Full Stack Assignment - 2

1) Student Management API

Follow these steps.

Set Up a Node.js Project

* Initialize a Node.js project using npm init.
* Install Express and Mongoose.
* Create an app.js file to set up the server.

Connect to MongoDB

* Use Mongoose to establish a connection with MongoDB.
* Create an database.js file to store the database configuration.

Create a Simple Student Model (student.js)

* Define a student schema with fields id, name, age, course, email.

Create a student (POST request)

* Implement an endpoint (POST /students) to create and save a new student to the database.
* Validate required fields before saving.

Get All students (GET request)

* + Implement an endpoint (GET /students) to fetch all students from the database.

Get a student by ID (GET request)

* + Implement an endpoint (GET /students/:id) to fetch a specific student by their id.
  + Handle errors for invalid or non-existing id.

Update a student (PUT request)

* Implement an endpoint (PUT /students/:id) to update student’s details.
* Ensure partial updates work properly.

Delete a student (DELETE request)

* + Implement an endpoint (DELETE /students/:id) to remove a student from the database.

**Note:** Create control.js for all CRUD functions, and routes.js for setting up the routes

-> Solution: app.js

const express = require('express');

const connectDB = require('./database');

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

const app = express();

connectDB();  // connect to MongoDB

app.use(express.json());  // middleware to parse JSON request bodies

app.use('/api', studentRoutes);  // use student routes

app.get('/', (req, res) => {  // default route

    res.send('Server is running...');

});

const PORT = 3000;

app.listen(PORT, () => {

    console.log(`Server is running on port ${PORT}`);

});

-> database.js

const mongoose = require('mongoose');

const mongoURI = 'mongodb://localhost:27017/student'; // uri

const connectDB = async () => {

    try {

        await mongoose.connect(mongoURI);  // connects to mongodb using uri

        console.log('MongoDB connected successfully');

    } catch (error) {

        console.error('MongoDB connection error:', error);

        process.exit(1);

    }

};

module.exports = connectDB;

-> student.js

const mongoose = require('mongoose');

// create student schema

const studentSchema = new mongoose.Schema({

    id: {

        type: Number,

        required: true,

        unique: true

    },

    name: {

        type: String,

        required: true

    },

    age: {

        type: Number,

        required: true

    },

    course: {

        type: String,

        required: true

    },

    email: {

        type: String,

        required: true,

        unique: true

    }

});

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

module.exports = Student;

-> routes.js

const express = require('express');

const router = express.Router();

const studentController = require('./control');

// define routes

router.post('/students', studentController.createStudent);

router.get('/students', studentController.getAllStudents);

router.get('/students/:id', studentController.getStudentById);

router.put('/students/:id', studentController.updateStudent);

router.delete('/students/:id', studentController.deleteStudent);

module.exports = router;

-> control.js

const Student = require('./student');

// create a new student (POST)

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

    try {

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

        if (!id || !name || !age || !course || !email) {

            return res.status(400).json({ error: 'All fields are required' });

        }

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

        await newStudent.save();

        res.status(201).json({ message: 'Student created successfully', student: newStudent });

    } catch (error) {

        console.error('Error creating student:', error);

        res.status(500).json({ error: 'Internal Server Error' });

    }

};

// get all students (GET)

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

    try {

        const students = await Student.find();

        res.json(students);

    } catch (error) {

        console.error('Error fetching students:', error);

        res.status(500).json({ error: 'Internal Server Error' });

    }

};

// get a student by id (GET)

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

    try {

        const studentId = parseInt(req.params.id);

        if (isNaN(studentId)) {

            return res.status(400).json({ error: 'Invalid student ID' });

        }

        const student = await Student.findOne({ id: studentId });

        if (!student) {

            return res.status(404).json({ error: 'Student not found' });

        }

        res.json(student);

    } catch (error) {

        console.error('Error fetching student:', error);

        res.status(500).json({ error: 'Internal Server Error' });

    }

};

// update a student (PUT)

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

    try {

        const studentId = parseInt(req.params.id);

        if (isNaN(studentId)) {

            return res.status(400).json({ error: 'Invalid student ID' });

        }

        const updatedStudent = await Student.findOneAndUpdate(

            { id: studentId },

            { $set: req.body },

            { new: true, runValidators: true }

        );

        if (!updatedStudent) {

            return res.status(404).json({ error: 'Student not found' });

