**1**. <!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>

<style>

.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" required>

<span id="firstNameError" class="error"></span>

<br>

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

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

<span id="lastNameError" class="error"></span>

<br>

<label for="age">Age:</label>

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

<span id="ageError" class="error"></span>

<br>

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

</form>

<script>

function validateForm() {

// Get form values

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

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

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

// Regular expression to check if names contain only alphabets

var nameRegex = /^[a-zA-Z]+$/;

// Validate First Name

if (!nameRegex.test(firstName)) {

document.getElementById('firstNameError').innerHTML = 'First Name should only contain alphabets.';

return false;

} else {

document.getElementById('firstNameError').innerHTML = '';

}

// Validate Last Name

if (!nameRegex.test(lastName)) {

document.getElementById('lastNameError').innerHTML = 'Last Name should only contain alphabets.';

return false;

} else {

document.getElementById('lastNameError').innerHTML = '';

}

// Validate Age

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

document.getElementById('ageError').innerHTML = 'Age should be between 18 and 50.';

return false;

} else {

document.getElementById('ageError').innerHTML = '';

}

// Form is valid

return true;

}

</script>

</body>

</html>

**2**.

<!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>

<style>

.error {

color: red;

}

</style>

</head>

<body>

<h2>Employee Registration Form</h2>

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

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

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

<span id="dobError" class="error"></span>

<br>

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

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

<span id="joiningDateError" class="error"></span>

<br>

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

<input type="number" id="salary" name="salary" required>

<span id="salaryError" class="error"></span>

<br>

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

</form>

<script>

function validateForm() {

// Get form values

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

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

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

// Validate Date of Birth

if (!isValidDate(dob)) {

document.getElementById('dobError').innerHTML = 'Invalid Date of Birth.';

return false;

} else {

document.getElementById('dobError').innerHTML = '';

}

// Validate Joining Date

if (!isValidDate(joiningDate)) {

document.getElementById('joiningDateError').innerHTML = 'Invalid Joining Date.';

return false;

} else {

document.getElementById('joiningDateError').innerHTML = '';

}

// Validate Salary

if (isNaN(salary) || salary <= 0) {

document.getElementById('salaryError').innerHTML = 'Invalid Salary.';

return false;

} else {

document.getElementById('salaryError').innerHTML = '';

}

// Form is valid

return true;

}

function isValidDate(dateString) {

// Check if the date string is a valid date

var regex = /^\d{4}-\d{2}-\d{2}$/;

return regex.test(dateString);

}

</script>

</body>

</html>

**3**.

<!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>

.error {

color: red;

}

</style>

</head>

<body>

<h2>Login Form</h2>

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

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

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

<span id="emailError" class="error"></span>

<br>

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

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

<br>

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

</form>

<script>

function validateForm() {

// Get form values

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

// Regular expression for email validation

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

// Validate Email

if (!emailRegex.test(email)) {

document.getElementById('emailError').innerHTML = 'Invalid email address.';

return false;

} else {

document.getElementById('emailError').innerHTML = '';

}

// Form is valid

return true;

}

</script>

</body>

</html>

**4.**

<!DOCTYPE html>

<html ng-app="myApp">

<head>

<title>AngularJS ng-click Example</title>

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

</head>

<body>

<div ng-controller="myController">

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

</div>

<script>

// Define the AngularJS module

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

// Define the controller

myApp.controller('myController', function ($scope) {

// Function to display an alert message

$scope.displayAlert = function () {

alert('Button Clicked!'); // You can replace this with any action you want

};

});

</script>

</body>

</html>

**5.**

<!DOCTYPE html>

<html ng-app="myApp">

<head>

<title>AngularJS Addition Example</title>

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

</head>

<body>

<div ng-controller="myController" ng-init="initialize()">

<h2>AngularJS Addition Example</h2>

<label for="num1">Number 1:</label>

<input type="number" id="num1" ng-model="number1" required>

<label for="num2">Number 2:</label>

<input type="number" id="num2" ng-model="number2" required>

<button ng-click="addNumbers()" ng-disabled="disableButton">Add</button>

<p ng-show="showResult">Result: <span ng-bind="result"></span></p>

</div>

<script>

// Define the AngularJS module

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

// Define the controller

myApp.controller('myController', function ($scope) {

// Initialize scope variables

$scope.number1 = 0;

