**Batch: C1**

**Roll No.: 16010122221**

**Experiment / assignment / tutorial No.6**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

Title: Demonstrate the Use of ExpressJS

**AIM:** To implement the ExpressJs Concepts based on the following topics.

# Problem Definition:

**Problem statement:**

Consider the basic concepts of Express.js, which are useful in the creation of an application. Considering the following points, demonstrate the functionality of each with a simple script

# Scaffolding:

* + Demonstrate express scaffolding to fulfil the following requirements.

Example: Consider Grocery Delivery Application and demonstrate the Scaffolding Scaffold the application to create different routes such as. Sign up Page: (Root/ Homepage)

1. Serving static files using Express.js: With the help of Built in middleware, express. Static () to demonstrate the usage of serving static files in express.

To demonstrate the above make a use of

* + Use of images where it should accept any type of image
  + Use of CSS and HTMLfiles.
  + Make a Use json file of employee information, add file to the static folder, and show the response on the browser

Note: · Assume your own data whenever required to perform the operation. Write code for everything

# Resources used:

* + [Express.js Documentation](https://expressjs.com/)
  + Node.js Documentation
  + [MDN Web Docs](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs)
  + [YouTube Tutorials on Express.js](https://www.youtube.com/results?search_query=express.js%2Btutorial)

# Expected OUTCOME of Experiment:

**CO 2:**. Illustrate the concepts of various front-end, back-end web application development technologies & frameworks using different web development tools.

# Books/ Journals/ Websites referred:

* "Express in Action" by Evan Hahn
* "Node.js Design Patterns" by Mario Casciaro and Luciano Mammino
* Various online resources and tutorials for Express.js and Node.js.

# Pre Lab/ Prior Concepts:

**Write details about the following content**

# Express:

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. It simplifies the process of building web servers and APIs, handling routes, middleware, and requests/responses efficiently.

# Basic Routing:

Routing in Express allows you to define the endpoints of your application. Each route can respond to different HTTP methods (GET, POST, PUT, DELETE) and can execute middleware functions, serve HTML pages, or return JSON data.

# Middleware:

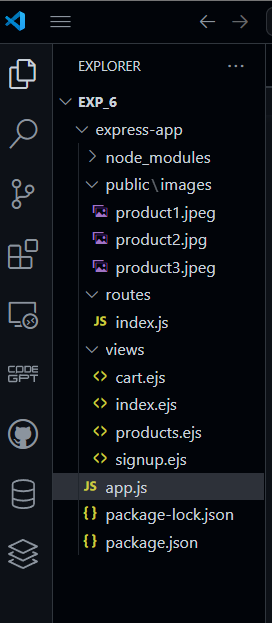
Middleware functions are functions that have access to the request object (req), the response object (res), and the next middleware function in the application's request-response cycle. They can perform actions on the request, modify the request/response objects, and end the request-response cycle.

# Templating:

Templating is a way to generate HTML dynamically by inserting data into HTML templates. In Express, templating engines like EJS, Pug, or Handlebars can be used to render views based on server-side data.

# Implementation Details:

* 1. Project Structure:



# Serve Static Files in Express:

In your app.js, ensure that the Express server is set up to serve static files from the public folder:

# App.js

const express = require("express");

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

const path = require("path");

const indexRouter = require("./routes/index");

const app = express();

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

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

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

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

app.use(express.static(path.join(\_\_dirname, "public")));

app.use("/", indexRouter);

app.listen(PORT, () => {

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

});

* 1. **Store Product Data with Image References:**

In your routes/index.js, ensure that your products have an associated image URL that points to the images in the public/images folder:

# Routes/index.js

const express = require("express");

const router = express.Router();

const users = [];

const products = [

  { id: 1, name: "img1", price: 50000, image: "product1.jpeg" },

  { id: 2, name: "img2", price: 40000, image: "product2.jpg" },

  { id: 3, name: "img3", price: 30000, image: "product3.jpeg" }

];

let cart = [];

router.get("/", (req, res) => res.render("index"));

router.get("/signup", (req, res) => res.render("signup", { error: null }));

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

  const { username, password } = req.body;

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

  if (existingUser) {

    res.render("signup", { error: "Username already exists." });

  } else {

    users.push({ username, password });

    res.send(`<h2>Signup Successful!</h2><a href="/products">Go to Products</a>`);

  }

});

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

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

});

router.post("/add-to-cart", (req, res) => {

  const productId = parseInt(req.body.productId);

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

  if (product) {

    cart.push(product);

  }

  res.redirect("/products");

});

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

  res.render("cart", { cart });

});

module.exports = router;

# Modify products.ejs to Display Images:

<!DOCTYPE html>

<html>

<head><title>Products</title></head>

<body>

  <h1>Products</h1>

  <ul>

*<*% products.forEach(product => { %>

      <li>

        <img src="*<*%= product.image %>" alt="*<*%= product.name %>" style="width: 100px; height: 100px;" />

*<*%= product.name %> - $*<*%= product.price %>

        <form action="/add-to-cart" method="POST">

          <input type="hidden" name="productId" value="*<*%= product.id %>" />

          <button type="submit">Add to Cart</button>

        </form>

      </li>

*<*% }); %>

  </ul>

  <a href="/cart">View Cart</a>

  <a href="/">Home</a>

</body>

</html>

* 1. **Add Cart Logic:**

Implement the "Add to Cart" functionality in your index.js file. You need a route that will handle adding products to the cart:

Index.js

=

const express = require("express");

const router = express.Router();

const users = [];

const products = [

  { id: 1, name: "img1", price: 50000, image: "product1.jpeg" },

  { id: 2, name: "img2", price: 40000, image: "product2.jpg" },

  { id: 3, name: "img3", price: 30000, image: "product3.jpeg" }

];

let cart = [];

router.get("/", (req, res) => res.render("index"));

router.get("/signup", (req, res) => res.render("signup", { error: null }));

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

  const { username, password } = req.body;

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

  if (existingUser) {

    res.render("signup", { error: "Username already exists." });

  } else {

    users.push({ username, password });

    res.send(`<h2>Signup Successful!</h2><a href="/products">Go to Products</a>`);

  }

});

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

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

});

router.post("/add-to-cart", (req, res) => {

  const productId = parseInt(req.body.productId);

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

  if (product) {

    cart.push(product);

  }

  res.redirect("/products");

});

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

  res.render("cart", { cart });

});

module.exports = router;

# Create the Cart View (cart.ejs):

You also need to create a cart.ejs file to display the items in the cart: Cart.ejs

<!DOCTYPE html>

<html>

<head><title>Cart</title></head>

<body>

  <h1>Your Cart</h1>

  <ul>

*<*% if (cart.length === 0) { %>

      <li>Your cart is empty.</li>

*<*% } else { %>

*<*% cart.forEach(item => { %>

        <li>*<*%= item.name %> - $*<*%= item.price %></li>

*<*% }); %>

*<*% } %>

  </ul>

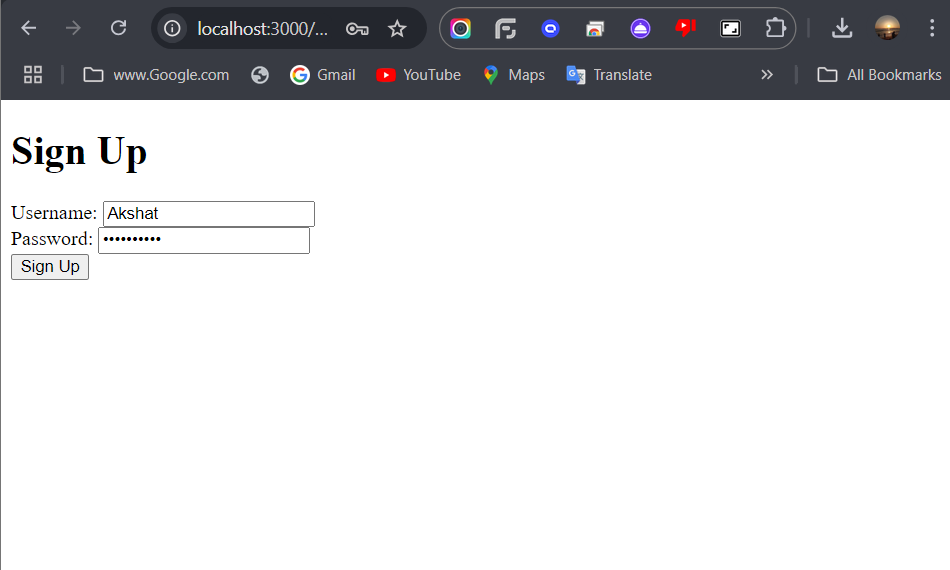
  <a href="/products">Go to Products</a>

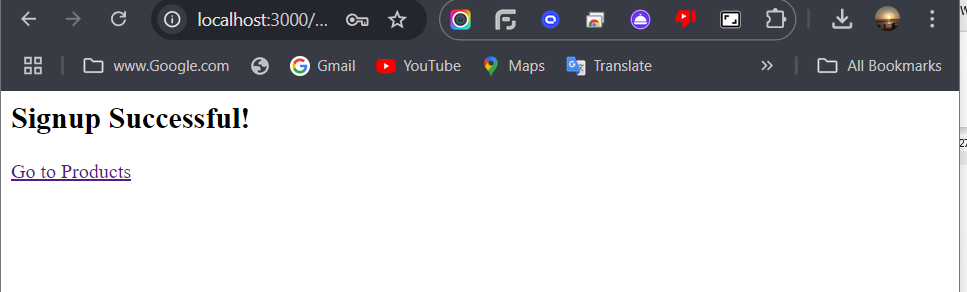
  <a href="/">Home</a>

</body>

</html>

Output-









Steps for Execution:

# Scaffolding

* + **Step 1:** Open your terminal and run the express command to scaffold the application structure.
  + **Step 2:** Navigate to the project directory and install the required dependencies using npm install.
  + **Step 3:** Open the routes/index.js file and implement the necessary routes.
  + **Step 4:** Run the server using npm start.
  + **Step 5:** Open a browser and navigate to http://localhost:3000 to verify that the routes are working as expected.

# Serving Static Files

* + **Step 1:** Create the project directory and initialize Node.js.
  + **Step 2:** Install Express and create a public folder for static assets.
  + **Step 3:** Add the necessary static files (HTML, CSS, images, JSON) in the public folder.
  + **Step 4:** Use express.static() in app.js to serve static files.
  + **Step 5:** Run the server and access the static files by navigating to http://localhost:3000.

**Conclusion:** You have successfully created a MERN application with a products page that displays product images, prices, and an "Add to Cart" feature, along with a cart page to view selected items.