

**TRƯỜNG ĐẠI HỌC TRÀ VINH  
KHOA KỸ THUẬT VÀ CÔNG NGHỆ**



**ISO 9001:2015**

**TRẦN THÁI HƯNG**

**XÂY DỰNG API VÀ HỆ THỐNG CHẤM ĐIỂM  
TRỰC TUYẾN CHO ĐỒ ÁN SINH VIÊN CNTT**

**ĐỒ ÁN TỐT NGHIỆP  
NGÀNH CÔNG NGHỆ THÔNG TIN**

**TRÀ VINH, NĂM 2024**

TRƯỜNG ĐẠI HỌC TRÀ VINH  
KHOA KỸ THUẬT VÀ CÔNG NGHỆ

**XÂY DỰNG API VÀ HỆ THỐNG CHẤM ĐIỂM  
TRỰC TUYẾN CHO ĐỒ ÁN SINH VIÊN CNTT**

**ĐỒ ÁN TỐT NGHIỆP  
NGÀNH CÔNG NGHỆ THÔNG TIN**

Sinh viên: **Trần Thái Hưng**

Lớp: **DA20TTB**

MSSV: **110120031**

GVHD: **TS. Nguyễn Bảo An**

**TRÀ VINH, NĂM 2024**

## LỜI MỞ ĐẦU

Trong bối cảnh hiện nay, việc sử dụng công nghệ thông tin trong giáo dục trở nên cần thiết. Đặc biệt, trong đào tạo ngành Công nghệ Thông tin, việc quản lý và đánh giá các đồ án của sinh viên rất quan trọng để nâng cao chất lượng đào tạo. Đề tài “Xây dựng API và hệ thống chấm điểm trực tuyến cho đồ án sinh viên CNTT” nhằm xây dựng một hệ thống chấm điểm, giúp giảng viên tiết kiệm thời gian và nâng cao độ chính xác trong đánh giá.

Mục tiêu của đề tài này là phát triển một khung chương trình đào tạo với các chuẩn đầu ra được xác định rõ ràng cùng với danh mục môn học tương ứng. Đề tài cũng nhằm xây dựng các rubrics để liên kết các tiêu chí đánh giá với các chuẩn đầu ra của từng môn học, nhằm đảm bảo tính chính xác và nhất quán trong quá trình đánh giá. Bên cạnh đó, đề tài sẽ thiết kế và triển khai giao diện chấm điểm trực tuyến cho giảng viên, với mục tiêu nâng cao tính tiện lợi và hiệu quả trong công tác chấm điểm. Cuối cùng, hệ thống sẽ cung cấp tính năng in phiếu chấm và nhập điểm, hỗ trợ giảng viên trong việc quản lý và lưu trữ kết quả đánh giá một cách hiệu quả.

Đề tài này gồm 5 chương, với nội dung cụ thể như sau:

### **Chương 1: Đặt vấn đề**

Trình bày lý do chọn đề tài, xác định mục tiêu nghiên cứu, nội dung cụ thể sẽ được khai thác, đối tượng nghiên cứu, phạm vi áp dụng và phương pháp tiếp cận được sử dụng.

### **Chương 2: Cơ sở lý thuyết**

Giới thiệu các kiến thức nền tảng cùng các công nghệ dự kiến áp dụng trong đề tài, bao gồm các lĩnh vực lập trình, quản lý cơ sở dữ liệu.

### **Chương 3: Thực hiện hóa nghiên cứu**

Mô tả chi tiết quá trình phát triển hệ thống.

### **Chương 4: Kết quả nghiên cứu**

Chương này tổng hợp các kết quả đạt được từ việc triển khai hệ thống.

### **Chương 5: Kết luận và hướng phát triển**

Đánh giá tổng thể hiệu quả của hệ thống đã xây dựng.

## LỜI CẢM ƠN

Trước tiên, em xin gửi lời cảm ơn sâu sắc đến quý thầy cô và các giảng viên tại Trường Đại học Trà Vinh, đặc biệt là các thầy cô ở Khoa Kỹ thuật & Công nghệ, bộ môn Công nghệ thông tin, đã tạo mọi điều kiện thuận lợi để em hoàn thành bài báo cáo này.

Em cũng xin chân thành cảm ơn thầy Nguyễn Bảo Ân – Giảng viên Khoa Kỹ thuật & Công nghệ, Trường Đại học Trà Vinh, vì sự nhiệt tình và tận tâm trong việc giảng dạy và hỗ trợ em cũng như toàn thể sinh viên trong Khoa.

Do kiến thức và kinh nghiệm còn hạn chế, cùng với thời gian nghiên cứu có hạn, bài báo cáo có thể còn một số thiếu sót. Em rất mong nhận được sự thông cảm và đóng góp ý kiến từ quý thầy cô để em có thể học hỏi và hoàn thiện bản thân hơn.

Cuối cùng, em xin kính chúc quý thầy cô luôn sức khỏe dồi dào và thành công trong công tác giảng dạy

Em xin chân thành cảm ơn!

Trà Vinh, ngày ... tháng ... năm 2024

Sinh viên thực hiện

Trần Thái Hưng

## **NHẬN XÉT** **(Của giảng viên hướng dẫn trong đồ án, khoá luận của sinh viên)**

## **Giảng viên hướng dẫn**

**UBND TỈNH TRÀ VINH  
TRƯỜNG ĐẠI HỌC TRÀ VINH**

# CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM

## Độc lập – Tự do – Hạnh Phúc

# BẢN NHẬN XÉT ĐỒ ÁN, KHÓA LUẬN TỐT NGHIỆP

(Của giảng viên hướng dẫn)

Họ và tên sinh viên: ..... MSSV: .....

Ngành: ..... Khóa: .....

Tên đề tài: .....

Ho và tên Giảng viên hướng dẫn: .....

Chức danh: ..... Hoc vi: .....

## NHÂN XÉT

## 1. Nội dung đề tài:

## 2. Ưu điểm:

.....  
.....  
.....

### 3. Khuyết điểm:

.....  
.....  
.....  
.....

4. Điểm mới đề tài:

.....  
.....  
.....  
.....  
.....

5. Giá trị thực trên đề tài:

.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....

7. Đề nghị sửa chữa bổ sung:

.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....

8. Đánh giá:

.....  
.....  
.....  
.....

Trà Vinh, ngày ..... tháng ..... năm 2024

Giảng viên hướng dẫn

(Ký & ghi rõ họ tên)

## MỤC LỤC

<b>CHƯƠNG 1. ĐẶT VẤN ĐỀ .....</b>	<b>1</b>
1.1. Lý do chọn đề tài.....	1
1.2. Mục tiêu .....	1
1.3. Nội dung.....	2
1.4. Đối tượng và phạm vi nghiên cứu .....	2
1.5. Phương pháp nghiên cứu .....	3
<b>CHƯƠNG 2. CƠ SỞ LÝ THUYẾT .....</b>	<b>4</b>
2.1. Công nghệ backend.....	4
2.1.1. Tổng quan về Node.js .....	4
2.1.2. Tổng quan về Express.js .....	6
2.1.3. Tổng quan về Puppeteer.js.....	7
2.2. Công nghệ Frontend.....	8
2.2.1. Tổng quan React.js .....	8
2.2.2. Tổng quan Tailwind CSS.....	9
2.3. Tổng quan MySQL .....	10
2.4. Tổng quan RESTful API.....	12
2.5. Các khái niệm về chương trình .....	14
2.5.1 Khái niệm về chương trình đào tạo .....	14
2.5.2 Khái niệm về chuẩn đầu ra của chương trình .....	14
2.5.3 Khái niệm về mục tiêu chương trình .....	14
2.5.4 Khái niệm về chuẩn đầu ra môn học .....	15
2.5.5 Khái niệm và Chức năng của Rubric Đánh giá. ....	15
<b>CHƯƠNG 3. HIỆN THỰC HÓA NGHIÊN CỨU .....</b>	<b>16</b>
3.1. Mô tả bài toán .....	16
3.2. Yêu cầu chức năng.....	17

3.3. Đặt tả hệ thống .....	18
3.4. Lược đồ cơ sở dữ liệu .....	22
3.5. Kiến trúc hệ thống.....	33
3.6. Thiết kế API.....	34
3.6.1. Định tuyến API trong Express.js .....	34
3.6.2. Định tuyến API chương trình (program) .....	36
3.6.3. Định tuyến API mục tiêu chương trình (PO).....	37
3.6.4. Định tuyến API chuẩn đầu ra chương trình (PLO).....	38
3.6.5. Định tuyến API mối quan hệ PLO và PO (PLO_PO) .....	39
3.6.6. Định tuyến API môn học (subject) .....	40
3.6.7. Định tuyến API chuẩn đầu ra của môn học (CLO) .....	41
3.6.8. Định tuyến API mối quan hệ PLO và CLO (PLO_CLO).....	42
3.6.9. Định tuyến API chương của môn học (chapter) .....	43
3.6.10. Định tuyến API mối quan hệ chapter và CLO (chapter_CLO) .....	44
3.6.11. Định tuyến API rubric.....	45
3.6.12. Định tuyến API rubric items.....	46
3.6.13. Định tuyến API đánh giá meta (meta assessment) .....	47
3.6.14. Định tuyến API lần đánh giá (assessment) .....	48
3.6.15. Định tuyến API assessment items.....	49
<b>CHƯƠNG 4. KẾT QUẢ NGHIÊN CỨU .....</b>	<b>50</b>
4.1. Chức năng Admin .....	50
4.1.1. Quản lý chương trình đào tạo (QL-01).....	50
4.1.2. Quản lý CDR của chương trình (QL-02).....	53
4.1.3. Quản lý mục tiêu chương trình (QL-03).....	56
4.1.4. Quản lý mối quan hệ giữa PLO và PO (QL-04).....	59
4.1.5. Quản lý môn học (QL-05) .....	60

4.1.6. Quản lý CDR của môn học (QL-06).....	64
4.1.7. Quản lý mối quan hệ giữa CLO và PLO (QL-07) .....	67
4.1.8. Quản lý mối quan hệ giữa CLO và các chương học (QL-08) .....	68
4.1.9. Quản lý các chương học của môn học (QL-09).....	69
4.2. Chức năng giảng viên .....	72
4.2.1. Quản lý bảng tiêu chí (QL-10).....	72
4.2.2. Quản lý các tiêu chí của bảng tiêu chí (QL-11).....	75
4.2.3. Quản lý đánh giá tổng hợp (QL-12) .....	78
4.2.4. Mời giảng viên tham gia đánh giá (QL-13) .....	83
4.2.5. Quản lý quy trình chấm điểm và đánh giá (QL-14).....	84
4.3. Giao diện chấm cho mobile .....	93
<b>CHƯƠNG 5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN .....</b>	<b>94</b>
5.1. Kết luận.....	94
5.2. Hướng phát triển .....	94
<b>DANH MỤC TÀI LIỆU THAM KHẢO.....</b>	<b>96</b>
<b>PHỤ LỤC .....</b>	<b>97</b>

## MỤC LỤC ẢNH

Hình 2.1. Tổng quan Node.js .....	4
Hình 2.2. Tổng quan Express.js .....	6
Hình 2.3. Tổng quan về Puppeteer.js .....	7
Hình 2.4. Tổng quan React.js .....	8
Hình 2.5. Tổng quan Tailwind CSS.....	9
Hình 2.6. Tổng quan MySQL .....	10
Hình 2.7. Tổng quan RESTful API.....	12
Hình 3.1. Sơ đồ use case Giảng viên. ....	18
Hình 3.2. Sơ đồ use case Admin.....	19
Hình 3.3. Lược đồ cơ sở dữ liệu. ....	22
Hình 3.4. Kiến trúc hệ thống.....	33
Hình 3.5. Định tuyến API trong Express.js .....	34
Hình 3.6. Các định tuyến được liên kết đến các router.....	35
Hình 3.7. Định tuyến API chương trình (program) .....	36
Hình 3.8. Định tuyến API mục tiêu chương trình (PO).....	37
Hình 3.9. Định tuyến API chuẩn đầu ra chương trình (PLO).....	38
Hình 3.10. Định tuyến API mối quan hệ PLO và PO (PLO_PO) .....	39
Hình 3.11. Định tuyến API môn học (subject) .....	40
Hình 3.12. Định tuyến API chuẩn đầu ra của môn học (CLO) .....	41
Hình 3.13. Định tuyến API mối quan hệ PLO và CLO (PLO_CLO).....	42
Hình 3.14. Định tuyến API chương của môn học (chapter) .....	43
Hình 3.15. Định tuyến API mối quan hệ chapter và CLO (chapter_CLO) .....	44
Hình 3.16. Định tuyến API rubric.....	45
Hình 3.17. Định tuyến API rubric items .....	46
Hình 3.18. Định tuyến API đánh giá meta (meta assessment) .....	47
Hình 3.19. Định tuyến API lần đánh giá (assessment) .....	48
Hình 3.20. Định tuyến API assessment items.....	49
Hình 4.1. Trang quản lý chương trình đào tạo.....	50
Hình 4.2. Chức năng tạo chương trình.....	50

Hình 4.3. Chức năng cập nhật chương trình .....	51
Hình 4.4. Cửa sổ nhập chương trình bằng Excel .....	52
Hình 4.5. Mẫu nhập liệu chương trình bằng file Excel .....	52
Hình 4.6. Trang quản lý CDR chương trình .....	53
Hình 4.7. Chức năng cập nhật CDR chương trình.....	54
Hình 4.8. Chức năng tạo mới chương trình .....	54
Hình 4.9. Trang quản lý CDR chương trình đã ẩn.....	55
Hình 4.10. Trang quản lý mục tiêu chương trình .....	56
Hình 4.11. Chức năng cập nhật mục tiêu chương trình .....	57
Hình 4.12. Chức năng tạo mới mục tiêu chương trình .....	57
Hình 4.13. Trang quản lý mục tiêu chương trình đã ẩn .....	58
Hình 4.14. Trang quản lý ánh xạ PO và PLO .....	59
Hình 4.15. Trang quản lý môn học .....	60
Hình 4.16. Chức năng tạo mới môn học .....	61
Hình 4.17. Chức năng lọc theo loại .....	61
Hình 4.18. Chức năng lọc theo ngày .....	61
Hình 4.19. Chức năng cập nhật môn học .....	62
Hình 4.20. Cửa sổ nhập môn học bằng Excel .....	62
Hình 4.21. Mẫu nhập liệu môn học bằng file Excel .....	63
Hình 4.22. Trang quản lý môn học đã ẩn.....	63
Hình 4.23. Trang quản lý CDR môn học .....	64
Hình 4.24. Chức năng tạo mới CDR môn học .....	65
Hình 4.25. Chức năng cập nhật CDR môn học .....	65
Hình 4.26. Trang quản lý môn học đã ẩn.....	66
Hình 4.27. Trang quản lý ánh xạ CLO môn công nghệ phần mềm với PLO .....	67
Hình 4.28. Trang quản lý ánh xạ chương với CDR môn học .....	68
Hình 4.29. Trang quản lý chương của môn học .....	69
Hình 4.30. Chức năng thêm mới chương môn học.....	69
Hình 4.31. Chức năng cập nhật chương môn học.....	70
Hình 4.32. Trang quản lý chương môn học đã ẩn .....	71
Hình 4.33. Trang quản lý bảng tiêu chí .....	72
Hình 4.34. Chức năng thêm mới bảng tiêu chí .....	73

Hình 4.35. Chức năng cập nhật bảng tiêu chí .....	73
Hình 4.36. Trang quản lý bảng tiêu chí đã ẩn.....	74
Hình 4.37. Trang quản lý tiêu chí của bảng tiêu chí.....	75
Hình 4.38. Chức năng tạo mới tiêu chí .....	75
Hình 4.39. Chức năng cập nhật tiêu chí.....	76
Hình 4.40. Các chức năng lọc của trang tiêu chí .....	76
Hình 4.41. Trang quản lý tiêu chí đã ẩn.....	77
Hình 4.42. Trang xem tổng quan tiêu chí .....	78
Hình 4.43. Trang quản lý đánh giá .....	78
Hình 4.44. Chức năng thêm mới đánh giá .....	79
Hình 4.45. Chức năng cập nhật đánh giá .....	80
Hình 4.46. PDF mẫu đánh giá tại lớp .....	81
Hình 4.47. Chức năng xem điểm cuối .....	81
Hình 4.48. Trang quản lý đánh giá đã ẩn.....	82
Hình 4.49. Chức năng quản lý phân công.....	83
Hình 4.50. Trang nhận kết quả phân công.....	83
Hình 4.51. Trang quản lý đánh giá điểm cho sinh viên .....	84
Hình 4.52. Cửa sổ cập nhật đề tài bằng Excel .....	84
Hình 4.53. Nội dung bên trong mẫu cập nhật.....	85
Hình 4.54. Các chức năng lọc của trang đánh giá điểm cho sinh viên .....	85
Hình 4.55. Trang đánh giá đơn cho sinh viên.....	86
Hình 4.56. Các tiêu chí đã được giảng viên đánh giá .....	86
Hình 4.57. Hiển thị kết quả đánh giá đơn .....	87
Hình 4.58. In PDF điểm sinh viên .....	88
Hình 4.59. Chức năng chấm theo nhóm .....	89
Hình 4.60. Trang đánh giá theo nhóm .....	89
Hình 4.61. Các tiêu chí đã được giảng viên đánh giá cho nhóm .....	90
Hình 4.62. In PDF cho nhóm sinh viên (2/2).....	90
Hình 4.63. In PDF cho nhóm sinh viên (2/2).....	91
Hình 4.64. Xem biểu đồ phân tích điểm số sinh viên .....	92
Hình 4.65. Giao diện chấm điểm đơn .....	93
Hình 4.66. Giao diện chấm điểm nhóm .....	93

## MỤC LỤC BẢNG

Bảng 3.1. Mô tả Actor.....	19
Bảng 3.2. Mô tả use case .....	19
Bảng 3.3. Mô tả bảng chương trình đào tạo .....	22
Bảng 3.4. Mô tả bảng PO.....	23
Bảng 3.5. Mô tả bảng PLO .....	23
Bảng 3.6. Mô tả bảng ánh xạ PLO với PO .....	24
Bảng 3.7. Mô tả bảng môn học .....	24
Bảng 3.8. Mô tả bảng CLO .....	25
Bảng 3.9. Mô tả bảng chương của môn học .....	26
Bảng 3.10. Mô tả bảng ánh xạ CLO với chương .....	26
Bảng 3.11. Mô tả bảng ánh xạ CLO với PLO .....	26
Bảng 3.12. Mô tả bảng tiêu chí đánh giá rubric.....	27
Bảng 3.13. Mô tả bảng tiêu chí đánh giá .....	27
Bảng 3.14. Mô tả bảng đánh giá meta .....	28
Bảng 3.15. Mô tả bảng lần đánh giá .....	29
Bảng 3.16. Mô tả bảng thành phần của lần đánh giá .....	29
Bảng 3.17. Mô tả bảng giảng viên .....	30
Bảng 3.18. Mô tả bảng lớp.....	30
Bảng 3.19. Mô tả bảng lớp môn học.....	31
Bảng 3.20. Mô tả bảng tuyển sinh lớp môn học .....	32
Bảng 3.21. Mô tả bảng sinh viên .....	32

## **DANH MỤC TỪ VIẾT TẮT**

Từ viết tắt	Ý nghĩa
API	Application programming interface
CLO	Course Learning Outcome
PLO	Program Learning Outcome
PO	Program Objective
CDR	Chuẩn đầu ra
CNTT	Công nghệ thông tin
CT	Chương trình
GVGD	Giảng viên giảng dạy
DOM	Document object model
HTTP	Hypertext Transfer Protocol
CSS	Cascading Style Sheets
CRUD	Create, Read, Update, Delete
UI	User Interface
I/O	Input/Output
ECMAScript	European Computer Manufacturers Association Script

## CHƯƠNG 1. ĐẶT VẤN ĐỀ

### 1.1. Lý do chọn đề tài

Đánh giá đồ án của sinh viên ngành CNTT là một yếu tố then chốt trong việc xác định trình độ hiểu biết và kỹ năng thực hành của sinh viên. Quy trình đánh giá không chỉ phản ánh năng lực chuyên môn mà còn cung cấp thông tin quan trọng cho việc điều chỉnh và cải thiện chương trình đào tạo. Đánh giá chính xác giúp nhà trường nhận diện các kỹ năng đã đạt yêu cầu cũng như các lĩnh vực cần cải thiện, từ đó hỗ trợ định hướng công tác đào tạo và phát triển kỹ năng cho sinh viên.

Hiện nay, việc chấm điểm đồ án sinh viên trên giấy đang gặp nhiều thách thức và hạn chế. Trước hết, giảng viên phải ghi nhận điểm số và nhận xét thủ công trên từng phiếu giấy, điều này tiềm ẩn nguy cơ thất lạc hoặc sai sót trong quá trình lưu trữ và tổng hợp thông tin. Bên cạnh đó, việc tra cứu và tổng hợp kết quả từ nhiều phiếu chấm tốn nhiều thời gian, gây cản trở cho quá trình phân tích và đưa ra quyết định đánh giá chính xác.

Để khắc phục những vấn đề này, việc xây dựng một hệ thống chấm điểm trực tuyến là cần thiết. Hệ thống mới sẽ nâng cao tính minh bạch và khả năng theo dõi kết quả, đồng thời giúp giảng viên tiết kiệm thời gian và công sức trong việc quản lý và tổng hợp dữ liệu. Hệ thống cũng sẽ hỗ trợ nhiều phương pháp đánh giá, bao gồm đánh giá theo nhóm, phản ánh toàn diện hơn khả năng và kỹ năng của sinh viên. Việc áp dụng các công nghệ hiện đại như ReactJS và ExpressJS đảm bảo hiệu suất cao, cải thiện trải nghiệm người dùng và tạo điều kiện thuận lợi cho việc bảo trì và mở rộng hệ thống trong tương lai.

Với những lợi ích nổi bật như cải thiện tính minh bạch, tiết kiệm thời gian cho giảng viên, quản lý dữ liệu hiệu quả, hỗ trợ đánh giá nhóm và tích hợp công nghệ hiện đại, việc chọn đề tài “Xây dựng API và hệ thống chấm điểm trực tuyến cho đồ án sinh viên CNTT” là hoàn toàn hợp lý và cần thiết. Đề tài này không chỉ giải quyết các thách thức hiện tại mà còn mở ra cơ hội cải tiến quy trình đánh giá, nâng cao chất lượng đào tạo và hỗ trợ sự phát triển toàn diện của sinh viên.

### 1.2. Mục tiêu

Phát triển một khung chương trình đào tạo với các CDR chương trình rõ ràng và danh mục môn học tương ứng với các CDR đó.

Xây dựng các rubrics để gắn kết các tiêu chí đánh giá với các chuẩn đầu ra của từng môn học, nhằm đảm bảo tính chính xác và nhất quán trong quá trình đánh giá.

Cung cấp tính năng in phiếu chấm và nhập điểm chấm, giúp giảng viên có thể dễ dàng quản lý và lưu trữ kết quả đánh giá.

Thiết kế và triển khai giao diện chấm điểm trực tuyến cho giảng viên, dựa trên các phiếu chấm, để nâng cao tính tiện lợi và hiệu quả trong việc chấm điểm.

### **1.3. Nội dung**

Nghiên cứu quy trình hiện tại của TVU trong đánh giá môn học, bao gồm các bước, công cụ và tiêu chuẩn, nhằm xác định các điểm mạnh và điểm yếu.

Phát triển ứng dụng web cho hệ thống đánh giá, hỗ trợ các tính năng chấm điểm, quản lý dữ liệu, và tra cứu kết quả trực tuyến.

Phát triển giao diện chấm cho mobile.

Thiết kế và triển khai RESTful API để kết nối các thành phần của hệ thống đánh giá, ứng dụng web và cơ sở dữ liệu.

### **1.4. Đối tượng và phạm vi nghiên cứu**

#### **Đối tượng nghiên cứu:**

Các phương pháp và công cụ được sử dụng trong đánh giá môn học tại TVU, bao gồm các bước, tiêu chuẩn và cách thức triển khai.

Các khái niệm và cấu trúc của chương trình đào tạo, đặc biệt là các CDR và mục tiêu học tập liên quan đến các môn học.

Các yếu tố kỹ thuật và chức năng cần thiết để phát triển ứng dụng di động và web cho hệ thống đánh giá, cũng như các RESTful API hỗ trợ kết nối và quản lý dữ liệu.

#### **Phạm vi nghiên cứu:**

Phát triển và triển khai các ứng dụng này chỉ trong bối cảnh của hệ thống đánh giá đồ án môn học của TVU, không bao gồm các ứng dụng hay hệ thống ngoài phạm vi này.

Thiết kế và triển khai API sẽ tập trung vào việc hỗ trợ kết nối và quản lý dữ liệu trong hệ thống đánh giá của TVU, không mở rộng ra các hệ thống khác.

## **1.5. Phương pháp nghiên cứu**

**Nghiên cứu lý thuyết:** Đánh giá các khái niệm cơ bản và nguyên tắc thiết kế của backend và frontend trong phát triển hệ thống. Phân tích các công nghệ, công cụ, và phương pháp được sử dụng để xây dựng và tối ưu hóa các thành phần phía máy chủ và phía người dùng, nhằm đảm bảo hiệu quả và tính khả thi của hệ thống.

### **Nghiên cứu thực nghiệm:**

Tiến hành trải nghiệm quy trình chấm điểm hiện tại tại TVU để đánh giá và phân tích các phương pháp và quy trình đang được áp dụng. Qua đó, thu thập dữ liệu từ giảng viên và sinh viên để nhận diện các vấn đề và nhu cầu cụ thể trong hệ thống chấm điểm, từ đó đề xuất các cải tiến phù hợp.

**Phát triển ứng dụng:** Thiết kế, lập trình, và triển khai các ứng dụng di động và web cho hệ thống đánh giá. Thực hiện thử nghiệm và đánh giá các ứng dụng này để đảm bảo tính năng và hiệu suất đáp ứng yêu cầu của người dùng.

**Xây dựng và kiểm thử RESTful API:** Thiết kế và triển khai RESTful API để hỗ trợ kết nối giữa các thành phần của hệ thống. Thực hiện kiểm thử API để đảm bảo khả năng hoạt động ổn định và hiệu quả trong việc quản lý và truyền tải dữ liệu.

**Đánh giá và cải tiến:** Thu thập phản hồi từ người dùng (giảng viên và sinh viên) về các ứng dụng và API. Phân tích phản hồi để thực hiện các điều chỉnh và cải tiến cần thiết, đảm bảo hệ thống đánh giá hoạt động hiệu quả và đáp ứng nhu cầu thực tế.

## CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

### 2.1. Công nghệ backend

#### 2.1.1. Tổng quan về Node.js

Node.js là một môi trường thực thi JavaScript không chỉ trên trình duyệt mà còn trên máy chủ và các thiết bị khác. Được xây dựng trên trình thực thi JavaScript V8 của Google, Node.js cho phép chạy mã JavaScript trực tiếp trên phần cứng máy tính, nhờ vào khả năng biên dịch mã JavaScript sang mã máy của V8. Điều này mở rộng khả năng của JavaScript, cho phép thực hiện các chức năng như quản lý hệ thống tệp và thao tác với các tệp, điều mà trình duyệt web không hỗ trợ.



[Nguồn ảnh](#)

Hình 2.1. Tổng quan Node.js

#### Một số câu lệnh cơ bản trong Node.js:

- node <filename>: Thực thi chương trình từ tệp chỉ định.
- npm init: Tạo dự án mới và tệp package.json.
- npm install <packageName>: Cài đặt gói từ npm.
- npm install -g <packageName>: Cài đặt gói toàn cầu.
- require('<moduleName>'): Nhập mô-đun trong tệp Node.js.

- `console.log()`: In ra màn hình console.
- `process.env`: Truy cập biến môi trường.
- `module.exports`: Xuất mô-đun cho các tệp khác.
- `__dirname`: Lấy đường dẫn thư mục hiện tại.
- `fs.readFile()`: Đọc dữ liệu từ tệp không đồng bộ.

### **Ưu điểm:**

- Xử lý không đồng bộ: Node.js hoạt động trong một quy trình đơn mà không chẵn mã JavaScript, với I/O không đồng bộ tích hợp sẵn. [1]
- Xử lý I/O không chẵn: Node.js không chẵn luồng trong các thao tác I/O như đọc mạng hoặc truy cập cơ sở dữ liệu, giúp tối ưu hóa hiệu suất. [1]
- Xử lý hàng nghìn kết nối: Có khả năng xử lý hàng nghìn kết nối đồng thời với một máy chủ duy nhất, nâng cao hiệu suất. [1]
- Chia sẻ ngôn ngữ: Lập trình viên frontend có thể sử dụng JavaScript cho cả phía máy chủ và client mà không cần học ngôn ngữ mới. [1]
- Hỗ trợ ECMAScript mới: Cho phép sử dụng các tiêu chuẩn ECMAScript mới mà không cần chờ cập nhật trình duyệt. [1]
- Hỗ trợ module CommonJS và ES: Từ Node.js v12, hỗ trợ cả `require()` và `import` cho linh hoạt trong quản lý mô-đun.

### **Nhược điểm:**

Một số module và thư viện không được duy trì đồng đều, dẫn đến vấn đề tương thích và sự không đồng nhất, gây thách thức cho tính ổn định và bảo trì hệ thống.

### **Các ứng dụng của Node.js:**

- Ứng dụng web: Từ các trang web đơn giản đến các ứng dụng web phức tạp.
- Ứng dụng thời gian thực: Chat, thông báo và trò chơi trực tuyến.
- API và dịch vụ web: Xây dựng API RESTful và các dịch vụ web.

### **2.1.2. Tổng quan về Express.js**

Express.js là một framework cho Node.js, giúp đơn giản hóa việc phát triển API và ứng dụng web. Nó tổ chức ứng dụng thông qua lớp trung gian (Middleware) và hệ thống định tuyến (routing), tối ưu hóa xử lý yêu cầu HTTP và hiển thị HTML động. Express.js thường kết hợp với Node.js trong stack MEAN (MongoDB, Express.js, Angular, Node.js), và hỗ trợ xây dựng ứng dụng trang đơn (SPA) và các ứng dụng thời gian thực như WebSocket và WebRTC. [2] [3]



[Nguồn ảnh](#)

*Hình 2.2. Tổng quan Express.js*

#### **Ưu điểm:**

- Đơn giản và dễ sử dụng: Cú pháp dễ hiểu giúp lập trình viên nhanh chóng triển khai các tính năng.
- Hỗ trợ Middleware: Cung cấp hệ thống Middleware linh hoạt cho xác thực, ghi log, nén dữ liệu và xử lý lỗi. [3]
- Hiệu suất cao: Xây dựng trên Node.js, xử lý nhanh các yêu cầu web đồng thời và mở rộng tốt.
- Khả năng mở rộng: Hỗ trợ nhiều kết nối đồng thời nhờ mô hình I/O không chặn và Event-driven. [2]

#### **Nhược điểm:**

- Hạn chế tính năng tích hợp: Cung cấp các tính năng cơ bản, nhưng các chức năng phức tạp thường yêu cầu thêm module bổ sung. [2]

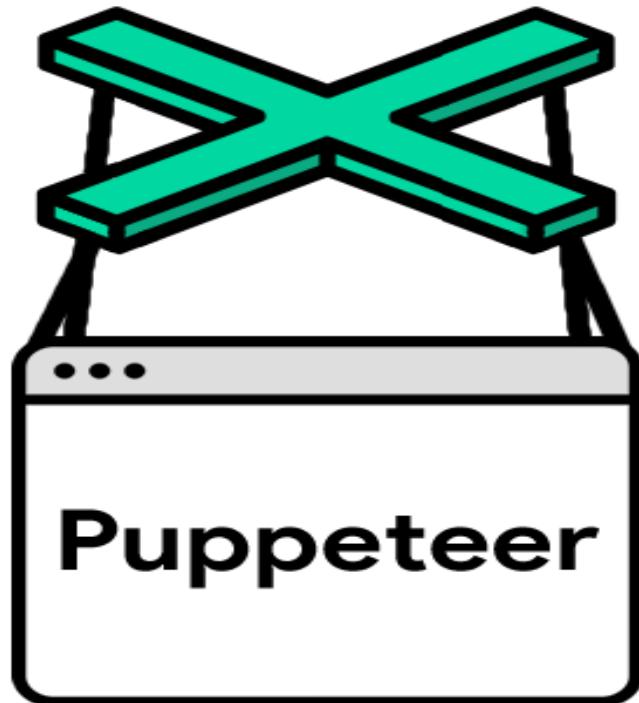
- Khó khăn với Middleware: Việc hiểu và triển khai Middleware có thể phức tạp, đặc biệt với những người mới, yêu cầu kiến thức sâu về tích hợp và tương tác với Middleware. [3]

### Các ứng dụng của Express.js

- Ứng dụng web: Xây dựng các trang web và ứng dụng web động.
- API RESTful: Tạo và quản lý các API cho giao tiếp dữ liệu.
- Ứng dụng thời gian thực: Hỗ trợ các ứng dụng cần tính năng thời gian thực như chat hoặc thông báo.

#### 2.1.3. Tổng quan về Puppeteer.js

Puppeteer là một thư viện JavaScript dành cho Node.js được phát triển bởi Google. Thư viện này cung cấp một API cấp cao để điều khiển trình duyệt Chrome hoặc Chromium thông qua giao thức DevTools Protocol. Puppeteer chủ yếu được sử dụng để tự động hóa trình duyệt, thực hiện kiểm thử giao diện người dùng, thu thập dữ liệu từ web (web scraping), và thực hiện các tác vụ như chụp ảnh màn hình hoặc tạo PDF từ các trang web.



[Nguồn ảnh](#)

Hình 2.3. Tổng quan về Puppeteer.js

Các Đặc Điểm Chính của Puppeteer:

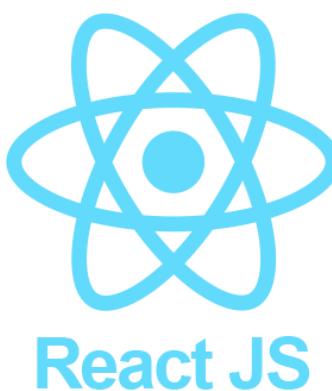
- Tạo PDF từ Nội Dung HTML: Puppeteer hỗ trợ tạo PDF trực tiếp từ nội dung HTML, bao gồm cả trang web và mã HTML nội bộ, với khả năng tùy chỉnh định dạng và cấu trúc của PDF.
- Tùy Chỉnh Kích Thước và Định Dạng: Có thể chỉ định kích thước trang PDF qua các tùy chọn như format (ví dụ: 'A4', 'Letter') và tùy chỉnh kích thước trang bằng width và height.
- Xoay PDF: Puppeteer cho phép điều chỉnh hướng của trang PDF bằng tùy chọn landscape để xoay theo chiều ngang hoặc giữ định dạng dọc.
- In Nền: Tùy chọn printBackground in các phần tử nền của trang HTML, bao gồm hình ảnh và màu nền, đảm bảo PDF phản ánh chính xác nội dung gốc.

Hỗ Trợ CSS và JavaScript: Hỗ trợ đầy đủ CSS và JavaScript, cho phép áp dụng kiểu dáng và hiệu ứng động khi tạo PDF, đảm bảo tài liệu phản ánh chính xác giao diện và chức năng của trang web.

## 2.2. Công nghệ Frontend

### 2.2.1. Tổng quan React.js

React.js là một thư viện JavaScript được sử dụng để phát triển giao diện người dùng, đặc biệt là trong các ứng dụng web đơn trang (Single Page Applications - SPA). Được phát triển bởi Facebook, React.js nổi bật với kiến trúc thành phần (components), cho phép tái sử dụng mã nguồn và quản lý giao diện một cách hiệu quả và linh hoạt. [4]



[Nguồn ảnh](#)

Hình 2.4. Tổng quan React.js

### **Cơ chế hoạt động của React.js:**

Kiến trúc thành phần (Component-Based Architecture): React chia giao diện thành các thành phần nhỏ, độc lập, và tái sử dụng, bao gồm HTML, CSS, và JavaScript, giúp xây dựng giao diện phức tạp từ các khối đơn giản.

DOM ảo (Virtual DOM): React sử dụng DOM ảo để tối ưu hóa việc cập nhật giao diện, chỉ thay đổi những phần cần thiết khi trạng thái ứng dụng thay đổi, từ đó cải thiện hiệu suất.

Liên kết dữ liệu một chiều (One-Way Data Binding): Dữ liệu trong React truyền từ component cha xuống component con qua props, giúp kiểm soát luồng dữ liệu và giảm lỗi.

Quản lý trạng thái (State Management): React quản lý dữ liệu thay đổi theo thời gian qua state, tự động cập nhật giao diện khi state thay đổi để đảm bảo tính nhất quán.

#### **2.2.2. Tổng quan Tailwind CSS**

Tailwind CSS là một framework CSS theo hướng tiện ích (utility-first CSS framework) được thiết kế để tối ưu hóa quá trình xây dựng giao diện người dùng với tính linh hoạt và hiệu suất cao. Công cụ này cho phép các nhà phát triển xây dựng giao diện nhanh chóng và hiệu quả mà không cần phải viết mã CSS tùy chỉnh từ đầu.



[Nguồn ảnh](#)

*Hình 2.5. Tổng quan Tailwind CSS*

### **Ưu điểm của Tailwind CSS:**

- Tiện ích: Cung cấp các lớp tiện ích (utility classes) cho phép xây dựng giao diện mà không cần viết CSS tùy chỉnh.
- Thiết kế mobile: Cung cấp lớp điều kiện cho các kích thước màn hình khác nhau, hỗ trợ thiết kế đáp ứng (responsive design).
- Hiệu suất cao: Giảm kích thước tệp CSS thông qua việc loại bỏ các lớp không sử dụng, từ đó cải thiện hiệu suất tải trang.
- Tích hợp dễ dàng: Có khả năng tích hợp tốt với các công cụ phát triển như PostCSS và Webpack.

### **2.3. Tổng quan MySQL**

MySQL là hệ quản trị cơ sở dữ liệu quan hệ mã nguồn mở do Oracle Corporation phát triển và duy trì. Sử dụng ngôn ngữ SQL để quản lý và truy xuất dữ liệu, MySQL nổi bật với hiệu suất cao, khả năng mở rộng linh hoạt và hỗ trợ nhiều tính năng cơ sở dữ liệu hiện đại. Điều này làm cho MySQL trở thành lựa chọn phổ biến cho các ứng dụng web và hệ thống thông tin, từ quy mô nhỏ đến lớn.



[Nguồn ảnh](#)

*Hình 2.6. Tổng quan MySQL*

## Lịch sử phát triển

MySQL, một trong những hệ quản trị cơ sở dữ liệu mã nguồn mở phổ biến nhất, đã trải qua một quá trình phát triển đáng chú ý từ khi ra đời vào giữa thập niên 1990. MySQL được phát triển lần đầu tiên bởi công ty MySQL AB tại Thụy Điển vào năm 1995, với mục tiêu cung cấp một hệ quản trị cơ sở dữ liệu hiệu quả, dễ sử dụng và có hiệu năng cao. Phiên bản đầu tiên của MySQL được xây dựng dựa trên mã nguồn từ ISAM, một thư viện lưu trữ dữ liệu được sử dụng trong các hệ thống quản lý cơ sở dữ liệu khác.

Trong suốt những năm 2000, MySQL đã nhanh chóng trở thành lựa chọn hàng đầu cho các ứng dụng web, nhờ vào khả năng tích hợp tốt với các ngôn ngữ lập trình như PHP và hỗ trợ các ứng dụng mã nguồn mở khác như Apache và Linux, tạo thành một phần quan trọng của LAMP stack. Năm 2008, Sun Microsystems mua lại MySQL AB với giá 1 tỷ USD, và đến năm 2010, Oracle Corporation tiếp quản Sun Microsystems, qua đó nắm quyền kiểm soát MySQL.

### Kiến trúc logic của MySQL:

- Quản lý kết nối và bảo mật (Connection Management and Security): Cách MySQL xử lý các kết nối của client, bao gồm việc xác thực người dùng và quản lý các luồng kết nối. [5]
- Tối ưu hóa và thực thi (Optimization and Execution): Cách MySQL tối ưu hóa các truy vấn thông qua việc xây dựng cây phân tích và áp dụng các kỹ thuật tối ưu hóa. [5]
- Kiểm soát đồng thời (Concurrency Control): Cơ chế kiểm soát đồng thời trong MySQL, bao gồm việc quản lý khóa đọc/ghi để đảm bảo tính toàn vẹn dữ liệu khi có nhiều truy vấn đồng thời. [5]

## Ứng dụng của MySQL

MySQL được ứng dụng rộng rãi trong nhiều lĩnh vực nhờ khả năng mở rộng và hiệu suất cao. Trong các ứng dụng web lớn như Facebook, Twitter và YouTube, MySQL xử lý và quản lý khối lượng dữ liệu lớn liên quan đến người dùng, tương tác và nội dung. Nó cũng đóng vai trò quan trọng trong các hệ thống thương mại điện tử như Shopify và nền tảng quản lý nội dung như WordPress.com, nơi lưu trữ dữ liệu sản phẩm, đơn hàng và nội

dung trang web.

## 2.4. Tổng quan RESTful API

### Khái niệm

RESTful API là một kiểu API dựa trên nguyên tắc kiến trúc REST. RESTful API sử dụng các phương thức HTTP và các tiêu chuẩn web để thực hiện việc trao đổi dữ liệu giữa client và server một cách hiệu quả và rõ ràng.



[Nguồn ảnh](#)

Hình 2.7. Tổng quan RESTful API

### Nguyên tắc cơ bản:

**Stateless:** Mỗi yêu cầu từ client đến server phải chứa toàn bộ thông tin cần thiết để server xử lý mà không cần lưu trữ trạng thái giữa các yêu cầu, giúp giảm tải cho server và đơn giản hóa quản lý hệ thống. [6]

**Cacheable:** Phản hồi từ server nên được đánh dấu để cho phép client hoặc proxy lưu trữ và sử dụng lại, nhằm giảm tải cho server và cải thiện hiệu suất hệ thống. [6]

**Uniform Interface:** Tất cả giao diện giữa client và server phải tuân thủ tiêu chuẩn chung, giúp đơn giản hóa phát triển và sử dụng API bằng cách cung cấp các phương thức truy cập và thao tác dữ liệu đồng nhất. [6]

**Layered System:** Kiến trúc RESTful có thể bao gồm nhiều lớp, như load balancers và proxies, cho phép hệ thống mở rộng và bảo mật tốt hơn thông qua việc tổ chức các lớp độc lập. [6]

## **Cấu trúc của RESTful API:**

Tài nguyên (Resources): Tài nguyên là các đối tượng hoặc dữ liệu mà API cung cấp, mỗi tài nguyên được đại diện bởi một URL. Ví dụ: /programs, / programs /1.

Phương thức HTTP: RESTful API sử dụng các phương thức HTTP tiêu chuẩn để thực hiện các thao tác CRUD trên tài nguyên:

Định dạng Dữ liệu (Data Format): Dữ liệu trao đổi giữa client và server thường được định dạng bằng JSON (JavaScript Object Notation) hoặc XML (eXtensible Markup Language). JSON là định dạng phổ biến nhất trong RESTful API nhờ tính đơn giản và dễ đọc.

Giao diện Đối tượng (Resource Representation): Dữ liệu trao đổi qua API có thể được biểu diễn dưới nhiều dạng khác nhau, nhưng thường là JSON hoặc XML. Representation chứa dữ liệu của tài nguyên và có thể bao gồm các thuộc tính và liên kết liên quan đến tài nguyên.

Trạng thái Không Lưu (Stateless): Mỗi yêu cầu từ client đến server phải chứa tất cả thông tin cần thiết để server xử lý yêu cầu. Server không lưu trữ trạng thái của client giữa các yêu cầu.

## **Thành phần của RESTful API:**

Địa chỉ URL (Endpoints): Các URL đại diện cho các tài nguyên và được cấu trúc theo cách rõ ràng và có thể mở rộng. Ví dụ: <https://api.example.com/users> cho tài nguyên người dùng.

Hình thức (Methods): Các phương thức HTTP được sử dụng để thực hiện các thao tác CRUD trên tài nguyên:

- GET để lấy thông tin tài nguyên.
- POST để tạo tài nguyên mới.
- PUT để cập nhật tài nguyên hiện có.
- DELETE để xóa tài nguyên.

Tiêu đề (Headers): Các tiêu đề HTTP cung cấp thông tin bổ sung về yêu cầu hoặc phản hồi, chẳng hạn như loại dữ liệu (Content-Type), mã trạng thái (Status Code), và thông

tin xác thực (Authorization).

Thân (Body): Đối với các phương thức POST và PUT, thân của yêu cầu chứa dữ liệu cần thiết để tạo mới hoặc cập nhật tài nguyên. Đối với các phương thức GET và DELETE, thân thường không cần thiết.

Mã Trạng Thái (Status Codes): Các mã trạng thái HTTP được sử dụng để thông báo kết quả của yêu cầu. Ví dụ:

- 200 OK: Yêu cầu thành công.
- 201 Created: Tài nguyên mới đã được tạo.
- 204 No Content: Xóa tài nguyên thành công, không có dữ liệu trả về.
- 400 Bad Request: Yêu cầu không hợp lệ.
- 404 Not Found: Tài nguyên không tìm thấy.

## 2.5. Các khái niệm về chương trình.

### 2.5.1 Khái niệm về chương trình đào tạo

Chương trình đào tạo là một cấu trúc hệ thống các môn học và hoạt động giáo dục được tổ chức có chủ đích nhằm trang bị cho người học kiến thức, kỹ năng và năng lực cần thiết trong một lĩnh vực học thuật nhất định. Chương trình này được xây dựng dựa trên các mục tiêu và CDR, đảm bảo người học có nền tảng lý thuyết vững chắc, phát triển kỹ năng chuyên môn, tư duy phản biện và các phẩm chất đáp ứng yêu cầu của thị trường lao động.

### 2.5.2 Khái niệm về chuẩn đầu ra của chương trình

Chuẩn đầu ra của chương trình, hay **Program Learning Outcomes (PLO)**, là tập hợp các kết quả học tập mà người học dự kiến đạt được sau khi hoàn thành toàn bộ chương trình đào tạo. PLO mô tả chi tiết các kỹ năng, kiến thức và phẩm chất mà sinh viên sẽ sở hữu sau khi tốt nghiệp, đồng thời đóng vai trò là tiêu chí đánh giá hiệu quả và chất lượng của chương trình. PLO cũng là cơ sở để nhà trường thiết kế chương trình học phù hợp với nhu cầu phát triển của ngành nghề và xã hội.

### 2.5.3 Khái niệm về mục tiêu chương trình

Mục tiêu chương trình, hay **Program Objectives (PO)**, là các mục tiêu dài hạn mà

chương trình đào tạo hướng tới nhằm phát triển năng lực cốt lõi cho sinh viên sau khi tốt nghiệp. PO phản ánh kỳ vọng về khả năng nghề nghiệp của sinh viên, bao gồm năng lực làm việc độc lập, lãnh đạo và phát triển nghề nghiệp, cũng như đóng góp cho cộng đồng. Những mục tiêu này định hình chương trình đào tạo, đảm bảo sinh viên được trang bị các phẩm chất và kỹ năng cần thiết cho sự nghiệp và cuộc sống.

#### **2.5.4 Khái niệm về chuẩn đầu ra môn học**

Chuẩn đầu ra môn học, hay **Course Learning Outcomes (CLO)**, là các kết quả học tập mong đợi mà sinh viên sẽ đạt được khi hoàn thành một môn học cụ thể. CLO mô tả chi tiết kiến thức, kỹ năng và năng lực mà sinh viên tích lũy được, góp phần vào việc đạt các chuẩn đầu ra của chương trình. CLO không chỉ định hướng cho sinh viên trong quá trình học tập mà còn là cơ sở để giảng viên thiết kế hoạt động giảng dạy và đánh giá, đảm bảo sự gắn kết giữa mục tiêu môn học và toàn bộ chương trình đào tạo.

#### **2.5.5 Khái niệm và Chức năng của Rubric Đánh giá.**

**Rubric đánh giá** là công cụ đánh giá được xây dựng từ các tiêu chí và thang đánh giá rõ ràng, giúp giảng viên đánh giá kết quả học tập của sinh viên một cách công bằng, nhất quán và hệ thống. Rubric thường được sử dụng trong giáo dục đại học để đánh giá các bài tập, dự án hoặc luận văn, xác định mức độ đạt chuẩn của sinh viên đối với từng yêu cầu học tập cụ thể.

##### **Chức năng của rubric đánh giá:**

- Hướng dẫn sinh viên hiểu rõ yêu cầu và tiêu chuẩn đánh giá.
- Hỗ trợ giảng viên đánh giá hiệu quả, công bằng và cung cấp phản hồi cụ thể.
- Giúp nhà trường kiểm soát và cải tiến chất lượng giảng dạy.

##### **Phân loại rubric đánh giá:**

- Rubric Tổng Hợp (Định Tính): Đưa ra đánh giá tổng quan theo các tiêu chí chung, phù hợp cho các bài viết và thuyết trình.
- Rubric Phân Tích (Định Lượng): Đánh giá chi tiết dựa trên các tiêu chí cụ thể, phù hợp cho các bài kiểm tra phân tích từng kỹ năng

## **CHƯƠNG 3. HIỆN THỰC HÓA NGHIÊN CỨU**

### **3.1. Mô tả bài toán**

Hệ thống chấm điểm trực tuyến cho đồ án sinh viên CNTT được thiết kế nhằm tối ưu hóa quy trình đánh giá và quản lý đồ án của sinh viên, phục vụ hai nhóm người dùng chính: Admin và Giảng viên.

#### **Vai trò của Admin:**

Admin không chỉ đảm nhận vai trò của một giảng viên mà còn có quyền quản lý toàn diện chương trình đào tạo. Cụ thể, Admin có quyền truy cập vào tất cả các chức năng mà giảng viên được phép sử dụng, đồng thời được cấp thêm quyền quản lý chương trình đào tạo. Trong vai trò này, Admin có trách nhiệm xác định và quản lý các PLO cũng như các PO. Để đảm bảo sự liên kết chặt chẽ giữa các mục tiêu học tập và mục tiêu chương trình, Admin còn quản lý mối quan hệ giữa PLO và PO thông qua các bảng ánh xạ tương ứng.

Ngoài ra, với quyền hạn quản lý môn học, Admin có khả năng tạo mới, chỉnh sửa và xóa các môn học trong chương trình đào tạo. Mỗi môn học bao gồm các CLO và các chương học liên quan. Trong trường hợp này, giảng viên chịu trách nhiệm quản lý mối quan hệ giữa CLO và PLO nhằm đảm bảo rằng các CLO hỗ trợ hiệu quả cho các PLO. Đồng thời, giảng viên cũng quản lý mối quan hệ giữa CLO và các chương học để đảm bảo sự nhất quán trong nội dung giảng dạy của môn học.

#### **Vai trò của giảng viên trong quy trình chấm điểm và đánh giá:**

Trong quy trình chấm điểm và đánh giá, giảng viên bắt đầu bằng việc tạo bảng tiêu chí (Rubric) để đánh giá đồ án của sinh viên. Sau khi bảng tiêu chí được thiết lập, giảng viên sẽ tiếp tục thêm nội dung cho tiêu chí và chọn điểm cho tiêu chí cụ thể. Để tạo một tiêu chí, giảng viên cần lựa chọn CLO tương ứng, sau đó chọn PLO từ ánh xạ giữa CLO và PLO, và cuối cùng là chọn chương học từ ánh xạ giữa CLO và chương học.

Khi thực hiện đánh giá cho sinh viên, giảng viên chọn lớp và môn học cần đánh giá, lựa chọn bảng tiêu chí đã được thiết lập và thêm mô tả chung cho đánh giá. Giảng viên cũng có thể mời các đồng nghiệp khác tham gia vào quá trình đánh giá. Quản lý các lần đánh giá bao gồm việc theo dõi điểm số của các giảng viên được mời chấm điểm và quản

lý quy trình mời thêm các giảng viên khác tham gia đánh giá. Các giảng viên được mời sẽ nhận liên kết đến trang chấm điểm của lớp. Trên trang này, họ có thể thực hiện các thao tác như chấm điểm theo nhóm hoặc cá nhân, chỉnh sửa kết quả nếu có sai sót, và in điểm theo cá nhân hoặc nhóm. Giảng viên chính có quyền cập nhật đề tài cho sinh viên và xem điểm cuối cùng sau khi các lần đánh giá hoàn tất.

