Module 2: Lab 2 Basic Full Stack Labs

Aishwarya Devi Akkim

Masters in information technology, Arizona State University

IFT458&544: Middleware Prog & Database Sec

Prof: Dinesh Sthapit

September 10, 2023

**Exercise 1:**

**Code app.js:**

const express = require('express');

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

const path = require('path');

const app = express();

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

// Set EJS as templating engine

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

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

console.log('views', path.join(\_\_dirname, 'views'));

//create a route for the home page

//The GET route for the form

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

res.render('index');

});

//create route for user to enter the numbers

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

const { num1,num2} = req.body;

const sum = Number(num1) + Number(num2);

const difference = Number(num1)-Number(num2);

const product = Number(num1) \* Number(num2);

const quotient = Number(num1) / Number(num2);

res.render("result", {sum,difference,product,quotient});

});

// const book = [];

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

// res.render("books", {books});

// });

//start the server on port 4000

//Note we are advertising the service on port number 4000 and not 3000 this time

var port = 4000

app.listen(port, () =>{

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

});

**Code index.ejs:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Document</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>

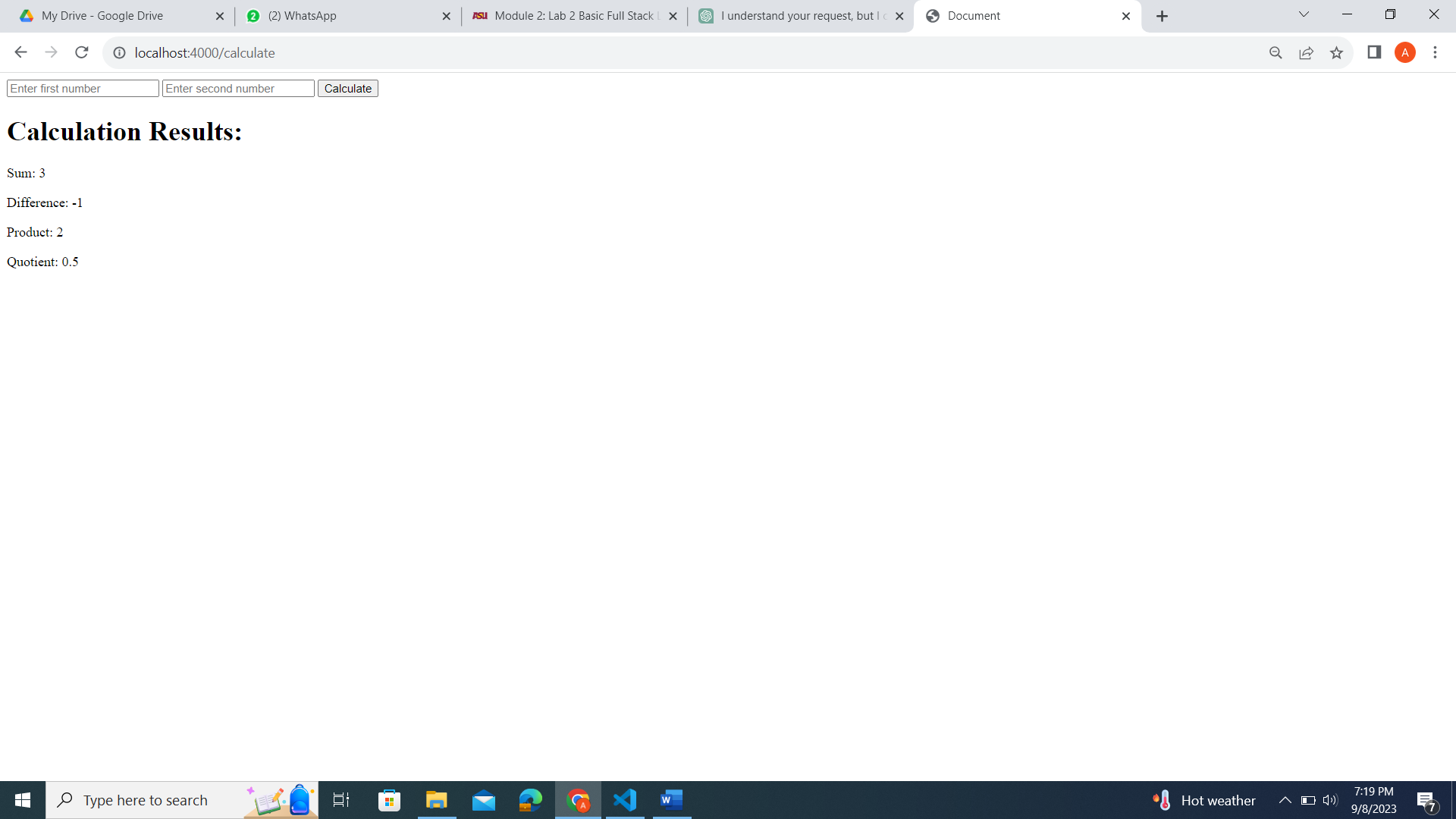
</form>

</body>

</html>

A screenshot of a computer

Description automatically generated



Differences:

Template Tags: In the .ejs code, you can see the use of <%= variable %> tags, which are used for embedding dynamic data. In the rendered HTML, these tags are replaced with actual data values. For example, <%= sum %> in .ejs is replaced with the calculated sum value in the rendered HTML.

Server-Side Rendering: .ejs code is typically used for server-side rendering. It allows you to generate HTML dynamically on the server, incorporating data from the server into the HTML before it's sent to the client. In contrast, traditional HTML is static and does not allow for dynamic content generation on the server.

Data Integration: .ejs code facilitates the integration of data from the server or other sources into the HTML template. In the example, the results of a calculation (sum, difference, product, quotient) are passed to the .ejs template and displayed in the HTML, allowing for dynamic content generation.

Benefits:

Dynamic Content: .ejs templates enable the rendering of dynamic content. You can inject data into the HTML on the server side, making it flexible for displaying different data sets without the need for multiple static HTML files.

Code Reusability: .ejs allows you to create reusable templates that can be populated with different data. This is particularly useful for generating consistent HTML structures for various pages on a website.

Separation of Concerns: .ejs promotes a separation of concerns by keeping logic (JavaScript) and presentation (HTML) separate. This separation enhances code maintainability and readability.

Functionality:

The .ejs code provides functionality for a calculation form that takes two numbers as input and displays their sum, difference, product, and quotient upon submission. The dynamic nature of .ejs allows these results to change based on user input or other dynamic data sources.

Additional Notes and Suggestions:

Error Handling: Ensure that proper error handling is in place when using .ejs templates. If data is not provided or an error occurs during rendering, handle these cases gracefully to prevent crashes or incorrect displays.

Security: Be cautious when embedding user-generated data into .ejs templates to prevent security vulnerabilities like cross-site scripting (XSS). Sanitize user inputs or use secure data binding techniques to mitigate such risks.

Performance: Consider caching .ejs templates when necessary to improve server performance, especially for frequently accessed templates.

Documentation: Maintain clear documentation for your .ejs templates, indicating how data should be passed to them and how they are intended to be used.

In summary, .ejs is a powerful tool for generating dynamic content on the server side, providing benefits such as dynamic content generation, code reusability, and separation of concerns. However, it's essential to use it carefully, ensuring proper error handling and security practices to create secure and efficient web applications.

**Exercise 2:**

**Code app.js:**

const express = require('express');

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

const path = require('path');

const app = express();

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

// Set EJS as templating engine

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

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

console.log('views', path.join(\_\_dirname, 'views'));

//create a route for the home page

//The GET route for the form

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

// res.render('index');

// });

//create route for user to enter the numbers

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

const { num1,num2} = req.body;

const sum = Number(num1) + Number(num2);

const difference = Number(num1)-Number(num2);

const product = Number(num1) \* Number(num2);

const quotient = Number(num1) / Number(num2);

res.render("result", {sum,difference,product,quotient});

});

const books = [];

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

res.render("books", {books});

});

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

const {title, author, publicationYear} = req.body;

books.push({title, author, publicationYear});

res.redirect("/");

});

//start the server on port 4000

//Note we are advertising the service on port number 4000 and not 3000 this time

var port = 4000

app.listen(port, () =>{

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

});

**Code book.ejs:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Document</title>

</head>

<body>

<ul>

<% books.forEach(book => { %>

<li><%= book.title%> by <%= book.author%> (Published: <%= book.publicationYear %>)</li>

<%}) ;%>

</ul>

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

<input type="text" name = "title" placeholder="Title" required>

<input type="text" name = "author" placeholder="Author" required>

<input type = "number" name = "publicationYear" placeholder="Publication Year" required>

