Experiment 4.2

Aim

To design and implement a Student Management System using Node.js, Express, MongoDB, and the MVC architecture, performing CRUD operations on student data.

Introduction

This experiment demonstrates how to build a backend application using Node.js and Express.js following the Model-View-Controller (MVC) architecture. MongoDB is used as the database to store student data, and Mongoose is used to manage database interactions. The MVC pattern helps in separating concerns, organizing code, and maintaining scalability and readability.

Procedure

1. Initialize a Node.js project and install required packages (`express`, `mongoose`, `dotenv`, `body-parser`).
2. Set up MongoDB connection using Mongoose.
3. Create a `.env` file to store database connection URI and server port.
4. Create `models` folder and define a `Student` model with properties: name, age, and course.
5. Create `controllers` folder and implement CRUD operations for student data.
6. Create `routes` folder and define routes to connect client requests to controller methods.
7. Organize code into MVC structure and start the server.
8. Test the API endpoints using tools like Postman or browser.

Code

server.js

```
const express = require("express");
const mongoose = require("mongoose");
const dotenv = require("dotenv");
const bodyParser = require("body-parser");
const studentRoutes = require("./routes/studentRoutes");

dotenv.config();
const app = express();

app.use(bodyParser.json());
app.use("/api/students", studentRoutes);

mongoose.connect(process.env.MONGO_URI)
```

```
  .then(() => console.log("MongoDB Connected"))
  .catch(err => console.log("DB Connection Error:", err));

const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

models/Student.js

```
const mongoose = require("mongoose");

const studentSchema = new mongoose.Schema({
  name: { type: String, required: true, trim: true },
  age: { type: Number, required: true },
  course: { type: String, required: true }
});

module.exports = mongoose.model("Student", studentSchema);
```

controllers/studentController.js

```
const Student = require("../models/Student");

exports.createStudent = async (req, res) => {
  try {
    const { name, age, course } = req.body;
    const newStudent = new Student({ name, age, course });
    await newStudent.save();
    res.status(201).json({ message: "Student added successfully", student:
newStudent });
  } catch (err) {
    res.status(500).json({ message: "Error creating student", error:
err.message });
  }
};

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

```
exports.getStudentById = async (req, res) => {
  try {
    const student = await Student.findById(req.params.id);
    if (!student) return res.status(404).json({ message: "Student not found" });
    res.status(200).json(student);
  } catch (err) {
    res.status(500).json({ message: "Error fetching student", error:
err.message });
  }
};

exports.updateStudent = async (req, res) => {
  try {
    const { name, age, course } = req.body;
    const updatedStudent = await Student.findByIdAndUpdate(req.params.id, {
name, age, course }, { new: true });
    if (!updatedStudent) return res.status(404).json({ message: "Student not
found" });
    res.status(200).json({ message: "Student updated", student:
updatedStudent });
  } catch (err) {
    res.status(500).json({ message: "Error updating student", error:
err.message });
  }
};

exports.deleteStudent = async (req, res) => {
  try {
    const deletedStudent = await Student.findByIdAndDelete(req.params.id);
    if (!deletedStudent) return res.status(404).json({ message: "Student not
found" });
    res.status(200).json({ message: "Student deleted successfully" });
  } catch (err) {
    res.status(500).json({ message: "Error deleting student", error:
err.message });
  }
};
```

routes/studentRoutes.js

```
const express = require("express");
const router = express.Router();
const studentController = require("../controllers/studentController");

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

module.exports = router;
```

Output

After running `node server.js` and testing endpoints via Postman or browser, the expected outputs are:

- **POST /api/students**: Adds a new student.
- **GET /api/students**: Returns a list of all students.
- **GET /api/students/:id**: Returns details of a single student.
- **PUT /api/students/:id**: Updates a student record.
- **DELETE /api/students/:id**: Deletes a student record.

Example Response (POST):

```
{
  "message": "Student added successfully",
  "student": {
    "_id": "672a7f8f93cd91e441ceab12",
    "name": "Meenakshi",
    "age": 20,
    "course": "Computer Science"
  }
}
```

Learning Outcomes

- Understand the MVC architecture and its implementation in Node.js.
- Learn to interact with MongoDB using Mongoose.
- Develop CRUD operations for managing student data.
- Organize code for better maintainability and scalability.
- Test RESTful APIs using Postman or similar tools.