**Link https://learnwithnil.blogspot.com/2024/05/full-stack-developement-practical-slips.html?m=1**

**Full Stack Developement Practical Slips Programs**

**Slip 1**

Create an HTML form for Login and write a JavaScript to validate email ID and Password using Regular Expression.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Login Form</title>

<style>

    /\* Add some basic styling for the form \*/

    body {

        font-family: Arial, sans-serif;

    }

    form {

        max-width: 300px;

        margin: 0 auto;

    }

    input[type="text"],

    input[type="password"],

    input[type="submit"] {

        width: 100%;

        padding: 10px;

        margin: 5px 0;

        border: 1px solid #ccc;

        box-sizing: border-box;

    }

    input[type="submit"] {

        background-color: #4CAF50;

        color: white;

        cursor: pointer;

    }

    input[type="submit"]:hover {

        background-color: #45a049;

    }

    .error {

        color: red;

    }

</style>

</head>

<body>

<h2>Login Form</h2>

<form id="loginForm" onsubmit="return validateForm()">

    <label for="email">Email:</label>

    <input type="text" id="email" name="email" placeholder="Enter your email">

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

    <input type="password" id="password" name="password" placeholder="Enter your password">

    <input type="submit" value="Login">

</form>

<script>

function validateForm() {

    // Fetch values from form

    var email = document.getElementById('email').value;

    var password = document.getElementById('password').value;

    // Regular expression for email validation

    var emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

    // Regular expression for password validation

    // Password must contain at least one uppercase letter, one lowercase letter, one digit, and be at least 8 characters long

    var passwordRegex = /^(?=.\*\d)(?=.\*[a-z])(?=.\*[A-Z]).{8,}$/;

    // Perform validation

    if (!emailRegex.test(email)) {

        alert('Please enter a valid email address');

        return false;

    }

    if (!passwordRegex.test(password)) {

        alert('Password must contain at least one uppercase letter, one lowercase letter, one digit, and be at least 8 characters long');

        return false;

    }

    // If validation passes, submit the form

    return true;

}

</script>

</body>

</html>

Create an HTML form that contain the Student Registration details and write a JavaScript to validate Student first and last name as it should not contain other than alphabets and age should be between 18 to 50

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Student Registration Form</title>

<style>

    /\* Add some basic styling for the form \*/

    body {

        font-family: Arial, sans-serif;

    }

    form {

        max-width: 300px;

        margin: 0 auto;

    }

    input[type="text"],

    input[type="number"],

    input[type="submit"] {

        width: 100%;

        padding: 10px;

        margin: 5px 0;

        border: 1px solid #ccc;

        box-sizing: border-box;

    }

    input[type="submit"] {

        background-color: #4CAF50;

        color: white;

        cursor: pointer;

    }

    input[type="submit"]:hover {

        background-color: #45a049;

    }

    .error {

        color: red;

    }

</style>

</head>

<body>

<h2>Student Registration Form</h2>

<form id="registrationForm" onsubmit="return validateForm()">

    <label for="firstName">First Name:</label>

    <input type="text" id="firstName" name="firstName" placeholder="Enter your first name">

    <label for="lastName">Last Name:</label>

    <input type="text" id="lastName" name="lastName" placeholder="Enter your last name">

    <label for="age">Age:</label>

    <input type="number" id="age" name="age" placeholder="Enter your age">

    <input type="submit" value="Register">

</form>

<script>

function validateForm() {

    // Fetch values from form

    var firstName = document.getElementById('firstName').value;

    var lastName = document.getElementById('lastName').value;

    var age = parseInt(document.getElementById('age').value);

    // Regular expression for first and last name validation (alphabets only)

    var nameRegex = /^[A-Za-z]+$/;

    // Perform validation

    if (!nameRegex.test(firstName)) {

        alert('First name should contain only alphabets');

        return false;

    }

    if (!nameRegex.test(lastName)) {

        alert('Last name should contain only alphabets');

        return false;

    }

    if (isNaN(age) || age < 18 || age > 50) {

        alert('Age should be between 18 and 50');

        return false;

    }

    // If validation passes, submit the form

    return true;

}

</script>

</body>

</html>

**Slip 2**

Create a Node.js file that will convert the output "Full Stack!" into reverse string.

**// reverseString.js**

**// Function to reverse a string**

**function reverseString(str) {**

**return str.split('').reverse().join('');**

**}**

**// Input string**

**const inputString = "Full Stack!";**

**// Reverse the input string**

**const reversedString = reverseString(inputString);**

**// Output the reversed string**

**console.log("Reversed String:", reversedString);**

To run this code, save it into a file named **reverseString.js**, and then execute it using Node.js:

bash

Copy code

node reverseString.js

This will output the reversed string:

yaml

Copy code

Reversed String: !kcatS lluF

Using node js create a web page to read two file names from user and append contents of first file into second file

To create a Node.js web application that allows users to input two file names and appends the contents of the first file into the second file, you can use the Express.js framework. Below is an example code:

First, make sure you have Node.js installed on your system. Then, create a directory for your project and initialize it with npm (**npm init -y**). After that, install Express.js (**npm install express**) and **body-parser** (**npm install body-parser**) for parsing POST request body.

Here's the code for your Node.js web application:

// index.js const express = require('express'); const bodyParser = require('body-parser'); const fs = require('fs'); const app = express(); const port = 3000; // Parse URL-encoded bodies (as sent by HTML forms) app.use(bodyParser.urlencoded({ extended: true })); // Serve HTML file app.get('/', (req, res) => { res.sendFile(\_\_dirname + '/index.html'); }); // Handle POST request to append contents of first file into second file app.post('/appendFiles', (req, res) => { const { file1, file2 } = req.body; // Read contents of file1 fs.readFile(file1, 'utf8', (err, data) => { if (err) { return res.status(500).send('Error reading file 1'); } // Append contents of file1 to file2 fs.appendFile(file2, data, (err) => { if (err) { return res.status(500).send('Error appending file 1 contents to file 2'); } res.send('Contents of file 1 appended to file 2 successfully'); }); }); }); // Start the server app.listen(port, () => { console.log(`Server is listening at http://localhost:${port}`); });

<!-- index.html -->

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>File Append</title>

</head>

<body>

    <h2>Append Contents of File 1 into File 2</h2>

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

        <label for="file1">File 1 Name:</label>

        <input type="text" id="file1" name="file1" required><br><br>

        <label for="file2">File 2 Name:</label>

        <input type="text" id="file2" name="file2" required><br><br>

        <input type="submit" value="Append">

    </form>

</body>

</html>

**Slip 3**

Using node js create a User Login System.

Creating a user login system in Node.js involves several steps, including setting up a server, handling user registration and authentication, and managing user sessions. Below is a basic example of how you can create a user login system using Node.js, Express.js, and a simple in-memory storage for user data.

First, make sure you have Node.js installed on your system. Then, create a directory for your project and initialize it with npm (**npm init -y**). After that, install Express.js (**npm install express**), **body-parser** (**npm install body-parser**), and **express-session** (**npm install express-session**).

Here's the code for your Node.js user login system:

// index.js

const express = require('express');

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

const session = require('express-session');

const app = express();

const port = 3000;

// Middleware

app.use(bodyParser.urlencoded({ extended: true }));

app.use(session({

    secret: 'secret\_key', // Change this to a random string

    resave: false,

    saveUninitialized: true

}));

// In-memory user database (Replace this with a real database in production)

const users = [

    { id: 1, username: 'user1', password: 'password1' },

    { id: 2, username: 'user2', password: 'password2' }

];

// Routes

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

    res.sendFile(\_\_dirname + '/index.html');

});

// Login route

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

    const { username, password } = req.body;

    const user = users.find(u => u.username === username && u.password === password);

    if (user) {

        req.session.userId = user.id;

        res.send('Login successful!');

    } else {

        res.send('Invalid username or password.');

    }

});

// Logout route

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

    req.session.destroy(err => {

        if (err) {

            console.error('Error destroying session:', err);

        }

        res.redirect('/');

    });

});

// Protected route (requires login)

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

    if (!req.session.userId) {

        return res.redirect('/');

    }

    const user = users.find(u => u.id === req.session.userId);

    if (!user) {

        return res.send('User not found.');

    }

    res.send(`Welcome, ${user.username}! <br><a href="/logout">Logout</a>`);

});

// Start the server

app.listen(port, () => {

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

});

<!-- index.html -->

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>User Login</title>

</head>

<body>

    <h2>User Login</h2>

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

        <label for="username">Username:</label>

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

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

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

        <input type="submit" value="Login">

    </form>

</body>

</html>

Create a node.js file that Select all records from the "Teacher" table, and find the Teachers whose salary is greater than 20,000.

To accomplish this task, you'll need to use a Node.js module to interact with your database. Assuming you're using MySQL as your database, you can use the **mysql** module to connect to your database and execute SQL queries.

First, make sure you have MySQL installed and running on your system. Then, install the **mysql** module using npm (**npm install mysql**).

