Name: Nidhi B. Valand

Roll no:72

Sub:701(Full Stack )

Here GitHub link:

Assignment-2

Que-1:

App.js

const express = require('express');

const mongoose = require('mongoose');

const multer = require('multer');

const path = require('path');

const app = express();

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

const mongoURI = 'mongodb://localhost:27017/upload';

mongoose.connect(mongoURI, { useNewUrlParser: true, useUnifiedTopology: true })

    .then(() => console.log('MongoDB connected!'))

    .catch(err => console.error('MongoDB connection error:', err));

// Middleware

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

app.use('/uploads', express.static('uploads')); // Serve uploaded files

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

app.listen(PORT, () => {

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

});

const storage = multer.diskStorage({

    destination: (req, file, cb) => {

        cb(null, 'uploads/');

    },

    filename: (req, file, cb) => {

        cb(null, Date.now() + path.extname(file.originalname)); // Appending extension

    }

});

const upload = multer({

    storage: storage,

    limits: { fileSize: 1 \* 1024 \* 1024 }, // Limit file size to 1MB

    fileFilter: (req, file, cb) => {

        const filetypes = /jpeg|jpg|png|gif|pdf/;

        const mimetype = filetypes.test(file.mimetype);

        const extname = filetypes.test(path.extname(file.originalname).toLowerCase());

        if (mimetype && extname) {

            return cb(null, true);

        }

        cb("Error: File type not supported");

    }

});

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

// Render registration form

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

    res.render('register');

});

// Handle user registration

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

    const { username, email } = req.body;

    const files = req.files.map(file => file.filename);

    const user = new User({ username, email, files });

    await user.save();

    res.redirect('/files');

});

// List uploaded files

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

    const users = await User.find();

    res.render('files', { users });

});

// Download file

app.get('/files/download/:filename', (req, res) => {

    const file = path.join(\_\_dirname, 'uploads', req.params.filename);

    res.download(file);

});

User.js

const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({

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

    email: { type: String, required: true },

    files: [{ type: String }]

});

module.exports = mongoose.model('User', userSchema);

views folder => 2 files =>files.js and register.js

files.ejs

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Uploaded Files</title>

</head>

<body>

    <h1>Uploaded Files</h1>

    <ul>

        <% users.forEach(user => { %>

            <li><strong><%= user.username %></strong> (<%= user.email %>):

