ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

DEVELOPER GUIDE CỦA ỨNG DỤNG

Giảng viên hướng dẫn: Thầy Trần Duy Thảo

Sinh viên thực hiện	Mã số sinh viên
Võ Văn Nam	22120222
Thái Đình Ngân	22120223

MỤC LỤC

I. CODING STANDARD

1. Quy ước đặt tên

Thành phần	Quy ước	Ví dụ
Biến/hàm	camelCase	getStudentById
Tên class	PascalCase	Student, ApiError
File/module	kebab-case	class.service.js
Constants	UPPER_SNAKE_CASE	MAX_STUDENTS
Tên branch	/feature, /frontend	feature/add-logging

2. Format của Code sử dụng Eslint

a. no-console: 1

- Không khuyến khích dùng console.log() trong production.
- Vẫn cho phép trong quá trình phát triển.

```
console.log('Debugging ... '); // warning
```

b. no-lonely-if: 1

- Không cho phép if đơn lẻ nằm trong khối else.
- Thay bằng else if.

```
// Sai
if (x) {
    // ...
} else {
    if (y) {
        // ...
    }
}

// Đúng
if (x) {
    // ...
} else if (y) {
    // ...
}
```

c. **no-unused-vars: 1**

- Cảnh báo biến khai báo nhưng không sử dụng.

```
const unused = 123; // warning
```

d. **no-trailing-spaces: 1**

- Không cho phép khoảng trắng ở cuối dòng.

e. **no-multi-spaces: 1**

- Không cho phép dùng nhiều khoảng trắng liên tiếp ngoại trừ căn chỉnh trong comment hoặc khai báo dạng bảng.

```
const abc = 5;      // có khoảng trăng
```

f. **no-multiple-empty-lines: 1**

- Không cho phép nhiều dòng trống liên tiếp.

```
console.log('A');


console.log('B'); // ⚠ warning
```

g. **space-before-blocks: ['error', 'always']**

- Phải có khoảng trắng trước {.

```
if (true) { // ❌
    // ...
}

if (true) { // ✅
    // ...
}
```

h. object-curly-spacing: [1, 'always']

- Phải có khoảng trắng bên trong dấu {} của object.

```
const obj1 = {a:1};        // ❌
const obj2 = { a: 1 };     // ✅
```

i. indent: ['warn', 4]

- Thụt đầu dòng 4 dấu cách.

```
function test() {
    console.log('Indent 4 spaces'); // ✅
}
```

j. array-bracket-spacing: 1

- Khuyến khích có khoảng trắng trong mảng.

```
const array1 = [1,2];      // ❌
const array2 = [ 1, 2 ];   // ⚠
```

k. linebreak-style: 0

- Không kiểm tra kiểu xuống dòng (LF vs CRLF) → phù hợp cho dự án làm việc trên nhiều HĐH.

l. no-unexpected-multiline: 'warn'

- Cảnh báo lỗi do dòng xuống không đúng cách.

m. keyword-spacing: 1

- Khuyến khích có khoảng trắng sau if, for, return, …

```
if(true) { }   // ❌
if (true) { }  // ⚠
```

n. comma-dangle: 1

- Khuyến khích có dấu phẩy cuối cùng trong object/array đa dòng.

```
const obj = {
    a: 1,
    b: 2 // ⚠ thiếu dấu phẩy khuyên nghị
};
```

o. **comma-spacing: 1**

- Cảnh báo nếu không có khoảng trắng sau dấu phẩy.

```
const array1 = [1,2,3];     // ❌
const array2 = [1, 2, 3];   // ⚠
```

p. **arrow-spacing: 1**

- Khuyến khích có khoảng trắng quanh => trong arrow function.

```
const add = (a, b)⇒a + b;       // ❌
const add = (a, b) ⇒ a + b;     // ⚠
```