$scope.number2 = 0;

$scope.result = 0;

$scope.showResult = false;

$scope.disableButton = true;

// Function to initialize scope variables

$scope.initialize = function () {

$scope.number1 = 0;

$scope.number2 = 0;

$scope.result = 0;

$scope.showResult = false;

$scope.disableButton = true;

};

// Function to add two numbers

$scope.addNumbers = function () {

$scope.result = $scope.number1 + $scope.number2;

$scope.showResult = true;

};

// Watch for changes in number1 and number2 to enable/disable the button

$scope.$watchGroup(['number1', 'number2'], function () {

$scope.disableButton = isNaN($scope.number1) || isNaN($scope.number2);

});

});

</script>

</body>

</html>

**6.**

<!DOCTYPE html>

<html ng-app="myApp">

<head>

<title>AngularJS Student Details Table</title>

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

</head>

<body>

<div ng-controller="myController" ng-init="initializeData()">

<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>

</div>

<script>

// Define the AngularJS module

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

// Define the controller

myApp.controller('myController', function ($scope) {

// Initialize scope variables with student data

$scope.students = [];

// Function to initialize student data

$scope.initializeData = function () {

for (var i = 1; i <= 10; i++) {

$scope.students.push({

id: i,

name: 'Student ' + i,

age: Math.floor(Math.random() \* 10) + 18, // Random age between 18 and 27

grade: 'Grade ' + Math.floor(Math.random() \* 5) + 1 // Random grade between 1 and 5

});

}

};

});

</script>

</body>

</html>

**7.**

<!DOCTYPE html>

<html ng-app="myApp">

<head>

<title>MSC(CS) Semester 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.min.js"></script>

</head>

<body>

<div ng-app="myApp">

<h2>MSC(CS) Semester II Syllabus</h2>

<div ng-view></div>

</div>

<script>

// Define the AngularJS module and configure routing

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

myApp.config(function ($routeProvider) {

$routeProvider

.when('/', {

templateUrl: 'home.html'

})

.when('/subject/:subjectId', {

templateUrl: 'subject.html',

controller: 'SubjectController'

})

.otherwise({ redirectTo: '/' });

});

// Define the controller for the home page

myApp.controller('HomeController', function ($scope) {

// Define the list of subjects

$scope.subjects = [

{ id: 1, name: 'Subject 1' },

{ id: 2, name: 'Subject 2' },

{ id: 3, name: 'Subject 3' },

// Add more subjects as needed

];

});

// Define the controller for individual subjects

myApp.controller('SubjectController', function ($scope, $routeParams) {

// Retrieve subject details based on subjectId from $routeParams

$scope.subjectId = $routeParams.subjectId;

// Replace the following with actual syllabus content for each subject

$scope.syllabusContent = 'Syllabus content for Subject ' + $scope.subjectId;

});

</script>

</body>

</html>

**8.**

<!DOCTYPE html>

<html ng-app="myApp">

<head>

<title>AngularJS Form Validation SPA</title>

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

</head>

<body>

<div ng-controller="myController">

<h2>User Details</h2>

<form name="userForm" novalidate>

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

<input type="text" id="name" name="name" ng-model="user.name" ng-pattern="/^[a-zA-Z ]+$/" required>

<span ng-show="userForm.name.$error.pattern">Name should contain only characters.</span>

<br>

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

<input type="text" id="mobile" name="mobile" ng-model="user.mobile" ng-pattern="/^\d{10}$/" required>

<span ng-show="userForm.mobile.$error.pattern">Mobile number should contain only 10 digits.</span>