                <% user.files.forEach(file => { %>

                    <a href="/files/download/<%= file %>"><%= file %></a>

                <% }) %>

            </li>

        <% }) %>

    </ul>

</body>

</html>

Register.ejs

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Register</title>

</head>

<body>

    <h1>Register</h1>

    <form action="/register" method="POST" enctype="multipart/form-data">

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

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

        <input type="file" name="files" multiple required><br>

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

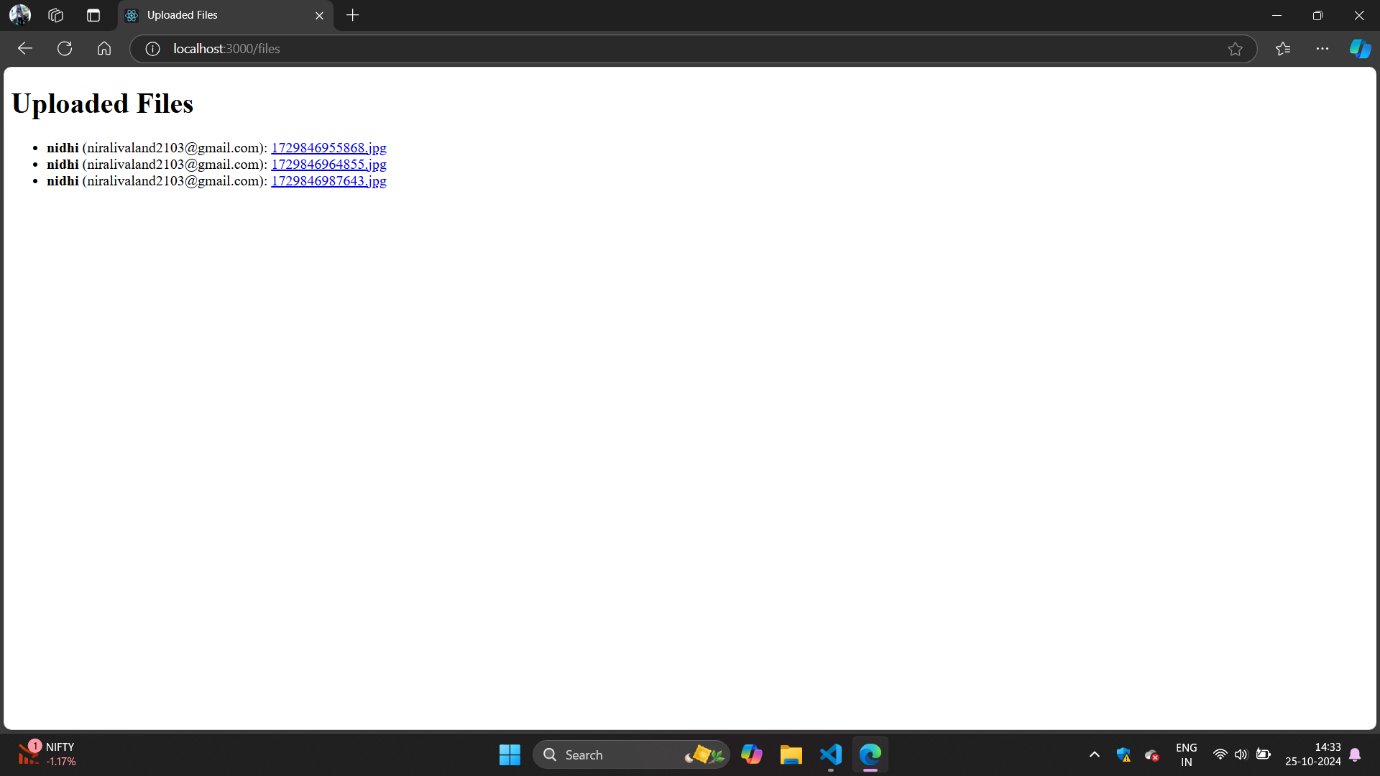
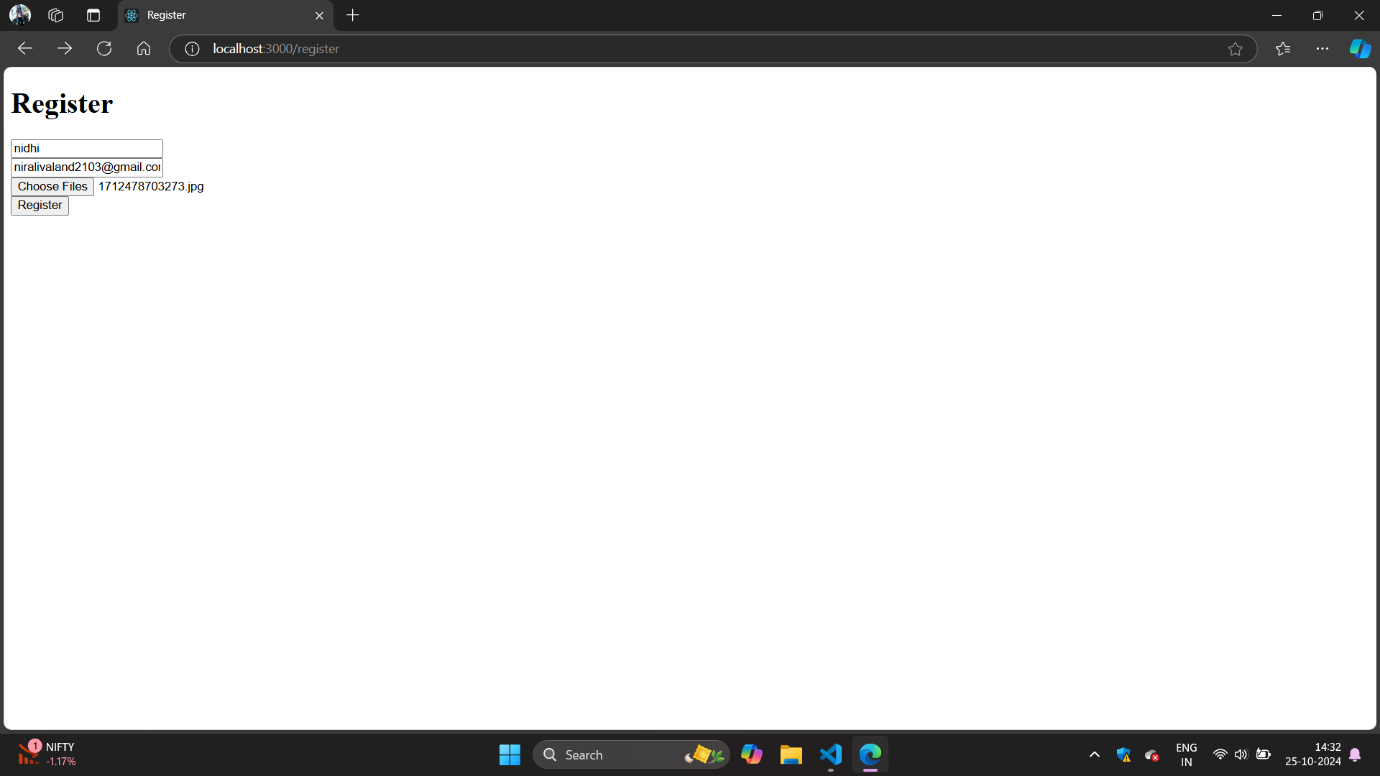
    </form>

</body>

</html>

Uploads folder which contains uploaded files

Output:



Que-2:

App.js

const express = require('express');

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

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

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

const path = require('path');

const app = express();

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

// Simple in-memory user store for demonstration purposes

const users = [{ username: 'Nidhi', password: 'Nidhi@123' }];

// Setup session

app.use(session({

    secret: 'secret\_key', // Replace with a strong secret in production

    resave: false,

    saveUninitialized: true,

    cookie: { maxAge: 60000 } // 1 minute

}));

// Flash messages middleware

app.use(flash());

// Middleware

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

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

app.use(express.static(path.join(\_\_dirname, 'public'))); // Serve static files

// Render login form

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

    res.render('login', { messages: req.flash('error') });

});

// Handle login

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

    const { username, password } = req.body;

    // Check user credentials

    const user = users.find(u => u.username === username && u.password === password);

    if (user) {

        req.session.user = user;

        req.flash('success', 'Logged in successfully!');

        return res.redirect('/dashboard');

    }

    req.flash('error', 'Invalid username or password');

    res.redirect('/login');

});

// Render dashboard

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

    if (!req.session.user) {

        req.flash('error', 'Please log in first');

        return res.redirect('/login');

