Phần 1:  
1. Node.js là một môi trường mã nguồn mở, được phát triển trên ngôn ngữ JavaScript. Sử dụng JavaScript cho cả frontend và backend. Có nhiều thư viện hỗ trợ.   
Xây dựng ứng dụng mạng có khả năng mở rộng cao như web server, API. Phát triển ứng dụng real-time (chat, game trực tuyến) . Xử lý tác vụ I/O không đồng bộ. Phát triển công cụ command line. Tạo microservices. Có thể thu thập dữ liệu biểu mẫu, tạo ra nội dung trang động  
Phù hợp để xây dựng API:  
Nhờ vào cơ chế non-blocking I/O (không chặn), Node.js có thể xử lý hàng nghìn yêu cầu đồng thời mà không bị gián đoạn.

API thường phải xử lý nhiều yêu cầu đồng thời, và Node.js rất mạnh mẽ trong việc xử lý bất đồng bộ. Giúp dễ dàng xử lý các tác vụ yêu cầu truy vấn dữ liệu trong cơ sở dữ liệu

Node.js rất dễ dàng mở rộng nhờ vào việc sử dụng mô hình sự kiện và callback. Việc này giúp ứng dụng API có thể chịu tải lớn mà không cần quá nhiều tài nguyên hệ thống.

Được phát triển vào năm 2008 đến nay vậy nên cộng đồng phát triển lớn cùng với các tài liệu phong phú.

2. So sánh “require” và “import”:  
**Cú pháp**:

* require là cú pháp của **CommonJS**, dùng để nhập khẩu module. (cú pháp động)
* import là cú pháp của **ES Modules (ESM)**, yêu cầu cấu hình type: "module" trong package.json. (cú pháp tĩnh)

**Tải module**:

* require hoạt động **đồng bộ**.
* import hoạt động **bất đồng bộ** và phải dùng ở cấp độ module.

**Khả năng sử dụng**:

* require có thể gọi ở bất kỳ đâu trong mã nguồn.
* import chỉ có thể gọi ở cấp module, không thể dùng trong các hàm hoặc điều kiện.(có nghĩa là chỉ có thể gọi ở đầu như các ngôn ngữ khác)

**Hỗ trợ cú pháp**:

* require chỉ hỗ trợ nhập khẩu theo cách truyền thống.
* import hỗ trợ cả nhập khẩu mặc định và nhập khẩu theo tên.

3. Một ứng dụng API RESTful thường có 5 phương thức HTTP:

GET: lấy dữ liệu

POST: tạo dữ liệu  
PUT: cập nhật hoặc tạo mới nếu chưa có  
PATCH: cập nhật một phần (chỉ thay đổi một phần dữ liệu thay vì toàn bộ)  
DELETE: xóa dữ liệu

Phần 2:  
  
B1: tạo folder dự án:”test”. Tải nodejs, express  
Terminal:

D:\test>npm init -y  
Wrote to D:\test\package.json:

{

"name": "test",

"version": "1.0.0",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"keywords": [],

"author": "",

"license": "ISC",

"description": ""

}

D:\test>npm install express body-parser

added 66 packages, and audited 67 packages in 3s

14 packages are looking for funding

run `npm fund` for details

found 0 vulnerabilities

B2:  
Tạo file index.js:  
// Import thư viện Express

const express = require('express');

// Khởi tạo ứng dụng Express

const app = express();

const PORT = 3000;

// Middleware để parse JSON body

app.use(express.json());

// Mảng lưu trữ dữ liệu sinh viên

let students = [

  { id: 1, name: 'Nguyễn Văn A', email: 'nguyenvana@example.com', gpa: 3.5 },

  { id: 2, name: 'Trần Thị B', email: 'tranthib@example.com', gpa: 3.8 },

  { id: 3, name: 'Lê Văn C', email: 'levanc@example.com', gpa: 3.2 }

];

// Hàm validate email

function isValidEmail(email) {

  const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

  return emailRegex.test(email);

}

// Hàm validate dữ liệu sinh viên

function validateStudent(student) {

  const errors = [];

  if (!student.name || student.name.trim() === '') {

    errors.push('Tên không được để trống');

  }

  if (!student.email || !isValidEmail(student.email)) {

    errors.push('Email không hợp lệ');