II. KIẾN TRÚC HỆ THỐNG CỦA ỨNG DỤNG

1. Lõi ứng dụng – Application Core

- Đây là tầng trong cùng của kiến trúc hệ thống. Mọi logic truy xuất dữ liệu, logic nghiệp vụ và các thành phần cốt lõi đều nằm trong tầng này. Trong dự án, các thành phần này được đặt trong các thư mục bên trong src/.
- Tầng này bao gồm ba phần chính:

models/

- Chứa các định nghĩa mô hình dữ liệu sử dụng Mongoose (MongoDB). Đây là nơi định hình cấu trúc collection, ràng buộc schema và các phương thức thao tác dữ liệu cơ bản.
- Ví dụ: Student, Class, Enrollment,...
- Tầng models/ không phụ thuộc vào bất kỳ tầng nào khác, và là trung tâm của lõi ứng dụng.

repositories/

- Chứa các lớp có nhiệm vụ truy xuất dữ liệu từ MongoDB thông qua các model. Tầng này giúp tách rời logic truy vấn DB ra khỏi logic nghiệp vụ.
- Ví dụ: student.repository.js, class.repository.js
- Các repository có thể sử dụng: findById, findAll, create, update, delete, v.v.
- Tầng này phụ thuộc vào models/ nhưng không phụ thuộc vào service.

services/

- Đây là lớp bên ngoài cùng của lõi ứng dụng. Tầng này chứa toàn bộ logic nghiệp vụ chính, được gọi từ controller.
- Service sử dụng repository để thao tác dữ liệu và có thể chứa:
  - Kiểm tra ràng buộc logic (class đã đủ người, student đã tồn tại,...).
  - Tính toán, phân quyền nội bộ.
  - Gọi đến nhiều repository khác nhau.
- Service phụ thuộc vào repositories/ và models/, nhưng không phụ thuộc vào controller/.

2. **Tầng giao diện – UI Layer**

- Tầng này chứa toàn bộ phần giao tiếp với client. Trong dự án của bạn, tầng UI được tổ chức thành hai phần:

controllers/

- Tầng này nhận request từ phía client (qua routes/), xử lý thông tin đầu vào và trả response.
- Controller có nhiệm vụ:
  - Gọi service tương ứng.
  - Không chứa logic nghiệp vụ.
  - Trả dữ liệu hoặc lỗi về client.
- Ví dụ: student.controller.js, class.controller.js

routes/

- Định nghĩa các endpoint HTTP RESTful. Ví dụ: `router.post("/students", validateRegistration, studentController.createStudent);`
- Tầng này sẽ gắn các middleware như validate, auth,… và trỏ đến controller tương ứng.

middlewares/

- Tầng trung gian có thể được gắn vào từng route hoặc toàn cục:
- Ví dụ: errorHandling.middleware.js: Bắt lỗi toàn cục.

validators/

- Xác định các schema để kiểm tra dữ liệu đầu vào từ request body/query/params.

utils/

- Chứa các hàm tiện ích như: ApiError.js: Lớp định nghĩa lỗi tuỳ chỉnh.
- Tầng này được tái sử dụng ở mọi nơi trong hệ thống.

3. **Tầng kiểm thử – Test Layer**
   - Tầng kiểm thử giúp kiểm tra từng thành phần của hệ thống, đảm bảo logic hoạt động đúng.
   - Kiểm thử đơn vị (unit test) các hàm trong tầng service.
   - Kiểm tra nghiệp vụ (ví dụ: không cho thêm học sinh nếu lớp đầy).
4. **Cấu hình & triển khai**

   config/

   - Chứa các thiết lập cấu hình toàn hệ thống:
     - Thông tin kết nối MongoDB.
     - Sử dụng dotenv để nạp .env.

   server.js

   - Điểm khởi động của ứng dụng:
     - Load middleware toàn cục.
     - Gắn các route chính.
     - Bắt đầu lắng nghe cổng.

   uploads/

   - Thư mục chứa các file được người dùng upload (nếu có): ảnh đại diện, tài liệu, v.v.