Tóm tắt các mã quản lý tương ứng cho các chức năng:

**QL-01: Quản lý chương trình đào tạo.**

**QL-02: Quản lý PLO.**

**QL-03: Quản lý PO.**

**QL-04: Quản lý ánh xạ giữa PLO và PO.**

**QL-05: Quản lý môn học.**

**QL-06: Quản lý CLO.**

**QL-07: Quản lý ánh xạ giữa CLO và PLO.**

**QL-08: Quản lý ánh xạ giữa CLO và các chương học.**

**QL-09: Quản lý các chương học của môn học.**

**QL-10: Tạo và quản lý bảng tiêu chí.**

**QL-11: Quản lý các tiêu chí của bảng tiêu chí.**

**QL-12: Tạo và quản lý đánh giá tổng hợp.**

**QL-13: Mời giảng viên tham gia đánh giá.**

**QL-14: Quản lý quy trình chấm điểm và đánh giá.**

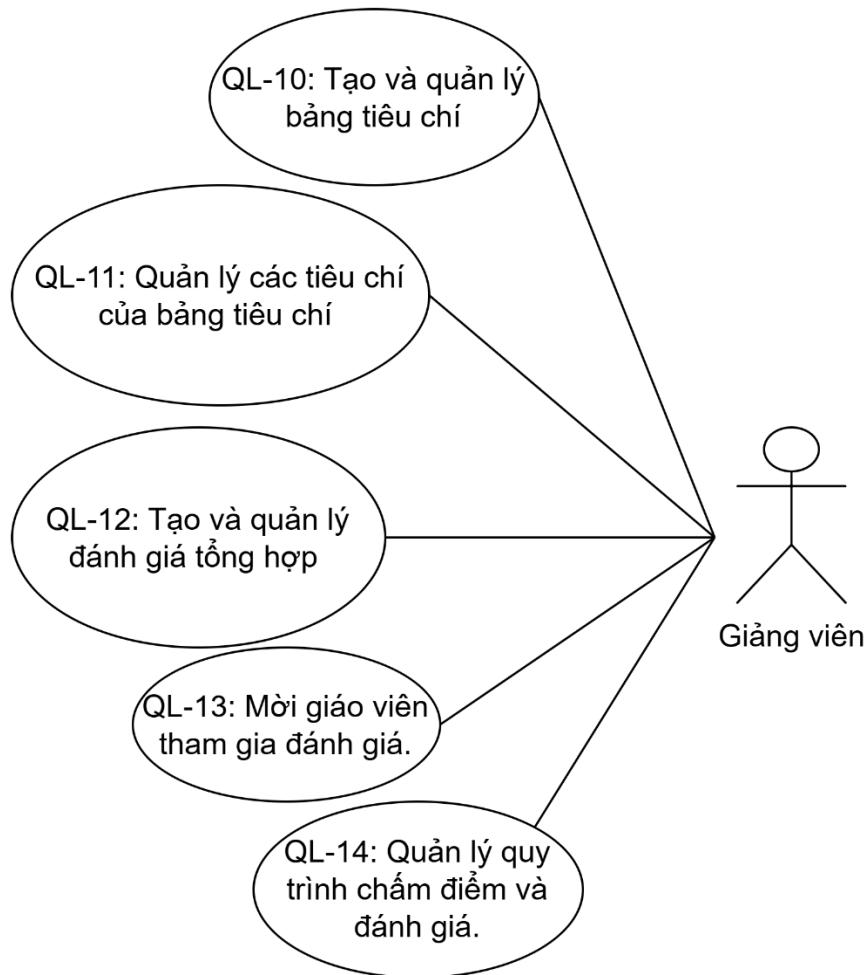
### **3.2. Yêu cầu chức năng**

**API:** Hỗ trợ các chức năng tạo mới, cập nhật, xem chi tiết và xóa các thực thể như chương trình đào tạo, PLO, PO, môn học, CLO, ánh xạ giữa PLO, PO, CLO và chương học, cũng như quản lý rubric, tiêu chí rubric, đánh giá tổng hợp và quy trình chấm điểm. Ngoài ra, API còn phục vụ việc tạo PDF phiếu chấm và PDF kết quả điểm sinh viên.

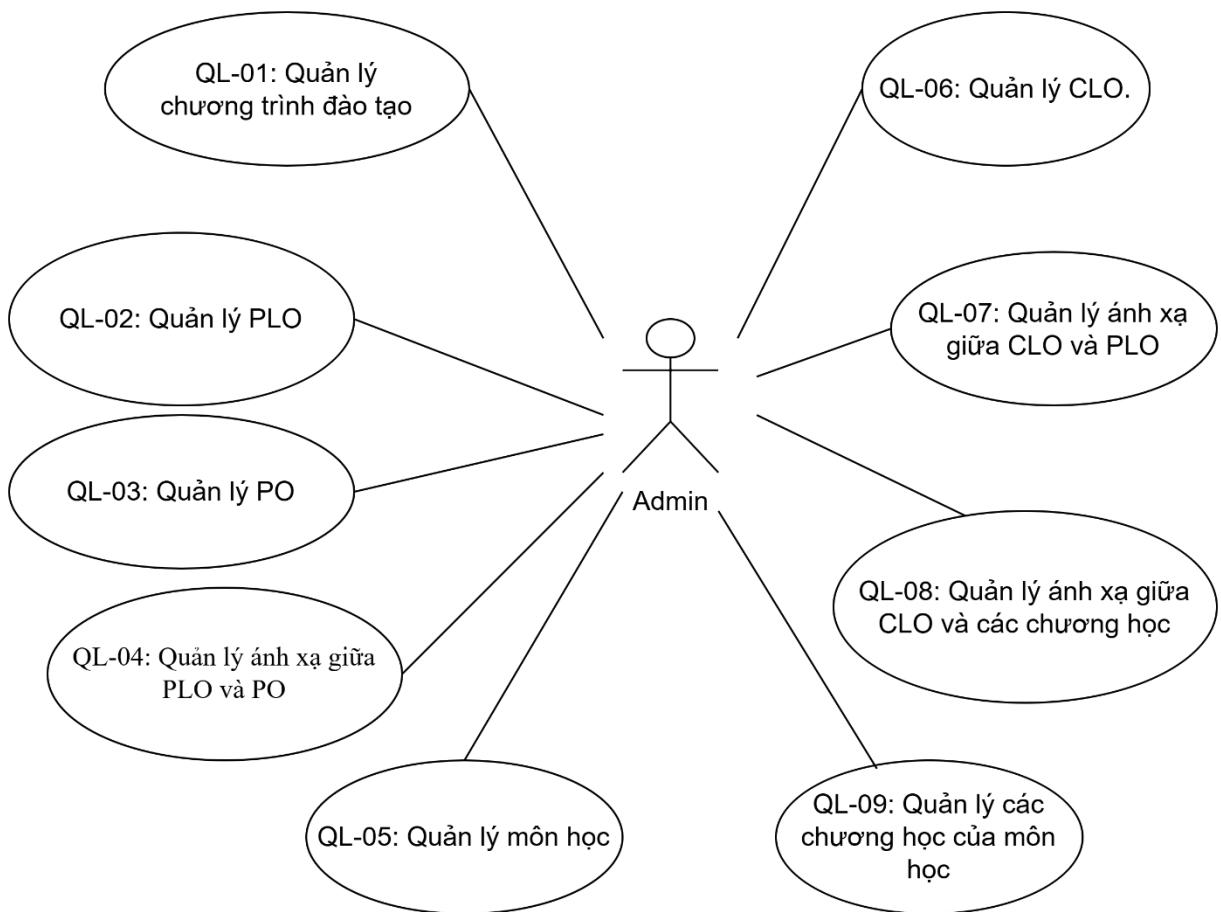
**Giao diện Frontend:** Cung cấp giao diện để thực hiện nhập liệu, hiển thị danh sách

và chi tiết thông tin của các thực thể. Yêu cầu các giao diện quản lý phải được tối ưu hóa cho thiết bị di động, đảm bảo tính hài hòa và thuận tiện cho việc sử dụng. Đặc biệt, cần cung cấp giao diện chấm điểm tối ưu cho thiết bị di động nhằm nâng cao trải nghiệm giảng viên.

### 3.3. Đặt tả hệ thống



Hình 3.1. Sơ đồ use case Giảng viên.



Hình 3.2. Sơ đồ use case Admin

Admin cũng là một giảng viên nên kế thừa tất cả chức năng của một giảng viên.

Bảng 3.1. Mô tả Actor

STT	Tên Actor	Ý nghĩa
1	Admin	Quản trị viên
2	Giảng viên	Giảng viên

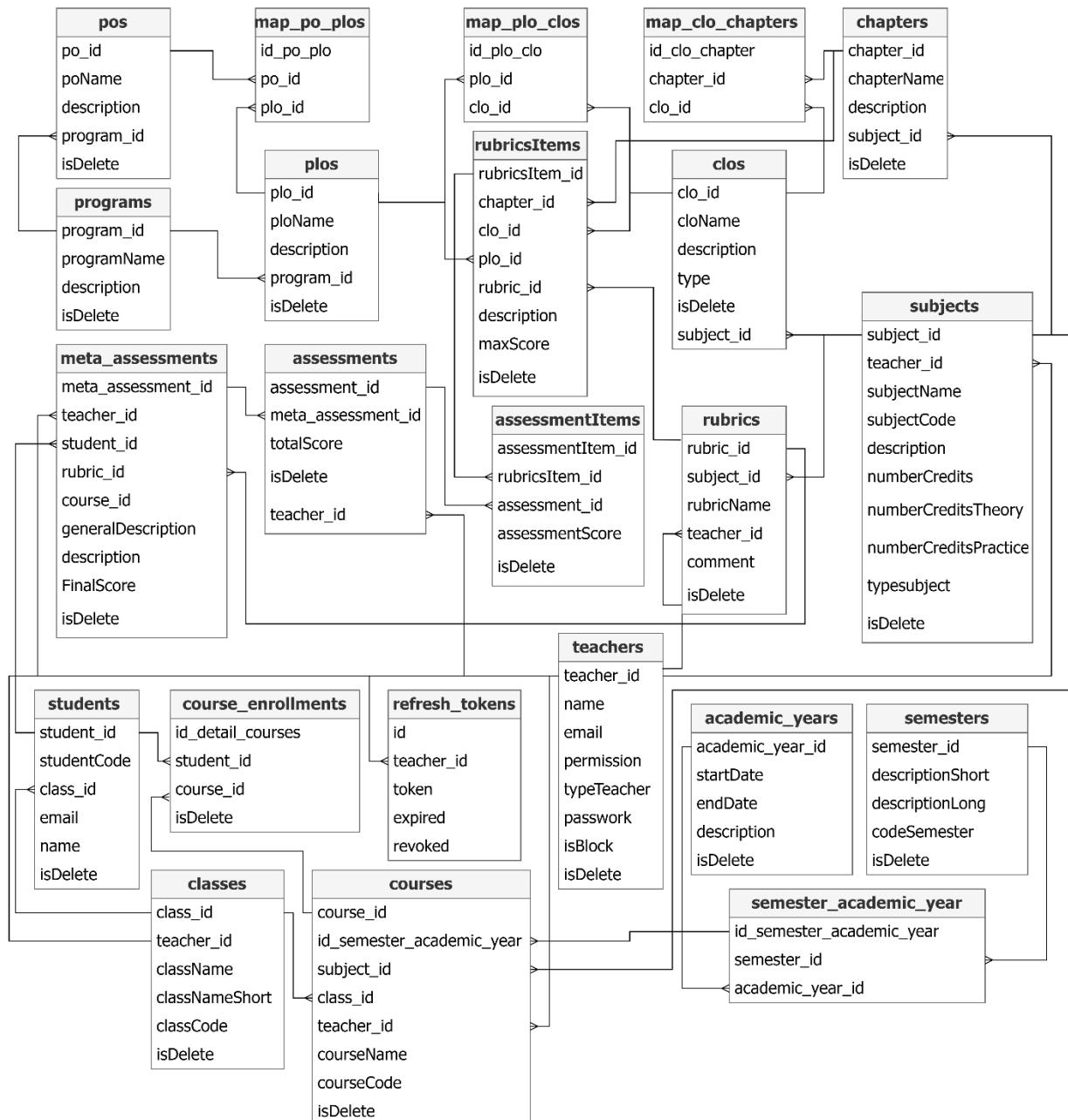
Bảng 3.2. Mô tả use case

STT	Tên Use-case	Ý Nghĩa
1	QL-01: Quản lý chương trình đào tạo	Quản trị viên có quyền tạo, chỉnh sửa và xóa các chương trình đào tạo.
2	QL-02: Quản lý PLO	Quản trị viên có quyền quản lý PLO, bao gồm việc thêm, chỉnh sửa và xóa các PLO.

3	QL-03: Quản lý PO	Quản trị viên có quyền quản lý các PO. Bao gồm việc thêm, chỉnh sửa và xóa các PO.
4	QL-04: Quản lý ánh xạ giữa PLO và PO	Quản trị viên có quyền quản lý ánh xạ giữa PLO và PO. Đảm bảo rằng các PLO hỗ trợ các PO.
5	QL-05: Quản lý môn học	Quản trị viên có quyền tạo, chỉnh sửa và xóa các môn học trong chương trình.
6	QL-06: Quản lý CLO	Quản trị viên có quyền quản lý CLO. Bao gồm thêm, chỉnh sửa và xóa các CLO của từng môn học.
7	QL-07: Quản lý ánh xạ giữa CLO và PLO	Quản trị viên có quyền quản lý ánh xạ giữa CLO và PLO. Đảm bảo rằng các CLO hỗ trợ các PLO.
8	QL-08: Quản lý ánh xạ giữa CLO và các chương học	Quản trị viên có quyền quản lý ánh xạ giữa CLO và các chương học trong môn học. Đảm bảo rằng các CLO phù hợp với nội dung của các chương học.
9	QL-09: Quản lý các chương học của môn học	Quản trị viên có quyền tạo, chỉnh sửa và xóa các chương học của một môn học.
10	QL-10: Tạo và quản lý bảng tiêu chí	Giảng viên có quyền tạo, chỉnh sửa và quản lý bảng tiêu chí dùng để đánh giá các dự án của sinh viên.
11	QL-11: Quản lý các tiêu chí của bảng tiêu chí	Giảng viên có quyền quản lý các tiêu chí cụ thể trong bảng tiêu chí. Bao gồm việc thêm, chỉnh sửa và xóa các tiêu chí đánh giá chi tiết trong bảng tiêu chí.
12	QL-12: Tạo và quản lý đánh giá tổng hợp	Giảng viên có quyền tạo và quản lý các đánh giá tổng hợp. Bao gồm việc thiết lập các thông tin liên quan đến làn đánh giá, như lớp học, môn học, Rubric, và mô tả đánh giá.
13	QL-13: Mời giảng viên tham gia đánh giá	Giảng viên có quyền mời các giảng viên khác tham gia vào quá trình đánh giá. Bao gồm việc gửi liên kết đến các giảng viên để họ có thể tham gia vào việc chấm điểm.

14	QL-14: Quản lý quy trình chấm điểm và đánh giá	Giảng viên có quyền quản lý toàn bộ quy trình chấm điểm và đánh giá, bao gồm việc theo dõi điểm số, chỉnh sửa kết quả và tổng hợp điểm của sinh viên. Đảm bảo rằng quy trình đánh giá được thực hiện một cách hiệu quả và chính xác.
----	--	--

### 3.4. Lược đồ cơ sở dữ liệu



Hình 3.3. Lược đồ cơ sở dữ liệu.

Bảng 3.3. Mô tả bảng chương trình đào tạo

<b>Thuộc tính</b>	<b>Điễn giải</b>	<b>Kiểu dữ liệu</b>	<b>Ràng buộc</b>	<b>Ghi chú</b>
program_id	Mã chương trình	varchar(20)	Primary key	
programName	Tên chương trình	text		

<b>Thuộc tính</b>	<b>Điễn giải</b>	<b>Kiểu dữ liệu</b>	<b>Ràng buộc</b>	<b>Ghi chú</b>
description	Mô tả	text		
isDelete	Trạng thái xóa mềm	tinyint		

Bảng 3.4. Mô tả bảng PO

<b>Thuộc tính</b>	<b>Điễn giải</b>	<b>Kiểu dữ liệu</b>	<b>Ràng buộc</b>	<b>Ghi chú</b>
po_id	Mã mục tiêu chương trình	int	Primary key	
program_id	Mã chương trình	varchar(20)	Foreign key	
poName	Tên mục tiêu	varchar(255)		
description	Mô tả	text		
isDelete	Trạng thái xóa mềm	tinyint		

Bảng 3.5. Mô tả bảng PLO

<b>Thuộc tính</b>	<b>Điễn giải</b>	<b>Kiểu dữ liệu</b>	<b>Ràng buộc</b>	<b>Ghi chú</b>
plo_id	Mã chuẩn đầu ra chương trình	int	Primary key	
program_id	Mã chương trình	varchar(20)	Foreign key	
ploName	Tên chuẩn đầu ra chương trình	varchar(255)		
description	Mô tả	text		

<b>Thuộc tính</b>	<b>Điễn giải</b>	<b>Kiểu dữ liệu</b>	<b>Ràng buộc</b>	<b>Ghi chú</b>
isDelete	Trạng thái xóa mềm	tinyint		

Bảng 3.6. Mô tả bảng ánh xạ PLO với PO

<b>Thuộc tính</b>	<b>Điễn giải</b>	<b>Kiểu dữ liệu</b>	<b>Ràng buộc</b>	<b>Ghi chú</b>
id_po_plo	Mã ánh xạ	int	Primary key	
po_id	Mã mục tiêu chương trình	int	Foreign key	
plo_id	Mã chuẩn đầu ra chương trình	int	Foreign key	

Bảng 3.7. Mô tả bảng môn học

<b>Thuộc tính</b>	<b>Điễn giải</b>	<b>Kiểu dữ liệu</b>	<b>Ràng buộc</b>	<b>Ghi chú</b>
subject_id	Mã môn học	int	Primary key	
teacher_id	Mã giảng viên	int	Foreign key	
subjectName	Tên môn học	text		
subjectCode	Mã giả	varchar(6)		
description	Mô tả	text		
numberCredits	Số tín chỉ	int		
numberCreditsTheory	Số tín chỉ lý thuyết	int		

<b>Thuộc tính</b>	<b>Diễn giải</b>	<b>Kiểu dữ liệu</b>	<b>Ràng buộc</b>	<b>Ghi chú</b>
numberCreditsPractice	Số tín chỉ thực hành	int		
typesubject	Loại môn học	Enum('Đại cương', 'Cơ sở ngành', 'Chuyên ngành', 'Thực tập và Đồ án')		
isDelete	Trạng thái xóa mềm	tinyint		

Bảng 3.8. Mô tả bảng CLO

<b>Thuộc tính</b>	<b>Diễn giải</b>	<b>Kiểu dữ liệu</b>	<b>Ràng buộc</b>	<b>Ghi chú</b>
clo_id	Mã chuẩn đầu ra môn học	int	Primary key	
subject_id	Mã môn học	int	Foreign key	
cloName	Tên chuẩn đầu ra môn học	varchar(20)		
description	Mô tả	text		
type	Loại chuẩn đầu ra	Enum(Kiến thức, Thái độ, Kỹ năng)		
isDelete	Trạng thái xóa mềm	tinyint		

Bảng 3.9. Mô tả bảng chương của môn học

Thuộc tính	Điễn giải	Kiểu dữ liệu	Ràng buộc	Ghi chú
chapter_id	Mã chương của môn học	int	Primary key	
subject_id	Mã môn học	int	Foreign key	
chapterName	Tên chương	varchar(100)		
description	Mô tả	text		
isDelete	Trạng thái xóa mềm	tinyint		

Bảng 3.10. Mô tả bảng ánh xạ CLO với chương

Thuộc tính	Điễn giải	Kiểu dữ liệu	Ràng buộc	Ghi chú
id_clo_chapter	Mã ánh xạ	int	Primary key	
chapter_id	Mã chương của môn học	int	Foreign key	
clo_id	Mã chuẩn đầu ra môn học	int	Foreign key	

Bảng 3.11. Mô tả bảng ánh xạ CLO với PLO

Thuộc tính	Điễn giải	Kiểu dữ liệu	Ràng buộc	Ghi chú
map_plo_clos	Mã ánh xạ	int	Primary key	

<b>Thuộc tính</b>	<b>Điễn giải</b>	<b>Kiểu dữ liệu</b>	<b>Ràng buộc</b>	<b>Ghi chú</b>
plo_id	Mã chuẩn đầu ra chương trình	int	Foreign key	
clo_id	Mã chuẩn đầu ra môn học	int	Foreign key	

Bảng 3.12. Mô tả bảng tiêu chí đánh giá rubric

<b>Thuộc tính</b>	<b>Điễn giải</b>	<b>Kiểu dữ liệu</b>	<b>Ràng buộc</b>	<b>Ghi chú</b>
rubric_id	Mã rubric	int	Primary key	
subject_id	Mã môn học	int	Foreign key	
teacher_id	Mã giảng viên	int	Foreign key	
rubricName	Tên rubric	varchar(255)		
comment	Bình luận	text		
isDelete	Trạng thái xóa	tinyint		

Bảng 3.13. Mô tả bảng tiêu chí đánh giá

<b>Thuộc tính</b>	<b>Điễn giải</b>	<b>Kiểu dữ liệu</b>	<b>Ràng buộc</b>	<b>Ghi chú</b>
rubricsItem_id	Mã thành phần rubric	int	Primary key	
rubric_id	Mã rubric	int	Foreign key	
chapter_id	Mã chương của môn học	int	Foreign key	

<b>Thuộc tính</b>	<b>Diễn giải</b>	<b>Kiểu dữ liệu</b>	<b>Ràng buộc</b>	<b>Ghi chú</b>
clo_id	Mã chuẩn đầu ra môn học	int	Foreign key	
plo_id	Mã chuẩn đầu ra chương trình	int	Foreign key	
description	Tiêu chí đánh giá	text		Lưu trữ nội dung với dạng HTML
maxScore	Điểm số tối đa đạt được của tiêu chí	double		
isDelete	Trạng thái xóa	tinyint		

Bảng 3.14. Mô tả bảng đánh giá meta

<b>Thuộc tính</b>	<b>Diễn giải</b>	<b>Kiểu dữ liệu</b>	<b>Ràng buộc</b>	<b>Ghi chú</b>
meta_assessment_id	Mã đánh giá meta	int	Primary key	
teacher_id	Mã giảng viên	int	Foreign key	
student_id	Mã sinh viên	int	Foreign key	
rubric_id	Mã rubric	int	Foreign key	
course_id	Mã lớp môn học	int	Foreign key	
generalDescription	Mô tả chung	text		
description	Tên đề tài của sinh viên	text		

<b>Thuộc tính</b>	<b>Diễn giải</b>	<b>Kiểu dữ liệu</b>	<b>Ràng buộc</b>	<b>Ghi chú</b>
FinalScore	Tổng điểm của nhiều giảng viên	Double		
date	Ngày đánh giá	date		
place	Địa điểm đánh giá	text		
isDelete	Trạng thái xóa	tinyint		

Bảng 3.15. Mô tả bảng lần đánh giá

<b>Thuộc tính</b>	<b>Diễn giải</b>	<b>Kiểu dữ liệu</b>	<b>Ràng buộc</b>	<b>Ghi chú</b>
assessment_id	Mã lần đánh giá	int	Primary key	
meta_assessment_id	Mã đánh giá meta	int	Foreign key	
teacher_id	Mã giảng viên	int	Foreign key	
totalScore	Điểm số đạt được	Double		
isDelete	Trạng thái xóa	tinyint		

Bảng 3.16. Mô tả bảng thành phần của lần đánh giá

<b>Thuộc tính</b>	<b>Diễn giải</b>	<b>Kiểu dữ liệu</b>	<b>Ràng buộc</b>	<b>Ghi chú</b>
assessmentItem_id	Mã thành phần của lần đánh giá	int	Primary key	
assessment_id	Mã lần đánh giá	int	Foreign key	
rubricsItem_id	Mã thành phần rubric	int	Foreign key	

<b>Thuộc tính</b>	<b>Điễn giải</b>	<b>Kiểu dữ liệu</b>	<b>Ràng buộc</b>	<b>Ghi chú</b>
assessmentScore	Điểm số đạt được	Double		
isDelete	Trạng thái xóa	tinyint		

Bảng 3.17. Mô tả bảng giảng viên

<b>Thuộc tính</b>	<b>Điễn giải</b>	<b>Kiểu dữ liệu</b>	<b>Ràng buộc</b>	<b>Ghi chú</b>
teacher_id	Mã giảng viên	int	Primary key	
name	Tên giảng viên	varchar(255)		
email	Email giảng viên	varchar(255)		
permission	Quyền	tinyint		
typeTeacher	Loại giảng viên	enum('GVCV', 'GVGD')		
passwork	Mật khẩu	varchar(255)		
isBlock	Trạng thái khóa	tinyint		
isDelete	Trạng thái xóa	tinyint		

Bảng 3.18. Mô tả bảng lớp

<b>Thuộc tính</b>	<b>Điễn giải</b>	<b>Kiểu dữ liệu</b>	<b>Ràng buộc</b>	<b>Ghi chú</b>
class_id	Mã lớp	int	Primary key	
teacher_id	Mã giảng viên chủ nhiệm	int	Foreign key	

Thuộc tính	Điễn giải	Kiểu dữ liệu	Ràng buộc	Ghi chú
className	Tên lớp	varchar(255)		
classNameShort	Tên lớp rút gọn	varchar(20)		
classCode	Mã lớp giả	varchar(10)		
isDelete	Trạng thái xóa	tinyint		

Bảng 3.19. Mô tả bảng lớp môn học

Thuộc tính	Điễn giải	Kiểu dữ liệu	Ràng buộc	Ghi chú
course_id	Mã lớp môn học	int	Primary key	
subject_id	Mã môn học	int	Foreign key	
teacher_id	Mã giảng viên	int	Foreign key	
class_id	Mã lớp	int	Foreign key	
id_semester_academic_year	Mã học kỳ năm học	int	Foreign key	
courseName	Tên lớp môn học	text		
courseCode	Mã giả môn học	varchar(20)		

<b>Thuộc tính</b>	<b>Diễn giải</b>	<b>Kiểu dữ liệu</b>	<b>Ràng buộc</b>	<b>Ghi chú</b>
isDelete	Trạng thái xóa	tinyint		

Bảng 3.20. Mô tả bảng tuyển sinh lớp môn học

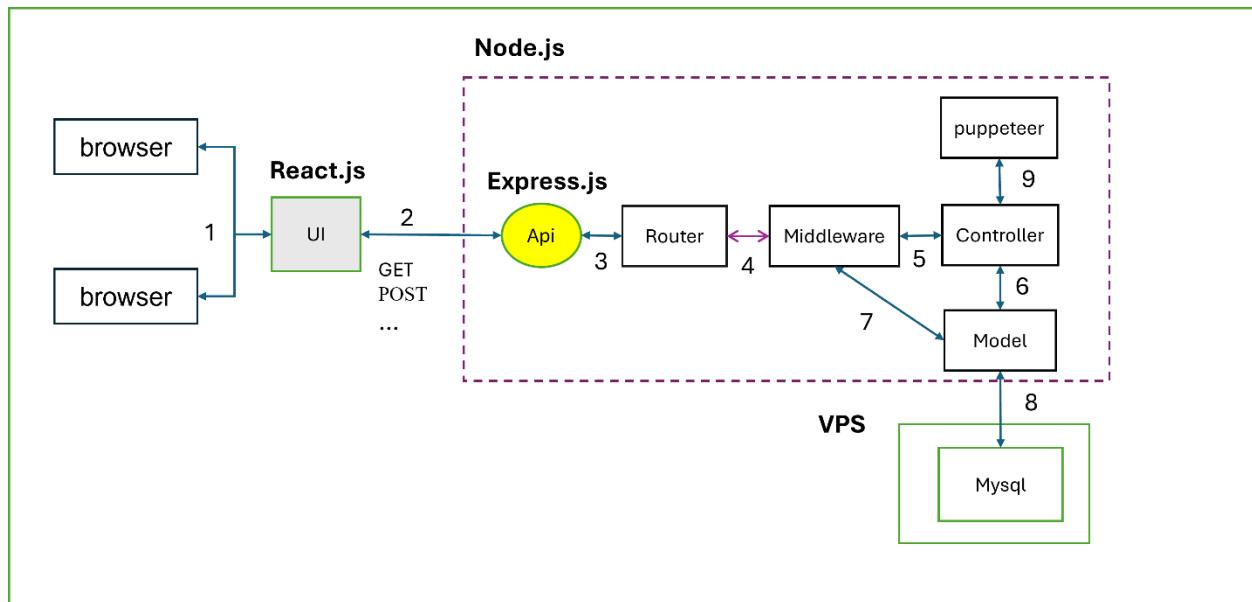
<b>Thuộc tính</b>	<b>Diễn giải</b>	<b>Kiểu dữ liệu</b>	<b>Ràng buộc</b>	<b>Ghi chú</b>
id_detail_courses	Mã tuyển sinh lớp môn học	int	Primary key	
course_id	Mã lớp môn học	int	Foreign key	
student_id	Mã sinh viên	int	Foreign key	
isDelete	Trạng thái xóa	tinyint		

Bảng 3.21. Mô tả bảng sinh viên

<b>Thuộc tính</b>	<b>Diễn giải</b>	<b>Kiểu dữ liệu</b>	<b>Ràng buộc</b>	<b>Ghi chú</b>
student_id	Mã sinh viên	int	Primary key	
class_id	Mã lớp	int	Foreign key	
studentCode	Mã giả sinh viên	varchar(9)		
name	Tên sinh viên	varchar(100)		
email	Email sinh viên	varchar(255)		

Thuộc tính	Diễn giải	Kiểu dữ liệu	Ràng buộc	Ghi chú
date_of_birth	Ngày sinh	date		
isDelete	Trạng thái xóa	tinyint		

### 3.5. Kiến trúc hệ thống



Hình 3.4. Kiến trúc hệ thống

#### Luồng xử lý dữ liệu của hệ thống:

Khi người dùng truy cập vào giao diện trang web, quá trình xử lý bắt đầu từ trình duyệt (browser), nơi giao diện người dùng (UI) thực hiện việc gọi các API thông qua CRUD để lấy dữ liệu và render lên giao diện.

#### Các bước xử lý chi tiết:

Người dùng tương tác với giao diện người dùng (UI) (1): Người dùng thực hiện các thao tác trên trang web thông qua UI.

Gửi yêu cầu đến API (2): Giao diện người dùng gửi các yêu cầu (request) đến API để lấy hoặc gửi dữ liệu đến và từ hệ thống.

Xử lý yêu cầu thông qua Router (3): Khi nhận được yêu cầu từ UI, Express.js xử lý định tuyến (routing) thông qua Router để xác định đúng đường dẫn (endpoint) API.

Xác thực và ủy quyền (4): Trước khi tiếp tục xử lý yêu cầu, Middleware thực hiện

các bước xác thực và ủy quyền (authentication and authorization). Nếu yêu cầu hợp lệ, hệ thống sẽ tiếp tục xử lý.

Xử lý yêu cầu bởi Controller (5): Controller tiếp nhận và xử lý yêu cầu đã được xác thực.

Kết nối với Model (6): Controller kết nối với Model để thực hiện các thao tác liên quan đến dữ liệu.

Xác thực đăng nhập và truy xuất dữ liệu (7): Middleware xác thực đăng nhập lấy dữ liệu bằng cách tương tác với Model để xác thực thông tin người dùng.

Tương tác với cơ sở dữ liệu (8): Model thực hiện kết nối đến cơ sở dữ liệu trên VPS, thao tác với dữ liệu và trả kết quả về cho Controller.

Phục vụ yêu cầu PDF (9): Hệ thống sẽ xử lý và phục vụ các yêu cầu liên quan đến PDF.

### 3.6. Thiết kế API

#### 3.6.1. Định tuyến API trong Express.js

Trong cấu trúc ứng dụng, đoạn mã sau thiết lập các định tuyến (routes) cho các API thông qua việc sử dụng nhiều router khác nhau. Đây là cách tổ chức các định tuyến API, với mỗi router đảm nhiệm một phần chức năng cụ thể trong ứng dụng. Dưới đây là mô tả chi tiết về cấu trúc các router và cách thức hoạt động của chúng:

```
● ● ●  
const express = require('express');  
const router = express.Router();  
  
router.use('/api', AuthRoutes);  
router.use('/api/admin', programRoutes);  
router.use('/api/admin', ploRoutes);  
router.use('/api/admin', cloRoutes);  
...  
  
module.exports = router;
```

Hình 3.5. Định tuyến API trong Express.js

**express.Router()**: Tạo một đối tượng router mới để định nghĩa các định tuyến cho các phần của ứng dụng. Đây là cách tổ chức các định tuyến theo module hoặc tính năng riêng biệt của ứng dụng.

```
● ● ●  
router.use('/api', AuthRoutes);  
router.use('/api/admin', programRoutes);  
router.use('/api/admin', ploRoutes);  
router.use('/api/admin', cloRoutes);
```

Hình 3.6. Các định tuyến được liên kết đến các router

Các định tuyến:

- **/api**: Định nghĩa định tuyến gốc cho các yêu cầu liên quan đến AuthRoutes, dùng để xử lý các chức năng xác thực người dùng.
- **/api/admin**: Định nghĩa định tuyến gốc cho các yêu cầu liên quan đến các router quản lý chương trình (programRoutes), mục tiêu học tập (ploRoutes), và kết quả học tập (cloRoutes). Các router này quản lý các chức năng liên quan đến việc quản lý và xử lý dữ liệu trong khu vực quản trị.

Một số lưu ý:

- **Định Tuyến Trong Router**: Các định tuyến trong các router sẽ được cấu hình bắt đầu từ định tuyến gốc mà router đó được gắn vào. Ví dụ, nếu một endpoint trong AuthRoutes được định nghĩa là /login, thì định tuyến đầy đủ sẽ là /api/login.
- **Tổ Chức Endpoint**: Mỗi router tổ chức các endpoint theo chức năng cụ thể. Việc này bao gồm định nghĩa định tuyến, xử lý các yêu cầu HTTP, sử dụng middleware và thực hiện các logic xử lý liên quan khác. Điều này giúp duy trì sự rõ ràng và tính phân chia trách nhiệm trong mã nguồn.
- **Mở Rộng Ứng Dụng**: Cấu trúc này có thể mở rộng với nhiều router khác nhau, mỗi router xử lý các yêu cầu liên quan đến các phần khác nhau của ứng dụng. Điều này giúp tổ chức mã nguồn một cách rõ ràng và dễ bảo trì, đồng thời hỗ trợ

mở rộng và bảo trì ứng dụng hiệu quả hơn.

### 3.6.2. Định tuyến API chương trình (program)



```
router.get('/programs', ensureAuthenticated,  
programController.index);  
router.post('/program', ensureAuthenticated, checkPermission(3),  
programController.create);  
router.get('/program/:id', ensureAuthenticated,  
programController.getByID);  
router.put('/program/:id', ensureAuthenticated, checkPermission(3),  
programController.update);  
router.delete('/program/:id', ensureAuthenticated,  
checkPermission(3), programController.delete);  
router.get('/program/isDelete/true', ensureAuthenticated,  
programController.isDeleteTotrue);  
router.get('/program/isDelete/false', ensureAuthenticated,  
programController.isDeleteTofalse);  
router.put('/program/isDelete/:id', ensureAuthenticated,  
checkPermission(3), programController.toggleIsDelete);  
router.get('/program/templates/post', ensureAuthenticated,  
checkPermission(3), programController.getFormPost);
```

Hình 3.7. Định tuyến API chương trình (program)

#### Middleware:

- ensureAuthenticated: Đảm bảo người dùng đã đăng nhập.
- checkPermission(3): Kiểm tra quyền truy cập cấp 3.

#### Các định tuyến:

- GET /programs: Danh sách chương trình.
- POST /program: Tạo chương trình mới (quyền cấp 3).
- GET /program/:id: Thông tin chương trình theo ID.
- PUT /program/:id: Cập nhật chương trình (quyền cấp 3).
- DELETE /program/:id: Xóa chương trình (quyền cấp 3).
- GET /program/isDelete/true: Chương trình đã xóa.
- GET /program/isDelete/false: Chương trình chưa xóa.

- PUT /program/isDelete/:id: Thay đổi trạng thái xóa (quyền cấp 3).
- GET /program/templates/post: Mẫu form đăng bài.

### 3.6.3. Định tuyến API mục tiêu chương trình (PO)



```
router.get('/pos', ensureAuthenticated, P0.index);
router.post('/po', ensureAuthenticated, checkPermission(3),
P0.create);
router.get('/po/:id', ensureAuthenticated, P0.getByID);
router.put('/po/:id', ensureAuthenticated, checkPermission(3),
P0.update);
router.delete('/po/:id', ensureAuthenticated, checkPermission(3),
P0.delete);
router.delete('/pos/multiple', ensureAuthenticated,
checkPermission(3), P0.deleteMultiple);
router.get('/pos/isDelete/true', ensureAuthenticated,
P0.isDeleteToTrue);
router.get('/pos/isDelete/false', ensureAuthenticated,
P0.isDeleteToFalse);
router.put('/pos/softDelete', ensureAuthenticated,
checkPermission(3), P0.softDeleteMultiple);
router.put('/po/:id/softDelete', ensureAuthenticated,
checkPermission(3), P0.toggleSoftDeleteById);
```

Hình 3.8. Định tuyến API mục tiêu chương trình (PO)

#### Các định tuyến:

- GET /pos: Danh sách PO.
- POST /po: Tạo PO mới (quyền cấp 3).
- GET /po/:id: Thông tin PO theo ID.
- PUT /po/:id: Cập nhật PO (quyền cấp 3).
- DELETE /po/:id: Xóa PO (quyền cấp 3).
- DELETE /pos/multiple: Xóa nhiều PO (quyền cấp 3).
- GET /pos/isDelete/true: PO đã xóa.
- GET /pos/isDelete/false: PO chưa xóa.
- PUT /pos/softDelete: Xóa mềm nhiều PO (quyền cấp 3).

- PUT /po/:id/softDelete: Chuyển đổi trạng thái xóa mềm PO theo ID (quyền cấp 3).

### 3.6.4. Định tuyến API chuẩn đầu ra chương trình (PLO)



```
router.get('/plos', ensureAuthenticated, PL0.index);
router.post('/plo', ensureAuthenticated, checkPermission(3),
PL0.create);
router.get('/plo/:id', ensureAuthenticated, PL0.getByID);
router.put('/plo/:id', ensureAuthenticated, checkPermission(3),
PL0.update);
router.delete('/plo/:id', ensureAuthenticated, checkPermission(3),
PL0.delete);
router.delete('/plos/multiple', ensureAuthenticated,
checkPermission(3), PL0.deleteMultiple);
router.get('/plos/isDelete/true', ensureAuthenticated,
PL0.isDeleteTotrue);
router.get('/plos/isDelete/false', ensureAuthenticated,
PL0.isDeleteTofalse);
router.put('/plos/softDelete', ensureAuthenticated,
checkPermission(3), PL0.softDeleteMultiple);
router.put('/plo/:id/softDelete', ensureAuthenticated,
checkPermission(3), PL0.toggleSoftDeleteById);
```

Hình 3.9. Định tuyến API chuẩn đầu ra chương trình (PLO)

#### Các định tuyến:

- GET /plos: Lấy danh sách tất cả các PLO.
- POST /plo: Tạo mới PLO (quyền cấp 3).
- GET /plo/:id: Lấy thông tin PLO theo ID.
- PUT /plo/:id: Cập nhật PLO theo ID (quyền cấp 3).
- DELETE /plo/:id: Xóa PLO theo ID (quyền cấp 3).
- DELETE /plos/multiple: Xóa nhiều PLO (quyền cấp 3).
- GET /plos/isDelete/true: Lấy PLO đã bị xóa.
- GET /plos/isDelete/false: Lấy PLO chưa bị xóa.
- PUT /plos/softDelete: Xóa mềm nhiều PLO (quyền cấp 3).

- PUT /plo/:id/softDelete: Chuyển đổi trạng thái xóa mềm PLO theo ID (quyền cấp 3).

### 3.6.5. Định tuyến API mới quan hệ PLO và PO (PLO\_PO)



```
router.get('/po-plo', ensureAuthenticated, P0_PL0.getAll);
router.post('/po-plo', ensureAuthenticated, checkPermission(3),
P0_PL0.SavePoPlo);
router.delete('/po-plo', ensureAuthenticated, checkPermission(3),
P0_PL0.DeletePoPlo);
```

Hình 3.10. Định tuyến API mới quan hệ PLO và PO (PLO\_PO)

#### Các định tuyến:

- GET /po-plo: Lấy tất cả các PO-PLO.
- POST /po-plo: Lưu PO-PLO (quyền cấp 3).
- DELETE /po-plo: Xóa PO-PLO (quyền cấp 3).

### 3.6.6. Định tuyến API môn học (subject)

```
router.get('/subjects', ensureAuthenticated,
  SubjectController.getSubjects);
router.get('/subject/:id', ensureAuthenticated,
  SubjectController.getByID);
router.get('/subject/:id/rubrics', ensureAuthenticated,
  SubjectController.getRubricsBySubjectId);
router.put('/subject/:id', ensureAuthenticated, checkPermission(3),
  SubjectController.update);
router.delete('/subject/:id', ensureAuthenticated,
  checkPermission(3), SubjectController.delete);
router.post('/subject', ensureAuthenticated,
  SubjectController.create);
router.get('/subjects/isDelete/false', ensureAuthenticated,
  SubjectController.isDeleteTofalse);
router.get('/subjects/isDelete/true', ensureAuthenticated,
  SubjectController.isDeleteTotrue);
router.delete('/subjects/multiple', ensureAuthenticated,
  checkPermission(3), SubjectController.deleteMultiple);
router.put('/subjects/softDelete', ensureAuthenticated,
  checkPermission(3), SubjectController.softDeleteMultiple);
router.put('/subject/:id/softDelete', ensureAuthenticated,
  checkPermission(3), SubjectController.toggleSoftDeleteById);
router.get('/subject/templates/post', ensureAuthenticated,
  SubjectController.getFormPost);
router.post('/subject/templates/update', ensureAuthenticated,
  checkPermission(3), SubjectController.getFormUpdate);
```

Hình 3.11. Định tuyến API môn học (subject)

Các định tuyến:

- GET /subjects: Lấy tất cả các môn học.
- GET /subject/:id: Lấy thông tin môn học theo ID.
- GET /subject/:id/rubrics: Lấy rubrics của môn học theo ID.
- PUT /subject/:id: Cập nhật thông tin môn học.
- DELETE /subject/:id: Xóa môn học.
- POST /subject: Tạo mới môn học.
- GET /subjects/isDelete/false: Lấy môn học chưa bị xóa.
- GET /subjects/isDelete/true: Lấy môn học đã bị xóa.
- DELETE /subjects/multiple: Xóa nhiều môn học.
- PUT /subjects/softDelete: Xóa mềm nhiều môn học.

- PUT /subject/:id/softDelete: Xóa mềm môn học theo ID.
- GET /subject/templates/post: Lấy form tạo mới môn học.
- POST /subject/templates/update: Cập nhật form môn học.

### 3.6.7. Định tuyến API chuẩn đầu ra của môn học (CLO)

```

router.get('/clos', ensureAuthenticated, CloController.index);
router.post('/clo', ensureAuthenticated, CloController.create);
router.get('/clo/:id', ensureAuthenticated, CloController.getID);
router.put('/clo/:id', ensureAuthenticated, CloController.update);
router.delete('/clo/:id', ensureAuthenticated,
CloController.delete);
router.delete('/clos/multiple', ensureAuthenticated,
CloController.deleteMultiple);
router.put('/clo/:id/softDelete', ensureAuthenticated,
CloController.toggleSoftDeleteById);
router.get('/clos/isDelete/true', ensureAuthenticated,
CloController.isDeleteTotrue);
router.get('/clos/isDelete/false', ensureAuthenticated,
CloController.isDeleteTofalse);
router.put('/clos/softDelete', ensureAuthenticated,
CloController.softDeleteMultiple);

```

Hình 3.12. Định tuyến API chuẩn đầu ra của môn học (CLO)

#### Các định tuyến:

- GET /clos: Lấy tất cả CLO.
- POST /clo: Tạo mới CLO.
- GET /clo/:id: Lấy thông tin CLO theo ID.
- PUT /clo/:id: Cập nhật thông tin CLO.
- DELETE /clo/:id: Xóa CLO.
- DELETE /clos/multiple: Xóa nhiều CLO.
- PUT /clo/:id/softDelete: Xóa mềm CLO theo ID.
- GET /clos/isDelete/true: Lấy CLO đã bị xóa.
- GET /clos/isDelete/false: Lấy CLO chưa bị xóa.
- PUT /clos/softDelete: Xóa mềm nhiều CLO.

- GET /clo/templates/post: Lấy form tạo mới CLO.
- POST /clo/templates/update: Cập nhật form CLO.

### 3.6.8. Định tuyến API mới quan hệ PLO và CLO (PLO\_CLO)



```
router.get('/plo-clo', ensureAuthenticated, PLO_CLO.index);
router.post('/plo-clo', ensureAuthenticated, PLO_CLO.SaveCloPlo);
router.delete('/plo-clo', ensureAuthenticated,
PLO_CLO.DeleteCloPlo);
```

Hình 3.13. Định tuyến API mới quan hệ PLO và CLO (PLO\_CLO)

#### Các định tuyến:

- GET /plo-clo: Lấy tất cả các mối quan hệ giữa PLO và CLO.
- POST /plo-clo: Lưu mối quan hệ giữa PLO và CLO.
- DELETE /plo-clo: Xóa mối quan hệ giữa PLO và CLO.

### 3.6.9. Định tuyến API chương của môn học (chapter)



```
router.get('/chapters', ensureAuthenticated,  
ChapterController.index);  
router.post('/chapter', ensureAuthenticated,  
ChapterController.create);  
router.get('/chapter/:id', ensureAuthenticated,  
ChapterController.getByID);  
router.put('/chapter/:id', ensureAuthenticated,  
ChapterController.update);  
router.delete('/chapter/:id', ensureAuthenticated,  
ChapterController.delete);  
router.delete('/chapters/multiple', ensureAuthenticated,  
ChapterController.deleteMultiple);  
router.get('/chapters/isDelete/true', ensureAuthenticated,  
ChapterController.isDeleteTotrue);  
router.get('/chapters/isDelete/false', ensureAuthenticated,  
ChapterController.isDeleteTofalse);  
router.put('/chapters/softDelete', ensureAuthenticated,  
ChapterController.softDeleteMultiple);  
router.put('/chapter/:id/softDelete', ensureAuthenticated,  
ChapterController.toggleSoftDeleteById);
```

Hình 3.14. Định tuyến API chương của môn học (chapter)

#### Các định tuyến:

- GET /chapters: Lấy danh sách tất cả các chương.
- POST /chapter: Tạo một chương mới.
- GET /chapter/:id: Lấy thông tin chương dựa trên ID.
- PUT /chapter/:id: Cập nhật thông tin chương dựa trên ID.
- DELETE /chapter/:id: Xóa chương dựa trên ID.
- DELETE /chapters/multiple: Xóa nhiều chương.
- GET /chapters/isDelete/true: Lấy các chương đã bị xóa.
- GET /chapters/isDelete/false: Lấy các chương chưa bị xóa.
- PUT /chapters/softDelete: Xóa mềm nhiều chương.
- PUT /chapter/:id/softDelete: Xóa mềm một chương theo ID.

- GET /chapter/templates/post: Lấy mẫu biểu mẫu để tạo chương.
- POST /chapter/templates/update: Cập nhật mẫu biểu mẫu chương.

### 3.6.10. Định tuyến API mới quan hệ chapter và CLO (chapter\_CLO)

```
router.get('/clo-chapter', ensureAuthenticated,
CLO CHAPTER.getCloChapter);
router.post('/clo-chapter', ensureAuthenticated,
CLO CHAPTER.SaveCloChapter);
router.delete('/clo-chapter', ensureAuthenticated,
CLO CHAPTER.DeleteCloChapter);
```

Hình 3.15. Định tuyến API mới quan hệ chapter và CLO (chapter\_CLO)

#### Các định tuyến:

- GET /clo-chapter: Lấy dữ liệu kết hợp giữa CLO và Chapter.
- POST /clo-chapter: Lưu thông tin kết hợp giữa CLO và Chapter.
- DELETE /clo-chapter: Xóa thông tin kết hợp giữa CLO và Chapter.

### 3.6.11. Định tuyến API rubric



```
router.get('/rubrics', ensureAuthenticated, RubricController.index);
router.post('/rubric', ensureAuthenticated,
RubricController.create);
router.get('/rubric/:id', ensureAuthenticated,
RubricController.getByID);
router.put('/rubric/:id', ensureAuthenticated,
RubricController.update);
router.get('/rubrics/isDelete/true', ensureAuthenticated,
RubricController.isDeleteTotrue);
router.get('/rubrics/isDelete/false', ensureAuthenticated,
RubricController.isDeleteTofalse);
router.get('/rubric/:id/items', ensureAuthenticated,
RubricController.getItemsByRubricId);
router.get('/rubrics/checkScore', ensureAuthenticated,
RubricController.getRubricsForCheckScore);
router.put('/rubrics/softDelete', ensureAuthenticated,
RubricController.softDeleteMultiple);
router.put('/rubric/:id/softDelete', ensureAuthenticated,
RubricController.toggleSoftDeleteById);
router.delete('/rubric/:id', ensureAuthenticated,
RubricController.delete);
router.delete('/rubrics/multiple', ensureAuthenticated,
RubricController.deleteMultiple);
```

Hình 3.16. Định tuyến API rubric

#### Các định tuyến:

- GET /rubrics: Lấy danh sách tất cả các rubrics.
- POST /rubric: Tạo một rubric mới.
- GET /rubric/:id: Lấy thông tin chi tiết của một rubric theo ID.
- PUT /rubric/:id: Cập nhật thông tin của rubric theo ID.
- GET /rubrics/isDelete/true: Lấy các rubric đã bị xóa.
- GET /rubrics/isDelete/false: Lấy các rubric chưa bị xóa.
- GET /rubric/:id/items: Lấy các mục liên quan đến rubric theo ID.
- GET /rubrics/checkScore: Lấy rubrics để kiểm tra điểm.
- PUT /rubrics/softDelete: Xóa mềm nhiều rubrics.

- PUT /rubric/:id/softDelete: Xóa mềm rubric theo ID.
- DELETE /rubric/:id: Xóa rubric theo ID.
- DELETE /rubrics/multiple: Xóa nhiều rubrics.

### 3.6.12. Định tuyến API rubric items



```
router.get('/rubric-items', ensureAuthenticated,
RubricItemController.index);
router.post('/rubric-item', ensureAuthenticated,
RubricItemController.create);
router.get('/rubric-item/:id', ensureAuthenticated,
RubricItemController.getByID);
router.post('/rubric-item/checkScore', ensureAuthenticated,
RubricItemController.checkScore);
router.put('/rubric-item/:id', ensureAuthenticated,
RubricItemController.update);
router.delete('/rubric-item/:id', ensureAuthenticated,
RubricItemController.delete);
router.delete('/rubric-items/multiple', ensureAuthenticated,
RubricItemController.deleteMultiple);
router.put('/rubric-items/softDelete', ensureAuthenticated,
RubricItemController.softDeleteMultiple);
router.put('/rubric-item/:id/softDelete', ensureAuthenticated,
RubricItemController.toggleSoftDeleteById);
router.get('/rubric-items/isDelete/true', ensureAuthenticated,
RubricItemController.isDeleteToTrue);
router.get('/rubric-items/isDelete/false', ensureAuthenticated,
RubricItemController.isDeleteToFalse);
```

Hình 3.17. Định tuyến API rubric items

#### Các định tuyến:

- GET /rubric-items: Lấy danh sách tất cả các rubric items.
- POST /rubric-item: Tạo một rubric item mới.
- GET /rubric-item/:id: Lấy thông tin chi tiết của một rubric item theo ID.
- POST /rubric-item/checkScore: Kiểm tra điểm cho một rubric item.
- PUT /rubric-item/:id: Cập nhật thông tin của rubric item theo ID.
- DELETE /rubric-item/:id: Xóa một rubric item theo ID.

- DELETE /rubric-items/multiple: Xóa nhiều rubric items.
- PUT /rubric-items/softDelete: Xóa mềm nhiều rubric items.
- PUT /rubric-item/:id/softDelete: Xóa mềm một rubric item theo ID.
- GET /rubric-items/isDelete/true: Lấy các rubric items đã bị xóa.
- GET /rubric-items/isDelete/false: Lấy các rubric items chưa bị xóa.

### 3.6.13. Định tuyến API đánh giá meta (meta assessment)



```

router.get('/meta-assessments', ensureAuthenticated,
MetaAssessmentController.index);
router.post('/meta-assessment', ensureAuthenticated,
MetaAssessmentController.create);
router.post('/meta-assessment/listStudent', ensureAuthenticated,
MetaAssessmentController.createListStudent);
router.get('/meta-assessment/:id', ensureAuthenticated,
MetaAssessmentController.show);
router.patch('/meta-assessment/:id', ensureAuthenticated,
MetaAssessmentController.updateDescription);
router.put('/meta-assessment/:id', ensureAuthenticated,
MetaAssessmentController.update);
router.delete('/meta-assessment/:id', ensureAuthenticated,
MetaAssessmentController.delete);
router.delete('/meta-assessments/multiple', ensureAuthenticated,
MetaAssessmentController.deleteMultiple);
router.patch('/meta-assessments/softDeleteByGeneralDescription',
ensureAuthenticated,
MetaAssessmentController.toggleSoftDeleteByGeneralDescription);
router.patch('/meta-assessments/updateByGeneralDescription',
ensureAuthenticated,
MetaAssessmentController.updateByGeneralDescription);
router.delete('/meta-assessments/deleteByGeneralDescription',
ensureAuthenticated,
MetaAssessmentController.deleteByGeneralDescription);
router.post('/meta-assessment/templates/data', ensureAuthenticated,
MetaAssessmentController.getFormUpdateDescriptionExcel);

```

Hình 3.18. Định tuyến API đánh giá meta (meta assessment)

#### Các định tuyến:

- GET /meta-assessments: Lấy danh sách tất cả các meta assessments.
- POST /meta-assessment: Tạo một meta assessment mới.

- POST /meta-assessment/listStudent: Tạo danh sách sinh viên cho meta assessment.
- GET /meta-assessment/:id: Lấy thông tin chi tiết của một meta assessment theo ID.
- PATCH /meta-assessment/:id: Cập nhật mô tả của meta assessment theo ID.
- PUT /meta-assessment/:id: Cập nhật meta assessment theo ID.
- DELETE /meta-assessment/:id: Xóa một meta assessment theo ID.
- DELETE /meta-assessments/multiple: Xóa nhiều meta assessments.
- PATCH /meta-assessments/softDeleteByGeneralDescription: Xóa mềm meta assessments theo mô tả chung.
- PATCH /meta-assessments/updateByGeneralDescription: Cập nhật meta assessments theo mô tả chung.
- DELETE /meta-assessments/deleteByGeneralDescription: Xóa meta assessments theo mô tả chung.
- POST /meta-assessment/templates/data: Lấy dữ liệu từ form cập nhật mô tả Excel.