    }

    res.render('dashboard', { user: req.session.user });

});

// Logout

// Logout

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

    req.flash('success', 'Logged out successfully'); // Set flash message before destroying session

    req.session.destroy(err => {

        if (err) {

            return res.redirect('/dashboard'); // Handle session destruction error

        }

        res.redirect('/login'); // Redirect to login after session is destroyed

    });

});

app.listen(PORT, () => {

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

});

Views folder=>

Dashboard.ejs

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Dashboard</title>

</head>

<body>

    <h1>Welcome, <%= user.username %></h1>

    <a href="/logout">Logout</a>

</body>

</html>

Login.ejs

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Login</title>

</head>

<body>

    <h1>Login</h1>

    <% if (messages.length) { %>

        <ul>

            <% messages.forEach(msg => { %>

                <li><%= msg %></li>

            <% }) %>

        </ul>

    <% } %>

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

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

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

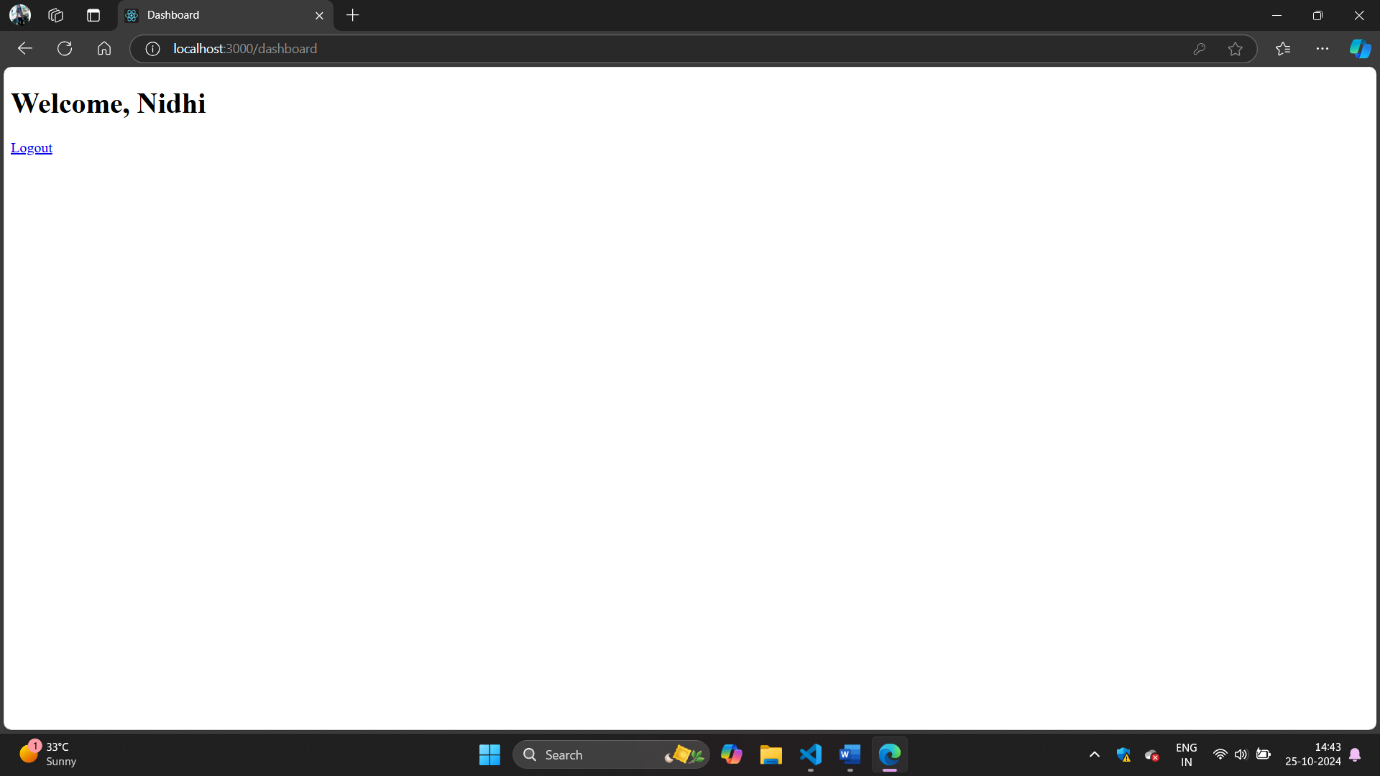
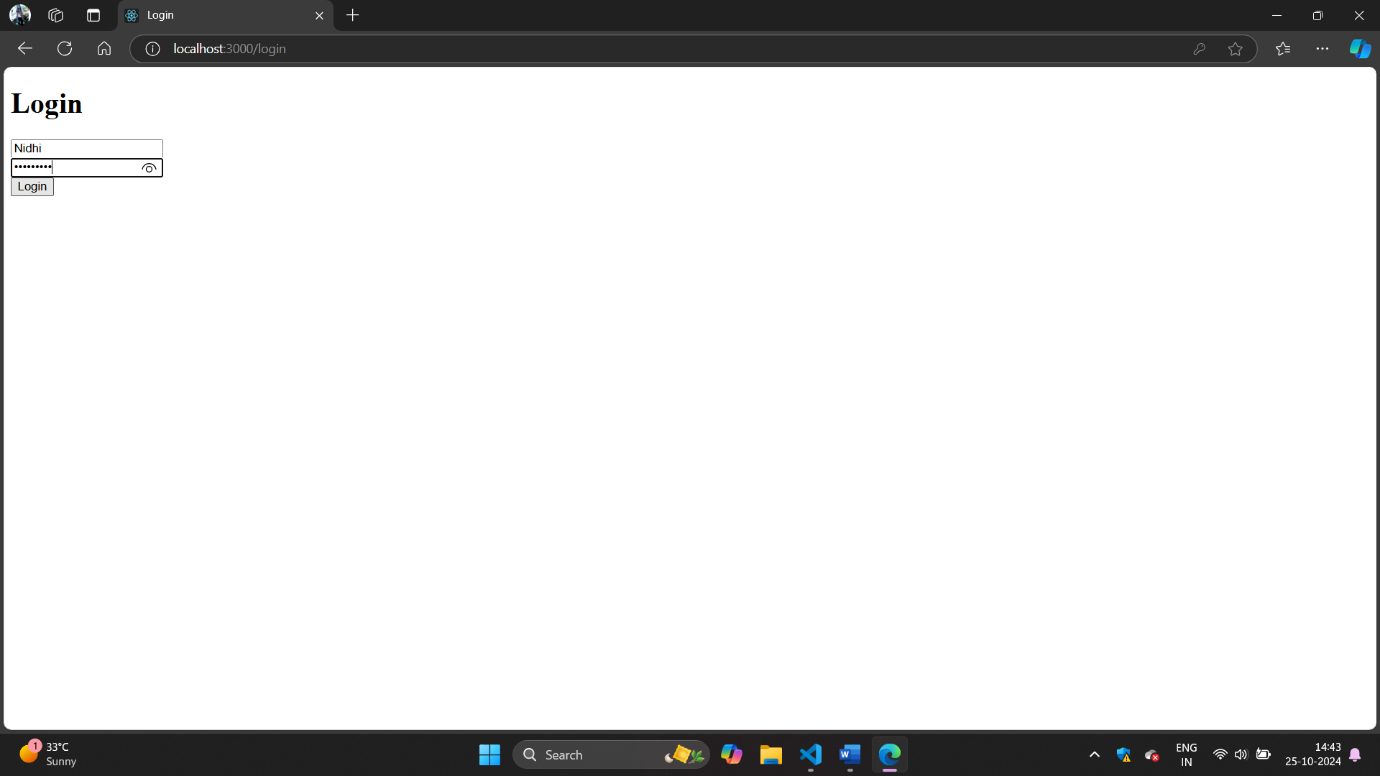
        <button type="submit">Login</button>