Here's a Node.js script that selects all records from the "Teacher" table and finds the teachers whose salary is greater than 20,000:

// teacherQuery.js

const mysql = require('mysql');

// Create a connection to the database

const connection = mysql.createConnection({

    host: 'localhost',

    user: 'your\_username',

    password: 'your\_password',

    database: 'your\_database'

});

// Connect to the database

connection.connect((err) => {

    if (err) {

        console.error('Error connecting to database:', err);

        return;

    }

    console.log('Connected to database');

});

// Select all records from the Teacher table where salary is greater than 20,000

const sqlQuery = "SELECT \* FROM Teacher WHERE salary > 20000";

// Execute the SQL query

connection.query(sqlQuery, (err, results) => {

    if (err) {

        console.error('Error executing query:', err);

        return;

    }

    console.log('Teachers with salary greater than 20,000:');

    console.log(results);

});

// Close the connection

connection.end();

Replace **'your\_username'**, **'your\_password'**, and **'your\_database'** with your actual MySQL username, password, and database name respectively.

This script connects to your MySQL database, executes the SQL query to select all records from the "Teacher" table where the salary is greater than 20,000, and logs the results to the console.

Save this code in a file named **teacherQuery.js**, and then run it using Node.js:

bash

Copy code

node teacherQuery.js

This will execute the script, connect to your MySQL database, and display the teachers whose salary is greater than 20,000.

**Slip 4**

Using node js create an eLearning System.

Creating a fully functional eLearning system in Node.js within this space is beyond the scope, but I can provide a simplified example to get you started. This example will cover basic user authentication, course management, and content delivery.

Here's a basic structure for the Node.js eLearning system:

1. **Set up the project**:
   * Create a new directory for your project and initialize it with npm (**npm init -y**).
   * Install required packages such as Express.js, Passport.js, and others (**npm install express passport passport-local body-parser express-session**).
2. **Create the server**:
   * Create an **index.js** file to set up the Express server

// index.js

const express = require('express');

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

const session = require('express-session');

const passport = require('passport');

const LocalStrategy = require('passport-local').Strategy;

const app = express();

const port = 3000;

// Middleware

app.use(bodyParser.urlencoded({ extended: true }));

app.use(session({ secret: 'secret\_key', resave: false, saveUninitialized: true }));

app.use(passport.initialize());

app.use(passport.session());

// Dummy user data (replace with a real user database)

const users = [

    { id: 1, username: 'user1', password: 'password1' },

    { id: 2, username: 'user2', password: 'password2' }

];

// Passport local strategy for authentication

passport.use(new LocalStrategy((username, password, done) => {

    const user = users.find(u => u.username === username && u.password === password);

    if (user) {

        return done(null, user);

    } else {

        return done(null, false, { message: 'Incorrect username or password' });

    }

}));

passport.serializeUser((user, done) => {

    done(null, user.id);

});

passport.deserializeUser((id, done) => {

    const user = users.find(u => u.id === id);

    done(null, user);

});

// Routes

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

    res.send('Welcome to the eLearning system');

});

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

    res.send('Login page');

});

app.post('/login', passport.authenticate('local', { successRedirect: '/profile', failureRedirect: '/login' }));

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

    if (req.isAuthenticated()) {

        res.send(`Welcome, ${req.user.username}!`);

    } else {

        res.redirect('/login');

    }

});

// Start the server

app.listen(port, () => {

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

});

Create an HTML form using AngularJS that contain the Student Registration details and validate Student first and last name as it should not contain other than alphabets and age should be between 18 to 50 and display greeting message depending on current time using ng-show (e.g. Good Morning, Good Afternoon, etc.)(Use AJAX)

<!DOCTYPE html>

<html lang="en" ng-app="studentRegistrationApp">

<head>

    <meta charset="UTF-8">

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

    <title>Student Registration Form</title>

    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

    <script>

        angular.module('studentRegistrationApp', [])

            .controller('RegistrationController', function($scope, $http) {

                $scope.student = {};

                $scope.submitForm = function() {

                    $http.post('/submit', $scope.student)

                        .then(function(response) {

                            console.log(response.data);

                        })

                        .catch(function(error) {

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

                        });

                };

                // Function to determine greeting message based on current time

                $scope.getGreetingMessage = function() {

                    var currentTime = new Date().getHours();

                    if (currentTime < 12) {

                        $scope.greetingMessage = "Good Morning";

                    } else if (currentTime < 18) {

                        $scope.greetingMessage = "Good Afternoon";

                    } else {

                        $scope.greetingMessage = "Good Evening";

                    }

                };

                // Call the function on page load

                $scope.getGreetingMessage();

            });

    </script>

</head>

<body ng-controller="RegistrationController">

<h2>Student Registration Form</h2>

<form ng-submit="submitForm()">

    <label for="firstName">First Name:</label>

    <input type="text" id="firstName" name="firstName" ng-model="student.firstName" required pattern="[A-Za-z]+"><br><br>

    <label for="lastName">Last Name:</label>

    <input type="text" id="lastName" name="lastName" ng-model="student.lastName" required pattern="[A-Za-z]+"><br><br>

    <label for="age">Age:</label>

    <input type="number" id="age" name="age" ng-model="student.age" required min="18" max="50"><br><br>

    <input type="submit" value="Register">

</form>

<!-- Display greeting message based on current time -->

<div ng-show="greetingMessage">{{ greetingMessage }}, {{ student.firstName }}</div>

</body>

</html>

**Slip 5**

Create a Node.js file that writes an HTML form, with an upload field.

// uploadFormGenerator.js

const fs = require('fs');

// HTML content for the form

const htmlContent = `

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>File Upload Form</title>

</head>

<body>

    <h2>File Upload Form</h2>

    <form action="/upload" method="post" enctype="multipart/form-data">

        <input type="file" name="file" id="file" required><br><br>

        <input type="submit" value="Upload">

    </form>

</body>

</html>

`;

// Write HTML content to a file

fs.writeFile('uploadForm.html', htmlContent, (err) => {

    if (err) {

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

    } else {

        console.log('HTML form with upload field has been created successfully.');

    }

});

This script generates an HTML file **uploadForm.html** with a form containing a file upload field (**<input type="file">**). You can then serve this HTML file using a web server (e.g., Express.js) to allow users to upload files.

To run this script, save it as **uploadFormGenerator.js** and execute it using Node.js:

bash

Copy code

node uploadFormGenerator.js

After running the script, you'll find the **uploadForm.html** file in the same directory, containing the HTML form with the upload field.

Q.2) Using angular js create a SPA to carry out validation for a username entered in a textbox. If the textbox is blank, alert “Enter username”. If the number of characters is less than three, alert ‟ Username is too short”. If value entered is appropriate the print “Valid username” and password should be minimum 8 characters.

<!DOCTYPE html> <html lang="en" ng-app="validationApp"> <head> <meta charset="UTF-8"> <meta name="viewport" content="width=device-width, initial-scale=1.0"> <title>Username and Password Validation</title> <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script> <script> var app = angular.module('validationApp', []); app.controller('ValidationController', function($scope) { $scope.validateUsername = function() { if (!$scope.username) { alert('Enter username'); } else if ($scope.username.length < 3) { alert('Username is too short'); } else { alert('Valid username'); } }; $scope.validatePassword = function() { if (!$scope.password) { alert('Enter password'); } else if ($scope.password.length < 8) { alert('Password is too short'); } else { alert('Valid password'); } }; }); </script> </head> <body ng-controller="ValidationController"> <h2>Username and Password Validation</h2> <label for="username">Username:</label> <input type="text" id="username" name="username" ng-model="username" ng-blur="validateUsername()"><br><br> <label for="password">Password:</label> <input type="password" id="password" name="password" ng-model="password" ng-blur="validatePassword()"><br><br> </body> </html>

**Slip 6**

Q.1) Write angular JS by using ng-click directive to display an alert message after clicking the element.

<!DOCTYPE html>

<html lang="en" ng-app="alertApp">

<head>

    <meta charset="UTF-8">

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

    <title>Click Alert</title>

    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

</head>

<body>

<h2>Click Alert</h2>

<!-- Use ng-click directive to display alert -->

<button ng-click="showAlert()">Click me</button>

<script>

    var app = angular.module('alertApp', []);

    app.controller('AlertController', function($scope) {

        $scope.showAlert = function() {

            alert('Button clicked!');

        };

    });

</script>

</body>

</html>

Q.2) Create a Node.js file that opens the requested file and returns the content to the client. If anything goes wrong, throw a 404 error.

// fileServer.js

const http = require('http');

const fs = require('fs');

const path = require('path');

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

    // Extract the requested file path from the URL

    const filePath = path.join(\_\_dirname, req.url);

    // Check if the requested file exists

    fs.access(filePath, fs.constants.F\_OK, (err) => {

        if (err) {

            // File not found, send 404 error response

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

            res.end('404 Not Found');

        } else {

            // File found, read its content and send to the client

            fs.readFile(filePath, (err, data) => {

                if (err) {

                    // Error reading file, send 500 error response

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

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

                } else {

                    // Send file content to the client

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

                    res.end(data);

                }

            });

        }

    });

});

const port = 3000;

server.listen(port, () => {

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

});

To use this file server, save the code in a file named **fileServer.js** and run it using Node.js:

bash

Copy code

node fileServer.js

Now, the server will be running on **http://localhost:3000**. You can access files by appending their paths to this URL. For example, if you have a file named **example.txt** in the same directory as the **fileServer.js**, you can access it using **http://localhost:3000/example.txt**. If the file exists, its content will be displayed; otherwise, a 404 error will be shown.

