

Practice 2 - MongoDB

Title: Student Management System Using MongoDB and MVC Architecture

Objective:

Learn how to design and build a Node.js application using the Model-View-Controller (MVC) architecture to manage student data stored in MongoDB. This task helps you understand how to separate concerns, structure backend logic clearly, and interact with a database using Mongoose.

Task Description:

Create a student management system using Node.js, Express.js, and MongoDB (with Mongoose). Define a Student model with properties like name, age, and course. Implement a controller to handle CRUD operations (create, read, update, delete) on student data. Set up routes to connect client requests to the appropriate controller methods. Use Mongoose to handle all database interactions. Organize your codebase into separate folders for models, controllers, and routes to follow MVC principles clearly.

Code Implementation (MVC Structure):

```
// Step 1: Initialize Node.js project
// npm init -y
// npm install express mongoose body-parser nodemon

// Folder Structure:
// ├── models
// │   ├── studentModel.js
// ├── controllers
// │   ├── studentController.js
// ├── routes
// │   ├── studentRoutes.js
// └── server.js

// models/studentModel.js
const mongoose = require('mongoose');
const studentSchema = new mongoose.Schema({
  name: { type: String, required: true },
  age: { type: Number, required: true },
  course: { type: String, required: true }
});
module.exports = mongoose.model('Student', studentSchema);

// controllers/studentController.js
const Student = require('../models/studentModel');

exports.getAllStudents = async (req, res) => {
  try {
    const students = await Student.find();
    res.status(200).json(students);
  } catch (err) {
    res.status(500).json({ message: err.message });
  }
};

exports.getStudentById = async (req, res) => {
  try {
    const student = await Student.findById(req.params.id);
    res.status(200).json(student);
  } catch (err) {
    res.status(500).json({ message: err.message });
  }
};
```

```

        res.status(404).json({ message: 'Student not found' });
    }
};

exports.createStudent = async (req, res) => {
    try {
        const student = new Student(req.body);
        const savedStudent = await student.save();
        res.status(201).json(savedStudent);
    } catch (err) {
        res.status(400).json({ message: err.message });
    }
};

exports.updateStudent = async (req, res) => {
    try {
        const updatedStudent = await Student.findByIdAndUpdate(req.params.id, req.body, { new: true });
        res.status(200).json(updatedStudent);
    } catch (err) {
        res.status(400).json({ message: err.message });
    }
};

exports.deleteStudent = async (req, res) => {
    try {
        const deletedStudent = await Student.findByIdAndDelete(req.params.id);
        res.status(200).json({ message: 'Student deleted', student: deletedStudent });
    } catch (err) {
        res.status(500).json({ message: err.message });
    }
};

// routes/studentRoutes.js
const express = require('express');
const router = express.Router();
const studentController = require('../controllers/studentController');

router.get('/', studentController.getAllStudents);
router.get('/:id', studentController.getStudentById);
router.post('/', studentController.createStudent);
router.put('/:id', studentController.updateStudent);
router.delete('/:id', studentController.deleteStudent);

module.exports = router;

// server.js
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
const studentRoutes = require('./routes/studentRoutes');

const app = express();
app.use(bodyParser.json());

mongoose.connect('mongodb://localhost:27017/studentDB', {
    useNewUrlParser: true,
    useUnifiedTopology: true
}).then(() => console.log('MongoDB Connected'))
    .catch(err => console.log(err));

app.use('/students', studentRoutes);

app.listen(3000, () => console.log('Server running on port 3000'));


```

Expected Output:

- GET /students → Retrieve all students
- GET /students/:id → Retrieve a student by ID
- POST /students → Add a new student
- PUT /students/:id → Update a student by ID
- DELETE /students/:id → Delete a student by ID


Sample Output Screenshots:

```
GET : http://localhost:3000/students Send 🔍

Body :  No body

Request GET Response 200
▶ HTTP/1.1 200 OK (6 headers)
1 ▼ [
2   {
3     "_id": "686f66da1801707c14d09e60",
4     "name": "Alice Johnson",
5     "age": 20,
6     "course": "Computer Science"
7   },
8   {
9     "_id": "686f66da1801707c14d09e61",
10    "name": "Bob Smith",
11    "age": 22,
12    "course": "Mechanical Engineering"
13  },
14  {
15    "_id": "686f66da1801707c14d09e62",
16    "name": "Charlie Lee",
17    "age": 19,
18    "course": "Business Administration"
19  }
20 ]
```

```
GET : http://localhost:3000/students/686f66da1801707c14d09e60 Send 🔍

Body :  No body


Request GET Response 200
▶ HTTP/1.1 200 OK (6 headers)
1 ▼ {
2   "_id": "686f66da1801707c14d09e60",
3   "name": "Alice Johnson",
4   "age": 20,
5   "course": "Computer Science"
6 }
```

```
POST : http://localhost:3000/students Send 🔍

Body : 1 ▼ {
2   "name": "David Miller",
3   "age": 21,
4   "course": "Electrical Engineering"
5 }

Request POST Response 201
▶ HTTP/1.1 201 Created (6 headers)
1 ▼ {
2   "name": "David Miller",
3   "age": 21,
4   "course": "Electrical Engineering",
5   "_id": "686f675ab50ac14a3b78ad91",
6   "__v": 0
7 }
```

```
DELETE : http://localhost:3000/students/686f66da1801707c14d09e61 Send 🔍

Body :  No body

Request DELETE Response 200
▶ HTTP/1.1 200 OK (6 headers)
1 ▼ {
2   "message": "Student deleted",
3   "student": {
4     "_id": "686f66da1801707c14d09e61",
5     "name": "Bob Smith",
6     "age": 22,
7     "course": "Mechanical Engineering"
8   }
9 }
```