    </form>

</body>

</html>

Output:



Que-3:

App.js

const express = require('express');

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

const RedisStore = require('connect-redis').default; // Import Redis store

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

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

const redis = require('redis');

const path = require('path');

const app = express();

const PORT = process.env.PORT || 8000;

// Configure Redis client

// const redisClient = redis.createClient();

const redisClient = redis.createClient({

    host: '127.0.0.1', // Change if necessary

    port: 6379 // Change if necessary

});

redisClient.on('error', (err) => console.log('Redis Client Error', err));

// Simple in-memory user store for demonstration purposes

const users = [{ username: 'testuser', password: 'password123' }];

// Setup session with Redis store

app.use(session({

    store: new RedisStore({ client: redisClient }),

    secret: 'secret\_key', // Replace with a strong secret in production

    resave: false,

    saveUninitialized: false,

    cookie: { maxAge: 60000 } // 1 minute

}));

// Flash messages middleware

app.use(flash());

// Middleware

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

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

app.use(express.static(path.join(\_\_dirname, 'public'))); // Serve static files

// Render login form

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

    res.render('login', { messages: req.flash('error') });

});

// Handle login

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

    const { username, password } = req.body;

    // Check user credentials

    const user = users.find(u => u.username === username && u.password === password);

    if (user) {

        req.session.user = user;

        req.flash('success', 'Logged in successfully!');

        return res.redirect('/dashboard');

    }

    req.flash('error', 'Invalid username or password');

    res.redirect('/login');

});

// Render dashboard

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

    if (!req.session.user) {

        req.flash('error', 'Please log in first');

        return res.redirect('/login');

    }

    res.render('dashboard', { user: req.session.user });

});

// Logout

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

    req.flash('success', 'Logged out successfully');

    req.session.destroy(err => {

        if (err) {

            return res.redirect('/dashboard');

        }

        res.redirect('/login');

    });

});

// Start the server

app.listen(PORT, () => {

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

});

// Connect to Redis

(async () => {

    try {

        await redisClient.connect();

        console.log('Connected to Redis');

    } catch (err) {

        console.error('Redis Client Error', err);

    }

})();

Que-4:

Server.js

const express = require('express');

const mongoose = require('mongoose');

const jwt = require('jsonwebtoken');

const bcrypt = require('bcryptjs');

const path = require('path');

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

const methodOverride = require('method-override');

const app = express();

const PORT = process.env.PORT || 3001;

// Middleware

app.use(express.json());

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

app.use(methodOverride('\_method')); // For PUT and DELETE methods

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

// Set views directory

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

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

// Session setup

app.use(session({

    secret: 'your\_secret\_key',

    resave: false,

    saveUninitialized: true,

}));

// Connect to MongoDB

mongoose.connect('mongodb://localhost:27017/studentDB', { useNewUrlParser: true, useUnifiedTopology: true })

    .then(() => console.log('MongoDB connected'))

    .catch(err => console.error(err));

// Student Schema

const studentSchema = new mongoose.Schema({

    name: String,

    email: { type: String, unique: true },

    password: String

});

const Student = mongoose.model('Student', studentSchema);

