the server

server.listen(port, () => {

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

});

Staticsfiles:  
index.js:  
<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Static File Server Example</title>

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

</head>

<body>

<h1>Hello, Static World!</h1>

<p>This is a simple example of serving static files with Node.js and Express.</p>

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

</body>

</html>

Script.js:  
document.addEventListener('DOMContentLoaded', function() {

    console.log('Script loaded! Ready for action.');

    });

Style.css:  
body {

    font-family: Arial, sans-serif;

    background-color: #f0f0f0;

    text-align: center;

    margin: 50px;

    }

    h1 {

    color: #333;

    }

    p {

    color: #666;

    }

1. Use the Express.js framework to create a basic web application with routing.

const express=require('express');

const app=express();

const port=6001;

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

    res.send("Welcome to home page");

})

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

    res.send("Welcome to about page");

})

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

    res.send("Welcome to profile page");

})

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

    res.send("Welcome to contact page");

});

app.listen(port,()=>{

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

})

1. Create a custom 404 error page for handling undefined routes.

const express=require('express');

const app=express();

const port=6001;

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

    res.send("Welcome to home page");

})

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

    res.send("Welcome to about page");

})

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

    res.send("Welcome to profile page");

})

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

    res.send("Welcome to contact page");

})

app.use((re,res,next)=>{

    res.status(404).send("Sorry,can't find that page");

})

app.listen(port,()=>{

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

})

1. Secure your routes with basic authentication using middleware.

const express = require('express');

const basicAuth = require('express-basic-auth');

const app = express();

// Define users and their passwords for basic authentication

const users = {

    'admin': 'password123',

    'john': 'secret456'

};

// Middleware for basic authentication

const authMiddleware = basicAuth({

    users: users,

    challenge: true, // Displays the login dialog when authentication fails

    unauthorizedResponse: 'Unauthorized'

});

// Home page

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

res.send(`

    <h1>Server is running</h1>

    <button onclick="navigateTo('/secured-route')">Access Secured Route</button>

    <button onclick="navigateTo('/unsecured-route')">Access Unsecured Route</button>

    <script>

        function navigateTo(route) {

        window.location.href = route;

        }

    </script>

`);

});

// Secured route

app.get('/secured-route', authMiddleware, (req, res) => {

    res.send('This is a secured route!');

});

// Unsecured route

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

    res.send('This is an unsecured route!');

});

const port = 5000;

app.listen(port, () => {

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

});

1. Create a RESTful API that returns data in JSON format.

const express = require("express");

const app = express();

const port = 3000;

// Sample data

const books = [

  { id: 1, title: "The Great Gatsby", author: "F. Scott Fitzgerald" },

  { id: 2, title: "To Kill a Mockingbird", author: "Harper Lee" },

  { id: 3, title: "1984", author: "George Orwell" },

];

// Middleware to parse JSON in request body

app.use(express.json());

// Define routes

app.get("/api/books", (req, res) => {

  res.json(books);

});

app.get("/api/books/:id", (req, res) => {

  const book = books.find((b) => b.id === parseInt(req.params.id));

  if (!book) return res.status(404).json({ message: "Book not found" });

  res.json(book);

});

// Run the server

app.listen(port, () => {

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

});

1. Create a custom error handler to format and send error responses.

const express = require("express");

const app = express();

const port = 3000;

// Sample data

const books = [

  { id: 1, title: "The Great Gatsby", author: "F. Scott Fitzgerald" },

  { id: 2, title: "To Kill a Mockingbird", author: "Harper Lee" },

  { id: 3, title: "1984", author: "George Orwell" },

];

// Middleware to parse JSON in request body

app.use(express.json());

// Define routes

app.get("/api/books", (req, res) => {

  res.json(books);

});

app.get("/api/books/:id", (req, res) => {

  const book = books.find((b) => b.id === parseInt(req.params.id));

  if (!book) return res.status(404).json({ message: "Book not found" });

  res.json(book);

});

// Run the server

app.listen(port, () => {

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

});