<br>

<label for="pincode">Pincode:</label>

<input type="text" id="pincode" name="pincode" ng-model="user.pincode" ng-pattern="/^\d{6}$/" required>

<span ng-show="userForm.pincode.$error.pattern">Pincode should contain only 6 digits.</span>

<br>

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

<input type="email" id="email" name="email" ng-model="user.email" ng-pattern="/^[^\s@]+@[^\s@]+\.[^\s@]+$/" required>

<span ng-show="userForm.email.$error.pattern">Invalid email address.</span>

<br>

<button ng-click="submitForm()" ng-disabled="userForm.$invalid">Submit</button>

</form>

</div>

<script>

// Define the AngularJS module

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

// Define the controller

myApp.controller('myController', function ($scope) {

// Initialize user object

$scope.user = {};

// Function to submit the form

$scope.submitForm = function () {

// Handle form submission logic here

console.log('Form submitted with user details:', $scope.user);

};

});

</script>

</body>

</html>

**9**.

<!DOCTYPE html>

<html ng-app="loginApp">

<head>

<title>AngularJS Login SPA</title>

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

</head>

<body>

<div ng-controller="LoginController">

<h2>Login</h2>

<form name="loginForm" novalidate>

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

<input type="email" id="email" name="email" ng-model="user.email" required>

<br>

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

<input type="password" id="password" name="password" ng-model="user.password" required>

<br>

<button ng-click="login()" ng-disabled="loginForm.$invalid">Login</button>

<p ng-show="loginFailed" style="color: red;">Invalid email or password. Please try again.</p>

</form>

</div>

<script>

// Define the AngularJS module

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

// Define the controller

loginApp.controller('LoginController', function ($scope) {

// Initialize user object

$scope.user = {};

// Simulated user credentials for demonstration purposes

var validUser = {

email: 'user@example.com',

password: 'password123'

};

// Variable to track login failure

$scope.loginFailed = false;

// Function to perform login

$scope.login = function () {

// Simulate server-side authentication logic

if ($scope.user.email === validUser.email && $scope.user.password === validUser.password) {

// Successful login

alert('Login successful!'); // Replace with your actual login logic

$scope.loginFailed = false;

} else {

// Failed login

$scope.loginFailed = true;

}

};

});

</script>

</body>

</html>

**10**.

<!DOCTYPE html>

<html ng-app="studentRegistrationApp">

<head>

<title>Student Registration Form</title>

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

</head>

<body>

<div ng-controller="StudentRegistrationController">

<h2>Student Registration Form</h2>

<form name="registrationForm" novalidate>

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

<input type="text" id="firstName" name="firstName" ng-model="student.firstName" ng-pattern="/^[a-zA-Z]+$/" required>

<span ng-show="registrationForm.firstName.$error.pattern">First Name should only contain alphabets.</span>

<br>

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

<input type="text" id="lastName" name="lastName" ng-model="student.lastName" ng-pattern="/^[a-zA-Z]+$/" required>

<span ng-show="registrationForm.lastName.$error.pattern">Last Name should only contain alphabets.</span>

<br>

<label for="age">Age:</label>

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

<span ng-show="registrationForm.age.$error.min || registrationForm.age.$error.max">Age should be between 18 and 50.</span>

<br>

<button ng-click="submitForm()" ng-disabled="registrationForm.$invalid">Submit</button>

</form>

<div ng-show="showGreetingMessage">

<p>{{ greetingMessage }}</p>

</div>

</div>

<script>

// Define the AngularJS module

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

// Define the controller

studentRegistrationApp.controller('StudentRegistrationController', function ($scope, $http) {

// Initialize student object

$scope.student = {};

// Initialize greeting variables

$scope.showGreetingMessage = false;

$scope.greetingMessage = '';

// Function to submit the form

$scope.submitForm = function () {

// Simulate server-side registration logic

// In a real-world scenario, you would perform AJAX request to the server

// For demonstration purposes, a dummy AJAX request is used

$http.get('https://jsonplaceholder.typicode.com/todos/1')