        }

        res.json({ message: 'Student updated successfully', student: updatedStudent });

    } catch (error) {

        console.error('Error updating student:', error);

        res.status(500).json({ error: 'Internal Server Error' });

    }

};

// delete a student (DELETE)

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

    try {

        const studentId = parseInt(req.params.id);

        if (isNaN(studentId)) {

            return res.status(400).json({ error: 'Invalid student ID' });

        }

        const result = await Student.deleteOne({ id: studentId });

        if (result.deletedCount === 0) {

            return res.status(404).json({ error: 'Student not found' });

        }

        res.json({ message: 'Record deleted successfully' });

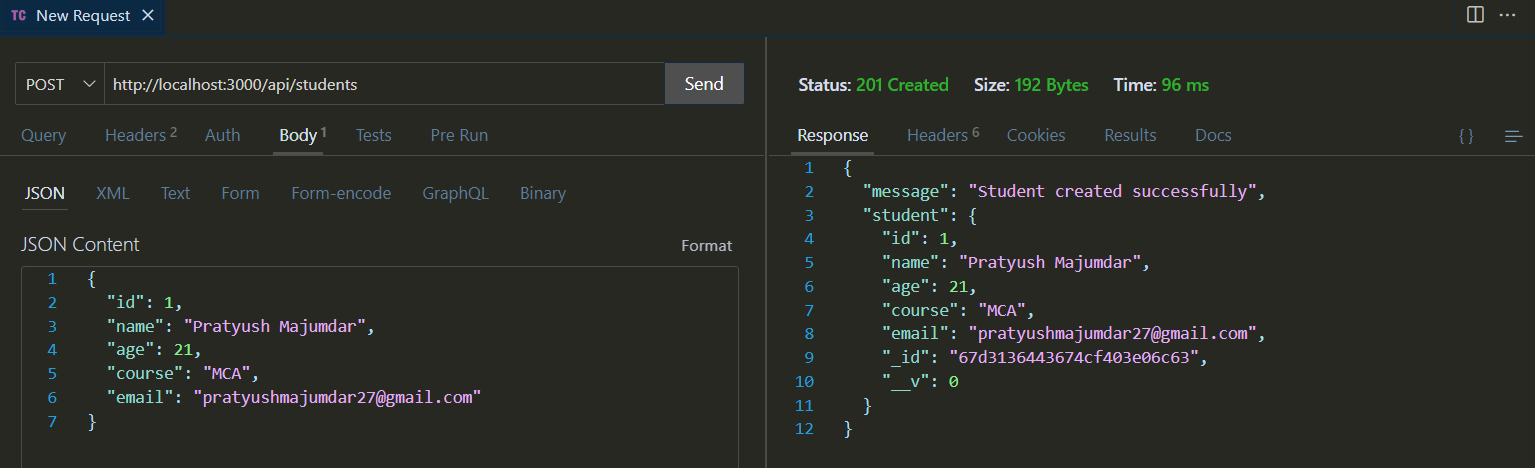
    } catch (error) {

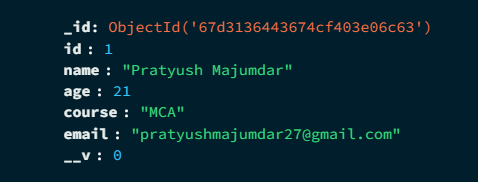
        console.error('Error deleting student:', error);