1. Implement Web Sockets for real-time communication in your application (Implement Socket.io library).

const express = require('express');

const http = require('http');

const { Server: socketIO } = require('socket.io');

const app = express();

const server = http.createServer(app);

const io = new socketIO(server);

app.use(express.static("public"));

io.on("connection", (socket) => {

  console.log("a user connected ");

  socket.on("chat message", (msg) => {

    console.log("message: " + msg);

    io.emit("chat message", msg);

  });

  socket.on("disconnect", () => {

    console.log("user disconnected");

  });

});

const PORT = 5000;

server.listen(PORT, () => {

  console.log(`server is running on ${PORT}`);

});

<html>

  <head>

    <title>Socket IO real time communcation</title>

  </head>

  <body>

    <ul id="messages">

      Messages

    </ul>

    <form id="form" action="">

      <input id="m" />

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

    </form>

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

    <script src="https://code.jquery.com/jquery-3.6.4.min.js"></script>

    <script>

      $(function () {

        const socket = io();

        socket.on("chat message", (msg) => {

          $("#messages").append($("<li>").text(msg));

        });

        $("form").submit(() => {

          console.log($("#m").val());

          socket.emit("chat message", $("#m").val());

          $("#m").val("");

          return false;

        });

      });

    </script>

  </body>

</html>

1. Enhance security by hashing and salting user passwords before storing them in the database. (use bcrypt).

const bcrypt = require('bcrypt');

// Function to hash and salt a password

async function hashPassword(password) {

    try {

        // Generate a salt

        const saltRounds = 10; // Recommended number of rounds

        const salt = await bcrypt.genSalt(saltRounds);

        // Hash the password with the salt

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

        return hashedPassword;

    } catch (error) {

        console.error("Error hashing password:", error);

        throw error;

    }

}

// Example password

const userPassword = "MySecurePassword123";

// Hash and salt the password

hashPassword(userPassword)

    .then(hashed => {

        console.log("Original Password:", userPassword);

        console.log("Hashed and Salted Password:", hashed);

    })

    .catch(error => {

        // Handle errors

    });

1. JWT Token Validation: Implement token validation middleware to ensure the integrity of JSON Web Tokens (JWT).

Server.js:  
const jwt = require('jsonwebtoken');

const express = require('express');

const bodyParser = require('body-parser');

const app = express();

const secretKey = 'yourSecretKey';

// Parse incoming requests with JSON payloads

app.use(bodyParser.json());

// Middleware to validate JWT tokens

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

    const token = req.header('Authorization');

    // Remove 'Bearer ' from the token if present

    const tokenWithoutBearer = token.replace(/^Bearer /, '');

    if (!tokenWithoutBearer) {

        return res.status(401).json({ error: 'Access denied. Token not provided.' });

    }

    try {

        const decoded = jwt.verify(tokenWithoutBearer, secretKey);

        req.user = decoded;

        next();

    } catch (err) {

        console.log(err)

        return res.status(401).json({ error: 'Invalid token.' });

    }

};

// Protected route example

app.get('/protected-route', validateToken, (req, res) => {

    res.json({ message: 'Access granted. This is a protected route.', user: req.user });

});

// Login route to generate a JWT (replace this with your actual authentication logic)

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

    const { username, password } = req.body;

    console.log(req.body)

    // Replace this with your actual user authentication logic

    if (username === 'user' && password === 'password') {

        const payload = { username };

        const token = jwt.sign(payload, secretKey, { expiresIn: '1h' });

        res.json({ token });

    } else {

        res.status(401).json({ error: 'Invalid credentials.' });

    }

});

// Start the server

const port = 3000;

app.listen(port, () => {

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

});

Client.js:  
  
const axios = require("axios");

// Replace with your server's URL

const serverUrl = "http://localhost:3000";

// Replace with your actual credentials for authentication

const credentials = {

  username: "user",

  password: "password",

};

// Log in to get the JWT token

axios

  .post(`${serverUrl}/login`, credentials)

  .then((response) => {

    const { token } = response.data;

    console.log(token);

    // Use the token to make a request to the protected route

    axios

      .get(`${serverUrl}/protected-route`, {

        headers: {

          Authorization: `Bearer ${token}`,

        },

      })

      .then((response) => {

        console.log(response.data);

      })

      .catch((error) => {

        console.error(error.response.data);

      });

  })

