**IFT 544: MIDDLEWARE PROG & DATABASE SEC (2023 FALL)**

Dharun Prasanth Dhandapani

M.S Information Technology

Arizona State University

**Module 2 Assignment: Exploring JavaScript Topics with**

**EJS, Node.js, and Express**

**Prof.** **Dinesh Sthapit**

September 03, 2023

**Advance JavaScript:**

Exercise 1: Variables:

* In the route handler, extract the input values and perform addition, subtraction, multiplication, and division operations on the numbers.

A screenshot of a computer program

Description automatically generated

* Pass the calculated results to another EJS template and display them. A screenshot of a computer

  Description automatically generatedA screenshot of a computer

  Description automatically generated

Application UI: A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

**Observation and Analysis:**

**.ejs Code Analysis:**

1. **JavaScript Embedding**: The .ejs file contains JavaScript code embedded within **<% %>** tags. These tags allow dynamic generation of HTML content by evaluating JavaScript expressions and statements.
2. **Control Structures**: In the .ejs file, there are no explicit control structures such as loops or conditionals. It's a simple form rendering template. Control structures could be added using EJS tags, such as **<% if (condition) { %> ... <% } %>**.
3. **Variable Rendering**: Variables are accessed and displayed within the HTML content using **<%= variableName %>** syntax. In this case, variables **sum**, **difference**, **product**, and **quotient** are displayed within the **<p>** tags using this syntax.
4. **EJS-Specific Syntax**: The **<% %>** tags and **<%= %>** tags are EJS-specific syntax used for embedding JavaScript and rendering dynamic content, respectively.

A screenshot of a computer

Description automatically generated

**HTML Code Analysis:**

1. **Dynamically Generated Content**: When you inspect the rendered HTML in the browser or view the HTML source code, you'll see that the values of **sum**, **difference**, **product**, and **quotient** have been dynamically generated and inserted into the HTML structure.
2. **Correspondence with .ejs Code**: The dynamically generated content in the HTML matches the expressions within the **<%= %>** tags in the .ejs file. For instance, the value of **sum** is displayed in the HTML as **<p>Sum: 9</p>**.
3. **Static Content**: The static parts of the HTML, such as the form structure, remain unchanged from the .ejs file. Only the dynamic content changes based on the values calculated and passed from the server. **A screenshot of a computer

   Description automatically generated**

**Comparing .ejs Code with HTML Code:**

* The .ejs file serves as a template that contains placeholders for dynamic data. JavaScript code within **<% %>** tags is used to calculate values and control dynamic behavior.
* The HTML code is the final rendered output that incorporates the calculated values from the .ejs file.

**Benefits of Using .ejs Templates over Static HTML:**

1. **Dynamic Content**: .ejs templates allow for dynamic content generation, making it possible to display data that can change based on various factors.
2. **Code Reusability**: Variables and logic can be reused across different parts of the template, reducing redundancy and making maintenance easier.
3. **Conditional Rendering**: EJS enables conditional rendering of HTML elements, allowing you to show or hide parts of the page based on conditions.
4. **Data Passing**: EJS templates facilitate passing data from the server to the client-side template, allowing the rendering of server-side data.
5. **Simpler Maintenance**: If changes are needed in the displayed content, you only need to modify the template (.ejs) file rather than editing each HTML file separately.
6. **Consistency**: EJS templates help maintain consistent design and layout across multiple pages by using the same template for different content.

**Example Scenario:**

Let's say you're building an e-commerce website. Each product page has similar structure but different product information. Using an .ejs template, you can create a standardized layout for product pages, dynamically inserting product details, images, and descriptions. This ensures consistent design while allowing each product page to display unique information.

Remember, .ejs is just one of many template engines available. The choice depends on your project's needs and your familiarity with the specific technology.

**Enhancement:**

In this enhanced .ejs file, we've added the following:

1. Array Iteration: We've defined an array called fruits containing fruit names. Inside the <ul> element, we use the <% fruits.forEach(fruit => { %> syntax to iterate over the array and dynamically generate <li> elements for each fruit.
2. Conditional Rendering: We've defined a boolean variable called showAdditionalContent. Using an <% if (showAdditionalContent) { %> statement, we conditionally display a paragraph with additional content based on the value of the variable.

The enhancements in the .ejs file have resulted in the following changes in the rendered HTML:

* An <ul> element with four <li> elements, each containing a fruit name from the fruits array.
* A paragraph <p> element with additional content is displayed because the `showAdditionalContent` variable is set to true.

