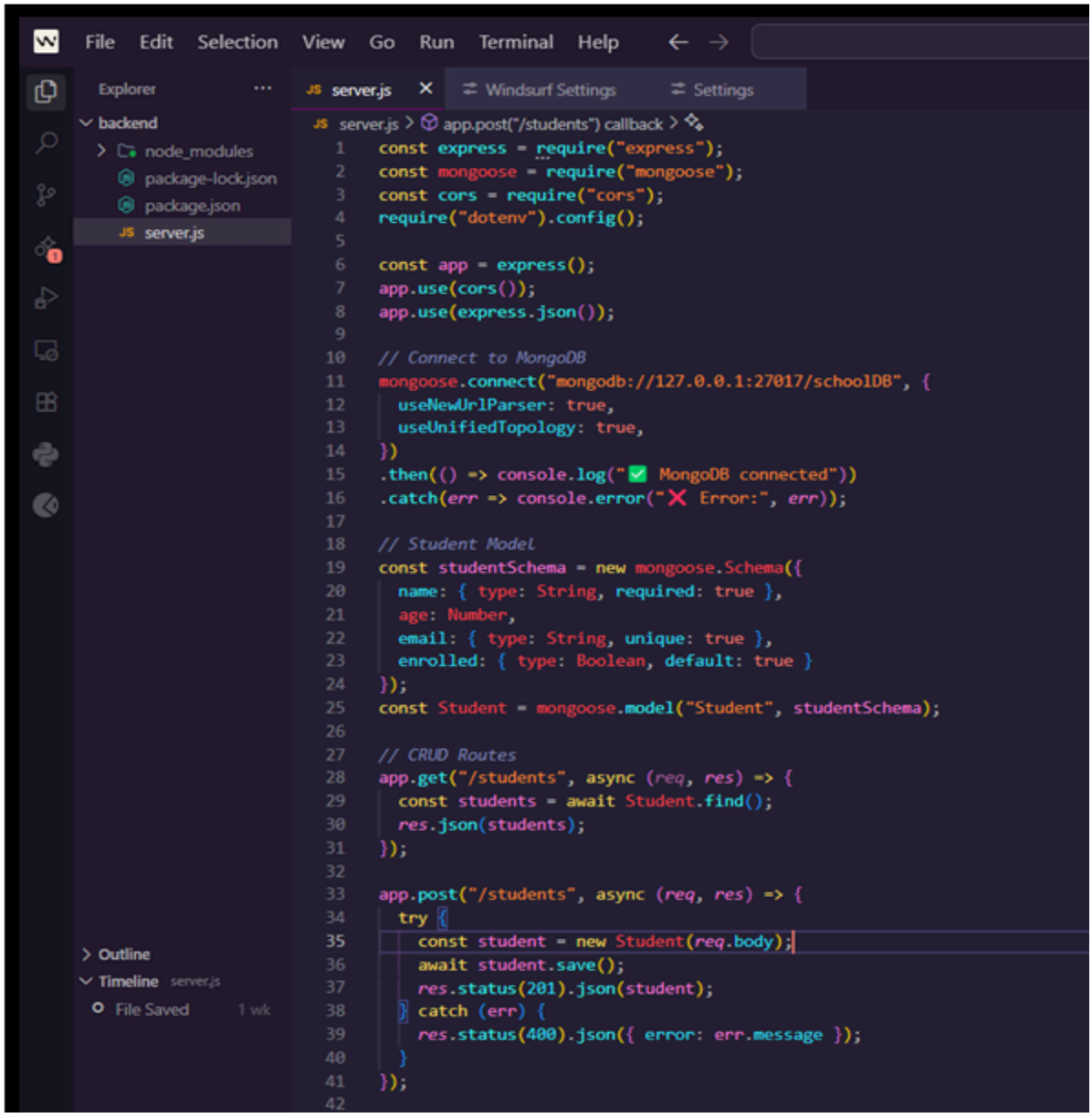
Experiment No. 5

Aim: Create