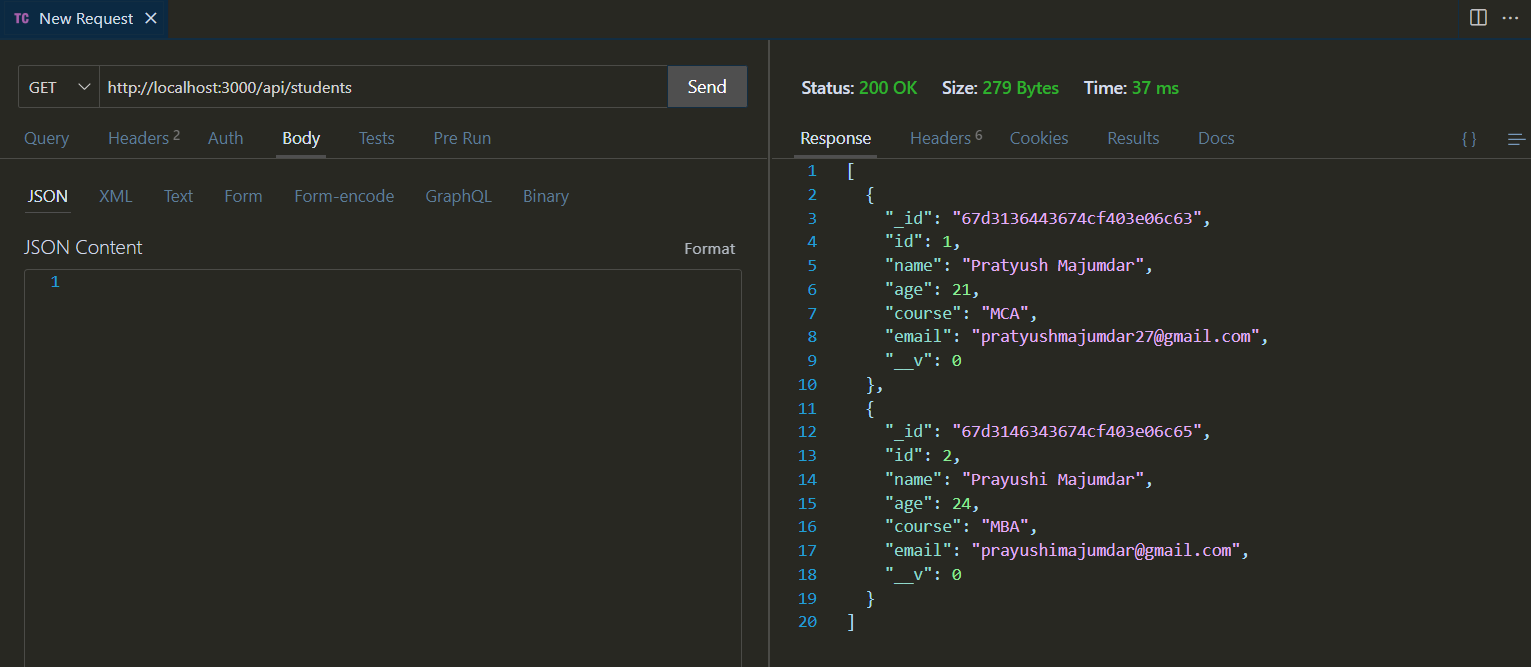
        res.status(500).json({ error: 'Internal Server Error' });

    }

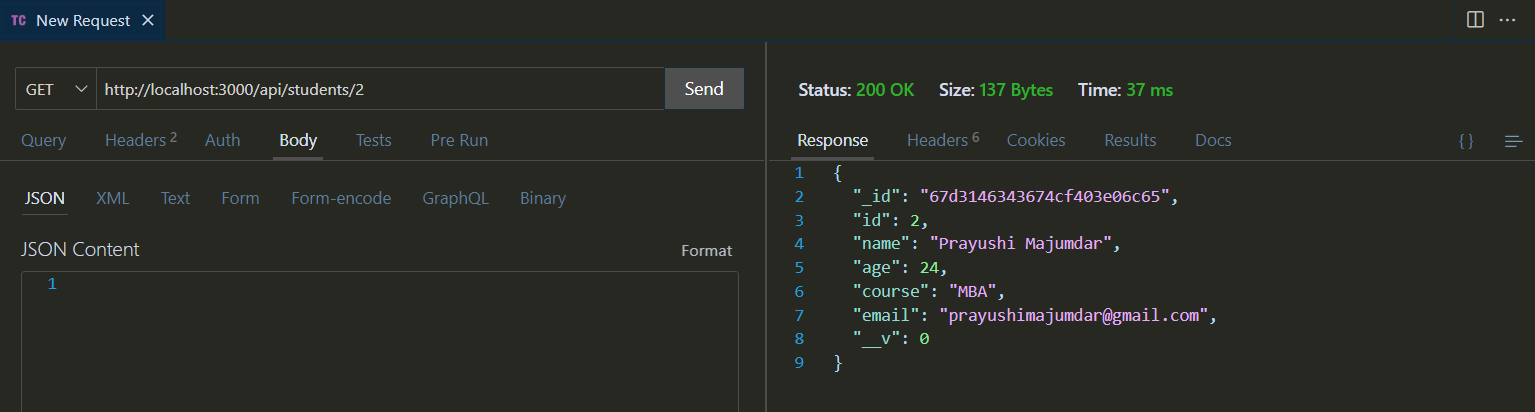
};

