

**U18ITE0226 – Full Stack Software Development**

**--------------------------------------------------**

**2024-2025**

**LAB MANUAL**

**DEPARTMENT OF CSE,IT & AI&DS**

**INDEX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Expt. No** | **Date** | **Title of the Experiment** | **Marks** | **Signature** |
|  | 06/01/25 | Create your own Node.js module and import and use modules in your web server application. |  |  |
|  | 20/01/25 | Develop asynchronous functions with callbacks, error handling, and control flow using callbacks. |  |  |
|  | 03/02/25 | Demonstrate JSON file data read and write using Node.js. |  |  |
|  | 10/02/25 | Create a RESTful API to serve JSON data. |  |  |
|  | 24/02/25 | Demonstrate RESTful endpoints using Express and HTTP methods to handle GET, POST, PUT, and DELETE requests. |  |  |
|  | 03/03/25 | Integrate a template engine (e.g., EJS or Pug) with Express and Render dynamic HTML views using templates. |  |  |
|  | 17/03/25 | Implement user authentication in your Express application. |  |  |
|  | 24/03/25 | Explore and integrate third-party Node.js extensions into your Express app and showcase the benefits of using extensions for specific features. |  |  |
|  | 31/03/25 | Create a multi-page web application with authentication, routing, and RESTful APIs. |  |  |
|  | 06/04/25 | Create a web Application with MongoDB and Node.js. |  |  |

**Average marks Faculty Signature**

**Experiment Number : 1 Date:**

## **Create a Node.js Module and Import & Use Modules in a Web Server Application**

### **PRE LAB EXERCISE**

**QUESTIONS:**

1. What are Node.js modules?
2. How do you create and export a custom module in Node.js?
3. What is the difference between built-in, third-party, and user-defined modules?
4. How do you import a module using require()?
5. What is the purpose of the module.exports object?

### **IN LAB EXERCISE**

#### ****OBJECTIVE:****

* To understand the creation of Node.js modules.
* To import and use custom and built-in modules in a web server application.

#### ****RESOURCES:****

* Node.js installed
* Any text editor (VS Code recommended)
* Basic knowledge of JavaScript

#### ****PROGRAM LOGIC & IMPLEMENTATION:****

1. Create a custom module (myModule.js).
2. Export functions from the module.
3. Import and use the module in a Node.js web server (server.js).

### **PROGRAM: Create and Use Node.js Modules**

#### ****Step 1: Create a Custom Module (****myModule.js****)****

// myModule.js

exports.greet = function(name) {

return `Hello, ${name}! Welcome to Node.js`;

};

exports.add = function(a, b) {

return a + b;

};

#### ****Step 2: Create a Web Server and Import Module (****server.js****)****

const http = require('http');

const myModule = require('./myModule'); // Import custom module

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

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

res.write(myModule.greet('Student')); // Using the greet function

res.write('\nSum: ' + myModule.add(10, 20)); // Using the add function

res.end();

});

server.listen(3000, () => {

console.log('Server running at http://localhost:3000');

});

### **POST LAB EXERCISE**

1. What is the purpose of module.exports in Node.js?
2. Modify the myModule.js file to include a function that multiplies two numbers.
3. How does Node.js differ from traditional JavaScript execution in a browser?
4. What happens if you try to import a module that does not exist?

**ASSESSMENT PATTERN.**

|  |  |  |
| --- | --- | --- |
| **Description** | **Max Marks** | **Marks Awarded** |
| Pre Lab Exercise | **5** |  |
| In Lab Exercise | **10** |  |
| Post Lab Exercise | **5** |  |
| Viva | **10** |  |
| **Total** | **30** |  |
| **Faculty Signature** | |  |

**Experiment Number : 2 Date:**

## **Develop Asynchronous Functions with Callbacks and Error Handling**

### **PRE LAB EXERCISE**

**QUESTIONS:**

1. What is the difference between synchronous and asynchronous programming?
2. What are callbacks in Node.js?
3. How do you handle errors in asynchronous code?
4. What is the event loop in Node.js?
5. Why is it important to use non-blocking code in Node.js?

### **IN LAB EXERCISE**

#### ****OBJECTIVE:****

* To implement asynchronous functions using callbacks.
* To handle errors in asynchronous functions.

