**Experiment Number : 9 Date:31/03/2025**

# **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?

Routing in Express.js defines how an application responds to different HTTP requests (GET, POST, etc.). It helps organize and manage different parts of a web application efficiently.

1. How does **authentication** work in a web application?

Authentication verifies a user's identity using credentials like a username and password. After verification, the server issues a token or session to keep the user logged in.

1. What is **Passport.js**, and why is it used?

Passport.js is an authentication middleware for Node.js. It simplifies user authentication with strategies like local login, Google, and Facebook authentication.

1. Why do we need to **hash passwords**, and what library is commonly used?

Hashing passwords makes them unreadable, protecting them from attacks. The most commonly used library for hashing in Node.js is **bcrypt**.

1. What is **session-based authentication**, and how does it work in Express.js?

Session-based authentication stores a user's login session on the server. When a user logs in, a session ID is created and stored in a cookie, allowing the server to recognize the user on future requests.

## **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('bcryptjs');//replace bacrypt as bcryptjs

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>

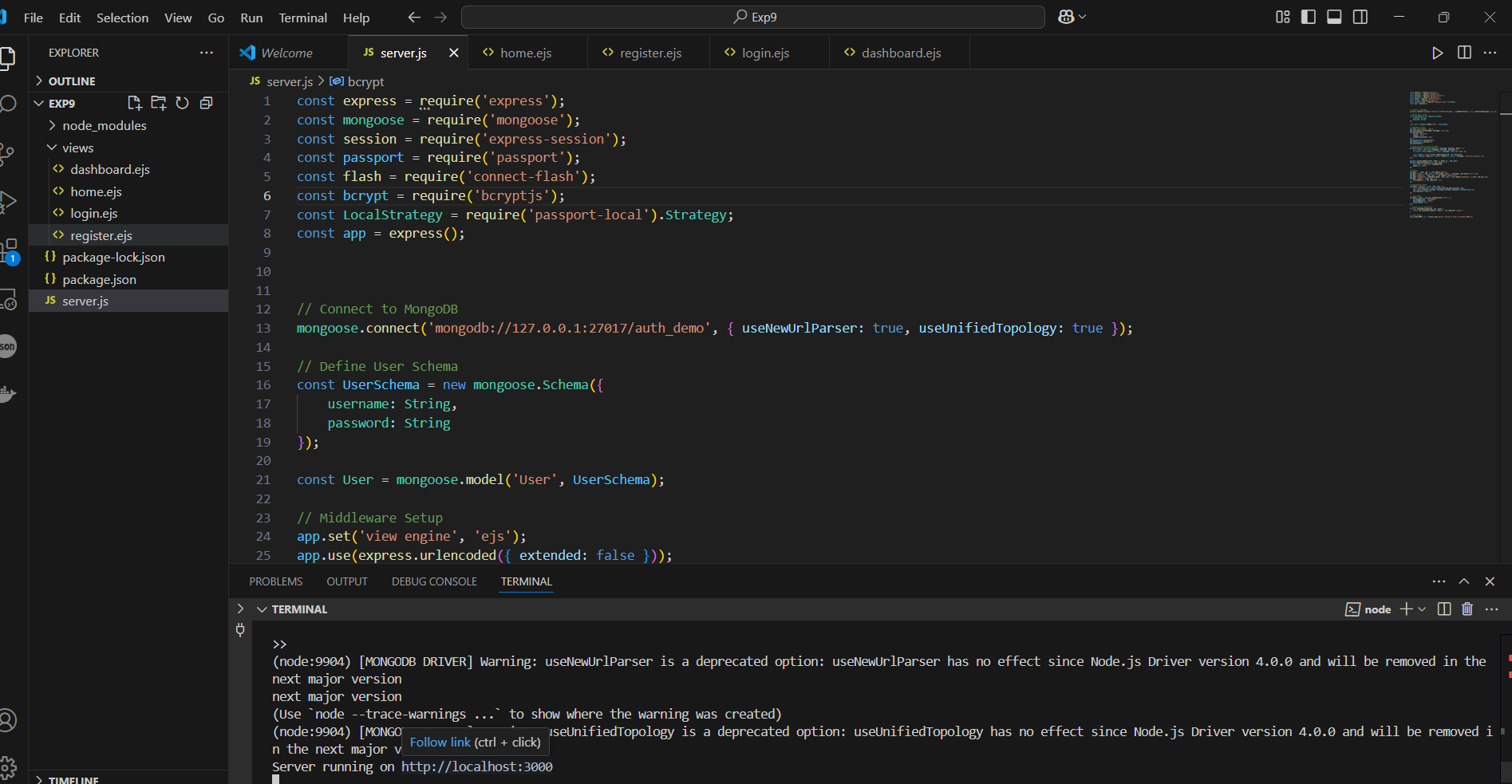
<body>

<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**.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a login box

AI-generated content may be incorrect.

A close-up of a website

AI-generated content may be incorrect.

**After Clicking Logout:**

A white background with black lines

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

## **POST LAB EXERCISE**

### **QUESTIONS:**

1. What is the role of **express-session** in user authentication?

It stores user session data on the server, allowing users to stay logged in across multiple requests.

1. Why do we use **bcrypt** for password hashing?

It securely hashes passwords using a strong encryption algorithm, making them difficult to crack.

1. How does **Passport.js** handle authentication?

It provides middleware to authenticate users using various strategies (e.g., local, OAuth) and manages session-based authentication.

1. What is the importance of **middleware** in Express routing?

Middleware processes requests before they reach routes, handling tasks like authentication, logging, and error handling.

1. What are the advantages of using **EJS templates** for dynamic pages?

* Embeds JavaScript into HTML for dynamic content
* Supports reusable components
* Integrates easily with Express
* Improves code organization

**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** | |  |