// Middleware for JWT verification

const authenticateJWT = (req, res, next) => {

    const token = req.session.token;

    if (!token) return res.redirect('/'); // Redirect if not logged in

    jwt.verify(token, 'your\_jwt\_secret', (err, user) => {

        if (err) return res.redirect('/'); // Redirect if token is invalid

        req.user = user;

        next();

    });

};

// Routes

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

    res.render('index');

});

// Register

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

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

    const newStudent = new Student({

        name: req.body.name,

        email: req.body.email,

        password: hashedPassword

    });

    try {

        await newStudent.save();

        res.status(201).send('Student registered');

    } catch (error) {

        res.status(400).send('Error registering student');

    }

});

// Login

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

    const student = await Student.findOne({ email: req.body.email });

    if (student && (await bcrypt.compare(req.body.password, student.password))) {

        const token = jwt.sign({ email: student.email }, 'your\_jwt\_secret', { expiresIn: '1h' });

        req.session.token = token;

        res.redirect('/students'); // Redirect to the students page

    } else {

        res.status(403).send('Invalid credentials');

    }

});

// Logout

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

    req.session.destroy(err => {

        if (err) return res.status(500).send('Could not log out');

        res.redirect('/');

    });

});

// View students

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

    try {

        const students = await Student.find();

        res.render('student', { students });  // Renders the student.ejs view

    } catch (error) {

        res.status(500).send('Error retrieving students');

    }

});

// Add student form

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

    res.render('insert');

});

// Handle adding a new student

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

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

    const newStudent = new Student({

        name: req.body.name,

        email: req.body.email,

        password: hashedPassword

    });

    try {

        await newStudent.save();

        res.redirect('/students');

    } catch (error) {

        res.status(400).send('Error creating student');

    }

});

// Update student form

app.get('/students/:id/edit', authenticateJWT, async (req, res) => {

    try {

        const student = await Student.findById(req.params.id);

        res.render('update', { student });

    } catch (error) {

        res.status(400).send('Error retrieving student');

    }

});

// Handle updating a student

app.put('/students/:id', authenticateJWT, async (req, res) => {

    const updateData = {

        name: req.body.name,

        email: req.body.email,

    };

    if (req.body.password) {

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

    }

    try {

        await Student.findByIdAndUpdate(req.params.id, updateData);

        res.redirect('/students');

    } catch (error) {

        res.status(400).send('Error updating student');

    }

});

// Handle deleting a student

app.delete('/students/:id', authenticateJWT, async (req, res) => {

    try {

        await Student.findByIdAndDelete(req.params.id);

        res.redirect('/students');

    } catch (error) {

        res.status(400).send('Error deleting student');

    }

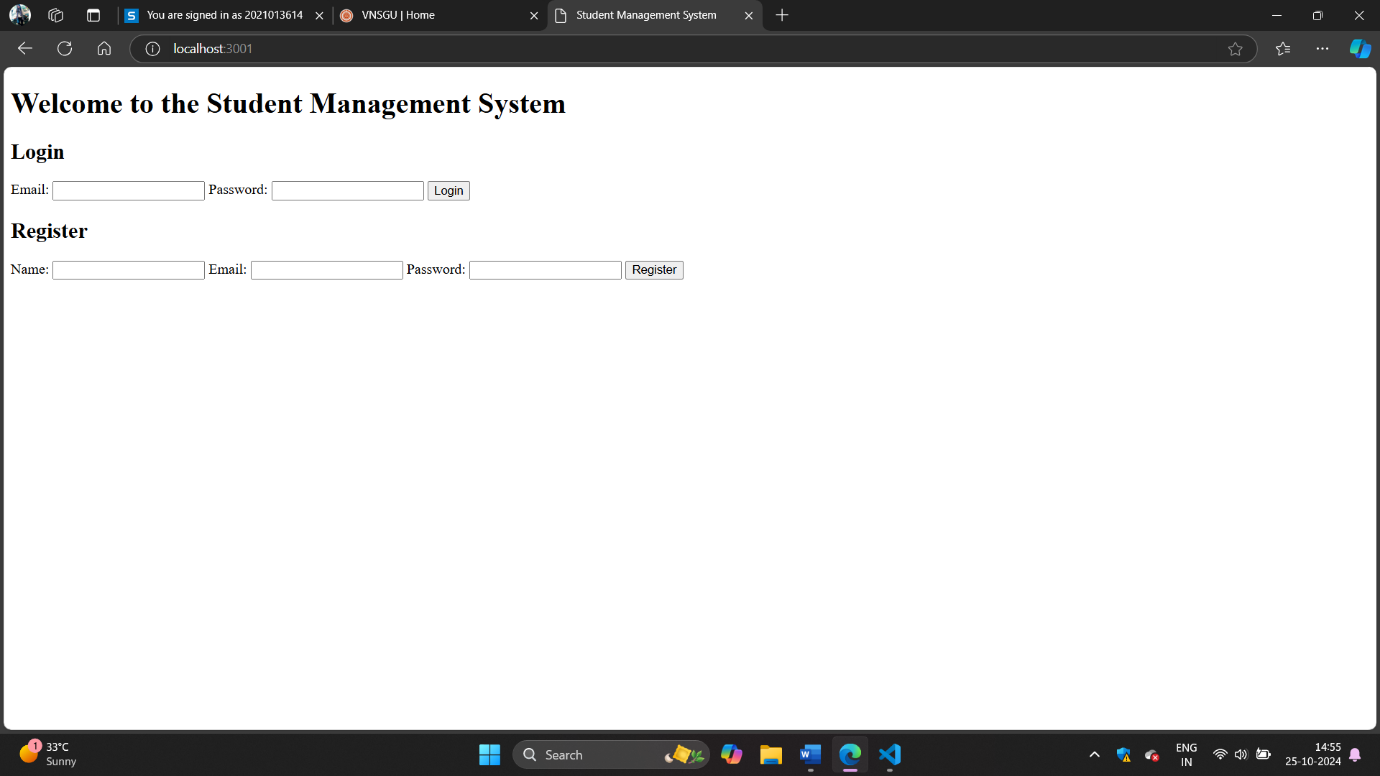
});

