|  |  |  |  |
| --- | --- | --- | --- |
| **Course Name:** | **Web Programming Laboratory 116U40L501** | **Semester:** | **V** |
| **Date of Performance:** | **22 / 10 / 2024** | **Batch No.:** | **B - 1** |
| **Faculty Name:** | **Prof. Madhura Pednekar** | **Roll No.:** | **16014022050** |
| **Faculty Sign & Date:** |  | **Grade / Marks:** | **\_\_\_ / 25** |

**Experiment No: 7**

**Title:** Implementation of CRUD operation with Mongoose

|  |
| --- |
| **Aim and Objective of the Experiment:** |
| Develop a backend system to manage student and course records using CRUD (Create, Read, Update, Delete) operations with Mongoose, a MongoDB object modelling tool. |

|  |
| --- |
| **COs to be achieved:** |
| **CO4:** Apply master front-end frameworks like React, building modern web applications. |

|  |
| --- |
| **­­­­­Problem Statement:** |
| Implement the schema design using Mongoose models for **Students** and **Courses**. The schemas should define relationships between students and courses, such as a many-to-many relationship where students can enroll in multiple courses, and each course can have multiple students. |

|  |
| --- |
| **Use Cases:** |
| An admin wants to enroll a new student and assign them to specific courses.  A professor wants to view all the students enrolled in a course.  A student changes their major, and their enrolled courses need to be updated accordingly.  An outdated course needs to be removed from the system, and any student enrollments in that course should be updated. |

|  |
| --- |
| **Code:** |
| 1. **config/db.js**   const mongoose = require('mongoose');  const connectDB = async () => {      try {          await mongoose.connect(process.env.MONGODB\_URI);          console.log('MongoDB connected successfully');      } catch (error) {          console.error('MongoDB connection error:', error);          process.exit(1);      }  };  module.exports = connectDB;   1. **models/course.js**   const mongoose = require('mongoose');  const courseSchema = new mongoose.Schema({      name: {          type: String,          required: true      },      code: {          type: String,          required: true,          unique: true      },      description: String,      students: [{          type: mongoose.Schema.Types.ObjectId,          ref: 'Student'      }]  }, {      timestamps: true  });  module.exports = mongoose.model('Course', courseSchema);   1. **models/student.js**   const mongoose = require('mongoose');  const studentSchema = new mongoose.Schema({      name: {          type: String,          required: true      },      email: {          type: String,          required: true,          unique: true      },      major: {          type: String,          required: true      },      enrolledCourses: [{          type: mongoose.Schema.Types.ObjectId,          ref: 'Course'      }]  }, {      timestamps: true  });  module.exports = mongoose.model('Student', studentSchema);   1. **routes/courseRoutes.js**   const express = require('express');  const router = express.Router();  const Course = require('../models/Course');  const Student = require('../models/Student');  // Create a new course  router.post('/', async (req, res) => {      try {          const course = new Course(req.body);          await course.save();          res.status(201).json(course);      } catch (error) {          res.status(400).json({ message: error.message });      }  });  // Get all courses  router.get('/', async (req, res) => {      try {          const courses = await Course.find().populate('students');          res.json(courses);      } catch (error) {          res.status(500).json({ message: error.message });      }  });  // Get students in a course  router.get('/:id/students', async (req, res) => {      try {          const course = await Course.findById(req.params.id).populate('students');          if (!course) {              return res.status(404).json({ message: "Course not found" });          }          res.json(course.students);      } catch (error) {          res.status(500).json({ message: error.message });      }  });  // Update course  router.put('/:id', async (req, res) => {      try {          const course = await Course.findByIdAndUpdate(              req.params.id,              req.body,              { new: true }          );          res.json(course);      } catch (error) {          res.status(400).json({ message: error.message });      }  });  // Delete course  router.delete('/:id', async (req, res) => {      try {          const course = await Course.findById(req.params.id);          if (!course) {              return res.status(404).json({ message: "Course not found" });          }          // Remove course from all enrolled students          for (const studentId of course.students) {              await Student.findByIdAndUpdate(studentId, {                  $pull: { enrolledCourses: course.\_id }              });          }          await course.remove();          res.json({ message: "Course deleted successfully" });      } catch (error) {          res.status(500).json({ message: error.message });      }  });  module.exports = router;   1. **routes/studentRoutes.js**   const express = require('express');  const router = express.Router();  const Student = require('../models/Student');  const Course = require('../models/Course');  // Create a new student  router.post('/', async (req, res) => {      try {          const student = new Student(req.body);          await student.save();          res.status(201).json(student);      } catch (error) {          res.status(400).json({ message: error.message });      }  });  // Get all students  router.get('/', async (req, res) => {      try {          const students = await Student.find().populate('enrolledCourses');          res.json(students);      } catch (error) {          res.status(500).json({ message: error.message });      }  });  // Enroll student in a course  router.post('/:studentId/enroll/:courseId', async (req, res) => {      try {          const student = await Student.findById(req.params.studentId);          const course = await Course.findById(req.params.courseId);          if (!student || !course) {              return res.status(404).json({ message: "Student or course not found" });          }          student.enrolledCourses.push(course.\_id);          course.students.push(student.\_id);          await student.save();          await course.save();          res.json(student);      } catch (error) {          res.status(400).json({ message: error.message });      }  });  // Update student  router.put('/:id', async (req, res) => {      try {          const student = await Student.findByIdAndUpdate(              req.params.id,              req.body,              { new: true }          );          res.json(student);      } catch (error) {          res.status(400).json({ message: error.message });      }  });  // Delete student  router.delete('/:id', async (req, res) => {      try {          const student = await Student.findById(req.params.id);          if (!student) {              return res.status(404).json({ message: "Student not found" });          }          // Remove student from all enrolled courses          for (const courseId of student.enrolledCourses) {              await Course.findByIdAndUpdate(courseId, {                  $pull: { students: student.\_id }              });          }          await student.remove();          res.json({ message: "Student deleted successfully" });      } catch (error) {          res.status(500).json({ message: error.message });      }  });  module.exports = router;   1. **.env**   PORT=3000  MONGODB\_URI=mongodb://localhost:27017/student-management   1. **server.js**   const express = require('express');  const dotenv = require('dotenv');  const connectDB = require('./config/db');  const studentRoutes = require('./routes/studentRoutes');  const courseRoutes = require('./routes/courseRoutes');  dotenv.config();  const app = express();  app.use(express.json());  // Connect to MongoDB  connectDB();  // Routes  app.use('/api/students', studentRoutes);  app.use('/api/courses', courseRoutes);  const PORT = process.env.PORT || 3000;  app.listen(PORT, () => {      console.log(`Server is running on port ${PORT}`);  }); |