#### ****RESOURCES:****

* Node.js installed
* Any text editor

#### ****PROGRAM LOGIC & IMPLEMENTATION:****

1. Create an asynchronous function to read a file.
2. Implement error handling using a callback.

### **PROGRAM: Asynchronous File Read with Callbacks**

const fs = require('fs');

function readFileAsync(fileName, callback) {

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

if (err) {

callback(err, null);

} else {

callback(null, data);

}

});

}

// Using the function

readFileAsync('example.txt', (err, data) => {

if (err) {

console.log('Error reading file:', err.message);

} else {

console.log('File content:', data);

}

});

### **POST LAB EXERCISE**

1. What are the advantages of asynchronous programming in Node.js?
2. Rewrite the program using Promises instead of callbacks.
3. What happens if fs.readFile() is called on a file that does not exist?
4. Explain the difference between callbacks and promises.

**ASSESSMENT PATTERN.**

|  |  |  |
| --- | --- | --- |
| **Description** | **Max Marks** | **Marks Awarded** |
| Pre Lab Exercise | **5** |  |
| In Lab Exercise | **10** |  |
| Post Lab Exercise | **5** |  |
| Viva | **10** |  |
| **Total** | **30** |  |
| **Faculty Signature** | |  |

**Experiment Number : 3 Date:**

## **Demonstrate JSON File Data Read and Write Using Node.js**

### **PRE LAB EXERCISE**

**QUESTIONS:**

1. What is JSON?
2. How do you read a JSON file in Node.js?
3. How do you write data to a JSON file in Node.js?
4. What is the difference between synchronous and asynchronous file handling?

### **IN LAB EXERCISE**

#### ****OBJECTIVE:****

* To read and write JSON data using Node.js file system module.

#### ****RESOURCES:****

* Node.js installed
* JSON file

#### ****PROGRAM LOGIC & IMPLEMENTATION:****

1. Create a JSON file (data.json).
2. Read the JSON file and display content.
3. Modify the JSON content and write back to the file.

### **PROGRAM: JSON Read and Write in Node.js**

#### ****Step 1: Create a JSON File (****data.json****)****

json

CopyEdit

{

"name": "Alice",

"age": 25,

"city": "New York"

}

#### ****Step 2: Read and Modify JSON (****jsonReadWrite.js****)****

const fs = require('fs');

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

if (err) {

console.log('Error reading file:', err);

return;

}

let jsonData = JSON.parse(data);

console.log('Original Data:', jsonData);

// Modify JSON data

jsonData.age = 30;

fs.writeFile('data.json', JSON.stringify(jsonData, null, 2), (err) => {

if (err) {

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

return;

}

console.log('Data updated successfully');

});

});

### **POST LAB EXERCISE**

1. What method is used to parse JSON data in JavaScript?
2. Modify the program to add a new key-value pair to the JSON object.
3. How would you handle an error if the JSON file is incorrectly formatted?
4. Explain the difference between JSON.stringify() and JSON.parse().

**ASSESSMENT PATTERN.**

|  |  |  |
| --- | --- | --- |
| **Description** | **Max Marks** | **Marks Awarded** |
| Pre Lab Exercise | **5** |  |
| In Lab Exercise | **10** |  |
| Post Lab Exercise | **5** |  |
| Viva | **10** |  |
| **Total** | **30** |  |
| **Faculty Signature** | |  |

**Experiment Number : 4 Date:**

## **Create a RESTful API to Serve JSON Data**

### **PRE LAB EXERCISE**

**QUESTIONS:**

1. What is a RESTful API?
2. What are the different HTTP methods used in RESTful APIs?
3. What is the difference between GET, POST, PUT, and DELETE methods?
4. How do you create an API in Node.js using Express?

### **IN LAB EXERCISE**

#### ****OBJECTIVE:****

* To create a RESTful API in Node.js that serves JSON data.
* To use Express.js for handling API requests.

#### ****RESOURCES:****

* Node.js installed
* Express.js
* Postman for API testing

#### ****PROGRAM LOGIC & IMPLEMENTATION:****

1. Create a Node.js application.
2. Install Express.js.
3. Define routes for GET, POST, PUT, and DELETE.
4. Serve JSON data as API responses.