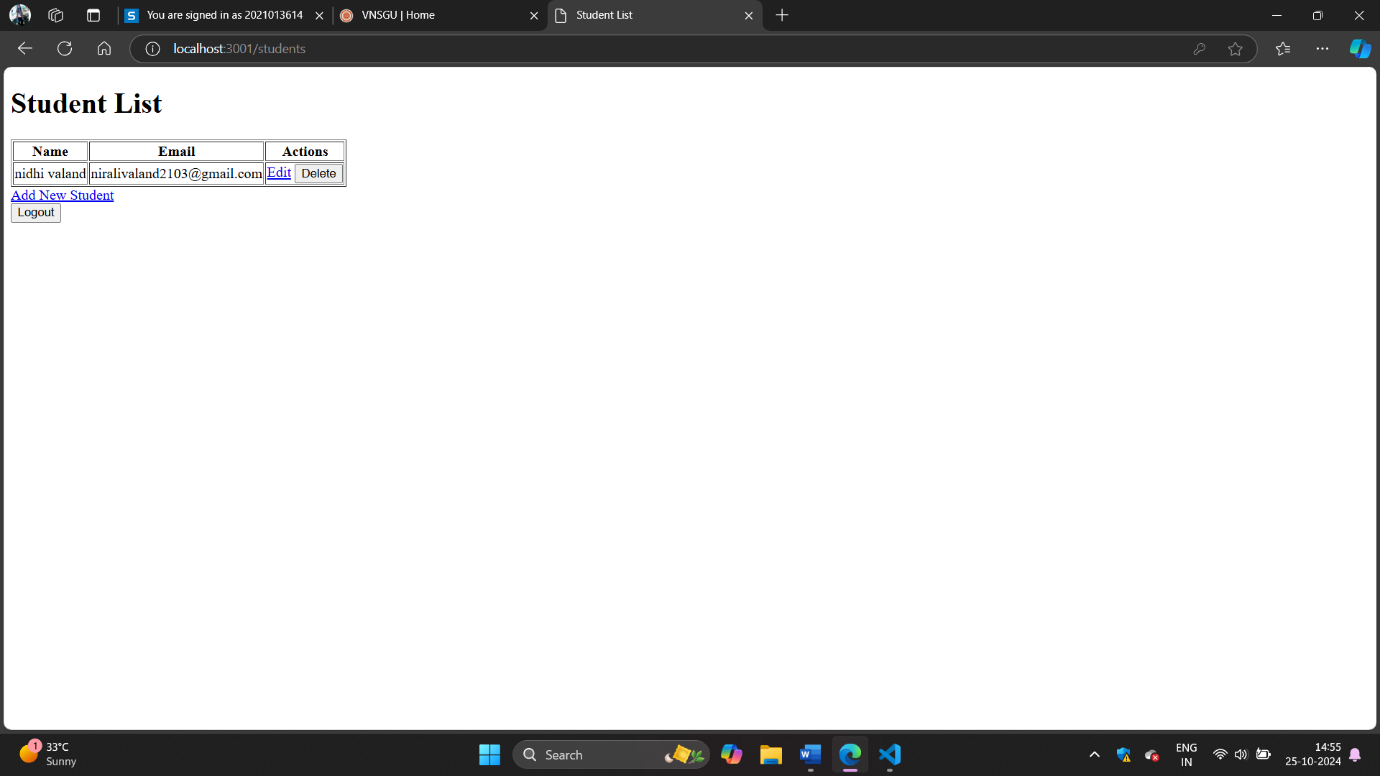
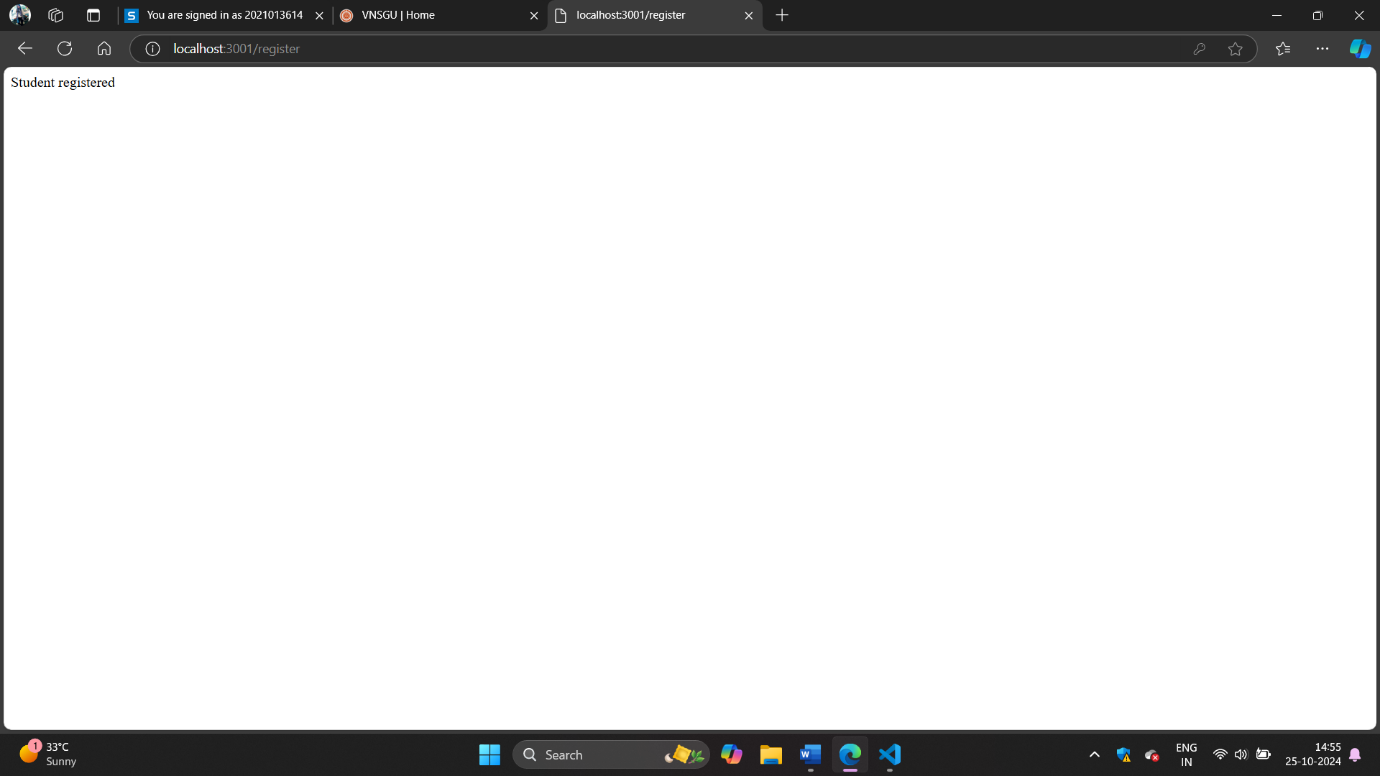
// Start server