.then(function (response) {

// Dummy success response

console.log(response);

$scope.showGreetingMessage = true;

$scope.setGreetingMessage();

})

.catch(function (error) {

// Dummy error response

console.error(error);

});

};

// Function to set the greeting message based on the current time

$scope.setGreetingMessage = function () {

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

if (currentTime >= 5 && currentTime < 12) {

$scope.greetingMessage = 'Good Morning!';

} else if (currentTime >= 12 && currentTime < 18) {

$scope.greetingMessage = 'Good Afternoon!';

} else {

$scope.greetingMessage = 'Good Evening!';

}

};

});

</script>

</body>

</html>

**11.**

<!DOCTYPE html>

<html ng-app="dateTimeApp">

<head>

<title>AngularJS Date and Time Display</title>

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

</head>

<body>

<div ng-controller="DateTimeController">

<h2>Current Date and Time</h2>

<p>{{ currentDateTime | date: 'fullDate' }} {{ currentDateTime | date: 'shortTime' }}</p>

</div>

<script>

// Define the AngularJS module

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

// Define the controller

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

// Initialize current date and time

$scope.currentDateTime = new Date();

// Function to update the current date and time at regular intervals

var updateDateTime = function () {

$scope.currentDateTime = new Date();

};

// Set up the $interval service to call updateDateTime every second (1000 milliseconds)

var intervalPromise = $interval(updateDateTime, 1000);

// Cancel the interval when the controller is destroyed (to avoid memory leaks)

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

$interval.cancel(intervalPromise);

});

});

</script>

</body>

</html>

**12.**

<!DOCTYPE html>

<html ng-app="validationApp">

<head>

<title>AngularJS Username and Password Validation</title>

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

</head>

<body>

<div ng-controller="ValidationController">

<h2>Username and Password Validation</h2>

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

<input type="text" id="username" name="username" ng-model="user.username" required>

<br>

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

<input type="password" id="password" name="password" ng-model="user.password" required>

<br>

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

</div>

<script>

// Define the AngularJS module

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

// Define the controller

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

// Initialize user object

$scope.user = {};

// Function to validate username and password

$scope.validateCredentials = function () {

// Validate username

if (!$scope.user.username) {

alert('Enter username.');

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

alert('Username is too short.');

} else {

// Validate password

if (!$scope.user.password) {

alert('Enter password.');

} else if ($scope.user.password.length < 8) {

alert('Password should be minimum 8 characters.');

} else {

alert('Valid username and password.');

}

}

};

});

</script>

</body>

</html>

**13.**

<!DOCTYPE html>

<html ng-app="locationApp">

<head>

<title>AngularJS Current Location</title>

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

</head>

<body>

<div ng-controller="LocationController">

<h2>Current Web Page Location</h2>

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

</div>

<script>

// Define the AngularJS module

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

// Define the controller

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

// Get the current location using $location service

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

});

</script>

</body>

</html>

14.

// Import the 'readline' module for reading input

const readline = require('readline');

// Create an interface to read from the standard input

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout

});

// Prompt the user for input

rl.question('Enter a string: ', (inputString) => {

// Convert the input string to uppercase

const uppercaseString = inputString.toUpperCase();

// Display the result

console.log('Uppercase Output:', uppercaseString);

// Close the readline interface

rl.close();

});

**node convertToUpper.js**

**15.**

const readline = require('readline');

const fs = require('fs');

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout

});

rl.question('Enter the name of the source file: ', (sourceFileName) => {

rl.question('Enter the name of the destination file: ', (destinationFileName) => {

// Read the contents of the source file

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

if (err) {

console.error(`Error reading ${sourceFileName}: ${err.message}`);

rl.close();

} else {

// Append the contents to the destination file

fs.appendFile(destinationFileName, data, 'utf8', (err) => {

if (err) {

console.error(`Error appending to ${destinationFileName}: ${err.message}`);