### **PROGRAM: Create a RESTful API (**server.js**)**

const express = require('express');

const app = express();

app.use(express.json()); // Middleware to parse JSON

let users = [

{ id: 1, name: "Alice", age: 25 },

{ id: 2, name: "Bob", age: 30 }

];

// GET request - Fetch users

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

res.json(users);

});

// POST request - Add a new user

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

const newUser = req.body;

users.push(newUser);

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

});

// PUT request - Update a user

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

const userId = parseInt(req.params.id);

let user = users.find(u => u.id === userId);

if (!user) return res.status(404).send('User not found');

user.name = req.body.name;

user.age = req.body.age;

res.json(user);

});

// DELETE request - Remove a user

app.delete('/users/:id', (req, res) => {

users = users.filter(u => u.id !== parseInt(req.params.id));

res.json({ message: 'User deleted' });

});

app.listen(3000, () => console.log('Server running on port 3000'));

### **POST LAB EXERCISE**

1. How does express.json() help in handling API requests?
2. What is the difference between req.params and req.body?
3. Test the API using **Postman** or **cURL** and observe the response.
4. Modify the API to return users based on age.

**ASSESSMENT PATTERN.**

|  |  |  |
| --- | --- | --- |
| **Description** | **Max Marks** | **Marks Awarded** |
| Pre Lab Exercise | **5** |  |
| In Lab Exercise | **10** |  |
| Post Lab Exercise | **5** |  |
| Viva | **10** |  |
| **Total** | **30** |  |
| **Faculty Signature** | |  |

**Experiment Number : 5 Date:**

# **Demonstrate RESTful Endpoints using Express and HTTP Methods**

## **PRE LAB EXERCISE**

### **Objective:**

* Understand RESTful APIs and their importance in web development.
* Learn the HTTP methods (GET, POST, PUT, DELETE) and their purpose.
* Set up a basic Express.js server for handling API requests.

### **QUESTIONS:**

1. What is a RESTful API, and why is it important?
2. List the common HTTP methods used in RESTful APIs.
3. What is the difference between GET, POST, PUT, and DELETE methods?
4. How does Express.js help in building RESTful APIs?
5. What is the significance of middleware in handling API requests?

## **IN LAB EXERCISE**

### **Objective:**

* Implement a RESTful API using Express.js.
* Handle different HTTP methods (GET, POST, PUT, DELETE).
* Test API endpoints using **Postman** or **cURL**.

### **Resources Required:**

* Node.js installed
* Express.js package (npm install express)
* Postman or cURL for API testing

### **Step 1: Install Required Packages**

Open your terminal and run:

npm init -y

npm install express

### **Step 2: Implement RESTful API (**server.js**)**

js

CopyEdit

const express = require('express');

const app = express();

app.use(express.json()); // Middleware to parse JSON

// Sample data

let products = [

{ id: 1, name: "Laptop", price: 1000 },

{ id: 2, name: "Phone", price: 500 }

];

// GET request - Fetch all products

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

res.json(products);

});

// GET request - Fetch product by ID

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

const product = products.find(p => p.id == req.params.id);

if (!product) return res.status(404).send('Product not found');

res.json(product);

});

// POST request - Add a new product

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

const newProduct = { id: products.length + 1, name: req.body.name, price: req.body.price };

products.push(newProduct);

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

});

// PUT request - Update an existing product

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

const product = products.find(p => p.id == req.params.id);

if (!product) return res.status(404).send('Product not found');

product.name = req.body.name;

product.price = req.body.price;

res.json(product);

});

// DELETE request - Remove a product

app.delete('/products/:id', (req, res) => {

products = products.filter(p => p.id != req.params.id);

res.json({ message: 'Product deleted' });

});

// Start server

app.listen(3000, () => console.log('Server running on port 3000'));

### **Step 3: Test the API using Postman or cURL**

#### ****GET All Products:****

* **URL:** http://localhost:3000/products
* **Method:** GET
* **Expected Response:**

[

{ "id": 1, "name": "Laptop", "price": 1000 },

{ "id": 2, "name": "Phone", "price": 500 }

]

#### ****GET Product by ID:****

* **URL:** http://localhost:3000/products/1
* **Method:** GET

{ "id": 1, "name": "Laptop", "price": 1000 }

#### ****POST New Product:****

* **URL:** http://localhost:3000/products
* **Method:** POST
* **Body (JSON):**

{ "name": "Tablet", "price": 600 }

* **Expected Response:**

{ "id": 3, "name": "Tablet", "price": 600 }

#### ****PUT Update Product:****

* **URL:** http://localhost:3000/products/2
* **Method:** PUT
* **Body (JSON):**

{ "name": "Smartphone", "price": 550 }

* **Expected Response:**

{ "id": 2, "name": "Smartphone", "price": 550 }

#### ****DELETE a Product:****

* **URL:** http://localhost:3000/products/1
* **Method:** DELETE
* **Expected Response:**

{ "message": "Product deleted" }

## **POST LAB EXERCISE**

### **Objective:**

* Verify and analyze API responses.
* Enhance the API by adding error handling and validation.
* Implement authentication and authorization.