app.listen(PORT, () => {

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

});





Que-5:

Server.js

const express = require('express');

const mongoose = require('mongoose');

const jwt = require('jsonwebtoken');

const bcrypt = require('bcryptjs');

const cors = require('cors');

const path = require('path');

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

const methodOverride = require('method-override');

const app = express();

const PORT = process.env.PORT || 8001;

// Middleware

app.use(express.json());

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

app.use(methodOverride('\_method')); // For PUT and DELETE methods

app.use(cors());

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

// Set views directory

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

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

// Session setup

app.use(session({

    secret: 'your\_secret\_key',

    resave: false,

    saveUninitialized: true,

}));

// Connect to MongoDB

mongoose.connect('mongodb://localhost:27017/employeeDB', { useNewUrlParser: true, useUnifiedTopology: true })

    .then(() => console.log('MongoDB connected'))

    .catch(err => console.error(err));

// Employee Schema

const employeeSchema = new mongoose.Schema({

    name: String,

    email: { type: String, unique: true },

    password: String

});

const Employee = mongoose.model('Employee', employeeSchema);

// Middleware for JWT verification

const authenticateJWT = (req, res, next) => {

    const token = req.session.token;

    if (!token) return res.redirect('/login'); // Redirect to login if not logged in

    jwt.verify(token, 'your\_jwt\_secret', (err, user) => {

        if (err) return res.redirect('/login'); // Redirect if token is invalid

        req.user = user;

        next();

    });

};

// Routes

// Registration Page

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

    res.render('register');

});

// Register new employee

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

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

    const newEmployee = new Employee({

        name: req.body.name,

        email: req.body.email,

        password: hashedPassword

    });

    try {

        await newEmployee.save();

        res.redirect('/login'); // Redirect to login page after registration

    } catch (error) {

        res.status(400).send('Error registering employee');

    }

});

// Login Page

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

    res.render('login');

});

// Login employee

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

    const employee = await Employee.findOne({ email: req.body.email });

    if (employee && (await bcrypt.compare(req.body.password, employee.password))) {

        const token = jwt.sign({ email: employee.email }, 'your\_jwt\_secret', { expiresIn: '1h' });

        req.session.token = token;

        res.redirect('/employees'); // Redirect to the employee list page after login

    } else {

        res.status(403).send('Invalid credentials');

    }

});

// Logout

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

    req.session.destroy(err => {

        if (err) return res.status(500).send('Could not log out');

        res.redirect('/login');

    });

});

// View all employees (Protected route)

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

    try {

        const employees = await Employee.find();

        res.render('employeeList', { employees });

    } catch (error) {

        res.status(500).send('Error retrieving employees');

    }

});

// Add employee form (Protected route)

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

    res.render('addEmployee');

});

// Handle adding a new employee

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

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

    const newEmployee = new Employee({

        name: req.body.name,

        email: req.body.email,

        password: hashedPassword

    });

    try {

        await newEmployee.save();

        res.redirect('/employees');

    } catch (error) {

        res.status(400).send('Error creating employee');

    }

});

// Update employee form (Protected route)

app.get('/employees/:id/edit', authenticateJWT, async (req, res) => {

    try {

        const employee = await Employee.findById(req.params.id);

        res.render('editEmployee', { employee });