### 3.6.14. Định tuyến API lần đánh giá (assessment)

```

● ● ●

router.get('/assessment/checkTeacher', ensureAuthenticated,
assessmentsController.checkTeacherInAssessment);
router.get('/assessment', ensureAuthenticated,
assessmentsController.getAssessments);
router.patch('/assessment/:id/totalScore', ensureAuthenticated,
assessmentsController.updateStotalScore);
router.get('/assessment/:id', ensureAuthenticated,
assessmentsController.getByID);
router.get('/assessment/:id/items', ensureAuthenticated,
assessmentsController.GetitemsByID);
router.post('/assessment', ensureAuthenticated,
assessmentsController.create);
router.put('/assessment/:id', ensureAuthenticated,
assessmentsController.update);
router.delete('/assessment/:id', ensureAuthenticated,
assessmentsController.delete);
router.delete('/assessments/multiple', ensureAuthenticated,
assessmentsController.deleteMultiple);
router.delete('/assessment/teacher/:teacherId', ensureAuthenticated,
assessmentsController.deleteByTeacherId);

```

Hình 3.19. Định tuyến API lần đánh giá (assessment)

### Các định tuyến:

- GET /assessment/checkTeacher: Kiểm tra giảng viên trong assessment.
- GET /assessment: Lấy danh sách tất cả các assessments.
- PATCH /assessment/:id/totalScore: Cập nhật tổng điểm của assessment theo ID.
- GET /assessment/:id: Lấy thông tin chi tiết của assessment theo ID.
- GET /assessment/:id/items: Lấy các mục của assessment theo ID.
- POST /assessment: Tạo một assessment mới.
- PUT /assessment/:id: Cập nhật assessment theo ID.
- DELETE /assessment/:id: Xóa assessment theo ID.
- DELETE /assessments/multiple: Xóa nhiều assessments.
- DELETE /assessment/teacher/:teacherId: Xóa assessment theo ID của giảng viên.

### 3.6.15. Định tuyến API assessment items



```
router.post('/assessment-item', ensureAuthenticated,  
assessmentItemsController.create);  
router.put('/assessment-item/:id', ensureAuthenticated,  
assessmentItemsController.update);
```

Hình 3.20. Định tuyến API assessment items

### Các định tuyến:

- POST /assessment-item: Tạo một assessment item mới. Phương thức POST được sử dụng để thêm tài nguyên mới vào hệ thống.
- PUT /assessment-item/:id: Cập nhật assessment item theo ID. Phương thức PUT được sử dụng để sửa đổi tài nguyên hiện có dựa trên ID được cung cấp.

## CHƯƠNG 4. KẾT QUẢ NGHIÊN CỨU

### 4.1. Chức năng Admin

#### 4.1.1. Quản lý chương trình đào tạo (QL-01)

SET

Tổng quan

Chấm điểm

Chương trình

Mục tiêu CT

CDR CT

PO\_PLO

Học phần

Tiêu chí ĐG

Lớp

Giáo viên

Sinh viên

Lớp môn học

Nguyễn Bảo An 121214

Chương trình

Tạo mới

Chỉnh sửa

Nhập bằng Excel

Mô tả

IT Công nghệ thông tin

Trình độ đào tạo: Đại học

Mã ngành đào tạo: 7480201

Loại hình đào tạo: Chính quy

Số tín chỉ yêu cầu: 150

Thời gian đào tạo: 4 năm

Đối tượng tuyển sinh: Theo Quy định của Bộ Giáo dục và Đào tạo và Quy định của Trường Đại học Trà Vinh

Thang điểm: Theo quy định của Trường Đại học Trà Vinh

Điều kiện tốt nghiệp: Theo quy định

Văn bằng tốt nghiệp: Kỹ sư

Chương trình đào tạo chuẩn tham khảo:

i. Chương trình đào tạo Cử nhân ngành Công nghệ Thông tin của Trường Đại học Công nghệ Thông tin, Đại học quốc gia Tp. HCM.

ii. Chương trình đào tạo Cử nhân ngành Công nghệ Thông tin của Trường Đại học Khoa học Tự nhiên, Đại học quốc gia Tp. HCM.

iii. Chương trình đào tạo Cử nhân ngành Công nghệ Thông tin của Trường Đại học Georgia Southern.

iv. Chương trình đào tạo Cử nhân ngành Công nghệ Thông tin của Trường Đại học California State University, Northridge.

v. Chương trình đào tạo cử nhân Công nghệ Thông tin theo khuyến nghị của Association for Computing Machinery (ACM).

Hình 4.1. Trang quản lý chương trình đào tạo

Khi quản trị viên truy cập trang chi tiết chương trình đào tạo CNTT, hệ thống sẽ gửi yêu cầu HTTP GET đến API '/api/admin/program/IT' để lấy thông tin chi tiết, bao gồm tên và mô tả chương trình. Dữ liệu trả về sẽ được hiển thị trên trình duyệt, cho phép quản trị viên thực hiện các thao tác quản lý như chỉnh sửa hoặc cập nhật.

SET

Tổng quan

Chấm điểm

Chương trình

Mục tiêu CT

CDR CT

PO\_PLO

Học phần

Tiêu chí ĐG

Lớp

Giáo viên

Sinh viên

Lớp môn học

Nguyễn Bảo An 121214

Chương trình

Tạo mới

Tạo chương trình

Mã chương trình

Tên chương trình

Mô tả:

IT Công nghệ t

Trình độ đào tạo: Đại h

Mã ngành đào tạo: 748

Loại hình đào tạo: Chín

Số tín chỉ yêu cầu: 150

Thời gian đào tạo: 4 n

Đối tượng tuyển sinh:

Thang điểm: Theo quy

Điều kiện tốt nghiệp: T

Văn bằng tốt nghiệp: K

Chương trình đào tạo chuẩn tham khảo:

i. Chương trình đào tạo Cử nhân ngành Công nghệ Thông tin của Trường Đại học Công nghệ Thông tin, Đại học quốc gia Tp. HCM.

ii. Chương trình đào tạo Cử nhân ngành Công nghệ Thông tin của Trường Đại học Khoa học Tự nhiên, Đại học quốc gia Tp. HCM.

iii. Chương trình đào tạo Cử nhân ngành Công nghệ Thông tin của Trường Đại học Georgia Southern.

iv. Chương trình đào tạo Cử nhân ngành Công nghệ Thông tin của Trường Đại học California State University, Northridge.

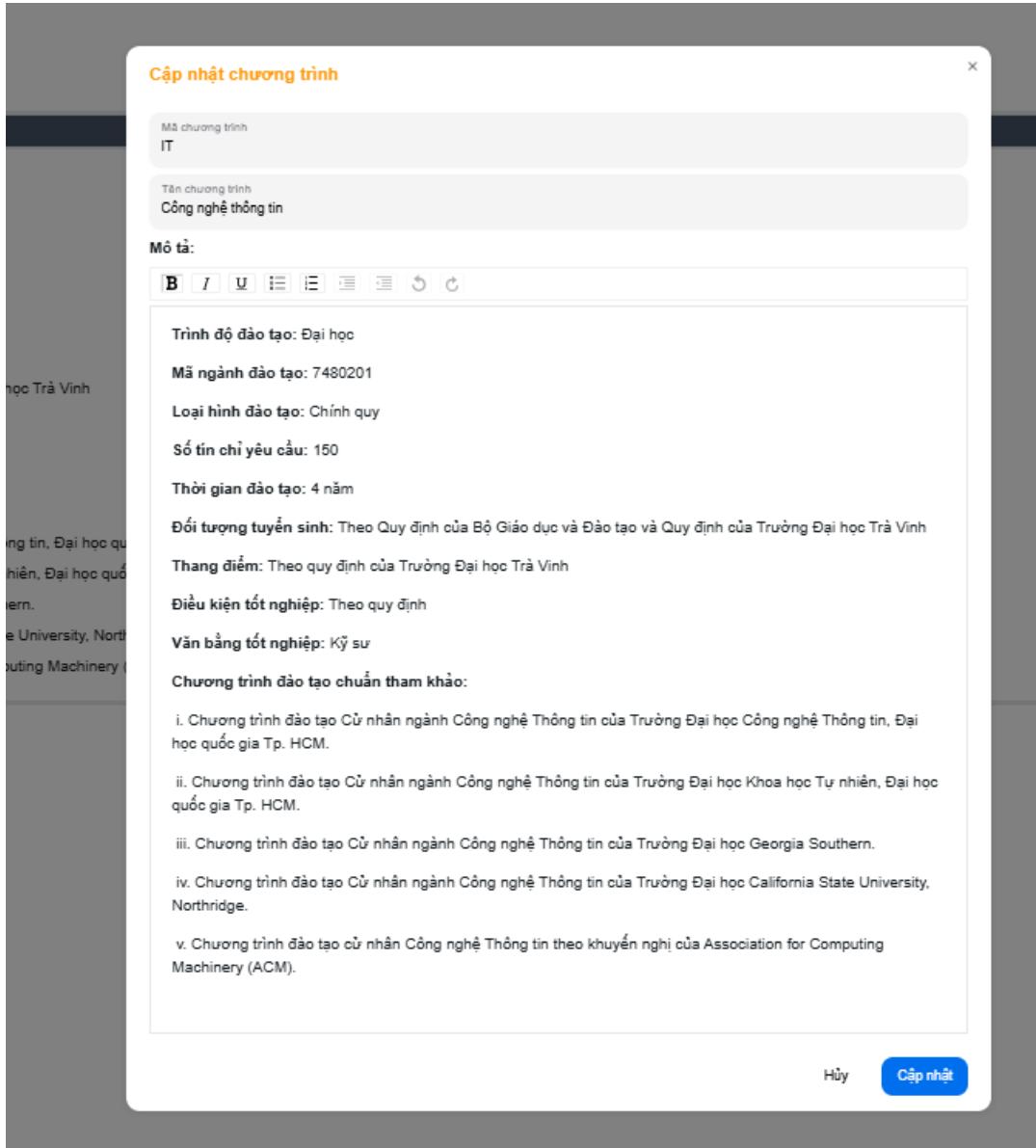
v. Chương trình đào tạo cử nhân Công nghệ Thông tin theo khuyến nghị của Association for Computing Machinery (ACM).

Hủy

Tạo

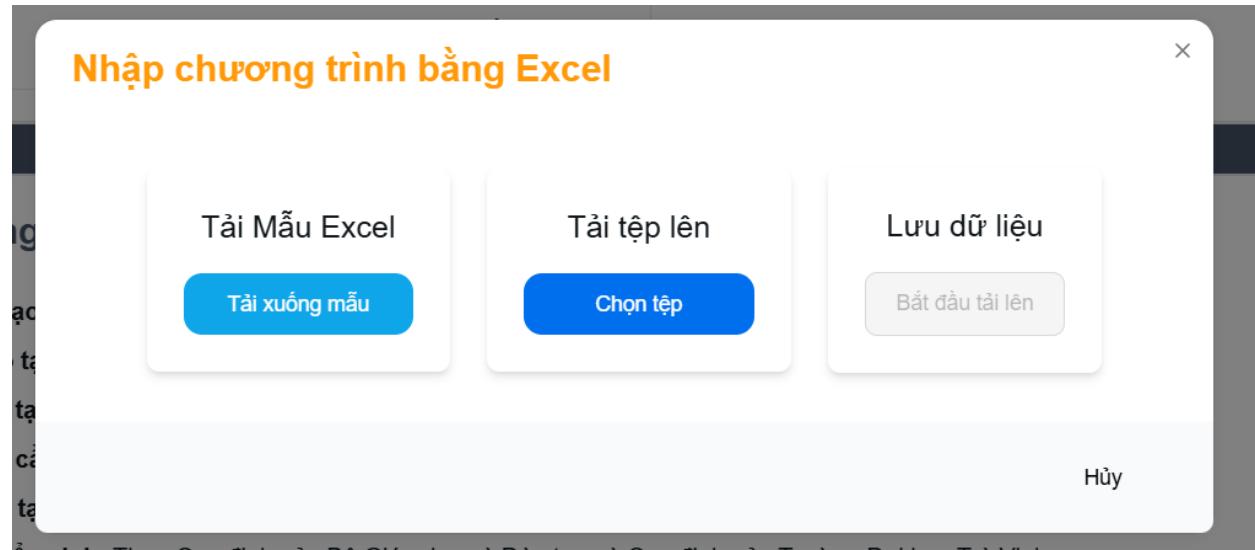
Hình 4.2. Chức năng tạo chương trình

Để thêm một chương trình đào tạo, quản trị viên nhấn nút “Tạo mới”. Một cửa sổ nhập liệu sẽ xuất hiện, cho phép nhập thông tin cần thiết. Sau khi hoàn tất, quản trị viên nhấn “Tạo”, và hệ thống sẽ gửi yêu cầu HTTP POST đến API ‘/api/admin/program’. Nếu dữ liệu được lưu thành công, hệ thống sẽ hiển thị thông báo “Tạo thành công”.



Hình 4.3. Chức năng cập nhật chương trình

Để thực hiện chức năng cập nhật thông tin của một chương trình đào tạo, quản trị viên sẽ nhấn vào nút “Chỉnh sửa” trên giao diện. Một cửa sổ cập nhật sẽ xuất hiện, cho phép quản trị viên chỉnh sửa các thông tin cần thiết. Sau khi hoàn tất việc chỉnh sửa, quản trị viên nhấn nút “Cập nhật”. Hệ thống sẽ gửi yêu cầu HTTP PUT đến API với đường dẫn ‘/api/admin/program/IT’. Nếu quá trình cập nhật dữ liệu thành công, hệ thống sẽ hiển thị thông báo “Cập nhật thành công”.



Hình 4.4. Cửa sổ nhập chương trình bằng Excel

	A	B	C	D	E
1	Mã chương trình	Tên chương trình	Mô tả (Html)		
2	IT	Công nghệ thông tin	<p><strong>Trình độ đào tạo:</strong> Đại học</p><p><strong>Mã ngành đào tạo:</strong> 7480		
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					

Hình 4.5. Mẫu nhập liệu chương trình bằng file Excel

Để thêm dữ liệu bằng Excel, quản trị viên nhấn nút “Nhập bằng Excel”. Một cửa sổ chức năng sẽ xuất hiện, cho phép quản trị viên nhấn “Tải xuống mẫu” để gửi yêu cầu đến API có đường dẫn ‘/api/admin/program/templates/post’. API sẽ trả về tệp mẫu Excel chứa thông tin về CDR, mục tiêu chương trình, và mối quan hệ giữa chúng.

Sau khi tải xuống và điền dữ liệu vào tệp mẫu, quản trị viên chọn “Chọn tệp” để tải lên tệp Excel đã chỉnh sửa. Để hoàn tất, quản trị viên nhấn “Bắt đầu tải lên”. Hệ thống sẽ gửi tệp Excel đến API có đường dẫn ‘/api/admin/importExcel/program’ thông qua yêu cầu HTTP POST. Nếu thành công, hệ thống sẽ thông báo kết quả.

#### 4.1.2. Quản lý CDR của chương trình (QL-02)

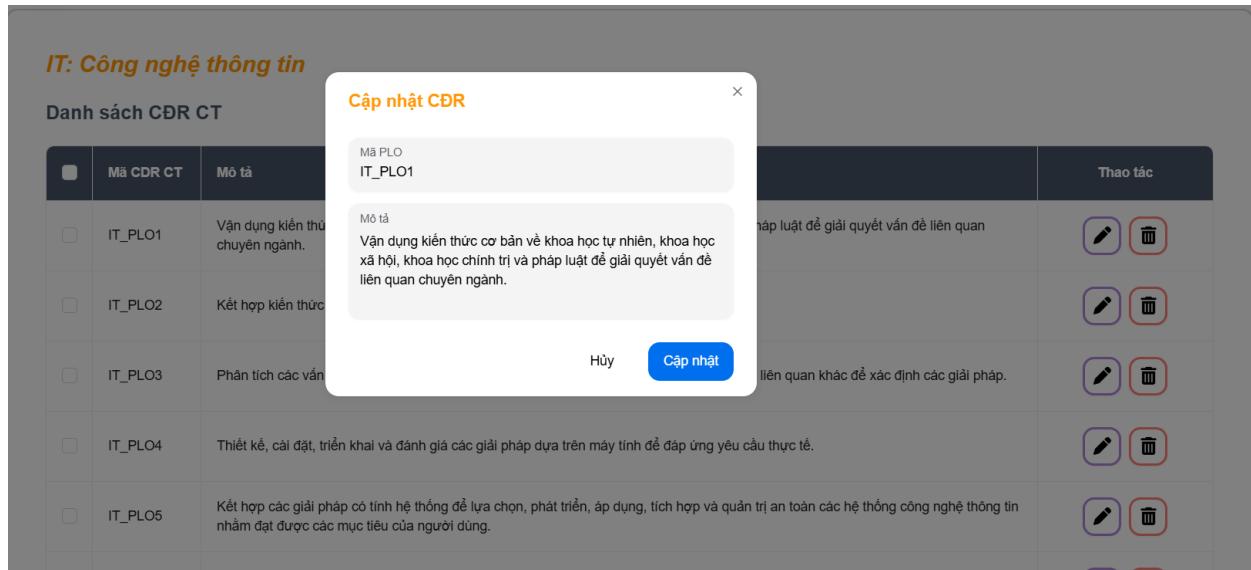
	Mã CDR CT	Mô tả	Thao tác
<input type="checkbox"/>	IT_PLO1	Vận dụng kiến thức cơ bản về khoa học tự nhiên, khoa học xã hội, khoa học chính trị và pháp luật để giải quyết vấn đề liên quan chuyên ngành.	
<input type="checkbox"/>	IT_PLO2	Kết hợp kiến thức chuyên môn để giải quyết các vấn đề liên quan chuyên ngành.	
<input type="checkbox"/>	IT_PLO3	Phân tích các vấn đề tính toán phức tạp, áp dụng các nguyên lý của máy tính và kiến thức liên quan khác để xác định các giải pháp.	
<input type="checkbox"/>	IT_PLO4	Thiết kế, cài đặt, triển khai và đánh giá các giải pháp dựa trên máy tính để đáp ứng yêu cầu thực tế.	
<input type="checkbox"/>	IT_PLO5	Kết hợp các giải pháp có tính hệ thống để lựa chọn, phát triển, áp dụng, tích hợp và quản trị an toàn các hệ thống công nghệ thông tin nhằm đạt được các mục tiêu của người dùng.	

Hình 4.6. Trang quản lý CDR chương trình

Khi quản trị viên truy cập trang danh sách các CDR của chương trình, hệ thống hiển thị giao diện cho phép xem danh sách các CDR bằng cách gửi yêu cầu HTTP GET đến API có đường dẫn '/api/admin/plos/isDelete/false'. API sẽ trả về danh sách các CDR.

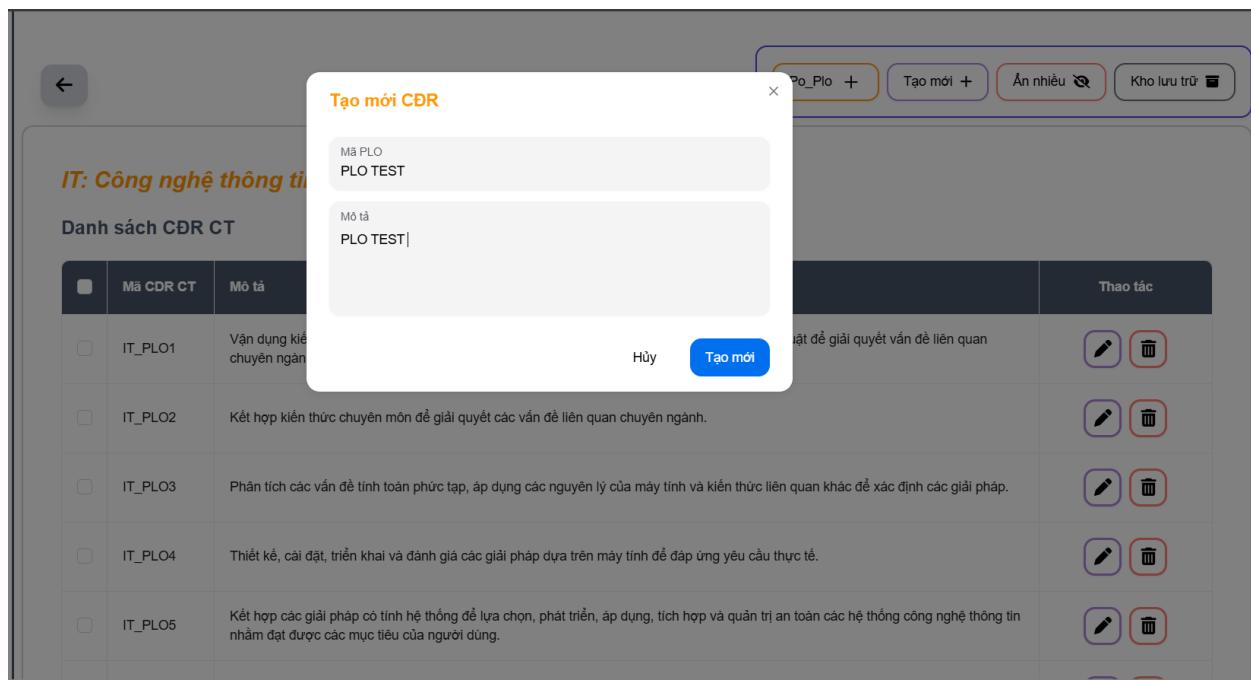
Để trực quan hóa mã phân cấp của chương trình, hệ thống gọi đến API có đường dẫn '/api/admin/program/IT', cung cấp thông tin về mã và tên chương trình để quản trị viên dễ dàng quản lý các CDR trong ngữ cảnh của cấu trúc chương trình đào tạo.

Để ẩn CDR, quản trị viên có thể chọn một hoặc nhiều CDR từ danh sách. Đối với một CDR, nhấn vào biểu tượng thùng rác sẽ gửi yêu cầu HTTP PUT đến API có đường dẫn '/api/admin/plo/\${plo\_id}/softDelete', trong đó \${plo\_id} là định danh của CDR. Đối với nhiều CDR, nhấn vào nút “Ẩn nhiều” sẽ gửi yêu cầu HTTP PUT đến API có đường dẫn '/api/admin/plos/softDelete' với mảng định danh các CDR cần ẩn. Sau khi yêu cầu hoàn tất, hệ thống sẽ cập nhật danh sách và thông báo cho quản trị viên.



Hình 4.7. Chức năng cập nhật CDR chương trình

Để thực hiện chức năng cập nhật CDR, hệ thống cho phép quản trị viên chọn CDR cần chỉnh sửa từ danh sách và nhấn vào biểu tượng bút để mở cửa sổ chỉnh sửa. Sau khi hoàn tất các chỉnh sửa, quản trị viên nhấn “Cập nhật”. Hệ thống sẽ gửi yêu cầu HTTP PUT đến API có đường dẫn ‘/api/admin/plos/{plo\_id}’ , trong đó {plo\_id} là định danh của CDR cần cập nhật. Nếu quá trình cập nhật thành công, hệ thống sẽ thông báo về sự thay đổi.



Hình 4.8. Chức năng tạo mới chương trình

Để thực hiện chức năng thêm một CDR, quản trị viên nhấn vào nút “Tạo mới” trên giao diện người dùng. Khi đó, một cửa sổ nhập liệu sẽ xuất hiện, cho phép nhập các thông

tin cần thiết để tạo chuẩn đầu ra. Sau khi hoàn tất, quản trị viên nhấn “Tạo”. Hệ thống sẽ gửi yêu cầu HTTP POST đến API với đường dẫn ‘/api/admin/plo’. Nếu quá trình lưu trữ thành công, hệ thống sẽ hiển thị thông báo xác nhận.

	Tên PLO	Mô tả	Form
<input type="checkbox"/>	PLO TEST	PLO TEST	

Hình 4.9. Trang quản lý CDR chương trình đã ẩn

Để truy cập kho lưu trữ các CDR đã bị xóa, quản trị viên nhấn vào nút “Kho lưu trữ” trên giao diện. Hệ thống sẽ lấy dữ liệu từ API ‘/api/admin/plos/isDelete/true’ và hiển thị danh sách các CDR đã bị xóa.

Để khôi phục một CDR, quản trị viên nhấn vào biểu tượng “Quay lại” bên cạnh CDR cần khôi phục. Hệ thống gửi yêu cầu HTTP PUT đến API có đường dẫn ‘/api/admin/plo/\${\_id}/softDelete’, trong đó \${\_id} là định danh của CDR. Để khôi phục nhiều CDR cùng lúc, quản trị viên chọn các CDR cần khôi phục và nhấn vào biểu tượng “Quay lại”. Hệ thống gửi yêu cầu HTTP PUT đến API có đường dẫn ‘/api/admin/plos/softDelete’ với mảng các định danh của các CDR.

Để xóa một CDR cụ thể, quản trị viên chọn CDR từ danh sách và nhấn vào biểu tượng thùng rác. Hệ thống gửi yêu cầu HTTP DELETE đến API ‘/api/admin/plo/{plo\_id}’, trong đó {plo\_id} là định danh của CDR. Để xóa nhiều CDR cùng lúc, quản trị viên chọn nhiều CDR từ danh sách và nhấn vào biểu tượng thùng rác. Hệ thống sẽ gửi yêu cầu HTTP DELETE đến API có đường dẫn ‘/api/admin/plos/multiple’ với mảng các định danh của các CDR cần xóa.

### 4.1.3. Quản lý mục tiêu chương trình (QL-03)

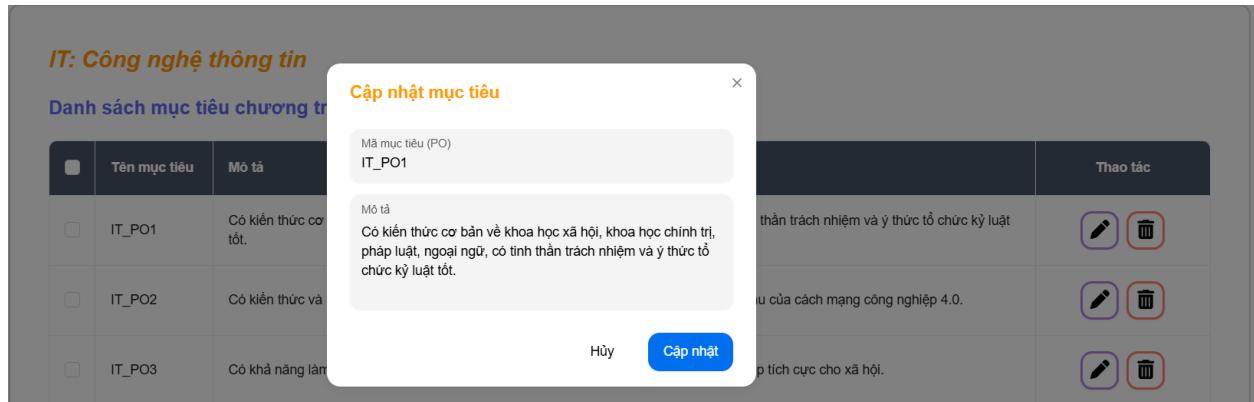
	Tên mục tiêu	Mô tả	Thao tác
<input type="checkbox"/>	IT_PO1	Có kiến thức cơ bản về khoa học xã hội, khoa học chính trị, pháp luật, ngoại ngữ, có tinh thần trách nhiệm và ý thức tổ chức kỷ luật tốt.	
<input type="checkbox"/>	IT_PO2	Có kiến thức và kỹ năng chuyên môn trong lĩnh vực công nghệ thông tin, đáp ứng yêu cầu của cách mạng công nghiệp 4.0.	
<input type="checkbox"/>	IT_PO3	Có khả năng làm việc hiệu quả trong lĩnh vực công nghệ thông tin, tạo ra những đóng góp tích cực cho xã hội.	
<input type="checkbox"/>	IT_PO4	Có khả năng làm việc độc lập hoặc làm việc theo nhóm, hướng dẫn, giám sát những người khác thực hiện nhiệm vụ trong lĩnh vực công nghệ thông tin.	
<input type="checkbox"/>	IT_PO5	Có khả năng khám phá tri thức, giải quyết vấn đề, tư duy hệ thống, phát triển phẩm chất cá nhân và nghề nghiệp thông qua hoạt động học tập suốt đời.	

Hình 4.10. Trang quản lý mục tiêu chương trình

Quản trị viên có thể truy cập danh sách các CDR của một mã môn học bằng cách gửi yêu cầu HTTP GET đến API tại '/api/admin/clos? subject\_id =\${id} & isDelete=false', trong đó \${id} là mã môn học. API sẽ trả về các CDR chưa bị xóa của môn học đó.

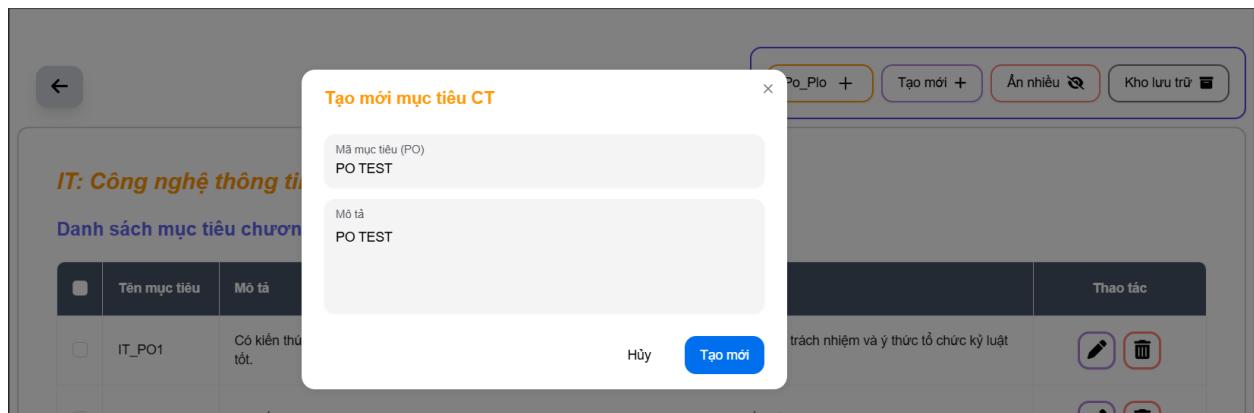
Hệ thống cung cấp thông tin chi tiết về mã môn học và các chi tiết liên quan để hỗ trợ quản trị viên trong việc quản lý cấu trúc phân cấp của môn học, thông qua API '/api/admin/subject/\${id}', nơi \${id} là mã của môn học.

Để ẩn một hoặc nhiều PO, quản trị viên chọn PO từ danh sách và nhấn biểu tượng thùng rác hoặc “Ẩn nhiều”. Hệ thống gửi yêu cầu HTTP PUT đến API có đường dẫn '/api/admin/po/\${po\_id}/softDelete' để ẩn một PO, hoặc tại '/api/admin/pos/softDelete' với mảng định danh của các PO cần ẩn. Sau khi ẩn thành công, hệ thống sẽ thông báo và cập nhật danh sách PO.



Hình 4.11. Chức năng cập nhật mục tiêu chương trình

Sau khi xem danh sách các PO, quản trị viên chọn PO cần cập nhật bằng cách nhấp vào biểu tượng bút. Cửa sổ chỉnh sửa sẽ mở ra, cho phép chỉnh sửa thông tin PO. Sau khi thực hiện thay đổi, quản trị viên nhấp “Cập nhật”. Hệ thống gửi yêu cầu HTTP PUT đến API tại đường dẫn ‘/api/admin/pos/{po\_id}’, trong đó {po\_id} là định danh của PO. Nếu cập nhật thành công, hệ thống sẽ thông báo cho quản trị viên về sự thay đổi.



Hình 4.12. Chức năng tạo mới mục tiêu chương trình

Để thêm một PO mới, quản trị viên nhấp nút “Tạo mới” trên giao diện. Cửa sổ nhập liệu sẽ mở ra, cho phép nhập thông tin cần thiết. Sau khi hoàn tất, quản trị viên nhấp “Tạo”. Hệ thống gửi yêu cầu HTTP POST đến API tại đường dẫn ‘/api/admin(po’. Nếu lưu trữ thành công, hệ thống sẽ thông báo cho quản trị viên.

	Tên PO	Mô tả	Form
<input type="checkbox"/>	PO TEST	PO TEST	

Hình 4.13. Trang quản lý mục tiêu chương trình đã ẩn

Để truy cập kho lưu trữ các PO đã bị xóa, quản trị viên nhấn vào nút “Kho lưu trữ” trên giao diện. Hệ thống sẽ gửi yêu cầu HTTP GET đến API ‘/api/admin/pos/isDelete/true’ để hiển thị danh sách các PO với trạng thái isDelete là true.

Để khôi phục một PO đã bị ẩn, quản trị viên nhấn vào biểu tượng “Quay lại” bên cạnh PO cần khôi phục. Hệ thống gửi yêu cầu HTTP PUT đến API tại đường dẫn ‘/api/admin/pos/{\_id}/softDelete’, trong đó \${\_id} là định danh của PO. Để khôi phục nhiều PO cùng lúc, quản trị viên chọn các PO cần khôi phục và nhấn vào biểu tượng “Quay lại”. Hệ thống gửi yêu cầu HTTP PUT đến API có đường dẫn ‘/api/admin/pos/softDelete’ với mảng định danh các PO cần khôi phục.

Để xóa một PO cụ thể, quản trị viên chọn PO từ danh sách và nhấn vào biểu tượng thùng rác. Hệ thống gửi yêu cầu HTTP DELETE đến API ‘/api/admin/pos/\${po\_id}’, trong đó \${po\_id} là định danh của PO cần xóa. Để xóa nhiều PO đồng thời, quản trị viên chọn nhiều PO và nhấn vào biểu tượng thùng rác, sau đó gửi yêu cầu HTTP DELETE đến API có đường dẫn ‘/api/admin/pos/multiple’ kèm theo mảng định danh các PO cần xóa.

#### 4.1.4. Quản lý mối quan hệ giữa PLO và PO (QL-04)

PLO	Nội dung	IT_PO1	IT_PO2	IT_PO3	IT_PO4	IT_PO5
IT_PLO1	Vận dụng kiến thức cơ bản về khoa học tự nhiên, khoa học xã hội, khoa học chính trị và pháp luật để giải quyết vấn đề liên quan chuyên ngành.	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
IT_PLO2	Kết hợp kiến thức chuyên môn để giải quyết các vấn đề liên quan chuyên ngành.	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
IT_PLO3	Phân tích các vấn đề tính toán phức tạp, áp dụng các nguyên lý của máy tính và kiến thức liên quan khác để xác định các giải pháp.	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
IT_PLO4	Thiết kế, cài đặt, triển khai và đánh giá các giải pháp dựa trên máy tính để đáp ứng yêu cầu thực tế.	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
IT_PLO5	Kết hợp các giải pháp có tính hệ thống để lựa chọn, phát triển, áp dụng, tích hợp và quản trị an toàn các hệ thống công nghệ thông tin nhằm đạt được các mục tiêu của người dùng.	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
IT_PLO6	Sử dụng tốt tiếng Anh trong soạn thảo, đọc tài liệu và giao tiếp, đạt bậc 4/6 theo	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Hình 4.14. Trang quản lý ánh xạ PO và PLO

Khi quản trị viên truy cập trang ánh xạ giữa PO và PLO, hệ thống sẽ thực hiện các bước sau. Đầu tiên, hệ thống gửi yêu cầu HTTP GET đến API có đường dẫn ‘/api/admin/pos/isDelete/false’ để lấy danh sách các PO chưa bị xóa. Tiếp theo, hệ thống tiếp tục gửi yêu cầu HTTP GET đến API có đường dẫn ‘/api/admin/plos/isDelete/false’ nhằm nhận danh sách các PLO chưa bị xóa. Cuối cùng, để lấy thông tin về ánh xạ giữa PO và PLO, hệ thống sẽ gửi yêu cầu HTTP GET đến API có đường dẫn ‘/api/admin/po-plo’ và nhận về thông tin liên kết giữa các PO và PLO.

Dựa trên dữ liệu từ ba API này, hệ thống tạo ra một mảng ánh xạ, cung cấp cái nhìn tổng quan về mối quan hệ giữa PO và PLO, hỗ trợ quản trị viên trong việc quản lý và điều chỉnh các liên kết.

Ngoài chức năng xem ánh xạ, hệ thống cho phép quản trị viên thay đổi ánh xạ bằng cách chọn các liên kết cần cập nhật và nhấn nút lưu. Hệ thống sẽ gửi yêu cầu HTTP DELETE để xóa các liên kết hiện tại hoặc HTTP POST để thêm các liên kết mới, giúp duy trì và điều chỉnh các liên kết giữa PO và PLO một cách linh hoạt.

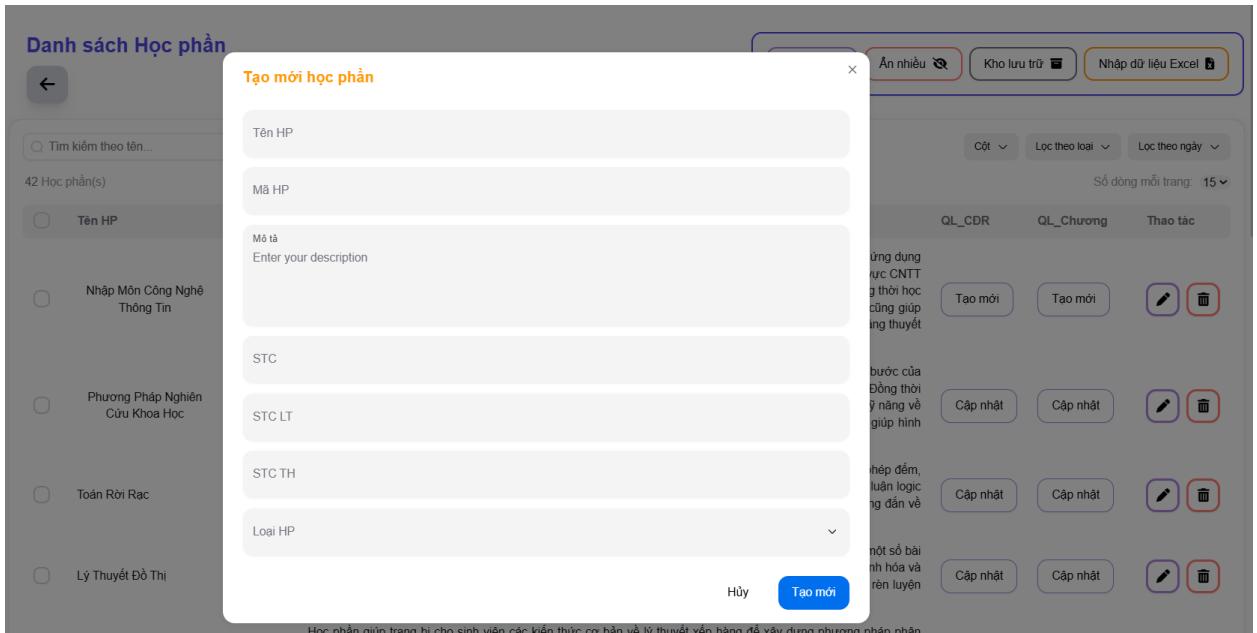
#### 4.1.5. Quản lý môn học (QL-05)

Tên HP	Mã HP	Mô tả	QL_CDR	QL_Chương	Thao tác
Nhập Môn Công Nghệ Thông Tin	IT01	Học phần giúp trang bị cho sinh viên các kiến thức cơ bản về ngành CNTT, trinh bày được những lĩnh vực ứng dụng cơ bản của CNTT trong cuộc sống, những phẩm chất, kỹ năng, kiến thức mà những người làm trong lĩnh vực CNTT cần có. Lập trình được một ứng dụng cơ bản thông qua ngôn ngữ lập trình trực quan Alice và Scratch. Đồng thời học phần cũng nhằm rèn luyện cho sinh viên các kỹ năng làm việc nhóm và kỹ năng thuyết trình. Học phần cũng giúp hình thành cho sinh viên thái độ và nhận thức đúng đắn về ngành CNTT và kỹ năng làm việc nhóm và kỹ năng thuyết trình.	<a href="#">Tạo mới</a>	<a href="#">Tạo mới</a>	<a href="#"></a> <a href="#"></a>
Phương Pháp Nghiên Cứu Khoa Học	IT03	Học phần giúp trang bị cho sinh viên các kiến thức cơ bản về phương pháp nghiên cứu như trình bày các bước của quy trình nghiên cứu khoa học và áp dụng các bước của quy trình nghiên cứu khoa học vào đề tài cụ thể. Đồng thời môn học còn trang bị cho sinh viên các kỹ năng cơ bản và cần thiết cho người làm nghiên cứu đó là các kỹ năng về cá nhân, nghề nghiệp, phẩm chất cũng như các kỹ năng về giao tiếp và làm việc nhóm. Học phần cũng giúp hình thành cho sinh viên rèn luyện thái độ và nhận thức đúng đắn về chuyên ngành đang theo học.	<a href="#">Cập nhật</a>	<a href="#">Cập nhật</a>	<a href="#"></a> <a href="#"></a>
Toán Rời Rạc	IT04	Học phần giúp trang bị cho sinh viên các kiến thức cơ bản về toàn học ứng dụng trong tin học như logic, phép đếm, quan hệ và đại số Boolean. Đồng thời học phần cũng nhằm rèn luyện cho sinh viên các kỹ năng tư duy, suy luận logic và chứng minh toán học. Học phần cũng giúp hình thành cho sinh viên rèn luyện thái độ và nhận thức đúng đắn về vai trò của toán học trong ngành công nghệ thông tin.	<a href="#">Cập nhật</a>	<a href="#">Cập nhật</a>	<a href="#"></a> <a href="#"></a>

Hình 4.15. Trang quản lý môn học

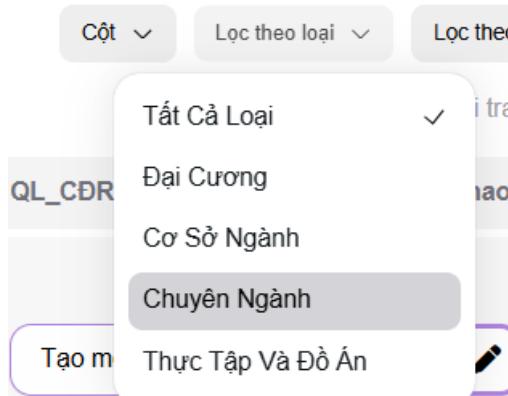
Khi quản trị viên truy cập trang danh sách môn học, hệ thống cung cấp giao diện để xem danh sách môn học. Quản trị viên gửi yêu cầu HTTP GET đến API có đường dẫn ‘/api /admin/subjects/isDelete/false’ để nhận danh sách các môn học chưa bị xóa.

Để ẩn một hoặc nhiều môn học, quản trị viên chọn môn học cần ẩn và nhấn biểu tượng thùng rác hoặc nút “Ẩn nhiều”. Hệ thống gửi yêu cầu HTTP PUT đến API có đường dẫn ‘/api/admin/subject/\${\_id}/softDelete’ để ẩn một môn học, hoặc ‘/api/admin/subjects/softDelete’ với mảng chứa các định danh của các học môn học ẩn. Sau khi ẩn thành công, hệ thống sẽ thông báo và cập nhật giao diện.

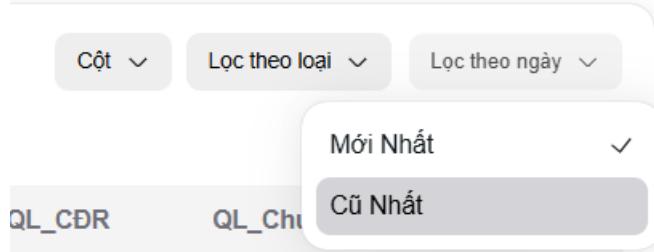


Hình 4.16. Chức năng tạo mới môn học

Để thêm môn học mới, quản trị viên nhấn nút “Tạo mới” trên giao diện. Cửa sổ nhập liệu xuất hiện cho phép nhập thông tin cần thiết. Sau khi hoàn tất, quản trị viên nhấn nút “Tạo”. Hệ thống gửi yêu cầu HTTP POST đến API có đường dẫn ‘/api/admin/subject’. Nếu thành công, hệ thống thông báo thêm môn học mới đã hoàn tất.



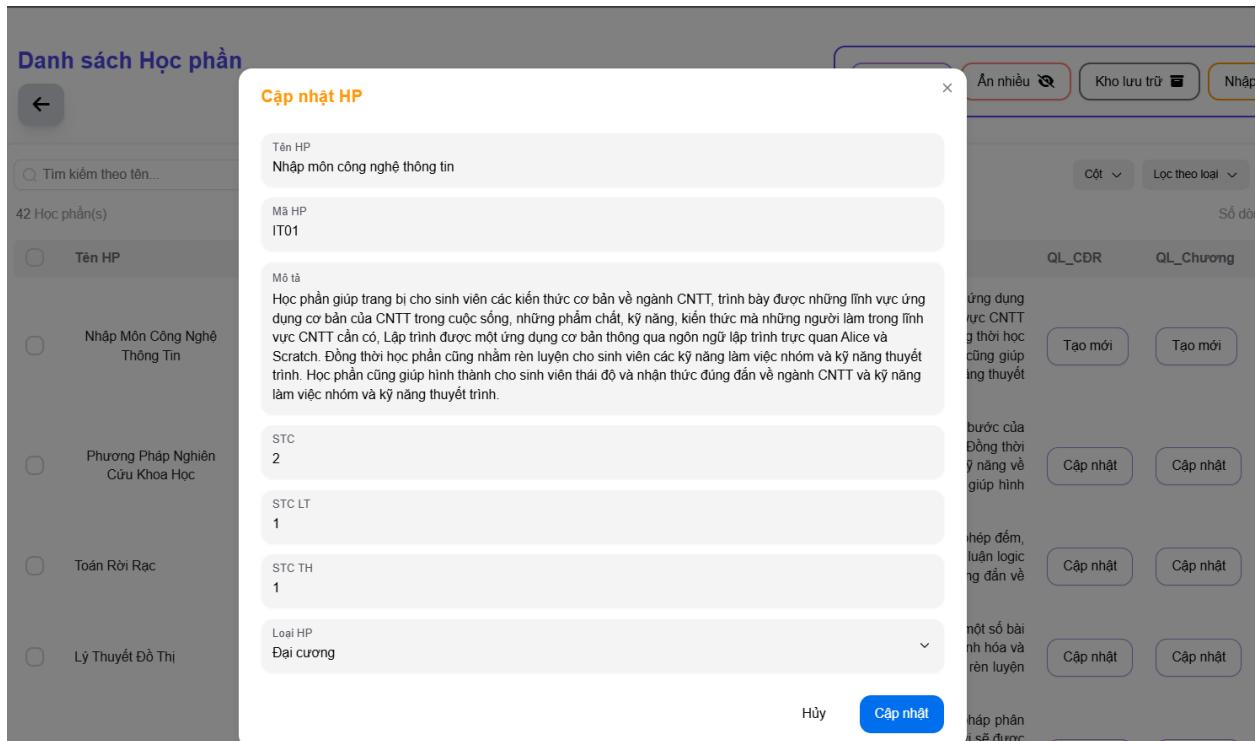
Hình 4.17. Chức năng lọc theo loại



Hình 4.18. Chức năng lọc theo ngày

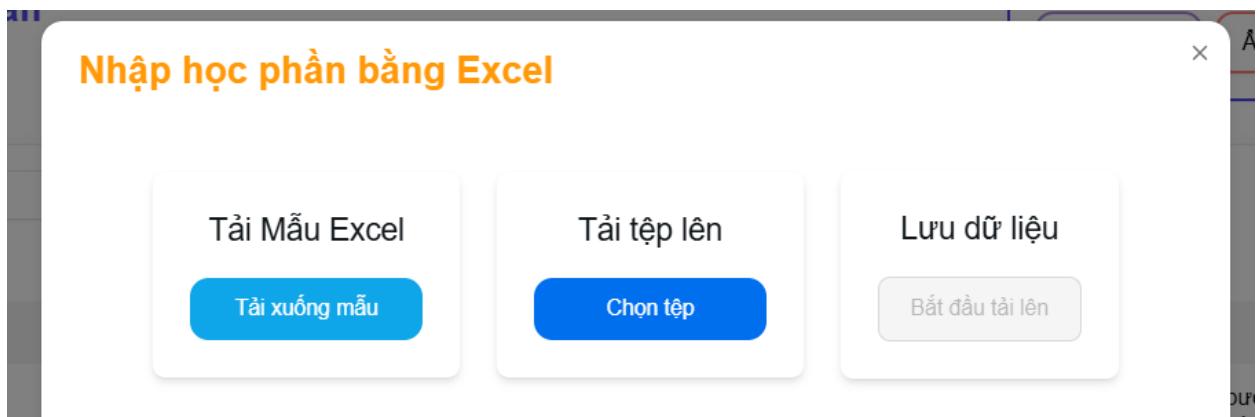
Chức năng lọc danh sách môn học giúp quản trị viên quản lý hiệu quả hơn. Hệ thống

cung cấp tùy chọn lọc theo loại môn học (Đại cương, Cơ sở ngành, Chuyên ngành, Thực tập, Đồ án) và sắp xếp theo ngày (mới nhất hoặc cũ nhất).



Hình 4. 19. Chức năng cập nhật môn học

Để cập nhật thông tin môn học, quản trị viên nhấp vào biểu tượng bút tương ứng trên giao diện. Một cửa sổ cập nhật sẽ xuất hiện, cho phép thay đổi thông tin cần thiết. Sau khi hoàn tất chỉnh sửa, quản trị viên nhấp nút “Cập nhật”. Hệ thống gửi yêu cầu HTTP PUT đến API với đường dẫn ‘/api/admin/subject/\${subject\_id}’, trong đó \${subject\_id} là định danh của môn học. Nếu cập nhật thành công, hệ thống sẽ thông báo “Cập nhật thành công”.



Hình 4.20. Cửa sổ nhập môn học bằng Excel

A	B	C	D	E
Tên HP	Mã HP	Mô tả	Số tin chỉ	STC LT
1 Nhập môn công nghệ thông tin	IT01	Học phần giúp trang bị cho sinh viên các kiến thức cơ bản về ngành CNTT, trình bày được những lĩnh vực ứng dụng cơ bản của CNTT trong cuộc sống, những phẩm chất, kỹ năng, kiến thức mà những người làm trong lĩnh vực CNTT cần có, Lập trình được một ứng dụng cơ bản thông qua ngôn ngữ lập trình trực quan Alice và Scratch. Đồng thời học phần cũng nhằm rèn luyện cho sinh viên các kỹ năng làm việc nhóm và kỹ năng thuyết trình. Học phần cũng giúp hình thành cho sinh viên thái độ và nhận thức đúng đắn về ngành CNTT và kỹ năng làm việc nhóm và kỹ năng thuyết trình.	2	1
2 Thống kê và phân tích dữ liệu	IT02	Học phần giúp trang bị cho sinh viên các kiến thức cơ bản về xác suất và thống kê để phân tích dữ liệu trong nghiên cứu. Sinh viên sẽ được học cách biên tập dữ liệu, cách mô tả dữ liệu bằng biểu đồ và bảng các đặc trưng liệu trong thống kê, cách thu thập dữ liệu, thiết kế nghiên cứu và kiểm định các giả thuyết thông kê trên ngôn ngữ Python/R. Nhờ đó sinh viên có thể ứng dụng các kiến thức của môn học để thực hiện các nghiên cứu khoa học và học các môn học có ứng dụng kiến thức thống kê như: Trí tuệ nhân tạo, Khai phá dữ liệu... Đồng thời, kỹ năng sử dụng ngôn ngữ Python/R giúp sinh viên có thể dễ dàng tham gia các cộng đồng phân tích số liệu để có thể tự học và nghiên cứu sâu hơn. Học phần cũng giúp sinh viên hình thành thái độ và nhận thức đúng đắn về vai trò của thống kê và khoa học dữ liệu trong bối cảnh hiện đại.	3	2
3 Phương pháp nghiên cứu khoa học	IT03	Học phần giúp trang bị cho sinh viên các kiến thức cơ bản về phương pháp nghiên cứu như: trình bày các bước của quy trình nghiên cứu khoa học và áp dụng các bước của quy trình nghiên cứu khoa học vào để tài chí. Đồng thời môn học còn trang bị cho sinh viên các kỹ năng cơ bản và cần thiết cho người làm nghiên cứu đó là các kỹ năng về cá nhân, nghề nghiệp, phẩm chất cũng như các kỹ năng về giao tiếp và làm việc nhóm.	2	1
4 Toán rời rạc	IT04	Học phần cũng giúp hình thành cho sinh viên rèn luyện thái độ và nhận thức đúng đắn về chuyên ngành đang theo học. Học phần giúp trang bị cho sinh viên các kiến thức cơ bản về toán học ứng dụng trong tin học như logic, phép đếm, quan hệ và đại số Boolean. Đồng thời học phần cũng nhằm rèn luyện cho sinh viên các kỹ năng tư duy, suy luận logic và chứng minh toán học. Học phần cũng giúp hình thành cho sinh viên rèn luyện thái độ và nhận thức đúng đắn về vai trò của toán học trong ngành công nghệ thông tin.	2	1

Hình 4.21. Mẫu nhập liệu môn học bằng file Excel

Để nhập dữ liệu môn học bằng Excel, quản trị viên nhấn vào nút “Nhập bằng Excel” trên giao diện. Một cửa sổ sẽ mở ra, cho phép tải xuống tệp mẫu bằng cách nhấn vào “Tải xuống mẫu”. API với đường dẫn ‘/subject/templates/post’ sẽ trả về tệp Excel mẫu chứa thông tin cần thiết về môn học và chuẩn đầu ra.