**Slip 7**

Q.1) Create angular JS Application that show the current Date and Time of the System (Use Interval Service

<!DOCTYPE html>

<html lang="en" ng-app="dateTimeApp">

<head>

    <meta charset="UTF-8">

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

    <title>Date and Time Display</title>

    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

</head>

<body ng-controller="DateTimeController">

<h2>Date and Time Display</h2>

<p>{{ dateTime }}</p>

<script>

    angular.module('dateTimeApp', [])

        .controller('DateTimeController', function($scope, $interval) {

            // Function to update date and time every second

            function updateDateTime() {

                $scope.dateTime = new Date();

            }

            // Update date and time initially

            updateDateTime();

            // Update date and time every second using $interval service

            var intervalPromise = $interval(updateDateTime, 1000);

            // Cancel interval when controller is destroyed to avoid memory leaks

            $scope.$on('$destroy', function() {

                $interval.cancel(intervalPromise);

            });

        });

</script>

</body>

</html>

Q.2) Create a node js file named main.js for event-driven application. There should be a main loop that listens for events, and then triggers a callback function when one of those events is detected.

// main.js

// Import the EventEmitter class from the events module

const EventEmitter = require('events');

// Create an instance of the EventEmitter class

const eventEmitter = new EventEmitter();

// Attach event listeners

eventEmitter.on('event1', () => {

    console.log('Event 1 triggered');

});

eventEmitter.on('event2', () => {

    console.log('Event 2 triggered');

});

// Main loop that listens for events

console.log('Listening for events...');

setInterval(() => {

    // Simulate event detection

    const randomEvent = Math.random() < 0.5 ? 'event1' : 'event2';

    // Emit the detected event

    eventEmitter.emit(randomEvent);

}, 2000); // Check for events every 2 seconds

In this example:

* We import the **EventEmitter** class from the built-in **events** module.
* We create an instance of the **EventEmitter** class named **eventEmitter**.
* We attach event listeners for two events: **event1** and **event2**.
* We set up a main loop that listens for events. In this loop, we simulate event detection every 2 seconds and emit a random event (**event1** or **event2**) using the **emit()** method of the **EventEmitter** instance.
* When an event is emitted, the corresponding event listener callback function is triggered, and a message is logged to the console.

To run this file, save it as **main.js** and execute it using Node.js:

bash

Copy code

node main.js

You should see messages logged to the console indicating that events are being triggered and handled.

**Slip 8**

Create a Simple Web Server using node js

// Import the required modules

const http = require('http');

// Define the port on which the server will listen

const PORT = 3000;

// Create a function to handle incoming requests

const requestHandler = (request, response) => {

    // Set the response HTTP header with HTTP status code and Content-Type

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

    // Send the response body (content) to the client

    response.end('Hello World!\n');

};

// Create an instance of the HTTP server

const server = http.createServer(requestHandler);

// Start the server and make it listen for incoming requests

server.listen(PORT, () => {

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

});

* We import the **http** module, which provides functionality to create HTTP servers.
* We define the port number (3000) on which the server will listen for incoming requests.
* We create a function **requestHandler** to handle incoming HTTP requests. This function will be called every time a request is made to the server.
* Inside the **requestHandler** function, we set the HTTP response header with status code 200 (indicating success) and content type as plain text. Then, we send the response body containing the text "Hello World!" to the client.
* We create an instance of the HTTP server using **http.createServer()** and pass the **requestHandler** function as an argument.
* Finally, we start the server by calling **server.listen()** and specify the port number. We also log a message to the console indicating that the server is running.

To run this code, save it in a file (e.g., **server.js**) and execute it using Node.js:

bash

Copy code

node server.js

After running the server, you can access it in your web browser at **http://localhost:3000**, and you should see the message "Hello World!" displayed on the page

Q.2) Using angular js display the 10 student details in Table format (using ng-repeat directive use Array to store data

<!DOCTYPE html>

<html lang="en" ng-app="studentApp">

<head>

    <meta charset="UTF-8">

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

    <title>Student Details</title>

    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

</head>

<body>

<h2>Student Details</h2>

<table border="1">

    <thead>

        <tr>

            <th>Name</th>

            <th>Age</th>

            <th>Grade</th>

        </tr>

    </thead>

    <tbody>

        <tr ng-repeat="student in students">

            <td>{{ student.name }}</td>

            <td>{{ student.age }}</td>

            <td>{{ student.grade }}</td>

        </tr>

    </tbody>

</table>

<script>

    angular.module('studentApp', [])

        .controller('StudentController', function($scope) {

            // Array to store student details

            $scope.students = [

                { name: 'John Doe', age: 20, grade: 'A' },

                { name: 'Jane Smith', age: 22, grade: 'B' },

                { name: 'Alice Johnson', age: 21, grade: 'A-' },

                { name: 'Bob Brown', age: 19, grade: 'B+' },

                { name: 'Emily Davis', age: 20, grade: 'A' },

                { name: 'Michael Wilson', age: 23, grade: 'B-' },

                { name: 'Sarah Taylor', age: 21, grade: 'A' },

                { name: 'David Martinez', age: 22, grade: 'B' },

                { name: 'Jessica Thompson', age: 20, grade: 'A+' },

                { name: 'Andrew Lee', age: 24, grade: 'B' }

            ];

        });

</script>

</body>

</html>

**Slip 9**

Q.1) Create a Node.js file that writes an HTML form, with a concatenate two string.

// Import the required modules

const fs = require('fs');

// Define the HTML content for the form

const htmlContent = `

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>String Concatenation Form</title>

</head>

<body>

<h2>String Concatenation Form</h2>

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

    <label for="string1">Enter string 1:</label>

    <input type="text" id="string1" name="string1"><br><br>

    <label for="string2">Enter string 2:</label>

    <input type="text" id="string2" name="string2"><br><br>

    <input type="submit" value="Concatenate">

</form>

</body>

</html>

`;

// Write the HTML content to a file named concatenateForm.html

fs.writeFile('concatenateForm.html', htmlContent, (err) => {

    if (err) {

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

    } else {

        console.log('HTML form for string concatenation has been created successfully.');

    }

});

This script generates an HTML file **concatenateForm.html** with a form containing two input fields for entering strings and a button to concatenate them.

To run this script, save it as **generateForm.js** and execute it using Node.js:

bash

Copy code

node generateForm.js

After running the script, you'll find the **concatenateForm.html** file in the same directory, containing the HTML form for string concatenation.

Q.2) Create a Node.js file that opens the requested file and returns the content to the client If anything goes wrong, throw a 404 error

const http = require('http');

const fs = require('fs');

const path = require('path');

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

    // Extract the requested file path from the URL

    const filePath = path.join(\_\_dirname, req.url);

    // Check if the requested file exists

    fs.access(filePath, fs.constants.F\_OK, (err) => {

        if (err) {

            // File not found, send 404 error response

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

            res.end('404 Not Found');

        } else {

            // File found, read its content and send to the client

            fs.readFile(filePath, (err, data) => {

                if (err) {

                    // Error reading file, send 500 error response

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

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

                } else {

                    // Determine content type based on file extension

                    let contentType = 'text/plain';

                    const ext = path.extname(filePath);

                    if (ext === '.html') {

                        contentType = 'text/html';

                    } else if (ext === '.css') {

                        contentType = 'text/css';

                    } else if (ext === '.js') {

                        contentType = 'text/javascript';

                    }

                    // Send file content with appropriate content type

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

                    res.end(data);

                }

            });

        }

    });

});

const port = 3000;

server.listen(port, () => {

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

});

This script creates a basic HTTP server using Node.js. It listens for incoming requests, checks if the requested file exists, and sends its content to the client. If the file is not found, it sends a 404 error response. If there is an error reading the file, it sends a 500 error response.

To use this script, save it as **fileServer.js** and run it using Node.js:

bash

Copy code

node fileServer.js

After running the server, you can access files by appending their paths to the server URL (e.g., **http://localhost:3000/example.html**). If the file exists, its content will be displayed; otherwise, a 404 error will be shown.

**Slip 10**

Q.1) Create a Node.js file that demonstrate create college database and table in MySQL

// Import the mysql module

const mysql = require('mysql');

// Create a connection to the MySQL server

const connection = mysql.createConnection({

    host: 'localhost', // Change this to your MySQL server host

    user: 'root',      // Change this to your MySQL username

    password: 'password',  // Change this to your MySQL password

    port: '3306'       // Change this to your MySQL server port if necessary

});

// Connect to the MySQL server

connection.connect((err) => {

    if (err) {

        console.error('Error connecting to MySQL server:', err);

        return;

    }

    console.log('Connected to MySQL server');

    // Create the college database

    connection.query('CREATE DATABASE IF NOT EXISTS college', (err) => {

        if (err) {

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

            connection.end(); // Close the connection

            return;

        }

        console.log('Database "college" created successfully');

        // Use the college database

        connection.query('USE college', (err) => {

            if (err) {

                console.error('Error selecting database:', err);

                connection.end(); // Close the connection

                return;

            }

            console.log('Using database "college"');

            // Create the students table

            connection.query(`CREATE TABLE IF NOT EXISTS students (

                id INT AUTO\_INCREMENT PRIMARY KEY,

                name VARCHAR(255) NOT NULL,

                age INT NOT NULL,

                gender ENUM('Male', 'Female', 'Other') NOT NULL,

                course VARCHAR(255) NOT NULL

            )`, (err) => {

                if (err) {

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

                } else {

                    console.log('Table "students" created successfully');

                }

                // Close the connection

                connection.end();

            });

        });

    });

});

Before running this script, make sure to replace the connection parameters (host, user, password, and port) with your MySQL server details.