  }

  if (student.gpa === undefined || student.gpa < 0 || student.gpa > 10) {

    errors.push('GPA phải từ 0 đến 10');

  }

  return errors;

}

// Route chính - trang chào mừng

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

  res.send(`

    <h1>Ứng dụng Quản lý Sinh viên</h1>

    <p>Sử dụng /students để xem danh sách sinh viên</p>

  `);

});

// 1. Route GET /students - Trả về toàn bộ danh sách sinh viên

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

  res.json(students);

});

// 2. Route POST /students - Thêm mới sinh viên

app.post('/students', (req, res) => {

  const { name, email, gpa } = req.body;

  const newStudent = {

    id: students.length > 0 ? Math.max(...students.map(s => s.id)) + 1 : 1,

    name,

    email,

    gpa: parseFloat(gpa)

  };

  const validationErrors = validateStudent(newStudent);

  if (validationErrors.length > 0) {

    return res.status(400).json({ errors: validationErrors });

  }

  students.push(newStudent);

  res.status(201).json(newStudent);

});

// 3. Route PUT /students/:id - Cập nhật sinh viên

app.put('/students/:id', (req, res) => {

  const id = parseInt(req.params.id);

  const { name, email, gpa } = req.body;

  const studentIndex = students.findIndex(s => s.id === id);

  if (studentIndex === -1) {

    return res.status(404).json({ message: 'Không tìm thấy sinh viên' });

  }

  const updatedStudent = {

    id,

    name,

    email,

    gpa: parseFloat(gpa)

  };

  const validationErrors = validateStudent(updatedStudent);

  if (validationErrors.length > 0) {

    return res.status(400).json({ errors: validationErrors });

  }

  students[studentIndex] = updatedStudent;

  res.json(updatedStudent);

});

// 4. Route DELETE /students/:id - Xóa sinh viên

app.delete('/students/:id', (req, res) => {

  const id = parseInt(req.params.id);

  const studentIndex = students.findIndex(s => s.id === id);

  if (studentIndex === -1) {

    return res.status(404).json({ message: 'Không tìm thấy sinh viên' });

  }

  const deletedStudent = students.splice(studentIndex, 1)[0];

  res.json({ message: 'Xóa sinh viên thành công', student: deletedStudent });

});