### **QUESTIONS:**

1. What is the purpose of using req.params in Express?
2. How can we validate incoming request data in a RESTful API?
3. How can authentication be integrated into a RESTful API?
4. How would you modify this API to store data in a **MongoDB** database?
5. What are the benefits of using express.json() middleware in handling API requests?

### **Enhancements (For Further Learning)**

* Implement **input validation** using express-validator.
* Connect the API with **MongoDB** using mongoose.
* Add **authentication** using JWT.

**ASSESSMENT PATTERN.**

|  |  |  |
| --- | --- | --- |
| **Description** | **Max Marks** | **Marks Awarded** |
| Pre Lab Exercise | **5** |  |
| In Lab Exercise | **10** |  |
| Post Lab Exercise | **5** |  |
| Viva | **10** |  |
| **Total** | **30** |  |
| **Faculty Signature** | |  |

**Experiment Number : 6 Date:**

# **Integrate a Template Engine (EJS/Pug) with Express and Render Dynamic HTML Views**

## **PRE LAB EXERCISE**

### **Objective:**

* Understand the role of **template engines** in web applications.
* Learn how to integrate **EJS (Embedded JavaScript) or Pug** with Express.js to render **dynamic web pages**.
* Implement a simple **Express.js server** that dynamically generates HTML pages using data.

### **QUESTIONS:**

1. What is a **template engine**, and why is it used in web applications?
2. Compare **EJS** and **Pug** in terms of syntax and usability.
3. How does a **template engine** improve code reusability in web applications?
4. What is the difference between **server-side rendering (SSR)** and **client-side rendering (CSR)**?
5. How do you pass dynamic data from an Express route to a template engine?

## **IN LAB EXERCISE**

### **Objective:**

* Set up an **Express.js server**.
* Use **EJS or Pug** to generate **dynamic web pages**.
* Pass **dynamic data** from the server to the frontend.

### **Resources Required:**

* **Node.js**, **Express.js**, **EJS/Pug**, **Postman**, **Browser (Chrome/Firefox)**.

### **Step 1: Install Required Packages**

npm init -y

npm install express ejs pug

### **Step 2: Set Up Express Server**

**Create server.js**

const express = require('express');

const app = express();

const path = require('path');

// Set EJS as the template engine

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

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

// Sample Data

const users = [

{ name: "Alice", age: 25 },

{ name: "Bob", age: 30 },

{ name: "Charlie", age: 22 }

];

// Route to Render EJS Template

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

res.render('users', { users });

});

app.listen(3000, () => console.log('Server running on http://localhost:3000'));

### **Step 3: Create the Views Folder and Template Files**

**Create views/users.ejs**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>User List</title>

</head>

<body>

<h1>User List</h1>

<ul>

<% users.forEach(user => { %>

<li><%= user.name %> - <%= user.age %> years old</li>

<% }) %>

</ul>

</body>

</html>

### **Step 4: Run the Server and Test the Output**

1. Start the server:

node server.js

1. Open a browser and visit:

bash

http://localhost:3000/users

1. The **list of users** will be dynamically displayed on the webpage.

### **Using Pug as an Alternative Template Engine**

1. Modify **server.js** to use **Pug** instead of **EJS**:

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

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

1. Create **views/users.pug**:

doctype html

html

head

title User List

body

h1 User List

ul

each user in users

li= user.name + " - " + user.age + " years old"

1. Restart the server and visit http://localhost:3000/users to see the same **dynamic content** but rendered with **Pug** instead of **EJS**.

## **POST LAB EXERCISE**

### **QUESTIONS:**

1. What are the advantages of using a **template engine** over plain HTML?
2. How does **EJS** handle loops and conditionals?
3. What are the key differences between **EJS** and **Pug**?
4. How can you pass **form data** from an HTML page to an Express.js route?
5. How can template engines be used to create **layouts and partials** for reusability?