To execute this script, save it as **createCollegeDBTable.js** and run it using Node.js:

bash

Copy code

node createCollegeDBTable.js

This script connects to the MySQL server, creates a database named "college", switches to that database, and creates a table named "students" with columns for student ID, name, age, gender, and course.

Q.2 Write node js script to build Your Own Node.js Module. Use require (‘http’) module is a builtin Node module that invokes the functionality of the HTTP library to create a local server. Also use the export statement to make functions in your module available externally. Create a new text file to contain the functions in your module called, “modules.js” and add this function to return today’s date and time.

// modules.js // Function to get today's date and time function getCurrentDateTime() { const currentDate = new Date(); return currentDate.toLocaleString(); } // Export the function to make it available externally module.exports = { getCurrentDateTime: getCurrentDateTime };

// app.js

// Import the module

const myModule = require('./modules');

// Call the exported function to get the current date and time

const dateTime = myModule.getCurrentDateTime();

console.log('Current Date and Time:', dateTime);

Save both files (**modules.js** and **app.js**) in the same directory, and then run the **app.js** script using Node.js:

bash

Copy code

node app.js

This will output the current date and time to the console using the function from the module.

**Slip 11**

**Q1.**) Create a Node.js file that demonstrates create Movie database and table in MySQL

// Import the mysql module

const mysql = require('mysql');

// Create a connection to the MySQL server

const connection = mysql.createConnection({

    host: 'localhost', // Change this to your MySQL server host

    user: 'root',      // Change this to your MySQL username

    password: 'password',  // Change this to your MySQL password

    port: '3306'       // Change this to your MySQL server port if necessary

});

// Connect to the MySQL server

connection.connect((err) => {

    if (err) {

        console.error('Error connecting to MySQL server:', err);

        return;

    }

    console.log('Connected to MySQL server');

    // Create the movie database

    connection.query('CREATE DATABASE IF NOT EXISTS movie', (err) => {

        if (err) {

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

            connection.end(); // Close the connection

            return;

        }

        console.log('Database "movie" created successfully');

        // Use the movie database

        connection.query('USE movie', (err) => {

            if (err) {

                console.error('Error selecting database:', err);

                connection.end(); // Close the connection

                return;

            }

            console.log('Using database "movie"');

            // Create the movies table

            connection.query(`CREATE TABLE IF NOT EXISTS movies (

                id INT AUTO\_INCREMENT PRIMARY KEY,

                title VARCHAR(255) NOT NULL,

                director VARCHAR(255) NOT NULL,

                year INT NOT NULL,

                genre VARCHAR(255) NOT NULL

            )`, (err) => {

                if (err) {

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

                } else {

                    console.log('Table "movies" created successfully');

                }

                // Close the connection

                connection.end();

            });

        });

    });

});

Q.2) Write node js application that transfer a file as an attachment on web and enables browser to prompt the user to download file using express js

Sure, here's how you can create a Node.js application using Express.js to transfer a file as an attachment and prompt the user to download the file:

First, install the required dependencies:

bash

Copy code

npm install express

Then, create a file named **app.js** and add the following code:

// Import required modules

const express = require('express');

const fs = require('fs');

const path = require('path');

// Create an Express application

const app = express();

// Define a route to handle file download

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

    // Path to the file to be downloaded

    const filePath = path.join(\_\_dirname, 'example.txt'); // Change this to the path of your file

    // Check if the file exists

    fs.access(filePath, fs.constants.F\_OK, (err) => {

        if (err) {

            // File not found, send 404 error response

            res.status(404).send('File not found');

        } else {

            // Set headers to prompt download

            res.setHeader('Content-Disposition', 'attachment; filename="example.txt"');

            res.setHeader('Content-Type', 'text/plain');

            // Create a read stream to read the file

            const fileStream = fs.createReadStream(filePath);

            // Pipe the file stream to the response

            fileStream.pipe(res);

        }

    });

});

// Start the server

const port = 3000;

app.listen(port, () => {

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

});

**Slip 12**

Q.1) Create a node.js file that Select all records from the "customers" table, and display the result object on console.

// Import the mysql module

const mysql = require('mysql');

// Create a connection to the MySQL server

const connection = mysql.createConnection({

    host: 'localhost', // Change this to your MySQL server host

    user: 'root',      // Change this to your MySQL username

    password: 'password',  // Change this to your MySQL password

    database: 'your\_database\_name' // Change this to your MySQL database name

});

// Connect to the MySQL server

connection.connect((err) => {

    if (err) {

        console.error('Error connecting to MySQL server:', err);

        return;

    }

    console.log('Connected to MySQL server');

    // Select all records from the "customers" table

    connection.query('SELECT \* FROM customers', (err, rows) => {

        if (err) {

            console.error('Error executing query:', err);

        } else {

            // Display the result object on the console

            console.log('Result:', rows);

        }

        // Close the connection

        connection.end();

    });

});

Before running this script, make sure to replace the connection parameters (host, user, password, and database) with your MySQL server details.

To execute this script, save it as **selectCustomers.js** and run it using Node.js:

bash

Copy code

node selectCustomers.js

This script connects to the MySQL server, selects all records from the "customers" table, and displays the result object on the console.

**Q.**.2) Create an HTML form for Student Feedback Form with Name, Email ID, Mobile No., feedback (Not good, good, very good, excellent) and write a JavaScript to validate all field using Regular Expression

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Student Feedback Form</title>

</head>

<body>

<h2>Student Feedback Form</h2>

<form id="feedbackForm" onsubmit="return validateForm()">

    <label for="name">Name:</label>

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

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

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

    <label for="mobile">Mobile No.:</label>

    <input type="tel" id="mobile" name="mobile" pattern="[0-9]{10}" required><br><br>

    <label for="feedback">Feedback:</label>

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

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

        <option value="Not good">Not good</option>

        <option value="Good">Good</option>

        <option value="Very good">Very good</option>

        <option value="Excellent">Excellent</option>

    </select><br><br>

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

</form>

<script>

    function validateForm() {

        // Regular expression for mobile number (10 digits)

        const mobileRegex = /^[0-9]{10}$/;

        // Get form input values

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

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

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

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

        // Validate name (should not be empty)

        if (name.trim() === '') {

            alert('Please enter your name');

            return false;

        }

        // Validate email (using HTML5 built-in validation)

        if (!document.getElementById('email').checkValidity()) {

            alert('Please enter a valid email address');

            return false;

        }

        // Validate mobile number (should match pattern)

        if (!mobile.match(mobileRegex)) {

            alert('Please enter a valid 10-digit mobile number');

            return false;

        }

        // Validate feedback (should not be empty)

        if (feedback === '') {

            alert('Please select your feedback');

            return false;

        }

        // Form is valid

        return true;

    }

</script>

</body>

</html>

**Slip 13**

Q.1) Create a Node.js file that will convert the output "HELLO WORLD!" into lower-case letters.

Sure, here's a Node.js script that converts the output "HELLO WORLD!" into lowercase letters:

javascript

Copy code

// Convert the output "HELLO WORLD!" to lowercase const output = "HELLO WORLD!".toLowerCase(); // Display the lowercase output console.log(output);

Save this code in a file named **convertToLower.js** and run it using Node.js:

bash

Copy code

node convertToLower.js

This script will output "hello world!" in lowercase letters.

Q.2) Create an HTML form that contain the Student Registration details and write a JavaScript to validate Student first and last name as it should not contain other than alphabets and age should be between 18 to 50

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Student Registration</title>

</head>

<body>

<h2>Student Registration Form</h2>

<form id="registrationForm" onsubmit="return validateForm()">

    <label for="firstName">First Name:</label>

    <input type="text" id="firstName" name="firstName" required><br><br>

    <label for="lastName">Last Name:</label>

    <input type="text" id="lastName" name="lastName" required><br><br>

    <label for="age">Age:</label>

    <input type="number" id="age" name="age" required><br><br>

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

</form>

<script>

function validateForm() {

    // Regular expression for alphabets only

    const nameRegex = /^[A-Za-z]+$/;

    // Get form input values

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

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

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

    // Validate first name (should contain only alphabets)

    if (!firstName.match(nameRegex)) {

        alert('First name should contain only alphabets');

        return false;

    }

    // Validate last name (should contain only alphabets)

    if (!lastName.match(nameRegex)) {

        alert('Last name should contain only alphabets');

        return false;

    }

    // Validate age (should be between 18 and 50)

    if (age < 18 || age > 50) {

        alert('Age should be between 18 and 50');

        return false;

    }

    // Form is valid

    return true;

}

</script>

</body>

</html>

**Slip 14**

Q.1) Create a Simple Web Server using node js.