secure, production-ready RESTful API

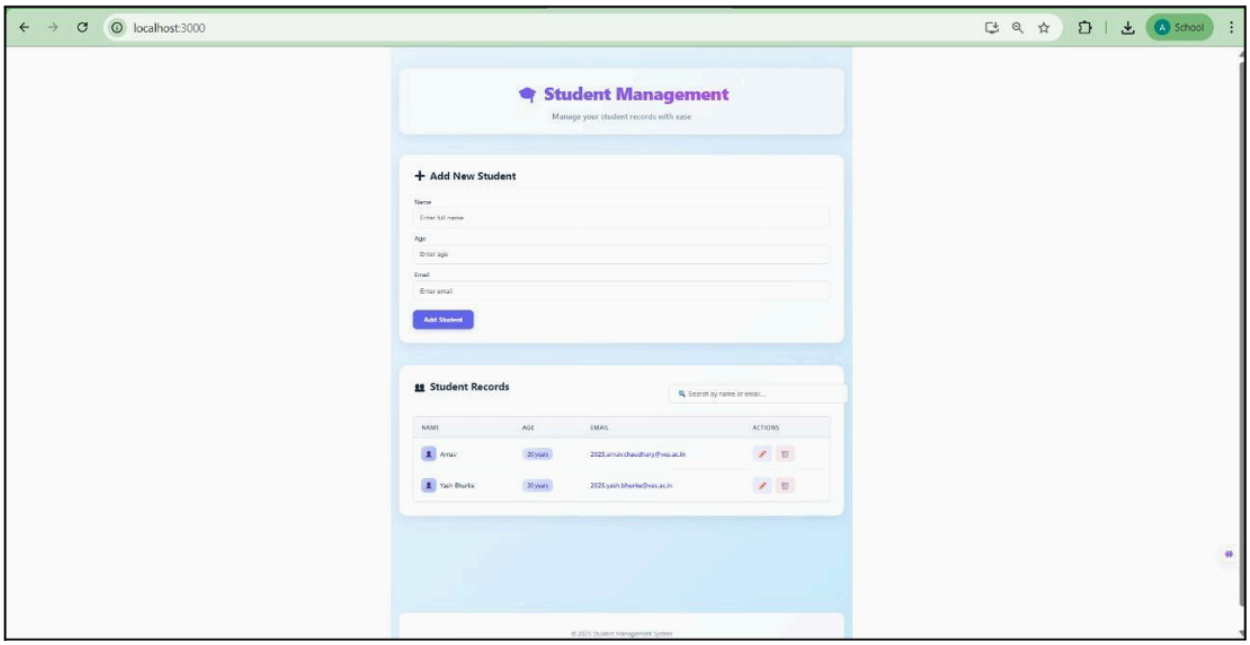
Code :