III. TỔ CHỨC MÃ NGUỒN

1. **Core Layer – Tầng lõi ứng dụng**

   src/models/

   - Chứa các schema định nghĩa dữ liệu bằng Mongoose. Mỗi schema đại diện cho một bảng (collection) trong MongoDB như Student, Class, Grade,...
   - Tất cả thực thể cốt lõi (entities) được định nghĩa tại đây. Không có sự phụ thuộc nào từ models/ sang các thư mục khác, do đó đây là tầng lõi độc lập nhất của ứng dụng.

   src/repositories/

- Chứa các hàm truy cập dữ liệu cho từng model. Thư mục này đóng vai trò Data Access Layer, giúp phân tách rõ ràng giữa logic nghiệp vụ và thao tác DB.

- Ví dụ: student.repository.js: findById, findAll, create, delete,...

src/services/

- Chứa toàn bộ logic nghiệp vụ của hệ thống. Đây là nơi mà controller sẽ gọi để xử lý yêu cầu từ client.
- Ví dụ:
  - student.service.js: kiểm tra sinh viên đã tồn tại chưa trước khi thêm.
  - class.service.js: kiểm tra lớp còn chỗ không trước khi ghi danh.
- Một số người gọi đây là tầng Business Access Layer (BAL).

2. **Routing & Giao diện lập trình ứng dụng (API)**

src/routes/

- Chứa các file khai báo điểm cuối API (RESTful). Mỗi route đại diện cho một nhóm chức năng như: /students, /classes, /grades, …
- Các route gắn middleware, gọi controller tương ứng.

src/controllers/

- Chứa các hàm xử lý request từ client. Đây là nơi tiếp nhận đầu vào, kiểm tra đơn giản và gọi service.
- Controller không chứa logic phức tạp.

src/middlewares/

- Xử lý yêu cầu trung gian như:
  - Xác thực token JWT.
  - Kiểm tra quyền truy cập.
  - Validate dữ liệu.
  - Bắt lỗi.
- Middleware có thể gắn ở cấp route hoặc toàn hệ thống.

src/validators/

- Chứa các schema kiểm tra dữ liệu đầu vào, thường dùng thư viện Joi. Việc tách riêng validator giúp dễ quản lý và tái sử dụng.

src/utils/

- Chứa các tiện ích dùng chung:
  - ApiError.js: Định nghĩa lỗi tuỳ chỉnh.
  - logger.js: Ghi log hệ thống.
  - tokenGenerator.js: Tạo JWT.

src/config/

- Chứa cấu hình toàn hệ thống:
  - Kết nối cơ sở dữ liệu
  - Thiết lập JWT
  - Biến môi trường
  - Sử dụng dotenv để đọc file .env.

server.js

- Tập tin khởi động server:
  - Gắn route chính.
  - Khởi động middleware.
  - Mở port và bắt đầu lắng nghe request.

3. **Kiểm thử – Test Layer (src/tests/)**

tests/services/

- Test logic trong các service. Dùng framework như Jest.
- Ví dụ:
  - Kiểm tra không thể thêm sinh viên nếu trùng ID.
  - Kiểm tra không thể ghi danh nếu lớp đầy.

tests/controllers/

- Test API bằng cách gửi HTTP request thật (dùng Supertest).
- Ví dụ: Gửi POST /students với dữ liệu hợp lệ và kiểm tra kết quả trả về.

4. **Seeder – Dữ liệu mẫu (src/seeders/)**

- Chứa dữ liệu mẫu giúp khởi tạo DB khi phát triển:
- student.seeder.js
- class.seeder.js

5. **Uploads – Lưu trữ tạm (/uploads/)**

- Thư mục để lưu ảnh, tài liệu hoặc file mà người dùng upload.

IV. BẮT ĐẦU VỚI VIỆC PHÁT TRIỂN ỨNG DỤNG

1. Giới thiệu

- Ứng dụng này là một nền tảng Node.js mã nguồn mở, được thiết kế với backend RESTful API cho hệ thống quản lý sinh viên, lớp học, điểm số,...
- Đồng thời, ứng dụng hỗ trợ quy trình phát triển chuẩn, mã nguồn dễ đọc, dễ dàng mở rộng và test.