Sau khi điền dữ liệu vào tệp mẫu, quản trị viên nhấn nút “Chọn tệp” để tải lên tệp Excel đã chỉnh sửa. Để hoàn tất, nhấn nút “Bắt đầu tải lên”. Hệ thống sẽ gửi tệp Excel đến API với đường dẫn ‘/api/admin/importExcel/subject’ qua yêu cầu HTTP POST. Nếu tải lên thành công, hệ thống sẽ thông báo kết quả cho quản trị viên.

Hình 4.22. Trang quản lý môn học đã ẩn

Khi quản trị viên nhấn vào nút “Kho lưu trữ” trên giao diện quản lý, hệ thống gửi yêu cầu HTTP GET đến API với đường dẫn ‘/api/admin/subjects/isDelete/true’ để lấy danh

sách các môn học đã bị xóa (isDelete = true).

Để khôi phục một môn học bị ẩn, quản trị viên nhấn vào biểu tượng “Quay lại” bên cạnh môn học đó. Hệ thống gửi yêu cầu HTTP PUT đến API với đường dẫn ‘/api/admin/subject/\${\_id}/softDelete’, trong đó \${\_id} là định danh của môn học. Để khôi phục nhiều môn học cùng lúc, quản trị viên chọn các môn học cần khôi phục và nhấn vào biểu tượng “Quay lại”, sau đó gửi yêu cầu HTTP PUT đến API với đường dẫn ‘/api/admin/subjects/softDelete’ với mảng định danh các môn học.

Để xóa một môn học cụ thể, quản trị viên chọn môn học và nhấn vào biểu tượng thùng rác. Hệ thống gửi yêu cầu HTTP DELETE đến API với đường dẫn ‘/api/admin/subject/\${\_id}’, trong đó \${\_id} là định danh của môn học. Để xóa nhiều môn học đồng thời, quản trị viên chọn các môn học và nhấn vào biểu tượng thùng rác, sau đó gửi yêu cầu HTTP DELETE đến API với đường dẫn ‘/api/admin/subjects/multiple’ với mảng định danh các môn học.

#### 4.1.6. Quản lý CDR của môn học (QL-06)

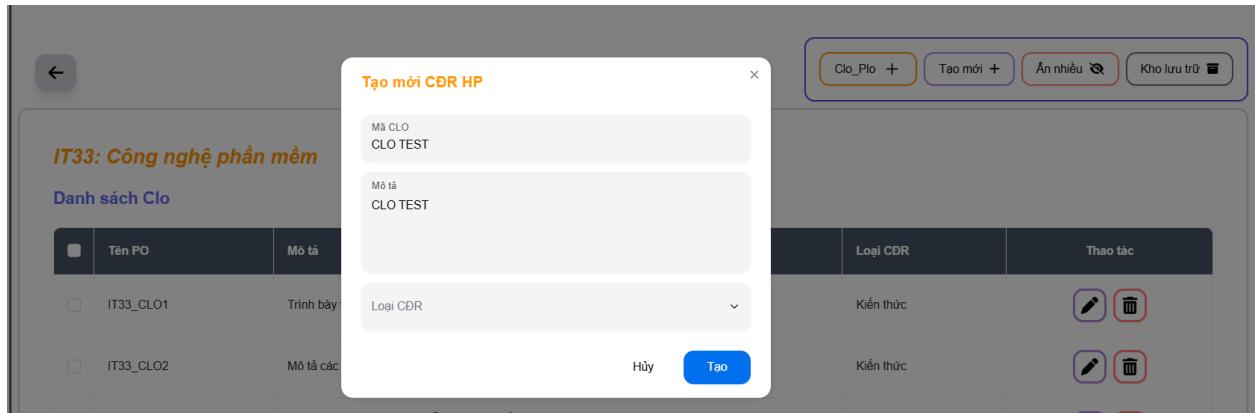
The screenshot shows a web-based application interface for managing CDRs. On the left, there is a sidebar with a tree menu under the 'SET' category. The menu items include: Tổng quan, Chấm điểm, Chương trình (selected), Mục tiêu CT, CDR CT, PO\_PLO, Học phần, Tiêu chí ĐG, Lớp, Giáo viên, Sinh viên, Lớp môn học, API, and a user profile for 'Nguyễn Bảo An' (ID: 121214). The main content area has a header 'IT33: Công nghệ phần mềm' and a sub-header 'Danh sách Clo'. Below this is a table with the following data:

Clo	Tên PO	Mô tả	Loại CDR	Thao tác
	IT33_CLO1	Trình bày tổng quan về công nghệ phần mềm	Kiến thức	
	IT33_CLO2	Mô tả các mô hình phát triển phần mềm	Kiến thức	
	IT33_CLO3	Vận dụng mô hình đã học để xây dựng phần mềm	Kiến thức	
	IT33_CLO4	Vận dụng cách thức tổ chức và hoạt động nhóm hiệu quả	Kiến thức	
	IT33_CLO5	Sử dụng tiếng Anh để đọc tài liệu chuyên ngành	Kiến thức	
	IT33_CLO6	Thể hiện đạo đức nghề nghiệp, tinh thần trách nhiệm và tác phong chuyên nghiệp trong công việc	Kiến thức	

Hình 4.23. Trang quản lý CDR môn học

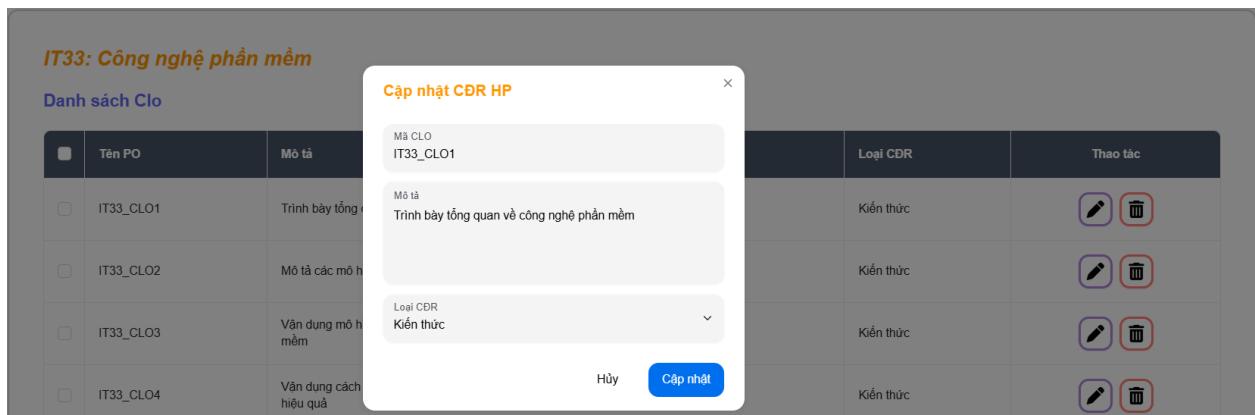
Khi quản trị viên truy cập trang xem danh sách các CDR của môn học, hệ thống cung cấp giao diện hiển thị toàn bộ danh sách các CDR liên quan đến môn học đó. Để lấy danh sách các CDR chưa bị xóa (isDelete = false), quản trị viên gửi yêu cầu HTTP GET đến API có đường dẫn ‘/api/admin/clos?subject\_id=\${id}&isDelete=false’, trong đó \${id} là định danh của môn học. API sẽ trả về danh sách các CDR hiện có của môn học.

Để ẩn một hoặc nhiều CDR, quản trị viên chọn CDR cần ẩn và nhấn biểu tượng thùng rác để gửi yêu cầu HTTP PUT đến API với đường dẫn ‘/api/admin/clo/\${\_id}/softDelete’, trong đó \${\_id} là định danh của CDR. Đối với việc ẩn nhiều CDR cùng lúc, quản trị viên chọn các CDR và nhấn nút “Ẩn nhiều”, hệ thống sẽ gửi yêu cầu HTTP PUT đến API với đường dẫn ‘/api/admin/clos/softDelete’ với mảng các định danh của các CDR cần ẩn. Sau khi hoàn tất, hệ thống sẽ thông báo và cập nhật danh sách CDR trên giao diện.



Hình 4.24. Chức năng tạo mới CDR môn học

Để thêm một CDR của môn học, quản trị viên nhấn vào nút “Tạo mới” trên giao diện. Một cửa sổ nhập liệu sẽ xuất hiện, cho phép nhập thông tin cần thiết để tạo CDR. Sau khi nhập dữ liệu xong, quản trị viên nhấn nút “Tạo”. Hệ thống gửi yêu cầu HTTP POST đến API có đường dẫn ‘/api/admin/clo’. Nếu lưu trữ thành công, hệ thống sẽ thông báo cho quản trị viên về việc thêm CDR mới.



Hình 4.25. Chức năng cập nhật CDR môn học

Sau khi xem danh sách CDR, quản trị viên chọn CDR cần cập nhật bằng cách nhấn vào biểu tượng bút. Cửa sổ chỉnh sửa xuất hiện để thực hiện thay đổi. Sau khi hoàn tất, quản trị viên nhấn nút “Cập nhật”. Hệ thống gửi yêu cầu HTTP PUT đến API có đường

dẫn ‘/api/admin/clo/\${clo\_id}’, trong đó `\${clo\_id}` là định danh của CDR. Nếu cập nhật thành công, hệ thống thông báo cho quản trị viên.

The screenshot shows a user interface for managing data. On the left, there's a sidebar with a tree view containing nodes like 'SET', 'Tổng quan', 'Chấm điểm', 'Chương trình >', 'Học phần', 'Tiêu chí ĐG', 'Lớp', 'Giáo viên', 'Sinh viên', 'Lớp môn học', and 'API'. Below this is a user profile icon with the name 'Nguyễn Bảo An' and ID '121214'. The main content area has a header 'IT33: Công nghệ phần mềm' and a sub-header 'Danh sách CDR HP đã ẩn'. It displays a table with three columns: 'Tên CLO' (Name), 'Mô tả' (Description), and 'Thao tác' (Actions). There is one row in the table with the value 'CLO TEST' in both columns, and two small icons in the 'Thao tác' column. At the bottom right of the table area, there are navigation buttons for 'Previous' and 'Next'.

Hình 4.26. Trang quản lý môn học đã ẩn

Để truy cập kho lưu trữ các CDR đã bị xóa, quản trị viên nhấp vào nút “Kho lưu trữ” trên giao diện. Hệ thống gửi yêu cầu HTTP GET đến API có đường dẫn ‘/api/admin/clos?subject\_id=\${id}&isDelete=true’ để nhận danh sách các CDR đã bị xóa.

Để khôi phục một CDR đã bị ẩn, quản trị viên nhấp vào biểu tượng “Quay lại” bên cạnh CDR cần khôi phục và gửi yêu cầu HTTP PUT đến API có đường dẫn ‘/api/admin/clo/\${\_id}/softDelete’. Để khôi phục nhiều CDR cùng lúc, quản trị viên chọn các CDR cần khôi phục và nhấp vào biểu tượng “Quay lại”, sau đó gửi yêu cầu HTTP PUT đến API có đường dẫn ‘/api/admin/clos/softDelete’ với mảng định danh các CDR.

Để xóa một CDR cụ thể, quản trị viên chọn CDR từ danh sách và nhấp vào biểu tượng thùng rác, gửi yêu cầu HTTP DELETE đến API ‘/api/admin/clo/\${\_id}’. Để xóa nhiều CDR đồng thời, quản trị viên chọn các CDR cần xóa và nhấp vào biểu tượng thùng rác, sau đó gửi yêu cầu HTTP DELETE đến API với đường dẫn ‘/api/admin/clos/multiple’ với mảng định danh các CDR.

#### 4.1.7. Quản lý mối quan hệ giữa CLO và PLO (QL-07)

CLO	IT_PLO1	IT_PLO2	IT_PLO3	IT_PLO4	IT_PLO5	IT_PLO6	IT_PLO7	IT_PLO8	IT_PLO9	IT_PLO10
IT33_CLO1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
IT33_CLO2	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
IT33_CLO3	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
IT33_CLO4	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
IT33_CLO5	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
IT33_CLO6	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Hình 4.27. Trang quản lý ánh xạ CLO môn công nghệ phần mềm với PLO

Khi quản trị viên truy cập trang xem ánh xạ giữa CLO và PLO, hệ thống sẽ thực hiện các bước sau. Trước tiên, hệ thống gửi yêu cầu HTTP GET đến API có đường dẫn '/api/admin/clos?subject\_id=\${id}&isDelete=false' để lấy danh sách các CLO chưa bị xóa cho môn học có định danh là \${id}. Sau đó, hệ thống gửi yêu cầu HTTP GET đến API có đường dẫn '/api/admin/plos?isDelete=false' nhằm nhận danh sách các PLO chưa bị xóa. Cuối cùng, hệ thống sẽ gửi yêu cầu HTTP GET đến API có đường dẫn '/api/admin/subject/\${id}?only\_clo\_ids=true' để nhận mảng các định danh CLO liên quan đến môn học. Sau đó, hệ thống gửi tiếp yêu cầu HTTP GET đến API có đường dẫn '/api/admin/plo-clo' với tham số id\_clos chứa danh sách các định danh CLO. API này sẽ trả về thông tin ánh xạ giữa CLO và PLO.

Dựa trên dữ liệu từ các API trên, hệ thống tạo ra một mảng ánh xạ, cung cấp cái nhìn tổng quan về mối quan hệ giữa các chuẩn đầu ra môn học và chương trình, hỗ trợ quản trị viên trong việc quản lý và điều chỉnh các liên kết.

Ngoài chức năng xem ánh xạ, hệ thống còn cho phép quản trị viên thay đổi ánh xạ bằng cách chọn các liên kết ánh xạ cần cập nhật và nhấn nút lưu. Khi thay đổi ánh xạ, hệ thống gửi yêu cầu HTTP DELETE đến API có đường dẫn '/api/admin/plo-clo' để xóa các liên kết hiện tại hoặc gửi yêu cầu HTTP POST đến cùng API để thêm các liên kết mới, giúp duy trì và điều chỉnh các liên kết giữa CLO và PLO một cách linh hoạt và hiệu quả.

#### 4.1.8. Quản lý mối quan hệ giữa CLO và các chương học (QL-08)

Chương	IT33_CLO1	IT33_CLO2	IT33_CLO3	IT33_CLO4	IT33_CLO5	IT33_CLO6
IT33_C01	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
IT33_C02	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
IT33_C03	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
IT33_C04	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
IT33_C05	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
IT33_C06	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
IT33_C07	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Hình 4.28. Trang quản lý ánh xạ chương với CDR môn học

Khi quản trị viên truy cập trang xem ánh xạ giữa CLO và chương, hệ thống sẽ thực hiện các bước sau. Đầu tiên, hệ thống gửi yêu cầu HTTP GET đến API có đường dẫn '/api/admin/clos?subject\_id=\${id}&isDelete=false' để lấy danh sách các chuẩn đầu ra (CLO) chưa bị xóa cho môn học có định danh \${id}. Tiếp theo, hệ thống gửi yêu cầu HTTP GET đến API có đường dẫn '/api/admin/chapters?subject\_id=\${id}&isDelete=false' nhằm nhận danh sách các chương chưa bị xóa cho môn học có định danh \${id}. Cuối cùng, hệ thống sẽ gửi yêu cầu HTTP GET đến API '/api/admin/subject/\${id}?only\_chapter\_ids=true' để nhận mảng các định danh chương liên quan đến môn học. Sau đó, hệ thống gửi tiếp yêu cầu HTTP GET đến API '/api/admin/clo-chapter' với tham số chapter\_ids chứa danh sách các định danh chương. API này sẽ trả về thông tin ánh xạ giữa CLO và các chương.

Dựa trên dữ liệu từ các API này, hệ thống tạo ra một mảng ánh xạ, cung cấp cái nhìn tổng quan về mối quan hệ giữa các chuẩn đầu ra và các chương, hỗ trợ quản trị viên trong việc quản lý và điều chỉnh các liên kết.

Ngoài chức năng xem ánh xạ, hệ thống còn cho phép quản trị viên thay đổi ánh xạ bằng cách chọn các liên kết ánh xạ cần cập nhật và nhấn nút lưu. Khi thay đổi ánh xạ, hệ thống sẽ gửi yêu cầu HTTP DELETE đến API '/api/admin/clo-chapter' để xóa các liên kết hiện tại hoặc gửi yêu cầu HTTP POST đến cùng API để thêm các liên kết mới. Quá trình này giúp duy trì và điều chỉnh các liên kết giữa CLO và các chương trong hệ thống một cách linh hoạt và hiệu quả.

#### 4.1.9. Quản lý các chương học của môn học (QL-09)

Tên Chương	Mô tả	Thao tác
IT33_C01	Tổng quan về công nghệ phần mềm	
IT33_C02	Mô hình phát triển phần mềm Agile	
IT33_C03	Xác định yêu cầu	
IT33_C04	Kiến trúc phần mềm	
IT33_C05	Triển khai phần mềm	

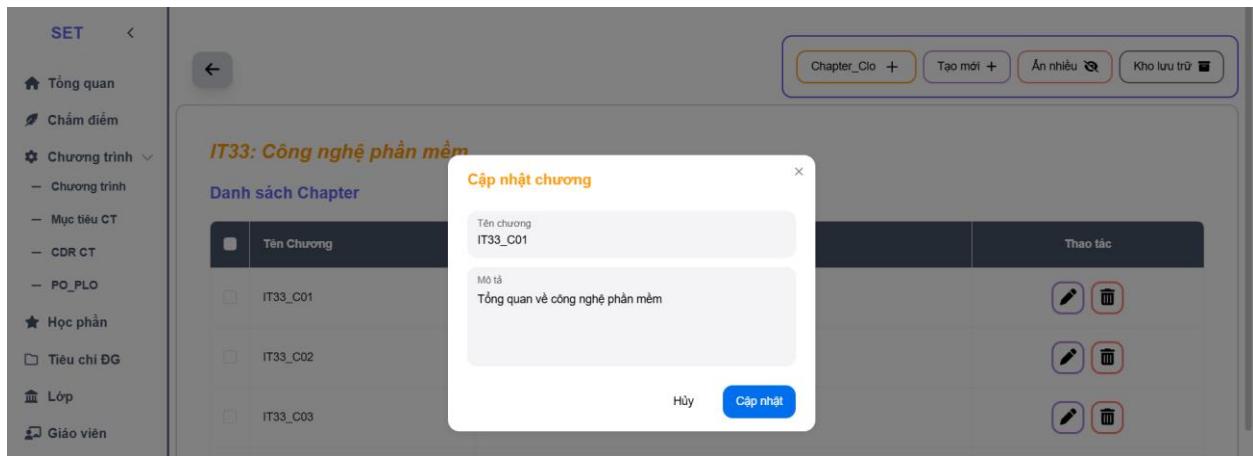
Hình 4.29. Trang quản lý chương của môn học

Khi quản trị viên truy cập trang xem danh sách các chương của môn học, hệ thống sẽ hiển thị toàn bộ danh sách các chương liên quan đến môn học đó. Để lấy danh sách các chương chưa bị xóa (với isDelete là false), quản trị viên gửi yêu cầu HTTP GET đến API tại đường dẫn '/api/admin/chapters?subject\_id=\${id}&isDelete=false', trong đó \${id} là định danh của môn học. API này sẽ trả về danh sách các chương hiện có của môn học.

Để ẩn một hoặc nhiều CDR, quản trị viên chọn CDR và nhấn biểu tượng thùng rác để gửi yêu cầu HTTP PUT đến API '/api/admin/clo/\${\_id}/softDelete', trong đó \${\_id} là định danh của CDR. Để ẩn nhiều CDR cùng lúc, quản trị viên chọn các CDR và nhấn nút “Ẩn nhiều”, hệ thống gửi yêu cầu HTTP PUT đến API '/api/admin/clos/softDelete' với mảng định danh của các CDR. Sau khi hoàn tất, hệ thống sẽ thông báo và cập nhật danh sách CDR trên giao diện.

Hình 4. 30. Chức năng thêm mới chương môn học

Để thêm một chương mới vào môn học, quản trị viên nhấp vào nút “Tạo mới” trên giao diện người dùng. Một cửa sổ nhập liệu sẽ xuất hiện, cho phép quản trị viên nhập các thông tin cần thiết để tạo chương mới. Sau khi hoàn tất việc nhập dữ liệu, quản trị viên nhấp nút “Tạo”. Hệ thống sẽ gửi yêu cầu HTTP POST đến API tại đường dẫn ‘/api/admin/chapter’. Nếu quá trình lưu trữ dữ liệu thành công, hệ thống sẽ thông báo cho quản trị viên về việc thêm chương mới thành công.



Hình 4.31. Chức năng cập nhật chương môn học

Sau khi xem danh sách các chương, quản trị viên có thể chọn chương cần cập nhật bằng cách nhấn vào biểu tượng bút. Cửa sổ chỉnh sửa sẽ xuất hiện, cho phép quản trị viên thực hiện các thay đổi cần thiết đối với thông tin của chương. Sau khi hoàn tất việc chỉnh sửa, quản trị viên nhấp nút “Cập nhật”. Hệ thống sẽ gửi yêu cầu HTTP PUT đến API với đường dẫn ‘/api/admin/chapter/\${chapter\_id}’, trong đó ‘\${chapter\_id}’ là định danh của chương cần cập nhật. Hệ thống sẽ xử lý yêu cầu và cập nhật thông tin của chương tương ứng. Nếu việc cập nhật thành công, hệ thống sẽ thông báo cho quản trị viên về sự thay đổi.

The screenshot shows a software interface with a sidebar on the left containing various menu items such as 'SET', 'Tổng quan', 'Chấm điểm', 'Chương trình' (selected), 'Mục tiêu CT', 'CDR CT', 'PO\_PLO', 'Học phần', 'Tiêu chí ĐG', 'Lớp', 'Giáo viên', 'Sinh viên', 'Lớp môn học', and a user profile for 'Nguyễn Bảo An'. The main area displays a table titled 'Danh sách Chapter đã ẩn' (List of hidden chapters) under the heading 'IT33: Công nghệ phần mềm'. The table has columns for 'Tên Chương' (Chapter Name), 'Mô tả' (Description), and 'Thao tác' (Actions). It contains one row for 'Chương TEST' with the description 'Chương TEST'. The 'Thao tác' column for this row includes a trash can icon and a red square icon. A small blue box with the number '1' is visible at the bottom right of the table.

Hình 4.32. Trang quản lý chương môn học đã ẩn

Khi quản trị viên truy cập kho lưu trữ các chương đã bị xóa, hệ thống gửi yêu cầu HTTP GET đến API '/api/admin/chapters?subject\_id=\${id}&isDelete=true' để lấy danh sách các chương đã bị xóa (isDelete=true) cho môn học có định danh \${id}.

Để khôi phục một chương đã bị ẩn, quản trị viên chọn chương cần khôi phục và gửi yêu cầu HTTP PUT đến API '/api/admin/chapter/\${\_id}/softDelete'. Để khôi phục nhiều chương cùng lúc, gửi yêu cầu đến API '/api/admin/chapters/softDelete' với mảng các định danh của các chương cần khôi phục. Sau khi khôi phục thành công, hệ thống sẽ thông báo và cập nhật danh sách các chương.

Để xóa một chương cụ thể, quản trị viên chọn chương từ danh sách và gửi yêu cầu HTTP DELETE đến API '/api/admin/chapter/\${\_id}'. Để xóa nhiều chương đồng thời, gửi yêu cầu đến API '/api/admin/chapters/multiple' kèm theo dữ liệu là mảng các định danh của các chương cần xóa. Sau khi xóa thành công, hệ thống sẽ thông báo và cập nhật danh sách các chương.

## 4.2. Chức năng giảng viên

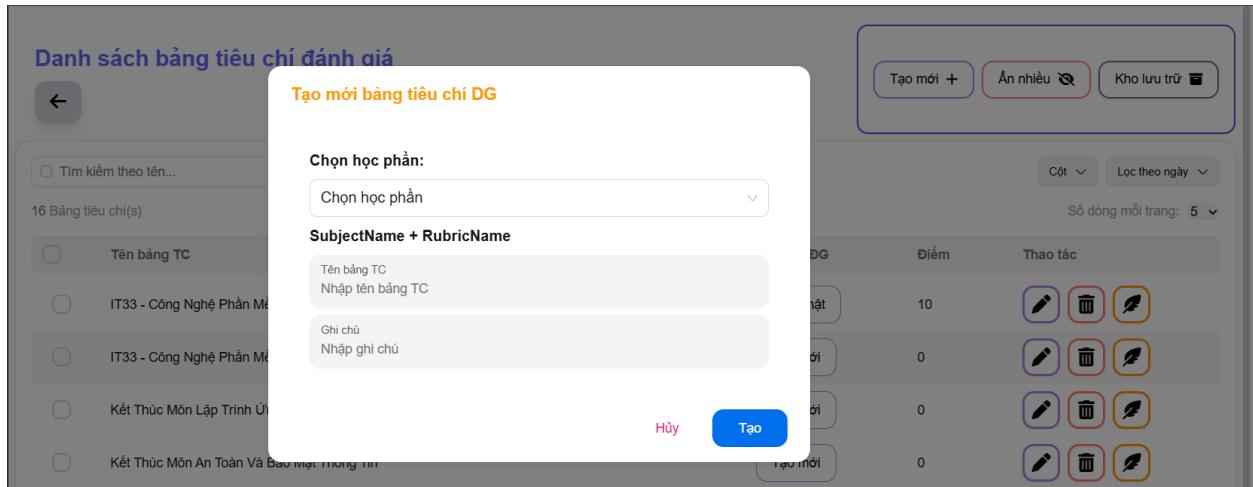
### 4.2.1. Quản lý bảng tiêu chí (QL-10)

The screenshot shows a user interface for managing rubrics. On the left, there is a sidebar with a navigation menu under 'SET'. The 'Tiêu chí DG' (Rubric) option is selected. The main area is titled 'Danh sách bảng tiêu chí đánh giá' (List of evaluation rubrics). It displays a table with 16 rows, each representing a rubric. The columns are 'Tên bảng TC' (Table name), 'Tiêu chí ĐG' (Evaluation criterion), 'Điểm' (Score), and 'Thao tác' (Actions). Each row has a checkbox next to the table name, a 'Cập nhật' (Update) button, a score value, and three icons for edit, delete, and preview. At the bottom of the table, it says '0 trong số 16 mục đã chọn' (0 out of 16 items selected). There are also buttons for 'Tạo mới +' (Create new +), 'Ẩn nhiều' (Hide many), and 'Kho lưu trữ' (Storage庫).

Hình 4.33. Trang quản lý bảng tiêu chí

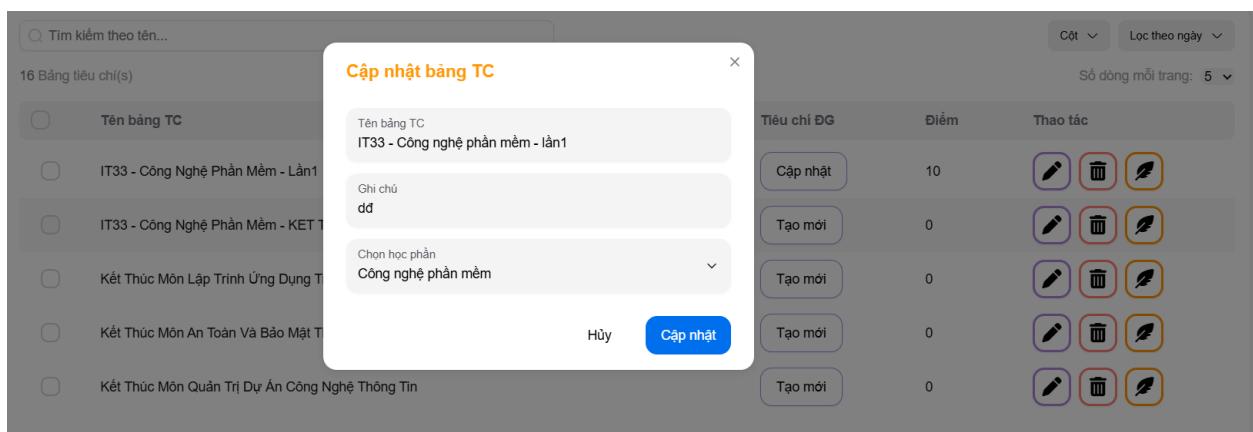
Khi quản trị viên truy cập trang xem danh sách các bảng tiêu chí (rubrics) của chương trình, hệ thống thực hiện các bước sau. Để lấy danh sách các tiêu chí chưa bị xóa, quản trị viên gửi yêu cầu HTTP GET đến API '/api/admin/rubrics/ checkScore? teacher\_id =\${teacher\_id}&isDelete=false', trong đó \${teacher\_id} là định danh của giảng viên. API này trả về danh sách các tiêu chí chưa bị xóa cho giảng viên cụ thể.

Để ẩn một hoặc nhiều bảng tiêu chí, quản trị viên chọn tiêu chí cần ẩn và nhấn biểu tượng thùng rác để gửi yêu cầu HTTP PUT đến API '/api/admin/rubric/\${\_id}/softDelete', trong đó \${\_id} là định danh của tiêu chí. Đối với việc ẩn nhiều bảng tiêu chí cùng lúc, quản trị viên chọn các bảng tiêu chí và nhấn nút “Ẩn nhiều”, hệ thống gửi yêu cầu HTTP PUT đến API '/api/admin/rubrics/softDelete' với mảng định danh của các tiêu chí. Sau khi hoàn tất, hệ thống sẽ thông báo và cập nhật danh sách các bảng tiêu chí trên giao diện.



Hình 4.34. Chức năng thêm mới bảng tiêu chí

Để thêm một bảng tiêu chí mới, quản trị viên nhấn vào nút “Tạo mới” trên giao diện. Một cửa sổ nhập liệu sẽ xuất hiện, cho phép quản trị viên nhập các thông tin cần thiết để tạo bảng tiêu chí mới. Sau khi nhập xong dữ liệu, quản trị viên nhấn nút “Tạo”. Hệ thống sẽ gửi yêu cầu HTTP POST đến API ‘/api/admin/rubric’. Nếu việc thêm bảng tiêu chí thành công, hệ thống sẽ thông báo cho quản trị viên và cập nhật danh sách bảng tiêu chí.



Hình 4.35. Chức năng cập nhật bảng tiêu chí

Để chỉnh sửa thông tin của một bảng tiêu chí cụ thể, quản trị viên lựa chọn bảng tiêu chí cần chỉnh sửa từ danh sách hiện có và nhấn vào biểu tượng bút. Một cửa sổ chỉnh sửa sẽ được mở, cho phép quản trị viên cập nhật thông tin của bảng tiêu chí đó. Sau khi thực hiện các chỉnh sửa cần thiết, quản trị viên nhấn nút “Cập nhật”. Hệ thống sẽ gửi yêu cầu HTTP PUT đến API ‘/api/admin/rubric/\${rubric\_id}’, trong đó \${rubric\_id} là định danh của bảng tiêu chí cần được cập nhật. Nếu quá trình cập nhật thành công, hệ thống sẽ thông báo cho quản trị viên về sự thay đổi và cập nhật danh sách bảng tiêu chí trên giao diện.

Hình 4.36. Trang quản lý bảng tiêu chí đã ẩn

Để truy cập kho lưu trữ các tiêu chí đã bị xóa, quản trị viên nhấn vào nút “Kho lưu trữ” trên giao diện. Hệ thống gửi yêu cầu HTTP GET đến API với đường dẫn ‘/api/admin/rubrics? isDelete=true’, cung cấp danh sách các tiêu chí đã bị xóa.

Để khôi phục một tiêu chí đã bị ẩn, quản trị viên nhấn vào biểu tượng “Quay lại” bên cạnh tiêu chí cần khôi phục. Hệ thống gửi yêu cầu HTTP PUT đến API có đường dẫn ‘/api/admin/rubric/\${\_id}/softDelete’, trong đó \${\_id} là định danh của tiêu chí. Để khôi phục nhiều tiêu chí cùng lúc, quản trị viên chọn các tiêu chí cần khôi phục và nhấn vào biểu tượng “Quay lại”. Hệ thống gửi yêu cầu HTTP PUT đến API với đường dẫn ‘/api/admin/rubrics/softDelete’ với mảng các định danh của các tiêu chí cần khôi phục.

Để xóa một tiêu chí cụ thể, quản trị viên chọn tiêu chí từ danh sách và nhấn vào biểu tượng thùng rác. Hệ thống gửi yêu cầu HTTP DELETE đến API với đường dẫn ‘/api/admin/rubric/\${\_id}’, trong đó \${\_id} là định danh của tiêu chí. Để xóa nhiều tiêu chí đồng thời, quản trị viên chọn các tiêu chí từ danh sách và nhấn vào biểu tượng thùng rác. Hệ thống gửi yêu cầu HTTP DELETE đến API ‘/api/admin/rubrics/multiple’, kèm theo dữ liệu là mảng các định danh của các tiêu chí cần xóa.

#### 4.2.2. Quản lý các tiêu chí của bảng tiêu chí (QL-11)

CDR HP	CDR CT	Chương	Mô tả TC	Điểm TC	Thao tác
IT33_CLO1 Trình Bay Tổng Quan Về Công Nghệ Phần Mềm	IT_PLO2 Kết Hợp Kiến Thức Chuyên Môn Để Giải Quyết Các Vấn Đề Liên Quan Chuyên Ngành.	IT33_C01 Tổng Quan Về Công Nghệ Phần Mềm	<b>Phản Mô Đầu Hợp Lý:</b> Đặc Tính Như Cầu Ứng Dụng Phù Hợp Nêu Được Tính Cản Thiết Của Phần Mềm Phân Tán/Microservices/Cloud	1.5	<span style="color: purple;">Edit</span> <span style="color: red;">Delete</span>
IT33_CLO3 Vận Dụng Mô Hình Đã Học Để Xây Dựng Phần Mềm	IT_PLO2 Kết Hợp Kiến Thức Chuyên Môn Để Giải Quyết Các Vấn Đề Liên Quan Chuyên Ngành.	IT33_C07 DevOps Và Quản Lý Mã Nguồn	<b>Kế Hoạch Phát Triển Phần Mềm:</b> Product Backlog Phân Chia Sprint Sprint Backlog Phân Công Thành Phần Va Kế Hoạch Burndown Chart Hợp Lý	2	<span style="color: purple;">Edit</span> <span style="color: red;">Delete</span>
IT33_CLO4 Vận Dụng Cách Thức	IT_PLO7 Phối Hợp Tốt Các Kỹ	IT33_C04	<b>Thiết Kế Kiến Trúc Phần Mềm</b> Mô Tả Các Năng Tính Phần Trong Kiến Trúc Của Phần Mềm RESTful API	2	<span style="color: purple;">Edit</span> <span style="color: red;">Delete</span>

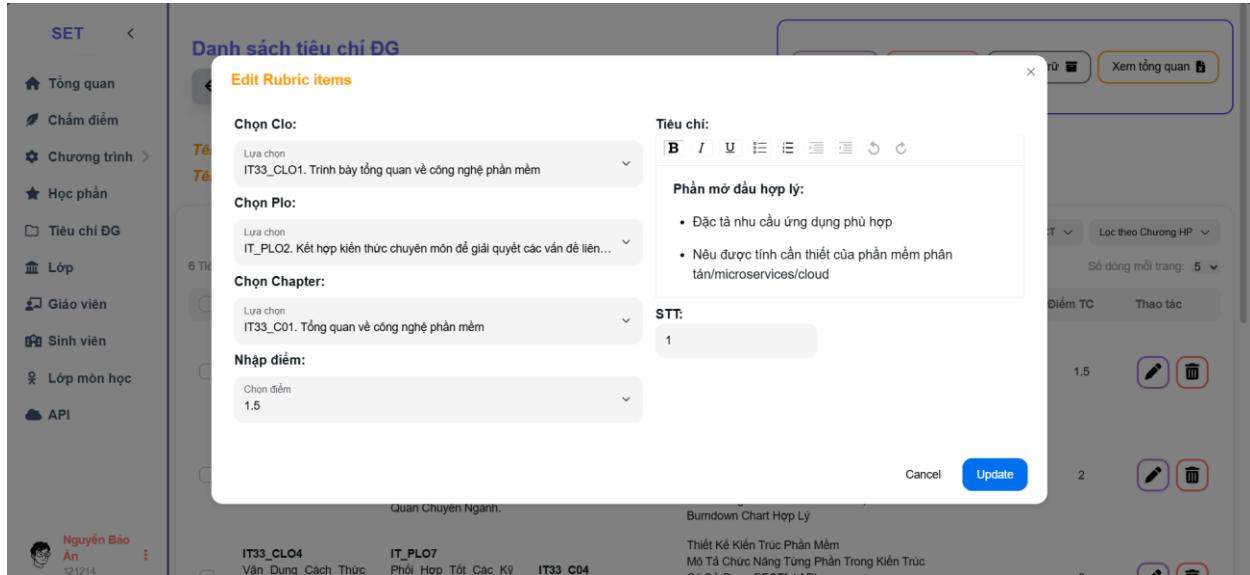
Hình 4.37. Trang quản lý tiêu chí của bảng tiêu chí.

Đầu tiên, để lấy danh sách các tiêu chí, quản trị viên gửi yêu cầu HTTP GET đến API có đường dẫn ‘/api/admin/rubrics/checkScore?teacher\_id = \${teacher\_id} & isDelete = false’. Tham số \${teacher\_id} là định danh của giảng viên và API này trả về danh sách các tiêu chí chưa bị xóa liên quan đến giảng viên cụ thể.

Để ẩn một hoặc nhiều tiêu chí, quản trị viên chọn tiêu chí từ danh sách và nhấn biểu tượng thùng rác để gửi yêu cầu HTTP PUT đến API ‘/api/admin/rubric/\${\_id}/softDelete’ cho từng tiêu chí. Đối với nhiều tiêu chí, chọn các tiêu chí và nhấn nút “Ẩn nhiều”, hệ thống sẽ gửi yêu cầu HTTP PUT đến API ‘/api/admin/rubrics/softDelete’ với mảng định danh của các tiêu chí. Sau khi hoàn tất, hệ thống sẽ thông báo và cập nhật danh sách tiêu chí trên giao diện.

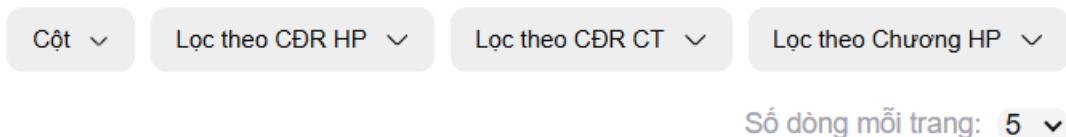
Hình 4.38. Chức năng tạo mới tiêu chí.

Để thêm một tiêu chí mới, quản trị viên nhấn vào nút “Tạo mới” trên giao diện. Một cửa sổ nhập liệu sẽ xuất hiện, cho phép nhập các thông tin cần thiết để tạo tiêu chí. Sau khi hoàn tất việc nhập dữ liệu, quản trị viên nhấn nút “Tạo”. Hệ thống sẽ gửi yêu cầu HTTP POST đến API ‘/api/admin/rubric’. Nếu việc lưu trữ dữ liệu thành công, hệ thống sẽ thông báo cho quản trị viên về việc thêm tiêu chí mới.



Hình 4.39. Chức năng cập nhật tiêu chí

Để cập nhật thông tin của một tiêu chí, quản trị viên chọn tiêu chí cần chỉnh sửa bằng cách nhấn vào biểu tượng bút. Cửa sổ chỉnh sửa sẽ hiện lên, cho phép quản trị viên thực hiện các thay đổi cần thiết. Sau khi hoàn tất việc chỉnh sửa, quản trị viên nhấn nút “Cập nhật”. Hệ thống sẽ gửi yêu cầu HTTP PUT đến API ‘/api/admin/rubric/\${rubric\_id}’, trong đó \${rubric\_id} là định danh của tiêu chí cần cập nhật. Nếu việc cập nhật thành công, hệ thống sẽ thông báo cho quản trị viên và cập nhật thông tin của tiêu chí tương ứng.



Hình 4.40. Các chức năng lọc của trang tiêu chí

	Tên CLO	Tên PLO	Tên Chapter	Điểm	Form
<input type="checkbox"/>	IT33_CLO5	IT_PLO6	IT33_C07		

Hình 4.41. Trang quản lý tiêu chí đã ẩn

Để truy cập kho lưu trữ các tiêu chí đã bị xóa, quản trị viên nhấn vào nút “Kho lưu trữ” trên giao diện. Hệ thống gửi yêu cầu HTTP GET đến API ‘/api/admin/rubrics?isDelete = true’ để lấy danh sách các tiêu chí đã bị xóa.

Để khôi phục một tiêu chí đã bị ẩn, quản trị viên nhấn vào biểu tượng “Quay lại” bên cạnh tiêu chí cần khôi phục. Hệ thống gửi yêu cầu HTTP PUT đến API có đường dẫn ‘/api/admin/rubric/\${\_id}/softDelete’, trong đó \${\_id} là định danh của tiêu chí. Để khôi phục nhiều tiêu chí cùng lúc, quản trị viên chọn các tiêu chí cần khôi phục và nhấn vào biểu tượng “Quay lại”. Hệ thống gửi yêu cầu HTTP PUT đến API với đường dẫn ‘/api/admin/rubrics/softDelete’, kèm theo mảng các định danh của các tiêu chí cần khôi phục.

Để xóa một tiêu chí cụ thể, quản trị viên chọn tiêu chí từ danh sách và nhấn vào biểu tượng thùng rác. Hệ thống gửi yêu cầu HTTP DELETE đến API có đường dẫn ‘/api/admin/rubric/\${\_id}’, trong đó \${\_id} là định danh của tiêu chí. Để xóa nhiều tiêu chí đồng thời, quản trị viên chọn các tiêu chí từ danh sách và nhấn vào biểu tượng thùng rác. Hệ thống gửi yêu cầu HTTP DELETE đến API ‘/api/admin/rubrics/multiple’, kèm theo dữ liệu là mảng các định danh của các tiêu chí cần xóa.

CLO	PLO	Tiêu chí	Tổng điểm
IT33_CLO1	IT_PLO2	<b>Phản mở đầu hợp lý:</b> Đặc tả nhu cầu ứng dụng phù hợp Nêu được tính cần thiết của phần mềm phân tán/microservices/cloud	1.5
IT33_CLO3	IT_PLO2	<b>Kế hoạch phát triển phần mềm:</b> Product backlog Phân chia sprint Sprint backlog Phân công thành phần và kế hoạch Burndown chart hợp lý	2
IT33_CLO4	IT_PLO7	<b>Thiết kế kiến trúc phần mềm</b> Mô tả chức năng từng phần trong kiến trúc Có sử dụng RESTful API Sử dụng công nghệ frontend phù hợp <b>Có sử dụng microservices/Phần mềm dựa trên đám mây</b>	2
IT33_CLO4	IT_PLO8	Sử dụng Git/GitHub Sử dụng Docker Có triển khai lên host	2

Hình 4.42. Trang xem tổng quan tiêu chí

#### 4.2.3. Quản lý đánh giá tổng hợp (QL-12)

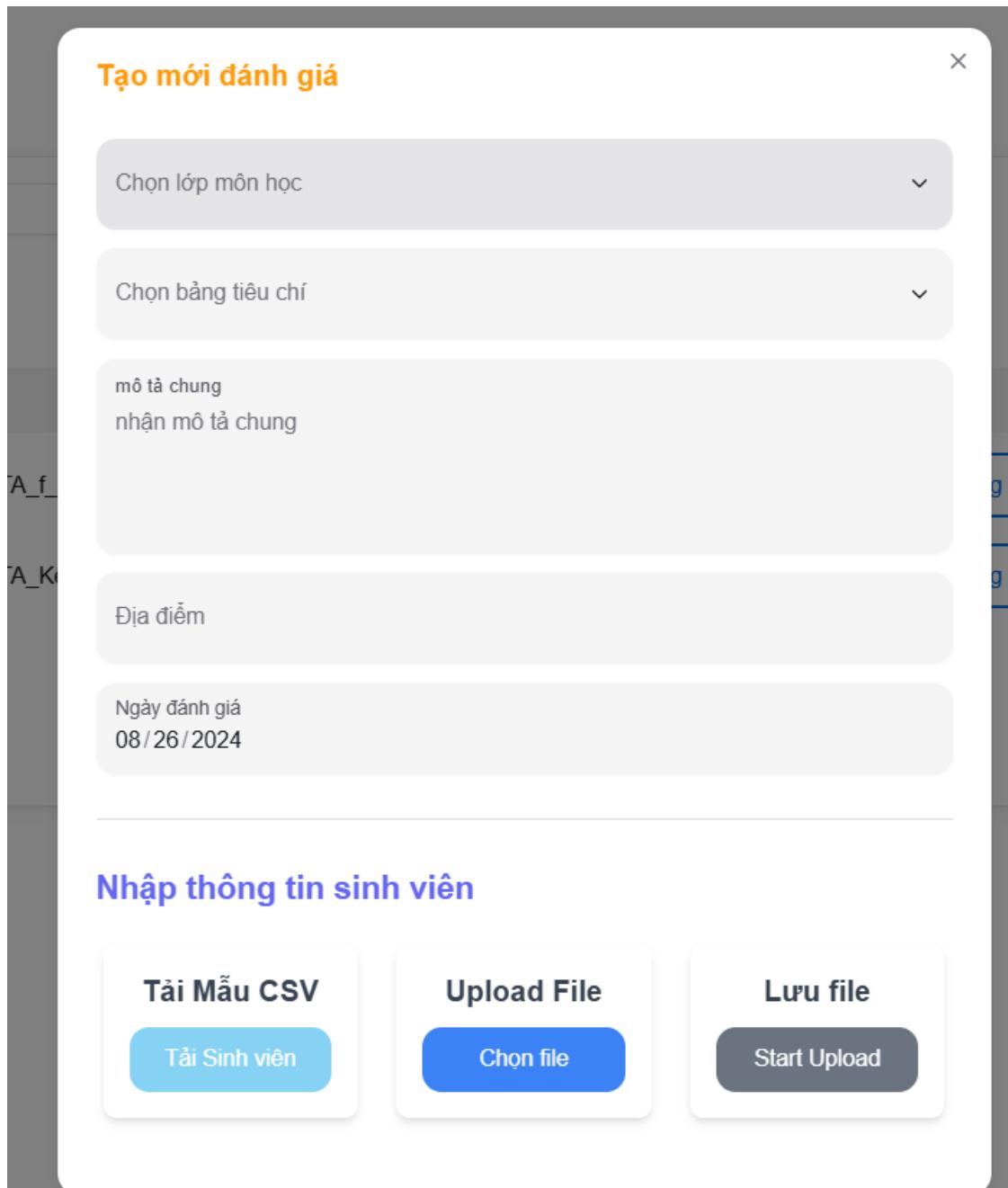
Mô tả chung	Lớp môn học	Phản công	% SV Điểm khác 0	Thao tác
100004 - IT33_Công Nghệ Phần Mềm DA21TTA_f_2024-8-25	Công Nghệ Phần Mềm DA21TTA	Phản công	0%	
100004 - IT33_Công Nghệ Phần Mềm DA21TTA_Kết Thúc Môn_2024-8-24	Công Nghệ Phần Mềm DA21TTA	Phản công	8%	

Hình 4.43. Trang quản lý đánh giá

Khi giảng viên truy cập vào trang quản lý đánh giá, hệ thống sẽ gửi yêu cầu HTTP GET tới API '/api/admin/meta-assessments?teacher\_id=\${teacher\_id}&isDelete=false', trong đó \${teacher\_id} đại diện cho mã định danh của giảng viên. API này sẽ phản hồi với danh sách các đánh giá do giảng viên đó tạo ra. Tại giao diện quản lý, giảng viên có thể xem thông tin chi tiết về từng đánh giá, bao gồm tỷ lệ sinh viên đã được chấm điểm và tình trạng phân công nhiệm vụ chấm điểm.

Để quản lý việc phân công chấm điểm cho từng đánh giá, giảng viên thực hiện thao tác nhấp chuột vào phần phân công tương ứng với mỗi đánh giá.

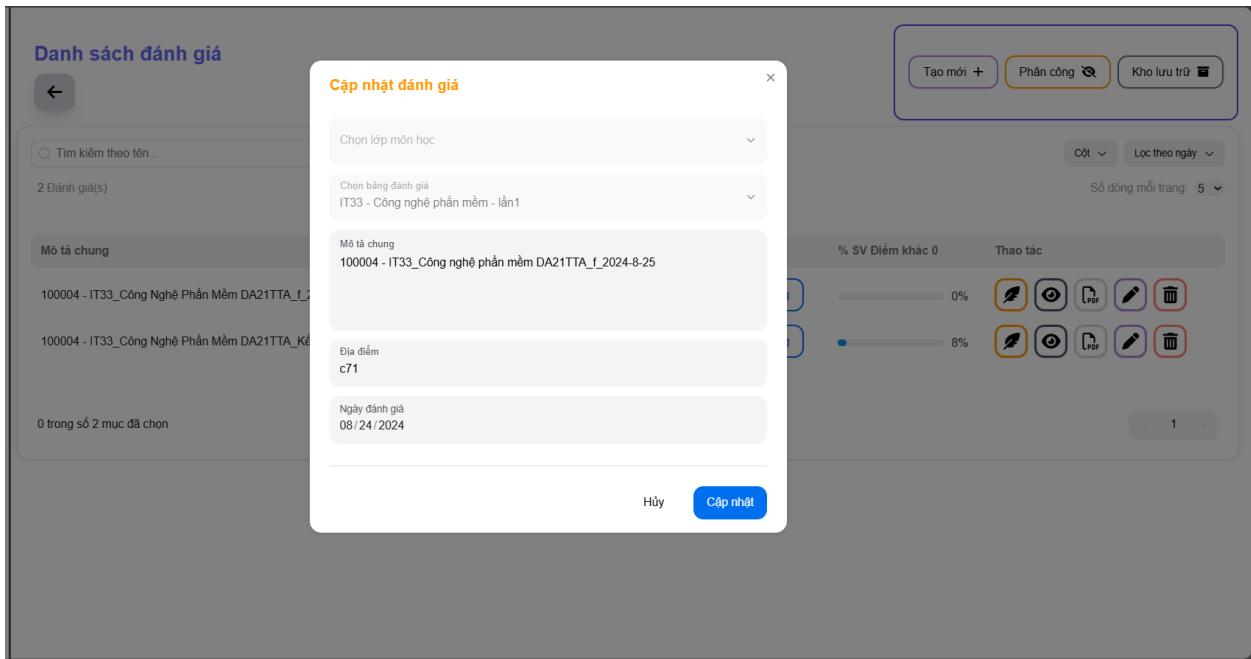
Để ẩn một đánh giá dựa trên mô tả chung, giảng viên chỉ cần nhấn vào biểu tượng thùng rác. Hệ thống sẽ gửi yêu cầu HTTP PATCH tới API '/api/admin/meta-assessments/softDeleteByGeneralDescription'. Dữ liệu yêu cầu sẽ bao gồm mô tả chung của đánh giá và thuộc tính isDelete được đặt thành true, nhằm thực hiện việc ẩn đánh giá tương ứng.



Hình 4.44. Chức năng thêm mới đánh giá

Để thêm một đánh giá mới, quản trị viên nhấn vào nút “Tạo mới” trên giao diện. Một cửa sổ nhập liệu sẽ hiện ra, cho phép quản trị viên chọn lớp môn học, bảng tiêu chí tương ứng và nhập các thông tin cần thiết. Để thêm sinh viên, quản trị viên nhấn vào nút tải sinh viên. Hệ thống sẽ gửi yêu cầu POST tới API có đường dẫn ‘api/admin/course-

enrollment /templates/data' để nhận về tệp Excel chứa danh sách sinh viên của lớp môn học đã chọn. Sau khi tải tệp lên, quản trị viên nhấn vào nút “Start Upload” để hoàn tất việc tạo đánh giá cho lớp học. Cuối cùng, hệ thống sẽ gửi yêu cầu POST tới API tại đường dẫn ‘/api/admin/importExcel/meta-assessment’ để thực hiện quá trình nhập liệu từ file Excel chứa thông tin liên quan đến đánh giá meta.