// Import the required modules

const http = require('http');

// Define the port on which the server will listen

const PORT = 3000;

// Create a function to handle incoming requests

const requestHandler = (request, response) => {

    // Set the response HTTP header with HTTP status code and Content-Type

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

    // Send the response body (content) to the client

    response.end('Hello World!\n');

};

// Create an instance of the HTTP server

const server = http.createServer(requestHandler);

// Start the server and make it listen for incoming requests

server.listen(PORT, () => {

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

});

* We import the **http** module, which provides functionality to create HTTP servers.
* We define the port number (3000) on which the server will listen for incoming requests.
* We create a function **requestHandler** to handle incoming HTTP requests. This function will be called every time a request is made to the server.
* Inside the **requestHandler** function, we set the HTTP response header with status code 200 (indicating success) and content type as plain text. Then, we send the response body containing the text "Hello World!" to the client.
* We create an instance of the HTTP server using **http.createServer()** and pass the **requestHandler** function as an argument.
* Finally, we start the server by calling **server.listen()** and specify the port number. We also log a message to the console indicating that the server is running.

To run this code, save it in a file (e.g., **server.js**) and execute it using Node.js:

bash

Copy code

node server.js

After running the server, you can access it in your web browser at **http://localhost:3000**, and you should see the message "Hello World!" displayed on the page

Q.2) Create an HTML form that contain the Employee Registration details and write a JavaScript to validate DOB, Joining Date, and Salary

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Employee Registration Form</title>

</head>

<body>

<h2>Employee Registration Form</h2>

<form id="registrationForm" onsubmit="return validateForm()">

    <label for="name">Name:</label>

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

    <label for="dob">Date of Birth:</label>

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

    <label for="joiningDate">Joining Date:</label>

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

    <label for="salary">Salary:</label>

**Slip 15**

Q.1) Create a node.js file that Select all records from the "students" table, and display the result object on console.

// Import the mysql module

const mysql = require('mysql');

// Create a connection to the MySQL server

const connection = mysql.createConnection({

    host: 'localhost', // Change this to your MySQL server host

    user: 'root',      // Change this to your MySQL username

    password: 'password',  // Change this to your MySQL password

    database: 'your\_database\_name' // Change this to your MySQL database name

});

// Connect to the MySQL server

connection.connect((err) => {

    if (err) {

        console.error('Error connecting to MySQL server:', err);

        return;

    }

    console.log('Connected to MySQL server');

    // Select all records from the "students" table

    connection.query('SELECT \* FROM students', (err, rows) => {

        if (err) {

            console.error('Error executing query:', err);

        } else {

            // Display the result object on the console

            console.log('Result:', rows);

        }

        // Close the connection

        connection.end();

    });

});

Before running this script, make sure to replace the connection parameters (host, user, password, and database) with your MySQL server details.

Save this code in a file named **selectStudents.js** and run it using Node.js:

bash

Copy code

node selectStudents.js

This script will connect to the MySQL server, select all records from the "students" table, and display the result object on the console.

Q.2) Create an HTML form for Employee and write a JavaScript to validate name, email ID, mobile number, department, joining date using Regular Expression

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Employee Details Form</title>

</head>

<body>

<h2>Employee Details Form</h2>

<form id="employeeForm" onsubmit="return validateForm()">

    <label for="name">Name:</label>

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

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

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

    <label for="mobile">Mobile Number:</label>

    <input type="text" id="mobile" name="mobile" required><br><br>

    <label for="department">Department:</label>

    <input type="text" id="department" name="department" required><br><br>

    <label for="joiningDate">Joining Date:</label>

    <input type="text" id="joiningDate" name="joiningDate" placeholder="YYYY-MM-DD" required><br><br>

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

</form>

<script>

function validateForm() {

    // Regular expressions for validation

    const nameRegex = /^[A-Za-z]+$/;

    const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

    const mobileRegex = /^[0-9]{10}$/;

    const dateRegex = /^\d{4}-\d{2}-\d{2}$/;

    // Get form input values

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

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

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

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

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

    // Validate name (should contain only alphabets)

    if (!name.match(nameRegex)) {

        alert('Name should contain only alphabets');

        return false;

    }

    // Validate email

    if (!email.match(emailRegex)) {

        alert('Please enter a valid email address');

        return false;

    }

    // Validate mobile number (should be 10 digits)

    if (!mobile.match(mobileRegex)) {

        alert('Please enter a valid 10-digit mobile number');

        return false;

    }

    // Validate department (should not be empty)

    if (department.trim() === '') {

        alert('Please enter department');

        return false;

    }

    // Validate joining date (should be in YYYY-MM-DD format)

    if (!joiningDate.match(dateRegex)) {

        alert('Please enter a valid joining date in YYYY-MM-DD format');

        return false;

    }

    // Form is valid

    return true;

}

</script>

</body>

</html>

**Slip 16**

Q.1) Using node js create a Recipe Book.

To create a Recipe Book using Node.js, you can implement a simple command-line application that allows users to manage recipes. Below is an example of how you can structure your Node.js application for a Recipe Book:

1. **Initialize the Project:** Create a new directory for your project and initialize it with **npm init**.
2. **Install Dependencies:** Install any dependencies you may need. For a basic Recipe Book, you may only need the built-in **fs** module for file system operations.
3. **Create the File Structure:** Create the necessary directories and files for your project, such as **recipes** directory to store recipe files and a main JavaScript file (**index.js**) to contain your application logic.
4. **Implement the Application:** Write the logic to allow users to add, view, edit, and delete recipes.
5. **Testing:** Test your application to ensure it works as expected.

const fs = require('fs');

const readline = require('readline');

const rl = readline.createInterface({

    input: process.stdin,

    output: process.stdout

});

const recipesDir = './recipes/';

// Create recipes directory if it doesn't exist

if (!fs.existsSync(recipesDir)) {

    fs.mkdirSync(recipesDir);

}

function addRecipe() {

    rl.question('Enter recipe name: ', (name) => {

        rl.question('Enter recipe ingredients: ', (ingredients) => {

            const recipe = {

                name: name,

                ingredients: ingredients

            };

            const fileName = `${recipesDir}${name.toLowerCase().replace(/\s/g, '-')}.json`;

            fs.writeFile(fileName, JSON.stringify(recipe), (err) => {

                if (err) throw err;

                console.log('Recipe added successfully!');

                rl.close();

            });

        });

    });

}

function viewRecipe() {

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

        if (err) throw err;

        console.log('Recipes:');

        files.forEach((file, index) => {

            console.log(`${index + 1}. ${file.replace('.json', '').replace(/-/g, ' ')}`);

        });

        rl.question('Enter recipe number to view: ', (num) => {

            const fileName = files[num - 1];

            fs.readFile(`${recipesDir}${fileName}`, 'utf8', (err, data) => {

                if (err) throw err;

                console.log('\nRecipe Details:');

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

                rl.close();

            });

        });

    });

}

function mainMenu() {

    console.log('\nRecipe Book');

    console.log('1. Add Recipe');

    console.log('2. View Recipe');

    console.log('3. Exit');

    rl.question('Enter your choice: ', (choice) => {

        switch (choice) {

            case '1':

                addRecipe();

                break;

            case '2':

                viewRecipe();

                break;

            case '3':

                rl.close();

                break;

            default:

                console.log('Invalid choice!');

                mainMenu();

        }

    });

}

mainMenu();

 Q.2) Create a js file named main.js for event-driven application. There should be a main loop that listens for events, and then triggers a call-back function when one of those events is detected

main.js

// Import the events module

const EventEmitter = require('events');

// Create an EventEmitter object

const eventEmitter = new EventEmitter();

// Attach event listener for 'customEvent'

eventEmitter.on('customEvent', (arg1, arg2) => {

    console.log('Custom event received:', arg1, arg2);

});

// Main loop to listen for events

console.log('Main loop started. Press CTRL+C to exit.');

process.stdin.on('data', (data) => {

    // Trigger 'customEvent' when user enters input

    eventEmitter.emit('customEvent', 'Hello', data.toString().trim());

});

In this **main.js** file:

* We import the built-in **events** module and create an EventEmitter object named **eventEmitter**.
* We attach an event listener for the custom event named **'customEvent'**. When this event is emitted, the callback function is triggered, which logs the event arguments to the console.
* We start a main loop to listen for events. In this example, we use **process.stdin.on('data', ...)** to listen for user input. When the user enters input, we emit the custom event **'customEvent'** with the input data as arguments.
* The main loop continues to run until the user presses CTRL+C to exit the program.

You can run this **main.js** file using Node.js:

bash

Copy code

node main.js

Then, you can input data and observe the output as the custom event is triggered

**Slip 17**

Q.1) Using angular js Create a SPA that show Syllabus content of all subjects of M.Sc (CS) Sem-II (use ng-view

1. **Set up AngularJS and ngRoute:** Include the AngularJS and ngRoute scripts in your HTML file.
2. **Define Routes:** Configure routes for your SPA using the $routeProvider.
3. **Create Views:** Create views for each route using ngView directive.
4. **Implement Controllers:** Implement controllers to handle data and logic for each view.