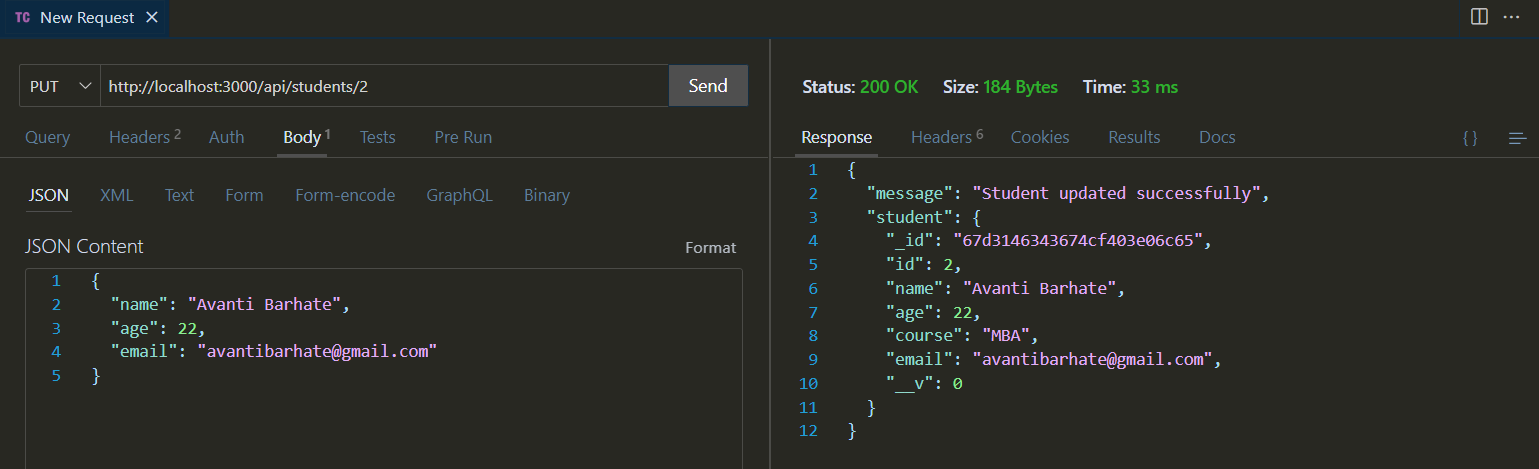
-> Output: 1) Add student (POST)



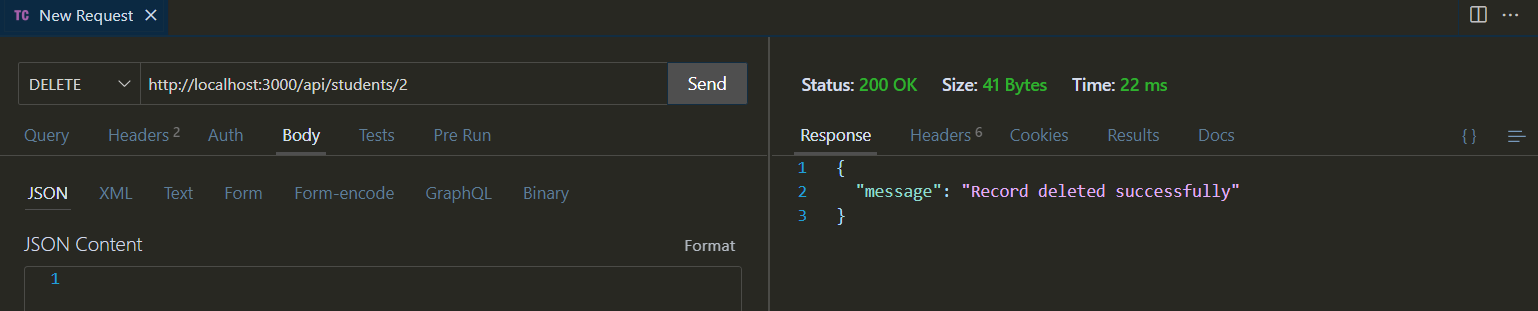
2) View all students (GET)

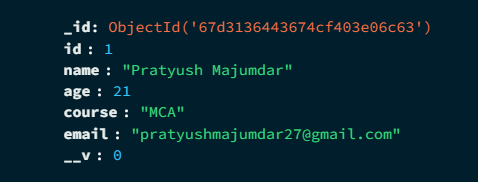


3) View student by id (GET)

4) Update student by id (PUT)



5) Delete student by id (DELETE)



2) Book Library API

Follow these steps.