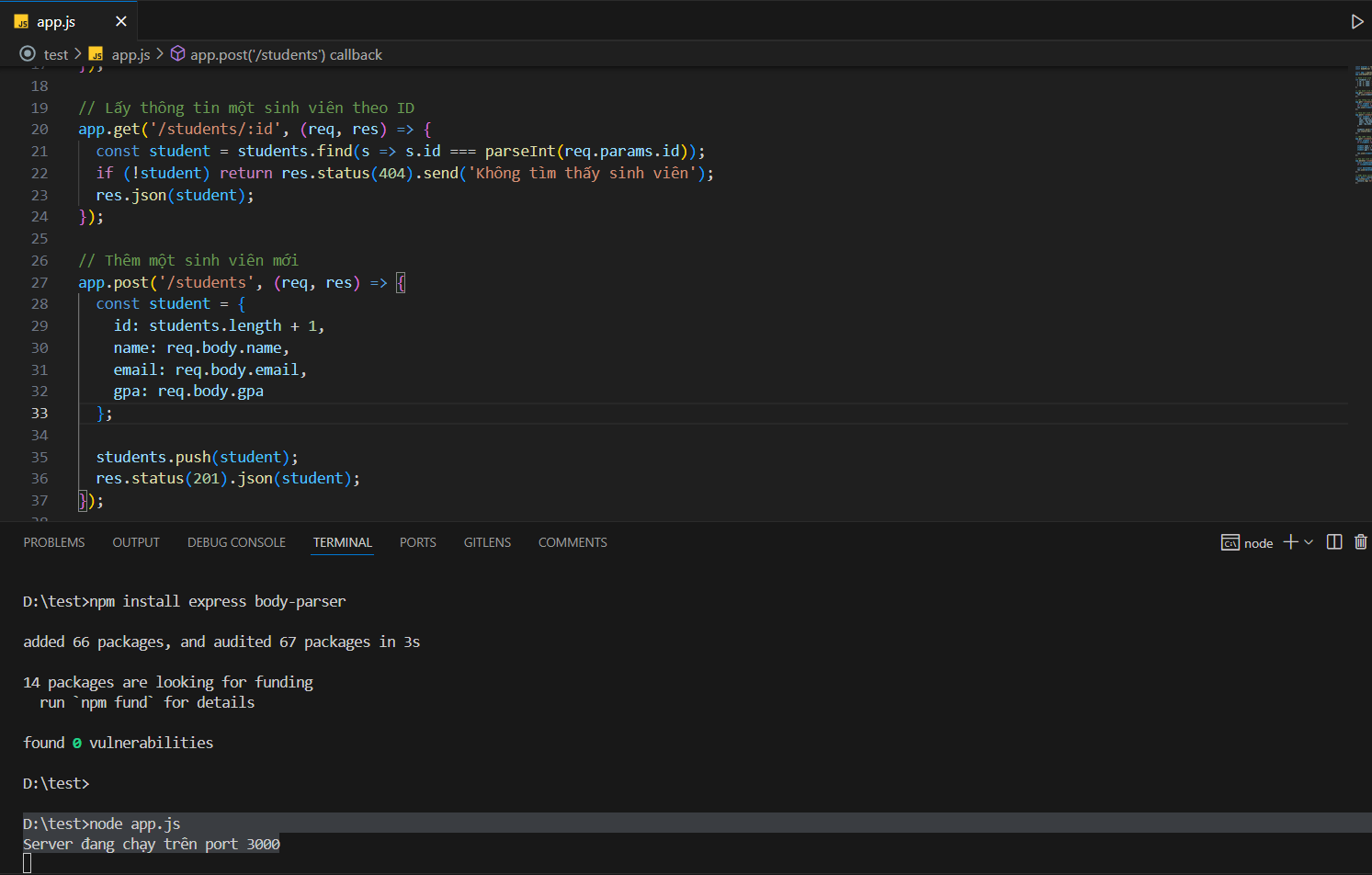
// Khởi động server

app.listen(PORT, () => {

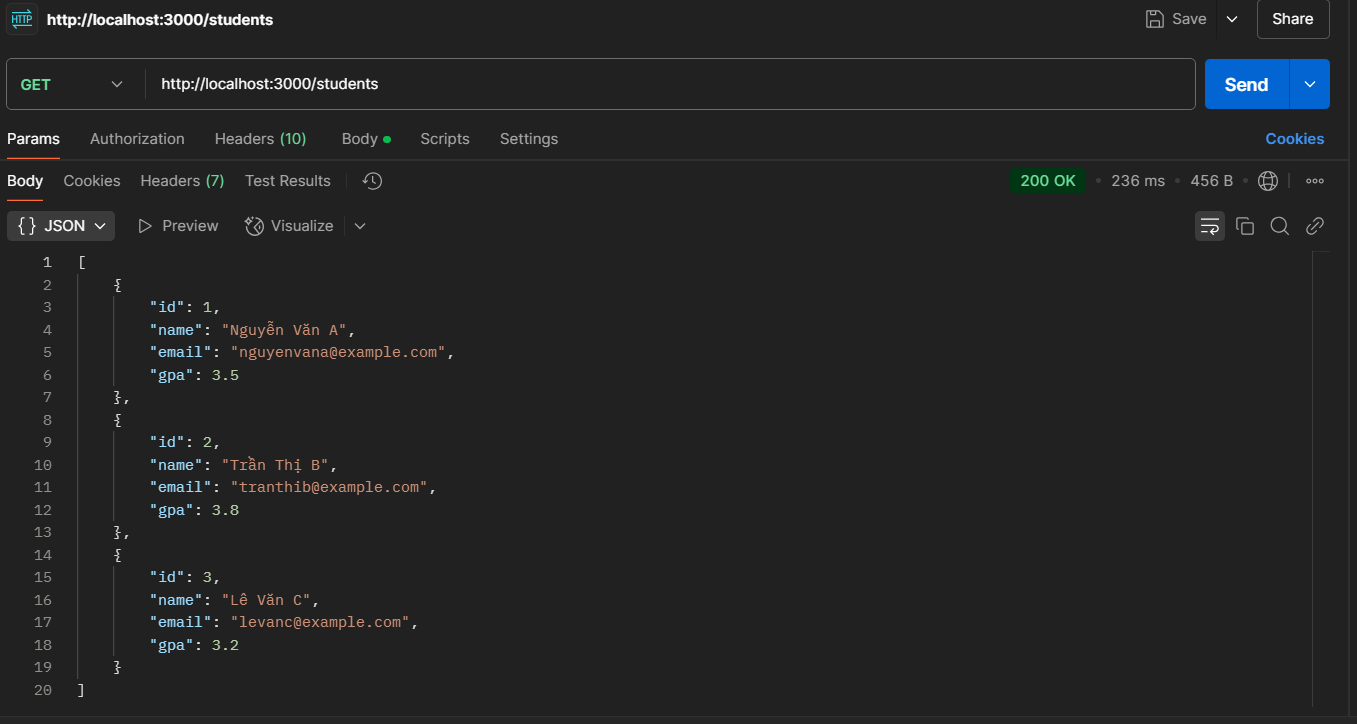
  console.log(`Server đang chạy tại http://localhost:${PORT}`);

