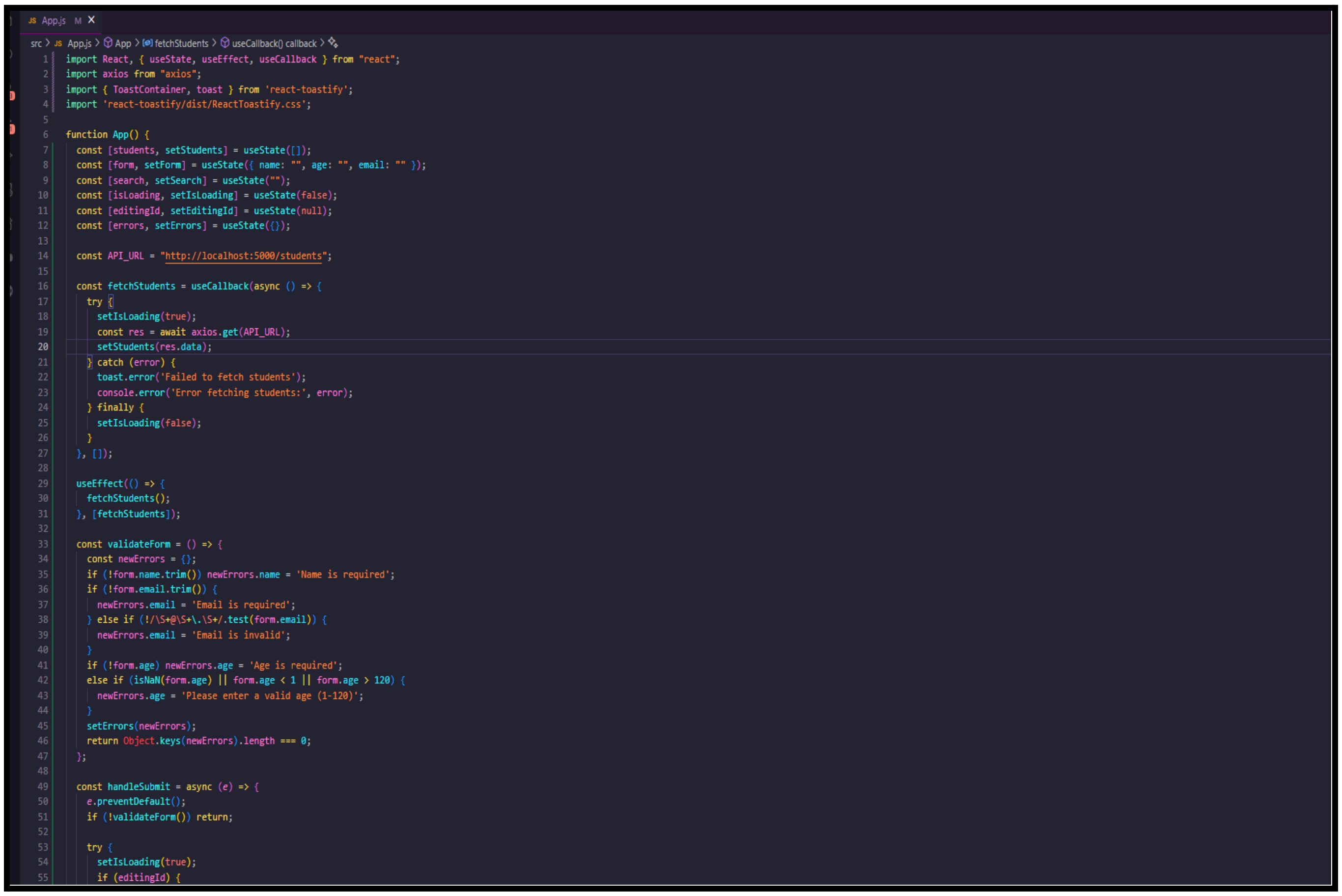
EXPERIMENT 5

Create Secure , production ready **RESTful APIS**

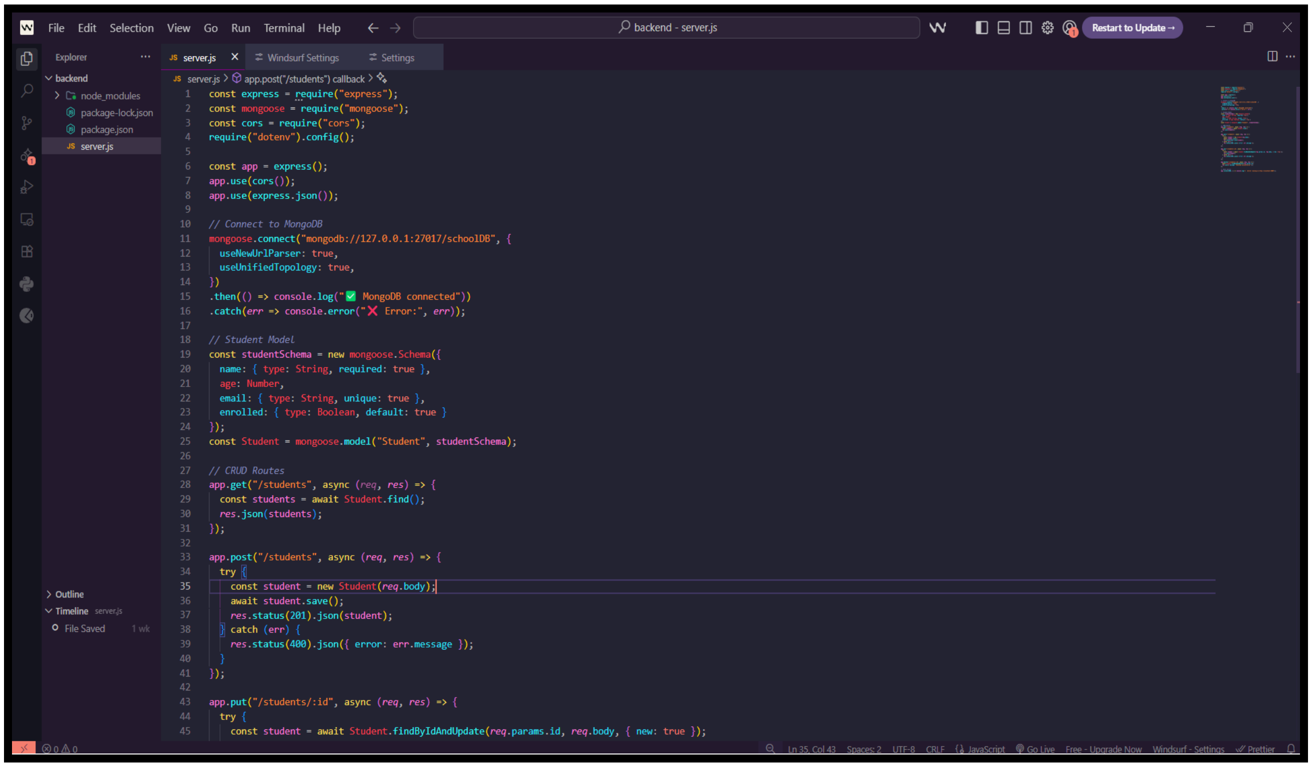
1. CODE

Figure 1.1: app.js File



App.js M ×
src > App.js > App > fetchStudents > useCallback() callback >

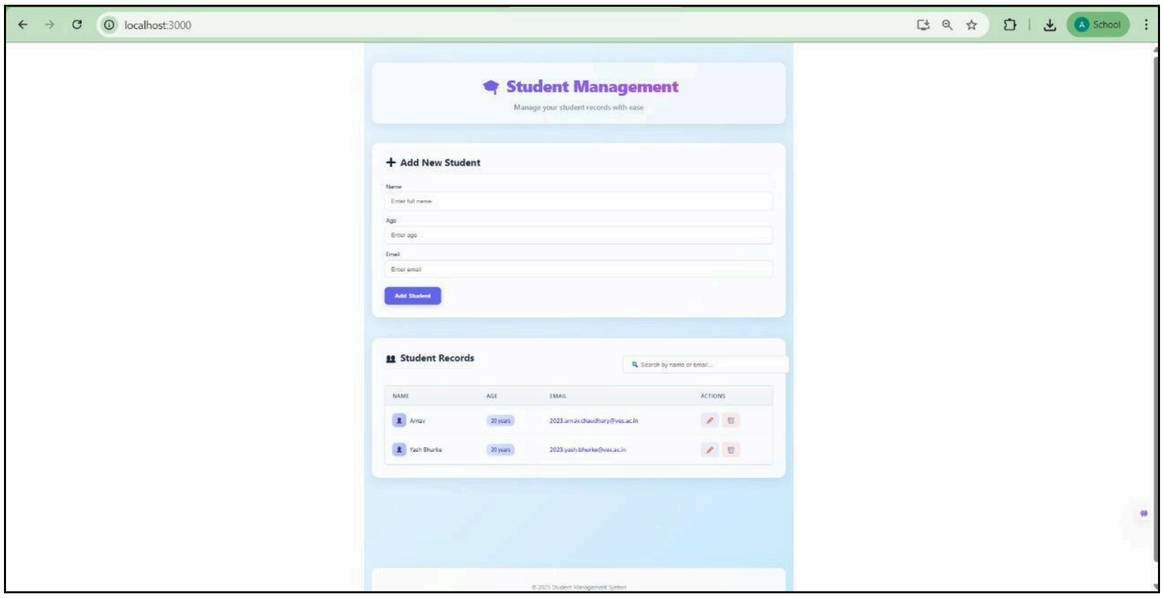
```javascript
import React, { useState, useEffect, useCallback } from "react";
import axios from "axios";
import { ToastContainer, toast } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';

function App() {
  const [students, setStudents] = useState([]);
  const [form, setForm] = useState({ name: "", age: "", email: "" });
  const [search, setSearch] = useState("");
  const [isLoading, setIsLoading] = useState(false);
  const [editingId, setEditingId] = useState(null);
  const [errors, setErrors] = useState({});

  const API_URL = "http://localhost:5000/students";

  const fetchStudents = useCallback(async () => {
    try {
      setIsLoading(true);
      const res = await axios.get(API_URL);
      setStudents(res.data);
    } catch (error) {