Set Up a Node.js Project

* Initialize a Node.js project using npm init.
* Install Express and Mongoose.
* Create an app.js file to set up the server.

Connect to MongoDB

* Use Mongoose to establish a connection with MongoDB.
* Create an database.js file to store the database configuration.

Create a Simple book Model (book.js)

* Define a book schema with fields id, title, author, price, publishedYear.

Create a book (POST request)

* Implement an endpoint (POST /books) to create and save a new book to the database.
* Validate required fields before saving.

Get All books (GET request)

* + Implement an endpoint (GET /books) to fetch all books from the database.

Get a book by ID (GET request)

* + Implement an endpoint (GET /books/:id) to fetch a specific book by their id.
  + Handle errors for invalid or non-existing id.

Update a book (PUT request)

* Implement an endpoint (PUT /books/:id) to update book’s details.
* Ensure partial updates work properly.

Delete a book (DELETE request)

* + Implement an endpoint (DELETE /books/:id) to remove a book from the database.

**Note:** Create control.js for all CRUD functions, and routes.js for setting up the routes

-> Solution: app.js

const express = require('express');

const connectDB = require('./database');

const bookRoutes = require('./routes');

const app = express();

connectDB();  // connect to MongoDB

app.use(express.json());  // middleware to parse JSON request bodies

app.use('/api', bookRoutes);  // use student routes

app.get('/', (req, res)=> {  // default route

    res.send("Server is running");

})

const port = 3000;

app.listen(port, () => {

    console.log(`Server is running on port ${port}`);

})

-> database.js

const mongoose = require('mongoose');

const mongoURI = "mongodb://localhost:27017/library";  // uri

const connectDB = async () => {

    try {

        await mongoose.connect(mongoURI);  // connects to mongodb using uri

        console.log("MongoDB connected successfully");

    } catch(error) {

        console.error('MongoDB connection error: ', error);

        process.exit(1);

    }

};

module.exports = connectDB;

-> book.js

const mongoose = require('mongoose');

// create book schema

const bookSchema = new mongoose.Schema({

    id: {

        type: Number,

        required: true,

        unique: true

    },

    title: {

        type: String,

        required: true

    },

    author: {

        type: String,

        required: true

    },

    price: {

        type: Number,

        required: true,

    },

    publishedYear: {

        type: Number,

        required: true,

    },

});

const Book = mongoose.model('Book', bookSchema);

module.exports = Book;

-> routes.js

const express = require('express');

const router = express.Router();

const bookController = require('./control');

// define routes

router.post('/books', bookController.createBook);

router.get('/books', bookController.getAllBooks);

router.get('/books/:id', bookController.getBookById);

router.put('/books/:id', bookController.updateBookById);

router.delete('/books/:id', bookController.deleteBookById);

module.exports = router;

-> control.js

const Book = require('./book');

// create a new book (POST)

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

    try {

        const {id, title, author, price, publishedYear} = req.body;

        if (!id || !title || !author || !price || !publishedYear) {

            return res.status(400).json({error: 'All fields are required'});

        }

        const newBook = new Book({id, title, author, price, publishedYear})

        await newBook.save();

        res.status(201).json({ message: 'Book created successfully', book: newBook });

    } catch(error) {

        console.error('Error creating book', e);

        res.status(500).json({error: "Internal server error"});

    }

};

// get all books (GET)

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

    try {

        const books = await Book.find();

        res.json(books);

    } catch(error) {

        console.error('Error creating book', error);

        res.status(500).json({error: "Internal server error"});

    }

};

// get a book by id (GET)

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

    try {

        const bookId = parseInt(req.params.id);

        if(isNaN(bookId)) {

            return res.status(400).json({error: "Invalid Book ID"})

        }

        const book = await Book.findOne({id: bookId});

        if (!book) {

            return res.status(404).json({ error: 'Student not found' });

        }

        res.json(book);

    } catch(error) {

        console.error('Error creating book', error);

        res.status(500).json({error: "Internal server error"});

    }

};

// update a book by id (PUT)