  .catch((error) => {

    console.error(error.response.data);

  });

1. Develop the backend server application for an authentication system (login, register).

Same a bigger7

1. Read and parse a JSON file, then display its contents in a structured format.

const fs = require('fs/promises');

const Table = require('cli-table');

// Define an asynchronous function

const readAndDisplayJson = async () => {

  // Step 1: Read the JSON file

  const filePath = "sample.json";

  try {

    const jsonData = await fs.readFile(filePath, "utf-8");

    // Step 2: Parse the JSON data

    const data = JSON.parse(jsonData);

    // Step 3: Display the contents in a table format

    const table = new Table();

    for (const key in data) {

      if (Object.prototype.hasOwnProperty.call(data, key)) {

        table.push([key, JSON.stringify(data[key])]);

      }

    }

    console.log("JSON file contents as a table:");

    console.log(table.toString());

  } catch (error) {

    if (error.code === "ENOENT") {

      console.error(`Error: File not found at path '${filePath}'`);

    } else {

      console.error(`An unexpected error occurred: ${error.message}`);

    }

  }

};

// Call the asynchronous function

readAndDisplayJson();

{

"name" : "doremon",

"place" : "nellore"

}

1. File system in Node js: Create a new text file and perform read, write, and append operations.

//write

// var fs=require('fs');

// fs.writeFile("write.txt","console.log('done')",function(err){

//     console.log("helloo world");

// })

//read

// var fs=require('fs');

// fs.readFile("write.txt","utf-8",function(err,data){

// console.log(data);

// })

//append

var fs=require('fs');

fs.appendFile("write.txt","console.log('hello')",function(err){

console.log("added successfully")

})

1. File system in Node js: Perform rename a file and copy a file to another file, delete file operations.

var fs=require('fs');

// fs.writeFile("file.txt","console.log('hi')",function(err){

//     console.log("file created");

// })

//rename

// fs.rename("file.txt","renamedfile.txt",function(err){

//     console.log("file renamed");

// })

//copy a file to another file:

// var fs=require('fs');

// fs.writeFile("first.txt","hello",function(err){

//     console.log("first file created");

// })

// fs.writeFile("second.txt","hello second",function(err){

//     console.log("second file created");

// })

// fs.copyFile("first.txt","second.txt",function(err){

//     console.log("copied")

// })

//delete

// var fs=require('fs');

// fs.writeFile("file.txt","console.log('hi')",function(err){

//     console.log("second file created");

// })

fs.unlink("file.txt",function(err){

    console.log("file deleted")

})

Bigger projects:

1. Develop the important backend (Node Js, Express Js, MongoDB) functionalities for a task manager application.

const express = require('express');

const mongoose = require('mongoose');

const app = express();

const PORT = 6001;

app.use(express.json());

const Task = require('./taskModel')

mongoose.connect('mongodb+srv://haripriya2813:hari@cluster0.qr9lxce.mongodb.net/?retryWrites=true&w=majority', {

  useNewUrlParser: true,

  useUnifiedTopology: true,

}).then(()=>{

    // Get all tasks

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

            try {

            const tasks = await Task.find();

            res.json(tasks);

            } catch (err) {

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

            }

        });

    // Create a new task

        app.post('/new-task', async (req, res) => {

            const task = new Task({

                title: req.body.title,

                description: req.body.description,

            });

            try {

                const newTask = await task.save();

                res.status(201).json(newTask);

            } catch (err) {

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

            }

        });

    // Update a task

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

            try{

                const task = await Task.findById(req.params.id);

                task.title = req.body.title;

                task.description = req.body.description;

                const updatedTask = await task.save();

                res.status(201).json(updatedTask);

            }catch(err){

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

            }

        });

    // Delete a task

        app.delete('/delete-task/:id', async (req, res) => {

            try{

                await Task.deleteOne({\_id: req.params.id});

                res.status(201).json({message: "task deleted"});

            }catch(err){

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

            }

        });

}

)

app.listen(PORT, () => {

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

});

const mongoose = require('mongoose');

const taskSchema = new mongoose.Schema({

    title: {

        type: String,

        required: true,

    },

    description: String,

        createdAt: {

        type: Date,

        default: Date.now,

      },

});

const Task = mongoose.model('Task', taskSchema);

module.exports = Task;