      toast.error('Failed to fetch students');
      console.error('Error fetching students:', error);
    } finally {
      setIsLoading(false);
    }
  }, []);

  useEffect(() => {
    fetchStudents();
  }, [fetchStudents]);

  const validateForm = () => {
    const newErrors = {};
    if (!form.name.trim()) newErrors.name = 'Name is required';
    if (!form.email.trim()) {
      newErrors.email = 'Email is required';
    } else if (!/\S+@\S+\.\S+/.test(form.email)) {
      newErrors.email = 'Email is invalid';
    }
    if (!form.age) newErrors.age = 'Age is required';
    else if (isNaN(form.age) || form.age < 1 || form.age > 120) {
      newErrors.age = 'Please enter a valid age (1-120)';
    }
    setErrors(newErrors);
    return Object.keys(newErrors).length === 0;
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    if (!validateForm()) return;

    try {
      setIsLoading(true);
      if (editingId) {
```



```
const express = require("express");
const mongoose = require("mongoose");
const cors = require("cors");
require("dotenv").config();

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

// Connect to MongoDB
mongoose.connect("mongodb://127.0.0.1:27017/schoolDB", {
  useNewUrlParser: true,
  useUnifiedTopology: true,
})
.then(() => console.log("✅ MongoDB connected"))
.catch(err => console.error("❌ Error:", err));

// Student Model
const studentSchema = new mongoose.Schema({
  name: { type: String, required: true },
  age: Number,
  email: { type: String, unique: true },
  enrolled: { type: Boolean, default: true }
});
const Student = mongoose.model("Student", studentSchema);

// CRUD Routes
app.get("/students", async (req, res) => {
  const students = await Student.find();
  res.json(students);
});

app.post("/students", async (req, res) => {
  try {
    const student = new Student(req.body);
    await student.save();
    res.status(201).json(student);
  } catch (err) {
    res.status(400).json({ error: err.message });
  }
});

app.put("/students/:id", async (req, res) => {
  try {
    const student = await Student.findByIdAndUpdate(req.params.id, req.body, { new: true });
```

Figure 1.2: server.js File

2. OUTPUT



Student Management

Manage your student records with ease

+ Add New Student

Name

Enter full name

Age

Enter age

Email

Enter email

Add Student

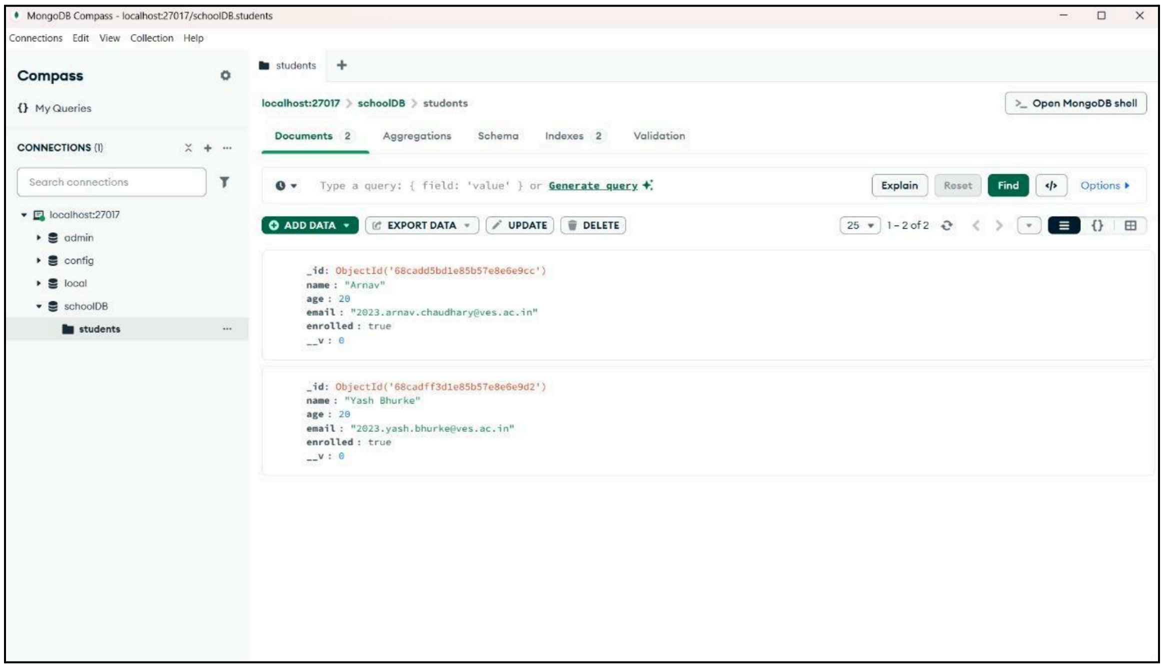
Student Records

Search by name or email...

NAME	AGE	EMAIL	ACTIONS
Arnav	20 years	2023.arnav.chaudhary@ves.ac.in	
Yash Bhurke	20 years	2023.yash.bhurke@ves.ac.in	

© 2025 Student Management System

Figure 2.1: Frontend Interface



```
_id: ObjectId('68cadd5bd1e85b57e8e6e9cc')
name : "Arnav"
age : 20
email : "2023.arnav.chaudhary@ves.ac.in"
enrolled : true
__v : 0
```

```
_id: ObjectId('68cadff3d1e85b57e8e6e9d2')
name : "Yash Bhurke"
age : 20
email : "2023.yash.bhurke@ves.ac.in"
enrolled : true
__v : 0
```

Figure 2.2: MongoDB Compass Database View