**ASSESSMENT PATTERN.**

|  |  |  |
| --- | --- | --- |
| **Description** | **Max Marks** | **Marks Awarded** |
| Pre Lab Exercise | **5** |  |
| In Lab Exercise | **10** |  |
| Post Lab Exercise | **5** |  |
| Viva | **10** |  |
| **Total** | **30** |  |
| **Faculty Signature** | |  |

**Experiment Number : 7 Date:**

# **Implement User Authentication in an Express Application**

## **PRE LAB EXERCISE**

### **Objective:**

* Understand authentication and authorization in web applications.
* Learn about **bcrypt** for password hashing and **JWT (JSON Web Token)** for authentication.
* Set up a basic Express.js application with **user registration and login**.

### **QUESTIONS:**

1. What is authentication, and why is it important?
2. How does JWT help in securing web applications?
3. What is the role of **bcrypt** in password security?
4. What are the key differences between **session-based** and **token-based** authentication?
5. How does middleware help in authentication?

## **IN LAB EXERCISE**

### **Objective:**

* Implement **user authentication** in an Express.js application.
* Use **bcrypt** for password hashing.
* Implement **JWT-based authentication** for secure access.

### **Resources Required:**

* **Node.js**, **Express.js**, **MongoDB**, **Postman**, **bcrypt**, **jsonwebtoken**.

### **Step 1: Install Required Packages**

npm init -y

npm install express mongoose bcryptjs jsonwebtoken dotenv cors

### **Step 2: Set Up Express and MongoDB**

**Create a .env file:**

PORT=3000

JWT\_SECRET=your\_secret\_key

MONGO\_URI=mongodb://localhost:27017/authDB

### **Step 3: Create** server.js

const express = require('express');

const mongoose = require('mongoose');

const dotenv = require('dotenv');

const cors = require('cors');

const authRoutes = require('./routes/authRoutes');

dotenv.config();

const app = express();

app.use(express.json());

app.use(cors());

mongoose.connect(process.env.MONGO\_URI, {

useNewUrlParser: true,

useUnifiedTopology: true

}).then(() => console.log("MongoDB Connected"));

app.use('/api/auth', authRoutes);

app.listen(process.env.PORT, () => console.log(`Server running on port ${process.env.PORT}`));

### **Step 4: Create Authentication Routes**

**Create routes/authRoutes.js**

const express = require('express');

const bcrypt = require('bcryptjs');

const jwt = require('jsonwebtoken');

const User = require('../models/User');

const router = express.Router();

// Register Route

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

const { username, password } = req.body;

const hashedPassword = await bcrypt.hash(password, 10);

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

await newUser.save();

res.json({ message: "User registered successfully!" });

});

// Login Route

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

const { username, password } = req.body;

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

if (!user) return res.status(400).json({ error: "User not found" });

const isMatch = await bcrypt.compare(password, user.password);

if (!isMatch) return res.status(400).json({ error: "Invalid credentials" });

const token = jwt.sign({ id: user.\_id }, process.env.JWT\_SECRET, { expiresIn: "1h" });

res.json({ message: "Login successful", token });

});

module.exports = router;

### **Step 5: Create User Model**

**Create models/User.js**

js

CopyEdit

const mongoose = require('mongoose');

const UserSchema = new mongoose.Schema({

username: { type: String, required: true, unique: true },

password: { type: String, required: true }

});

module.exports = mongoose.model("User", UserSchema);

### **Step 6: Test with Postman**

1. **Register a user:**
   * URL: http://localhost:3000/api/auth/register
   * Method: POST

{ "username": "admin", "password": "password123" }

* + Expected Response:

{ "message": "User registered successfully!" }

1. **Login:**
   * URL: http://localhost:3000/api/auth/login
   * Method: POST
   * Body:

{ "username": "admin", "password": "password123" }

* + Expected Response:

{ "message": "Login successful", "token": "your\_jwt\_token" }

## **POST LAB EXERCISE**

### **QUESTIONS:**

1. How does bcrypt ensure password security?
2. What is the significance of jsonwebtoken in authentication?
3. How can we implement role-based authentication in an Express app?
4. How would you store JWT tokens securely on the client side?
5. What additional security measures should be taken in authentication systems?

