Student Management System Using MongoDB and MVC Architecture

Aim

To design and implement a Student Management System using MongoDB and MVC architecture to efficiently manage student records such as registration, updates, retrieval, and deletion.

Objectives

- To understand and apply the Model-View-Controller (MVC) design pattern.
- To integrate MongoDB as a NoSQL database for student information management.
- To develop CRUD (Create, Read, Update, Delete) operations for student records.
- To build a scalable and modular application that separates business logic, user interface, and data handling.

Theory

MVC (Model-View-Controller) is a software design pattern that separates an application into three interconnected components:

- Model: Manages the data, logic, and rules of the application. In this case, it interacts with MongoDB.
- View: Represents the UI that displays the data to the user.
- Controller: Handles input from the user, updates the model, and refreshes the view.

MongoDB is a NoSQL database that stores data in the form of documents (BSON/JSON-like structure). It is highly flexible, scalable, and well-suited for handling large amounts of unstructured data. Combining MVC with MongoDB allows clear separation of concerns while ensuring efficient storage and retrieval of student records.

Procedure

1. Set up MongoDB and create a database named 'studentDB'.
2. Define a student schema/model with fields such as Roll Number, Name, Course, and Year.
3. Implement the Controller to handle CRUD operations:
   - Add new student records
   - Display all student records
   - Update student details
   - Delete student records
4. Design Views for interacting with the system (can be console-based or web-based).
5. Connect Model, View, and Controller to form a complete MVC-based system.

Code (Example in Node.js with Express and MongoDB)

```
// model/student.js
const mongoose = require('mongoose');
const studentSchema = new mongoose.Schema({
  rollNo: Number,
  name: String,
  course: String,
  year: Number
});
module.exports = mongoose.model('Student', studentSchema);

// controller/studentController.js
const Student = require('../model/student');

exports.addStudent = async (req, res) => {
  const student = new Student(req.body);
  await student.save();
  res.send('Student Added Successfully');
};

exports.getStudents = async (req, res) => {
  const students = await Student.find();
  res.json(students);
};

exports.updateStudent = async (req, res) => {
  await Student.updateOne({ rollNo: req.params.rollNo }, req.body);
  res.send('Student Updated Successfully');
};

exports.deleteStudent = async (req, res) => {
  await Student.deleteOne({ rollNo: req.params.rollNo });
  res.send('Student Deleted Successfully');
};

// app.js
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
const studentController = require('./controller/studentController');

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

mongoose.connect('mongodb://localhost:27017/studentDB');
```

```
app.post('/student', studentController.addStudent);
app.get('/students', studentController.getStudents);
app.put('/student/:rollNo', studentController.updateStudent);
app.delete('/student/:rollNo', studentController.deleteStudent);

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

Output

1. Student record added successfully.
2. Display of all student records in JSON format.
3. Student details updated successfully.
4. Student record deleted successfully.

Learning Outcomes

1. Understood the MVC design pattern and its implementation in real-world applications.
2. Gained practical knowledge of MongoDB integration with Node.js and Express.
3. Learned to perform CRUD operations effectively using a NoSQL database.