} else {

console.log(`Contents of ${sourceFileName} appended to ${destinationFileName}.`);

}

rl.close();

});

}

});

});

});

rl.on('close', () => {

process.exit(0);

});

**node appendFiles.js**

**16.**

const http = require('http');

const fs = require('fs');

const path = require('path');

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

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

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

if (err) {

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

// File not found (404 error)

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

res.end('404 Not Found');

} else {

// Other error

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

res.end('Internal Server Error');

}

} else {

// File found, send 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}`);

});

**node fileServer.js**

**17.**

const http = require('http');

const fs = require('fs');

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

if (req.method === 'GET') {

// Serve the HTML form for GET requests

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

const formHtml = `

<!DOCTYPE html>

<html>

<head>

<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="fileUpload" required>

<br>

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

</form>

</body>

</html>

`;

res.end(formHtml);

} else if (req.method === 'POST' && req.url === '/upload') {

// Handle file upload for POST requests

const form = new formidable.IncomingForm();

form.parse(req, (err, fields, files) => {

if (err) {

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

res.end('Internal Server Error');

} else {

const oldPath = files.fileUpload.path;

const newPath = \_\_dirname + '/uploads/' + files.fileUpload.name;

fs.rename(oldPath, newPath, (err) => {

if (err) {

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

res.end('Internal Server Error');

} else {

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

res.end('File uploaded successfully!');

}

});

}

});

} else {

// Invalid request method or URL

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

res.end('404 Not Found');

}

});

const PORT = 3000;

server.listen(PORT, () => {

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

});

18.

**npm install mysql**

const mysql = require('mysql');

// Create a connection to the MySQL server

const connection = mysql.createConnection({

host: 'localhost',

user: 'your\_username',

password: 'your\_password',

});

// Connect to MySQL

connection.connect((err) => {

if (err) {

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

return;

}

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

// Create a new database

const dbName = 'test\_db';

connection.query(`CREATE DATABASE IF NOT EXISTS ${dbName}`, (err) => {

if (err) {

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

connection.end();

return;

}

console.log(`Database "${dbName}" created successfully`);

// Use the new database

connection.query(`USE ${dbName}`, (err) => {

if (err) {

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

connection.end();

return;

}

// Create a new table

const tableName = 'users';

const createTableQuery = `

CREATE TABLE IF NOT EXISTS ${tableName} (

id INT AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(255) NOT NULL,

email VARCHAR(255) NOT NULL

)

`;

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

if (err) {

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

} else {

console.log(`Table "${tableName}" created successfully`);

}

// Close the MySQL connection

connection.end();

});

});

});

});

**node createDatabaseAndTable.js**

**19.** const mysql = require('mysql');

// Create a connection to the MySQL server

const connection = mysql.createConnection({

host: 'localhost',

user: 'your\_username',

password: 'your\_password',

database: 'your\_database',

});

// Connect to MySQL

connection.connect((err) => {

if (err) {

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

return;

}

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

// SELECT query to retrieve all records from the "customers" table

const selectQuery = 'SELECT \* FROM customers';

// Execute the SELECT query

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

if (err) {

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

} else {

// Display the result object on the console

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

}

// Close the MySQL connection

connection.end();

});

});

**node selectFromCustomers.js**

**20.**

const mysql = require('mysql');

// Create a connection to the MySQL server

const connection = mysql.createConnection({

host: 'localhost',

user: 'your\_username',

password: 'your\_password',

database: 'your\_database',

});

// Connect to MySQL

connection.connect((err) => {

if (err) {

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

return;

}

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

// Define multiple records to be inserted into the "student" table

const students = [

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

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

{ name: 'Bob Johnson', age: 21, grade: 'C' },

// Add more records as needed

];

// INSERT query to insert multiple records into the "student" table

const insertQuery = 'INSERT INTO student (name, age, grade) VALUES ?';