1. Develop the backend (Node Js, Express Js, MongoDB) for a CRUD application.

Same as above

1. Develop and application to send emails (use libraries link Nodemailer, etc.,).

const express = require('express');

const app = express();

const PORT = 6001;

app.use(express.json());

const nodemailer = require('nodemailer');

const transporter = nodemailer.createTransport({

    host: 'smtp.ethereal.email',

    port: 587,

    auth: {

        user: 'vernie52@ethereal.email',

        pass: 'SYhCW1Bt54JhSVTKPq'

    }

});

    // Send email

app.post('/send-mail', async (req, res) => {

    const {mailTo, subject, text} = req.body;

    try {

        let mailDetails = {

        from: 'vernie52@ethereal.email',

        to: mailTo,

        subject: subject,

        text: text

                };

                transporter.sendMail(mailDetails, function(err, data) {

                    if(err) {

                        console.log('Error Occurs');

                    } else {

                        console.log('Email sent successfully');

                        res.status(201).json({ message: "mail sent successfully" });

                    }

                });

            } catch (err) {

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

            }

        });

app.listen(PORT, () => {

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

});

1. Develop the backend for a job portal using Node JS, Express JS, MongoDB. Perform important functionalities such as add new job, update job, apply for job, approve job application, etc.,

const express = require('express');

const mongoose = require('mongoose');

const app = express();

const PORT = 6001;

app.use(express.json());

const { Job, Application } = require('./jobsModel');

mongoose.connect('mongodb+srv://haripriya2813:hari@cluster0.qr9lxce.mongodb.net/?retryWrites=true&w=majority', {

  useNewUrlParser: true,

  useUnifiedTopology: true,

}).then(()=>{

    // Get all jobs

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

            try {

            const jobs = await Job.find();

            res.json(jobs);

            } catch (err) {

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

            }

        });

    // Create a new job

        app.post('/new-job', async (req, res) => {

            const job = new Job({

                title: req.body.title,

                description: req.body.description,

                companyName:req.body.companyName,

            });

            try {

                const newjob = await job.save();

                res.status(201).json(newjob);

            } catch (err) {

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

            }

        });

    // Update a job

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

            try{

                const job = await Job.findById(req.params.id);

                job.title = req.body.title;

                job.description = req.body.description;

                const updatedjob = await job.save();

                res.status(201).json(updatedjob);

            }catch(err){

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

            }

        });

    // Delete a job

        app.delete('/delete-task/:id', async (req, res) => {

            try{

                await Job.deleteOne({\_id: req.params.id});

                res.status(201).json({message: "job deleted"});

            }catch(err){

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

            }

        });

    //Apply for a job

        app.post('/apply-job', async(req, res)=>{

            const {jobId, applicantName, applicantId} = req.body;

            try{

                const application = new Application({jobId, applicantName, applicantId});

                const newApplication = await application.save();

                res.status(201).json(newApplication);

            }catch(err){

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

            }

        })

    //Approve job application

        app.post('/approve-application/:id', async(req, res)=>{

            try{

                const application = await Application.findById(req.params.id);

                application.status = "Accepted";

                await application.save();

                res.status(201).json(application);

            }catch(err){

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

            }

        })

}

)

app.listen(PORT, () => {

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

});

const mongoose = require('mongoose');

const jobSchema = new mongoose.Schema({

  title: {

    type: String,

    required: true,

  },

  description:{

    type: String,

  },

  companyName: {

    type: String

  }

});

const applicationSchema = new mongoose.Schema({

    applicantId:{

        type: String

    },

    applicantName: {

        type: String

    },

    JobId:{

        type: String,

    },

    status: {

        type: String,

        default: "Pending"

    }

})

const Job = mongoose.model('Job', jobSchema);

const Application = mongoose.model('Application', applicationSchema);

module.exports = { Job, Application };

1. Develop a simple calculator using plain JavaScript or any frontend library (no server required).

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

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

    <title>Simple Calculator</title>

    <style>

      body {

        display: flex;

        justify-content: center;

        align-items: center;

        height: 100vh;

        margin: 0;

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

        background-color: #f5f5f5;