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

    try {

        const bookId = parseInt(req.params.id);

        if(isNaN(bookId)) {

            return res.status(400).json({error: "Invalid Book ID"})

        }

        const updatedBook = await Book.findOneAndUpdate(

            {id: bookId},

            {$set: req.body},

            {new: true, runValidators: true}

        );

        if (!updatedBook) {

            return res.status(404).json({ error: 'Student not found' });

        }

        res.json({message: 'Book updated successfully', book: updatedBook});

    } catch(error) {

        console.error('Error creating book', error);

        res.status(500).json({error: "Internal server error"});

    }

};

// delete a book by id (DELETE)

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

    try {

        const bookId = parseInt(req.params.id);

        if(isNaN(bookId)) {

            return res.status(400).json({error: "Invalid Book ID"})

        }

        const result = await Book.deleteOne(

            {id: bookId}

        );

        if (result.deletedCount === 0) {

            return res.status(404).json({ error: 'Student not found' });

        }

        res.json({message: 'Book deleted successfully'});

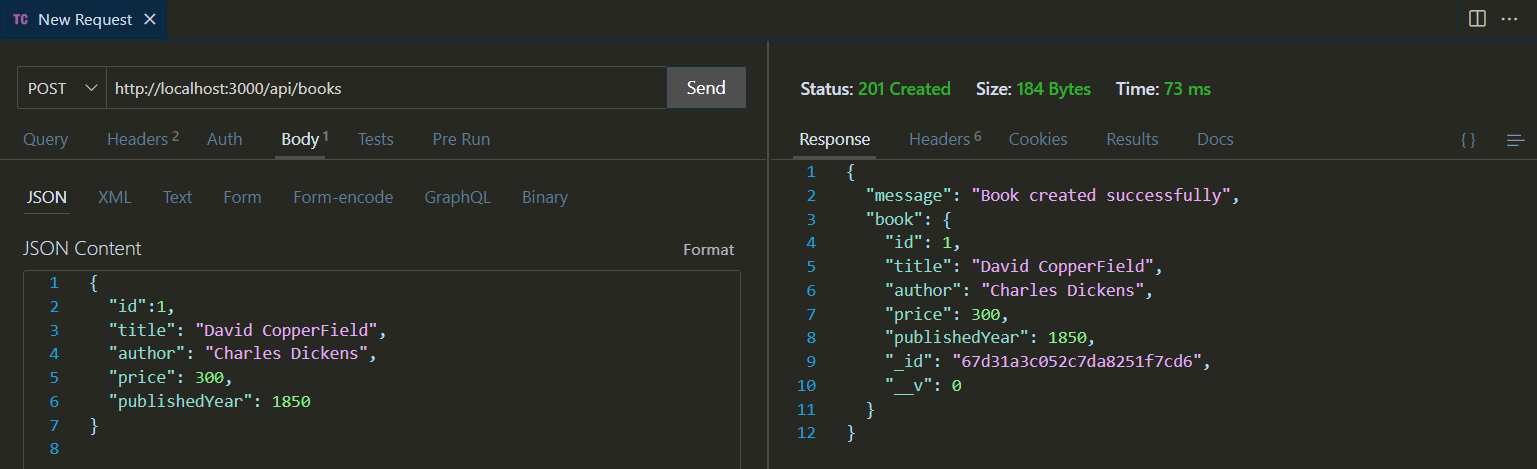
    } catch(error) {

        console.error('Error creating book', error);