Here's a basic example:

**index.html:**

html

Copy code

<!DOCTYPE html> <html lang="en" ng-app="myApp"> <head> <meta charset="UTF-8"> <title>M.Sc (CS) Sem-II Syllabus</title> <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script> <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular-route.js"></script> <script src="app.js"></script> </head> <body> <h2>M.Sc (CS) Sem-II Syllabus</h2> <div ng-view></div> </body> </html>

**app.js:**

javascript

Copy code

var app = angular.module('myApp', ['ngRoute']); app.config(function($routeProvider) { $routeProvider .when('/', { templateUrl: 'home.html', controller: 'HomeController' }) .when('/subject/:subjectName', { templateUrl: 'subject.html', controller: 'SubjectController' }) .otherwise({ redirectTo: '/' }); }); app.controller('HomeController', function($scope) { // Home controller logic }); app.controller('SubjectController', function($scope, $routeParams) { // Subject controller logic $scope.subjectName = $routeParams.subjectName; // You can fetch syllabus content for the selected subject here // For demo purposes, just display the subject name });

**home.html:**

html

Copy code

<p>Select a subject:</p> <ul> <li><a href="#/subject/Mathematics">Mathematics</a></li> <li><a href="#/subject/Computer Science">Computer Science</a></li> <!-- Add more subjects as needed --> </ul>

**subject.html:**

html

Copy code

<h3>{{subjectName}} Syllabus:</h3> <!-- Display syllabus content here -->

In this example:

* We define a main AngularJS module named 'myApp' and inject the 'ngRoute' module as a dependency.
* We configure routes using the $routeProvider service. The '/' route is for the home page and '/subject/:subjectName' route is for displaying syllabus content for a specific subject.
* We define controllers for the home page and subject page to handle logic and data.
* We create views using ngView directive to render the content of each route.
* The home page (**home.html**) displays a list of subjects with links to view their syllabus.
* The subject page (**subject.html**) displays the syllabus content for the selected subject.

You can expand this example by adding more subjects and fetching their syllabus content from a backend server.

Q.2) Using angular js display the 10 student details in Table format (using ng-repeat directive use Array to store data

<!DOCTYPE html>

<html lang="en" ng-app="myApp">

<head>

<meta charset="UTF-8">

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

<title>Student Details</title>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

</head>

<body>

<h2>Student Details</h2>

<table border="1">

    <thead>

        <tr>

            <th>Student ID</th>

            <th>Name</th>

            <th>Age</th>

            <th>Grade</th>

        </tr>

    </thead>

    <tbody>

        <tr ng-repeat="student in students">

            <td>{{ student.id }}</td>

            <td>{{ student.name }}</td>

            <td>{{ student.age }}</td>

            <td>{{ student.grade }}</td>

        </tr>

    </tbody>

</table>

<script>

var app = angular.module('myApp', []);

app.controller('MainController', function($scope) {

    $scope.students = [

        { id: 1, name: 'John', age: 20, grade: 'A' },

        { id: 2, name: 'Emma', age: 22, grade: 'B' },

        { id: 3, name: 'Michael', age: 21, grade: 'A+' },

        { id: 4, name: 'Sophia', age: 19, grade: 'B+' },

        { id: 5, name: 'William', age: 20, grade: 'A-' },

        { id: 6, name: 'Olivia', age: 21, grade: 'C' },

        { id: 7, name: 'James', age: 22, grade: 'A' },

        { id: 8, name: 'Ava', age: 20, grade: 'B' },

        { id: 9, name: 'Alexander', age: 23, grade: 'A+' },

        { id: 10, name: 'Mia', age: 19, grade: 'B-' }

    ];

});

</script>

</body>

</html>

**Slip 18**

Q.1) Using node js create a User Login System.

Creating a user login system in Node.js involves several steps, including setting up a server, handling user registration and authentication, and managing user sessions. Below is a basic example of how you can create a user login system using Node.js, Express.js, and a simple in-memory storage for user data.

First, make sure you have Node.js installed on your system. Then, create a directory for your project and initialize it with npm (**npm init -y**). After that, install Express.js (**npm install express**), **body-parser** (**npm install body-parser**), and **express-session** (**npm install express-session**).

Here's the code for your Node.js user login system:

// index.js

const express = require('express');

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

const session = require('express-session');

const app = express();

const port = 3000;

// Middleware

app.use(bodyParser.urlencoded({ extended: true }));

app.use(session({

    secret: 'secret\_key', // Change this to a random string

    resave: false,

    saveUninitialized: true

}));

// In-memory user database (Replace this with a real database in production)

const users = [

    { id: 1, username: 'user1', password: 'password1' },

    { id: 2, username: 'user2', password: 'password2' }

];

// Routes

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

    res.sendFile(\_\_dirname + '/index.html');

});

// Login route

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

    const { username, password } = req.body;

    const user = users.find(u => u.username === username && u.password === password);

    if (user) {

        req.session.userId = user.id;

        res.send('Login successful!');

    } else {

        res.send('Invalid username or password.');

    }

});

// Logout route

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

    req.session.destroy(err => {

        if (err) {

            console.error('Error destroying session:', err);

        }

        res.redirect('/');

    });

});

// Protected route (requires login)

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

    if (!req.session.userId) {

        return res.redirect('/');

    }

    const user = users.find(u => u.id === req.session.userId);

    if (!user) {

        return res.send('User not found.');

    }

    res.send(`Welcome, ${user.username}! <br><a href="/logout">Logout</a>`);

});

// Start the server

app.listen(port, () => {

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

});

<!-- index.html -->

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>User Login</title>

</head>

<body>

    <h2>User Login</h2>

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

        <label for="username">Username:</label>

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

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

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

        <input type="submit" value="Login">

    </form>

</body>

</html>

Q.2) Create a node.js file that Select all records from the "customers" table, and find the customers whose name starts from ‘A’.

// Import the mysql module

const mysql = require('mysql');

// Create a connection to the MySQL server

const connection = mysql.createConnection({

    host: 'localhost', // Change this to your MySQL server host

    user: 'root',      // Change this to your MySQL username

    password: 'password',  // Change this to your MySQL password

    database: 'your\_database\_name' // Change this to your MySQL database name

});

// Connect to the MySQL server

connection.connect((err) => {

    if (err) {

        console.error('Error connecting to MySQL server:', err);

        return;

    }

    console.log('Connected to MySQL server');

    // Select all records from the "customers" table

    connection.query('SELECT \* FROM customers', (err, rows) => {

        if (err) {

            console.error('Error executing query:', err);

        } else {

            // Filter customers whose name starts with 'A'

            const customersWithA = rows.filter(customer => customer.name.startsWith('A'));

            // Display customers whose name starts with 'A'

            console.log('Customers whose name starts with "A":', customersWithA);

        }

        // Close the connection

        connection.end();

    });

});

**Slip 19**

Q.1) Create a Node.js file that will convert the output "Hello World!" into upper-case letters.

Here's a Node.js script that converts the output "Hello World!" into uppercase letters:

javascript

Copy code

// Convert the output "Hello World!" to uppercase const output = "Hello World!".toUpperCase(); // Display the uppercase output console.log(output);

Save this code in a file named **convertToUpper.js** and run it using Node.js:

bash

Copy code

node convertToUpper.js

This script will output "HELLO WORLD!" in uppercase letters

Q.2) Using angular js create a SPA to accept the details such as name, mobile number, pin code and email address and make validation. Name should contain character only, address should contain SPPU M.Sc. Computer Science Syllabus 2023-24, mobile number should contain only 10 digit, Pin code should contain only 6 digit, email id should contain only one @, . Symbol

<!DOCTYPE html>

<html lang="en" ng-app="myApp">

<head>

<meta charset="UTF-8">

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

<title>Details Form</title>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

</head>

<body>

<h2>Details Form</h2>

<div ng-controller="FormController">

    <form ng-submit="submitForm()" novalidate>

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

        <input type="text" id="name" name="name" ng-model="formData.name" required pattern="[A-Za-z ]+"><br><br>

        <label for="mobile">Mobile Number:</label>

        <input type="text" id="mobile" name="mobile" ng-model="formData.mobile" required pattern="[0-9]{10}"><br><br>

        <label for="pincode">Pin Code:</label>

        <input type="text" id="pincode" name="pincode" ng-model="formData.pincode" required pattern="[0-9]{6}"><br><br>

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

        <input type="email" id="email" name="email" ng-model="formData.email" required pattern="[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$"><br><br>

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

    </form>

    <div ng-show="submitted">

        <h3>Form Data:</h3>

        <p>Name: {{ formData.name }}</p>

        <p>Mobile Number: {{ formData.mobile }}</p>

        <p>Pin Code: {{ formData.pincode }}</p>

        <p>Email Address: {{ formData.email }}</p>

    </div>

</div>

<script>

var app = angular.module('myApp', []);

app.controller('FormController', function($scope) {

    $scope.formData = {};

    $scope.submitted = false;

    $scope.submitForm = function() {

        // Set submitted flag to true to show form data

        $scope.submitted = true;