**ASSESSMENT PATTERN.**

|  |  |  |
| --- | --- | --- |
| **Description** | **Max Marks** | **Marks Awarded** |
| Pre Lab Exercise | **5** |  |
| In Lab Exercise | **10** |  |
| Post Lab Exercise | **5** |  |
| Viva | **10** |  |
| **Total** | **30** |  |
| **Faculty Signature** | |  |

**Experiment Number : 8 Date:**

# **Explore and Integrate Third-Party Node.js Extensions into an Express App**

## **PRE LAB EXERCISE**

### **Objective:**

* Understand the importance of **third-party extensions (middleware, libraries, and modules)** in Express.js applications.
* Learn how to integrate popular **Node.js extensions** to enhance functionality.
* Implement middleware for **logging, security, and API request handling**.

### **QUESTIONS:**

1. What are **third-party extensions** in Node.js, and why are they useful?
2. What is the purpose of **Express middleware**?
3. How do you install and use third-party **Node.js modules**?
4. Name some commonly used third-party packages in an **Express.js application**.
5. What is the difference between **built-in middleware** and **third-party middleware** in Express.js?

## **IN LAB EXERCISE**

### **Objective:**

* Install and use **third-party Node.js packages** in an Express application.
* Implement middleware for **logging (morgan), security (helmet), and request parsing (body-parser)**.
* Handle **CORS** (Cross-Origin Resource Sharing).

### **Resources Required:**

* **Node.js**, **Express.js**, **Morgan (Logger)**, **Helmet (Security)**, **Body-Parser (Request Parsing)**, **CORS Middleware**.

### **Step 1: Install Required Packages**

npm init -y

npm install express morgan helmet body-parser cors

### **Step 2: Set Up Express Server**

**Create server.js**

const express = require('express');

const morgan = require('morgan');

const helmet = require('helmet');

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

const cors = require('cors');

const app = express();

// Middleware Setup

app.use(morgan('dev')); // Logging HTTP requests

app.use(helmet()); // Security middleware

app.use(bodyParser.json()); // Parse JSON request bodies

app.use(cors()); // Enable CORS for cross-origin requests

// Sample Route

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

res.send('Welcome to the Express Server with Third-Party Extensions!');

});

// API Route to test Body-Parser

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

res.json({ message: 'Data received successfully', data: req.body });

});

app.listen(3000, () => console.log('Server running on http://localhost:3000'));

### **Step 3: Explanation of Third-Party Modules Used**

1. **Morgan (HTTP Request Logger)**
   * Helps in **logging incoming requests** for debugging.
   * Logs request details like **method, URL, status, response time**.
2. **Helmet (Security Middleware)**
   * Protects against common security vulnerabilities like **Cross-Site Scripting (XSS)** and **Clickjacking**.
3. **Body-Parser (Request Parsing Middleware)**
   * Parses incoming **JSON data** from POST requests.
   * Without this, req.body would be undefined.
4. **CORS (Cross-Origin Resource Sharing)**
   * Allows API access from **different domains**.
   * Prevents **CORS policy issues** when making frontend requests.

### **Step 4: Run and Test the Server**

1. Start the server:

node server.js

1. Open a browser and visit:

http://localhost:3000

* + You should see: **"Welcome to the Express Server with Third-Party Extensions!"**

1. Test **Body-Parser Middleware** using Postman:
   * **POST request** to http://localhost:3000/data
   * **Body (JSON format):**

{

"name": "John Doe",

"age": 28

}

* + **Response:**

{

"message": "Data received successfully",

"data": {

"name": "John Doe",

"age": 28

}

}

## **POST LAB EXERCISE**

### **QUESTIONS:**

1. What are the benefits of using **third-party middleware** in an Express app?
2. How does **Morgan** help in debugging an application?
3. Why is **Helmet** important for web security?
4. What is the role of **Body-Parser**, and why do we need it?
5. How does **CORS** help in web applications?

**ASSESSMENT PATTERN.**

|  |  |  |
| --- | --- | --- |
| **Description** | **Max Marks** | **Marks Awarded** |
| Pre Lab Exercise | **5** |  |
| In Lab Exercise | **10** |  |
| Post Lab Exercise | **5** |  |
| Viva | **10** |  |
| **Total** | **30** |  |
| **Faculty Signature** | |  |

**Experiment Number : 9 Date:**

# **Experiment 9: Create a Multi-Page Web Application with Authentication and Routing**

## **PRE LAB EXERCISE**

### **Objective:**

* Understand **Express.js Routing** and how to create a **multi-page web application**.
* Learn **user authentication** using **Passport.js and bcrypt** for hashing passwords.
* Implement session management with **express-session**.

