**EXPERIMENT 4.2**

## 🎯 Aim

To design and implement a Student Management System using MongoDB as the database and MVC (Model–View–Controller) architecture to efficiently manage student records such as adding, updating, deleting, and viewing student information.

## 📖 Theory

### 1. Student Management System (SMS)

A Student Management System is software that maintains student data such as name, roll number, course, grades, and contact information. It helps in:

* Maintaining records efficiently.
* Performing CRUD (Create, Read, Update, Delete) operations.
* Ensuring centralized access and easy retrieval of data.

### 2. MongoDB

* A NoSQL document-based database.
* Stores data in BSON (Binary JSON) format.
* Flexible schema: No need for predefined table structures.
* Suitable for applications needing scalability and performance.

### 3. MVC Architecture

* Model → Manages data and business logic.
* View → Handles the presentation (UI).
* Controller → Acts as a bridge between Model and View, processes user input.

In this project:

* Model → Defines the student schema in MongoDB.
* View → Provides frontend interface (HTML/EJS).
* Controller → Handles routes and operations (CRUD) with the model.

## 💻 Code (Sample Implementation)

We’ll use Node.js + Express + MongoDB for MVC implementation.

### Project Structure

student-management/

│-- app.js

│-- models/

│ └── student.js

│-- controllers/

│ └── studentController.js

│-- routes/

│ └── studentRoutes.js

│-- views/

│ ├── index.ejs

│ ├── addStudent.ejs

│ ├── editStudent.ejs

### 1. app.js (Entry point)

const express = require("express");

const mongoose = require("mongoose");

const bodyParser = require("body-parser");

const studentRoutes = require("./routes/studentRoutes");

const app = express();

// Middleware

app.use(bodyParser.urlencoded({ extended: true }));

app.set("view engine", "ejs");

// MongoDB connection

mongoose.connect("mongodb://127.0.0.1:27017/studentDB", {

useNewUrlParser: true,

useUnifiedTopology: true,

})

.then(() => console.log("MongoDB Connected ✅"))

.catch(err => console.log(err));

// Routes

app.use("/", studentRoutes);

// Server

app.listen(3000, () => console.log("Server running on http://localhost:3000"));

### 2. Model (models/student.js)

const mongoose = require("mongoose");

const studentSchema = new mongoose.Schema({

name: String,

rollNo: Number,

course: String,

email: String

});

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

### 3. Controller (controllers/studentController.js)

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

// Display all students

exports.getStudents = async (req, res) => {

const students = await Student.find();

res.render("index", { students });

};

// Add new student

exports.addStudentForm = (req, res) => {

res.render("addStudent");

};

exports.addStudent = async (req, res) => {

const { name, rollNo, course, email } = req.body;

const newStudent = new Student({ name, rollNo, course, email });

await newStudent.save();

res.redirect("/");

};

// Edit student

exports.editStudentForm = async (req, res) => {

const student = await Student.findById(req.params.id);

res.render("editStudent", { student });

};

exports.updateStudent = async (req, res) => {

await Student.findByIdAndUpdate(req.params.id, req.body);

res.redirect("/");

};

// Delete student

exports.deleteStudent = async (req, res) => {

await Student.findByIdAndDelete(req.params.id);

res.redirect("/");

};

### 4. Routes (routes/studentRoutes.js)

const express = require("express");

const router = express.Router();

const studentController = require("../controllers/studentController");

// Routes

router.get("/", studentController.getStudents);

router.get("/add", studentController.addStudentForm);

router.post("/add", studentController.addStudent);

router.get("/edit/:id", studentController.editStudentForm);

router.post("/edit/:id", studentController.updateStudent);

router.get("/delete/:id", studentController.deleteStudent);

module.exports = router;