      }

      #calculator {

        background-color: #fff;

        border-radius: 8px;

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

        padding: 20px;

        text-align: center;

      }

      input {

        width: 100%;

        padding: 10px;

        font-size: 20px;

        margin-bottom: 10px;

        box-sizing: border-box;

      }

      button {

        width: 50px;

        height: 50px;

        font-size: 18px;

        margin: 5px;

        cursor: pointer;

        background-color: #4caf50;

        color: #fff;

        border: none;

        border-radius: 4px;

        transition: background-color 0.3s;

      }

      button.operator {

        background-color: #ff9800;

      }

      button.clear {

        background-color: #e74c3c;

      }

      button.equal {

        background-color: #3498db;

      }

      button:hover {

        background-color: #45a049;

      }

    </style>

  </head>

  <body>

    <div id="calculator">

      <input id="display" readonly />

      <br />

      <button onclick="appendToDisplay('1')">1</button>

      <button onclick="appendToDisplay('2')">2</button>

      <button onclick="appendToDisplay('3')">3</button>

      <button onclick="appendToDisplay('+')" class="operator">+</button>

      <br />

      <button onclick="appendToDisplay('4')">4</button>

      <button onclick="appendToDisplay('5')">5</button>

      <button onclick="appendToDisplay('6')">6</button>

      <button onclick="appendToDisplay('-')" class="operator">-</button>

      <br />

      <button onclick="appendToDisplay('7')">7</button>

      <button onclick="appendToDisplay('8')">8</button>

      <button onclick="appendToDisplay('9')">9</button>

      <button onclick="appendToDisplay('\*')" class="operator">\*</button>

      <br />

      <button onclick="appendToDisplay('0')">0</button>

      <button onclick="clearDisplay()" class="clear">C</button>

      <button onclick="calculateResult()" class="equal">=</button>

      <button onclick="appendToDisplay('/')" class="operator">/</button>

    </div>

    <script>

      function appendToDisplay(value) {

        document.getElementById("display").value += value;

      }

      function clearDisplay() {

        document.getElementById("display").value = "";

      }

      function calculateResult() {

        const displayValue = document.getElementById("display").value;

        try {

          const result = eval(displayValue);

          document.getElementById("display").value = result;

        } catch (error) {

          document.getElementById("display").value = "Error";

        }

      }

    </script>

  </body>

</html>

1. Develop a simple BMI calculator using plain JavaScript or any frontend library.

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>BMI Calculator</title>

    <style>

        body {

            font-family: Arial, sans-serif;

            display: flex;

            align-items: center;

            justify-content: center;

            height: 100vh;

            margin: 0;

        }

        #calculator {

            text-align: center;

            border: 1px solid #ccc;

            padding: 20px;

            border-radius: 5px;

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

        }

    </style>

</head>

<body>

    <div id="calculator">

        <h2>BMI Calculator</h2>

        <label for="height">Height (cm):</label>

        <input type="number" id="height" placeholder="Enter height" required>

        <label for="weight">Weight (kg):</label>

        <input type="number" id="weight" placeholder="Enter weight" required>

        <button onclick="calculateBMI()">Calculate BMI</button>

        <div id="result"></div>

        <script>

            function calculateBMI() {

                const heightInput = document.getElementById('height');

                const weightInput = document.getElementById('weight');

                const resultDiv = document.getElementById('result');

                const height = parseFloat(heightInput.value);

                const weight = parseFloat(weightInput.value);

                if (isNaN(height) || isNaN(weight) || height <= 0 || weight <= 0) {

                    resultDiv.innerHTML = 'Please enter valid height and weight.';

                    return;

                }

                const bmi = (weight / Math.pow(height / 100, 2)).toFixed(2);

                let category;

                if (bmi < 18.5) {

                    category = 'Underweight';

                } else if (bmi < 24.9) {

                    category = 'Normal Weight';

                } else if (bmi < 29.9) {

                    category = 'Overweight';

                } else {

                    category = 'Obese';

                }

                resultDiv.innerHTML = `BMI: ${bmi} (${category})`;