// Execute the INSERT query with the array of records

connection.query(insertQuery, [students.map(student => [student.name, student.age, student.grade])], (err, results) => {

if (err) {

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

} else {

// Display the result object on the console

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

}

// Close the MySQL connection

connection.end();

});

});

**node insertIntoStudent.js**

**21.**

const mysql = require('mysql');

// Create a connection to the MySQL server

const connection = mysql.createConnection({

host: 'localhost',

user: 'your\_username',

password: 'your\_password',

database: 'your\_database',

});

// Connect to MySQL

connection.connect((err) => {

if (err) {

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

return;

}

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

// SELECT query to retrieve all records from the "customers" table

const selectQuery = 'SELECT \* FROM customers';

// Execute the SELECT query

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

if (err) {

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

connection.end();

return;

}

// Display the result object on the console

console.log('Result before deletion:', results);

// Specify the record to be deleted (replace with your condition)

const recordToDelete = { id: 1 };

// DELETE query to delete the specified record

const deleteQuery = 'DELETE FROM customers WHERE ?';

// Execute the DELETE query with the specified condition

connection.query(deleteQuery, recordToDelete, (err, deleteResult) => {

if (err) {

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

} else {

console.log('Record deleted successfully.');

}

// Display the result object on the console after deletion

console.log('Result after deletion:', deleteResult);

// Close the MySQL connection

connection.end();

});

});

});

**node selectAndDelete.js**

**22.**

const http = require('http');

// Create a simple HTTP server

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

// Set the content type to plain text

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

// Send the "Hello, World!" message to the client

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

});

// Listen on port 3000

const PORT = 3000;

server.listen(PORT, () => {

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

});

**node simpleServer.js**

**curl** [**http://localhost:3000**](http://localhost:3000)

**23.**

const express = require('express');

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

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

const bcrypt = require('bcrypt');

const app = express();

const PORT = 3000;

// In-memory data store for simplicity (replace with a database in a real-world scenario)

const users = [

{ id: 1, username: 'user1', password: '$2b$10$3NjVVi3w16LeQj93AxdTNOZjgsgBw3ehzFJ.yCh0DuyFfpbV.gw5y' }, // Password: password1

{ id: 2, username: 'user2', password: '$2b$10$uH3EHwKrSBR9WpokijUZgu.UHsF.XieJEEV4Cr7EC6G26/ie6lxNm' }, // Password: password2

];

// Middleware

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

app.use(

session({

secret: 'your\_secret\_key',

resave: true,

saveUninitialized: true,

})

);

// Set up a simple view engine

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

// Routes

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

res.render('index', { user: req.session.user });

});

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

res.render('login');

});

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

const { username, password } = req.body;

// Find user by username

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

// Check if the user exists and the password is correct

if (user && bcrypt.compareSync(password, user.password)) {

// Set the user in the session

req.session.user = { id: user.id, username: user.username };

res.redirect('/');

} else {

res.render('login', { error: 'Invalid username or password' });

}

});

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

// Destroy the session

req.session.destroy(() => {

res.redirect('/');

});

});

// Server listens on PORT

app.listen(PORT, () => {

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

});

**1 Install the required npm packages:**

npm install express body-parser express-session bcrypt ejs

2.**Create the views folder and create two EJS files inside it: index.ejs and login.ejs.**

* **views/index.ejs:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>User Login System</title>

</head>

<body>

<h1>Welcome <%= user ? user.username : 'Guest' %>!</h1>

<% if (user) { %>

<a href="/logout">Logout</a>

<% } else { %>

<a href="/login">Login</a>

<% } %>

</body>

</html>

**views/login.ejs**:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Login</title>

</head>

<body>

<h1>Login</h1>

<% if (error) { %>

<p style="color: red;"><%= error %></p>

<% } %>

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

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

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

<br>

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

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

<br>

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

</form>

</body>

</html>

**3. Run the server**:

node your-file-name.js

**24.**  **npm install express mongoose passport passport-local express-session bcrypt ejs**

const express = require('express');

const mongoose = require('mongoose');

const passport = require('passport');