### 5. Views (EJS Templates)

#### index.ejs

<!DOCTYPE html>

<html>

<head>

<title>Student Management</title>

</head>

<body>

<h1>Student Records</h1>

<a href="/add">➕ Add Student</a>

<ul>

<% students.forEach(student => { %>

<li>

<%= student.name %> - <%= student.course %>

| <a href="/edit/<%= student.\_id %>">✏️ Edit</a>

| <a href="/delete/<%= student.\_id %>">❌ Delete</a>

</li>

<% }) %>

</ul>

</body>

</html>

#### addStudent.ejs

<h2>Add Student</h2>

<form action="/add" method="POST">

<input type="text" name="name" placeholder="Name" required><br>

<input type="number" name="rollNo" placeholder="Roll No" required><br>

<input type="text" name="course" placeholder="Course" required><br>

<input type="email" name="email" placeholder="Email" required><br>

<button type="submit">Add</button>

</form>

#### editStudent.ejs

<h2>Edit Student</h2>

<form action="/edit/<%= student.\_id %>" method="POST">

<input type="text" name="name" value="<%= student.name %>" required><br>

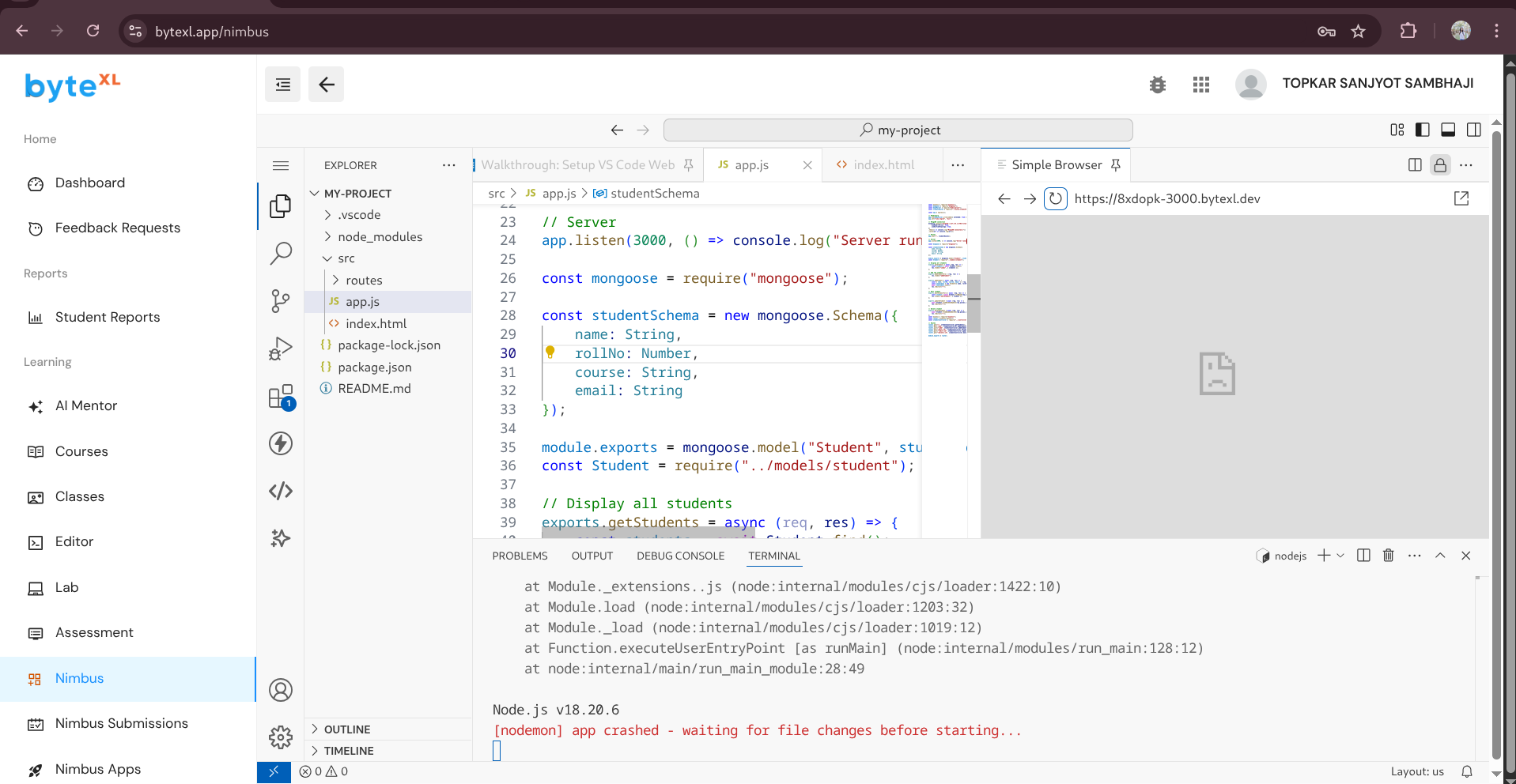
<input type="number" name="rollNo" value="<%= student.rollNo %>" required><br>

<input type="text" name="course" value="<%= student.course %>" required><br>

<input type="email" name="email" value="<%= student.email %>" required><br>

<button type="submit">Update</button>

</form>



## Learning Outcomes

1. Understanding of MVC Architecture
   * Separation of concerns improves code organization and scalability.
2. Practical use of MongoDB
   * Handling NoSQL database with schemas, queries, and CRUD operations.
3. Integration of Express.js with MongoDB
   * Learn to create routes, controllers, and views.
4. Full-Stack Development Skills
   * Backend (Node.js + MongoDB) + Frontend (EJS templates).
5. Problem-Solving Skills
   * Designing a real-world system that manages and manipulates student data.