|  |
| --- |
| **Stepwise-Procedure / Algorithm:** |
| 1. **Create a database in MongoDB**      1. **Starting application**      1. **CREATE Operation**  * **Creating course 1: Web Programming**      * **Creating course 2: Database Management Systems** * **Creating student 1:**  1. **READ Operation**  * **Get all courses**      * **Get all students**      1. **UPDATE Operation**  * **Updated description field in course table to ‘Updated Description Field’**      * **Updated name field in student table to ‘XYZ’**      1. **Enrolling student in course ‘Web Development’**      1. **DELETE Operation**  * **Deleting ‘Database Management Systems’ course from courses table**      * **Deleting student from students table** |

|  |
| --- |
| **Output** |
| 1. **Courses Database:**      1. **Students Database:**      1. **Updated courses database:**      1. **Updated students database:**      1. **Enroll student in course Web Development:**      1. **Courses table after deleting course ‘Database Management System’**      1. **Students table after deleting student** |

|  |
| --- |
| **Post Lab Subjective/Objective type Questions:** |
| **Students need to mention and explain with syntax used for problem statement they have selected.**  **Declaring Variables:**  const studentName = "John Doe"; // `const` is used to declare constant variables.  let courseCode = "CS101"; // `let` allows reassignment of variables.    **Defining Functions:**  function addStudent(name, major) {  // Function definition  }  const enrollStudent = (studentId, courseId) => {  // Arrow function syntax  };    **Creating an Object:**  const student = { name: "John Doe", major: "CS" }; // Key-value pairs define properties.    **Importing Modules:**  const express = require('express'); // Importing external modules.  const mongoose = require('mongoose');  **Defining Mongoose Schema:**  const StudentSchema = new mongoose.Schema({  name: String,  major: String  });    **HTTP Requests with Express:**  app.post('/students', (req, res) => {  // Handles POST request to create a new student.  }); |

|  |
| --- |
| **Conclusion:** |
| We have successfully understood the implementation of CRUD operation with Mongoose. |

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