<button type = "submit">Add Book</button>

</form>

</body>

</html>

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Exercise 3:**

**Code app.js:**

// Import the required modules

const express = require('express');

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

const path = require('path');

// Create an instance of express

const app = express();

// We use the 'body-parser' middleware to parse the incoming request bodies

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

// Set the view engine to ejs

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

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

console.log('views', path.join(\_\_dirname, 'views'));

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

res.render("user");

});

function User(name, age, email) {

this.name = name;

this.age = age;

this.email = email;

}

// route handler for the form

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

const { name, age, email } = req.body;

const user = new User(name, age, email);

const { name: userName, age: userAge, email: userEmail } = user;

res.render("userInfo", { userName, userAge, userEmail });

});

// Start the server on port 4000,

// Note we are advertising the service on port number 4000 and not 3000 this time

var port = 4000

// NOTE

// the quotes are replaced by back ticks ` next to key caps 1

app.listen(port, () => {

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

});

**Code userInfo.ejs:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Document</title>

</head>

<body>

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

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

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

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

<button type = "submit">Add User</button>

</form>

<p>

<h1>User Information:</h1>

<p>Name: <%= userName %></p>

<p>Age: <%= userAge %></p>

<p>Email: <%= userEmail %></p>

</p>

</body>

</html>

**Code user.ejs:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Document</title>

</head>

<body>

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

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

<input type="number" name = "Age" placeholder="Age" required>

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

<button type = "submit">Add User</button>

</form>

</body>

</html>

A screenshot of a computer

Description automatically generated

**Exercise 4:**

**Code app.js:**

// Import the required modules

const express = require('express');

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

const path = require('path');

// Create an instance of express

const app = express();

// We use the 'body-parser' middleware to parse the incoming request bodies

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

// Set the view engine to ejs

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

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

console.log('views', path.join(\_\_dirname, 'views'));

const fruits = ["Apple","Orange","Banana"];

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

res.render("fruits", {fruits});

});

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

const {fruit} = req.body;

fruits.push(fruit);

res.redirect("/");

});

// Start the server on port 4000,

// Note we are advertising the service on port number 4000 and not 3000 this time

var port = 4000

// NOTE

// the quotes are replaced by back ticks ` next to key caps 1

app.listen(port, () => {

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

});

**Code fruits.js:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Document</title>

</head>

<body>

<ul>

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

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

<% }); %>

</ul>

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

<input type="text" name="fruit" placeholder="Enter a fruit" required>

<button type="submit">Add Fruit</button>

</form>

</body>

</html>

A screenshot of a computer

Description automatically generated

**Exercise 5: Promises**

A screenshot of a computer

Description automatically generated

**Exercise 6: Async/Await**

**Code app.js:**

//Import the required modules

const express = require('express');

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

const path = require('path');

const axios = require('axios');

// Create an instance of express

const app = express();

// We use the 'body-parser' middleware to parse the incoming request bodies

app.use(bodyParser.json());

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

// Set the view engine to ejs

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

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

// console.log('views', path.join(\_\_dirname, 'views'));

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

res.render("httpRequest");

});

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

setTimeout(() => {

res.json({ message: "Asynchronous operation completed!" });

}, 2000);

});

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

const { url } = req.body;

console.log(`url - ${url}`)

try {

const response = await axios.get(url);

console.log(`response is ${response}`)

res.json(response.data);

}catch(error) {

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

}

})

// Start the server on port 4000,,

// Note we are advertising the service on port number 4000 and not 3000 this time

var port = 4000

//NOTE

// the quotes are replaced by back ticks ` next to keys caps 1

app.listen(port, () => {

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

});

**Code httpRequest.ejs:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Document</title>

</head>

<body>

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

<input type="text" name="url" placeholder="Enter a URL" required>

<button type="submit">Make Request</button>

</form>

<p id="response"></p>

<script>

const form = document.querySelector("form");

const response = document.getElementById("response");

function updateContent(data){

document.getElementById("response").textContent = JSON.stringify(data);

}

form.addEventListener("submit", async (event) => {

event.preventDefault();

const formData = new FormData(form);

const url = formData.get("url");

const response = await fetch("/makeRequest", {

method: "POST",

body: JSON.stringify({ url }),

headers: { "Content-Type": "application/json" },

});

const data = await response.json();

// response.textContent = JSON.stringify(data);

updateContent(data);

});

</script>

</body>

</html>

A screenshot of a computer

Description automatically generated

**Github Reference**: https://github.com/aakkim/Middleware