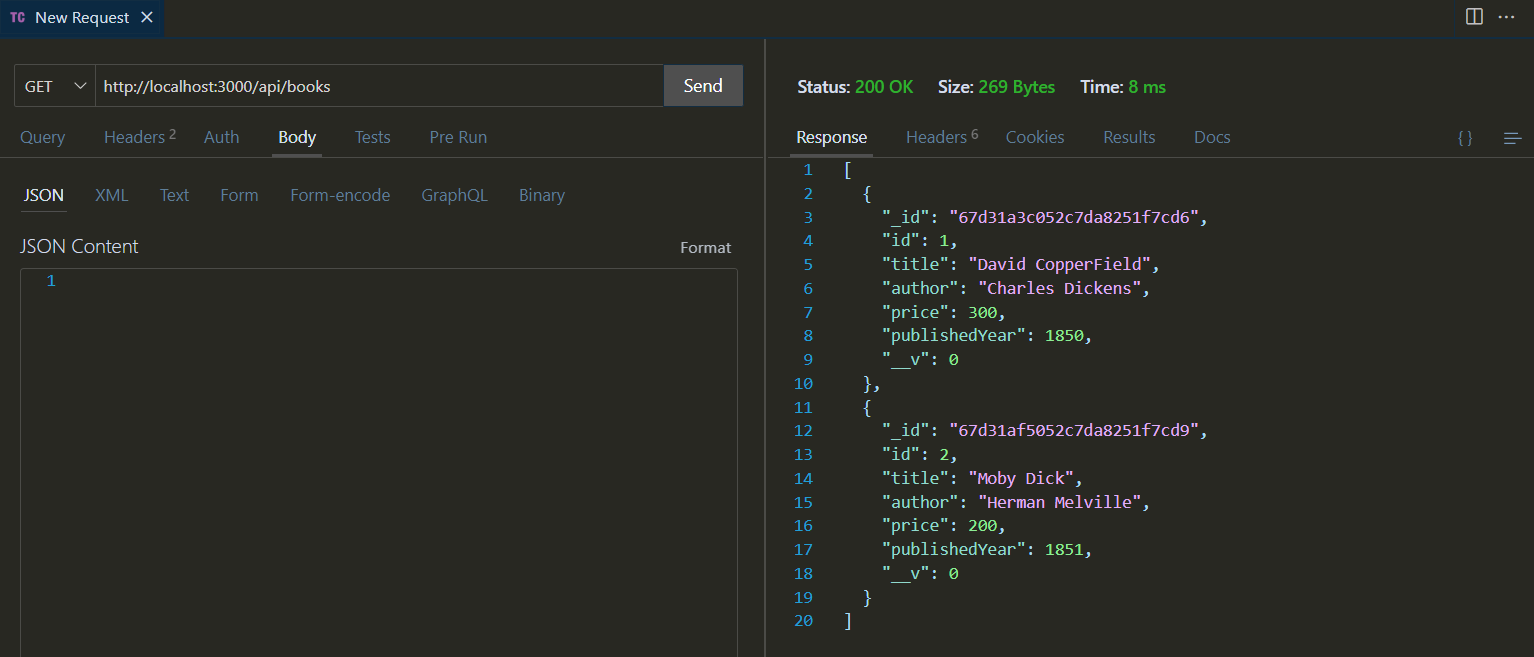
        res.status(500).json({error: "Internal server error"});

    }

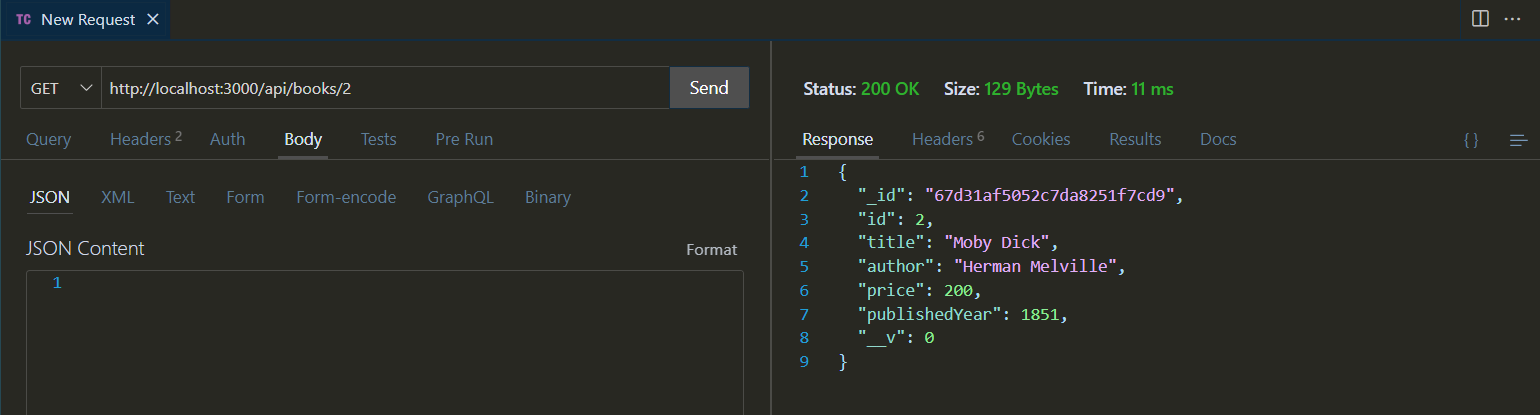
};