});

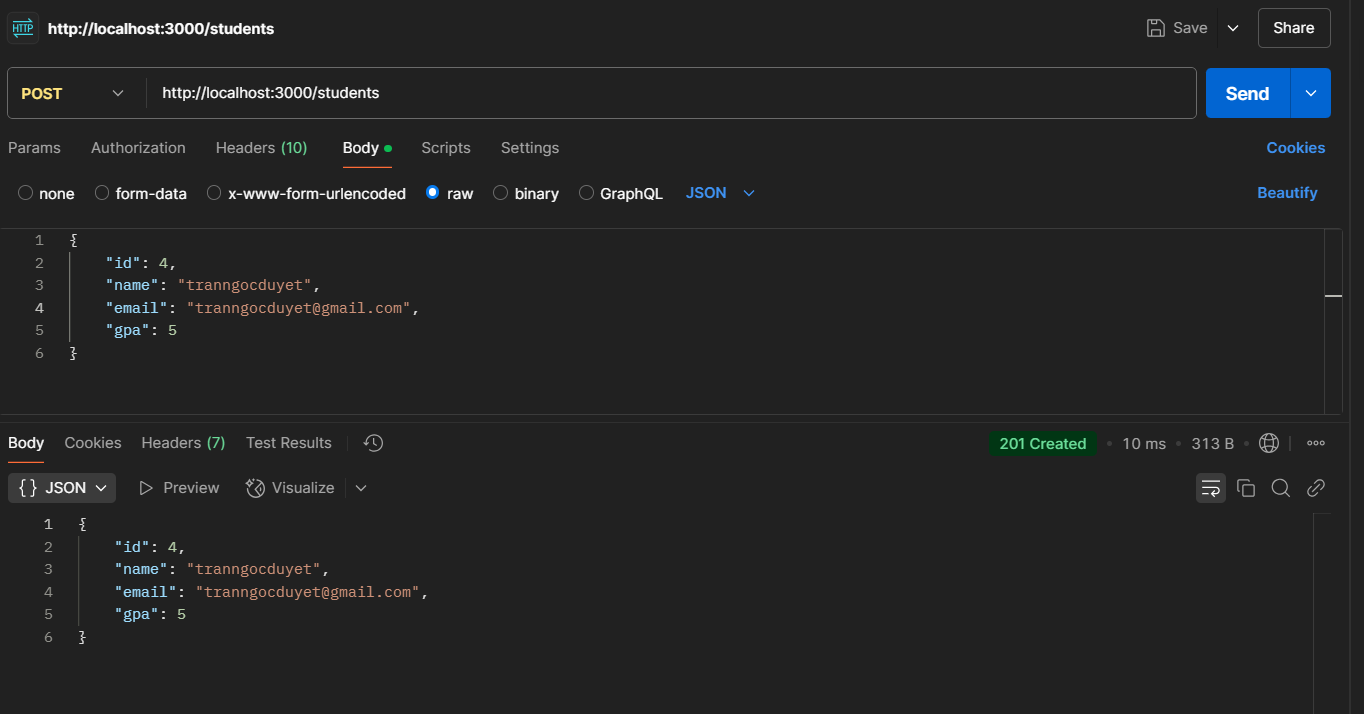
B3: chạy lệnh node app.js trên Terminal.