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

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

const bcrypt = require('bcrypt');

const app = express();

const PORT = process.env.PORT || 3000;

// Connect to MongoDB (replace 'your-database-url' with your actual MongoDB connection string)

mongoose.connect('your-database-url', { useNewUrlParser: true, useUnifiedTopology: true });

// Define User schema

const User = mongoose.model('User', {

username: String,

password: String,

});

// Define Course schema

const Course = mongoose.model('Course', {

title: String,

content: String,

});

// Middleware

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

app.use(session({ secret: 'your-secret-key', resave: true, saveUninitialized: true }));

app.use(passport.initialize());

app.use(passport.session());

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

// Passport configuration

passport.use(

new LocalStrategy((username, password, done) => {

User.findOne({ username: username }, (err, user) => {

if (err) return done(err);

if (!user) return done(null, false, { message: 'Incorrect username.' });

if (bcrypt.compareSync(password, user.password)) {

return done(null, user);

} else {

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

}

});

})

);

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

done(null, user.id);

});

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

User.findById(id, (err, user) => {

done(err, user);

});

});

// Routes

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

res.render('index', { user: req.user });

});

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

res.render('login');

});

app.post('/login', passport.authenticate('local', {

successRedirect: '/',

failureRedirect: '/login',

failureFlash: true

}));

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

req.logout();

res.redirect('/');

});

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

res.render('register');

});

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

const { username, password } = req.body;

// Check if the username already exists

const existingUser = await User.findOne({ username: username });

if (existingUser) {

return res.render('register', { error: 'Username already exists.' });

}

// Hash the password

const hashedPassword = bcrypt.hashSync(password, 10);

// Create a new user

const newUser = new User({ username: username, password: hashedPassword });

await newUser.save();

res.redirect('/login');

});

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

const courses = await Course.find();

res.render('courses', { user: req.user, courses: courses });

});

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

const courseId = req.params.id;

const course = await Course.findById(courseId);

if (!course) {

return res.status(404).send('Course not found.');

}

res.render('course', { user: req.user, course: course });

});

// Run the server

app.listen(PORT, () => {

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

});

**1**.**views/index.ejs**:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>eLearning System</title>

</head>

<body>

<h1>Welcome <%= user ? user.username : 'Guest' %>!</h1>

<% if (user) { %>

<a href="/logout">Logout</a>

<% } else { %>

<a href="/login">Login</a>

<a href="/register">Register</a>

<% } %>

<h2>Courses</h2>

<ul>

<% courses.forEach(course => { %>

<li><a href="/courses/<%= course.\_id %>"><%= course.title %></a></li>

<% }); %>

</ul>

</body>

</html>

**2.views/login.ejs**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Login</title>

</head>

<body>

<h1>Login</h1>

<% if (message) { %>

<p style="color: red;"><%= message %></p>

<% } %>

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

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

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

<br>

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

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

<br>

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

</form>

</body>

</html>

**3**.**views/register.ejs**:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Register</title>

</head>

<body>

<h1>Register</h1>

<% if (error) { %>

<p style="color: red;"><%= error %></p>

<% } %>

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

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

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

<br>

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

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

<br>

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

</form>

</body>

</html>

**4**.**views/courses.ejs**:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Courses</title>

</head>

<body>

<h1>Courses</h1>

<ul>

<% courses.forEach(course => { %>

<li><a href="/courses/<%= course.\_id %>"><%= course.title %></a></li>

<% }); %>

</ul>

</body>

</html>

**5.views/course.ejs**:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title><%= course.title %></title>

</head>

<body>

<h1><%= course.title %></h1>

<p><%= course.content %></p>

</body>

</html>

**25.**

**1.Initialize your project:**

mkdir recipe-book

cd recipe-book

npm init –y

**Install the required packages:**

npm install express mongoose body-parser

const express = require('express');

const mongoose = require('mongoose');

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

const app = express();

const PORT = process.env.PORT || 3000;