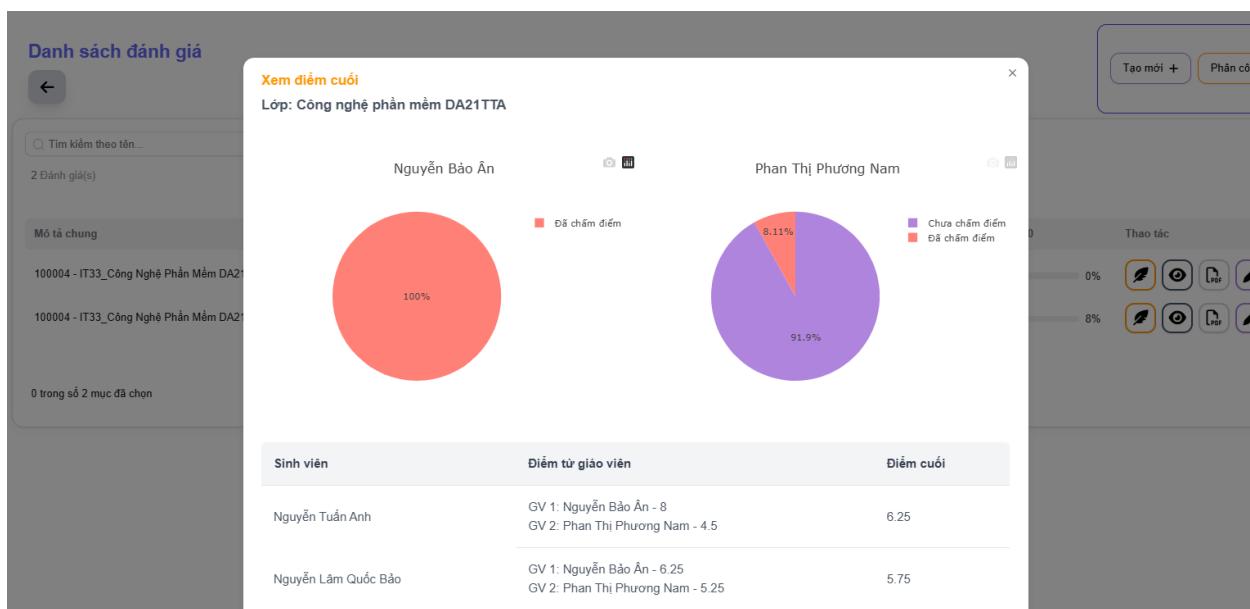
Hình 4.45. Chức năng cập nhật đánh giá

Sau khi xem danh sách các đánh giá, quản trị viên có thể chọn đánh giá cần cập nhật bằng cách nhấn vào biểu tượng bút. Một cửa sổ chỉnh sửa sẽ xuất hiện, cho phép quản trị viên thực hiện các thay đổi cần thiết đối với thông tin của đánh giá. Sau khi hoàn tất việc chỉnh sửa, quản trị viên nhấn nút “Cập nhật”. Hệ thống sẽ gửi yêu cầu HTTP PATCH tới API tại đường dẫn ‘api/admin/meta-assessments/updateByGeneralDescription’, với dữ liệu bao gồm mô tả cũ (GeneralDescription) và dữ liệu cập nhật (updateData). Yêu cầu này sẽ được xử lý để cập nhật thông tin của đánh giá tương ứng. Nếu việc cập nhật thành công, hệ thống sẽ thông báo cho quản trị viên về sự thay đổi.



Hình 4.46. PDF mẫu đánh giá tại lớp

Để tải mẫu phiếu chấm điểm, giảng viên chọn biểu tượng PDF tại đánh giá cụ thể. Hệ thống sẽ mở cửa sổ cho phép chọn giữa định dạng PDF ngang hoặc dọc. Sau khi chọn và xác nhận, hệ thống gửi yêu cầu HTTP POST tới API tại đường dẫn '/api/admin/pdf', với dữ liệu bao gồm chuỗi HTML của mẫu chấm điểm và thông tin chiều PDF (landscape). Yêu cầu được thực hiện với responseType là blob và withCredentials là true. Hệ thống cũng cung cấp chế độ zoom để giảng viên dễ dàng quan sát nội dung mẫu PDF.



Hình 4.47. Chức năng xem điểm cuối

Để xem điểm cuối giữa các giảng viên được phân công chấm, giảng viên chọn biểu tượng con mắt tại đánh giá cụ thể. Hệ thống sẽ mở một cửa sổ, cho phép giảng viên quan

sát điểm số của các giảng viên đã chấm.

The screenshot shows a user interface for managing assessments. On the left, there's a sidebar with navigation links like 'SET', 'Tổng quan', 'Chấm điểm', 'Chương trình', etc. The main area is titled 'Danh sách đánh giá đã ẩn' (List of hidden assessments). It displays a table with three columns: 'Mô tả chung' (General description), 'Mã lớp' (Class code), and 'Thao tác' (Actions). A single row is listed: '100004 - IT33\_Công nghệ phần mềm DA21TTA\_f\_2024-8-25' under 'Mô tả chung', 'Công nghệ phần mềm DA21TTA' under 'Mã lớp', and edit and delete icons under 'Thao tác'. At the bottom right of the table, there are navigation arrows (< >) and a refresh icon.

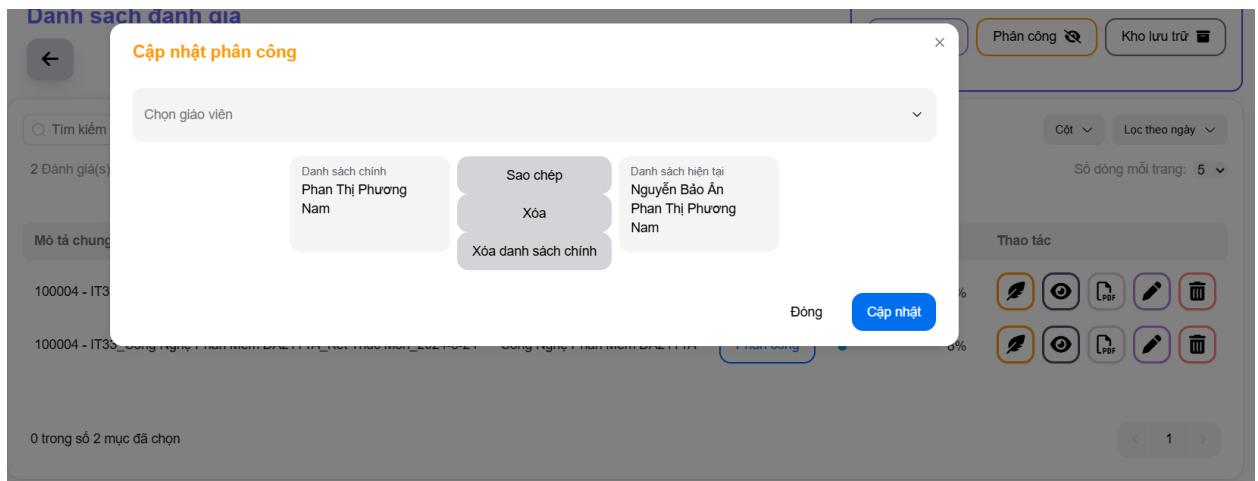
Hình 4.48. Trang quản lý đánh giá đã ẩn

Khi quản trị viên muốn xem danh sách các đánh giá đã bị xóa, hệ thống gửi yêu cầu HTTP GET đến API ‘api/admin/meta-assessments? teacher\_id = \${teacher\_id} & isDelete = true’, trong đó \${teacher\_id} là mã giảng viên. API này sẽ trả về danh sách các đánh giá đã bị xóa.

Để xóa một đánh giá cụ thể, quản trị viên chọn đánh giá từ danh sách và nhấn vào biểu tượng thùng rác. Hệ thống gửi yêu cầu HTTP DELETE đến API có đường dẫn ‘api/admin/meta-assessments/deleteByGeneralDescription’, với dữ liệu chứa thông tin cần xóa. Yêu cầu này sẽ xóa đánh giá khỏi hệ thống và cập nhật danh sách.

Khi muốn ẩn một đánh giá mà không xóa vĩnh viễn, quản trị viên chọn đánh giá cần ẩn và nhấn vào biểu tượng thùng rác. Hệ thống gửi yêu cầu HTTP PATCH đến API ‘api/admin/meta-assessments/softDeleteByGeneralDescription’, với dữ liệu bao gồm mô tả chung và thuộc tính isDelete được đặt là true. Yêu cầu này sẽ cập nhật trạng thái đánh giá thành ẩn và không hiển thị trong danh sách chính.

#### 4.2.4. Mời giảng viên tham gia đánh giá (QL-13)



Hình 4.49. Chức năng quản lý phân công

Khi chọn các giảng viên tham gia đánh giá, giảng viên nhấn vào nút “Sao chép” để chuyển tên giảng viên từ danh sách chính vào danh sách hiện tại.

Để tạo một đánh giá mới, giảng viên nhấn vào nút “Tạo mới”, và hệ thống sẽ gửi yêu cầu HTTP POST đến API ‘api/admin/assessment’ để lưu thông tin đánh giá mới.

Để cập nhật một đánh giá hiện có, giảng viên nhấn vào nút “Cập nhật”. Hệ thống sẽ so sánh danh sách giảng viên hiện tại với danh sách cũ. Nếu cần loại bỏ giảng viên nào, hệ thống gửi yêu cầu HTTP DELETE đến API ‘api/admin/assessment/teacher/\${teacherId}’ với teacherId là định danh của giảng viên cần xóa. Sau đó, hệ thống gửi yêu cầu HTTP POST đến API ‘api/admin/assessment’ để cập nhật thông tin đánh giá với danh sách giảng viên mới.



Hình 4.50. Trang nhận kết quả phân công

Để quan sát tình trạng mời đánh giá, giảng viên nhấn vào nút “Phân công” phía trên danh sách đánh giá. Hệ thống sẽ mở cửa sổ cho phép giảng viên xem danh sách các giảng viên được mời và người đã mời họ tham gia đánh giá. Để tiến hành đánh giá, giảng viên

có thể nhấn vào tên của người mòi hoặc vào nút liên kết để được chuyển đến trang đánh giá tương ứng.

#### 4.2.5. Quản lý quy trình chấm điểm và đánh giá (QL-14)

Tên đề tài	Lớp	SV	Điểm	Thao tác
Đề Tài 1	DA21TTA	110121002 Nguyễn Tuấn Anh	10	
Đề Tài 1	DA21TTA	110121007 Nguyễn Lâm Quốc Bảo	9.75	
Đề Tài 1	DA21TTA	110121010 Cao Nguyễn Hải Đăng	9.5	
	DA21TTA	110121017 Đinh Lê Bảo Duy	0	
	DA21TTA	110121024 Trần Bá Hiếu	0	

Hình 4.51. Trang quản lý đánh giá điểm cho sinh viên

Khi giảng viên truy cập trang chấm điểm, hệ thống sẽ gửi yêu cầu HTTP GET đến API ‘api/admin/assessment? generalDescription = {descriptionURL} & isDelete = false & teacher\_id = \${teacher\_id}’, trong đó \${descriptionURL} là tham số trên URL. Hệ thống sẽ xử lý yêu cầu và hiển thị danh sách sinh viên có trong đánh giá. Tại đây, giảng viên có thể quan sát thông tin của sinh viên, bao gồm lớp, điểm số và đề tài của từng sinh viên.

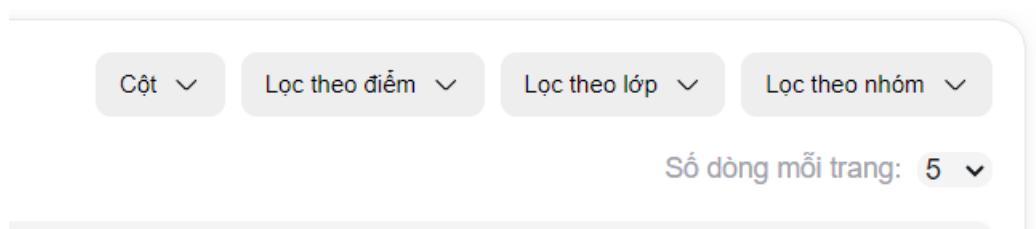
Để xóa sinh viên khỏi đề tài, giảng viên nhấn vào biểu tượng thùng rác. Hệ thống sẽ gửi yêu cầu HTTP DELETE đến API ‘/api/admin /meta-assessment/\${\_id}’, trong đó \${\_id} là định danh của sinh viên hoặc đề tài cần xóa.

Hình 4.52. Cửa sổ cập nhật đề tài bằng Excel

	A	B
1	Tên SV	Tên đề tài
2	Nguyễn Tuấn Anh	Đề tài 1
3	Nguyễn Lâm Quốc Bảo	Đề tài 1
4	Cao Nguyễn Hải Đăng	Đề tài 1
5	Dinh Lê Bảo Duy	Đề tài 2
6	Trần Bá Hiếu	Đề tài 2
7	Trương Hoàng Hưng	Đề tài 2
8	Lâm Quốc Huy	Đề tài 3
9	Huỳnh Nhựt Huy	Đề tài 3
10	Thượng Văn Anh Khoa	Đề tài 3
11	Nghị Tuấn Lộc	Đề tài 4
12	Nguyễn Thanh Lý	Đề tài 4
13	Trầm Ngọc Mai	Đề tài 4
14	Lâm Ngọc Triệu	Đề tài 5
15	Nguyễn Thiên Nhân	Đề tài 5
16	Cao Duy Nhân	Đề tài 5
17	Nguyễn Hoàng Nhân	Đề tài 6
18	Lâm Vĩnh Phát	Đề tài 6

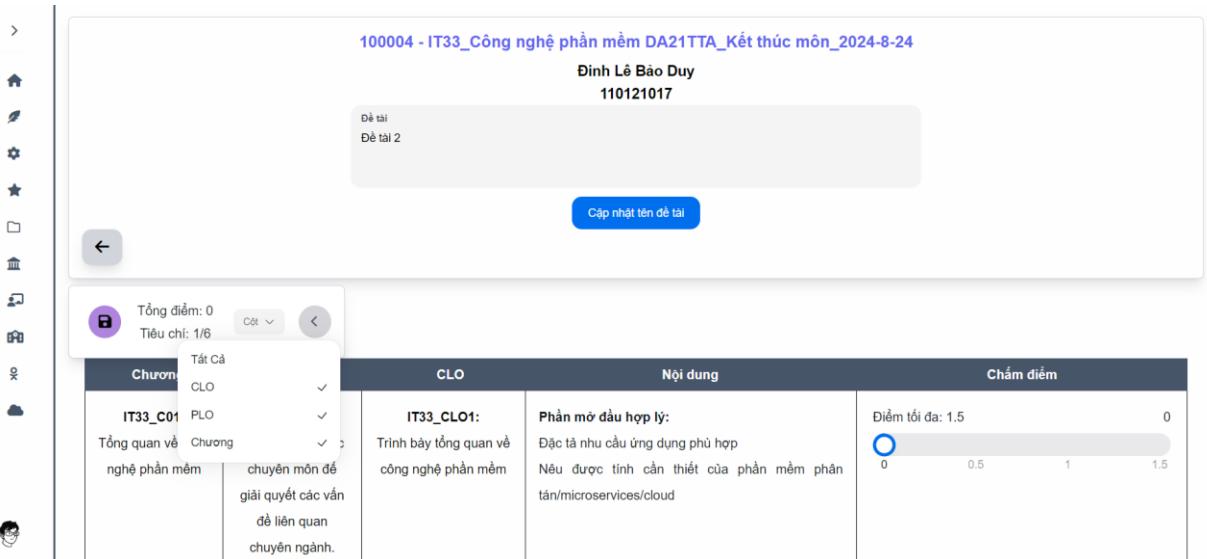
Hình 4.53. Nội dung bên trong mẫu cập nhật

Đối với giảng viên giảng dạy môn này, để cập nhật đề tài cho sinh viên, họ cần nhấn vào nút “Cập nhật đề tài”. Hệ thống sẽ mở cửa sổ tải mẫu cập nhật, nơi giảng viên có thể nhấn vào nút “Tải mẫu” để yêu cầu mẫu đề tài từ hệ thống. Sau khi nhận được mẫu, giảng viên chỉnh sửa đề tài và gửi file cập nhật trở lại hệ thống. Cuối cùng, khi nhấn vào nút “Bắt đầu tải lên”, hệ thống sẽ gửi yêu cầu PUT đến API tại đường dẫn ‘/api/admin/importExcel/meta-assessment/updateDescription’ để cập nhật thông tin đề tài dựa trên file đã gửi.



Hình 4.54. Các chức năng lọc của trang đánh giá điểm cho sinh viên

Chức năng lọc danh sách sinh viên đánh giá của hệ thống được thiết kế nhằm hỗ trợ giảng viên trong việc quản lý dữ liệu hiệu quả hơn. Hệ thống cung cấp các tùy chọn lọc linh hoạt, bao gồm khả năng lọc theo điểm (để phân loại sinh viên chưa được chấm điểm), theo lớp học, và theo nhóm đề tài. Các tùy chọn này giúp quản trị viên dễ dàng truy xuất và phân tích thông tin sinh viên theo các tiêu chí cụ thể.



Hình 4.55. Trang đánh giá đơn cho sinh viên

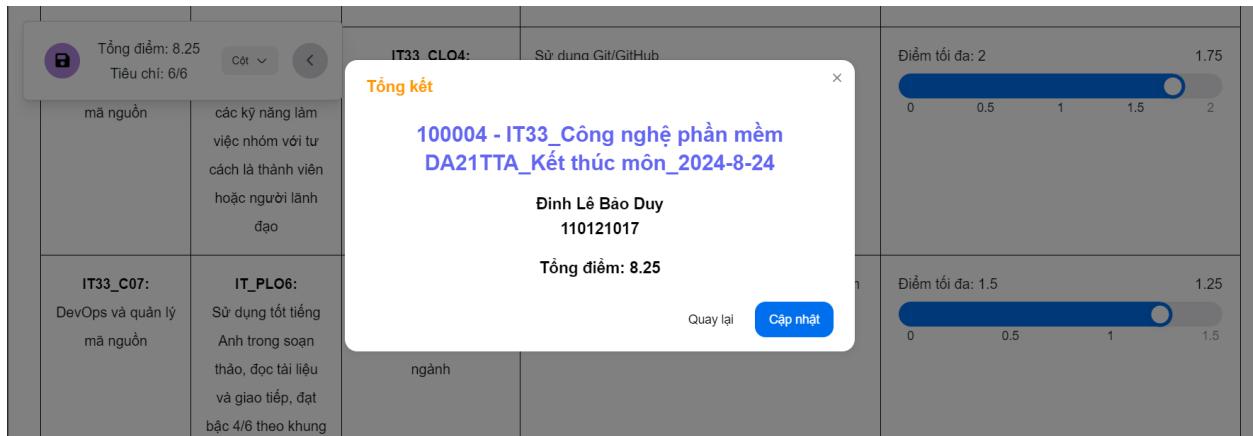
Để thực hiện chức năng chấm điểm cá nhân, giảng viên chọn biểu tượng cây bút tại sinh viên cần chấm. Sau khi nhấn vào, hệ thống sẽ chuyển giảng viên đến giao diện chấm điểm. Tại giao diện này, giảng viên chấm chính sẽ có quyền cập nhật đề tài của sinh viên. Để hỗ trợ việc chấm điểm một cách trực quan, hệ thống sẽ gửi yêu cầu đến API có đường dẫn ‘/api/admin/rubric/\${rubric\_id}/items?isDelete false’ để lấy dữ liệu và hiển thị các tiêu chí, mã chương, CDR Môn học, và CDR Chương trình tương ứng.

	Tổng điểm: 8.25 Tiêu chí: 6/6	<b>IT33_CLO4:</b> Vận dụng cách thức tổ chức và hoạt động nhóm hiệu quả	Sử dụng Git/GitHub Sử dụng Docker Có triển khai lên host	Điểm tối đa: 2 <div style="width: 100%;">1.75</div>
	<b>IT33_C07:</b> DevOps và quản lý mã nguồn	<b>IT_PLO6:</b> Sử dụng tốt tiếng Anh trong soạn thảo, đọc tài liệu và giao tiếp, đạt bậc 4/6 theo khung năng lực ngoại ngữ của Việt Nam.	<b>IT33_CLO5:</b> Sử dụng tiếng Anh để đọc tài liệu chuyên ngành	Điểm tối đa: 1.5 <div style="width: 100%;">1.25</div>
	<b>IT33_C07:</b> DevOps và quản lý mã nguồn	<b>IT_PLO9:</b> Coi trọng trách nhiệm nghề nghiệp	<b>IT33_CLO6:</b> Thể hiện đạo đức nghề nghiệp, tinh thần trách nhiệm	Điểm tối đa: 1 <div style="width: 75%;">0.75</div>

Hình 4.56. Các tiêu đã được giảng viên đánh giá

Tại giao diện chấm điểm, giảng viên sẽ nhìn thấy các mã chương, CDR Chương trình và CDR Môn học tương ứng. Để chấm điểm cho từng tiêu chí, giảng viên chỉ cần kéo thanh điểm đến mức mong muốn.

Khi hoàn thành đánh giá cho sinh viên, giảng viên nhấn vào biểu tượng lưu. Hệ thống sẽ gửi yêu cầu đến API PATCH đến ‘api/admin/meta-assessment/\${meta\_assessment\_id}’ để cập nhật điểm. Sau khi cập nhật thành công, hệ thống sẽ lưu lại các tiêu chí tương ứng cùng với số điểm bằng cách gửi yêu cầu POST đến /assessment-item.



Hình 4.57. Hiển thị kết quả đánh giá đơn

Sau khi lưu thành công, cửa sổ tổng kết sẽ xuất hiện, hiển thị điểm của sinh viên ngay lập tức. Nếu phát hiện sai sót, giảng viên có thể nhấn vào nút cập nhật để chỉnh sửa điểm số ngay trên giao diện này.

PHIẾU CHẤM ĐÁNH GIÁ BÁO CÁO  
HỌC PHẦN CÔNG NGHỆ PHẦN MỀM

(Mã HP: IT33)

Chủ đề:

1

SV: Nguyễn Tuấn Anh  
MSSV: 110121002

CLO	Nội dung báo cáo	Điểm	Điểm SV
IT33_CLO1: Trình bày tổng quan về công nghệ phần mềm	Phản mờ đầu hợp lý: Đặc tả nhu cầu ứng dụng phù hợp Nêu được tính cần thiết của phần mềm phân tán/microservices/cloud	1.5	1
IT33_CLO3: Vận dụng mô hình đã học để xây dựng phần mềm	Kế hoạch phát triển phần mềm: Product backlog Phân chia sprint Sprint backlog Phân công thành phần và kế hoạch Burndown chart hợp lý	2	1.5
IT33_CLO4: Vận dụng cách thức tổ chức và hoạt động nhóm hiệu quả	Thiết kế kiến trúc phần mềm Mô tả chức năng từng phần trong kiến trúc Có sử dụng RESTful API Sử dụng công nghệ frontend phù hợp Có sử dụng microservices/Phần mềm dựa trên đám mây	2	1.75
IT33_CLO4: Vận dụng cách thức tổ chức và hoạt động nhóm hiệu quả	Sử dụng Git/GitHub Sử dụng Docker Có triển khai lên host	2	1.75
IT33_CLO5: Sử dụng tiếng Anh để đọc tài liệu chuyên ngành	Có khía cạnh về bảo mật và an toàn của phần mềm	1.5	1.25
IT33_CLO6: Thể hiện đạo đức nghề nghiệp, tinh thần trách nhiệm và tác phong chuyên nghiệp trong công việc	Kỹ năng thuyết trình	1	0.75
	Tổng điểm:	8	

Nguyễn Tuấn Anh

Trà Vinh, ngày ... tháng ... năm ...  
GV CHẤM BÁO CÁO

8

Nguyễn Bảo Ân

Hình 4.58. In PDF điểm sinh viên

Tại giao diện quản lý, khi giảng viên muốn xuất PDF kết quả điểm đã đánh giá, họ chỉ cần nhấn vào biểu tượng PDF bên cạnh đối tượng mà họ mong muốn. Hệ thống sẽ xử lý yêu cầu và tạo ra một tệp PDF chứa mẫu chấm điểm kèm theo điểm số mà giảng viên đã chấm. PDF này sẽ được trả về để giảng viên có thể xem và lưu trữ.

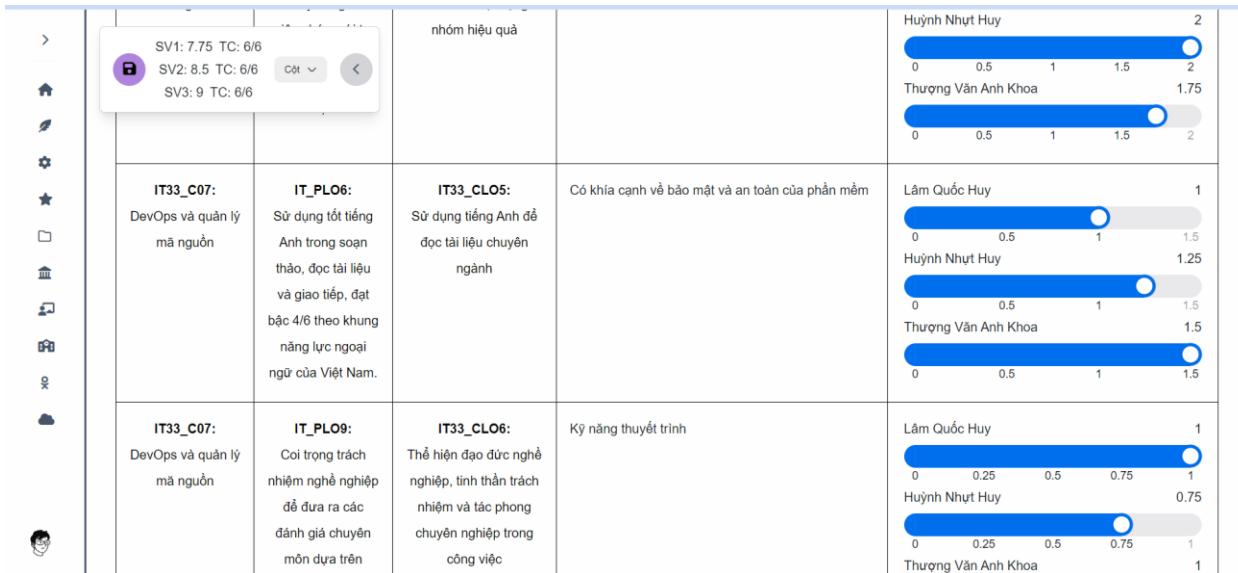
Danh sách sinh viên cần đánh giá					
<input type="button" value="Chấm theo nhóm"/> <input type="button" value="Tạo mới +"/> <input type="button" value="Cập nhật đề tài"/> <input type="button" value="PDF nhóm"/>					
<input type="text" value="Tim kiếm theo tên..."/> <input type="button" value="Cột"/> <input type="button" value="Lọc theo điểm"/> <input type="button" value="Lọc theo lớp"/> <input type="button" value="Lọc theo nhóm"/> <input type="button" value="Số dòng mỗi trang: 5"/>					
37 Sinh viên(s)	Tên đề tài	Lớp	SV	Điểm	Thao tác
<input checked="" type="checkbox"/>	Đề Tài 3	DA21TTA	110121029 Lâm Quốc Huy	0	<input type="button" value="Chỉnh sửa"/> <input type="button" value="PDF"/> <input type="button" value="Xoá"/>
<input checked="" type="checkbox"/>	Đề Tài 3	DA21TTA	110121033 Huỳnh Nhựt Huy	0	<input type="button" value="Chỉnh sửa"/> <input type="button" value="PDF"/> <input type="button" value="Xoá"/>
<input checked="" type="checkbox"/>	Đề Tài 3	DA21TTA	110121041 Thượng Văn Anh Khoa	0	<input type="button" value="Chỉnh sửa"/> <input type="button" value="PDF"/> <input type="button" value="Xoá"/>
3 trong số 37 mục đã chọn					
<input type="button" value="1"/> <input type="button" value="2"/> <input type="button" value="3"/> <input type="button" value="4"/> <input type="button" value="5"/> ... <input type="button" value="8"/> <input type="button" value="&gt;"/>					

Hình 4.59. Chức năng chấm theo nhóm

Để thực hiện chức năng chấm theo nhóm, giảng viên chọn nhóm cần chấm bằng cách sử dụng tùy chọn lọc theo nhóm. Sau khi nhóm đã được chọn, giảng viên đánh dấu tất cả sinh viên trong nhóm bằng cách nhấn vào ô “Chọn hết”. Tiếp theo, giảng viên nhấn vào nút “Chấm theo nhóm” để tiến hành đánh giá cho toàn bộ sinh viên trong nhóm đó.

100004 - IT33_CÔNG NGHỆ PHẦN MỀM DA21TTA_KẾT THÚC MÔN_2024-8-24					
Đề tài Đề tài 3					
Chọn sinh viên					
<input type="text" value="110121029 - Lâm Quốc Huy"/> <input type="text" value="110121033 - Huỳnh Nhựt Huy"/> <input type="text" value="110121041 - Thượng Văn Anh Khoa"/>					
<input type="button" value="←"/> <input checked="" type="radio"/> SV1: 0 TC: 0/6 <input type="radio"/> SV2: 0 TC: 0/6 <input type="radio"/> SV3: 0 TC: 0/6					
CHAPTER	PLO	CLO	Nội dung	Chấm điểm	
IT33_C01: Tổng quan về công nghệ phần mềm	IT_PLO2: Kết hợp kiến thức chuyên môn để giải quyết các vấn đề liên quan đến phần mềm	IT33_CLO1: Trình bày tổng quan về công nghệ phần mềm	Phản mờ đầu hợp lý: Đặc tả nhu cầu ứng dụng phù hợp Nếu được tính cần thiết của phần mềm phân tán/microservices/cloud	Lâm Quốc Huy <input type="radio"/> 0 <input type="radio"/> 0.5 <input type="radio"/> 1 <input type="radio"/> 1.5 Huỳnh Nhựt Huy <input type="radio"/> 0 <input type="radio"/> 0.5 <input type="radio"/> 1 <input type="radio"/> 1.5	0 0 0

Hình 4.60. Trang đánh giá theo nhóm



Hình 4.61. Các tiêu chí đã được giảng viên đánh giá cho nhóm

Khi giảng viên truy cập vào giao diện chấm nhóm, giao diện này sẽ có nhiều điểm tương đồng với giao diện chấm đơn. Tuy nhiên, thay vì chấm điểm cho từng sinh viên riêng lẻ, giảng viên sẽ thực hiện đánh giá đồng loạt cho tất cả các sinh viên trong nhóm đã chọn. Các tiêu chí, mã chương, CDR CT và CDR Môn học vẫn được hiển thị tương tự, và giảng viên có thể điều chỉnh điểm cho các tiêu chí như trong chế độ chấm đơn. Sau khi hoàn thành, giảng viên chỉ cần nhấn lưu để ghi nhận kết quả đánh giá cho cả nhóm.

TRƯỜNG ĐẠI HỌC TRÀ VINH		CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM															
KHOA KỸ THUẬT CÔNG NGHỆ		Độc lập - Tự do - Hạnh Phúc															
<b>PHIẾU CHẤM ĐÁNH GIÁ BÁO CÁO</b> <b>HỌC PHẦN CÔNG NGHỆ PHẦN MỀM</b> (Mã HP: IT33)																	
Chủ đề: 3																	
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 5px;">SV1:</td> <td style="padding: 5px;">Lâm Quốc Huy</td> </tr> <tr> <td style="padding: 5px;">MSSV:</td> <td style="padding: 5px;">110121029</td> </tr> </table>						SV1:	Lâm Quốc Huy	MSSV:	110121029								
SV1:	Lâm Quốc Huy																
MSSV:	110121029																
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 5px;">SV2:</td> <td style="padding: 5px;">Huỳnh Nhựt Huy</td> </tr> <tr> <td style="padding: 5px;">MSSV:</td> <td style="padding: 5px;">110121033</td> </tr> </table>						SV2:	Huỳnh Nhựt Huy	MSSV:	110121033								
SV2:	Huỳnh Nhựt Huy																
MSSV:	110121033																
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 5px;">SV3:</td> <td style="padding: 5px;">Thượng Văn Anh Khoa</td> </tr> <tr> <td style="padding: 5px;">MSSV:</td> <td style="padding: 5px;">110121041</td> </tr> </table>						SV3:	Thượng Văn Anh Khoa	MSSV:	110121041								
SV3:	Thượng Văn Anh Khoa																
MSSV:	110121041																
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center; padding: 5px;">CLO</th> <th style="text-align: center; padding: 5px;">Nội dung báo cáo</th> <th style="text-align: center; padding: 5px;">Điểm</th> <th style="text-align: center; padding: 5px;">SV1</th> <th style="text-align: center; padding: 5px;">SV2</th> <th style="text-align: center; padding: 5px;">SV3</th> </tr> </thead> <tbody> <tr> <td style="padding: 5px;">IT33_CLO1: Trình bày tổng quan về công nghệ phần mềm</td> <td style="padding: 5px;"> <b>Phản mò đầu hợp lý:</b>            Đặc tả nhu cầu ứng dụng phù hợp            Nếu được tính cần thiết của phần mềm phân            tán/microservices/cloud         </td> <td style="text-align: center; padding: 5px;">1.5</td> </tr> </tbody> </table>						CLO	Nội dung báo cáo	Điểm	SV1	SV2	SV3	IT33_CLO1: Trình bày tổng quan về công nghệ phần mềm	<b>Phản mò đầu hợp lý:</b> Đặc tả nhu cầu ứng dụng phù hợp Nếu được tính cần thiết của phần mềm phân tán/microservices/cloud	1.5	1.5	1.5	1.5
CLO	Nội dung báo cáo	Điểm	SV1	SV2	SV3												
IT33_CLO1: Trình bày tổng quan về công nghệ phần mềm	<b>Phản mò đầu hợp lý:</b> Đặc tả nhu cầu ứng dụng phù hợp Nếu được tính cần thiết của phần mềm phân tán/microservices/cloud	1.5	1.5	1.5	1.5												

Hình 4.62. In PDF cho nhóm sinh viên (2/2)

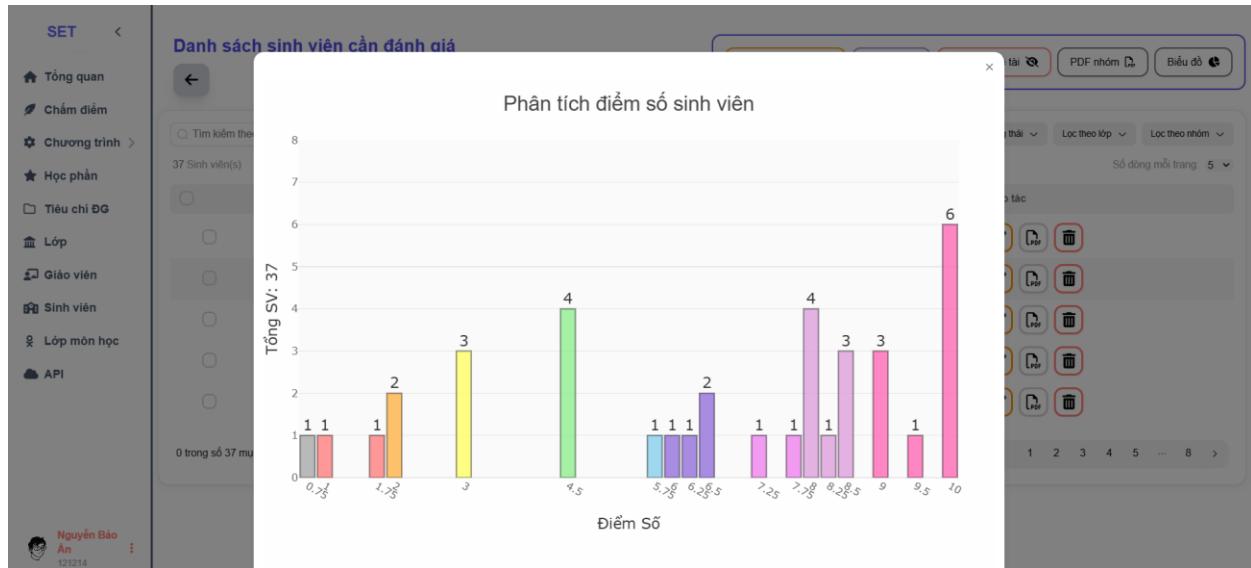
tổng quan về công nghệ phần mềm	Đưa ra nhu cầu ứng dụng phu nợp Nêu được tính cần thiết của phần mềm phân tán/microservices/cloud	1.5	1.5	1.5	1.5
IT33_CLO3: Văn dụng mô hình đã học để xây dựng phần mềm	Kế hoạch phát triển phần mềm: Product backlog Phân chia sprint Sprint backlog Phân công thành phần và kế hoạch Burndown chart hợp lý	2	1.25	2	2
IT33_CLO4: Văn dụng cách thức tổ chức và hoạt động nhóm hiệu quả	Thiết kế kiến trúc phần mềm Mô tả chức năng từng phần trong kiến trúc Có sử dụng RESTful API Sử dụng công nghệ frontend phù hợp Có sử dụng microservices/Phần mềm dựa trên đám mây	2	1.75	2	2
IT33_CLO4: Văn dụng cách thức tổ chức và hoạt động nhóm hiệu quả	Sử dụng Git/GitHub Sử dụng Docker Có triển khai lên host	2	2	2	2
IT33_CLO5: Sử dụng tiếng Anh để đọc tài liệu chuyên ngành	Có khía cạnh về bảo mật và an toàn của phần mềm	1.5	1.5	0.5	0.5
IT33_CLO6: Thể hiện đạo đức nghề nghiệp, tinh thần trách nhiệm và tác phong chuyên nghiệp trong công việc	Kỹ năng thuyết trình	1	0	0	0
	<b>Tổng điểm:</b>	<b>8</b>	<b>8</b>	<b>8</b>	

Lâm Quốc Huy	Huỳnh Nhựt Huy	Thượng Văn Anh Khoa
8	8	8

Trà Vinh, ngày ... tháng ... năm ...  
**GV CHẨM BÁO CÁO**  
**Nguyễn Bảo Ân**

Hình 4.63. In PDF cho nhóm sinh viên (2/2)

Để xuất PDF cho nhóm, giảng viên chuyển sang chế độ lọc theo nhóm và chọn nhóm cần tạo PDF. Sau khi đã chọn nhóm, giảng viên nhấn vào nút “PDF Nhóm.” Hệ thống sẽ tiến hành tạo tệp PDF chứa thông tin đánh giá của toàn bộ sinh viên trong nhóm, bao gồm các tiêu chí và điểm số mà giảng viên đã chấm.



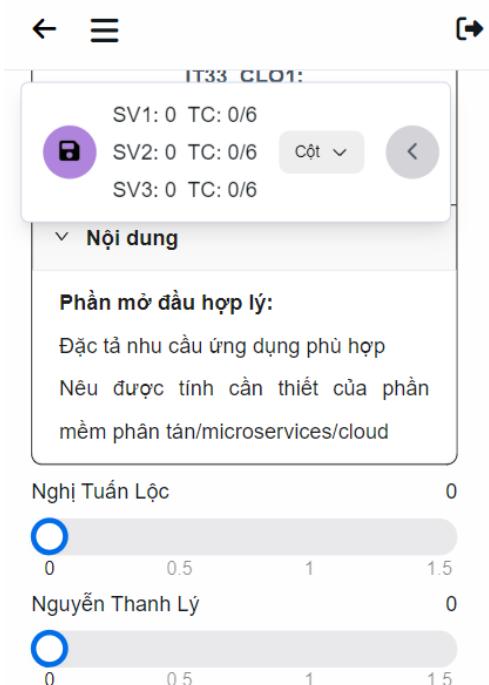
Hình 4.64. Xem biểu đồ phân tích điểm số sinh viên

Để xem biểu đồ phân tích điểm số, giảng viên thực hiện thao tác nhấp vào nút “Biểu đồ”. Khi nút được kích hoạt, một modal sẽ được mở ra, hiển thị biểu đồ phân bố điểm số của sinh viên. Biểu đồ này cung cấp cái nhìn trực quan về dữ liệu điểm số, hỗ trợ giảng viên trong việc đánh giá và phân tích kết quả học tập của sinh viên một cách hiệu quả.

### 4.3. Giao diện chấm cho mobile



Hình 4.65. Giao diện chấm điểm đơn



Hình 4.66. Giao diện chấm điểm nhóm

## CHƯƠNG 5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

### 5.1. Kết luận

Trong quá trình thực hiện đề tài “Xây dựng API và hệ thống chấm điểm trực tuyến cho đồ án sinh viên CNTT,” em đã tích lũy được nhiều kiến thức và kỹ năng quan trọng. Trước hết, em đã nắm vững phương pháp thiết kế API theo mô hình RESTful, một kiến trúc phổ biến và hiệu quả trong phát triển hệ thống phần mềm. Việc làm việc nhóm trong suốt quá trình này cũng giúp em phát triển kỹ năng hợp tác và giao tiếp, từ đó đảm bảo dự án được triển khai đồng bộ và đạt hiệu quả cao.

Bên cạnh đó, em đã tiếp cận và làm quen với các công cụ và công nghệ hiện đại như GitHub để quản lý mã nguồn, Docker để triển khai ứng dụng, và VPS để thử nghiệm hệ thống trong môi trường thực tế. Thông qua quá trình nghiên cứu và phát triển hệ thống, em cũng đã hiểu rõ hơn về quy trình chấm điểm tại trường, giúp em xây dựng một hệ thống phù hợp và tối ưu cho các hoạt động đánh giá.

Việc đối mặt với các bài toán phức tạp trong quá trình phát triển đã rèn luyện cho em kỹ năng giải quyết vấn đề, cũng như khả năng lập trình theo tình huống để đưa ra những giải pháp kịp thời và hiệu quả.

Đặc biệt, em đã phát triển kỹ năng thiết kế ứng dụng cho thiết bị di động, mở ra hướng tiếp cận mới trong việc triển khai hệ thống chấm điểm trực tuyến, giúp nâng cao tính tiện lợi và khả năng tương tác cho người dùng.

Tuy nhiên, trong quá trình triển khai, em nhận thấy một số hạn chế trong việc sử dụng phương thức HTTP. Cụ thể, việc sử dụng phương thức PUT thay cho PATCH trong một số trường hợp chỉ yêu cầu cập nhật một phần dữ liệu đã dẫn đến việc xử lý không tối ưu. Một số API cũng chỉ yêu cầu các tham số đơn giản nhưng lại sử dụng phương thức phức tạp hơn, làm giảm hiệu quả của hệ thống. Bên cạnh đó, việc tích hợp các thư viện UI lớn đã làm tăng dung lượng bộ nhớ của dự án, ảnh hưởng đến hiệu suất, đặc biệt là trên các hệ thống có tài nguyên giới hạn.

### 5.2. Hướng phát triển

Nâng cao trải nghiệm người dùng trên các thiết bị di động bằng cách tối ưu hóa giao diện. Điều này bao gồm việc cải thiện tính trực quan, tốc độ phản hồi, và tính tương thích

của hệ thống trên các nền tảng di động khác nhau.

Sau khi hoàn thiện và thử nghiệm, hệ thống sẽ được áp dụng vào thực tiễn để hỗ trợ quá trình chấm điểm đồ án tại các trường đại học. Việc này sẽ giúp đánh giá hiệu quả và độ tin cậy của hệ thống trong môi trường thực tế, từ đó cải thiện và hoàn thiện hệ thống hơn nữa.

## **DANH MỤC TÀI LIỆU THAM KHẢO**

- [1] J. Wilson, Node.js 8 the Right Way: Practical, Server-Side JavaScript That Scales, O'Reilly Media, 2018.
- [2] E. Hahn, Express in Action, Manning Publications, 2016.
- [3] C. H. a. S. Holmes, Getting MEAN with Mongo, Express.js, Angular, and Node, Second Edition, Manning Publications, 2019.
- [4] S. Stefanov, React: Up & Running, O'Reilly Media, Inc., 2021.
- [5] S. B. a. J. Tinley, High Performance MySQL, O'Reilly Media, Inc, 2021.
- [6] M. A. a. S. R. L. Richardson, RESTful Web APIs, O'Reilly Media, Inc., 2013.

## PHỤ LỤC

PHỤ LỤC 1. Kết nối cơ sở dữ liệu .....	98
PHỤ LỤC 2. Định nghĩa mô hình (model) với Sequelize .....	100
PHỤ LỤC 3. Triển khai VPS Ubuntu với Docker, phpMyAdmin và MySQL .....	104
PHỤ LỤC 4. Trigger SQL cập nhật điểm cuối.....	107
PHỤ LỤC 5. JSON Web Token .....	112
PHỤ LỤC 6. Tạo PDF Động từ HTML bằng Express và Puppeteer .....	116
PHỤ LỤC 7. Thiết lập Tailwind CSS và Responsive với Tailwind.....	118
PHỤ LỤC 8. Cấu hình Axios .....	119
PHỤ LỤC 9. API chương trình .....	119
PHỤ LỤC 10. API của môn học.....	130
PHỤ LỤC 11. API của đánh giá tổng hợp (meta-assessment) .....	156
PHỤ LỤC 12. API lần đánh giá (assessment) .....	170
PHỤ LỤC 13. API của bảng tiêu chí (Rubric) .....	184
PHỤ LỤC 14. API của tiêu chí (Rubric-item).....	192
PHỤ LỤC 15. API của mục tiêu CT (PO).....	197
PHỤ LỤC 16. API của CDR CT (PLO).....	203
PHỤ LỤC 17. API của CDR môn học (CLO).....	208
PHỤ LỤC 18. API của chương môn học .....	212
PHỤ LỤC 19. API của assessment item.....	216
PHỤ LỤC 20. API của ánh xạ CDR CT với mục tiêu CT .....	217
PHỤ LỤC 21. API của ánh xạ CDR CT với CDR môn học .....	218
PHỤ LỤC 22. API của ánh xạ chương môn học với CDR môn học .....	220

## PHỤ LỤC 1. Kết nối cơ sở dữ liệu

### Tạo một instance của Sequelize với các biến môi trường

```
const { Sequelize } = require('sequelize');
const dotenv = require('dotenv');
dotenv.config();

const sequelize = new Sequelize(
  process.env.DB_NAME,
  process.env.DB_USER,
  process.env.DB_PASSWORD,
  {
    host: process.env.DB_HOST,
    dialect: process.env.DB_DIALECT,
  }
);

async function Connection() {
  try {
    await sequelize.authenticate();
    console.log('Kết nối thành công.');
  } catch (error) {
    console.error(
      'Kết nối thất bại', error
    );
  }
}
Connection();
module.exports = sequelize;
```

*Code kết nối cơ sở dữ liệu*



```
const { Sequelize } = require('sequelize');
const dotenv = require('dotenv');
```

*Code cài đặt thư viện Sequelize*

Cài đặt các thư viện:

- **Sequelize**: Đây là một Object-Relational Mapping (ORM) cho phép bạn kết nối và thao tác với cơ sở dữ liệu một cách dễ dàng.
- **dotenv**: Thư viện này giúp quản lý các biến môi trường, giúp bảo mật thông tin nhạy cảm như tên người dùng và mật khẩu cơ sở dữ liệu.

```
● ○ ●  
const sequelize = new Sequelize(  
    process.env.DB_NAME,  
    process.env.DB_USER,  
    process.env.DB_PASSWORD,  
    {  
        host: process.env.DB_HOST,  
        dialect: process.env.DB_DIALECT,  
    }  
)
```

### *Code tạo đối tượng Sequelize*

Một đối tượng Sequelize được tạo ra với các tham số được lấy từ biến môi trường:

- **DB\_NAME**: Tên cơ sở dữ liệu.
- **DB\_USER**: Tên người dùng cơ sở dữ liệu.
- **DB\_PASSWORD**: Mật khẩu cơ sở dữ liệu.
- **DB\_HOST**: Địa chỉ máy chủ cơ sở dữ liệu (ví dụ: localhost).
- **DB\_DIALECT**: Loại cơ sở dữ liệu sử dụng (ví dụ: MySQL, PostgreSQL).

```
● ○ ●  
async function Connection() {  
    try {  
        await sequelize.authenticate();  
        console.log('Kết nối thành công.');//  
    } catch (error) {  
        console.error('Kết nối thất bại', error);  
    }  
}  
Connection();
```

### *Code kiểm tra kết nối*

Hàm Connection sẽ thực hiện các bước sau:

- **authenticate()**: Kiểm tra kết nối với cơ sở dữ liệu.
- Nếu kết nối thành công, thông báo “Kết nối cơ sở dữ liệu thành công” sẽ được

in ra. Nếu kết nối thất bại, thông báo lỗi sẽ được hiển thị.

## PHỤ LỤC 2. Định nghĩa mô hình (model) với Sequelize

Đoạn mã dưới đây mô tả quá trình định nghĩa một mô hình (model) sử dụng thư viện Sequelize để tương tác với bảng programs trong cơ sở dữ liệu. Mô hình này cung cấp cách thức truy xuất và thao tác với dữ liệu thông qua các phương thức ORM (Object-Relational Mapping).

```
const { DataTypes } = require('sequelize');
const sequelize = require('../config/database');

const ProgramModel = sequelize.define('Program', {
  program_id: {
    type: DataTypes.STRING(255),
    primaryKey: true,
    underscored: true
  },
  programName: {
    type: DataTypes.TEXT,
    allowNull: false
  },
  description: {
    type: DataTypes.TEXT,
    allowNull: false
  },
  isDelete: {
    type: DataTypes.TINYINT(1),
    defaultValue: 0
  },
  {
    timestamps: true,
    createdAt: 'createdAt',
    updatedAt: 'updatedAt',
    tableName: 'programs',
    underscored: false
  });
}

module.exports = ProgramModel;
```

*Code tạo model programs*



```
const { DataTypes } = require('sequelize');
const sequelize = require('../config/database');
```

*Code cài đặt thư viện phục vụ tạo model programs*

Cài đặt các thư viện:

- **DataTypes**: Đây là một đối tượng được import từ Sequelize, chứa các kiểu dữ liệu để định nghĩa các cột trong mô hình. Ví dụ, các kiểu dữ liệu phổ biến bao gồm STRING, TEXT, và TINYINT.
- **sequelize**: Đối tượng Sequelize đã được cấu hình kết nối với cơ sở dữ liệu, được import từ tệp database.js trong thư mục config.



```
const ProgramModel = sequelize.define('Program', {
  program_id: {
    type: DataTypes.STRING(255),
    primaryKey: true,
    underscored: true
  },
  programName: {
    type: DataTypes.TEXT,
    allowNull: false
  },
  description: {
    type: DataTypes.TEXT,
    allowNull: false
  },
  isDelete: {
    type: DataTypes.TINYINT(1),
    defaultValue: 0
  },
  {
    timestamps: true,
    createdAt: 'createdAt',
    updatedAt: 'updatedAt',
    tableName: 'programs',
    underscored: false
  });
});
```

*Định nghĩa mô hình Program trong Sequelize*

Cấu trúc của mô hình:

- **ProgramModel**: Mô hình này đại diện cho bảng programs trong cơ sở dữ liệu và được sử dụng để thực hiện các thao tác CRUD (Create, Read, Update, Delete).
- **program\_id**: Cột này có kiểu dữ liệu STRING(255) và được định nghĩa là khóa chính (primaryKey) của bảng. Mặc dù tùy chọn underscored thường được sử dụng để chuyển đổi tên cột thành định dạng snake\_case, nó không ảnh hưởng đến program\_id.
- **programName** và **description**: Cả hai cột này đều có kiểu dữ liệu TEXT và không cho phép giá trị null (allowNull: false), yêu cầu giá trị phải được cung cấp khi tạo hoặc cập nhật bản ghi.
- **isDelete**: Cột này có kiểu dữ liệu TINYINT(1) và được sử dụng để biểu diễn giá trị boolean. Giá trị mặc định là 0, thường được sử dụng để đánh dấu các bản ghi đã bị xóa mềm (soft delete).

Tùy chọn cho mô hình:

- **timestamps: true**: Tùy chọn này tự động thêm các cột createdAt và updatedAt vào bảng, giúp lưu trữ thời gian tạo và cập nhật của mỗi bản ghi.
- **tableName: 'programs'**: Cài đặt này đặt tên bảng là programs thay vì để Sequelize tự động tạo tên dựa trên tên mô hình.
- **underscored: false**: Tùy chọn này ngăn cản việc tự động chuyển đổi tên các cột từ định dạng camelCase sang snake\_case.

## Thiết lập quan hệ giữa mô hình

```
const { DataTypes } = require('sequelize');
const sequelize = require('../config/database');
const ProgramModel = require('./ProgramModel');

const PoModel = sequelize.define('PO', {
  po_id: {
    type: DataTypes.INTEGER.UNSIGNED,
    primaryKey: true,
    autoIncrement: true
  },
  program_id: {
    type: DataTypes.STRING(255),
    allowNull: false,
    references: {
      model: ProgramModel,
      key: 'program_id'
    }
  },
  ...
}

PoModel.belongsTo(ProgramModel, { foreignKey: 'program_id' });
module.exports = PoModel;
```

*Thiết lập mối quan hệ giữa các model*

Thiết lập quan hệ:

- **belongsTo**: Phương thức này thiết lập mối quan hệ giữa PoModel và ProgramModel, cho biết mỗi bản ghi trong bảng PO sẽ liên kết với một bản ghi trong bảng programs.
- **foreignKey: ‘program\_id’**: Thiết lập khóa ngoại program\_id trong bảng PO, liên kết với cột program\_id trong bảng programs. Điều này đảm bảo rằng mỗi bản ghi PO sẽ tham chiếu đến một ProgramModel.

### PHỤ LỤC 3. Triển khai VPS Ubuntu với Docker, phpMyAdmin và MySQL

Cài đặt các gói cần thiết để chạy Docker.

Tạo cấu hình MySQL cho phép sử dụng trigger mà không cần quyền SUPER  
nano my.cnf

```
● ● ● [mysqld] log_bin_trust_function_creators = 1
```

*Cấu hình mysql*

```
version: '3'

services:
  mysql:
    image: mysql:latest
    container_name: mysql
    environment:
      MYSQL_ROOT_PASSWORD: PASSWORD
      MYSQL_DATABASE: TVU
      MYSQL_USER: AdminTVU
      MYSQL_PASSWORD: PASSWORD
    ports:
      - "3306:3306"
    volumes:
      - ./tvu.sql:/docker-entrypoint-initdb.d/tvu.sql
      - ./my.cnf:/etc/mysql/my.cnf
    restart: always
    networks:
      - TVU_Network

  phpmyadmin:
    image: phpmyadmin/phpmyadmin:latest
    container_name: phpmyadmin
    environment:
      PMA_HOST: mysql
      PMA_PORT: 3306
    ports:
      - "8081:80"
    restart: always
    networks:
      - TVU_Network

networks:
  TVU_Network:
    driver: bridge
```

*Cấu hình docker*

## **Phiên bản cấu hình:**

- **version:** ‘3’: Xác định phiên bản của Docker Compose file format. Phiên bản 3 là phiên bản phổ biến và được sử dụng rộng rãi cho các cấu hình Docker Compose.