2. Công cụ cần thiết cho phát triển

- Node.js (>= 18)
- MongoDB (Sử dụng MongoDB Atlas)
- Postman (test API)
- VS Code (hoặc IDE tùy chọn)

3. Stack công nghệ sử dụng

- Application Layer (Giao tiếp người dùng):
  - Express.js
  - Winston: Dùng để **ghi log hệ thống (info, error, warn, debug)**
  - Morgan: Dùng để **ghi log các HTTP requests** đến server Express.
  - Mongoose: mapping giữa MongoDB và object
  - ESLint + Prettier: format mã nguồn
- Data Layer (Tầng dữ liệu):
  - MongoDB (NoSQL)
  - Dev Tool
  - Nodemon: reload khi thay đổi file
  - Jest: test unit

4. Cách chạy chương trình

- Đầu tiên chạy lệnh sau để clone repository về máy tính:

  git clone https://github.com/sitrismart/DeepBlue-Ex-TKPM.git

  cd DeepBlue-Ex-TKPM
- Tiếp theo thiết lập .env như trong file .env.example

a. Cài đặt và chạy Backend (BE)

- Mở terminal và chạy lệnh `npm install` để cài đặt tất cả các package từ package.json:

  npm install

- Sau khi tải xong, dùng lệnh `npm start` để chạy chương trình:

  npm start

- Backend sẽ được host local tại port theo cài đặt trong file .env, ví dụ:

  http://localhost:5134

b. Chạy test cho Backend

- Mở một terminal mới và di chuyển vào thư mục frontend, sau đó cài đặt các package cho backend:

  cd backend

  npm install

- Chạy lệnh `npm test` để chạy các unit test cho backend:

  npm test

c. Cài đặt và chạy Frontend (FE)

- Mở một terminal mới và di chuyển vào thư mục frontend, sau đó cài đặt các package cho frontend

  cd frontend

  npm install

- Sau khi cài xong, dùng lệnh sau để chạy chương trình frontend:

  npm run dev

- Frontend sẽ được host tại http://localhost:5173. Truy cập trang tại:

  http://localhost:5173

d. Chạy test cho Frontend

- Mở một terminal mới và di chuyển vào thư mục frontend, sau đó cài đặt các package cho frontend

  cd frontend

  npm install

- Chạy lệnh `npx vitest run` để chạy các unit test cho frontend:

  npx vitest run

e. Biên dịch

- Sử dụng lệnh `npm install` và `npm start` cho backend, lệnh `npm install` và `npm run dev` cho frontend, ngoài ra không cần biên dịch gì thêm.

f. Chạy chương trình

- Chạy chương trình bằng các lệnh trong hướng dẫn sử dụng trong terminal.

- Truy cập trang web tại: http://localhost:5173.

V. CẤU TRÚC BẢNG CỦA CƠ SỞ DỮ LIỆU

- Ứng dụng sử dụng MongoDB làm hệ quản trị cơ sở dữ liệu NoSQL, tổ chức dữ liệu theo mô hình schema-less nhưng được kiểm soát thông qua Mongoose schemas.

- Mỗi collection tương ứng với một mô hình nghiệp vụ trong hệ thống.

1. Student (Sinh viên)

Collection: `students`

Trường	Kiểu dữ liệu	Mô tả
studentId	String	Mã số sinh viên, duy nhất
fullName	String	Họ và tên
dateOfBirth	Date	Ngày sinh
gender	String	Giới tính
course	String	Khóa học (ví dụ: “22”)
permanentAddress	String	Địa chỉ hộ khẩu
currentAddress	String	Nơi ở hiện tại
mailingAddress	String	Địa chỉ nhận thư
email	String	Email, duy nhất

phoneNumber	String	Số điện thoại
identityNumber	String	Số CMND/CCCD/Hộ chiếu, duy nhất
identityIssuedDate	Date	Ngày cấp
identityExpiryDate	Date	Ngày hết hạn
identityIssuedPlace	String	Nơi cấp
identityCountry	String	Quốc gia
identityHasChip	Boolean	Có chip (dành cho CCCD)
identityNotes	String	Ghi chú (nếu có)
nationality	String	Quốc tịch
faculty	String	Mã khoa
program	String	Chương trình đào tạo
studentStatus	String	Trạng thái sinh viên

2. **Faculty (Khoa)**

   Collection: `faculties`

Trường	Kiểu dữ liệu	Mô tả
facultyId	String	Mã khoa, duy nhất
facultyName	String	Tên khoa