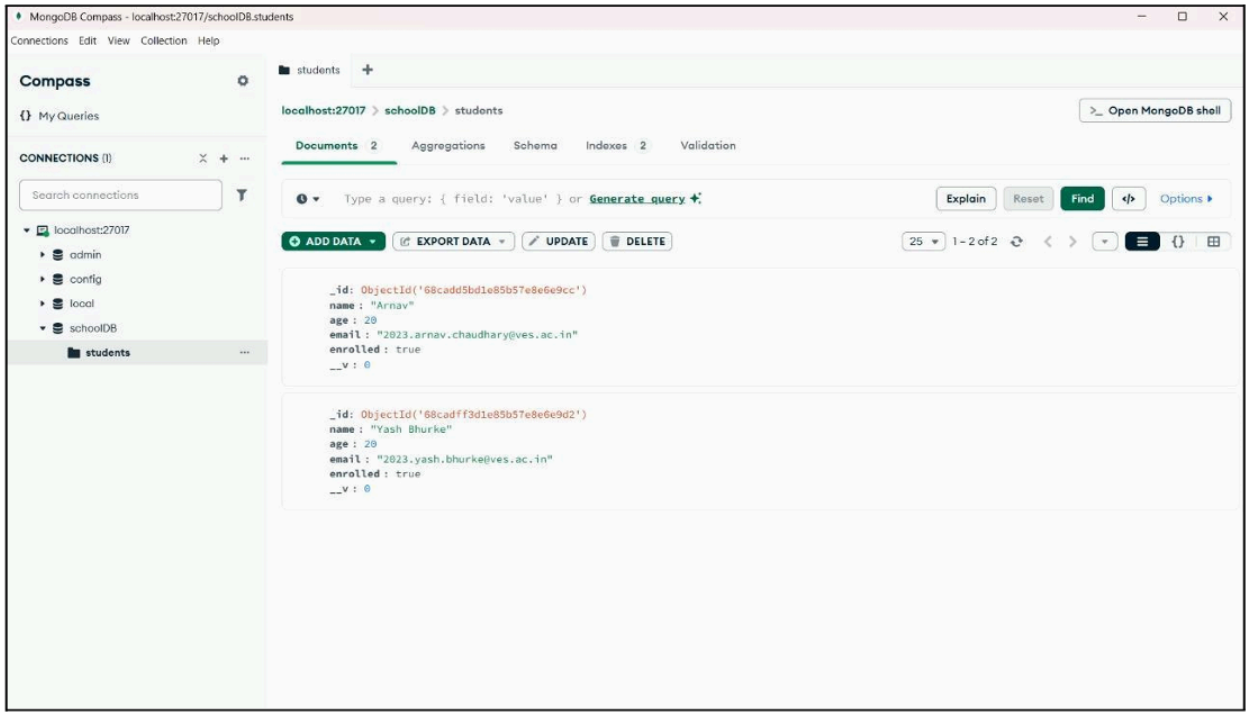
```
JS App.js M X

src > JS App.js > App > fetchStudents > useCallback() callback >

1  import React, { useState, useEffect, useCallback } from "react";
2  import axios from "axios";
3  import { ToastContainer, toast } from 'react-toastify';
4  import 'react-toastify/dist/ReactToastify.css';
5
6  function App() {
7    const [students, setStudents] = useState([]);
8    const [form, setForm] = useState({ name: "", age: "", email: "" });
9    const [search, setSearch] = useState("");
10   const [isLoading, setIsLoading] = useState(false);
11   const [editingId, setEditingId] = useState(null);
12   const [errors, setErrors] = useState({});
13
14   const API_URL = "http://localhost:5000/students";
15
16   const fetchStudents = useCallback(async () => {
17     try {
18       setIsLoading(true);
19       const res = await axios.get(API_URL);
20       setStudents(res.data);
21     } catch (error) {
22       toast.error('Failed to fetch students');
23       console.error('Error fetching students:', error);
24     } finally {
25       setIsLoading(false);
26     }
27   }, []);
28
29   useEffect(() => {
30     fetchStudents();
31   }, [fetchStudents]);
32
33   const validateForm = () => {
34     const newErrors = {};
35     if (!form.name.trim()) newErrors.name = 'Name is required';
36     if (!form.email.trim()) {
37       newErrors.email = 'Email is required';
38     } else if (!/\S+@\S+\.\S+/.test(form.email)) {
39       newErrors.email = 'Email is invalid';
40     }
```

```
41      if (!form.age) newErrors.age = 'Age is required';
42      else if (isNaN(form.age) || form.age < 1 || form.age > 120) {
43        newErrors.age = 'Please enter a valid age (1-120)';
44      }
45      setErrors(newErrors);
46      return Object.keys(newErrors).length === 0;
47    };
48
49    const handleSubmit = async (e) => {
50      e.preventDefault();
51      if (!validateForm()) return;
52
53      try {
54        setIsLoading(true);
55        if (editingId) {
```



Output :





Conclusion :

In conclusion, creating a **secure, production-ready RESTful API** requires following best practices such as implementing strong authentication and authorization, using HTTPS, validating and sanitizing inputs, handling errors safely, logging activity, rate limiting, and keeping dependencies up to date. Proper documentation, automated testing, and continuous monitoring further ensure reliability, scalability, and long-term maintainability in production.