    } catch (error) {

        res.status(400).send('Error retrieving employee');

    }

});

// Handle updating an employee

app.put('/employees/:id', authenticateJWT, async (req, res) => {

    const updateData = {

        name: req.body.name,

        email: req.body.email,

    };

    if (req.body.password) {

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

    }

    try {

        await Employee.findByIdAndUpdate(req.params.id, updateData);

        res.redirect('/employees');

    } catch (error) {

        res.status(400).send('Error updating employee');

    }

});

// Handle deleting an employee

app.delete('/employees/:id', authenticateJWT, async (req, res) => {

    try {

        await Employee.findByIdAndDelete(req.params.id);

        res.redirect('/employees');

    } catch (error) {

        res.status(400).send('Error deleting employee');

    }

});

// Start server

app.listen(PORT, () => {

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

});

Addemplyee.ejs

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Add Employee</title>

    <link rel="stylesheet" href="/styles.css">

</head>

<body>

    <div class="container">

        <h1>Add New Employee</h1>

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

            <div>

                <label for="name">Name:</label>

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

            </div>

            <div>

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

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

            </div>

            <div>

                <label for="password">Password:</label>

                <input type="password" id="password" name="password" required>

            </div>

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

        </form>

        <a href="/employees" class="btn">Back to Employee List</a>

    </div>

    <script src="/script.js"></script>

</body>

</html>

Editemployee.ejs

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Edit Employee</title>

    <link rel="stylesheet" href="/styles.css">

</head>

<body>

    <div class="container">

        <h1>Edit Employee</h1>

        <form action="/employees/<%= employee.\_id %>?\_method=PUT" method="POST">

            <div>

                <label for="name">Name:</label>

                <input type="text" id="name" name="name" value="<%= employee.name %>" required>

            </div>

            <div>

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

                <input type="email" id="email" name="email" value="<%= employee.email %>" required>

            </div>

            <div>

                <label for="password">New Password (leave blank to keep current):</label>

                <input type="password" id="password" name="password">

            </div>

            <button type="submit">Update Employee</button>

        </form>

        <a href="/employees" class="btn">Back to Employee List</a>

    </div>

    <script src="/script.js"></script>

</body>

</html>

Emplyeelist.ejs

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Employee List</title>

    <link rel="stylesheet" href="/styles.css"> <!-- Link to your CSS file -->

    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script> <!-- Optional jQuery -->

    <style>

        body {

            font-family: Arial, sans-serif;

            margin: 20px;

        }

        h1 {

            text-align: center;

        }

        table {

            width: 100%;

            border-collapse: collapse;

            margin-top: 20px;

        }

        table, th, td {

            border: 1px solid #ddd;

        }

        th, td {

            padding: 8px;

            text-align: left;

        }

        th {

            background-color: #f2f2f2;

        }

        tr:hover {

            background-color: #f1f1f1;

        }

        .action-buttons {

            display: flex;

            justify-content: space-between;

            margin: 10px 0;

        }

        a {

            text-decoration: none;

            color: white;

            padding: 10px 15px;

            border-radius: 5px;

        }

        .add-button {

            background-color: #4CAF50; /\* Green \*/

        }

        .logout-button {

            background-color: #f44336; /\* Red \*/

        }

        .edit-button {

            color: #007BFF; /\* Blue color for the Edit link \*/

            text-decoration: underline; /\* Underline for better visibility \*/

        }

        .edit-button:hover {

            color: #0056b3; /\* Darker blue on hover \*/

        }

        .delete-button {

            color: red; /\* Red for delete button \*/

            border: none;

            background: none;

            cursor: pointer;

        }

    </style>

</head>

<body>

    <h1>Employee List</h1>

    <div class="action-buttons">

        <a href="/employees/new" class="add-button">Add Employee</a>

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

            <button type="submit" class="logout-button">Logout</button>

        </form>

    </div>

    <table>

        <thead>

            <tr>

                <th>Name</th>

                <th>Email</th>

                <th>Actions</th>

            </tr>

        </thead>

        <tbody>

            <% employees.forEach(employee => { %>

                <tr>

                    <td><%= employee.name %></td>

                    <td><%= employee.email %></td>

                    <td>

                        <a href="/employees/<%= employee.\_id %>/edit" class="edit-button">Edit</a>

                        <form action="/employees/<%= employee.\_id %>?\_method=DELETE" method="POST" style="display:inline;">

                            <button type="submit" class="delete-button">Delete</button>

                        </form>

                    </td>

                </tr>

            <% }); %>

        </tbody>

    </table>

</body>

</html>

Login.ejs

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Login</title>

    <link rel="stylesheet" href="/styles.css">

</head>

<body>

    <div class="container">

        <h1>Login</h1>

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

            <div>

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

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

            </div>

            <div>

                <label for="password">Password:</label>

                <input type="password" id="password" name="password" required>

            </div>

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

        </form>

        <p>Don't have an account? <a href="/register">Register here</a></p>

    </div>

    <script src="/script.js"></script>

</body>

</html>

Register.ejs

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Register</title>

    <link rel="stylesheet" href="/styles.css">

</head>

<body>

    <div class="container">

        <h1>Register</h1>

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

            <div>

                <label for="name">Name:</label>

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

            </div>

            <div>

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

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

            </div>

            <div>

                <label for="password">Password:</label>

                <input type="password" id="password" name="password" required>

            </div>

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

        </form>

        <p>Already have an account? <a href="/login">Login here</a></p>

    </div>

    <script src="/script.js"></script>

</body>

</html>

Output:

