EAD-LAB-Experiment-8

Develop a backend application with REST API to perform CRUD operations on student data. (Use Postman Tool)

Backend application using a `src` directory that includes `models`, `routes`, `controllers`, and `config` folders. Here's how you can set it up:

Updated File Structure

```
lua
Copy code
student-api/
├── package.json
├── package-lock.json
└── src/
    ├── config/
    │   └── db.js
    ├── controllers/
    │   └── studentController.js
    ├── models/
    │   └── Student.js
    ├── routes/
    │   └── studentRoutes.js
    └── App.js
```

Step 1: Database Configuration

In `src/config/db.js`, set up the MongoDB connection.

```
javascript
Copy code
// src/config/db.js
const mongoose = require('mongoose');

// MongoDB connection
const connectDB = async () => {
    try {
        await mongoose.connect('mongodb://localhost:27017/studentDB', {
            useNewUrlParser: true,
            useUnifiedTopology: true,
        });
        console.log('MongoDB connected');
    } catch (error) {
        console.error('MongoDB connection failed:', error);
        process.exit(1); // Exit process with failure
    }
};

module.exports = connectDB;
```

Step 2: Create the Student Model

In `src/models/Student.js`, define the Mongoose model for student data.

```
javascript
Copy code
// src/models/Student.js
const mongoose = require('mongoose');

// Define the student schema
```

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

// Create the model
const Student = mongoose.model('Student', studentSchema);

module.exports = Student;
```

Step 3: Create the Controller

In `src/controllers/studentController.js`, implement the CRUD operations.

```
javascript
Copy code
// src/controllers/studentController.js
const Student = require('../models/Student');

// Create a new student
const createStudent = async (req, res) => {
    try {
        const student = new Student(req.body);
        await student.save();
        res.status(201).send(student);
    } catch (error) {
        res.status(400).send(error);
    }
};

// Read all students
const getAllStudents = async (req, res) => {
    try {
        const students = await Student.find();
        res.status(200).send(students);
    } catch (error) {
        res.status(500).send(error);
    }
};

// Read a single student by ID
const getStudentById = async (req, res) => {
    try {
        const student = await Student.findById(req.params.id);
        if (!student) {
            return res.status(404).send();
        }
        res.status(200).send(student);
    } catch (error) {
        res.status(500).send(error);
    }
};

// Update a student by ID
const updateStudent = async (req, res) => {
    try {
        const student = await Student.findByIdAndUpdate(req.params.id, req.body,
{ new: true, runValidators: true });
        if (!student) {
            return res.status(404).send();
        }
        res.status(200).send(student);
    } catch (error) {
        res.status(400).send(error);
    }
};

// Delete a student by ID
const deleteStudent = async (req, res) => {
    try {
        const student = await Student.findByIdAndDelete(req.params.id);
        if (!student) {
            return res.status(404).send();
        }
        res.status(200).send(student);
    } catch (error) {
        res.status(500).send(error);
    }
};

module.exports = {
    createStudent,
    getAllStudents,
    getStudentById,
    updateStudent,
    deleteStudent,
};
```

Step 4: Create the Routes

In `src/routes/studentRoutes.js`, define the routes and connect them to the controller functions.

```
javascript
Copy code
// src/routes/studentRoutes.js
const express = require('express');
const {
    createStudent,
    getAllStudents,
    getStudentById,
    updateStudent,
    deleteStudent,
} = require('../controllers/studentController');

const router = express.Router();

// Define the routes
router.post('/', createStudent);
router.get('/', getAllStudents);
router.get('/:id', getStudentById);
router.patch('/:id', updateStudent);
router.delete('/:id', deleteStudent);

module.exports = router;
```

Step 5: Set Up the Server

In `src/App.js`, set up the Express server and import the necessary modules.

```
javascript
Copy code
// src/ App.js
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const connectDB = require('./config/db');
const studentRoutes = require('./routes/studentRoutes');

const app = express();

// Middleware
app.use(cors());
app.use(bodyParser.json());

// Connect to MongoDB
connectDB();

// Routes
app.use('/students', studentRoutes);

// Start the server
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
    console.log(`Server is running on port ${PORT}`);
});
```

Step 6: Testing with Postman

1. **Start the Server:** Run the server using:

   ```
   Copy code
   node src/ App.js
   ```

2. **Using Postman:**
   - **Create a Student:**
     - **POST** to `http://localhost:3000/students`
     - **Body:** `{ "name": "John Doe", "age": 20, "email": "john@example.com" }`
   - **Read All Students:**
     - **GET** to `http://localhost:3000/students`
   - **Read a Student by ID:**
     - **GET** to `http://localhost:3000/students/{id}`
   - **Update a Student by ID:**
     - **PATCH** to `http://localhost:3000/students/{id}`
     - **Body:** `{ "age": 21 }`
   - **Delete a Student by ID:**
     - **DELETE** to `http://localhost:3000/students/{id}`