## **Dịch vụ MySQL:**

- **image:** mysql:latest: Sử dụng image MySQL mới nhất từ Docker Hub.
  - **container\_name:** mysql: Đặt tên cho container là mysql.
  - **environment:** Cung cấp các biến môi trường cho container MySQL, bao gồm:
    - o MySQL\_ROOT\_PASSWORD: Mật khẩu cho người dùng root.
    - o MySQL\_DATABASE: Tên cơ sở dữ liệu sẽ được tạo tự động.
    - o MySQL\_USER: Tên người dùng MySQL.
    - o MySQL\_PASSWORD: Mật khẩu của người dùng MySQL.
  - **ports:** Mở cổng 3306 trên máy chủ và ánh xạ đến cổng 3306 trong container để truy cập cơ sở dữ liệu từ bên ngoài.
  - **volumes:** Gắn kết tệp SQL và tệp cấu hình MySQL từ máy chủ vào container. Tệp ./tvu.sql sẽ được thực thi khi container khởi động để tạo cấu trúc cơ sở dữ liệu ban đầu.
  - **restart: always:** Đảm bảo container sẽ tự động khởi động lại nếu bị dừng hoặc nếu hệ thống khởi động lại.
  - **networks:** Kết nối container với mạng TVU\_Network được định nghĩa dưới đây.
- **Dịch vụ phpMyAdmin:**
- **image:** phpmyadmin/phpmyadmin:latest: Sử dụng image phpMyAdmin mới nhất từ Docker Hub.
  - **Container\_name:** phpmyadmin: Đặt tên cho container là phpmyadmin.
  - **environment:** Cung cấp thông tin về máy chủ MySQL và cổng để phpMyAdmin có thể kết nối:

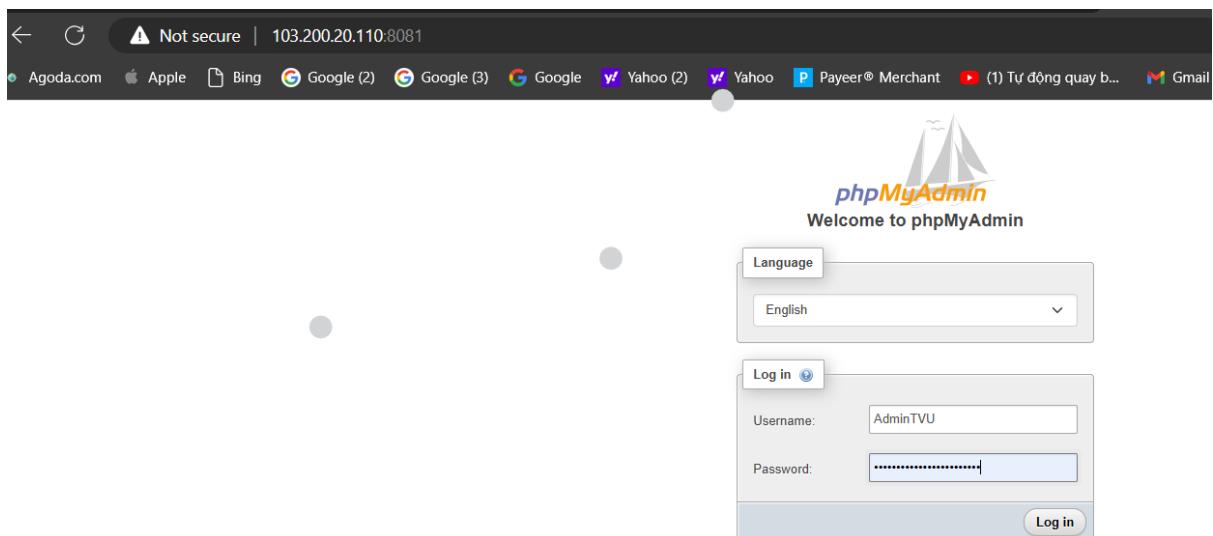
- PMADB\_HOST: Địa chỉ của máy chủ MySQL.
- PMADB\_PORT: Cổng của máy chủ MySQL.
- ports: Mở cổng 8081 trên máy chủ và ánh xạ đến cổng 80 trong container để truy cập phpMyAdmin qua trình duyệt web.
- restart: always: Đảm bảo container sẽ tự động khởi động lại nếu bị dừng hoặc nếu hệ thống khởi động lại.
- networks: Kết nối container với mạng TVU\_Network được định nghĩa dưới đây.

### Mạng TVU\_Network:

- driver: bridge: Sử dụng mạng bridge để kết nối các container với nhau. Mạng bridge là loại mạng mặc định trong Docker, cho phép các container trong cùng một mạng giao tiếp với nhau qua tên dịch vụ.

**Chạy docker:** docker-compose up

### Kiểm tra truy cập địa chỉ vps:



*Kiểm tra URL VPS*

*Kiểm tra URL thành công*

## PHỤ LỤC 4. Trigger SQL cập nhật điểm cuối

### 4.1. Trigger cập nhật FinalScore khi xóa 1 giảng viên phân công.

```

DELIMITER //

CREATE TRIGGER update_final_score_after_assessment_delete
AFTER DELETE ON assessments
FOR EACH ROW
BEGIN
    DECLARE scoreExists INT;
    DECLARE totalSum DOUBLE;
    DECLARE countAssessments INT;

    -- Xác định có bao nhiêu bản ghi còn lại với cùng meta_assessment_id có totalScore =
    -- 0
    SELECT COUNT(*)
    INTO scoreExists
    FROM assessments
    WHERE meta_assessment_id = OLD.meta_assessment_id
        AND totalScore = 0
        AND isDelete = 0;

    -- Nếu có ít nhất một bản ghi có totalScore = 0
    IF scoreExists > 0 THEN
        -- Cập nhật FinalScore thành 0
        UPDATE meta_assessments
        SET FinalScore = 0
        WHERE meta_assessment_id = OLD.meta_assessment_id;
    ELSE
        -- Tính tổng totalScore
        SELECT COALESCE(SUM(totalScore), 0)
        INTO totalSum
        FROM assessments
        WHERE meta_assessment_id = OLD.meta_assessment_id
            AND isDelete = 0;

        -- Tính số lượng bản ghi còn lại
        SELECT COUNT(*)
        INTO countAssessments
        FROM assessments
        WHERE meta_assessment_id = OLD.meta_assessment_id
            AND isDelete = 0;

        -- Nếu có ít nhất một bản ghi hợp lệ, cập nhật FinalScore
        IF countAssessments > 0 THEN
            UPDATE meta_assessments
            SET FinalScore = totalSum / countAssessments
            WHERE meta_assessment_id = OLD.meta_assessment_id;
        ELSE
            -- Nếu không còn bản ghi nào, cập nhật FinalScore thành 0
            UPDATE meta_assessments
            SET FinalScore = 0
            WHERE meta_assessment_id = OLD.meta_assessment_id;
        END IF;
    END IF;
END //

DELIMITER ;

```

*Trigger cập nhật*

Trigger này có nhiệm vụ kiểm tra và cập nhật FinalScore sau khi một bản ghi bị xóa trong bảng assessments. Cụ thể, trigger sẽ thực hiện các bước sau:

### **Khai báo biến:**

- scoreExists: Lưu số lượng bản ghi còn lại với totalScore = 0 cho meta\_assessment\_id cụ thể.
- totalSum: Lưu tổng điểm (totalScore) của tất cả các bản ghi còn lại cho meta\_assessment\_id cụ thể.
- countAssessments: Lưu số lượng bản ghi hợp lệ còn lại cho meta\_assessment\_id cụ thể.

### **Kiểm tra bản ghi có totalScore = 0:**

- Truy vấn kiểm tra số lượng bản ghi còn lại có totalScore = 0 cho meta\_assessment\_id của bản ghi đã bị xóa.

### **Cập nhật FinalScore:**

- Nếu còn bản ghi totalScore = 0: Nếu vẫn còn bản ghi có totalScore = 0, cập nhật FinalScore thành 0.
- Nếu không còn bản ghi totalScore = 0: Nếu không còn bản ghi nào với totalScore = 0, tính toán tổng điểm và số lượng bản ghi hợp lệ còn lại. Cập nhật FinalScore nếu có ít nhất một bản ghi hợp lệ, hoặc đặt FinalScore thành 0 nếu không còn bản ghi hợp lệ nào.

## 4.2. Trigger cập nhật FinalScore khi thêm 1 giảng viên phân công.

```
CREATE TRIGGER update_final_score_after_assessment_insert
AFTER INSERT ON assessments
FOR EACH ROW
BEGIN
    DECLARE scoreExists INT;
    DECLARE totalSum DOUBLE;
    DECLARE countAssessments INT;

    -- Xác định có bất kỳ bản ghi nào với cùng meta_assessment_id có totalScore = 0
    SELECT COUNT(*)
    INTO scoreExists
    FROM assessments
    WHERE meta_assessment_id = NEW.meta_assessment_id
        AND totalScore = 0
        AND isDelete = 0;

    -- Nếu có ít nhất một bản ghi có totalScore = 0
    IF scoreExists > 0 THEN
        -- Cập nhật FinalScore thành 0
        UPDATE meta_assessments
        SET FinalScore = 0
        WHERE meta_assessment_id = NEW.meta_assessment_id;
    ELSE
        -- Tính tổng totalScore
        SELECT COALESCE(SUM(totalScore), 0)
        INTO totalSum
        FROM assessments
        WHERE meta_assessment_id = NEW.meta_assessment_id
            AND isDelete = 0;

        -- Tính số lượng bản ghi
        SELECT COUNT(*)
        INTO countAssessments
        FROM assessments
        WHERE meta_assessment_id = NEW.meta_assessment_id
            AND isDelete = 0;

        -- Nếu có ít nhất một bản ghi hợp lệ, cập nhật FinalScore
        IF countAssessments > 0 THEN
            UPDATE meta_assessments
            SET FinalScore = totalSum / countAssessments
            WHERE meta_assessment_id = NEW.meta_assessment_id;
        END IF;
    END IF;
END;
```

### Khai báo biến:

- scoreExists: Để lưu số lượng bản ghi có totalScore = 0 cho meta\_assessment\_id cụ thể.
- totalSum: Để lưu tổng điểm (totalScore) của tất cả các bản ghi hiện có cho meta\_assessment\_id cụ thể.
- countAssessments: Để lưu số lượng bản ghi hợp lệ hiện có cho meta\_assessment\_id cụ thể.

**Kiểm tra bản ghi có totalScore = 0:**

- Truy vấn để đếm số lượng bản ghi có totalScore = 0 cho meta\_assessment\_id của bản ghi mới.

**Nếu có ít nhất một bản ghi có totalScore = 0:**

- Cập nhật FinalScore trong bảng meta\_assessments thành 0.

**Nếu không có bản ghi nào có totalScore = 0:**

1. Tính tổng điểm (totalScore) của tất cả các bản ghi hiện có liên quan đến meta\_assessment\_id của bản ghi mới.
2. Đếm số lượng bản ghi hiện có hợp lệ.
3. Nếu có ít nhất một bản ghi hợp lệ, cập nhật FinalScore bằng tổng điểm chia cho số lượng bản ghi hợp lệ. Nếu không còn bản ghi hợp lệ nào, đặt FinalScore thành 0.

### 4.3. Trigger cập nhật FinalScore khi giảng viên hoàn thành đánh giá cho sinh viên.

```
DELIMITER //

CREATE TRIGGER update_final_score_after_assessment_update
AFTER UPDATE ON assessments
FOR EACH ROW
BEGIN
    DECLARE scoreExists INT;
    DECLARE totalSum DOUBLE;
    DECLARE countAssessments INT;

    -- Kiểm tra sự tồn tại của các bản ghi với totalScore = 0
    SELECT COUNT(*)
    INTO scoreExists
    FROM assessments
    WHERE meta_assessment_id = NEW.meta_assessment_id
        AND totalScore = 0
        AND isDelete = 0;

    -- Nếu không còn bản ghi nào có totalScore = 0
    IF scoreExists = 0 THEN
        -- Tính tổng totalScore
        SELECT COALESCE(SUM(totalScore), 0)
        INTO totalSum
        FROM assessments
        WHERE meta_assessment_id = NEW.meta_assessment_id
            AND isDelete = 0;

        -- Đếm số lượng bản ghi
        SELECT COUNT(*)
        INTO countAssessments
        FROM assessments
        WHERE meta_assessment_id = NEW.meta_assessment_id
            AND isDelete = 0;

        -- Cập nhật FinalScore nếu có bản ghi hợp lệ
        IF countAssessments > 0 THEN
            UPDATE meta_assessments
            SET FinalScore = totalSum / countAssessments
            WHERE meta_assessment_id = NEW.meta_assessment_id;
        END IF;
    END IF;
END //

DELIMITER ;
```

Khai báo biến:

- scoreExists: Lưu số lượng bản ghi có totalScore = 0 cho meta\_assessment\_id cụ thể.
- totalSum: Lưu tổng điểm (totalScore) của tất cả các bản ghi hiện có cho meta\_assessment\_id cụ thể.
- countAssessments: Lưu số lượng bản ghi hợp lệ hiện có cho meta\_assessment\_id cụ thể.

#### **Kiểm tra bản ghi có totalScore = 0:**

- Truy vấn để đếm số lượng bản ghi có totalScore = 0 cho meta\_assessment\_id của bản ghi đã được cập nhật.

#### **Nếu không còn bản ghi nào có totalScore = 0:**

1. Tính tổng điểm (totalScore) của tất cả các bản ghi hiện có liên quan đến meta\_assessment\_id của bản ghi cập nhật.
2. Đếm số lượng bản ghi hiện có hợp lệ.

Nếu có ít nhất một bản ghi hợp lệ, cập nhật FinalScore bằng tổng điểm chia cho số lượng bản ghi hợp lệ. Nếu không còn bản ghi hợp lệ nào, đặt FinalScore thành 0

#### **PHỤ LỤC 5. JSON Web Token**

Đoạn mã dưới đây cấu hình xác thực JWT (JSON Web Token) sử dụng Passport.js trong một ứng dụng Node.js. Cấu hình này được thực hiện thông qua chiến lược JwtStrategy của thư viện passport-jwt. Các phần chính của đoạn mã được mô tả như sau:



```
const passport = require('passport');
const JwtStrategy = require('passport-jwt').Strategy;
const ExtractJwt = require('passport-jwt').ExtractJwt;
const TeacherModel = require('../models/TeacherModel');
const dotenv = require('dotenv');

dotenv.config();

const opts = {
  jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
  secretOrKey: process.env.JWT_SECRET
};

passport.use(new JwtStrategy(opts, async (jwt_payload, done) => {
  try {
    const user = await TeacherModel.findById(jwt_payload.id);
    if (user) {
      return done(null, user);
    } else {
      return done(null, false);
    }
  } catch (err) {
    return done(err, false);
  }
}));

module.exports = passport;
```

Cấu hình Passport.js với chiến lược JWT trong Node.js



```
const passport = require('passport');
const JwtStrategy = require('passport-jwt').Strategy;
const ExtractJwt = require('passport-jwt').ExtractJwt;
const TeacherModel = require('../models/TeacherModel');
const dotenv = require('dotenv');
```

Import các thư viện cần thiết cho cấu hình Passport.js với JWT

Cài đặt các thư viện:

- **passport**: Thư viện xác thực trong Node.js, hỗ trợ nhiều xác thực khác nhau.
- **JwtStrategy** và **ExtractJwt**: Được nhập từ thư viện passport-jwt, hỗ trợ xác thực thông qua JWT.

```
const opts = {  
    jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),  
    secretOrKey: process.env.JWT_SECRET  
};
```

*Thiết lập tùy chọn chiến lược JWT trong Passport.js*

Thiết lập tùy chọn cho JWT:

- **jwtFromRequest**: Xác định cách thức trích xuất JWT từ yêu cầu HTTP. Ở đây, token được lấy từ phần Authorization của header với kiểu Bearer.
- **secretOrKey**: Khóa bí mật được sử dụng để giải mã và xác thực JWT, được nạp từ biến môi trường JWT\_SECRET.

```
passport.use(new JwtStrategy(opts, async (jwt_payload, done) => {  
    try {  
        const user = await TeacherModel.findById(jwt_payload.id);  
        if (user) {  
            return done(null, user);  
        } else {  
            return done(null, false);  
        }  
    } catch (err) {  
        return done(err, false);  
    }  
}));
```

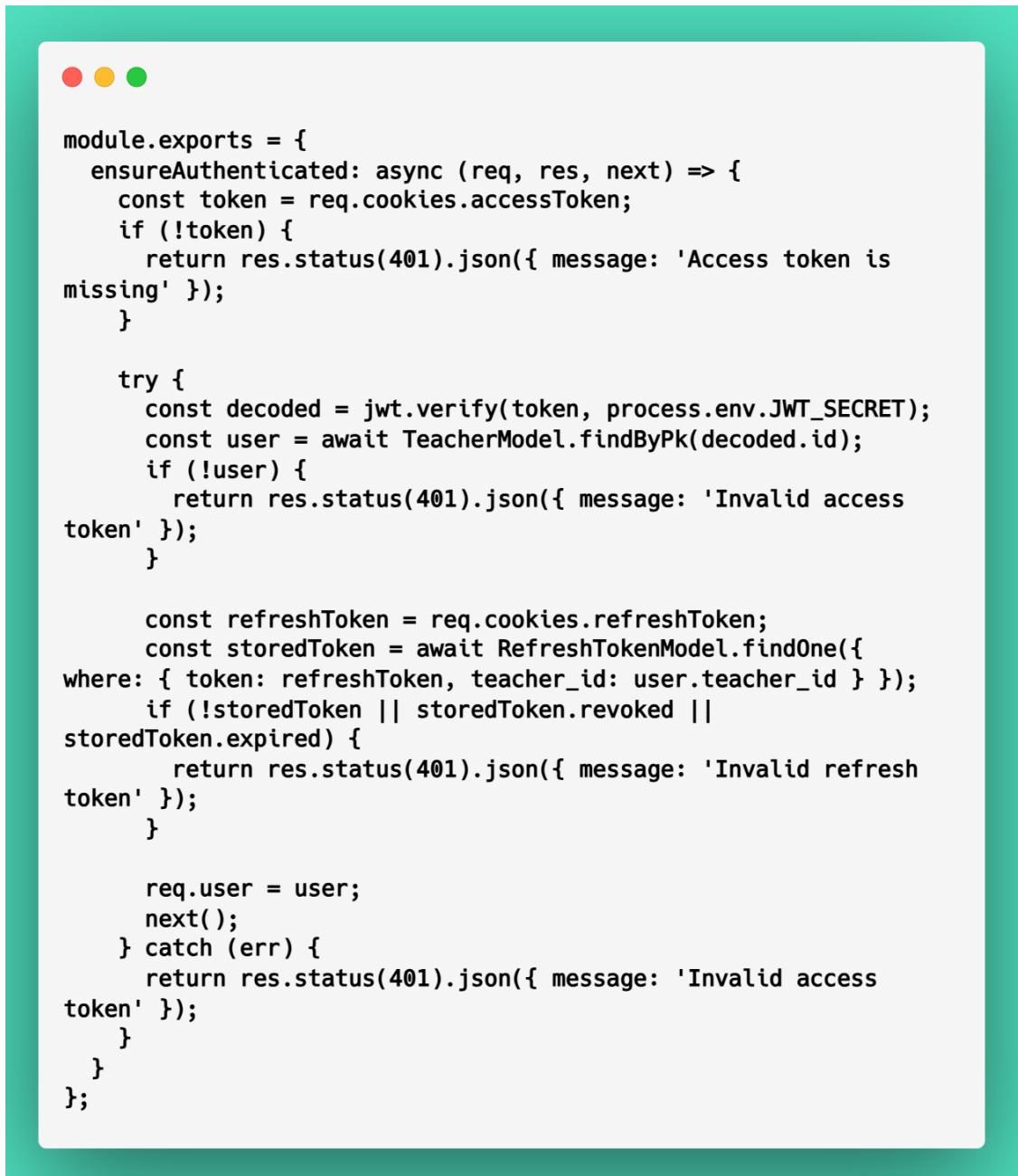
*Xác thực người dùng bằng chiến lược JWT trong Passport.js*

JWT với Passport:

- **JwtStrategy**: Được cấu hình với các tùy chọn đã định nghĩa trong opts.
- **jwt\_payload**: Payload trích xuất từ JWT, chứa thông tin người dùng. jwt\_payload.id được sử dụng để tìm kiếm người dùng trong cơ sở dữ liệu.
- **TeacherModel.findById(jwt\_payload.id)**: Truy vấn cơ sở dữ liệu để tìm

người dùng với id được lấy từ JWT.

- Nếu người dùng tồn tại, hàm done(null, user) sẽ trả về người dùng đó.
- Nếu không tìm thấy người dùng hoặc xảy ra lỗi, hàm done(null, false) hoặc done(err, false) sẽ được gọi để kết thúc quá trình xác thực.



```
module.exports = {
  ensureAuthenticated: async (req, res, next) => {
    const token = req.cookies.accessToken;
    if (!token) {
      return res.status(401).json({ message: 'Access token is missing' });
    }

    try {
      const decoded = jwt.verify(token, process.env.JWT_SECRET);
      const user = await TeacherModel.findById(decoded.id);
      if (!user) {
        return res.status(401).json({ message: 'Invalid access token' });
      }

      const refreshToken = req.cookies.refreshToken;
      const storedToken = await RefreshTokenModel.findOne({
        where: { token: refreshToken, teacher_id: user.teacher_id } });
      if (!storedToken || storedToken.revoked || storedToken.expired) {
        return res.status(401).json({ message: 'Invalid refresh token' });
      }

      req.user = user;
      next();
    } catch (err) {
      return res.status(401).json({ message: 'Invalid access token' });
    }
  };
};
```

#### *Định nghĩa middleware ensureAuthenticated*

Định nghĩa middleware:

- **Kiểm tra Access Token:** Nếu không có token trong yêu cầu, trả về lỗi với mã

trạng thái 401 (Unauthorized). Nếu token có mặt, tiến hành xác thực để đảm bảo tính hợp lệ của nó và truy xuất thông tin người dùng liên quan.

- **Kiểm tra Refresh Token:** Xác thực refresh token để đảm bảo nó hợp lệ. Nếu refresh token không hợp lệ, trả về lỗi với mã trạng thái 401 (Unauthorized).

**Xử lý yêu cầu:** Nếu cả access token và refresh token đều hợp lệ, đặt thông tin người dùng vào đối tượng req.user và tiếp tục xử lý yêu cầu theo quy trình ứng dụng.

## PHỤ LỤC 6. Tạo PDF Động từ HTML bằng Express và Puppeteer

```
● ● ●

const express = require('express');
const router = express.Router();
const puppeteer = require('puppeteer');
const path = require('path');
const { ensureAuthenticated } =
require('../middlewares/authMiddleware');

router.post('/pdf', ensureAuthenticated, async (req, res) => {
  const { html, landscape } = req.body;
  try {
    const browser = await puppeteer.launch();

    const page = await browser.newPage();
    const htmlContent = `${html}`;
    await page.setContent(htmlContent);

    const outputPath = path.join(__dirname,
      './public/pdf/output.pdf');

    const pdfOptions = {
      path: outputPath,
      format: 'A4',
      landscape: landscape,
      margin: {
        top: '2cm',
        bottom: '2cm',
        left: '1cm',
        right: '1cm'
      },
    };
    await page.pdf(pdfOptions);
    await browser.close();
    res.sendFile(outputPath);
  } catch (error) {
    console.error('Error:', error);
    res.status(500).send('Internal Server Error');
  }
});

module.exports = router;
```

Code cấu hình Puppeteer

### Các mô-đun sử dụng:

- express: Được sử dụng để tạo router và định nghĩa các endpoint của API.
- puppeteer: Được sử dụng để tương tác với trình duyệt headless và tạo file PDF

từ nội dung HTML.

- path: Cung cấp các phương thức để xử lý đường dẫn tệp.
- ensureAuthenticated: Đây là middleware dùng để bảo vệ route, chỉ cho phép người dùng đã xác thực truy cập.

### **Route /pdf:**

- Phương thức HTTP: POST.
- Middleware: ensureAuthenticated đảm bảo rằng người dùng phải đăng nhập trước khi tiến hành tạo file PDF.
- Dữ liệu yêu cầu POST:
  - o html: Chuỗi HTML dùng làm nội dung để tạo PDF.
  - o landscape: Một giá trị boolean để xác định định dạng trang PDF là ngang (landscape) hay dọc (portrait).

### **Quy trình tạo file PDF:**

- Khởi chạy trình duyệt headless bằng Puppeteer.
- Tạo một trang mới và thiết lập nội dung trang dựa trên chuỗi HTML nhận được từ yêu cầu.
- Đường dẫn nơi lưu file PDF được xác định thông qua outputPath, sử dụng mô-đun path.
- Các tùy chọn PDF, như kích thước trang, hướng trang (landscape/dọc), và lề trang được cấu hình dựa trên yêu cầu từ phía người dùng.
- Sau khi tạo xong file PDF, file này được trả về cho người dùng bằng phương thức res.sendFile.

### **Xử lý lỗi:**

Trong trường hợp phát sinh lỗi trong quá trình tạo PDF, lỗi sẽ được ghi lại trong console và phản hồi cho người dùng với mã lỗi 500 (Internal Server Error).

## PHỤ LỤC 7. Thiết lập Tailwind CSS và Responsive với Tailwind

### 6.1. Thiết lập Tailwind CSS

```
/* @type {import('tailwindcss').Config} */
const { nextui } = require("@nextui-org/react");

module.exports = {
  content: [
    "./src/**/*.{js,jsx,ts,tsx}",
    "./node_modules/@nextui-org/theme/dist/**/*.{js,ts,jsx,tsx}"
  ],
  theme: {
    extend: {},
  },
  darkMode: "class",
  plugins: [nextui()]
}
```

#### Cấu hình Tailwind

- **content:** Xác định các tệp mà Tailwind CSS sẽ quét để tìm kiếm các lớp CSS. Điều này bao gồm các tệp trong dự án và các thành phần từ NextUI.
- **theme:** Mở rộng cấu hình của theme Tailwind CSS. Hiện tại, không có sự thay đổi nào trong cấu hình theme.
- **darkMode:** Kích hoạt chế độ tối dựa trên các lớp CSS.
- **plugins:** Thêm plugin NextUI để tích hợp các thành phần của NextUI vào các lớp CSS của Tailwind CSS.

Việc cấu hình này cho phép sử dụng đồng bộ các lớp CSS của Tailwind và các thành phần của NextUI trong dự án.

### 6.2. Responsive với Tailwind

Tailwind CSS cung cấp các breakpoint như sm, md, lg, xl, và 2xl để dễ dàng điều chỉnh giao diện trên nhiều kích thước màn hình khác nhau. Các lớp CSS có thể thay đổi theo từng breakpoint.

```
<div class="bg-blue-500 p-4 sm:bg-green-500 md:bg-yellow-500  
lg:bg-red-500 xl:bg-purple-500">  
    Responsive Box  
</div>  
    return done(err, false);  
}  
});  
  
module.exports = passport;
```

### *Responsive với Tailwind*

Mặc định, div có nền màu xanh dương (bg-blue-500) và khoảng đệm là 4 (p-4). Khi kích thước màn hình đạt ngưỡng sm ( $\geq 640\text{px}$ ), màu nền thay đổi thành màu xanh lá cây (bg-green-500). Khi màn hình đạt kích thước md ( $\geq 768\text{px}$ ), nền sẽ chuyển sang màu vàng (bg-yellow-500). Đối với kích thước lg ( $\geq 1024\text{px}$ ), nền sẽ chuyển thành màu đỏ (bg-red-500). Cuối cùng, ở kích thước xl ( $\geq 1280\text{px}$ ), màu nền sẽ trở thành màu tím (bg-purple-500).

## **PHỤ LỤC 8. Cấu hình Axios**

```
import axios from "axios";  
  
const axiosAdmin = axios.create({  
    withCredentials: true,  
    baseURL: process.env.REACT_APP_API_DOMAIN_ADMIN  
});
```

### *Cấu hình Axios*

Đoạn mã này khởi tạo đối tượng axios tùy chỉnh có tên axiosAdmin với hai cấu hình chính: withCredentials: true, cho phép gửi kèm cookie để thực hiện xác thực JWT trong các yêu cầu HTTP, qua đó duy trì trạng thái phiên đăng nhập bảo mật, và baseURL, được thiết lập từ biến môi trường REACT\_APP\_API\_DOMAIN\_ADMIN, xác định URL cơ sở cho tất cả các yêu cầu HTTP gửi đến API quản trị viên.

## **PHỤ LỤC 9. API chương trình**

### **9.1. Lấy thông tin chương trình CNTT**

API Endpoint: GET /api/admin/program/:id

```
getByID: async (req, res) => {
  try {
    const { id } = req.params;
    const program = await ProgramModel.findOne({ where: {
      program_id: id
    } });
    if (!program) {
      return res.status(404).json({ message: 'Program not found' });
    }
    res.status(200).json(program);
  } catch (error) {
    console.error('Error fetching program:', error);
    res.status(500).json({ message: 'Server error' });
  }
}
```

### Hàm *getByID*

Hàm này là một phương thức bất đồng bộ, truy xuất thông tin chi tiết của chương trình từ cơ sở dữ liệu dựa trên id từ URL. Phương thức sử dụng `findOne` từ `ProgramModel` để tìm chương trình có `program_id` khớp với id. Nếu tìm thấy, nó trả về đối tượng chương trình dưới dạng JSON kèm mã HTTP 200. Nếu không, trả về mã HTTP 404 và thông báo chương trình không tồn tại. Khi xảy ra lỗi hệ thống hoặc truy vấn, hàm trả về mã HTTP 500 cùng thông báo lỗi và ghi lỗi vào `console`, giúp đảm bảo tính ổn định của hệ thống.

## 9.2. Tạo mới CT

API Endpoint: POST /api/admin/program

```
create: async (req, res) => {
  try {
    const { data } = req.body;
    const newProgram = await ProgramModel.create(data);
    res.status(201).json({ message: 'Data saved successfully',
      data: newProgram });
  } catch (error) {
    console.error('Error creating program:', error);
    res.status(500).json({ message: 'Server error' });
  }
},
```

### Hàm *create*

Hàm này là một phương thức bất đồng bộ, dùng để thêm chương trình mới vào cơ

sở dữ liệu dựa trên dữ liệu từ yêu cầu HTTP. Phương thức lấy thông tin từ phần thân của yêu cầu và sử dụng hàm create của ProgramModel để khởi tạo một bản ghi mới. Nếu thành công, hàm trả về đối tượng chương trình mới ở dạng JSON cùng mã HTTP 201, báo hiệu tài nguyên đã được tạo. Nếu gặp lỗi, chẳng hạn dữ liệu không hợp lệ hoặc lỗi hệ thống, hàm trả về mã HTTP 500 cùng thông báo lỗi, đồng thời ghi lại thông tin lỗi vào console để hỗ trợ gỡ lỗi.

### 9.3. Cập nhật CT

API Endpoint: PUT /api/admin/program/:id

```
update: async (req, res) => {
    try {
        const { id } = req.params;
        const { data } = req.body;
        const [updatedCount] = await ProgramModel.update(data, {
            where: { program_id: id } });
        if (updatedCount === 0) {
            return res.status(404).json({ message: 'Program not found' });
        }
        res.status(200).json({ message: 'Program updated successfully' });
    } catch (error) {
        console.error('Error updating program:', error);
        res.status(500).json({ message: 'Server error' });
    }
},
```

*Hàm update*

Hàm này là một phương thức bất đồng bộ, dùng để cập nhật thông tin chương trình dựa trên ID và dữ liệu mới từ yêu cầu HTTP. Đầu tiên, hàm trích xuất id từ URL và dữ liệu từ phần thân của yêu cầu. Sau đó, hàm sử dụng phương thức update của ProgramModel để cập nhật bản ghi có program\_id khớp với id. Nếu không có bản ghi nào được thay đổi (kết quả là 0), hàm trả về mã HTTP 404 và thông báo chương trình không tồn tại. Nếu cập nhật thành công, hàm phản hồi mã HTTP 200 kèm thông báo xác nhận. Trong trường hợp lỗi, hàm trả về mã HTTP 500 và ghi lỗi vào console để hỗ trợ gỡ lỗi.

## 9.4. Xóa CT

```
delete: async (req, res) => {
  try {
    const { id } = req.params;
    const deletedCount = await ProgramModel.destroy({ where: {
      program_id: id
    } });
    if (deletedCount === 0) {
      return res.status(404).json({ message: 'Program not found' });
    }
    res.status(200).json({ message: 'Program deleted successfully' });
  } catch (error) {
    console.error('Error deleting program:', error);
    res.status(500).json({ message: 'Server error' });
  }
},
```

### *Hàm delete*

Hàm này là một phương thức bất đồng bộ, được thiết kế để xóa một chương trình khỏi cơ sở dữ liệu dựa trên ID từ URL yêu cầu. Sau khi trích xuất id, hàm sử dụng phương thức destroy của ProgramModel để xóa chương trình có program\_id trùng khớp với id. Nếu không có bản ghi nào bị xóa (kết quả là 0), hàm phản hồi với mã HTTP 404 và thông báo chương trình không tồn tại. Nếu xóa thành công, hàm trả về mã HTTP 200 kèm thông báo xác nhận. Trong trường hợp lỗi, hàm trả về mã HTTP 500 và ghi lỗi vào console để hỗ trợ quá trình gỡ lỗi.

## 9.5. Nhận mẫu tạo CT bằng Excel

API Endpoint: GET /api/admin/program/templates/post

```
getFormPost: async (req, res) => {
    const workbook = new ExcelJS.Workbook();
    const programWorksheet = workbook.addWorksheet('Program');
    programWorksheet.columns = [
        { header: 'Mã chương trình', key: 'program_id', width: 20 },
        { header: 'Tên chương trình', key: 'programName', width: 20
    ],
        { header: 'Mô tả (Html)', key: 'description', width: 65 },
    ];
    programWorksheet.addRow({ program_id: 'IT', programName:
'Chương trình mẫu', description: '<p>Mô tả HTML mẫu</p>' });
    await protectWorksheet(programWorksheet, ['program_id']);
```

### *Hàm getFormPost(1/3)*

Hàm này được thiết kế để tạo tệp Excel chứa thông tin liên quan đến chương trình học, bao gồm các bảng dữ liệu như “Program”, “PO”, “PLO”, và “PLO\_PO”. Quá trình bắt đầu bằng việc tạo một workbook mới qua thư viện ExcelJS bằng phương thức Workbook(). Đầu tiên, hàm tạo worksheet “Program” và xác định các cột với tiêu đề: Mã chương trình, Tên chương trình, và Mô tả (HTML). Kích thước cột được điều chỉnh phù hợp với nội dung.

Sau đó, hàm thêm một hàng dữ liệu mẫu, với Mã chương trình là “IT” và Mô tả chứa chuỗi HTML mô tả chi tiết chương trình. Cuối cùng, hàm gọi protectWorksheet để bảo vệ cột Mã chương trình, đảm bảo rằng chỉ các ô không khóa mới có thể chỉnh sửa, giúp bảo vệ thông tin quan trọng trong worksheet.

```
const poWorksheet = workbook.addWorksheet('PO');
poWorksheet.columns = [
    { header: 'Tên Mục tiêu', key: 'poName', width: 20 },
    { header: 'Mô tả', key: 'description', width: 65 },
    { header: 'Mã chương trình', key: 'program_id', width: 20 },
];
poWorksheet.addRow({ poName: 'P01', description: 'P01',
program_id: 'IT' });

const ploWorksheet = workbook.addWorksheet('PL0');
ploWorksheet.columns = [
    { header: 'Tên Chuẩn đầu ra', key: 'ploName', width: 20 },
    { header: 'Mô tả', key: 'description', width: 65 },
    { header: 'Mã chương trình', key: 'program_id', width: 20 },
];
ploWorksheet.addRow({ ploName: 'PL01', description: '',
program_id: 'IT' });

const ploPoWorksheet = workbook.addWorksheet('PL0_PO');
ploPoWorksheet.columns = [
    { header: 'PL0', key: 'ploName', width: 20 },
    { header: 'PO', key: 'poName', width: 20 },
];
ploPoWorksheet.addRow({ ploName: 'PL01', poName: 'PL02' });
```

### *Hàm getFormPost(2/3)*

Để tạo worksheet thứ hai có tên “PO”, các cột được định nghĩa với tiêu đề: Tên Mục tiêu, Mô tả, và Mã chương trình. Sau khi thiết lập cột, một hàng dữ liệu được thêm vào với thông tin cho “PO1”, đảm bảo tính chính xác và đầy đủ.

Tiếp theo, worksheet thứ ba, “PLO”, sẽ được tạo với các cột có tiêu đề: Tên Chuẩn đầu ra, Mô tả, và Mã chương trình. Một hàng dữ liệu minh họa cho “PLO1” cũng sẽ được thêm vào để làm mẫu cho nội dung.

Cuối cùng, worksheet thứ tư, “PL0\_PO”, sẽ chứa hai cột: PLO và PO. Một hàng dữ liệu mẫu sẽ được thêm vào, thể hiện mối quan hệ giữa “PLO1” và “PO1”, nhằm minh họa sự liên kết giữa chuẩn đầu ra và các mục tiêu trong chương trình học.

```
const protectOptions = {
    selectLockedCells: true,
    selectUnlockedCells: true
};

await Promise.all([
    programWorksheet.protect('yourpassword', protectOptions),
]);

res.setHeader('Content-Type', 'application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet');
res.setHeader('Content-Disposition', 'attachment;
filename="ProgramsForm.xlsx"');
await workbook.xlsx.write(res);
res.end();
},|
```

### *Hàm getFormPost(3/3)*

Sau khi hoàn tất việc thiết lập các worksheet, hàm sẽ áp dụng chế độ bảo vệ cho worksheet “Program” với các tùy chọn cụ thể, cho phép người dùng lựa chọn giữa các ô đã được khóa và chưa được khóa. Cuối cùng, hàm thiết lập các tiêu đề phản hồi để chỉ định rằng nội dung gửi đi là tệp Excel và thực hiện gửi workbook đến client với tên tệp là “ProgramsForm.xlsx”. Tất cả các bước này đều được thực hiện một cách đồng bộ, đảm bảo rằng tệp Excel được tạo và gửi đi chính xác.

## **9.6. Lưu CT bằng Excel**

API Endpoint: POST /api/admin/importExcel/program

```
processSaveTemplate: async (req, res) => {
  if (!req.files) {
    return res.status(400).send('No file uploaded.');
  }

  const uploadDirectory = path.join(__dirname, '../uploads');
  const filename = req.files[0].filename;
  const filePath = path.join(uploadDirectory, filename);

  const workbook = new ExcelJS.Workbook();
  try {
    await workbook.xlsx.readFile(filePath);
  } catch (error) {
    return res.status(500).json({ message: 'Error reading the
uploaded file' });
  }

  const ProgramWorksheet = workbook.getWorksheet('Program');
  const PoWorksheet = workbook.getWorksheet('PO');
  const PloWorksheet = workbook.getWorksheet('PLO');
  const PloPoWorksheet = workbook.getWorksheet('PLO_PO');

  const jsonProgramData = [];
  const jsonPoData = [];
  const jsonPloData = [];
  const jsonPloPoData = [];

  // Read Program data
  ProgramWorksheet.eachRow((row, rowNum) => {
    if (rowNum > 1) {
      jsonProgramData.push({
        program_id: row.getCell(1).value,
        programName: row.getCell(2).value,
        description: row.getCell(3).value,
      });
    }
  });
}
```

#### Hàm processSaveTemplate(1/4)

Hàm này được thiết kế để xử lý việc tải lên và lưu trữ dữ liệu từ tệp Excel. Trong giai đoạn đầu, hàm kiểm tra xem có tệp nào được tải lên hay không; nếu không có, nó sẽ trả về mã lỗi 400 kèm theo thông báo “No file uploaded.” Nếu tệp đã được tải lên, hàm xác định đường dẫn lưu trữ và lấy tên tệp từ đối tượng req.files.

Tiếp theo, một workbook mới được khởi tạo thông qua thư viện ExcelJS, và hàm sẽ cố gắng đọc tệp Excel từ đường dẫn đã xác định. Nếu có lỗi xảy ra trong quá trình đọc tệp, hàm sẽ trả về mã lỗi 500 cùng với thông báo lỗi tương ứng.

```

const program_id_PoWorksheet = [];
PoWorksheet.eachRow((row, rowNum) => {
  if (rowNum > 1) {
    const id = row.getCell(3).value;
    if (id) {
      program_id_PoWorksheet.push(id);
    }
  }
});
PoWorksheet.eachRow((row, rowNum) => {
  if (rowNum > 1) {
    jsonPoData.push({
      poName: row.getCell(1).value,
      description: row.getCell(2).value,
      program_id: program_id_PoWorksheet[0],
    });
  }
});
const program_id_PloWorksheet = [];
PloWorksheet.eachRow((row, rowNum) => {
  if (rowNum > 1) {
    const id = row.getCell(3).value;
    if (id) {
      program_id_PloWorksheet.push(id);
    }
  }
});
PloWorksheet.eachRow((row, rowNum) => {
  if (rowNum > 1) {
    jsonPloData.push({
      ploName: row.getCell(1).value,
      description: row.getCell(2).value,
      program_id: program_id_PloWorksheet[0],
    });
  }
});
PloPoWorksheet.eachRow((row, rowNum) => {
  if (rowNum > 1) {
    jsonPloPoData.push({
      ploName: row.getCell(1).value,
      poName: row.getCell(2).value,
    });
  }
});

```

#### Hàm processSaveTemplate(2/4)

Mảng `program_id_PoWorksheet` được khởi tạo để lưu trữ mã chương trình từ worksheet “PO”. Mã chương trình được lấy từ ô thứ ba của mỗi hàng, bắt đầu từ hàng thứ hai, và nếu mã chương trình tồn tại, nó sẽ được thêm vào mảng này.

Tiếp theo, hàm `eachRow` được sử dụng để đọc từng hàng trong worksheet “PO”, và dữ liệu sẽ được lưu vào `jsonPoData`. Trong mảng này, tên mục tiêu (PO) sẽ được lấy từ ô đầu tiên, mô tả từ ô thứ hai, và mã chương trình sẽ được gán từ phần tử đầu tiên của mảng `program_id_PoWorksheet`.

Quá trình tương tự sẽ được lặp lại cho worksheet “PLO”. Mảng program\_id\_PloWorksheet sẽ được khởi tạo để lưu trữ mã chương trình từ worksheet “PLO”, với mã chương trình cũng được lấy từ ô thứ ba. Dữ liệu từ worksheet “PLO” sẽ được đọc và lưu vào jsonPloData, trong đó tên chuẩn đầu ra (PLO) được lấy từ ô đầu tiên, mô tả từ ô thứ hai, và mã chương trình từ phần tử đầu tiên của mảng program\_id\_PloWorksheet.

Cuối cùng, dữ liệu từ worksheet “PLO\_PO” sẽ được đọc và lưu trữ vào jsonPloPoData, trong đó tên PLO và tên PO sẽ được lấy từ các ô tương ứng.

```
const fetchPloPoIds = async (ploName, poName) => {
  try {
    const plo = await PloModel.findOne({ where: { ploName } });
    const po = await PoModel.findOne({ where: { poName } });

    if (plo && po) {
      return {
        plo_id: plo.plo_id,
        po_id: po.po_id,
      };
    }
    return null;
  } catch (error) {
    console.error('Error fetching PLO or PO:', error);
    return null;
  }
};
```

Hàm processSaveTemplate(3/4)

Hàm fetchPloPoIds là một hàm bất đồng bộ được thiết kế để tìm kiếm ID của các PLO và PO trong cơ sở dữ liệu. Hàm này nhận vào hai tham số: ploName và poName, và sử dụng các mô hình PloModel và PoModel để thực hiện việc tìm kiếm.

Nếu cả hai mục tiêu được tìm thấy, hàm sẽ trả về một đối tượng chứa plo\_id và po\_id. Ngược lại, nếu không tìm thấy hoặc khi xảy ra lỗi, hàm sẽ trả về giá trị null và ghi lại thông báo lỗi tương ứng trong console.

```

try {
    const createdPrograms = await
ProgramModel.bulkCreate(jsonProgramData);

    const [createdPlos, createdPos] = await Promise.all([
        PloModel.bulkCreate(jsonPloData),
        PoModel.bulkCreate(jsonPoData),
    ]);
    const ploIds = createdPlos.map(plo => plo.plo_id);
    const poIds = createdPos.map(po => po.po_id);
    const ploPoIds = await Promise.all(
        jsonPloPoData.map(async (item) => {
            const ids = await fetchPloPoIds(item.ploName,
item.poName);
            return ids;
        })
    );
    const filteredPloPoIds = ploPoIds.filter(id => id !==
null);
    await PoPloModel.bulkCreate(filteredPloPoIds);

    fs.unlinkSync(filePath);
    res.status(201).json({ message: 'Data saved successfully',
data: createdPrograms });
} catch (error) {
    console.error('Error saving data to the database:', error);
    res.status(500).json({ message: 'Error saving data to the
database' });
}
}

```

#### Hàm processSaveTemplate(4/4)

Đoạn mã này thực hiện chức năng lưu trữ dữ liệu vào cơ sở dữ liệu. Đầu tiên, nó sử dụng phương thức bulkCreate của ProgramModel để tạo ra nhiều bản ghi chương trình dựa trên dữ liệu trong jsonProgramData.

Tiếp theo, thông qua phương thức Promise.all, mã đồng thời lưu trữ các mục tiêu học tập (PLO) và mục tiêu chương trình (PO) bằng cách sử dụng bulkCreate cho cả PloModel và PoModel. Sau khi quá trình lưu trữ hoàn tất, mã sẽ thu thập danh sách ID của các PLO và PO đã được tạo.

Mã sau đó tạo một mảng chứa các ID PLO và PO tương ứng bằng cách gọi hàm fetchPloPoIds cho mỗi mục trong jsonPloPoData. Sau khi nhận được danh sách ID, mã sẽ lọc ra các giá trị không hợp lệ (null) và lưu trữ các mối quan hệ PLO-PO vào bảng

PoPloModel bằng cách sử dụng bulkCreate.

Cuối cùng, mã sẽ xóa tệp đã được tải lên và trả về phản hồi thành công với mã trạng thái 201, kèm theo thông báo và dữ liệu của các chương trình đã được tạo. Trong trường hợp có lỗi xảy ra trong quá trình này, mã sẽ ghi lại thông tin lỗi và trả về phản hồi với mã trạng thái 500.

## PHỤ LỤC 10. API của môn học

### 10.1. Lấy thông tin tất cả môn học

API Endpoint: GET /api/admin/subjects

**Tham số truy vấn:**

**teacher\_id:** Tham số này được sử dụng để xác định các môn học mà giảng viên cụ thể đang giảng dạy.

**course\_id:** Tham số này cho phép tìm kiếm môn học theo ID của khóa học, hỗ trợ việc phân loại nội dung giảng dạy.

**isDelete:** Tham số này giúp phân loại các môn học dựa trên trạng thái bị xóa, nâng cao khả năng thống kê và phân tích dữ liệu.



```
getSubjects: async (req, res) => {
  try {
    const { teacher_id, isDelete, course_id } = req.query;
    const whereCondition = {};
    if (teacher_id) {
      whereCondition.teacher_id = teacher_id;
    }
    if (isDelete !== undefined) {
      whereCondition.isDelete = JSON.parse(isDelete);
    }
  }
}
```

*Hàm getSubjects(1/3)*

Hàm này là một hàm bất đồng bộ (async) được thiết kế để truy xuất danh sách các môn học từ cơ sở dữ liệu theo các điều kiện lọc cụ thể. Hàm bắt đầu bằng việc lấy các tham số từ yêu cầu truy vấn (req.query), bao gồm teacher\_id, isDelete, và course\_id. Các tham số này có vai trò quan trọng trong việc xác định các điều kiện lọc cho các môn học trong

cơ sở dữ liệu.

Tiếp theo, một đối tượng whereCondition được khởi tạo nhằm lưu trữ các điều kiện lọc cho truy vấn. Nếu tham số teacher\_id tồn tại, nó sẽ được thêm vào đối tượng này. Đối với tham số isDelete, nếu giá trị không phải là undefined, hàm sẽ chuyển đổi giá trị từ chuỗi JSON sang kiểu boolean và đưa vào điều kiện lọc. Cuối cùng, hàm sẽ thực hiện truy vấn đến cơ sở dữ liệu sử dụng các điều kiện đã được xác định, từ đó trả về danh sách các môn học phù hợp với các tiêu chí lọc đã chỉ định.

```
● ● ●

if (course_id) {
    const course = await CourseModel.findOne({ where: {
course_id } });
    if (!course) {
        return res.status(404).json({ message: 'Course not found' });
    }

    const subject = await SubjectModel.findOne({ where: {
subject_id: course.subject_id } });
    if (!subject) {
        return res.status(404).json({ message: 'Subject not
found' });
    }
    return res.status(200).json(subject);
}
```

*Hàm getSubjects(2/3)*

Hàm getSubjects là một hàm bất đồng bộ (async) dùng để truy xuất danh sách các môn học từ cơ sở dữ liệu dựa trên các điều kiện lọc, bao gồm teacher\_id, isDelete, và course\_id. Khi nhận được tham số course\_id, hàm sẽ tìm kiếm khóa học tương ứng. Nếu không tìm thấy khóa học, hàm trả về mã trạng thái 404 và thông báo “Course not found”. Nếu khóa học tồn tại, hàm sẽ tiếp tục tìm kiếm các môn học liên quan và thông báo lỗi nếu môn học không được tìm thấy.

```

const subjects = await SubjectModel.findAll({
    where: whereCondition
});

// Nếu không có subjects, trả về 404
if (!subjects.length) {
    return res.status(404).json({ message: 'No subjects found' });
}

// Lấy CLOs và Chapters liên quan cho các subjects
const subjectIds = subjects.map(subject =>
subject.subject_id);
const Clos = await CloModel.findAll({ where: { subject_id: subjectIds } });
const Chapters = await ChapterModel.findAll({ where: { subject_id: subjectIds } });

for (const subject of subjects) {
    subject.dataValues.CLO = Clos.filter(clos =>
clos.subject_id === subject.subject_id);
    subject.dataValues.CHAPTER = Chapters.filter(chapter =>
chapter.subject_id === subject.subject_id);
}

res.status(200).json(subjects);
} catch (error) {
    console.error('Error fetching subjects:', error);
    res.status(500).json({ message: 'Internal server error' });
}
},

```

### *Hàm getSubjects(3/3)*

Trong trường hợp không có course\_id, hàm getSubjects sẽ truy vấn để lấy danh sách tất cả các môn học dựa trên các điều kiện đã được xác định trước. Nếu không tìm thấy môn học nào, mã trạng thái 404 sẽ được trả về cùng với thông báo “No subjects found”. Sau khi truy xuất danh sách môn học, hàm sẽ tiếp tục lấy các CLOs và Chapters liên quan, và gán các thông tin này vào thuộc tính tương ứng của từng môn học. Cuối cùng, hàm trả về danh sách môn học kèm theo thông tin bổ sung về CLOs và Chapters dưới dạng phản hồi JSON. Trong trường hợp xảy ra lỗi, mã sẽ ghi lại thông tin lỗi và trả về mã trạng thái 500 cùng với thông báo “Internal server error”.

## **10.2. Lấy thông tin 1 môn học**

API Endpoint: GET /api/admin/subject/:id

**Tham số truy vấn:**

**include\_clos:** Tham số này xác định việc có bao gồm các Course Learning Outcomes (CLOs) trong phản hồi hay không. Nếu được đặt là true, API sẽ trả về thông tin

chi tiết về các CLO liên quan đến môn học.

**include\_chapters:** Tham số này cho phép người dùng xác định việc có bao gồm các chương (Chapters) trong phản hồi hay không. Khi giá trị là true, API sẽ trả về thông tin về các chương liên quan đến môn học.

**only\_clo\_ids:** Tham số này chỉ định liệu API có chỉ trả về các ID của các CLO hay không. Nếu được đặt là true, phản hồi sẽ chỉ bao gồm danh sách ID của CLO mà không có thông tin mô tả.

**only\_chapter\_ids:** Tương tự như only\_clo\_ids, tham số này xác định liệu API có chỉ trả về các ID của các chương hay không. Khi giá trị được đặt là true, phản hồi sẽ chỉ bao gồm danh sách ID của chương mà không kèm theo thông tin chi tiết.

VD: GET /api/admin/subject/123? include\_clos=true & include\_chapters=true & only\_clo\_ids=true & only\_chapter\_ids=false

```
● ● ●

getByID: async (req, res) => {
    try {
        const { id } = req.params;
        const { include_clos, include_chapters, only_clo_ids,
only_chapter_ids } = req.query;

        // Tìm thông tin subject
        const subject = await SubjectModel.findOne({ where: {
            subject_id: id
        } });
        if (!subject) {
            return res.status(404).json({ message: 'Subject not found' });
        }
        const response = { subject };
    }
}
```

Hàm *getByID(1/5)*

Hàm này là một hàm bất đồng bộ (async) dùng để truy xuất thông tin chi tiết về một môn học (subject) dựa trên subject\_id được cung cấp. Quá trình thực hiện bắt đầu bằng việc thu thập các tham số từ yêu cầu truy vấn (request), bao gồm id (mã môn học) và các tham số tùy chọn như include\_clos, include\_chapters, only\_clo\_ids, và only\_chapter\_ids.