    };

});

</script>

</body>

</html>

**Slip 20**

Q.1) Create a Node.js file that demonstrate create student database and table in MySQL

// Import the mysql module

const mysql = require('mysql');

// Create a connection to the MySQL server

const connection = mysql.createConnection({

    host: 'localhost', // Change this to your MySQL server host

    user: 'root',      // Change this to your MySQL username

    password: 'password'  // Change this to your MySQL password

});

// Connect to the MySQL server

connection.connect((err) => {

    if (err) {

        console.error('Error connecting to MySQL server:', err);

        return;

    }

    console.log('Connected to MySQL server');

    // Create the student database

    connection.query('CREATE DATABASE IF NOT EXISTS student\_database', (err) => {

        if (err) {

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

        } else {

            console.log('Student database created or already exists');

            // Use the student database

            connection.query('USE student\_database', (err) => {

                if (err) {

                    console.error('Error using database:', err);

                } else {

                    console.log('Using student database');

                    // Create the student table

                    const createTableQuery = `

                        CREATE TABLE IF NOT EXISTS students (

                            id INT AUTO\_INCREMENT PRIMARY KEY,

                            name VARCHAR(255) NOT NULL,

                            age INT NOT NULL,

                            grade VARCHAR(10) NOT NULL

                        )

                    `;

                    connection.query(createTableQuery, (err) => {

                        if (err) {

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

                        } else {

                            console.log('Student table created or already exists');

                        }

                        // Close the connection

                        connection.end();

                    });

                }

            });

        }

    });

});

// Import the mysql module

const mysql = require('mysql');

// Create a connection to the MySQL server

const connection = mysql.createConnection({

    host: 'localhost', // Change this to your MySQL server host

    user: 'root',      // Change this to your MySQL username

    password: 'password'  // Change this to your MySQL password

});

// Connect to the MySQL server

connection.connect((err) => {

    if (err) {

        console.error('Error connecting to MySQL server:', err);

        return;

    }

    console.log('Connected to MySQL server');

    // Create the student database

    connection.query('CREATE DATABASE IF NOT EXISTS student\_database', (err) => {

        if (err) {

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

        } else {

            console.log('Student database created or already exists');

            // Use the student database

            connection.query('USE student\_database', (err) => {

                if (err) {

                    console.error('Error using database:', err);

                } else {

                    console.log('Using student database');

                    // Create the student table

                    const createTableQuery = `

                        CREATE TABLE IF NOT EXISTS students (

                            id INT AUTO\_INCREMENT PRIMARY KEY,

                            name VARCHAR(255) NOT NULL,

                            age INT NOT NULL,

                            grade VARCHAR(10) NOT NULL

                        )

                    `;

                    connection.query(createTableQuery, (err) => {

                        if (err) {

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

                        } else {

                            console.log('Student table created or already exists');

                        }

                        // Close the connection

                        connection.end();

                    });

                }

            });

        }

    });

});

**Q.**.2) Using angular js create a SPA to carry out validation for a username entered in a textbox. If the textbox is blank, alert “Enter username”. If the number of characters is less than three, alert ‟ Username is too short”. If value entered is appropriate the print “Valid username” and password should be minimum 8 characters

<!DOCTYPE html>

<html lang="en" ng-app="validationApp">

<head>

<meta charset="UTF-8">

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

<title>Username Validation</title>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

</head>

<body>

<h2>Username Validation</h2>

<div ng-controller="ValidationController">

    <label for="username">Username:</label>

    <input type="text" id="username" name="username" ng-model="username" ng-change="validateUsername()"><br><br>

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

    <input type="password" id="password" name="password" ng-model="password" ng-change="validatePassword()"><br><br>

    <button ng-click="validateForm()">Validate</button>

</div>

<script>

var app = angular.module('validationApp', []);

app.controller('ValidationController', function($scope) {

    $scope.username = '';

    $scope.password = '';

    $scope.validateUsername = function() {

        if (!$scope.username) {

            $scope.usernameError = 'Enter username';

        } else if ($scope.username.length < 3) {

            $scope.usernameError = 'Username is too short';

        } else {

            $scope.usernameError = '';

        }

    };

    $scope.validatePassword = function() {

        if ($scope.password.length < 8) {

            $scope.passwordError = 'Password is too short';

        } else {

            $scope.passwordError = '';

        }

    };

    $scope.validateForm = function() {

        $scope.validateUsername();

        $scope.validatePassword();

        if (!$scope.usernameError && !$scope.passwordError) {

            alert('Valid username and password');

        }

    };

});

</script>

</body>

</html>

**Slip 21**

Q.1) Create a Node.js file that demonstrate create Movie database and table in MySQL'

// Import the mysql module

const mysql = require('mysql');

// Create a connection to the MySQL server

const connection = mysql.createConnection({

    host: 'localhost', // Change this to your MySQL server host

    user: 'root',      // Change this to your MySQL username

    password: 'password'  // Change this to your MySQL password

});

// Connect to the MySQL server

connection.connect((err) => {

    if (err) {

        console.error('Error connecting to MySQL server:', err);

        return;

    }

    console.log('Connected to MySQL server');

    // Create the movie database

    connection.query('CREATE DATABASE IF NOT EXISTS movie\_database', (err) => {

        if (err) {

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

        } else {

            console.log('Movie database created or already exists');

            // Use the movie database

            connection.query('USE movie\_database', (err) => {

                if (err) {

                    console.error('Error using database:', err);

                } else {

                    console.log('Using movie database');

                    // Create the movie table

                    const createTableQuery = `

                        CREATE TABLE IF NOT EXISTS movies (

                            id INT AUTO\_INCREMENT PRIMARY KEY,

                            title VARCHAR(255) NOT NULL,

                            director VARCHAR(255),

                            release\_year INT

                        )

                    `;

                    connection.query(createTableQuery, (err) => {

                        if (err) {

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

                        } else {

                            console.log('Movie table created or already exists');

                        }

                        // Close the connection

                        connection.end();

                    });

                }

            });

        }

    });

});

Q.2) Write node js application that transfer a file as an attachment on web and enables browser to prompt the user to download file using express js.

To create a Node.js application using Express.js that transfers a file as an attachment on the web and enables the browser to prompt the user to download the file, you can use the **express** framework along with Node.js's built-in **fs** (file system) module. Below is an example:

1. First, install Express.js if you haven't already done so:

bash

Copy code

npm install express

1. Create a directory for your project and navigate into it:

bash

Copy code

mkdir file-download-app cd file-download-app

1. Create a sample file to be downloaded:

bash

Copy code

echo "Hello, this is a sample file!" > sample.txt

1. Create a Node.js script (**app.js**):

const express = require('express');

const fs = require('fs');

const path = require('path');

const app = express();

const port = 3000;

// Define route to download the file

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

    // Get the file path

    const filePath = path.join(\_\_dirname, 'sample.txt');

    // Set headers to force download

    res.setHeader('Content-Disposition', 'attachment; filename=sample.txt');

    res.setHeader('Content-Type', 'text/plain');

    // Create a read stream to the file

    const fileStream = fs.createReadStream(filePath);

    // Pipe the file stream to the response object

    fileStream.pipe(res);

});

// Start the server

app.listen(port, () => {

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

});

**Slip 22**

Q.1) Using node js create an Employee Registration Form validation.

  To create an Employee Registration Form validation using Node.js, you can use Express.js along with middleware such as **express-validator** to validate form data. Below is an example of how you can achieve this:

1. First, install Express.js and express-validator:

bash

Copy code

npm install express express-validator

1. Create a directory for your project and navigate into it:

mkdir employee-registration-form

cd employee-registration-form

const express = require('express'); const { body, validationResult } = require('express-validator'); const app = express(); const port = 3000; // Middleware to parse JSON and form data app.use(express.json()); app.use(express.urlencoded({ extended: true })); // Employee registration form validation middleware const validateEmployeeRegistrationForm = [ body('name').trim().isLength({ min: 3 }).withMessage('Name must be at least 3 characters'), body('email').trim().isEmail().withMessage('Invalid email address'), body('age').trim().isInt({ min: 18, max: 60 }).withMessage('Age must be between 18 and 60'), body('department').trim().notEmpty().withMessage('Department is required'), body('salary').trim().isNumeric().withMessage('Salary must be a number') ]; // Route for employee registration form app.post('/register', validateEmployeeRegistrationForm, (req, res) => { // Check for validation errors const errors = validationResult(req); if (!errors.isEmpty()) { return res.status(400).json({ errors: errors.array() }); } // If validation passes, process the form data // Here you can save the employee details to a database or perform other actions const { name, email, age, department, salary } = req.body; res.status(200).json({ message: 'Employee registered successfully', data: { name, email, age, department, salary } }); }); // Start the server app.listen(port, () => { console.log(`Server is running on http://localhost:${port}`); });

Q.2) Create a js file named main.js for event-driven application. There should be a main loop that listens for events, and then triggers a callback function when one of those events is detected.