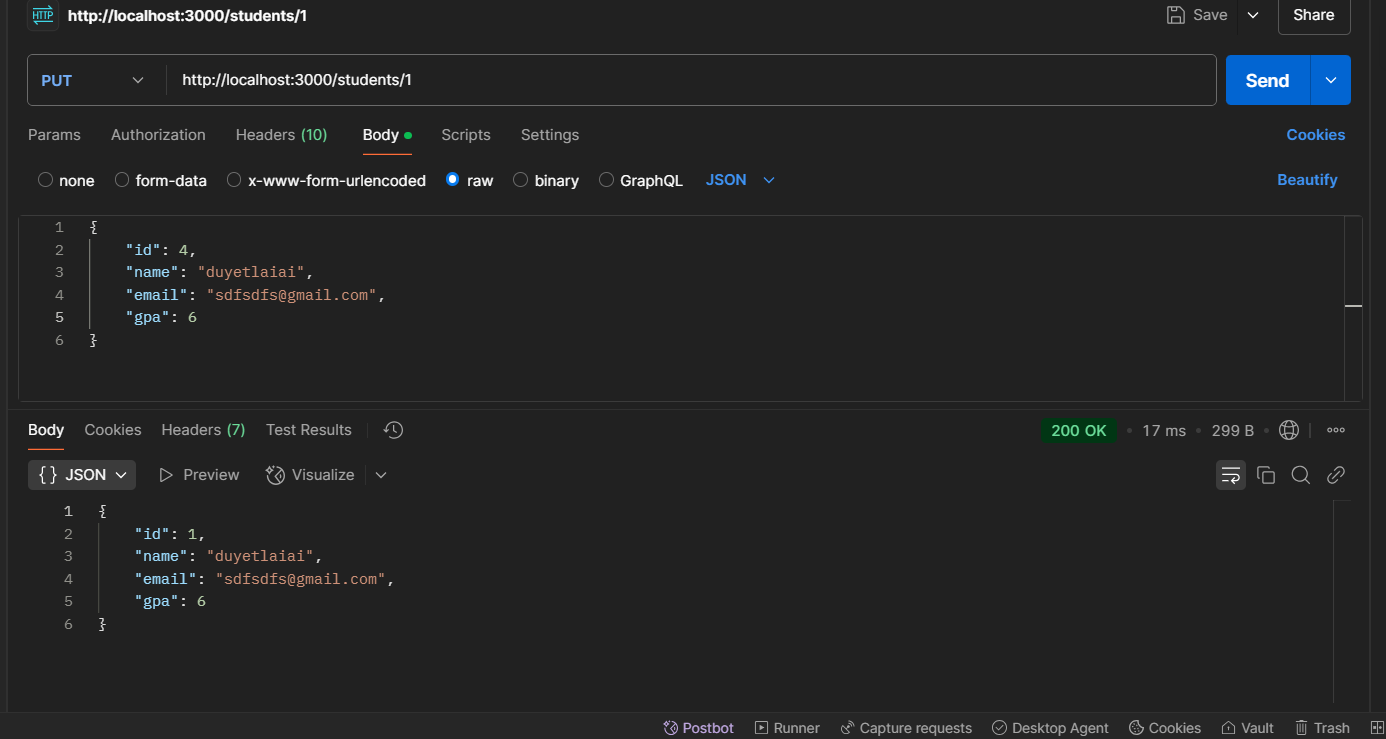


Vào Postnam kiểm tra.   
Get: http://localhost:3000/students

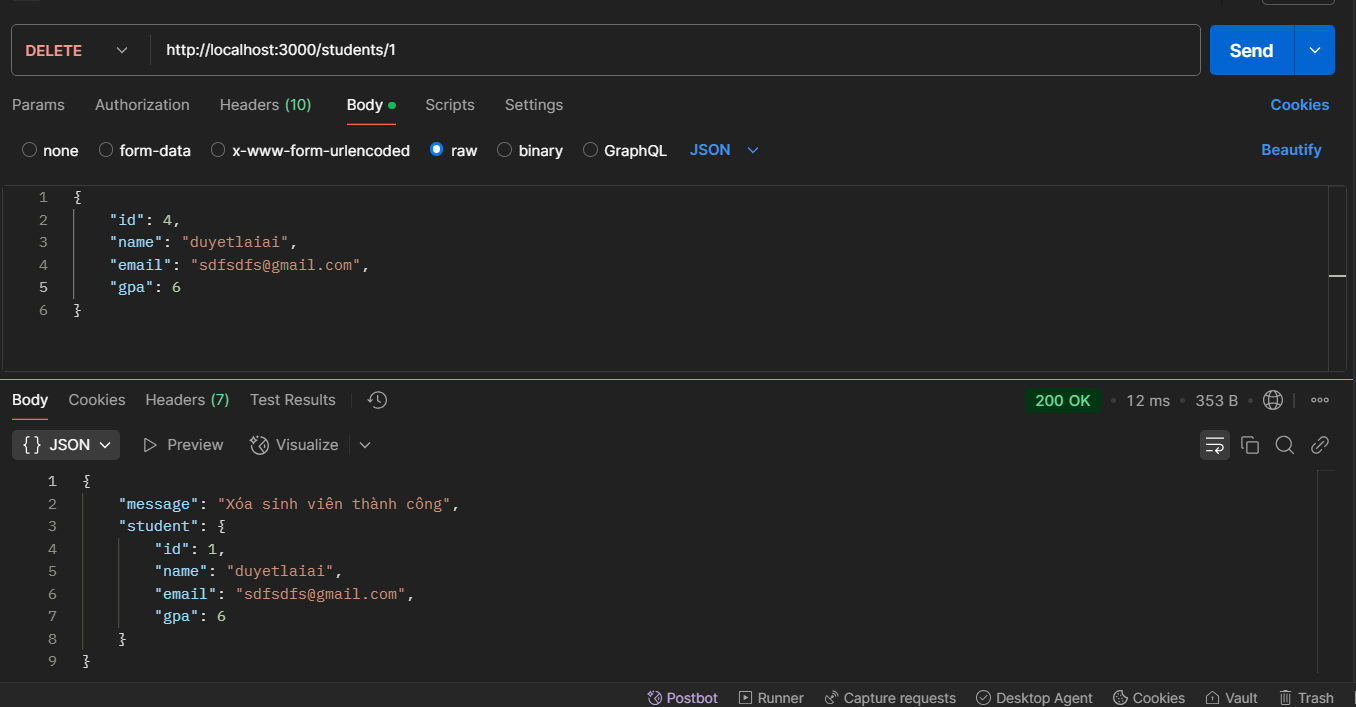


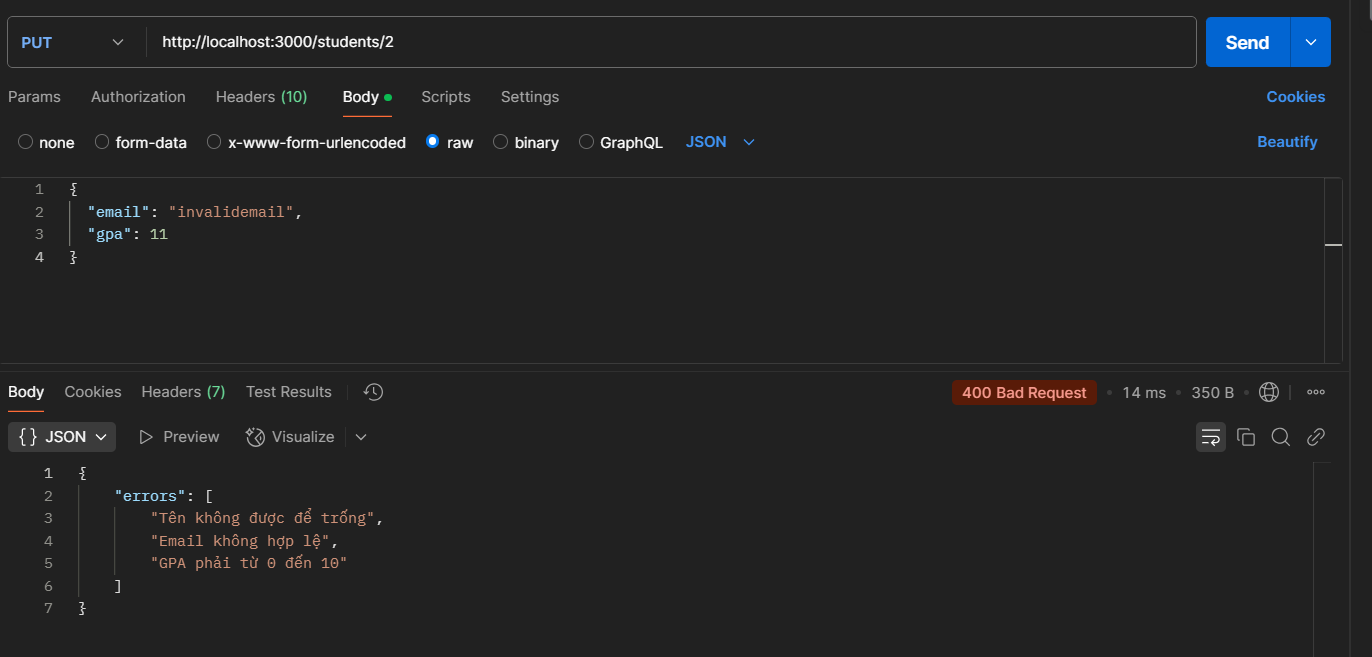
POST:http://localhost:3000/students



PUT:http://localhost:3000/students/1  


DELETE:http://localhost:3000/students/1



Valid:  


Phần 3:  
  
file index.js sau khi đã chỉnh sửa để kết nối với database:  
const express = require('express');

const mysql = require('mysql2/promise'); // Sử dụng promise interface

const app = express();

const PORT = 3000;

app.use(express.json());

// Cấu hình kết nối MySQL

const dbConfig = {

host: 'localhost',

user: 'root',

password: '',

database: 'test'

};

// Middleware kết nối database

app.use(async (req, res, next) => {

try {

req.db = await mysql.createConnection(dbConfig);

next();

} catch (err) {

console.error('Database connection error:', err);

res.status(500).json({ error: 'Database connection failed' });

}

});

// Đóng kết nối sau mỗi request

app.use((req, res, next) => {

res.on('finish', () => {

if (req.db) req.db.end();

});

next();