3. **Course (Học phần)**

Collection: `courses`

Trường	Kiểu dữ liệu	Mô tả
courseCode	String	Mã học phần, duy nhất
courseName	String	Tên học phần
creditHours	Number	Số tín chỉ
department	String	Bộ môn
description	String	Mô tả
prerequisite	String	Môn học tiên quyết

4. Class (Lớp học phần)

Collection: `classes`

Trường	Kiểu dữ liệu	Mô tả
classCode	String	Mã lớp học phần, duy nhất
courseCode	String	Mã học phần
academicYear	String	Niên khóa
semester	String	Học kỳ
lecturer	String	Giảng viên phụ trách
maxStudents	Number	Số lượng tối đa
registeredCount	Number	Số lượng sinh viên đã đăng ký
schedule	String	Thời khóa biểu

classroom	String	Phòng học

5. **Enrollment (Đăng ký lớp)**

Collection: `enrollments`

Trường	Kiểu dữ liệu	Mô tả
studentId	String	Mã sinh viên
classCode	String	Mã lớp học phần
enrollmentDate	Date	Ngày đăng ký

6. **Grade (Điểm số)**

Collection: `grades`

Trường	Kiểu dữ liệu	Mô tả
studentId	String	Mã sinh viên
courseCode	String	Mã học phần
courseName	String	Tên học phần
grade	Number	Điểm số
status	String	Trạng thái học phần
remark	String	Ghi chú

7. **Configuration (Cấu hình hệ thống)**

Collection: `configurations`

Trường	Kiểu dữ liệu	Mô tả
currentSchoolYear	String	Ví dụ: '2024-2025'
semester	Number	Số thứ tự học kỳ

VI. CẬP NHẬT THỰC THỂ ĐÃ TỒN TẠI

- Hướng dẫn cách cập nhật một entity hiện có trong hệ thống Node.js sử dụng MongoDB – cụ thể là thêm một thuộc tính mới cho entity Class.

1. Mô hình dữ liệu (Data Model)

- Vị trí tệp: models/class.model.js
- Trong hệ thống của bạn, mỗi thực thể (entity) như Class, Student, Grade,... được định nghĩa bằng Mongoose Schema. Để thêm một thuộc tính mới vào Class, bạn chỉ cần chỉnh sửa schema trong tệp tương ứng.
- Ví dụ: Thêm thuộc tính mới description

```
const classSchema = new Schema({
  classCode: { type: String, required: true, unique: true },
  subject: { type: String, required: true },
  semester: { type: String, required: true },
  year: { type: Number, required: true },
  maxStudents: { type: Number, required: true },
  registeredCount: { type: Number, default: 0 },
  description: { type: String, maxlength: 255 } // Thuộc tính mới
});
```

2. Mô hình hiển thị (View Model / DTO)

- Vị trí: controllers/class.controller.js hoặc routes/class.route.js
- Ta có thể thêm Validate để kiểm tra dữ liệu trước khi tạo.
- Ví dụ thêm validate:

```
body('description')
  .isLength({ max: 255 })
```

.withMessage('Mô tả không được dài quá 255 ký tự.')

3. **Cập nhật View (Giao diện nhập liệu nếu có)**
   - Cần cập nhật mã nguồn của FrontEnd để thêm trường description:

     <label for="description">Mô tả</label>

     <input type="text" name="description" id="description" class="form-control" maxlength="255" />

4. **Cập nhật Controller (Business Logic)**
   - Trong controller, cần gán dữ liệu từ req.body cho entity khi tạo hoặc cập nhật:

     const newClass = new Class({

     classCode,

     subject,

     semester,

     year,

     maxStudents,

     description // Gán thuộc tính mới

     });
   - Hoặc khi chỉnh sửa:

     existingClass.description = req.body.description;

5. **Kiểm thử**
   - Gửi POST/PUT request thông qua Postman hoặc giao diện để xác thực việc lưu dữ liệu mới thành công.
   - Đảm bảo dữ liệu được lưu đúng trong MongoDB.