Sau đó, hàm sử dụng mô hình SubjectModel để tìm kiếm thông tin tương ứng với

subject\_id trong cơ sở dữ liệu. Nếu không tìm thấy môn học, hàm sẽ trả về mã trạng thái HTTP 404 cùng với thông báo “Subject not found” để thông báo cho người dùng. Ngược lại, nếu môn học được tìm thấy, hàm sẽ tạo một đối tượng response với các thông tin chi tiết về môn học, đảm bảo cung cấp dữ liệu cần thiết cho các bước xử lý tiếp theo

```
● ● ●  
if (include_clos === 'true') {  
    const clos = await CloModel.findAll({  
        where: { subject_id: id, isDelete: false },  
        attributes: ['clo_id', 'cloName', 'description']  
    });  
    response.clos = clos.length ? clos : [];  
}
```

*Hàm getByID(2/5)*

Nếu tham số include\_clos được thiết lập là true, hàm sẽ truy vấn để lấy danh sách các CLOs liên quan đến môn học. Kết quả sẽ được thêm vào response, và nếu không có CLO nào, một mảng rỗng sẽ được thêm vào.

```
● ● ●  
if (include_chapters === 'true') {  
    const chapters = await ChapterModel.findAll({  
        where: { subject_id: id, isDelete: false },  
        attributes: ['chapter_id', 'chapterName', 'description']  
    });  
    response.chapters = chapters.length ? chapters : [];  
}
```

*Hàm getByID(3/5)*

Tương tự, nếu tham số include\_chapters được thiết lập là true, hàm sẽ truy vấn các chương (chapters) liên quan đến môn học và thêm vào response.



```
if (only_clo_ids === 'true') {
    const cloIds = await CloModel.findAll({
        where: { subject_id: id, isDelete: false },
        attributes: ['clo_id']
    });
    response.clo_ids = cloIds.map(clo => clo.clo_id);
}

if (only_chapter_ids === 'true') {
    const chapterIds = await ChapterModel.findAll({
        where: { subject_id: id, isDelete: false },
        attributes: ['chapter_id']
    });
    response.chapter_ids = chapterIds.map(chapter =>
    chapter.chapter_id);
}
```

#### *Hàm getByID(4/5)*

Nếu tham số `only_clo_ids` được thiết lập là `true`, hàm sẽ truy vấn chỉ để lấy các ID của CLOs và thêm vào phản hồi dưới dạng một mảng. Tương tự, nếu tham số `only_chapter_ids` được thiết lập là `true`, hàm sẽ truy vấn để lấy các ID của Chapters và thêm vào phản hồi.



```
res.status(200).json(response);
} catch (error) {
    console.error('Error fetching subject details:', error);
    res.status(500).json({ message: 'Internal server error' });
}
},
```

#### *Hàm getByID(5/5)*

Cuối cùng, hàm sẽ trả về một phản hồi JSON với mã trạng thái 200, chứa tất cả thông tin đã được thu thập. Nếu có bất kỳ lỗi nào xảy ra trong quá trình thực thi hàm, mã sẽ ghi log lỗi và trả về mã trạng thái 500 với thông báo “Internal server error”.

### 10.3. Lấy thông tin tất cả bảng tiêu chí dựa vào mã môn học

API Endpoint: GET /api/admin/subject/:id/rubrics

Tham số truy vấn: teacher\_id



```
getRubricsBySubjectId: async (req, res) => {
  try {
    const { id } = req.params;
    const { teacher_id } = req.query;
    const subject = await SubjectModel.findOne({ where: {
      subject_id: id
    } });
    if (!subject) {
      return res.status(404).json({ message: 'Subject not found' });
    }

    const whereConditions = { subject_id: id };
    if (teacher_id) {
      whereConditions.teacher_id = teacher_id;
    }
    const rubrics = await RubricModel.findAll({ where:
      whereConditions });
    if (!rubrics.length) {
      return res.status(404).json({ message: 'Rubrics not found' });
    }

    res.status(200).json(rubrics);
  } catch (error) {
    console.error('Error finding rubrics:', error);
    res.status(500).json({ message: 'Server error' });
  }
},
```

Hàm *getRubricsBySubjectId*

Hàm này được thiết kế để truy xuất danh sách rubrics (tiêu chí đánh giá) cho một môn học cụ thể, dựa trên mã môn học (subject\_id) và mã giảng viên (teacher\_id).

Hàm bắt đầu bằng việc thu thập subject\_id từ URL và teacher\_id từ truy vấn. Nó kiểm tra sự tồn tại của môn học qua mô hình SubjectModel. Nếu không tìm thấy, hàm trả về mã trạng thái 404 và thông báo “Subject not found”.

Nếu môn học tồn tại, hàm thiết lập điều kiện lọc và tìm kiếm rubrics qua mô hình

RubricModel. Nếu không có rubrics, mã trạng thái 404 và thông báo “Rubrics not found” được trả về. Ngược lại, hàm trả về rubrics dưới dạng JSON với mã trạng thái 200.

Nếu xảy ra lỗi, hàm ghi log và gửi phản hồi với mã trạng thái 500 cùng thông báo “Server error”.

#### 10.4. Cập nhật 1 môn học

API Endpoint: PUT /api/admin/subject/:id

```
● ● ●

update: async (req, res) => {
  try {
    const { id } = req.params;
    const { data } = req.body;
    const subject = await SubjectModel.findOne({ where: {
      subject_id: id
    } });
    if (!subject) {
      return res.status(404).json({ message: 'Subject not found' });
    }
    await SubjectModel.update(data, { where: { subject_id: id } });
    res.status(200).json({ message: `Successfully updated subject with ID: ${id}` });
  } catch (error) {
    console.error('Error updating subject:', error);
    res.status(500).json({ message: 'Internal server error' });
  }
},
```

*Hàm update*

Hàm này được thiết kế để cập nhật thông tin của một môn học cụ thể dựa trên mã môn học (subject\_id) từ tham số trong yêu cầu. Quá trình bắt đầu bằng việc lấy subject\_id từ URL và dữ liệu cập nhật từ phần thân yêu cầu.

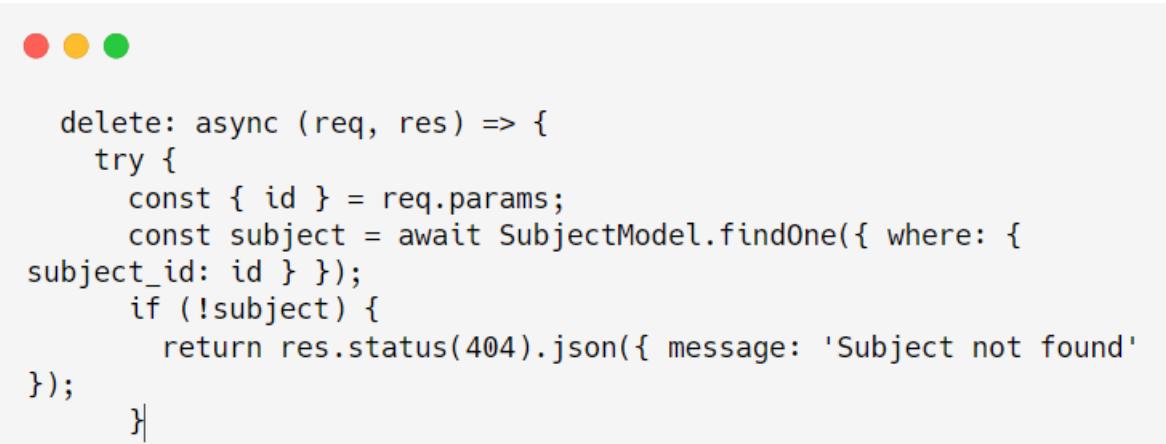
Hàm sử dụng mô hình SubjectModel để kiểm tra sự tồn tại của môn học. Nếu không tìm thấy, hàm trả về mã trạng thái 404 với thông báo “Subject not found”.

Nếu môn học tồn tại, hàm thực hiện cập nhật thông tin bằng phương thức update trên SubjectModel, sử dụng dữ liệu mới và điều kiện xác định môn học. Sau khi cập nhật thành công, hàm trả về mã trạng thái 200 cùng thông báo xác nhận và mã ID của môn học.

Nếu xảy ra lỗi trong quá trình thực hiện, hàm ghi lại lỗi và trả về mã trạng thái 500 cùng thông báo “Internal server error”, thông báo cho người dùng về sự cố.

## 10.5. Xóa 1 môn học

API Endpoint: DELETE /api/admin/subject/:id



```
delete: async (req, res) => {
  try {
    const { id } = req.params;
    const subject = await SubjectModel.findOne({ where: {
      subject_id: id
    } });
    if (!subject) {
      return res.status(404).json({ message: 'Subject not found' })
    }
  }
```

*Hàm delete(1/3)*

Hàm này được thiết kế để xóa một môn học dựa trên mã môn học (subject\_id) từ tham số trong yêu cầu. Khi hàm được gọi, nó sẽ lấy subject\_id từ URL để xác định môn học cần xóa.

Hàm sử dụng mô hình SubjectModel để tìm kiếm môn học tương ứng với subject\_id. Nếu không tìm thấy, hàm sẽ trả về mã trạng thái 404 cùng thông báo “Subject not found”.



```
const clos = await CloModel.findAll({ where: { subject_id: subject.subject_id } });
for (const clo of clos) {
    await CloChapterModel.destroy({ where: { clo_id: clo.clo_id } });
    await PloCloModel.destroy({ where: { clo_id: clo.clo_id } });
    await CloModel.destroy({ where: { clo_id: clo.clo_id } });
}

const chapters = await ChapterModel.findAll({ where: { subject_id: subject.subject_id } });
for (const chapter of chapters) {
    await CloChapterModel.destroy({ where: { chapter_id: chapter.chapter_id } });
    await RubricItemModel.destroy({ where: { chapter_id: chapter.chapter_id } });
    await ChapterModel.destroy({ where: { chapter_id: chapter.chapter_id } });
}
```

#### *Hàm delete(2/3)*

Đoạn mã xóa CLO thực hiện theo quy trình rõ ràng. Đầu tiên, hàm tìm tất cả các CLO liên quan đến môn học thông qua mô hình CloModel, nhằm xác định các kết nối giữa CLO và môn học cụ thể.

Sau khi xác định danh sách CLO, hàm tiếp tục xóa các liên kết giữa CLO và chương học (Chapters), đảm bảo không còn sự phụ thuộc nào giữa chúng trong cơ sở dữ liệu. Tiếp theo, hàm xóa tất cả các mục liên quan đến CLO, như đánh giá hoặc tiêu chí đánh giá. Cuối cùng, hàm thực hiện xóa CLO khỏi hệ thống.



```
await SubjectModel.destroy({ where: { subject_id: subject.subject_id } });
res.status(200).json({ message: 'Successfully deleted subject' });
} catch (error) {
    console.error('Error deleting subject:', error);
    res.status(500).json({ message: 'Internal server error' });
}
},
```

### *Hàm delete(3/3)*

Cuối cùng, nó xóa môn học và trả về phản hồi thành công.

## **10.6. Xóa nhiều môn học**

API Endpoint: DELETE /api/admin/subjects/multiple



```
deleteMultiple: async (req, res) => {
  const { subject_id } = req.query;
  try {
    const subjectIds = subject_id.map(id => parseInt(id));
    for (const id of subjectIds) {

      const subject = await SubjectModel.findOne({ where: {
        subject_id: id
      } });
      if (!subject) {
        return res.status(404).json({ message: 'Subject not
found' });
      }
    }
  }
}
```

### *Hàm deleteMultiple(1/3)*

Hàm này được thiết kế để xóa nhiều môn học dựa trên danh sách các subject\_id từ query parameters.

Duyệt qua từng subject\_id, tìm kiếm trong cơ sở dữ liệu. Nếu môn học không tồn tại, trả về mã trạng thái 404 và thông báo “Subject not found”.

```
const clos = await CloModel.findAll({ where: { subject_id: id } });
for (const clo of clos) {
    await CloChapterModel.destroy({ where: { clo_id: clo.clo_id } });
    await PloCloModel.destroy({ where: { clo_id: clo.clo_id } });
    await CloModel.destroy({ where: { clo_id: clo.clo_id } });
}
}

const chapters = await ChapterModel.findAll({ where: { subject_id: id } });
for (const chapter of chapters) {
    await CloChapterModel.destroy({ where: { chapter_id: chapter.chapter_id } });
    await RubricItemModel.destroy({ where: { chapter_id: chapter.chapter_id } });
    await ChapterModel.destroy({ where: { chapter_id: chapter.chapter_id } });
}
```

#### *Hàm deleteMultiple(2/3)*

Tìm các CLO và chương liên quan đến mỗi môn học, xóa các liên kết giữa chúng (CloChapter, PloClo) và xóa chính các CLO và chương.

```
await SubjectModel.destroy({ where: { subject_id: subjectIds } });

res.status(200).json({ message: 'Xóa nhiều CLO thành công' });
} catch (error) {
    console.error('Lỗi khi xóa nhiều CLO:', error);
    res.status(500).json({ message: 'Lỗi server nội bộ' });
},

```

#### *Hàm deleteMultiple(3/3)*

Cuối cùng, xóa tất cả các môn học trong danh sách và trả về mã trạng thái 200 kèm thông báo xác nhận thành công.

## 10.7. Tạo mới 1 môn học

API Endpoint: POST /api/admin/subject

```
create: async (req, res) => {
  try {
    const { data } = req.body;
    const newSubject = await SubjectModel.create(data);
    res.status(201).json(newSubject);
  } catch (error) {
    console.error('Error creating subject:', error);
    res.status(500).json({ message: 'Internal server error' });
  }
},
```

*Hàm create*

Hàm này có chức năng tạo mới một môn học trong cơ sở dữ liệu. Đầu tiên, hàm nhận thông tin môn học từ phần thân của yêu cầu, sau đó sử dụng phương thức SubjectModel.create để tạo một bản ghi môn học mới trong cơ sở dữ liệu. Nếu quá trình tạo thành công, hàm sẽ trả về thông tin của môn học mới kèm mã trạng thái 201 (Created). Ngược lại, nếu có lỗi xảy ra, hàm sẽ trả về mã trạng thái 500 cùng thông báo lỗi.

## 10.8. Lấy tất cả thông tin môn học chưa ẩn

API Endpoint: GET /api/admin/subjects/isDelete/false



```
isDeleteTofalse: async (req, res) => {
    try {
        const activeSubjects = await SubjectModel.findAll({
            where: { isDelete: false }
        });
        const subjectIds = activeSubjects.map(subject =>
subject.subject_id);
        const Clos = await CloModel.findAll({ where: { subject_id: subjectIds } });
        const Chapter = await ChapterModel.findAll({ where: { subject_id: subjectIds } });
        for (const subject of activeSubjects) {
            const closForSubject = Clos.filter(clos => clos.subject_id
=== subject.subject_id);
            const ChapterForSubject = Chapter.filter(Chapter =>
Chapter.subject_id === subject.subject_id);
            subject.dataValues.CLO = closForSubject;
            subject.dataValues.CHAPTER = ChapterForSubject;
        }
        res.status(200).json(activeSubjects);
    } catch (error) {
        console.error('Error fetching active subjects:', error);
        res.status(500).json({ message: 'Internal server error' });
    }
},
```

#### *Hàm isDeleteTofalse*

Mục đích của hàm này là lấy tất cả các môn học có isDelete đặt thành false, tức là các môn học đang là không bị xóa, và sau đó tìm và kết hợp thêm thông tin về CLO và các chương tương ứng với những môn học đó.

Đầu tiên, hàm truy xuất danh sách các môn học từ cơ sở dữ liệu có trạng thái chưa bị xóa (isDelete = false). Đây là bước lọc các môn học đang hoạt động để phục vụ cho quá trình xử lý tiếp theo.

Sau đó, hàm tìm kiếm và liên kết các kết quả học tập (CLO) và các chương (Chapter) liên quan đến từng môn học thông qua các bảng liên kết. Mỗi môn học sẽ được gán thêm thông tin chi tiết về các CLO và chương.

Cuối cùng, danh sách các môn học cùng với thông tin về CLO và chương sẽ được trả về dưới dạng JSON. Nếu có lỗi xảy ra trong quá trình thực thi, hàm sẽ trả về mã trạng

thái 500 kèm thông báo “Internal server error”.

### 10.9. Lấy tất cả thông tin môn học đã ẩn

API Endpoint: GET /api/admin/subjects/isDelete/true



```
isDeleteTotrue: async (req, res) => {
    try {
        const deletedSubjects = await SubjectModel.findAll({ where: { isDelete: true } });
        res.status(200).json(deletedSubjects);
    } catch (error) {
        console.error('Error fetching deleted subjects:', error);
        res.status(500).json({ message: 'Internal server error' });
    }
},
```

*Hàm isDeleteTotrue*

Hàm này có chức năng truy xuất danh sách các môn học đã bị xóa (được đánh dấu isDelete là true). Hàm được định nghĩa dưới dạng bất đồng bộ (async) và thực hiện như sau:

Đầu tiên, hàm sử dụng mô hình SubjectModel để tìm tất cả các môn học có trạng thái isDelete là true. Sau khi truy xuất thành công, hàm sẽ trả về danh sách các môn học đã xóa dưới dạng JSON cùng với mã trạng thái 200. Nếu có lỗi xảy ra trong quá trình thực hiện, hàm sẽ ghi lại lỗi và trả về mã trạng thái 500 cùng với thông báo “Internal server error”.

### 10.10. Xóa mềm 1 mục tiêu CT

API Endpoint: PUT /api/admin/subject/:id/softDelete

```

toggleSoftDeleteById: async (req, res) => {
    try {
        const { id } = req.params;
        const subject = await SubjectModel.findOne({ where: {
            subject_id: id
        } });
        if (!subject) {
            return res.status(404).json({ message: 'subject not found' });
        }
        const updatedIsDeleted = !subject.isDelete;
        await SubjectModel.update({ isDelete: updatedIsDeleted }, {
            where: { subject_id: id }
        });

        res.status(200).json({ message: `Toggled isDelete status to ${updatedIsDeleted}` });
    } catch (error) {
        console.error('Error toggling SubjectModel delete statuses:', error);
        res.status(500).json({ message: 'Server error' });
    }
},

```

### *Hàm toggleSoftDeleteById*

Hàm này có chức năng thay đổi trạng thái xóa mềm của một môn học dựa trên subject\_id được cung cấp trong tham số yêu cầu. Quy trình thực hiện của hàm như sau:

Đầu tiên, hàm thu thập subject\_id từ các tham số của yêu cầu và sử dụng mô hình SubjectModel để tìm kiếm môn học tương ứng. Nếu không tìm thấy môn học, hàm trả về mã trạng thái 404 cùng thông báo “subject not found”.

Nếu môn học tồn tại, hàm sẽ đảo ngược trạng thái isDelete của môn học đó và cập nhật giá trị mới vào cơ sở dữ liệu. Cuối cùng, hàm trả về mã trạng thái 200 cùng thông báo xác nhận rằng trạng thái isDelete đã được thay đổi thành công.

Trong trường hợp xảy ra lỗi trong quá trình thực hiện, hàm sẽ ghi lại lỗi và trả về mã trạng thái 500 với thông báo “Server error”. Hàm này cung cấp phương pháp linh hoạt để quản lý trạng thái xóa của các môn học trong hệ thống.

## 10.11. Xóa mềm nhiều mục tiêu CT

API Endpoint: PUT /api/admin/subjects/softDelete

```
softDeleteMultiple: async (req, res) => {
    try {
        const { data } = req.body;
        const { subject_id } = data;
        if (!Array.isArray(subject_id) || subject_id.length === 0) {
            return res.status(400).json({ message: 'No SubjectModel ids provided' });
        }

        const subjects = await SubjectModel.findAll({ where: { subject_id: subject_id } });
        if (subjects.length !== subject_id.length) {
            return res.status(404).json({ message: 'One or more SubjectModels not found' });
        }

        const updatedSubjects = await Promise.all(subjects.map(async (subject) => {
            const updatedIsDeleted = !subject.isDelete;
            await subject.update({ isDelete: updatedIsDeleted });
            return { subject_id: subject.subject_id, isDelete: updatedIsDeleted };
        }));

        res.json({ message: 'SubjectModel delete statuses toggled', updatedSubjects });
    } catch (error) {
        console.error('Error toggling SubjectModel delete status:', error);
        res.status(500).json({ message: 'Server error' });
    }
},
```

*Hàm softDeleteMultiple*

Hàm này thực hiện chức năng thay đổi trạng thái xóa mềm cho nhiều môn học dựa trên danh sách subject\_id nhận từ yêu cầu. Đầu tiên, hàm kiểm tra tính hợp lệ của danh sách subject\_id; nếu không hợp lệ, nó trả về mã trạng thái 400 cùng thông báo tương ứng.

Tiếp theo, hàm xác thực sự tồn tại của các môn học trong cơ sở dữ liệu; nếu một hoặc nhiều môn học không được tìm thấy, mã trạng thái 404 sẽ được trả về. Sau khi xác

nhận, hàm cập nhật trạng thái isDelete của từng môn học bằng cách sử dụng Promise.all, cho phép thực hiện cập nhật đồng thời và trả về thông tin về các môn học đã được cập nhật.

Cuối cùng, nếu có lỗi xảy ra trong quá trình thực hiện, hàm sẽ gửi phản hồi với mã trạng thái 500 và thông báo lỗi.

### 10.12. Nhận mẫu tạo môn học bằng Excel

API Endpoint: GET /api/admin/subject/templates/post

```
getFormPost: async (req, res) => {
    const workbook = new ExcelJS.Workbook();
    const SubjectWorksheet = workbook.addWorksheet('Subject');

    SubjectWorksheet.columns = [
        { header: 'Tên HP', key: 'subjectName', width: 30 },
        { header: 'Mã HP', key: 'subjectCode', width: 30 },
        { header: 'Mô tả', key: 'description', width: 100 },
        { header: 'Số tín chỉ', key: 'numberCredits', width: 10 },
        { header: 'STC LT', key: 'numberCreditsTheory', width: 10 },
        { header: 'STC TH', key: 'numberCreditsPractice', width: 10
    },
        { header: 'Loại học phần (Đại cương, Cơ sở ngành, Chuyên ngành, Thực tập và Đô án)', key: 'typesubject', width: 40 },
    ];

    const clooWorksheet = workbook.addWorksheet('CLO');
    clooWorksheet.columns = []
        { header: 'Mã Môn học', key: 'subjectCode', width: 20 },
        { header: 'Mã CDR', key: 'cloName', width: 20 },
        { header: 'Mô tả', key: 'description', width: 65 },
        { header: 'Loại', key: 'type', width: 20 },
    ];
}
```

*Hàm getFormPost(1/2)*

Hàm này được định nghĩa để tạo ra một tệp Excel chứa thông tin liên quan đến các học phần (môn học), bao gồm các worksheet cho “Subject”, “CLO”, “Chapter”, “CLO\_PLO”, và “CLO\_CHAPTER”.

Quá trình bắt đầu bằng việc khởi tạo một workbook mới thông qua thư viện ExcelJS. Đầu tiên, worksheet “Subject” được tạo với các cột bao gồm Tên HP, Mã HP, Mô tả, Số tín chỉ, STC LT, STC TH, và Loại học phần, với kích thước cột được thiết lập phù hợp.

Tiếp theo, worksheet “CLO” được thêm vào với các cột tương tự, bao gồm Mã Môn học, Mã CDR, Mô tả, và Loại.

```
● ● ●

const chapterWorksheet = workbook.addWorksheet('Chapter');
chapterWorksheet.columns = [
    { header: 'Mã Môn học', key: 'subjectCode', width: 20 },
    { header: 'Mã Chapter', key: 'chapterName', width: 20 },
    { header: 'Mô tả', key: 'description', width: 65 },
];
| 
const CLO_PLOWorksheet = workbook.addWorksheet('CLO_PLO');
CLO_PLOWorksheet.columns = [
    { header: 'Mã CLO', key: 'cloName', width: 20 },
    { header: 'Mã PLO', key: 'ploName', width: 20 },
];
| 
const CLO_CHAPTERWorksheet =
workbook.addWorksheet('CLO_CHAPTER');
CLO_CHAPTERWorksheet.columns = [
    { header: 'Mã CLO', key: 'cloName', width: 20 },
    { header: 'Mã CHAPTER', key: 'chapterName', width: 20 },
];
| 
res.setHeader('Content-Type', 'application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet');
res.setHeader('Content-Disposition', 'attachment;
filename="SubjectForm.xlsx"');
await workbook.xlsx.write(res);
res.end();
},
```

*Hàm getFormPost(2/2)*

Sau đó, worksheet “Chapter” cũng được tạo với các tiêu đề như Mã Môn học, Mã Chapter, và Mô tả. Tiếp tục, worksheet “CLO\_PLO” và “CLO\_CHAPTER” được thiết lập với các cột tương ứng cho việc liên kết giữa CLO và PLO cũng như giữa CLO và Chapter.

Cuối cùng, hàm thiết lập tiêu đề cho phản hồi để chỉ định rằng nội dung đang gửi là một tệp Excel, và gửi workbook đến client với tên tệp là “SubjectForm.xlsx”. Tất cả các bước này đảm bảo rằng tệp Excel được tạo ra và gửi đi một cách chính xác và đầy đủ.

### 10.13. Lưu môn học bằng Excel

API Endpoint: POST /api/admin/importExcel/subject



```
processSaveTemplateSubject: async (req, res) => {
  if (!req.files || req.files.length === 0) {
    return res.status(400).send('No file uploaded.');
  }

  const teacher_id = req.user.teacher_id;

  try {
    // Verify teacher exists
    const teacher = await TeacherModel.findOne({ where: {
      teacher_id } });
    if (!teacher) {
      return res.status(404).json({ message: 'Teacher not found' });
    }
  }

  const uploadDirectory = path.join(__dirname, '../uploads');
  const filename = req.files[0].filename;
  const filePath = path.join(uploadDirectory, filename);

  const workbook = new ExcelJS.Workbook();
  await workbook.xlsx.readFile(filePath);
```

#### *Hàm processSaveTemplateSubject(1/8)*

Hàm này có chức năng xử lý việc tải lên tệp mẫu cho môn học. Đầu tiên, hàm kiểm tra xem có tệp nào được tải lên hay không; nếu không có, nó trả về mã trạng thái 400 với thông báo “No file uploaded.” Tiếp theo, hàm lấy teacher\_id từ thông tin người dùng đã xác thực và kiểm tra sự tồn tại của giảng viên trong cơ sở dữ liệu. Nếu không tìm thấy giảng viên, hàm trả về mã trạng thái 404 kèm thông báo “Teacher not found.” Sau đó, nó xác định đường dẫn lưu trữ tệp và lấy tên tệp từ yêu cầu. Cuối cùng, nó khởi tạo một workbook mới bằng thư viện ExcelJS và đọc tệp Excel từ đường dẫn đã xác định để tiến hành xử lý tiếp theo.

```

const [subjects, clos, chapters, PloClos, CloChapters] = await
Promise.all([
    SubjectModel.findAll({ attributes: ['subject_id',
'subjectCode'] }),
    CloModel.findAll({ attributes: ['cloName'] }),
    ChapterModel.findAll({ attribSutes: ['chapterName'] }),
    PloCloModel.findAll({
        include: [
            { model: PloModel, attributes: ['ploName'] },
            { model: CloModel, attributes: ['cloName'] },
        ],
    }),
    CloChapterModel.findAll({
        include: [
            { model: CloModel, attributes: ['cloName'] },
            { model: ChapterModel, attributes: ['chapterName'] },
        ],
    }),
]);

```

```

const PloClosCodes = PloClos.map(item => ({
    cloName: item.CLO.cloName,
    ploName: item.PLO.ploName,
}));
const CloChaptersCodes = CloChapters.map(item => ({
    cloName: item.CLO.cloName,
    chapterName: item.Chapter.chapterName,
}));

```

#### *Hàm processSaveTemplateSubject(2/8)*

Đoạn mã tiếp theo thực hiện việc lấy dữ liệu song song từ nhiều mô hình trong cơ sở dữ liệu bằng cách sử dụng Promise.all. Cụ thể, nó lấy danh sách các môn học, kết quả học tập (CLO), chương học, cùng với các liên kết giữa PLO và CLO, cũng như giữa CLO và chương. Các thuộc tính cần thiết được chỉ định rõ ràng cho từng mô hình để tối ưu hóa dữ liệu. Sau đó, nó ánh xạ các kết quả học tập và chương học để tạo ra hai mảng PloClosCodes và CloChaptersCodes, trong đó mỗi phần tử chứa tên CLO và PLO hoặc tên chương tương ứng, giúp dễ dàng xử lý thông tin trong các bước tiếp theo.



```
const SubjectWorksheet = workbook.getWorksheet('Subject');
const CLOWorksheet = workbook.getWorksheet('CLO');
const ChapterWorksheet = workbook.getWorksheet('Chapter');
const CLO_PLOWorksheet = workbook.getWorksheet('CLO_PLO');
const CLO_CHAPTERWorksheet =
workbook.getWorksheet('CLO_CHAPTER');

const SubjectData = [];
const CLOData = [];
const ChapterData = [];
const CLO_PLOData = [];
const CLO_CHAPTERData = [];

SubjectWorksheet.eachRow((row, rowNum) => {
    if (rowNum > 1) {
        SubjectData.push({
            subjectName: row.getCell(1).value,
            subjectCode: row.getCell(2).value,
            description: row.getCell(3).value,
            teacher_id: teacher.teacher_id,
            numberCredits: row.getCell(4).value,
            numberCreditsTheory: row.getCell(5).value,
            numberCreditsPractice: row.getCell(6).value,
            typesubject: row.getCell(7).value,
        });
    }
});

CLOWorksheet.eachRow((row, rowNum) => {
    if (rowNum > 1) {
        CLOData.push({
            subjectCode: row.getCell(1).value,
            cloName: row.getCell(2).value,
            description: row.getCell(3).value,
            type: row.getCell(4).value,
        });
    }
});|
```

*Hàm processSaveTemplateSubject(3/8)*

```

ChapterWorksheet.eachRow((row, rowNum) => {
  if (rowNum > 1) {
    ChapterData.push({
      subjectCode: row.getCell(1).value,
      chapterName: row.getCell(2).value,
      description: row.getCell(3).value,
    });
  }
});

CLO_PLOWorksheet.eachRow((row, rowNum) => {
  if (rowNum > 1) {
    CLO_PLOData.push({
      cloName: row.getCell(1).value,
      ploName: row.getCell(2).value,
    });
  }
});

CLO_CHAPTERWorksheet.eachRow((row, rowNum) => {
  if (rowNum > 1) {
    CLO_CHAPTERData.push({
      cloName: row.getCell(1).value,
      chapterName: row.getCell(2).value,
    });
  }
});

```

#### *Hàm processSaveTemplateSubject(4/8)*

Tiếp theo, đoạn mã khởi tạo các worksheet và mảng dữ liệu trống để lưu trữ thông tin từ worksheet “Subject”. Nó sử dụng phương thức eachRow để duyệt qua từng hàng, bắt đầu từ hàng thứ hai (bỏ qua tiêu đề). Tại đây, mã trích xuất các thông tin như tên môn học, mã môn học, mô tả, số tín chỉ, và loại môn học, sau đó thêm vào mảng SubjectData. Các worksheet khác như “CLO”, “Chapter”, “CLO\_PLO”, và “CLO\_CHAPTER” cũng được xử lý tương tự, thu thập thông tin và lưu vào các mảng tương ứng (CLOData, ChapterData, CLO\_PLOData, và CLO\_CHAPTERData).

```

// Filter and insert new subjects
const filteredSubjectData = SubjectData.filter(subject =>
!subjects.some(s => s.subjectCode === subject.subjectCode));
const createdSubjects = filteredSubjectData.length > 0 ?
await SubjectModel.bulkCreate(filteredSubjectData) : [];
const allSubjects = [...subjects, ...createdSubjects];

```

#### *Hàm processSaveTemplateSubject(5/8)*

Đoạn mã sau thực hiện việc lọc và chèn các môn học mới vào cơ sở dữ liệu. Nó tạo ra một mảng filteredSubjectData, chỉ bao gồm các môn học chưa có trong danh sách hiện tại, dựa trên mã môn học (subjectCode). Nếu có môn học mới, mã sẽ sử dụng bulkCreate của SubjectModel để chèn chúng vào cơ sở dữ liệu; nếu không, một mảng rỗng sẽ được gán cho createdSubjects. Cuối cùng, nó kết hợp danh sách các môn học đã có và các môn học mới thành một mảng allSubjects, cung cấp cái nhìn tổng thể về tất cả các môn học trong hệ thống.

```
// Create CLO and Chapter results
const CLOResults = CLOData.map(item => {
  const subject = allSubjects.find(s => s.subjectCode === item.subjectCode);
  if (subject) {
    return {
      subject_id: subject.subject_id,
      cloName: item.cloName,
      description: item.description,
      type: item.type,
    };
  }
  return null;
}).filter(result => result && !clos.some(c => c.cloName === result.cloName));

const ChapterResults = ChapterData.map(item => {
  const subject = allSubjects.find(s => s.subjectCode === item.subjectCode);
  if (subject) {
    return {
      subject_id: subject.subject_id,
      chapterName: item.chapterName,
      description: item.description,
    };
  }
  return null;
}).filter(result => result && !chapters.some(c => c.chapterName === result.chapterName));

await Promise.all([
  CLOResults.length > 0 && CloModel.bulkCreate(CLOResults),
  ChapterResults.length > 0 && ChapterModel.bulkCreate(ChapterResults),
]);
```

Hàm processSaveTemplateSubject(6/8)

Tiếp theo, mã tạo ra các kết quả cho CLO và chương từ dữ liệu đã được nhập. Đổi

với từng mục trong CLOData, nó tìm kiếm môn học tương ứng trong allSubjects, và nếu tìm thấy, tạo một đối tượng chứa thông tin cần thiết (bao gồm subject\_id, cloName, description, và type). Sau đó, nó lọc các kết quả để chỉ giữ lại những mục chưa tồn tại trong danh sách CLO. Tương tự cho ChapterData, mã tạo ra các đối tượng chương và lọc để chỉ giữ lại các chương chưa có trong danh sách chương. Cuối cùng, mã sử dụng Promise.all để thực hiện đồng thời việc chèn các kết quả CLO và chương vào cơ sở dữ liệu thông qua bulkCreate, chỉ thực hiện nếu có dữ liệu cần thêm.

```
● ● ●

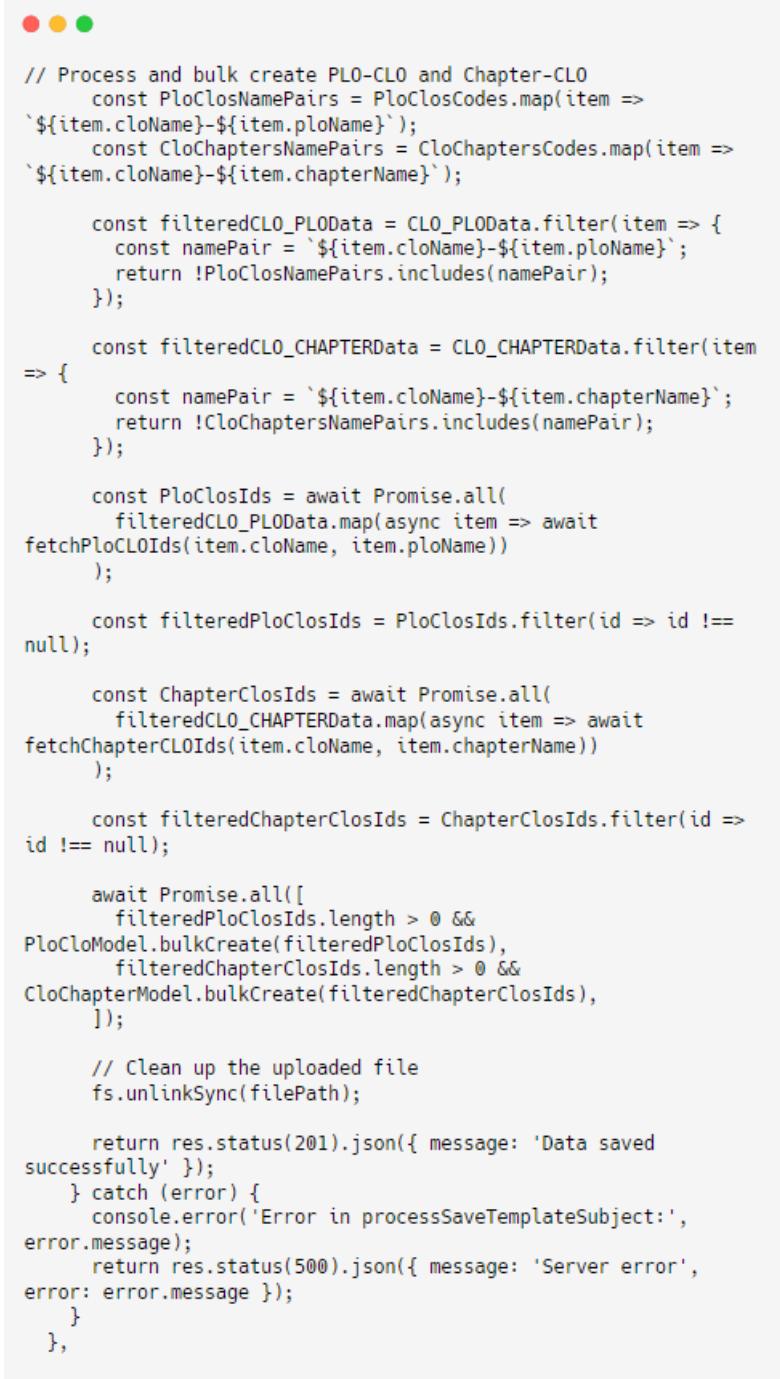
// Fetch and process PLO-CLO and Chapter-CLO IDs
const fetchPloCLOIds = async (cloName, ploName) => {
    try {
        const clo = await CloModel.findOne({ where: { cloName } });
        const plo = await PloModel.findOne({ where: { ploName } });
        if (plo && clo) {
            return { clo_id: clo.clo_id, plo_id: plo.plo_id };
        }
        return null;
    } catch (error) {
        console.error('Error fetching PLO or CLO:', error);
        return null;
    }
};

const fetchChapterCLOIds = async (cloName, chapterName) =>
{
    try {
        const clo = await CloModel.findOne({ where: { cloName } });
        const chapter = await ChapterModel.findOne({ where: { chapterName } });
        if (chapter && clo) {
            return { clo_id: clo.clo_id, chapter_id: chapter.chapter_id };
        }
        return null;
    } catch (error) {
        console.error('Error fetching Chapter or CLO:', error);
        return null;
    }
};
```

#### Hàm processSaveTemplateSubject(7/8)

Đoạn mã tiếp theo định nghĩa hai hàm bắt đồng bộ để lấy ID của PLO-CLO và Chapter-CLO từ cơ sở dữ liệu. Hàm fetchPloCLOIds nhận vào tên CLO và tên PLO, tìm kiếm các đối tượng tương ứng trong cơ sở dữ liệu. Nếu tìm thấy cả hai, hàm trả về một đối

tượng chứa ID của CLO và PLO. Nếu không tìm thấy, hoặc xảy ra lỗi, nó trả về null và ghi lại thông báo lỗi. Tương tự, hàm fetchChapterCLOIDs hoạt động tương tự nhưng cho CLO và Chapter.



```
// Process and bulk create PLO-CLO and Chapter-CLO
const PloClosNamePairs = PloClosCodes.map(item =>
` ${item.cloName}-${item.ploName}`);
const CloChaptersNamePairs = CloChaptersCodes.map(item =>
` ${item.cloName}-${item.chapterName}`);

const filteredCLO_PLOData = CLO_PLOData.filter(item => {
  const namePair = ` ${item.cloName}-${item.ploName}`;
  return !PloClosNamePairs.includes(namePair);
});

const filteredCLO_CHAPTERData = CLO_CHAPTERData.filter(item => {
  const namePair = ` ${item.cloName}-${item.chapterName}`;
  return !CloChaptersNamePairs.includes(namePair);
});

const PloClosIds = await Promise.all(
  filteredCLO_PLOData.map(async item => await
fetchPloCLOIDs(item.cloName, item.ploName))
);

const filteredPloClosIds = PloClosIds.filter(id => id !==
null);

const ChapterClosIds = await Promise.all(
  filteredCLO_CHAPTERData.map(async item => await
fetchChapterCLOIDs(item.cloName, item.chapterName))
);

const filteredChapterClosIds = ChapterClosIds.filter(id =>
id !== null);

await Promise.all([
  filteredPloClosIds.length > 0 &&
PloCloModel.bulkCreate(filteredPloClosIds),
  filteredChapterClosIds.length > 0 &&
CloChapterModel.bulkCreate(filteredChapterClosIds),
]);

// Clean up the uploaded file
fs.unlinkSync(filePath);

return res.status(201).json({ message: 'Data saved
successfully' });
} catch (error) {
  console.error('Error in processSaveTemplateSubject:', error.message);
  return res.status(500).json({ message: 'Server error',
error: error.message });
}
},
```

### Hàm processSaveTemplateSubject(8/8)

Cuối cùng, đoạn mã xử lý và tạo hàng loạt các mối quan hệ PLO-CLO và Chapter-CLO trong cơ sở dữ liệu. Nó tạo ra các cặp tên từ PloClosCodes và CloChaptersCodes để sử dụng trong việc lọc. Sau đó, dữ liệu được lọc để loại bỏ những mục đã tồn tại trong cơ

sở dữ liệu dựa trên các cặp tên này. Nếu có dữ liệu cần lưu, chúng sẽ được thêm vào cơ sở dữ liệu thông qua bulkCreate. Cuối cùng, tệp đã tải lên sẽ được xóa và hàm trả về phản hồi thành công. Nếu có lỗi xảy ra trong quá trình này, thông báo lỗi sẽ được ghi lại và trả về cho client.

## PHỤ LỤC 11. API của đánh giá tổng hợp (meta-assessment)

### 11.1. Lấy tất cả thông tin đánh giá tổng hợp

API Endpoint: GET /api/admin/meta-assessments

**Tham số truy vấn:**

**teacher\_id:** Tham số này cho phép lọc các môn học dựa trên ID của giảng viên. Khi tham số này được cung cấp, API sẽ chỉ trả về các đánh giá liên quan đến giảng viên có ID tương ứng.

**isDelete:** Tham số này được sử dụng để lọc các đánh giá dựa trên trạng thái bị xóa hay không.

**generalDescription:** Tham số này cho phép lọc đánh giá theo mô tả chung. VD 1 lớp được tạo đánh giá tồn tại một mô tả chung.

```
index: async (req, res) => {
  try {
    const { isDelete, teacher_id, generalDescription } =
      req.query;
```

Hàm này là một hàm bất đồng bộ nhận các tham số từ truy vấn (req.query), bao gồm isDelete, teacher\_id, và generalDescription.

```
● ● ●
```

```
if (teacher_id && generalDescription) {
    const assessments = await MetaAssessmentModel.findAll({
        where: [
            { teacher_id: parseInt(teacher_id), generalDescription: generalDescription, isDelete: isDelete === 'true' }
        ],
        include: [
            {
                model: CourseModel,
                attributes: ['course_id', 'courseCode', 'courseName']
            },
            {
                model: StudentModel,
                attributes: ['student_id', 'studentCode', 'name', 'class_id'],
                include: [
                    {
                        model: ClassModel,
                        attributes: ['classNameShort']
                    }
                ]
            },
            {
                model: RubricModel,
                attributes: ['rubric_id', 'rubricName']
            }
        ]
    });
    return res.status(200).json(assessments);
}
```

Đoạn mã này thực hiện truy vấn để lấy thông tin đánh giá dựa trên teacher\_id và generalDescription. Khi cả hai tham số được cung cấp, hàm truy vấn MetaAssessmentModel để tìm các đánh giá khớp yêu cầu, đồng thời lọc các đánh giá đã xóa bằng điều kiện isDelete. Truy vấn cũng bao gồm thông tin từ các mô hình liên quan như CourseModel, StudentModel, và RubricModel. Nếu thành công, hàm trả về mã trạng thái 200 cùng danh sách đánh giá dưới dạng JSON.

```

else if (generalDescription) {
    const metaAssessments = await MetaAssessmentModel.findAll({
        where: [
            { generalDescription: generalDescription },
            { isDelete: isDelete === 'true' }
        ]
    });

    if (metaAssessments.length === 0) {
        return res.status(200).json([]);
    }
    const metaAssessmentIds = metaAssessments.map(meta =>
        meta.meta_assessment_id);
    const assessments = await AssessmentModel.findAll({
        where: [
            { meta_assessment_id: metaAssessmentIds }
        ],
        include: [
            {
                model: TeacherModel
            }
        ]
    });
    const updatedMetaAssessments = metaAssessments.map(meta => {
        meta.dataValues.assessments =
            assessments.filter(assessment => assessment.meta_assessment_id ===
                meta.meta_assessment_id);
        return meta;
    });
    return res.status(200).json({
        meta_assessment_ids: metaAssessmentIds,
        assessments: assessments,
        meta_assessment: updatedMetaAssessments
    });
}

```

Đoạn mã này xử lý trường hợp chỉ có generalDescription được cung cấp. Nếu có giá trị cho tham số này, hàm truy vấn MetaAssessmentModel để tìm các đánh giá phù hợp và lọc theo điều kiện isDelete. Nếu không tìm thấy đánh giá nào, hàm trả về mảng rỗng với mã trạng thái 200. Nếu có kết quả, hàm trích xuất meta\_assessment\_id từ các đánh giá và truy vấn AssessmentModel để lấy các đánh giá tương ứng. Cuối cùng, hàm ánh xạ các đánh giá vào từng đối tượng metaAssessment, và trả về mã trạng thái 200 cùng với danh sách meta\_assessment\_id, các đánh giá và đối tượng đã cập nhật, cung cấp cái nhìn tổng quát về các đánh giá liên quan đến mô tả được cung cấp.

```

else if (teacher_id) {
    const teacherId = parseTnt(teacher_id);

```

*Điều kiện tồn tại teacher\_id*

Nếu có teacher\_id trong yêu cầu, đoạn mã sẽ chuyển đổi giá trị này thành số nguyên để sử dụng cho các truy vấn sau.

```

if (teacher_id)

const assessments = await MetaAssessmentModel.findAll({
  where: {
    teacher_id: teacherId,
    isDelete: isDelete === 'true'
  },
  attributes: [
    'course_id',
    'generalDescription',
    [Sequelize.fn('COUNT', Sequelize.col('meta_assessment_id')),
     'assessmentCount'],
    [Sequelize.fn('COUNT', Sequelize.fn('DISTINCT',
      Sequelize.col('student_id'))), 'studentCount'],
    [Sequelize.fn('SUM', Sequelize.literal('CASE WHEN FinalScore
      = 0 THEN 1 ELSE 0 END')), 'zeroScoreCount']
  ],
  group: ['course_id', 'generalDescription'],
  include: [
    {model: CourseModel,
     attributes: ['courseCode', 'courseName']}
  ]
});

```

### *Truy vấn và lấy số lượng điểm*

Truy vấn MetaAssessmentModel để lấy danh sách các meta-assessments cho giảng viên cụ thể, bao gồm thông tin tổng quan như course\_id, mô tả chung, số lượng đánh giá, số lượng sinh viên và số lượng điểm 0. Kết quả được nhóm theo course\_id và generalDescription.

```

if (teacher_id)

if (assessments.length === 0) {
  return res.status(404).json({ message: 'No meta-assessments
  found for this user' });
}

```

### *Kiểm tra truy vấn*

Nếu không tìm thấy bất kỳ meta-assessment nào, trả về mã trạng thái 404 và thông báo rằng không có đánh giá nào cho người dùng.

```

if (teacher_id)

const result = await Promise.all(assessments.map(async assessment =>
{
  let status;
  const assessmentCount =
  parseInt(assessment.dataValues.assessmentCount);
  const zeroScoreCount =
  parseInt(assessment.dataValues.zeroScoreCount);

  if (assessmentCount === 0) {
    status = 100;
  } else {
    status = ((assessmentCount - zeroScoreCount) /
  assessmentCount) * 100;
  }
  status = Math.round(status);
})

```

*Tính toán lượng % sinh viên được chấm điểm (bên trong result)*

Tính toán trạng thái của mỗi meta-assessment dựa trên số lượng đánh giá và số lượng điểm 0. Nếu không có đánh giá nào, trạng thái sẽ là 100 ngược lại, tính toán tỷ lệ thành công và làm tròn giá trị.

```
● ● ● if (teacher_id)

const foundAssessment = await MetaAssessmentModel.findOne({
    where: {
        generalDescription: assessment.generalDescription,
        isDelete: isDelete === 'true'
    },
    attributes: ["meta_assessment_id", "teacher_id", "rubric_id",
    "course_id", "generalDescription", "date", "place", "isDelete",
    "createdAt"],
    include: [
        {
            model: RubricModel,
            include: [
                {
                    model: SubjectModel,
                }
            ],
            model: CourseModel,
        }
    ]
});
```

*Tìm một meta-assessment (bên trong result)*

Tìm kiếm thông tin chi tiết về meta-assessment dựa trên mô tả chung, bao gồm thông tin như meta\_assessment\_id, teacher\_id, và các thông tin liên quan đến rubric và khóa học.

```
● ● ● if (teacher_id)

const Assessment = await AssessmentModel.findAll({
    where: {
        meta_assessment_id: foundAssessment.meta_assessment_id,
        isDelete: isDelete === 'true'
    },
    include: [
        {
            model: TeacherModel,
            where: {
                isDelete: isDelete === 'true'
            }
        }
    ]
});
```

*Tìm kiếm assessment hay giảng viên được mời dựa vào id đánh giá (bên trong result)*

Truy vấn để lấy tất cả các đánh giá liên quan đến meta-assessment đã tìm thấy, bao gồm thông tin về giảng viên.

```
● ● ● if (teacher_id)

let statusAllot = true;
if (Assessment.length === 0) {
    statusAllot = false;
}
```

*Kiểm tra giảng viên được mời vào đánh giá chưa (bên trong result)*

Kiểm tra xem có đánh giá nào liên quan không. Nếu không có, đặt statusAllot thành false.

```

if (teacher_id)

if (foundAssessment && foundAssessment.Rubric) {
    const rubricItems = await RubricsItemModel.findAll({
        where: [
            { rubric_id: foundAssessment.Rubric.rubric_id, isDelete: isDelete === 'true' },
            { include: [
                { model: CloModel, attributes: ['clo_id', 'cloName', 'description'], where: { isDelete: isDelete === 'true' } },
                { model: ChapterModel, attributes: ['chapter_id', 'chapterName', 'description'], where: { isDelete: isDelete === 'true' } },
                { model: PloModel, attributes: ['plo_id', 'ploName', 'description'], where: { isDelete: isDelete === 'true' } }
            ] }
        ],
        foundAssessment.Rubric.dataValues.rubricItems = rubricItems;
    });
}

```

*Lấy bảng tiêu chí để phục vụ PDF phiếu chấm (bên trong result)*

Nếu tìm thấy rubric, lấy tất cả các mục của rubric cùng với các thông tin liên quan đến CLO, chapter và PLO, và gán vào thuộc tính của rubric.

```

if (teacher_id)

return {
    teacher_id: foundAssessment.teacher_id,
    course_id: assessment.course_id,
    generalDescription: assessment.generalDescription,
    course: `${assessment.course.courseCode} - ${assessment.course.courseName}`,
    courseCode: assessment.course.courseCode,
    courseName: assessment.course.courseName,
    assessmentCount: parseInt(assessment.dataValues.assessmentCount),
    studentCount: parseInt(assessment.dataValues.studentCount),
    zeroScoreCount: parseInt(assessment.dataValues.zeroScoreCount),
    status: status,
    Assessment: Assessment || [],
    statusAllot: statusAllot,
    metaAssessment: foundAssessment,
    createdAt: foundAssessment ? foundAssessment.createdAt : null,
    isDelete: foundAssessment ? foundAssessment.isDelete : null
};

```

*Trả về kết quả result*

Trả về một đối tượng chứa tất cả thông tin đã thu thập được cho từng meta-assessment, bao gồm thông tin tổng quan và chi tiết.

```

if (teacher_id)

const ViewMetaAssessments = await MetaAssessmentModel.findAll({
    where: {
        teacher_id: teacherId,
        isDelete: isDelete === 'true'
    },
    include: [
        {
            model: CourseModel,
            attributes: ['courseCode', 'courseName']
        },
        {
            model: StudentModel
        }
    ]
});

```

*Lấy tất cả thông tin đánh giá của giảng viên chính*

Truy vấn để lấy tất cả các meta-assessments khác của giảng viên, bao gồm thông tin về khóa học và sinh viên.

```

if (teacher_id)

const ResultViewMetaAssessments = await
Promise.all(ViewMetaAssessments.map(async assessment => {
    const Assessment = await AssessmentModel.findAll({
        where: {
            meta_assessment_id: assessment.meta_assessment_id,
            isDelete: isDelete === 'true'
        },
        include: [
            {
                model: TeacherModel,
                where: {
                    isDelete: isDelete === 'true'
                }
            }
        ]
    });
    assessment.dataValues.Assessment = Assessment;
    return assessment;
}));
result.forEach(resItem => {
    resItem.ViewMetaAssessments =
    ResultViewMetaAssessments.filter(viewMetaAssessment =>
        viewMetaAssessment.generalDescription ===
    resItem.generalDescription
    );
});
return res.status(200).json(result);

```

*Map tìm lấy ra điểm GVGD và GV được mời*

Lặp qua các meta-assessments để lấy danh sách đánh giá và gán chúng vào thuộc tính Assessment của từng đối tượng. Sau đó, kết nối kết quả để gán các đánh giá tương ứng vào từng meta-assessment.

Trả về kết quả cuối cùng với mã trạng thái 200 và dữ liệu dưới dạng JSON.