});

// Route GET /students-from-db

app.get('/students-from-db', async (req, res) => {

try {

const [rows] = await req.db.query('SELECT \* FROM students');

res.json(rows);

} catch (err) {

console.error('Database query error:', err);

res.status(500).json({ error: 'Failed to fetch students' });

}

});

// Các route cũ vẫn giữ nguyên

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

res.json(students);

});

// 2. Route POST /students - Thêm mới sinh viên

app.post('/students', (req, res) => {

  const { name, email, gpa } = req.body;

  const newStudent = {

    id: students.length > 0 ? Math.max(...students.map(s => s.id)) + 1 : 1,

    name,

    email,

    gpa: parseFloat(gpa)

  };

  const validationErrors = validateStudent(newStudent);

  if (validationErrors.length > 0) {

    return res.status(400).json({ errors: validationErrors });

  }

  students.push(newStudent);

  res.status(201).json(newStudent);

});

// 3. Route PUT /students/:id - Cập nhật sinh viên

app.put('/students/:id', (req, res) => {

  const id = parseInt(req.params.id);

  const { name, email, gpa } = req.body;

  const studentIndex = students.findIndex(s => s.id === id);

  if (studentIndex === -1) {

    return res.status(404).json({ message: 'Không tìm thấy sinh viên' });

  }

  const updatedStudent = {

    id,

    name,

    email,

    gpa: parseFloat(gpa)

  };

  const validationErrors = validateStudent(updatedStudent);

  if (validationErrors.length > 0) {

    return res.status(400).json({ errors: validationErrors });

  }

  students[studentIndex] = updatedStudent;

  res.json(updatedStudent);

});

// 4. Route DELETE /students/:id - Xóa sinh viên

app.delete('/students/:id', (req, res) => {

  const id = parseInt(req.params.id);

  const studentIndex = students.findIndex(s => s.id === id);

  if (studentIndex === -1) {

    return res.status(404).json({ message: 'Không tìm thấy sinh viên' });

  }

  const deletedStudent = students.splice(studentIndex, 1)[0];

  res.json({ message: 'Xóa sinh viên thành công', student: deletedStudent });

});

// Khởi động server

app.listen(PORT, () => {

  console.log(`Server đang chạy tại http://localhost:${PORT}`);

});