// Define an EventEmitter class

class EventEmitter {

    constructor() {

        this.events = {};

    }

    // Register an event listener for a specific event

    on(event, listener) {

        if (!this.events[event]) {

            this.events[event] = [];

        }

        this.events[event].push(listener);

    }

    // Emit an event, triggering all registered listeners for that event

    emit(event, ...args) {

        if (this.events[event]) {

            this.events[event].forEach(listener => listener(...args));

        }

    }

}

// Create an instance of EventEmitter

const eventEmitter = new EventEmitter();

// Register event listeners

eventEmitter.on('message', (message) => {

    console.log('Received message:', message);

});

eventEmitter.on('error', (error) => {

    console.error('Error:', error);

});

// Simulate events being emitted

setTimeout(() => {

    eventEmitter.emit('message', 'Hello, world!');

}, 1000);

setTimeout(() => {

    eventEmitter.emit('error', 'Something went wrong');

}, 2000);

* We define an **EventEmitter** class that allows registering event listeners (**on** method) and emitting events (**emit** method).
* We create an instance of **EventEmitter** named **eventEmitter**.
* We register two event listeners for events named 'message' and 'error'.
* We simulate events being emitted after certain intervals using **setTimeout**.
* When the 'message' event is emitted, the corresponding listener logs the received message to the console.
* When the 'error' event is emitted, the corresponding listener logs the error message to the console.

You can run this JavaScript file using Node.js:

bash

Copy code

node main.js

This will output:

yaml

Copy code

Received message: Hello, world! Error: Something went wrong

This demonstrates an event-driven application where events are emitted and listeners are triggered

**Slip 23**

Q.1) Write node js script to interact with the file system, and serve a web page from a File

html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Simple Web Page</title>

</head>

<body>

<h1>Hello from Node.js!</h1>

<p>This is a simple web page served by Node.js.</p>

</body>

</html>

js

const http = require('http');

const fs = require('fs');

const path = require('path');

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

    // Load the HTML file

    const filePath = path.join(\_\_dirname, 'index.html');

    fs.readFile(filePath, (err, data) => {

        if (err) {

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

            res.end('Internal Server Error');

            return;

        }

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

        res.end(data);

    });

});

const port = 3000;

server.listen(port, () => {

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

});

Q.2) Write node js script to build Your Own Node.js Module. Use require (“http”) module is a builtin Node module that invokes the functionality of the HTTP library to create a local server. Also use the export statement to make functions in your module available externally. Create a new text file to contain the functions in your module called, “modules.js” and add this function to return today’s date and time.

1. Create a new text file named **modules.js**:

javascript

Copy code

// modules.js // Function to return today's date and time function getCurrentDateTime() { const currentDate = new Date(); return currentDate.toLocaleString(); } // Export the getCurrentDateTime function to make it available externally module.exports.getCurrentDateTime = getCurrentDateTime;

1. Create another Node.js script (**main.js**) where you will use the module:

javascript

Copy code

// main.js // Require the modules.js file to use the getCurrentDateTime function const myModule = require('./modules'); // Use the getCurrentDateTime function from the module const dateTime = myModule.getCurrentDateTime(); console.log('Today\'s date and time:', dateTime);

1. Run the **main.js** script:

bash

Copy code

node main.js

This will output something like:

vbnet

Copy code

Today's date and time: 5/5/2024, 3:35:17 PM

Explanation:

* In the **modules.js** file, we define a function **getCurrentDateTime** that returns today's date and time using the **Date** object.
* We use the **module.exports** to export the **getCurrentDateTime** function, making it available externally.
* In the **main.js** file, we require the **modules.js** file using the **require** function, which loads the exported function.
* We then use the **getCurrentDateTime** function from the module to get today's date and time.
* Finally, we log the date and time to the consol

**Slip 24**

Q.1) Using node js create an eLearning System.

Creating a fully functional eLearning system in Node.js within this space is beyond the scope, but I can provide a simplified example to get you started. This example will cover basic user authentication, course management, and content delivery.

Here's a basic structure for the Node.js eLearning system:

1. **Set up the project**:
   * Create a new directory for your project and initialize it with npm (**npm init -y**).
   * Install required packages such as Express.js, Passport.js, and others (**npm install express passport passport-local body-parser express-session**).
2. **Create the server**:
   * Create an **index.js** file to set up the Express server

// index.js

const express = require('express');

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

const session = require('express-session');

const passport = require('passport');

const LocalStrategy = require('passport-local').Strategy;

const app = express();

const port = 3000;

// Middleware

app.use(bodyParser.urlencoded({ extended: true }));

app.use(session({ secret: 'secret\_key', resave: false, saveUninitialized: true }));

app.use(passport.initialize());

app.use(passport.session());

// Dummy user data (replace with a real user database)

const users = [

    { id: 1, username: 'user1', password: 'password1' },

    { id: 2, username: 'user2', password: 'password2' }

];

// Passport local strategy for authentication

passport.use(new LocalStrategy((username, password, done) => {

    const user = users.find(u => u.username === username && u.password === password);

    if (user) {

        return done(null, user);

    } else {

        return done(null, false, { message: 'Incorrect username or password' });

    }

}));

passport.serializeUser((user, done) => {

    done(null, user.id);

});

passport.deserializeUser((id, done) => {

    const user = users.find(u => u.id === id);

    done(null, user);

});

// Routes

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

    res.send('Welcome to the eLearning system');

});

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

    res.send('Login page');

});

app.post('/login', passport.authenticate('local', { successRedirect: '/profile', failureRedirect: '/login' }));

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

    if (req.isAuthenticated()) {

        res.send(`Welcome, ${req.user.username}!`);

    } else {

        res.redirect('/login');

    }

});

// Start the server

app.listen(port, () => {

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

});

Q.2) Using angular js create a SPA to carry out validation for a username entered in a textbox. If the textbox is blank, alert “Enter username”. If the number of characters is less than three, alert ‟ Username is too short”. If value entered is appropriate the print “Valid username” and password should be minimum 8 characters

<!DOCTYPE html>

<html lang="en" ng-app="validationApp">

<head>

<meta charset="UTF-8">

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

<title>Username Validation</title>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

</head>

<body>

<h2>Username Validation</h2>

<div ng-controller="ValidationController">

    <label for="username">Username:</label>

    <input type="text" id="username" name="username" ng-model="username" ng-change="validateUsername()"><br><br>

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

    <input type="password" id="password" name="password" ng-model="password" ng-change="validatePassword()"><br><br>

    <button ng-click="validateForm()">Validate</button>

</div>

<script>

var app = angular.module('validationApp', []);

app.controller('ValidationController', function($scope) {

    $scope.username = '';

    $scope.password = '';

    $scope.validateUsername = function() {

        if (!$scope.username) {

            alert('Enter username');

        } else if ($scope.username.length < 3) {

            alert('Username is too short');

        } else {

            alert('Valid username');

        }

    };

    $scope.validatePassword = function() {

        if ($scope.password.length < 8) {

            alert('Password is too short');

        }

    };

    $scope.validateForm = function() {

        $scope.validateUsername();

        $scope.validatePassword();

    };

});

</script>

</body>

</html>

**Slip 25**

Q.1) Create an angular JS Application that shows the location of the current web page.

 <!DOCTYPE html>

<html lang="en" ng-app="locationApp">

<head>

<meta charset="UTF-8">

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

<title>Current Location</title>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

</head>

<body>

<h2>Current Location</h2>

<div ng-controller="LocationController">

    <p>The current location is: <strong>{{ currentLocation }}</strong></p>

</div>

<script>

var app = angular.module('locationApp', []);

app.controller('LocationController', function($scope, $location) {

    $scope.currentLocation = $location.absUrl();

});

</script>

</body>

</html>

Q.2) Create a js file named main.js for event-driven application. There should be a main loop that listens for events, and then triggers a callback function when one of those events is detected.

// Define an EventEmitter class

class EventEmitter {

    constructor() {

        this.events = {};

    }

    // Method to register event listeners for a specific event

    on(event, listener) {

        if (!this.events[event]) {

            this.events[event] = [];

        }

        this.events[event].push(listener);

    }

    // Method to emit an event, triggering all registered listeners for that event

    emit(event, ...args) {

        if (this.events[event]) {

            this.events[event].forEach(listener => listener(...args));

        }

    }

}

// Create an instance of EventEmitter

const eventEmitter = new EventEmitter();

// Register event listeners

eventEmitter.on('message', (message) => {

    console.log('Received message:', message);

});

eventEmitter.on('error', (error) => {

    console.error('Error:', error);

});

// Main loop that listens for events and triggers callback functions

setInterval(() => {

    // Simulate events being emitted

    eventEmitter.emit('message', 'Hello, world!');

    eventEmitter.emit('error', 'Something went wrong');

}, 2000);

In this example:

* We define an **EventEmitter** class that allows registering event listeners (**on** method) and emitting events (**emit** method).
* We create an instance of **EventEmitter** named **eventEmitter**.
* We register two event listeners for events named 'message' and 'error'.
* We use **setInterval** to simulate events being emitted every 2 seconds.
* When the 'message' event is emitted, the corresponding listener logs the received message to the console.
* When the 'error' event is emitted, the corresponding listener logs the error message to the console.

You can run this JavaScript file using Node.js:

bash

Copy code

node main.js

This will continuously output messages and errors to the console based on the events being emitted.

                                                                      End