VII. TẠO THÊM ROUTE MỚI

1. **Cấu trúc routes trong ứng dụng**
   - Các route của ứng dụng được tổ chức trong thư mục /routes, mỗi tệp đại diện cho một nhóm tài nguyên hoặc chức năng riêng biệt. Mỗi tệp route tương ứng với một controller.
   - Ví dụ cấu trúc:

routes/

│

├── student.route.js

├── class.route.js

├── enrollment.route.js

└── statistics.route.js **Route mới cần đăng ký**

2. **Các bước tạo thêm Route**

- Ở đường dẫn: /routes/statistics.route.js

```
import express from "express";
import statisticsController from "../controllers/statistics.controller";
const router = express.Router();

router.get('/grades/average', statisticsController.getAverageGrade);
router.get('/students/active', statisticsController.getActiveStudents);

module.exports = router;
```

- Tiếp theo đăng ký route với ứng dụng ở /routes/index.js

```
router.use("/statistics", statisticsRoutes);
```

- Sau khi đăng ký, các endpoint mới có thể được truy cập qua:
  - GET /api/statistics/grades/average
  - GET /api/statistics/students/active

VIII. UNIT TEST

1. **Mục đích**

- Tài liệu này mô tả quy trình viết và tổ chức Unit Test trong dự án Node.js quản lý sinh viên. Mục tiêu nhằm đảm bảo chất lượng phần mềm, phát hiện sớm lỗi logic trong các hàm xử lý, controller, service và đảm bảo tính ổn định khi mở rộng hoặc bảo trì hệ thống.

2. **Công cụ sử dụng**

- Jest: Framework kiểm thử đơn vị phổ biến dành cho Node.js

3. **Tổ chức thư mục kiểm thử**

- Mỗi module trong hệ thống nên có thư mục kiểm thử tương ứng. Tên file kiểm thử nên kết thúc bằng ".test.js".
- Ví dụ:

tests/

├── controllers/

│ └── student.controller.test.js

├── services/

│ └── grade.service.test.js

├── models/

│ └── student.model.test.js

└── setup.js (thiết lập ban đầu)

4. **Nguyên tắc viết Unit Test**
   - Mỗi hàm nghiệp vụ chính cần có ít nhất một kiểm thử thành công và một kiểm thử thất bại.
   - Ưu tiên kiểm thử logic nghiệp vụ, tránh phụ thuộc vào bên ngoài (mock Repository, Database...).
   - Tên kiểm thử nên mô tả rõ mục đích, ví dụ: "should return student when given valid ID".
   - Đảm bảo test bao phủ cả các nhánh điều kiện (if, else, throw...).
5. **Tiêu chí kiểm thử**
   - Controller: ≥ 80% coverage
   - Service: ≥ 90% coverage
   - Repository/Model: ≥ 80% coverage
   - Khuyến nghị chạy `jest --coverage` để theo dõi độ bao phủ.
6. **Cách chạy kiểm thử**
   - Thực thi lệnh sau để chạy toàn bộ test:

     npm test
   - Thực thi lệnh sau để xem báo cáo độ bao phủ:

     npm run test -- --coverage
7. **Yêu cầu với thành viên phát triển**
   - Mỗi tính năng mới đều phải có Unit Test tương ứng.
   - Không được merge code lên nhánh chính (main/dev) nếu chưa có kiểm thử hợp lệ.

- Trong quá trình review, reviewer có trách nhiệm kiểm tra test case và coverage.

8. Tổng kết

- Viết kiểm thử đơn vị là trách nhiệm của mỗi lập trình viên trong nhóm. Việc duy trì và thực hiện kiểm thử thường xuyên sẽ giúp hệ thống ổn định, dễ mở rộng và hạn chế lỗi trong quá trình phát triển lâu dài.

IX. WEB API

1. Mô tả

- Web API cho ứng dụng Quản lý Sinh viên cung cấp giao diện RESTful API tuân theo chuẩn OpenAPI 3.0, cho phép truy cập vào các chức năng của hệ thống như Quản lý Sinh viên, Lớp, Môn học, Điểm số, Đăng ký lớp v.v. API giúc các nhà phát triển xây dựng các ứng dụng bên ngoài (Frontend hoặc Backend) để khai thác dữ liệu trong hệ thống.

2. Khả năng API cung cấp

- API Frontend
  - Dùng để hiển thị thông tin lên website hoặc ứng dụng mobile.
  - Cung cấp các API: danh sách lớp, danh sách sinh viên, thông tin chi tiết môn học, xem điểm, lịch sử đăng ký v.v.
- API Backend
  - Truy cập các chức năng admin: Tạo - Cập nhật - Xóa Sinh viên, Quản lý lớp, Phòng ban, Giáo viên, Môn học, Nhập điểm, Thông kê v.v.

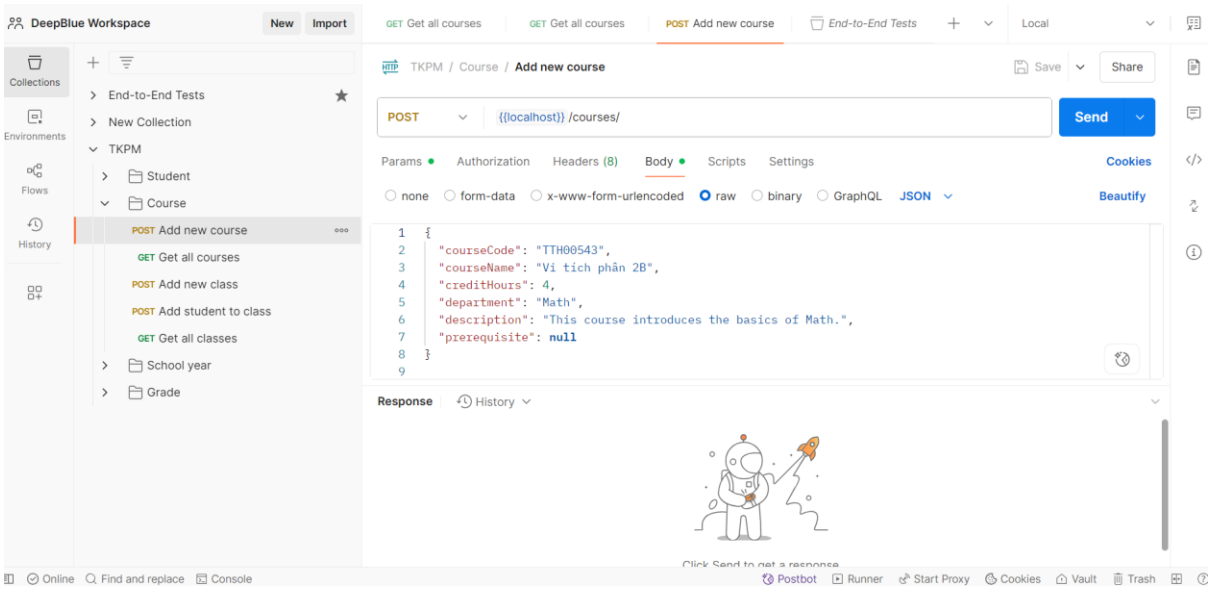
3. Xem danh sách API

- Sau khi khởi động backend, truy cập Swagger UI tại:

  http://localhost:{PORT}/api-docs

4. Kiểm thử API bằng Postman:

- Ta có thể sử dụng Postman để kiểm thử các giao thức GET, POST


DeepBlue Workspace
New
Import
Collections
Environments
Flows
History
End-to-End Tests
New Collection
TKPM
Student
Course
POST Add new course
GET Get all courses
POST Add new class
POST Add student to class
GET Get all classes
School year
Grade
GET Get all courses
GET Get all courses
POST Add new course
End-to-End Tests
Local
TKPM / Course / Add new course
Save
Share
POST
{{localhost}} /courses/
Send
Params
Authorization
Headers (8)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"courseCode": "TTH00543",
"courseName": "Vi tích phân 2B",
"creditHours": 4,
"department": "Math",
"description": "This course introduces the basics of Math.",
"prerequisite": null
}
Response
History
Click Send to get a response
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Vault
Trash