|  |  |  |  |
| --- | --- | --- | --- |
| **Course Name:** | **Web Programming Laboratory 116U40L501** | **Semester:** | **V** |
| **Date of Performance:** | **14 / 10 / 2024** | **Batch No:** | **B - 1** |
| **Faculty Name:** | **Prof. Madhura Pednekar** | **Roll No:** | **16014022050** |
| **Faculty Sign & Date:** |  | **Grade/Marks:** | **\_\_\_ / 25** |

**Experiment No: 6**

**Title:** Implementation of Node.JS, and Express. JS and MongoDB

|  |
| --- |
| **Aim and Objective of the Experiment:** |
| **Use Node.js with Express.js and MongoDB for building a full-stack JavaScript application.** |

|  |
| --- |
| **COs to be achieved:** |
| **CO5**: Develop back-end applications with Node.js, integrate databases, implement security, and deploy web projects. |

## 

|  |
| --- |
| **Problem Statement:** |
| 1. **Developing a RESTful API for User Management using Node.js, Express.js, and MongoDB.** |

|  |
| --- |
| **Problem Statement:** |
| Objective:  Design and implement a RESTful API for a basic user management system, allowing the creation, retrieval, updating, and deletion of user records. The API will be built using Node.js and Express.js for server-side logic, with MongoDB as the database for storing user data.  Context:  In many web applications, user management is a fundamental requirement. This involves creating user profiles, retrieving user information, updating user details, and removing users from the system. The goal of this experiment is to build a backend system that efficiently handles these operations using modern JavaScript technologies. |
| **Requirements**:   1. **API Endpoints**:    * **Create a User**: An endpoint that allows the creation of a new user profile by submitting user details such as name, email, and age.    * **Retrieve Users**: Endpoints that allow the retrieval of all users or a specific user by ID.    * **Update a User**: An endpoint that enables the modification of an existing user's details using their unique identifier.    * **Delete a User**: An endpoint to remove a user from the system by their unique identifier. 2. **Database Integration**:    * Utilize MongoDB to store user data, with Mongoose as an Object Data Modeling (ODM) library for MongoDB and Node.js.    * Implement schema validation for user data to ensure consistency and integrity. 3. **Error Handling**:    * Implement appropriate error handling to manage cases such as invalid user data, non-existent user IDs, and database connection failures. 4. **Scalability**:    * The system should be designed to handle a large number of user records and support future enhancements, such as adding authentication or more complex data structures.   **Expected Outcomes**:   * A functional RESTful API that performs all CRUD (Create, Read, Update, Delete) operations for user management. * A MongoDB database that accurately stores and retrieves user information. * Documentation for the API detailing how each endpoint should be used, including required fields and expected responses.   **Challenges**:   * Ensuring data consistency and integrity through schema validation. * Efficiently handling errors and edge cases, such as duplicate user entries or invalid input data. * Setting up a robust connection between the application and MongoDB, ensuring reliable data storage and retrieval.   **Use Case**:  This API could serve as the backend for a user management module in a larger web application, such as an e-commerce platform, social networking site, or content management system. It provides a foundational backend service that can be extended with additional features like authentication, user roles, and more complex data relationships in the future. |