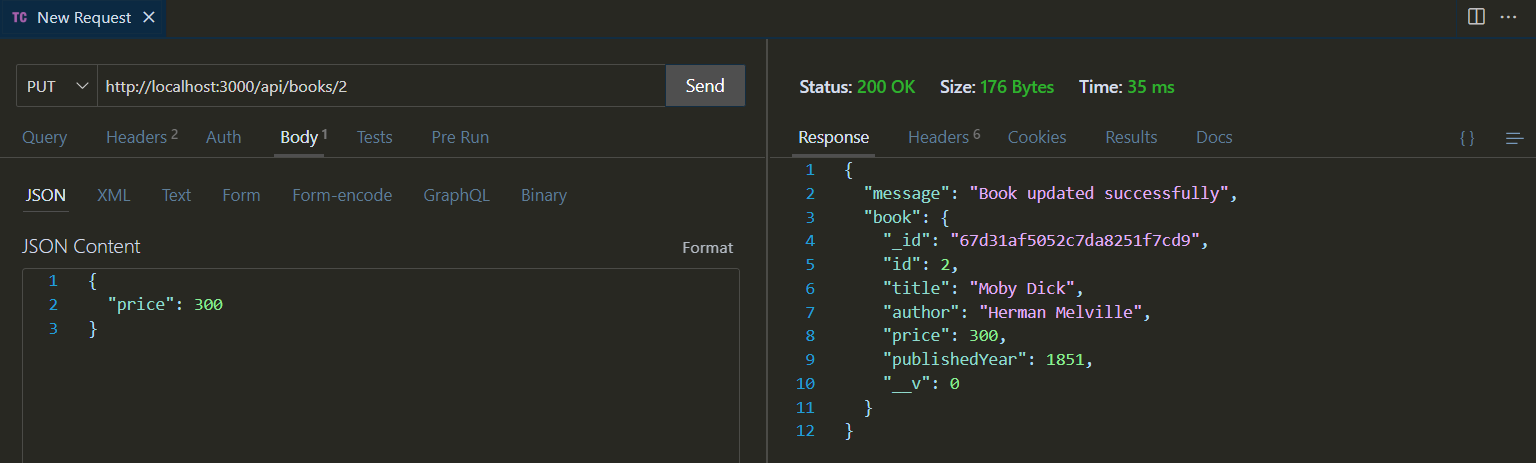
-> Output: 1) Add book (POST)



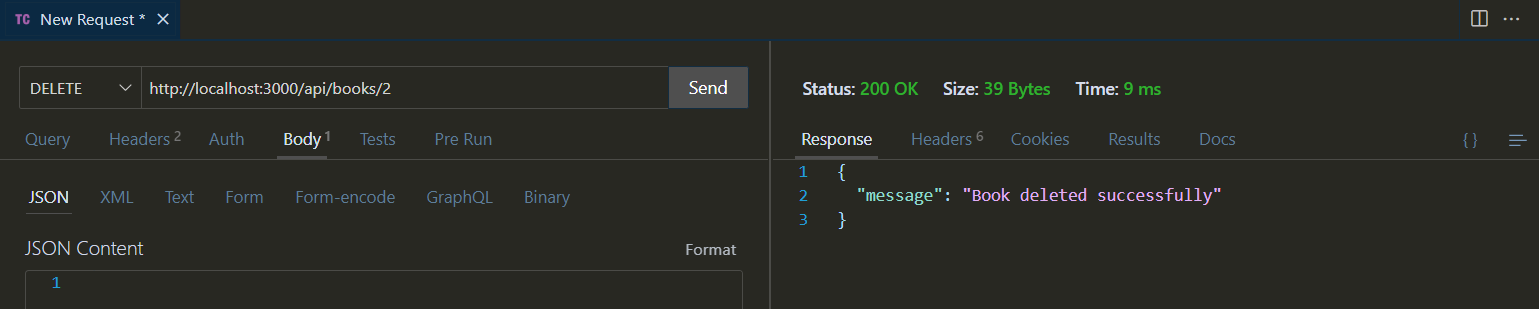
2) View all books (GET)



3) View book by id (GET)

4) Update book by id (PUT)



5) Delete book by id (DELETE)