// Connect to MongoDB

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

mongoose.connection.on('error', console.error.bind(console, 'MongoDB connection error:'));

// Middleware

app.use(bodyParser.json());

// Define Recipe Schema

const recipeSchema = new mongoose.Schema({

title: String,

ingredients: [String],

instructions: String,

});

const Recipe = mongoose.model('Recipe', recipeSchema);

// Routes

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

try {

const recipes = await Recipe.find();

res.json(recipes);

} catch (error) {

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

}

});

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

const { title, ingredients, instructions } = req.body;

const newRecipe = new Recipe({ title, ingredients, instructions });

try {

const savedRecipe = await newRecipe.save();

res.json(savedRecipe);

} catch (error) {

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

}

});

// Start the server

app.listen(PORT, () => {

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

});

**Run your MongoDB server**

Mongod

**Run your Node.js application:**

node app.js

**26.**

**1.Create an HTML file named** **index.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, this is a simple web page served by Node.js!</h1>

</body>

</html>

**2.Create a Node.js script named** **server.js**:

const fs = require('fs');

const http = require('http');

const path = require('path');

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

// Read the HTML file

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

fs.readFile(filePath, 'utf8', (err, content) => {

if (err) {

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

res.end('Internal Server Error');

console.error(err);

} else {

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

res.end(content);

}

});

});

const PORT = process.env.PORT || 3000;

server.listen(PORT, () => {

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

});

**Run your Node.js script**

node server.js

**27.**

**Create a new file named modules.js with the following content:**

// modules.js

const http = require('http');

// Function to create a local server

function createServer(port, callback) {

const server = http.createServer(callback);

server.listen(port, () => {

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

});

}

// Function to return today's date and time

function getDateTime() {

const now = new Date();

return now.toISOString();

}

// Exporting functions to make them available externally

module.exports = {

createServer,

getDateTime,

};

**Now, you can create another file (e.g., app.js) to use this module: Now, you can create another file (e.g., app.js) to use this module:**

// app.js

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

// Using the createServer function from the module

myModule.createServer(3000, (req, res) => {

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

res.end('Hello, this is your local server!');

});

// Using the getDateTime function from the module

const currentDateTime = myModule.getDateTime();

console.log(`Current date and time: ${currentDateTime}`);

**Run your app.js file using Node.js:**

node app.js

28.

// main.js

const EventEmitter = require('events');

// Creating an instance of EventEmitter

const myEmitter = new EventEmitter();

// Callback function for the 'eventOne' event

function eventOneHandler() {

console.log('Event One triggered!');

}

// Callback function for the 'eventTwo' event

function eventTwoHandler(data) {

console.log(`Event Two triggered with data: ${data}`);

}

// Adding event listeners

myEmitter.on('eventOne', eventOneHandler);

myEmitter.on('eventTwo', eventTwoHandler);

// Main loop listening for events

function mainLoop() {

// Simulating events being detected

setTimeout(() => {

myEmitter.emit('eventOne'); // Triggering eventOne

}, 1000);

setTimeout(() => {

myEmitter.emit('eventTwo', 'Some data'); // Triggering eventTwo with data

}, 2000);

}

// Starting the main loop

mainLoop();

**To run this script, use the following command:**

node main.js

**29**.

**Install the necessary dependencies:**

npm init -y

npm install express

// app.js

const express = require('express');

const fs = require('fs');

const path = require('path');

const app = express();

const port = 3000;

// Endpoint to trigger file download

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

// File path (replace with your actual file path)

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

const fileName = 'downloaded\_file.txt';

// Set response headers to trigger download

res.setHeader('Content-Disposition', `attachment; filename=${fileName}`);

res.setHeader('Content-Type', 'application/octet-stream');

// Create a read stream from the file

const fileStream = fs.createReadStream(filePath);

// Pipe the file stream to the response stream

fileStream.pipe(res);

});

// Start the server

app.listen(port, () => {

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

});

**Run your Express application:**

node app.js

**30**.