|  |
| --- |
| **Code:** |
| **Backend:**   * + - * 1. config/database.js:   const mongoose = require('mongoose');  const connectDB = async () => {    try {      await mongoose.connect('mongodb://localhost:27017/finance\_app', {        useNewUrlParser: true,        useUnifiedTopology: true,      });      console.log('MongoDB connected successfully');    } catch (error) {      console.error('MongoDB connection error:', error);      process.exit(1);    }  };  module.exports = connectDB;   * + - * 1. models/user.js:   const mongoose = require('mongoose');  const userSchema = new mongoose.Schema({    name: {      type: String,      required: [true, 'Name is required'],      trim: true,      minlength: 2,      maxlength: 50    },    email: {      type: String,      required: [true, 'Email is required'],      unique: true,      trim: true,      lowercase: true,      match: [/^\w+([.-]?\w+)\*@\w+([.-]?\w+)\*(\.\w{2,3})+$/, 'Please enter a valid email']    },    age: {      type: Number,      required: [true, 'Age is required'],      min: [18, 'Must be at least 18 years old'],      max: [120, 'Age must be less than 120']    },    createdAt: {      type: Date,      default: Date.now    }  });  module.exports = mongoose.model('User', userSchema);   * + - * 1. middleware/errorHandler.js:   const errorHandler = (err, req, res, next) => {      console.error(err.stack);        if (err.name === 'ValidationError') {        return res.status(400).json({          success: false,          error: Object.values(err.errors).map(val => val.message)        });      }        if (err.code === 11000) {        return res.status(400).json({          success: false,          error: 'Duplicate field value entered'        });      }        res.status(500).json({        success: false,        error: 'Server Error'      });    };      module.exports = errorHandler;   * + - * 1. controllers/controller.js:   // controllers/controller.js  const User = require('../models/user');  // Make sure this path is correct  // Create new user  exports.createUser = async (req, res, next) => {    try {      const user = await User.create(req.body);      res.status(201).json({        success: true,        data: user      });    } catch (error) {      next(error);    }  };  // Get all users  exports.getUsers = async (req, res, next) => {    try {      const users = await User.find();      res.status(200).json({        success: true,        count: users.length,        data: users      });    } catch (error) {      next(error);    }  };  // Get single user  exports.getUser = async (req, res, next) => {    try {      const user = await User.findById(req.params.id);      if (!user) {        return res.status(404).json({          success: false,          error: 'User not found'        });      }      res.status(200).json({        success: true,        data: user      });    } catch (error) {      next(error);    }  };  // Update user  exports.updateUser = async (req, res, next) => {    try {      const user = await User.findByIdAndUpdate(        req.params.id,        req.body,        {          new: true,          runValidators: true        }      );      if (!user) {        return res.status(404).json({          success: false,          error: 'User not found'        });      }      res.status(200).json({        success: true,        data: user      });    } catch (error) {      next(error);    }  };  // Delete user  exports.deleteUser = async (req, res, next) => {    try {      const user = await User.findByIdAndDelete(req.params.id);      if (!user) {        return res.status(404).json({          success: false,          error: 'User not found'        });      }      res.status(200).json({        success: true,        data: {}      });    } catch (error) {      next(error);    }  };   * + - * 1. server.js   const express = require('express');  const connectDB = require('./config/database');  const errorHandler = require('./middleware/errorHandler');  const cors = require('cors');  const app = express();  // Connect to database  connectDB();  // Middleware  app.use(cors());  app.use(express.json());  // Import controller  const userController = require('./controllers/controller');  // Define routes  app.post('/api/users', userController.createUser);  app.get('/api/users', userController.getUsers);  app.get('/api/users/:id', userController.getUser);  app.put('/api/users/:id', userController.updateUser);  app.delete('/api/users/:id', userController.deleteUser);  // Error handler  app.use(errorHandler);  // Serve static files  app.use(express.static('public'));  const PORT = process.env.PORT || 3000;  app.listen(PORT, () => {    console.log(`Server running on port ${PORT}`);  });  **Frontend:**   * + - * 1. public/index.html   <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>User Management System</title>      <style>          \* {              margin: 0;              padding: 0;              box-sizing: border-box;              font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;          }          body {              background-color: #f0f2f5;              color: #333;              line-height: 1.6;          }          .container {              max-width: 1200px;              margin: 0 auto;              padding: 20px;          }          .header {              background-color: #ffffff;              padding: 20px;              border-radius: 8px;              box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);              margin-bottom: 20px;          }          .header h1 {              color: #1a73e8;              margin-bottom: 10px;          }          .form-section {              background-color: #ffffff;              padding: 20px;              border-radius: 8px;              box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);              margin-bottom: 20px;          }          .form-group {              margin-bottom: 15px;          }          .form-group label {              display: block;              margin-bottom: 5px;              font-weight: 500;          }          .form-group input {              width: 100%;              padding: 8px 12px;              border: 1px solid #ddd;              border-radius: 4px;              font-size: 14px;          }          .btn {              background-color: #1a73e8;              color: white;              padding: 10px 20px;              border: none;              border-radius: 4px;              cursor: pointer;              font-size: 14px;              transition: background-color 0.3s;          }          .btn:hover {              background-color: #1557b0;          }          .btn-danger {              background-color: #dc3545;          }          .btn-danger:hover {              background-color: #bb2d3b;          }          .users-table {              width: 100%;              background-color: #ffffff;              border-radius: 8px;              box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);              overflow: hidden;          }          .users-table th,          .users-table td {              padding: 12px;              text-align: left;              border-bottom: 1px solid #ddd;          }          .users-table th {              background-color: #f8f9fa;              font-weight: 600;          }          .users-table tr:last-child td {              border-bottom: none;          }          .users-table tr:hover {              background-color: #f8f9fa;          }          .action-buttons {              display: flex;              gap: 8px;          }          .message {              padding: 10px;              border-radius: 4px;              margin-bottom: 20px;          }          .success {              background-color: #d4edda;              color: #155724;              border: 1px solid #c3e6cb;          }          .error {              background-color: #f8d7da;              color: #721c24;              border: 1px solid #f5c6cb;          }          @media (max-width: 768px) {              .container {                  padding: 10px;              }              .users-table {                  display: block;                  overflow-x: auto;              }              .form-section {                  padding: 15px;              }          }      </style>  </head>  <body>      <div class="container">          <div class="header">              <h1>User Management System</h1>              <p>Add, update, and manage users in the system</p>          </div>          <div class="form-section">              <h2>Add User</h2>              <form id="userForm">                  <input type="hidden" id="userId">                  <div class="form-group">                      <label for="name">Name</label>                      <input type="text" id="name" required>                  </div>                  <div class="form-group">                      <label for="email">Email</label>                      <input type="email" id="email" required>                  </div>                  <div class="form-group">                      <label for="age">Age</label>                      <input type="number" id="age" required min="18" max="120">                  </div>                  <button type="submit" class="btn" id="submitBtn">Add User</button>                  <button type="button" class="btn" id="resetBtn" style="display: none;">Cancel Edit</button>              </form>          </div>          <div id="messageDiv"></div>          <table class="users-table">              <thead>                  <tr>                      <th>Name</th>                      <th>Email</th>                      <th>Age</th>                      <th>Actions</th>                  </tr>              </thead>              <tbody id="usersList"></tbody>          </table>      </div>      <script>          const API\_URL = 'http://localhost:3000/api/users';          let isEditing = false;          // Show message function          function showMessage(message, type) {              const messageDiv = document.getElementById('messageDiv');              messageDiv.className = `message ${type}`;              messageDiv.textContent = message;              setTimeout(() => {                  messageDiv.textContent = '';                  messageDiv.className = '';              }, 3000);          }          // Fetch all users          async function fetchUsers() {              try {                  const response = await fetch(API\_URL);                  const data = await response.json();                  const usersList = document.getElementById('usersList');                  usersList.innerHTML = '';                  data.data.forEach(user => {                      usersList.innerHTML += `                          <tr>                              <td>${user.name}</td>                              <td>${user.email}</td>                              <td>${user.age}</td>                              <td class="action-buttons">                                  <button class="btn" onclick="editUser('${user.\_id}')">Edit</button>                                  <button class="btn btn-danger" onclick="deleteUser('${user.\_id}')">Delete</button>                              </td>                          </tr>                      `;                  });              } catch (error) {                  showMessage('Error fetching users', 'error');              }          }          // Add/Update user          document.getElementById('userForm').addEventListener('submit', async (e) => {              e.preventDefault();              const userId = document.getElementById('userId').value;              const userData = {                  name: document.getElementById('name').value,                  email: document.getElementById('email').value,                  age: document.getElementById('age').value              };              try {                  const url = isEditing ? `${API\_URL}/${userId}` : API\_URL;                  const method = isEditing ? 'PUT' : 'POST';                  const response = await fetch(url, {                      method: method,                      headers: {                          'Content-Type': 'application/json'                      },                      body: JSON.stringify(userData)                  });                  const data = await response.json();                  if (data.success) {                      showMessage(`User ${isEditing ? 'updated' : 'added'} successfully`, 'success');                      resetForm();                      fetchUsers();                  } else {                      showMessage(data.error, 'error');                  }              } catch (error) {                  showMessage('Error processing request', 'error');              }          });          // Edit user          async function editUser(id) {              try {                  const response = await fetch(`${API\_URL}/${id}`);                  const data = await response.json();                    if (data.success) {                      const user = data.data;                      document.getElementById('userId').value = user.\_id;                      document.getElementById('name').value = user.name;                      document.getElementById('email').value = user.email;                      document.getElementById('age').value = user.age;                        document.getElementById('submitBtn').textContent = 'Update User';                      document.getElementById('resetBtn').style.display = 'inline-block';                      isEditing = true;                  }              } catch (error) {                  showMessage('Error fetching user details', 'error');              }          }          // Delete user          async function deleteUser(id) {              if (confirm('Are you sure you want to delete this user?')) {                  try {                      const response = await fetch(`${API\_URL}/${id}`, {                          method: 'DELETE'                      });                      const data = await response.json();                      if (data.success) {                          showMessage('User deleted successfully', 'success');                          fetchUsers();                      } else {                          showMessage(data.error, 'error');                      }                  } catch (error) {                      showMessage('Error deleting user', 'error');                  }              }          }          // Reset form          function resetForm() {              document.getElementById('userForm').reset();              document.getElementById('userId').value = '';              document.getElementById('submitBtn').textContent = 'Add User';              document.getElementById('resetBtn').style.display = 'none';              isEditing = false;          }          document.getElementById('resetBtn').addEventListener('click', resetForm);          // Initial load          fetchUsers();      </script>  </body>  </html> |