### **QUESTIONS:**

1. What is **routing** in Express.js, and why is it important?
2. How does **authentication** work in a web application?
3. What is **Passport.js**, and why is it used?
4. Why do we need to **hash passwords**, and what library is commonly used?
5. What is **session-based authentication**, and how does it work in Express.js?

## **IN LAB EXERCISE**

### **Objective:**

* Develop a **multi-page web application** with Express.js.
* Implement **user authentication** with **registration, login, and logout**.
* Use **session management** to store authenticated user sessions.

### **Resources Required:**

* **Node.js**, **Express.js**, **EJS (Embedded JavaScript Templates)**, **MongoDB (for user storage)**.
* **bcrypt** (for password hashing), **Passport.js** (for authentication), **express-session** (for session management).

### **Step 1: Install Required Packages**

npm init -y

npm install express ejs mongoose bcrypt passport passport-local express-session connect-flash

### **Step 2: Set Up Express Server**

**Create server.js**

const express = require('express');

const mongoose = require('mongoose');

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

const passport = require('passport');

const flash = require('connect-flash');

const bcrypt = require('bcrypt');

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

const app = express();

// Connect to MongoDB

mongoose.connect('mongodb://127.0.0.1:27017/auth\_demo', { useNewUrlParser: true, useUnifiedTopology: true });

// Define User Schema

const UserSchema = new mongoose.Schema({

username: String,

password: String

});

const User = mongoose.model('User', UserSchema);

// Middleware Setup

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

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

app.use(session({

secret: 'mysecret',

resave: false,

saveUninitialized: false

}));

app.use(passport.initialize());

app.use(passport.session());

app.use(flash());

// Passport Authentication Strategy

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

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

if (!user) return done(null, false, { message: 'User not found' });

const isMatch = await bcrypt.compare(password, user.password);

return isMatch ? done(null, user) : done(null, false, { message: 'Incorrect password' });

}));

passport.serializeUser((user, done) => done(null, user.id));

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

const user = await User.findById(id);

done(null, user);

});

// Routes

app.get('/', (req, res) => res.render('home'));

app.get('/login', (req, res) => res.render('login', { message: req.flash('error') }));

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

app.get('/dashboard', isAuthenticated, (req, res) => res.render('dashboard', { user: req.user }));

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

req.logout(() => res.redirect('/'));

});

// Registration Route

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

const hashedPassword = await bcrypt.hash(req.body.password, 10);

await User.create({ username: req.body.username, password: hashedPassword });

res.redirect('/login');

});

// Login Route

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

successRedirect: '/dashboard',

failureRedirect: '/login',

failureFlash: true

}));

// Authentication Middleware

function isAuthenticated(req, res, next) {

return req.isAuthenticated() ? next() : res.redirect('/login');

}

// Start Server

app.listen(3000, () => console.log('Server running on http://localhost:3000'));

### **Step 3: Create Views (EJS Templates)**

**Create a views folder** and add the following files:

#### ****1.**** views/home.ejs ****(Homepage)****

<!DOCTYPE html>

<html lang="en">

<head><title>Home</title></head>

<body>

<h2>Welcome to Our App</h2>

<a href="/register">Register</a> | <a href="/login">Login</a>

</body>

</html>

#### ****2.**** views/register.ejs ****(User Registration Page)****

<!DOCTYPE html>

<html lang="en">

<head><title>Register</title></head>

<body>

<h2>Register</h2>

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

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

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

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

</form>

</body>

</html>

#### ****3.**** views/login.ejs ****(Login Page)****

<!DOCTYPE html>

<html lang="en">

<head><title>Login</title></head>

<body>

<h2>Login</h2>

<% if (message.length > 0) { %>

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

<% } %>

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

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

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

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

</form>

</body>

</html>

#### ****4.**** views/dashboard.ejs ****(Dashboard Page)****

<!DOCTYPE html>

<html lang="en">

<head><title>Dashboard</title></head>

<body>

<h2>Welcome, <%= user.username %>!</h2>

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

</body>

</html>

### **Step 4: Run and Test the Application**

1. Start MongoDB:

mongod

1. Run the server:

node server.js

1. Open the browser and visit:

http://localhost:3000

1. Register a new user, log in, and access the **dashboard**.

## **POST LAB EXERCISE**

### **QUESTIONS:**

1. What is the role of **express-session** in user authentication?
2. Why do we use **bcrypt** for password hashing?
3. How does **Passport.js** handle authentication?
4. What is the importance of **middleware** in Express routing?
5. What are the advantages of using **EJS templates** for dynamic pages?