            }

        </script>

    </div>

</body>

</html>

1. Develop a simple authentication (login, register) app (both frontend and backend).

Index.js:  
const express=require('express');

const path=require("path");

const bcrypt=require("bcrypt");

const collection=require("./config");

const app=express();

app.use(express.json());

app.use(express.urlencoded({extended:false}))

//use EJS as the view engine

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

//static file

app.use(express.static("public"));

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

    res.render("login");

})

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

    res.render("signup");

})

app.post("/signup",async(req,res)=>{

    const data={

        name: req.body.username,

        password:req.body.password

    }

    const existingUser=await collection.findOne({name:data.name});

    if(existingUser){

        res.send("User already exists. Please choose a different username.");

    }

    else{

        const saltRounds=10;

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

        data.password=hashedPassword;

        const userdata=await collection.insertMany(data);

        console.log(userdata);

     }

})

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

    try{

        const check=await collection.findOne({name:req.body.username});

        if(!check){

            res.send("user name cannot found");

        }

        const isPasswordMatch=await bcrypt.compare(req.body.password,check.password);

        if(isPasswordMatch){

            res.render("home");

        }

        else{

            res.send("wrong password");

        }

    }catch{

        res.send("wrong details")

    }

})

const port=5000;

app.listen(port,()=>{

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

})

Config.js:  
const mongoose=require("mongoose");

const connect=mongoose.connect("mongodb+srv://haripriya2813:hari@cluster0.qr9lxce.mongodb.net/?retryWrites=true&w=majority");

connect.then(()=>{

    console.log("Database connected Successfully");

})

.catch(()=>{

    console.log("Database cannot be connected");

})

const LoginSchema=new mongoose.Schema({

    name:{

        type:String,

        required:true

    },

    password:{

        type:String,

        required:true

    }

})

const collection=new mongoose.model("users",LoginSchema)

module.exports=collection;

home.ejs:  
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Home page</title>

</head>

<body>

    <h1 style="text-align: center;">Welcome to Home page</h1>

</body>

</html>

Login.ejs:  
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Login</title>

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

</head>

<body>

    <div class="form-container">

        <h2>Login</h2>

        <form action="/login" method="post">

            <div class="form-group">

                <label for="name">Name:</label>

                <input type="text" id="name" name="username" placeholder="Enter Your name" required autocomplete="off">

            </div>

            <div class="form-group">

                <label for="password">Password:</label>

                <input type="password" id="password" name="password" placeholder="password" required>

            </div>

            <button type="submit" class="submit-btn">Login</button>

        </form>

        <p>Don't have an account? <a href="/signup">Signup</a></p>

    </div>

</body>

</html>

Signup.ejs:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Login</title>

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

</head>

<body>

    <div class="form-container">

        <h2>Signup</h2>

        <form action="/signup" method="post">

            <div class="form-group">

                <label for="name">Name:</label>

                <input type="text" id="name" name="username" placeholder="Enter Your name" required autocomplete="off">

            </div>

            <div class="form-group">

                <label for="password">Password:</label>

                <input type="password" id="password" name="password" placeholder="password" required>

            </div>

            <button type="submit" class="submit-btn">Signup</button>

        </form>

    </div>

</body>

</html>

1. Develop a currency conversion application with the conversion rates stored in the backend database.

Public/Index.html:

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

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

    <title>Currency Converter</title>

  </head>

  <body>

    <h1>Currency Converter</h1>

    <label for="fromCurrency">From Currency:</label>

    <select id="fromCurrency">

      <!-- Populate this dynamically based on backend data -->

    </select>

    <label for="toCurrency">To Currency:</label>

    <select id="toCurrency">

      <!-- Populate this dynamically based on backend data -->

    </select>

    <label for="amount">Amount:</label>

    <input type="number" id="amount" placeholder="Enter amount" />

    <button onclick="convertCurrency()">Convert</button>

    <p id="result"></p>

    <script>

      // Fetch conversion rates from the backend

      async function fetchConversions() {

        const response = await fetch("/api/conversions");

        const conversions = await response.json();

        const fromCurrencySelect = document.getElementById("fromCurrency");

        const toCurrencySelect = document.getElementById("toCurrency");

        // Function to add unique option to select

        const addUniqueOption = (selectElement, value, text) => {

          const optionExists = Array.from(selectElement.options).some(

            (option) => option.value === value

          );

          if (!optionExists) {

            const option = document.createElement("option");

            option.value = value;

            option.text = text;

            selectElement.appendChild(option);

          }

        };

        // Populate dropdowns with currency options

        conversions.forEach((conversion) => {

          addUniqueOption(fromCurrencySelect, conversion.from, conversion.from);

          addUniqueOption(toCurrencySelect, conversion.to, conversion.to);

        });

      }

      // Perform currency conversion

      async function convertCurrency() {

        const fromCurrency = document.getElementById("fromCurrency").value;

        const toCurrency = document.getElementById("toCurrency").value;

        const amount = document.getElementById("amount").value;

        const response = await fetch("/api/convert", {

          method: "POST",

          headers: {

            "Content-Type": "application/json",

          },

          body: JSON.stringify({

            from: fromCurrency,

            to: toCurrency,

            amount: parseFloat(amount),

          }),

        });

        const result = await response.json();

        document.getElementById(

          "result"

        ).innerText = `Result: ${result.result}`;

      }

      // Fetch conversions on page load

      window.onload = fetchConversions;

    </script>

  </body>

</html>

Server.js:

const express = require("express");

const mongoose = require("mongoose");

const app = express();

const PORT = 3005;

// Connect to the in-memory database (replace this with a real database in production)

mongoose.connect("mongodb://localhost:27017/emplyee", {

  useNewUrlParser: true,

  useUnifiedTopology: true,

});

const currencySchema = new mongoose.Schema({

  code: String,

  name: String,

  symbol: String,

});

// Create a schema and model for the conversion rates

const conversionSchema = new mongoose.Schema({

  from: String,

  to: String,

  rate: Number,

});

const Currency = mongoose.model("Currency", currencySchema);

const Conversion = mongoose.model("Conversion", conversionSchema);

// Clear data on server start

const clearPreviousAndInsertNewData = async () => {

  try {

    await Currency.deleteMany({});

    await Conversion.deleteMany({});

    console.log("Data cleared successfully.");

    insertSampleData();

  } catch (error) {

    console.error("Error clearing data:", error);

  }

};

// Inserting sample data

const currenciesData = [

  { code: "USD", name: "United States Dollar", symbol: "$" },

  { code: "EUR", name: "Euro", symbol: "€" },

  { code: "INR", name: "Indian Rupee", symbol: "₹" },

];

const conversionsData = [

  { from: "USD", to: "EUR", rate: 0.94 }, // 1 USD = 0.94 EUR

  { from: "USD", to: "INR", rate: 83.31 }, // 1 USD = 83.31 INR

  { from: "EUR", to: "USD", rate: 1.07 }, // 1 EUR = 1.07 USD

  { from: "EUR", to: "INR", rate: 89.08 }, // 1 EUR = 89.08 INR

  { from: "INR", to: "USD", rate: 0.012 }, // 1 INR = 0.012 USD

  { from: "INR", to: "EUR", rate: 0.011 }, // 1 INR = 0.011 EUR

];

const insertSampleData = async () => {

  try {

    await Currency.insertMany(currenciesData);

    await Conversion.insertMany(conversionsData);

    console.log("Sample data inserted successfully.");

  } catch (error) {

    console.error("Error inserting sample data:", error);

  }

};

clearPreviousAndInsertNewData();

// Express middleware to parse JSON

app.use(express.json());

// API endpoint to get conversion rates

app.get("/api/conversions", async (req, res) => {

  try {

    const conversions = await Conversion.find();

    res.json(conversions);

  } catch (error) {

    res.status(500).json({ error: "Internal Server Error" });

  }

});

// API endpoint to perform currency conversion

app.post("/api/convert", async (req, res) => {

  const { from, to, amount } = req.body;

  try {

    const conversion = await Conversion.findOne({ from, to });

    if (!conversion) {

      return res.status(404).json({ error: "Conversion rate not found" });

    }

    const result = amount \* conversion.rate;

    res.json({ result });

  } catch (error) {

    res.status(500).json({ error: "Internal Server Error" });

  }

});

app.use(express.static("public"));

app.listen(PORT, () => {

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

});