|  |
| --- |
| **Stepwise-Procedure / Algorithm:** |
| 1. Install Node.js: Ensure you have Node.js installed on your system. You can download it from [Node.js official website](https://nodejs.org/). 2. Create the Server Using Express.js 3. Define a MongoDB Schema and Model 4. Create Routes for CRUD Operations 5. Test the Application |

|  |
| --- |
| **Output** |
| * + - 1. npm init -y      * + - 1. npm install express mongoose cors      * + - 1. node server.js      * + - 1. Entering data |
| **Link for the Output:** |
| <https://drive.google.com/file/d/1975j4zZ8xOVuZtuWlRNMdO6ZJ8UanjR6/view?usp=sharing> |

|  |
| --- |
| **Post Lab Subjective/Objective type Questions:** |
| Students need to mention and explain with syntax the tags used for problem statement they have selected.  const: Defines constants in JavaScript.  const mongoose = require('mongoose');  module.exports: Exports modules/functions to be used in other files.  module.exports = connectDB;  async / await: Handles asynchronous operations.  const connectDB = async () => { /\*...\*/ }  mongoose.connect(): Connects to MongoDB database.  mongoose.connect('mongodb://localhost:27017/finance\_app', { /\* options \*/ });  try...catch: Error handling.  try { /\*...\*/ } catch (error) { /\*...\*/ }  mongoose.Schema: Defines a schema in Mongoose.  const userSchema = new mongoose.Schema({ /\* schema definition \*/ });  mongoose.model(): Creates a model based on a schema.  module.exports = mongoose.model('User', userSchema);  Object.values(): Retrieves the values of an object.  Object.values(err.errors).map(val => val.message);  res.status(): Sets the HTTP status code for a response.  res.status(201).json({ success: true, data: user });  app.use(): Middleware function for handling requests.  app.use(express.json());  app.get() / app.post() / app.put() / app.delete(): Define HTTP routes.  app.get('/api/users', userController.getUsers);  find() / findById() / findByIdAndUpdate() / findByIdAndDelete(): Mongoose methods for querying the database.  const users = await User.find(); |

|  |
| --- |
| **Conclusion:** |
| Successfully developed a full-stack JavaScript application using Node.js, Express.js, and MongoDB for efficient server-side processing and data management. This setup enabled seamless integration between the backend and frontend, ensuring a smooth user experience. |

|  |
| --- |
| **Signature of faculty in-charge with Date:** |