**ASSESSMENT PATTERN.**

|  |  |  |
| --- | --- | --- |
| **Description** | **Max Marks** | **Marks Awarded** |
| Pre Lab Exercise | **5** |  |
| In Lab Exercise | **10** |  |
| Post Lab Exercise | **5** |  |
| Viva | **10** |  |
| **Total** | **30** |  |
| **Faculty Signature** | |  |

**Experiment Number : 10 Date:**

# **Experiment 10: Create a Web Application with MongoDB and Node.js**

## **PRE LAB EXERCISE**

### **Objective:**

* Develop a web application using **Node.js** and **MongoDB**.
* Implement **CRUD (Create, Read, Update, Delete)** operations.
* Use **Express.js** for handling server requests.
* Create a **RESTful API** for managing data.

### **QUESTIONS:**

1. What is MongoDB, and how is it different from relational databases?
2. How does **Express.js** help in building a web server?
3. What is **Mongoose**, and why is it used?
4. What are the HTTP methods used in a RESTful API?
5. How does **MongoDB store data**, and what is a **document** in MongoDB?

## **POST LAB EXERCISE**

### **QUESTIONS:**

1. What is the purpose of **Mongoose** in a Node.js application?
2. How do you create a new document in MongoDB using Mongoose?
3. Explain the significance of **method-override** in this application.
4. What are the different HTTP methods used in this CRUD application?
5. How can you improve security in a MongoDB-based web application?

**ASSESSMENT PATTERN.**

|  |  |  |
| --- | --- | --- |
| **Description** | **Max Marks** | **Marks Awarded** |
| Pre Lab Exercise | **5** |  |
| In Lab Exercise | **10** |  |
| Post Lab Exercise | **5** |  |
| Viva | **10** |  |
| **Total** | **30** |  |
| **Faculty Signature** | |  |

**ASSESSMENT RUBRICS**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **PRE LAB WORK -** This evaluates the student’s preparation and understanding before the lab session | | | | | |
| **Criteria** | **Excellent (5)** | **Good (4)** | **Satisfactory (3)** | **Needs Improvement (1-2)** | **Not Attempted (0)** |
| **Understanding of Objectives** | Demonstrates complete understanding of the lab objectives. | Partial understanding of objectives. | Basic understanding; some gaps present. | Minimal understanding of objectives. | No preparation evident. |
| **Pre-Lab Task Completion** | Pre-lab exercises are fully completed and accurate. | Mostly completed with minor errors. | Partially completed; several errors. | Very limited attempt. | No submission. |
| **IN LAB WORK -** This evaluates the execution and problem-solving abilities during the lab. | | | | | |
| **Criteria** | **Excellent (9-10)** | **Good (7-8)** | **Satisfactory (5-6)** | **Needs Improvement (1-4)** | **Not Attempted (0)** |
| **Implementation Accuracy** | Code/solution fully meets the problem statement with no errors. | Mostly correct with minor errors. | Partially correct; significant errors present. | Minimal progress made towards solving the problem. | No implementation. |
| **Problem-Solving Approach** | Demonstrates excellent logic and efficiency. | Good logic; minor inefficiencies. | Basic logic; significant inefficiencies. | Poor logic or random attempts. | No problem-solving effort. |
| **Tool/Technology Usage** | Effectively uses all relevant tools and software for the course. | Good use of tools with minor gaps. | Basic usage; limited familiarity with tools. | Struggles with tools; frequent errors. | No tool usage evident. |
| **Code/Design Quality** | Clean, efficient, and well-documented code/design. | Mostly clean and efficient; minor issues in documentation. | Basic implementation; lacks clarity or efficiency. | Poor quality; messy or unclear code/design. | No code or design submitted. |
| **POST LAB WORK-** This evaluates the analysis, reflection, and reporting after the lab session | | | | | |
| **Criteria** | **Excellent (5)** | **Good (4)** | **Satisfactory (3)** | **Needs Improvement (1-2)** | **Not Attempted (0)** |
| **Result Analysis** | Provides a thorough and accurate analysis of results. | Accurate analysis with minor omissions. | Basic analysis; some errors or omissions. | Limited or unclear analysis. | No analysis provided. |
| **Post-Lab Questions/Reflections** | Thoughtful responses; demonstrates deeper understanding. | Good responses with minor gaps in reasoning. | Basic responses; some misconceptions. | Minimal or unclear responses. | No attempt. |