Source code:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Enhanced Example</title>

</head>

<body>

<form action="/calculate" method="POST">

<input type="number" name="num1" placeholder="Enter first number" required>

<input type="number" name="num2" placeholder="Enter second number" required>

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

</form>

<p>

<h1>Calculate Results:</h1>

<p>Sum: <%= sum %></p>

<p>Difference: <%= difference %></p>

<p>Product: <%= product %></p>

<p>Quotient: <%= quotient %></p>

</p>

<% const fruits = ['Apple', 'Banana', 'Orange', 'Mango']; %>

<ul>

<% fruits.forEach(fruit => { %>

<li><%= fruit %></li>

<% }); %>

</ul>

<% const showAdditionalContent = true; %>

<% if (showAdditionalContent) { %>

<p>This additional content is being displayed conditionally.</p>

<% } %>

</body>

</html>

**Screenshots: A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated**

**Rubrics criteria:**

Created an array of loans and add five instances of the above JavaScript objects based on the above specifications.

 Created a List of all the array elements with the grand total loan amount using console.log statements.

**Source code:**

**App.js file:**

const express = require('express');

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

const path = require('path');

const app = express();

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

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

app.set('views', path.join(\_\_dirname, 'views'));

let loans = []; // Changed to an array of loans

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

res.render('index', { loans: loans }); // Passed 'loans' array to the EJS template

});

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

const loan = {

loanamount: parseFloat(req.body.loanamount),

interest: parseFloat(req.body.interest),

loanTermYears: parseInt(req.body.loanTermYears),

loanType: req.body.loanType,

loanTotal: 0, // Added loanTotal property

};

loan.loanTotal = loan.loanamount + (loan.loanamount \* loan.interest \* loan.loanTermYears) / 100;

loans.push(loan);

let grandTotal = 0;

loans.forEach(loan => {

grandTotal += loan.loanTotal;

});

console.log('Loan added:', loan);

console.log('Current loans:', loans);

console.log('Grand Total Loan Amount:', grandTotal);

res.redirect('/');

});

app.listen(3000, () => {

console.log("Server is running on port 3000");

});

**.ejs file:**

<!DOCTYPE html>

<html>

<head>

<title>Student Information Form</title>

</head>

<body>

<!-- This is the form for the user to input their data -->

<!-- when the form is submitted, it sends a POST request to the server at the specified action URL ("/")-->

<form action="/" method="POST">

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

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

<label for="middleName">Middle Name:</label><br>

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

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

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

<label for="address">Address:</label><br>

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

<label for="phone">Phone:</label><br>

<input type="tel" id="phone" name="phone"><br>

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

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

<label for="description">Description:</label><br>

<textarea id="description" name="description"></textarea><br>

<label for="loanamount">Loan Amount:</label><br>

<input type="number" id="loanamount" name="loanamount"><br>

<label for="interest">Interest: </label><br>

<input type="number" id="interest" name="interest"><br>

<label for="loanTermYears">Loan Term Years: </label><br>

<input type="number" id="loanTermYears" name="loanTermYears"><br>

<label for="loanType">Loan Type: </label><br>

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

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

</form>

<!-- This is where we display the submitted student data -->

<!-- EJS allows us to use Javascript in our HTML. Here, we loop over the 'students' array and create a new list item for each -->

<ul>

<% loans.forEach(function(loan) { %>

<li>

<p>Loan Amount: <%= loan.loanamount %> </p>

<p>Interest: <%= loan.interest %> </p>

<p>Loan Term Years: <%= loan.loanTermYears %> </p>

<p>Loan Type: <%= loan.loanType %> </p>

<p>Loan Total: <%= loan.loanTotal.toFixed(2) %> </p> <!-- Display loanTotal -->

</li>

<% }); %>

</ul>

<!-- This is a link that allows the user to add more student data -->

<!-- when clicked, it sends a GET request to the server at the specified URL ("/") -->

<a href="/">Add New Student</a>

</body>

</html>

**Output screenshots: A screenshot of a computer

Description automatically generatedA screen shot of a computer

Description automatically generated**

**A screenshot of a computer program

Description automatically generatedA screen shot of a computer

Description automatically generatedA screenshot of a computer program

Description automatically generatedA screenshot of a computer

Description automatically generated**