## 11.2. Tạo đánh giá tổng hợp

API Endpoint: POST /api/admin/meta-assessment

```

create: async (req, res) => {
  const { data } = req.body;
  try {
    const newMetaAssessment = await MetaAssessmentModel.create(data);
    res.status(201).json({
      message: 'Tạo MetaAssessment thành công',
      meta_assessment_id: newMetaAssessment.meta_assessment_id,
      data: newMetaAssessment
    });
  } catch (error) {
    console.error('Lỗi khi tạo meta assessment mới:', error);
    res.status(400).json({ message: 'Yêu cầu không hợp lệ' });
  }
}

```

*Hàm create tương tự các API khác*

### 11.3. Lấy thông tin 1 đánh giá tổng hợp

API Endpoint: GET /api/admin/meta-assessment/:id

```

show: async (req, res) => {
  try {
    const { id } = req.params;
    const metaAssessment = await MetaAssessmentModel.findById(id);
    if (metaAssessment) {
      res.json(metaAssessment);
    } else {
      res.status(404).json({ message: 'Meta assessment không tìm thấy' });
    }
  } catch (error) {
    console.error('Lỗi khi lấy meta assessment:', error);
    res.status(500).json({ message: 'Lỗi server' });
  }
}

```

*Hàm show*

### 11.3. Cập nhật đề tài

API Endpoint: PATCH /api/admin/meta-assessment/:id

```

updateDescription: async (req, res) => {
  try {
    const { id } = req.params;
    const { description } = req.body;

    // Cập nhật description trong MetaAssessmentModel
    await MetaAssessmentModel.update(
      { description: description },
      { where: { meta_assessment_id: id } }
    );
    res.status(200).json({ message: 'MetaAssessment updated successfully' });
  } catch (error) {
    console.error('Error updating MetaAssessment:', error);
    res.status(500).json({ message: 'Error updating MetaAssessment', error });
  }
}

```

### *Hàm updateDescription*

Hàm này cập nhật mô tả của một meta-assessment trong cơ sở dữ liệu. Đầu tiên, hàm lấy ID từ tham số URL và mô tả mới từ thân yêu cầu. Sau đó, nó sử dụng phương thức update của MetaAssessmentModel để cập nhật trường mô tả. Nếu thành công, hàm trả về mã trạng thái 200; nếu có lỗi, hàm ghi lại lỗi và trả về mã trạng thái 500.

#### **11.4. Xóa mềm bằng mô tả chung**

API Endpoint: PATCH /api/admin/meta-assessments/softDeleteByGeneralDescription

```
● ● ●

toggleSoftDeleteByGeneralDescription: async (req, res) => {
    try {
        const { GeneralDescriptions, isDelete } = req.body;
        if (!Array.isArray(GeneralDescriptions) ||
            GeneralDescriptions.length === 0) {
            return res.status(400).json({ message: 'GeneralDescription array is required and cannot be empty' });
        }

        const metaAssessments = await MetaAssessmentModel.findAll({
            where: {
                generalDescription: GeneralDescriptions
            }
        });
        console.log('Found metaAssessments:', metaAssessments);

        if (metaAssessments.length === 0) {
            return res.status(404).json({ message: 'No metaAssessments found for the provided GeneralDescriptions' });
        }

        const updated = await Promise.all(metaAssessments.map(async
            (meta_assessment) => {
                if (isDelete === null) {
                    return { meta_assessment_id:
                        meta_assessment.meta_assessment_id, isDelete:
                        meta_assessment.isDelete };
                } else {
                    const updatedIsDeleted = isDelete !== undefined ? isDelete :
                        !meta_assessment.isDelete;
                    await meta_assessment.update({ isDelete: updatedIsDeleted });
                    return { meta_assessment_id:
                        meta_assessment.meta_assessment_id, isDelete: updatedIsDeleted };
                }
            }));
        res.status(200).json({ message: 'Processed isDelete status',
            updated });
    } catch (error) {
        console.error('Error toggling assessment delete statuses:', error);
        res.status(500).json({ message: 'Server error' });
    }
}.
```

### *Hàm toggleSoftDeleteByGeneralDescription*

Hàm này chuyển đổi trạng thái “xóa mềm” của các meta-assessment dựa trên mô tả chung. Đầu tiên, hàm kiểm tra tính hợp lệ của mảng GeneralDescriptions; nếu không hợp lệ, nó trả về mã trạng thái 400. Sau đó, hàm truy vấn cơ sở dữ liệu để tìm các meta-assessment tương ứng; nếu không tìm thấy, trả về mã trạng thái 404. Nếu có kết quả, hàm sử dụng Promise.all để xử lý từng meta-assessment. Nếu isDelete là null, hàm không thay đổi gì; nếu có giá trị, nó sẽ cập nhật trạng thái isDelete. Cuối cùng, hàm trả về mã trạng thái 200 cùng danh sách các meta-assessment đã cập nhật. Nếu xảy ra lỗi, hàm ghi lại và trả về mã trạng thái 500.

#### **11.4. Cập nhật bằng mô tả chung**

API Endpoint: PATCH /api/admin/meta-assessments/updateByGeneralDescription

```

updateByGeneralDescription: async (req, res) => {
    try {
        const { GeneralDescription, updateData } = req.body;

        console.log(updateData);
        let metaAssessments = await MetaAssessmentModel.findAll({
            where: { generalDescription: GeneralDescription } });
        if (metaAssessments.length === 0) {
            return res.status(404).json({ message: "No metaAssessments found" });
        }
        if (updateData.generalDescription) {
            const existingAssessment = await
MetaAssessmentModel.findOne({ where: { generalDescription:
updateData.generalDescription } });

            if (existingAssessment) {
                return res.status(400).json({ message: "An assessment with
the new generalDescription already exists" });
            }
        }

        const updatedAssessments = await
Promise.all(metaAssessments.map(async (meta_assessment) => {
            if (updateData.rubric_id !== undefined) {
                meta_assessment.rubric_id = updateData.rubric_id;
            }
            if (updateData.course_id !== undefined) {
                meta_assessment.course_id = updateData.course_id;
            }
            if (updateData.generalDescription !== undefined) {
                meta_assessment.generalDescription =
updateData.generalDescription;
            }
            if (updateData.place !== undefined) {
                meta_assessment.place = updateData.place;
            }
            if (updateData.date !== undefined) {
                meta_assessment.date = updateData.date;
            }

            await meta_assessment.save();
            return meta_assessment;
        }));
        res.status(200).json(updatedAssessments);
    } catch (error) {
        console.error("Error updating metaAssessments:", error);
        res.status(500).json({ message: "Error updating
metaAssessments", error });
    }
}.

```

### *Hàm updateByGeneralDescription*

Hàm này cập nhật thông tin của các meta-assessment theo mô tả chung. Nó nhận dữ liệu từ yêu cầu và truy vấn cơ sở dữ liệu để tìm các meta-assessment tương ứng; nếu không có, trả về mã trạng thái 404. Hàm kiểm tra xem mô tả mới trong updateData có tồn tại không; nếu có, trả về mã trạng thái 400. Nếu tất cả điều kiện được thỏa mãn, hàm cập nhật các thuộc tính như rubric\_id, course\_id, generalDescription, place, và date, rồi trả về mã trạng thái 200 cùng danh sách các meta-assessment đã cập nhật. Nếu xảy ra lỗi, mã trạng thái 500 được trả về.

## 11.5. Xóa bằng mô tả chung

API Endpoint: DELETE /api/admin/meta-assessments/updateByGeneralDescription

```
deleteByGeneralDescription: async (req, res) => {
    try {
        const { GeneralDescriptions } = req.body;
        console.log(GeneralDescriptions);
        if (!Array.isArray(GeneralDescriptions) ||
GeneralDescriptions.length === 0) {
            return res.status(400).json({ message: 'Descriptions array
is required and cannot be empty' });
        }
        const metaAssessments = await
MetaAssessmentModel.findAll({where: { generalDescription:
GeneralDescriptions}
});
        if (metaAssessments.length === 0) {
            return res.status(404).json({ message: 'No assessments found
for the provided GeneralDescriptions' });
        }
        const metaAssessmentIds = metaAssessments.map(meta =>
meta.meta_assessment_id);
        const assessments = await AssessmentModel.findAll({ where: {
meta_assessment_id: metaAssessmentIds
}});
        const assessmentIds = assessments.map(assessment =>
assessment.assessment_id);
        await AssessmentItemModel.destroy({ where: {assessment_id:
assessmentIds }
});
        await AssessmentModel.destroy({ where: { meta_assessment_id:
metaAssessmentIds }
});
        const deletedCount = await MetaAssessmentModel.destroy({where:
{meta_assessment_id: metaAssessmentIds}
});
        res.status(200).json({ message: 'Successfully deleted
assessments, assessment items, and meta assessments', deletedCount
});
    } catch (error) {
        console.error('Error deleting assessments:', error);
        res.status(500).json({ message: 'Server error' });
    }
}
```

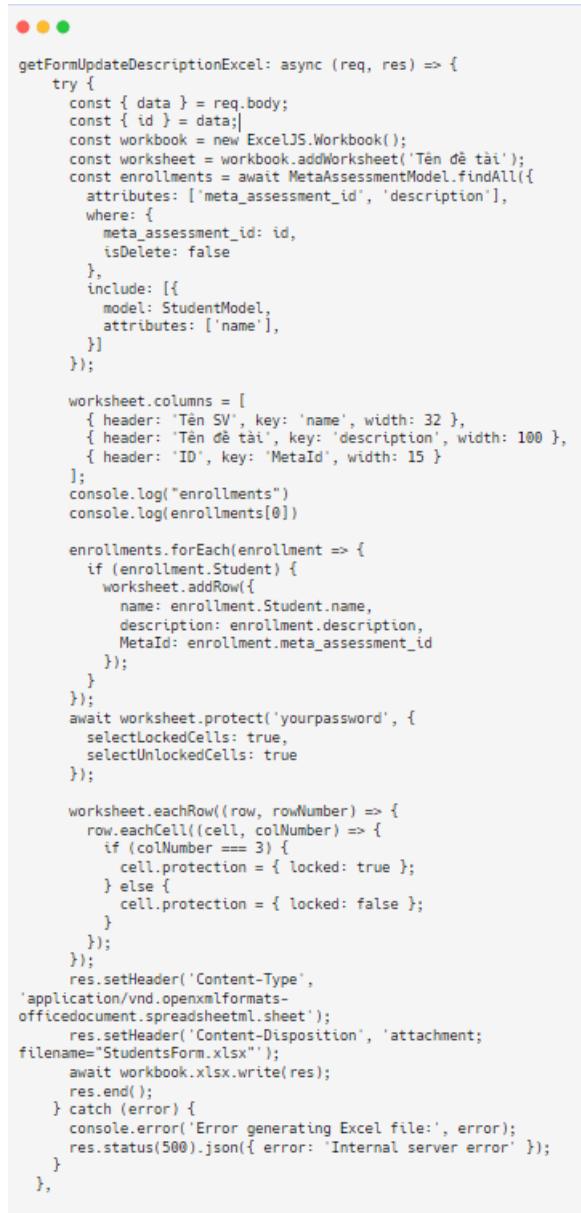
*Hàm deleteByGeneralDescription*

Hàm này thực hiện việc xóa các meta-assessment dựa trên mô tả chung (General Descriptions). Đầu tiên, hàm kiểm tra tính hợp lệ của đầu vào; nếu không phải là mảng hoặc rỗng, nó trả về mã trạng thái 400. Tiếp theo, hàm tìm các meta-assessment phù hợp; nếu không tìm thấy, mã trạng thái 404 được trả về.

Hàm lấy danh sách các meta\_assessment\_id và truy vấn các đánh giá (Assessment) liên quan. Sau khi xác định assessment\_id, nó xóa các mục đánh giá (AssessmentItem) và các đánh giá (Assessment). Cuối cùng, hàm xóa các meta-assessment và trả về mã trạng thái 200 cùng thông báo thành công, bao gồm số lượng bản ghi đã bị xóa. Nếu có lỗi xảy ra, hàm sẽ ghi lại lỗi và trả về mã trạng thái 500.

## 11.6. Nhận mẫu cập nhật đề tài bằng Excel

API Endpoint: POST /api/admin/meta-assessment/templates/data



```
getFormUpdateDescriptionExcel: async (req, res) => {
    try {
        const { data } = req.body;
        const { id } = data;
        const workbook = new ExcelJS.Workbook();
        const worksheet = workbook.addWorksheet('Tên đề tài');
        const enrollments = await MetaAssessmentModel.findAll({
            attributes: ['meta_assessment_id', 'description'],
            where: {
                meta_assessment_id: id,
                isDelete: false
            },
            include: [
                { model: StudentModel, attributes: ['name'] }
            ]
        });

        worksheet.columns = [
            { header: 'Tên SV', key: 'name', width: 32 },
            { header: 'Tên đề tài', key: 'description', width: 100 },
            { header: 'ID', key: 'MetaId', width: 15 }
        ];
        console.log("enrollments")
        console.log(enrollments[0])

        enrollments.forEach(enrollment => {
            if (enrollment.Student) {
                worksheet.addRow({
                    name: enrollment.Student.name,
                    description: enrollment.description,
                    MetaId: enrollment.meta_assessment_id
                });
            }
        });
        await worksheet.protect('yourpassword', {
            selectLockedCells: true,
            selectUnlockedCells: true
        });

        worksheet.eachRow((row, pageNumber) => {
            row.eachCell((cell, colNumber) => {
                if (colNumber === 3) {
                    cell.protection = { locked: true };
                } else {
                    cell.protection = { locked: false };
                }
            });
        });
        res.setHeader('Content-Type',
            'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet');
        res.setHeader('Content-Disposition', 'attachment;
filename="StudentsForm.xlsx"');
        await workbook.xlsx.write(res);
        res.end();
    } catch (error) {
        console.error('Error generating Excel file:', error);
        res.status(500).json({ error: 'Internal server error' });
    }
},
```

Hàm *getFormUpdateDescriptionExcel*

Hàm này tạo và cung cấp file Excel chứa thông tin về sinh viên và đề tài của họ từ cơ sở dữ liệu. Đầu tiên, hàm nhận id của các bản ghi đánh giá (MetaAssessment) từ yêu cầu của người dùng và truy vấn để lấy thông tin như mã đánh giá (meta\_assessment\_id), mô tả đề tài, và tên sinh viên.

File Excel được tạo với các cột tiêu đề ‘Tên SV’, ‘Tên đề tài’, và ‘ID’, trong đó cột ‘ID’ được bảo vệ bằng mật khẩu để ngăn chỉnh sửa, trong khi các cột khác có thể chỉnh

sửa. Sau khi điền dữ liệu và bảo vệ bảng tính, hàm thiết lập tiêu đề HTTP để gửi file Excel cho người dùng. Nếu có lỗi xảy ra, hàm sẽ trả về phản hồi lỗi với mã trạng thái HTTP 500 và thông báo cụ thể.

## 11.7. Cập nhật đề tài đánh giá bằng Excel

```
● ● ●

updateDescriptionFromExcel: async (req, res) => {
    if (!req.files || !req.files.length) {
        return res.status(400).json({ message: 'No file uploaded.' });
    }

    try {
        const uploadDirectory = path.join(__dirname, '../uploads');
        const filename = req.files[0].filename;
        const filePath = path.join(uploadDirectory, filename);

        const workbook = new ExcelJS.Workbook();
        await workbook.xlsx.readFile(filePath);
        const worksheet = workbook.getWorksheet('Tên đề tài');

        const Updates = [];

        worksheet.eachRow((row, rowNum) => {
            if (rowNum > 1) {
                const Data = {
                    description: row.getCell(2).value,
                    meta_assessment_id: row.getCell(3).value
                };
                Updates.push(Data);
            }
        });

        fs.unlinkSync(filePath);

        await Promise.all(Updates.map(async (data) => {
            const [affectedRows] = await MetaAssessmentModel.update(
                { description: data.description },
                { where: { meta_assessment_id: data.meta_assessment_id } }
            );

            if (affectedRows === 0) {
                console.warn(`No MetaAssessment found with ID ${data.meta_assessment_id} for update`);
            }
        }));
        res.status(200).json({ message: 'description updated successfully' });
    } catch (error) {
        console.error(`Error updating description from Excel file: ${error}`);
        res.status(500).json({ error: 'Internal server error' });
    }
},
```

*Hàm updateDescriptionFromExcel*

Hàm này cập nhật mô tả của các đề tài từ file Excel do người dùng tải lên. Khi nhận yêu cầu, hàm kiểm tra sự tồn tại của file; nếu không có, nó trả về mã lỗi 400.

Sau khi xác nhận file tồn tại, hàm tải file lên thư mục tạm thời và đọc dữ liệu từ bảng tính có tên “Tên đề tài”. Nó duyệt qua từng hàng để lấy thông tin về mô tả (description) và ID của meta-assessment (meta\_assessment\_id), lưu chúng vào mảng Updates. Sau khi thu thập dữ liệu, hàm xóa file Excel để giải phóng dung lượng và cập nhật các bản ghi trong cơ sở dữ liệu bằng ID từ file. Nếu không tìm thấy bản ghi nào tương ứng, nó ghi lại cảnh báo trong console.

Cuối cùng, hàm trả về thông báo thành công “description updated successfully” nếu không có lỗi xảy ra; nếu có lỗi, hàm sẽ trả về mã 500 cùng thông báo lỗi chi tiết.

## PHỤ LỤC 12. API lần đánh giá (assessment)

### 12.1. Kiểm tra giảng viên có tồn tại trong đánh giá không (quan trọng)

API Endpoint: GET /api/admin/assessment/checkTeacher

**Tham số truy vấn:**

**teacher\_id:** Tham số này cho phép lọc các môn học dựa trên ID của giảng viên. Khi tham số này được cung cấp, API sẽ chỉ trả về các đánh giá liên quan đến giảng viên có ID tương ứng.

**meta\_assessment\_id:** Mã đánh giá tổng quát.

```
checkTeacherInAssessment: async (req, res) => {
  const { teacher_id, meta_assessment_id } = req.query;
  console.log(teacher_id + ':', meta_assessment_id);
  try {
    const assessment = await AssessmentModel.findOne({
      where: {
        teacher_id: teacher_id,
        meta_assessment_id: meta_assessment_id,
        isDelete: false
      }
    });
    console.log("assessment" + ':', assessment);

    if (assessment) {
      return res.status(200).json({ exists: true });
    } else {
      return res.status(200).json({ exists: false });
    }
  } catch (error) {
    console.error("Error checking teacher in assessment:", error.message);
    return res.status(500).json({ error: "Internal Server Error" });
  }
},
```

*Hàm checkTeacherInAssessment*

Hàm này được thiết kế để kiểm tra sự tồn tại của một giảng viên trong một đánh giá cụ thể dựa trên teacher\_id và meta\_assessment\_id. Hàm thực hiện truy vấn đến mô hình AssessmentModel nhằm xác định liệu có bản ghi nào tồn tại với các thông số đã chỉ định và điều kiện isDelete là false. Kết quả của truy vấn sẽ trả về thông tin tồn tại của giảng viên trong đánh giá thông qua trường exists trong phản hồi JSON. Nếu có lỗi xảy ra trong quá trình truy vấn, hàm sẽ ghi lại lỗi và trả về mã lỗi 500 cùng thông điệp tương ứng. Hàm này có vai trò quan trọng trong việc quản lý và theo dõi sự phân công của giảng viên trong hệ thống đánh giá.

## 12.2. Nhận lần đánh giá theo điều kiện

API Endpoint: GET /api/admin/assessment

**Tham số truy vấn:**

**teacher\_id:** Tham số này cho phép lọc các môn học dựa trên ID của giảng viên. Khi tham số này được cung cấp, API sẽ chỉ trả về các đánh giá liên quan đến giảng viên có ID tương ứng.

**isDelete:** Trạng thái đã xóa mềm

**generalDescription:** Mô tả chung của đánh giá tổng quát

```
● ● ●  
getAssessments: async (req, res) => {  
    try {  
        const { isDelete, teacher_id, generalDescription } =  
            req.query;
```

*Hàm getAssessments*

Hàm này thực hiện chức năng truy vấn và trả về danh sách các đánh giá trong hệ thống. Hàm này hỗ trợ nhiều điều kiện lọc dựa trên các tham số isDelete, teacher\_id, và generalDescription từ yêu cầu. Đặc biệt, hàm tích hợp việc truy xuất thông tin chi tiết từ các mô hình liên quan như MetaAssessmentModel, CourseModel, Student Model, và RubricModel.

```
if (teacher_id && generalDescription) {
  const assessments = await AssessmentModel.findAll({
    where: {
      teacher_id: parseInt(teacher_id),
      isDelete: isDelete === 'true'
    },
    include: [
      {
        model: MetaAssessmentModel,
        where: { isDelete: false },
        include: [
          {
            model: CourseModel, attributes: ['courseCode', 'courseName']
          },
          {
            model: StudentModel,
            attributes: ['studentCode', 'name', 'class_id'],
            include: [
              {
                model: ClassModel,
                attributes: ['classNameShort']
              }
            ],
            {
              model: RubricModel,
              where: { isDelete: false },
              include: [{ model: SubjectModel }]
            }
          ],
          {
            model: TeacherModel,
            where: { isDelete: false },
          }
        ]
      });
}
```

#### *Điều kiện teacher\_id và generalDescription*

Khi cả teacher\_id và generalDescription được cung cấp, hàm sẽ bắt đầu quá trình truy vấn từ AssessmentModel để lấy tất cả các đánh giá liên quan đến giảng viên cụ thể.

```
if(teacher_id && generalDescription)

for (const assessment of assessments) {
  const rubricId = assessment.MetaAssessment.Rubric?.rubric_id;

  if (!rubricId) continue; // Skip if rubric_id is not found
```

#### *Lập qua từng đánh giá (1/5)*

Sau khi có danh sách các đánh giá, hàm sẽ kiểm tra từng đánh giá để xác định xem có rubric\_id không. Nếu không có, hàm sẽ bỏ qua đánh giá đó.

```
for (const assessment of assessments)

// Fetch rubric items for the current rubric
const rubricItems = await RubricItemModel.findAll({
  where: {
    rubric_id: rubricId,
    isDelete: false
  },
  include: [
    {
      model: CloModel,
      attributes: ['clo_id', 'cloName', 'description']
    },
    {
      model: ChapterModel,
      attributes: ['chapter_id', 'chapterName', 'description']
    },
    {
      model: PloModel,
      attributes: ['plo_id', 'ploName', 'description']
    }
  ]
});|
```

*Lập qua từng đánh giá (2/5)*

Tiếp theo, hàm sẽ truy xuất các mục rubric liên quan đến rubricId.

```
for (const assessment of assessments)

const rubricsItemIds = rubricItems.map(item =>
  item.rubricsItem_id);
```

*Lập qua từng đánh giá (3/5)*

Sau khi có danh sách các mục rubric, hàm sẽ lấy tất cả các ID.

```
for (const assessment of assessments)

const assessmentItems = await AssessmentItemModel.findAll({
  where: {
    rubricsItem_id: rubricsItemIds,
    assessment_id: assessment.assessment_id
  }
});|
```

*Lập qua từng đánh giá (4/5)*

Hàm sẽ tìm các mục đánh giá liên quan đến các ID mục rubric đã lấy được.

```
for (const assessment of assessments)

    // Attach assessment items to each rubric item
    rubricItems.forEach(rubricItem => {
        rubricItem.dataValues.AssessmentItems =
            assessmentItems.filter(
                assessmentItem => assessmentItem.rubricsItem_id ===
                    rubricItem.rubricsItem_id
            );
    });

    // Merge rubric items into the assessment object

    assessment.dataValues.MetaAssessment.Rubric.dataValues.RubricIte
ms = rubricItems;
}
```

### Lập qua từng đánh giá (5/5)

Cuối cùng, hàm sẽ gán các mục đánh giá vào từng mục rubric để tạo cấu trúc dữ liệu hoàn chỉnh.

```
if(teacher_id && generalDescription)

const metaAssessments = await MetaAssessmentModel.findAll({
    where: {
        generalDescription: generalDescription,
        isDelete: isDelete === 'true'
    },
    include: [
        {
            model: CourseModel,
            attributes: ['course_id', 'courseCode', 'courseName']
        },
        {
            model: StudentModel,
            attributes: ['student_id', 'studentCode', 'name',
            'class_id'],
            include: [
                {
                    model: ClassModel,
                    attributes: ['classNameShort']
                }
            ]
        },
        {
            model: RubricModel,
            attributes: ['rubric_id', 'rubricName']
        }
    ]
});
```

### Hàm tìm kiếm dựa vào mô tả chung

Hàm sẽ tìm tất cả các meta-assessments dựa trên generalDescription và thuộc tính isDelete. Điều này giúp lọc các meta-assessments.

```
● ● ● if (teacher_id && generalDescription)

const metaAssessmentIds = assessments.map(assessment =>
  assessment.meta_assessment_id);
const filteredMetaAssessments =
  metaAssessments.filter(metaAssessment =>
    metaAssessmentIds.includes(metaAssessment.meta_assessment_id)
);
```

### *Lọc meta\_assessment\_id*

hàm sẽ lọc các meta-assessments để chỉ lấy những cái có meta\_assessment\_id nằm trong danh sách AssessmentIdMetas đã được lấy ở đoạn trước.

```
● ● ● if (teacher_id && generalDescription)

const result = filteredMetaAssessments.map(metaAssessment => {
  const associatedAssessment = assessments.find(assessment =>
    assessment.dataValues.meta_assessment_id ===
    metaAssessment.dataValues.meta_assessment_id
  );
  return {
    ...metaAssessment.dataValues,
    assessment: associatedAssessment ?
      associatedAssessment.dataValues : null
  };
});
```

### *Kết hợp thông tin*

Hàm sẽ kết hợp thông tin đánh giá với các meta-assessments đã lọc. Điều này giúp trả về cấu trúc dữ liệu hoàn chỉnh hơn cho từng meta-assessment.

```
● ● ● if (teacher_id && generalDescription)

return res.status(200).json(result);
```

### *Trả kết quả*

Hàm trả về kết quả với mã trạng thái 200 nếu mọi thứ diễn ra suôn sẻ. Kết quả này sẽ bao gồm danh sách các meta-assessments cùng với các thông tin đánh giá liên quan.

```
● ● ●

} else if (teacher_id) {
    const teacherId = parseInt(teacher_id);
    const assessments = await AssessmentModel.findAll({
        where: [
            { teacher_id: teacherId },
            { isDelete: isDelete === 'true' }
        ],
        include: [
            {
                model: TeacherModel
            },
            {
                model: MetaAssessmentModel,
                include: [
                    {
                        model: TeacherModel
                    }
                ]
            }
        ]
    });

    if (assessments.length === 0) {
        return res.status(404).json({ message: 'No assessments found for this user' });
    }
    return res.status(200).json(assessments);
}
```

### *Điều kiện tồn tại teacher\_id*

Khi có teacher\_id trong yêu cầu, hàm sẽ chuyển đổi giá trị này thành số nguyên và thực hiện truy vấn đến AssessmentModel để tìm kiếm tất cả các đánh giá liên quan đến giảng viên cụ thể, đồng thời lọc ra những đánh giá không bị xóa. Nếu không có đánh giá nào được tìm thấy, hàm sẽ trả về mã trạng thái 404 cùng với thông báo “No assessments found for this user”. Ngược lại, nếu có ít nhất một đánh giá, hàm sẽ trả về danh sách các đánh giá đó với mã trạng thái 200.

```

    } else if (generalDescription) {
        const metaAssessment = await
MetaAssessmentModel.findOne({
    where: {
        generalDescription: generalDescription,
        isDelete: isDelete === 'true'
    }
});

if (metaAssessment) {
    const assessments = await AssessmentModel.findAll({
        where: {
            meta_assessment_id:
metaAssessment.meta_assessment_id
        }
    });
    const teacherIds = assessments.map(assessment =>
assessment.teacher_id);
    res.json({
        assessments,
        teacherIds
    });
} else {
    res.status(404).json({ message: 'assessment không tìm
thấy' });
}
} else {
    const assessments = await AssessmentModel.findAll();
    return res.status(200).json(assessments);
}
} catch (error) {
    console.error('Error fetching assessments:', error);
    res.status(500).json({ message: 'Server error' });
}
},

```

*Điều kiện tồn tại generalDescription và không có điều kiện nào thỏa*

Khi có generalDescription trong yêu cầu, hàm sẽ tìm kiếm một bản ghi Meta Assessment dựa trên mô tả tổng quát và trạng thái không bị xóa. Nếu tìm thấy bản ghi này, hàm sẽ truy vấn các đánh giá liên quan đến meta\_assessment\_id của nó. Sau đó, hàm sẽ tạo một danh sách các teacher\_id từ các đánh giá đã tìm thấy và trả về cả danh sách đánh giá cùng với danh sách teacherIds. Nếu không tìm thấy MetaAssessment, hàm sẽ trả về mã trạng thái 404 kèm thông báo “assessment không tìm thấy”. Nếu không có điều kiện nào được thỏa mãn, hàm sẽ truy vấn tất cả các đánh giá có sẵn và trả về với mã trạng thái 200.

### 12.3. Cập nhật điểm khi giảng viên hoàn thành đánh giá

API Endpoint: PATCH /api/admin/assessment/:id/totalScore

```
updateTotalScore: async (req, res) => {
  try {
    const { id } = req.params;
    const { data } = req.body;

    const updatedProgram = await AssessmentModel.update(data, {
      where: { assessment_id: id } });

    if (updatedProgram[0] === 0) {
      return res.status(404).json({ message: 'Assessment not found' });
    }

    res.json(updatedProgram);
  } catch (error) {
    console.error('Lỗi cập nhật assessments:', error);
    res.status(500).json({ message: 'Lỗi server' });
  }
},
```

#### *Hàm updateTotalScore*

Hàm này được sử dụng để cập nhật điểm số tổng của một bài đánh giá dựa trên assessment\_id được cung cấp trong tham số URL. Khi nhận được yêu cầu, hàm sẽ trích xuất id từ tham số và data từ nội dung yêu cầu. Tiến hành cập nhật dữ liệu trong cơ sở dữ liệu thông qua mô hình AssessmentModel. Nếu không tìm thấy bài đánh giá tương ứng với assessment\_id, hàm sẽ trả về mã trạng thái 404 cùng thông điệp “Assessment not found”. Ngược lại, nếu cập nhật thành công, hàm sẽ trả về dữ liệu đã cập nhật với mã trạng thái 200. Trong trường hợp xảy ra lỗi trong quá trình thực hiện, hàm sẽ ghi lại lỗi và trả về mã trạng thái 500 với thông điệp “Lỗi server”.

#### **12.4. Lấy tất cả thông tin liên quan đến lần đánh giá dựa vào id**

API Endpoint: GET /api/admin/assessment/:id/items

```
GetitemsByID: async (req, res) => {
  try {
    const { id } = req.params;
    // Step 1: Fetch the assessment with basic associations
    const assessments = await AssessmentModel.findOne({
      where: { assessment_id: id, isDelete: false },
      include: [
        {
          model: MetaAssessmentModel,
          where: { isDelete: false },
          include: [
            { model: CourseModel, attributes: ['courseCode', 'courseName'] },
            {
              model: StudentModel,
              attributes: ['studentCode', 'name', 'class_id'],
              include: [{ model: ClassModel, attributes: ['classNameShort'] }]
            },
            { model: RubricModel, where: { isDelete: false } }
          ],
          { model: TeacherModel, where: { isDelete: false } }
        ],
      ];
    });
  }
};
```

### *Hàm GetitemsByID*

Hàm này bắt đầu với việc lấy assessment\_id từ tham số của yêu cầu HTTP. Sau đó, nó thực hiện truy vấn để tìm bài đánh giá dựa trên assessment\_id và các điều kiện khác như isDelete: false để đảm bảo rằng chỉ các bài đánh giá chưa bị xóa mới được truy xuất. Dữ liệu bài đánh giá bao gồm các thông tin liên quan đến khóa học, sinh viên, lớp học, rubric và giảng viên.

```
if (!assessments || !assessments.MetaAssessment.Rubric) {
  return res.status(404).json({ message: 'Assessment or Rubric not found' });
}
```

### *Kiểm tra sự tồn tại*

Kiểm tra sự tồn tại của dữ liệu rubric. Nếu không tìm thấy bài đánh giá hoặc rubric, hệ thống trả về phản hồi lỗi 404 với thông báo “Assessment or Rubric not found”. Điều này đảm bảo rằng chỉ các bài đánh giá hợp lệ mới được xử lý tiếp.

```
const rubricItems = await RubricItemModel.findAll({
    where: { rubric_id: assessments.MetaAssessment.Rubric.rubric_id, isDelete: false },
    include: [
        { model: CloModel, attributes: ['clo_id', 'cloName', 'description'] },
        { model: ChapterModel, attributes: ['chapter_id', 'chapterName', 'description'] },
        { model: PloModel, attributes: ['plo_id', 'ploName', 'description'] }
    ]
});
```

### *Tìm kiếm tiêu chí đánh giá*

Khi bài đánh giá và rubric đã được xác nhận là hợp lệ, hệ thống tiếp tục tìm kiếm các mục rubric (rubric items) liên quan bằng cách truy vấn trong mô hình RubricItemModel. Các mục rubric bao gồm thông tin về các mục tiêu học tập của khóa học (CLO), chương (Chapter), và mục tiêu học tập của chương trình (PLO). Dữ liệu được truy xuất nhằm cung cấp thông tin chi tiết hơn về các tiêu chí đánh giá.

```
const rubricsItemIds = rubricItems.map(item =>
item.rubricsItem_id);

const assessmentItems = await AssessmentItemModel.findAll({
    where: { rubricsItem_id: rubricsItemIds, assessment_id: id }
});

rubricItems.forEach(rubricItem => {
    rubricItem.dataValues.AssessmentItems = assessmentItems.filter(
        assessmentItem => assessmentItem.rubricsItem_id ===
rubricItem.rubricsItem_id
    );
});

assessments.dataValues.MetaAssessment.Rubric.dataValues.RubricIte
ms = rubricItems;
```

### *Tìm kiếm điểm chi tiết, lập và tích hợp thông tin*

Hệ thống tìm kiếm các mục đánh giá liên quan đến từng mục rubric bằng cách truy vấn từ AssessmentItemModel dựa trên danh sách rubricsItem\_id. Quá trình này liên kết các mục đánh giá với từng mục rubric, giúp mỗi mục rubric chứa thông tin đánh giá cụ thể hơn. Sau khi hoàn tất việc liên kết, đoạn mã tích hợp tất cả thông tin này vào bài đánh giá chính (assessments).

```
    res.status(200).json(assessments);
  } catch (error) {
    console.error('Error fetching assessments:', error);
    res.status(500).json({ message: 'Server error' });
  }
},
```

### Trả dữ liệu về

Cuối cùng, tất cả thông tin chi tiết về bài đánh giá, bao gồm các tiêu chí rubric và các mục đánh giá, được gửi về cho người dùng dưới dạng định dạng JSON. Hệ thống cũng trả về mã trạng thái 200 để xác nhận rằng yêu cầu đã được thực hiện thành công.

## 12.5. Xóa giảng viên khỏi lần đánh giá

API Endpoint: DELETE /api/admin/assessment/teacher/:teacherId

```
deleteByTeacherId: async (req, res) => {
  const { teacherId } = req.params;
  try {
    const assessments = await AssessmentModel.findAll({ where:
    { teacher_id: teacherId } });
    const assessmentIds = assessments.map(assessment =>
    assessment.assessment_id);

    if (assessmentIds.length > 0) {
      await AssessmentItemModel.destroy({
        where: {
          assessment_id: assessmentIds
        }
      });
      const result = await AssessmentModel.destroy({
        where: { teacher_id: teacherId }
      });

      if (result > 0) {
        res.status(200).json({ message: `Successfully deleted
        ${result} assessments.` });
      } else {
        res.status(404).json({ message: 'No assessments found
        for the given teacher_id.' });
      }
    } else {
      res.status(404).json({ message: 'No assessments found for
        the given teacher_id.' });
    }
  } catch (error) {
    console.error("Error deleting assessments by teacher_id:",
    error);
    res.status(500).json({ message: 'An error occurred while
    deleting assessments.' });
  }
},
```

Hàm deleteByTeacherId

Đoạn mã này thực hiện việc xóa các bài đánh giá dựa trên teacherId được truyền từ

tham số yêu cầu. Đầu tiên, nó tìm tất cả các bài đánh giá liên quan đến giảng viên thông qua mô hình AssessmentModel và tạo ra một mảng assessmentIds chứa ID của các bài đánh giá đó. Nếu mảng này không rỗng, hệ thống sẽ xóa tất cả các mục đánh giá tương ứng từ mô hình AssessmentItemModel và sau đó xóa các bài đánh giá từ mô hình AssessmentModel.

Nếu xóa thành công, hệ thống trả về mã trạng thái 200 và thông báo số lượng bài đánh giá đã bị xóa. Nếu không tìm thấy bài đánh giá nào, mã trạng thái 404 sẽ được trả về kèm thông báo thích hợp. Trong trường hợp xảy ra lỗi, mã trạng thái 500 và thông báo lỗi sẽ được gửi đến người dùng

## 12.6. Nhận thông tin 1 đánh giá

API Endpoint: GET /api/admin/assessment/:id

```
● ● ●

getByID: async (req, res) => {
  try {
    const { id } = req.params;
    const assessment = await AssessmentModel.findOne({
      where: { assessment_id: id },
      include: [
        {
          model: MetaAssessmentModel,
          where: { isDelete: false },
          include: [
            { model: RubricModel, where: { isDelete: false } },
            { model: SubjectModel, where: { isDelete: false } },
            { model: CourseModel, where: { isDelete: false } },
            { model: StudentModel, where: { isDelete: false } },
          ],
        },
      ],
    });
    if (!assessment) return res.status(404).json({ message: 'Assessment not found' });
    res.json(assessment);
  } catch (error) {
    console.error('Error fetching assessment:', error);
    res.status(500).json({ message: 'Server error' });
  }
},
```

Hàm `getByID`

## 12.7. Tạo mới lần đánh giá (hay phân công giảng viên chấm)

API Endpoint: POST /api/admin/assessment

```
create: async (req, res) => {
    try {
        const { data } = req.body;
        console.log(data);
        const Assessment = await AssessmentModel.create(data);
        res.json(Assessment);
    } catch (error) {
        console.error('Lỗi tạo Assessment:', error);
        res.status(500).json({ message: 'Lỗi server' });
    }
},
```

Hàm create

## 12.8. Cập nhật đánh giá

API Endpoint: PUT /api/admin/assessment/:id

```
update: async (req, res) => {
    try {
        const { id } = req.params;
        const { data } = req.body;
        const updatedProgram = await AssessmentModel.update(data, {
            where: { assessment_id: id } });
        if (updatedProgram[0] === 0) {
            return res.status(404).json({ message: 'assessments not found' });
        }
        res.json(updatedProgram);
    } catch (error) {
        console.error('Lỗi cập nhật assessments:', error);
        res.status(500).json({ message: 'Lỗi server' });
    }
},
```

Hàm update

## 12.9. Xóa 1 đánh giá

API Endpoint: DELETE /api/admin/assessment/:id

```
delete: async (req, res) => {
  try {
    const { id } = req.params;
    await AssessmentModel.destroy({ where: { assessment_id: id } });
    res.json({ message: 'Xóa assessments thành công' });
  } catch (error) {
    console.error('Lỗi xóa assessments:', error);
    res.status(500).json({ message: 'Lỗi server' });
  }
},
```

Hàm delete

## 12.10. Xóa nhiều đánh giá

API Endpoint: DELETE /api/admin/assessments/multiple

**Tham số truy vấn:**

**assessment\_id:** mảng id đánh giá cần xóa.

```
deleteMultiple: async (req, res) => {
  const { assessment_id } = req.query;
  try {
    const assessmentIds = assessment_id.map(id =>
      parseInt(id));
    await AssessmentModel.destroy({ where: { assessment_id: assessment_id } });
    res.status(200).json({ message: 'Xóa nhiều assessment thành công' });
  } catch (error) {
    console.error('Lỗi khi xóa nhiều assessment:', error);
    res.status(500).json({ message: 'Lỗi server nội bộ' });
  }
},
```

Hàm deleteMultiple

## PHỤ LỤC 13. API của bảng tiêu chí (Rubric)

### 13.1. Lấy tất cả thông tin liên quan đến bản tiêu chí dựa vào id

API Endpoint: GET /api/admin/rubric/:id/items

**Tham số truy vấn:**

**isDelete:** Tham số này được sử dụng để lọc bảng tiêu chí dựa trên trạng thái bị xóa hay không.

**include\_clos:** Dạng true false khi là true sẽ lấy ra thêm mảng dữ liệu CLO và gắn kèm vào dữ liệu gửi về.

```

● ● ●

getItemsByRubricId: async (req, res) => {
  try {
    const { id } = req.params;
    const { isDelete, include_clos } = req.query;

    const rubric = await RubricModel.findOne({
      where: { rubric_id: id },
      include: [{ model: SubjectModel, attributes: ['subject_id', 'subjectCode', 'subjectName'] }]
    });

    if (!rubric) return res.status(404).json({ message: 'Rubric not found' });

    const rubricItems = await RubricItemModel.findAll({
      where: { rubric_id: rubric.rubric_id, isDelete: isDelete === 'true' },
      include: [
        { model: CloModel, attributes: ['clo_id', 'cloName', 'description'] },
        { model: ChapterModel, attributes: ['chapter_id', 'chapterName', 'description'] },
        { model: PloModel, attributes: ['plo_id', 'ploName', 'description'] }
      ]
    });

    for (let item of rubricItems) {
      item.dataValues.chapters = await ChapterModel.findAll({
        where: { chapter_id: (await MapCloChapterModel.findAll({ where: { clo_id: item.clo_id } })).map(chapter => chapter.chapter_id) }
      });

      item.dataValues.plos = await PloModel.findAll({
        where: { plo_id: (await PloCloModel.findAll({ where: { clo_id: item.clo_id } })).map(plo => plo.plo_id) }
      });
    }

    if (include_clos === 'true') {
      rubric.dataValues.CloData = await CloModel.findAll({ where: { subject_id: rubric.subject_id, isDelete: isDelete === 'true' } });
    }

    rubric.dataValues.rubricItems = rubricItems;
    rubric.dataValues.CloData = await CloModel.findAll({ where: { subject_id: rubric.subject_id } });

    return res.json({ rubric });
  } catch (error) {
    console.error('Error getting rubric items:', error);
    res.status(500).json({ message: 'Internal server error' });
  }
}

```

### *Hàm getItemsByRubricId*

Đoạn mã này thực hiện việc lấy thông tin chi tiết về một rubric dựa trên rubric\_id được cung cấp trong tham số yêu cầu. Đầu tiên, nó tìm kiếm rubric trong mô hình RubricModel và bao gồm thông tin từ mô hình SubjectModel. Nếu không tìm thấy rubric, hệ thống trả về mã trạng thái 404 cùng với thông báo “Rubric not found”.

Sau đó, đoạn mã tiếp tục truy vấn các mục rubric (rubric items) từ mô hình RubricItemModel, lọc dựa trên rubric\_id và giá trị của isDelete (nếu có). Các mục rubric này bao gồm thông tin về các mục tiêu học tập của khóa học (CLO), chương (Chapter), và mục tiêu học tập của chương trình (PLO).

Tiếp theo, cho mỗi mục rubric, hệ thống lấy thông tin chi tiết về các chương liên quan và các PLO liên quan thông qua các truy vấn bổ sung. Nếu tham số include\_clos được đặt là ‘true’, hệ thống sẽ truy vấn thêm thông tin về các CLO liên quan đến subject\_id của rubric.

Cuối cùng, toàn bộ thông tin về rubric, các mục rubric và CLO sẽ được tích hợp vào đối tượng rubric và trả về cho người dùng dưới dạng JSON. Trong trường hợp có lỗi xảy ra, mã trạng thái 500 và thông báo lỗi sẽ được gửi về.

### 13.2. Lấy bảng tiêu chí và kiểm tra đủ điểm của giảng viên đã tạo

API Endpoint: GET /api/admin/rubrics/checkScore

#### Tham số truy vấn:

**teacher\_id:** Tham số này cho phép lọc các bảng tiêu chí dựa trên ID của giảng viên. Khi tham số này được cung cấp, API sẽ chỉ trả về các đánh giá liên quan đến giảng viên có ID tương ứng.

**isDelete:** Tham số này được sử dụng để lọc bảng tiêu chí dựa trên trạng thái bị xóa hay không.

```
● ● ●

getRubricsForCheckScore: async (req, res) => {
  try {
    const { teacher_id, isDelete } = req.query;
    const whereCondition = { teacher_id, isDelete: isDelete !== undefined ? isDelete === 'true' : false };

    const rubrics = await RubricModel.findAll({ where:
      whereCondition });
    const rubricIds = rubrics.map(rubric => rubric.rubric_id);

    const results = await RubricItemModel.findAll({
      attributes: ['rubric_id', [Sequelize.fn('SUM',
        Sequelize.col('maxScore')), 'total_score']],
      where: { rubric_id: rubricIds, isDelete:
        whereCondition.isDelete },
      group: ['rubric_id']
    });

    const rubricScores = results.map(result => ({
      rubric_id: result.rubric_id,
      total_score: result.dataValues.total_score
    }));

    rubrics.forEach(rubric => {
      rubric.dataValues.RubricItem = rubricScores.filter(item =>
        item.rubric_id === rubric.rubric_id);
    });

    res.json({ rubric: rubrics });
  } catch (error) {
    console.error('Error getting all rubrics:', error);
    res.status(500).json({ message: 'Internal server error' });
  }
},
```

*Hàm getRubricsForCheckScore phục vụ tạo lần đánh giá tổng hợp*

Hàm này lấy các tham số teacher\_id và isDelete từ yêu cầu và xây dựng điều kiện tìm kiếm cho mô hình RubricModel. Tiếp theo, hàm truy vấn tất cả các rubric phù hợp với điều kiện đã định và trích xuất rubric\_id để sử dụng trong truy vấn tiếp theo.

Hàm sau đó truy vấn các mục rubric từ RubricItemModel và tính tổng điểm tối đa (maxScore) cho từng rubric bằng cách sử dụng hàm SUM, với kết quả được nhóm theo rubric\_id. Sau khi có tổng điểm, hàm tích hợp thông tin này vào từng đối tượng rubric thông qua thuộc tính RubricItem. Cuối cùng, hàm trả về danh sách rubric kèm theo điểm số dưới dạng JSON. Nếu xảy ra lỗi, hàm ghi lại và trả về mã trạng thái 500 với thông báo lỗi.

### 13.3. Lấy tất cả thông tin bản tiêu chí

API Endpoint: GET /api/admin/rubrics

```
● ● ●  
index: async (req, res) => {  
    try {  
        const rubrics = await RubricModel.findAll();  
        res.json(rubrics);  
    } catch (error) {  
        console.error('Error getting all rubrics:', error);  
        res.status(500).json({ message: 'Internal server error' });  
    }  
},
```

*Hàm index*

### 13.4. Tạo mới một bảng tiêu chí

API Endpoint: POST /api/admin/rubric

```
● ● ●  
create: async (req, res) => {  
    try {  
        const { data } = req.body;  
        const newrubric = await RubricModel.create(data);  
        res.status(201).json(newrubric);  
    } catch (error) {  
        console.error('Error creating rubric:', error);  
        res.status(500).json({ message: 'Server error' });  
    }  
},
```

*Hàm create*

### 13.5. Lấy thông tin 1 tiêu chí

API Endpoint: GET /api/admin/rubric/:id

```
getByID: async (req, res) => {
  try {
    const { id } = req.params;
    const rubric = await RubricModel.findOne({ where: { rubric_id: id } });
    if (!rubric) {
      return res.status(404).json({ message: 'rubric not found' });
    }
    res.status(200).json(rubric);
  } catch (error) {
    console.error('Error finding rubric:', error);
    res.status(500).json({ message: 'Server error' });
  }
}
```

*Hàm getByID*

### 13.6. Cập nhật 1 tiêu chí

API Endpoint: PUT /api/admin/rubric/:id

```
update: async (req, res) => {
  try {
    const { id } = req.params;
    const { data } = req.body;
    const rubric = await RubricModel.findOne({ where: { rubric_id: id } });
    if (!rubric) {
      return res.status(404).json({ message: 'rubric not found' });
    }
    const updatedrubric = await RubricModel.update(data, { where: { rubric_id: id } });
    res.json({ message: `Successfully updated rubric with ID: ${id}` });
  } catch (error) {
    console.error('Error updating rubric:', error);
    res.status(500).json({ message: 'Server error' });
  }
};
```

### *Hàm update*

### **13.7. Lấy tất cả thông tin bảng tiêu chí chưa ẩn**

API Endpoint: GET /api/admin/rubrics/isDelete/false

```
● ● ●

isDeleteTofalse: async (req, res) => {
  try {
    const rubrics = await RubricModel.findAll({ where: { isDelete: false } });
    if (!rubrics) {
      return res.status(404).json({ message: 'No rubrics found' });
    }
    res.json(rubrics);
  } catch (error) {
    console.error('Error finding rubrics with isDelete false:', error);
    res.status(500).json({ message: 'Server error' });
  }
}.
```

Hàm isDeleteTofalse

### 13.7. Lấy tất cả thông tin bảng tiêu chí đã ẩn

API Endpoint: GET /api/admin/rubrics/isDelete/true

```
● ● ●

isDeleteTotrue: async (req, res) => {
  try {
    const rubrics = await RubricModel.findAll({ where: { isDelete: true } });
    if (!rubrics) {
      return res.status(404).json({ message: 'No rubrics found' });
    }
    res.json(rubrics);
  } catch (error) {
    console.error('Error finding rubrics with isDelete true:', error);
    res.status(500).json({ message: 'Server error' });
  }
}.
```

Hàm isDeleteTotrue

### 13.8. Xóa mềm 1 bảng tiêu chí

API Endpoint: PUT /api/admin/rubric/:id/softDelete

```

toggleSoftDeleteById: async (req, res) => {
    try {
        const { id } = req.params;
        const rubric = await RubricModel.findOne({ where: { rubric_id: id } });
        if (!rubric) {
            return res.status(404).json({ message: 'rubric not found' });
        }
        const updatedIsDeleted = !rubric.isDelete;
        await RubricModel.update({ isDelete: updatedIsDeleted }, { where: { rubric_id: id } });

        res.status(200).json({ message: `Toggled isDelete status to ${updatedIsDeleted}` });

    } catch (error) {
        console.error('Error toggling PLO delete statuses:', error);
        res.status(500).json({ message: 'Server error' });
    }
},

```

*Hàm toggleSoftDeleteById*

### 13.9. Xóa mềm nhiều bằng tiêu chí

API Endpoint: PUT /api/admin/rubrics/softDelete

```

softDeleteMultiple: async (req, res) => {
    try {
        const { data } = req.body;
        const { rubric_id } = data;
        if (!Array.isArray(rubric_id) || rubric_id.length === 0) {
            return res.status(400).json({ message: 'No PLO ids provided' });
        }
        const rubrics = await RubricModel.findAll({ where: { rubric_id: rubric_id } });
        if (rubrics.length !== rubric_id.length) {
            return res.status(404).json({ message: 'One or more RubricModels not found' });
        }

        const updated = await Promise.all(rubrics.map(async (rubric) => {
            const updatedIsDeleted = !rubric.isDelete;
            await rubric.update({ isDelete: updatedIsDeleted });
            return { rubric_id: rubric.rubric_id, isDelete: updatedIsDeleted };
        }));

        res.json({ message: 'RubricModel delete statuses toggled', updated });
    } catch (error) {
        console.error('Error toggling RubricModel delete status:', error);
        res.status(500).json({ message: 'Server error' });
    }
},

```

*Hàm softDeleteMultiple*

### 13.10. Xóa 1 bảng tiêu chí

API Endpoint: DELETE /api/admin/rubric/:id

```
● ● ●

delete: async (req, res) => {
  try {
    const { id } = req.params;
    const rubric = await RubricModel.findOne({ where: { rubric_id: id } });
    if (!rubric) {
      return res.status(404).json({ message: 'rubric not found' });
    }

    const rubricItems = await RubricItemModel.findAll({ where: { rubric_id: rubric.rubric_id } });
    for (const RubricItem of rubricItems) {
      await RubricItemModel.destroy({ where: { rubricsItem_id: RubricItem.rubricsItem_id } });
    }
    await RubricModel.destroy({ where: { rubric_id: rubric.rubric_id } });
    res.status(200).json({ message: 'Successfully deleted rubric' });
  } catch (error) {
    console.error('Error deleting rubric:', error);
    res.status(500).json({ message: 'Server error' });
  }
},
```

*Hàm delete*

### 13.11. Xóa 1 bảng tiêu chí

API Endpoint: DELETE /api/admin/rubrics/multiple

```
● ● ●

deleteMultiple: async (req, res) => {
  const { rubric_id } = req.query;
  try {
    const rubricIds = rubric_id.map(id => parseInt(id));
    for (const id of rubricIds) {
      const RubricItems = await RubricItemModel.findAll({ where: { rubric_id: id } });
      for (const RubricItem of RubricItems) {
        await RubricItemModel.destroy({ where: { rubricsItem_id: RubricItem.rubricsItem_id } });
      }
    }

    await RubricModel.destroy({ where: { rubric_id: rubric_id } });
    res.status(200).json({ message: 'Xóa nhiều rubric thành công' });
  } catch (error) {
    console.error('Lỗi khi xóa nhiều rubric:', error);
    res.status(500).json({ message: 'Lỗi server nội bộ' });
  }
},
```

*Hàm deleteMultiple*

## PHỤ LỤC 14. API của tiêu chí (Rubric-item)

### 14.1. Lưu tiêu chí và kiểm tra có vượt quá điểm

API Endpoint: POST /api/admin/rubric-item/checkScore

```
checkScore: async (req, res) => {
  try {
    const { data } = req.body;
    const { rubric_id } = data.data;

    const rubricItems = await RubricItemModel.findAll({
      where: { rubric_id, isDelete: false }
    });

    const length = rubricItems.length;

    const results = length > 0
      ? await RubricItemModel.findAll({
          attributes: ['rubric_id', [Sequelize.fn('SUM', Sequelize.col('maxScore')), 'total_maxScore']],
          where: { rubric_id, isDelete: false }
        })
      : [{ dataValues: { total_maxScore: 0 } }]; // Default value if no items

    const total_maxScore =
      parseFloat(results[0].dataValues.total_maxScore || 0);
    const maxScore = parseFloat(data.maxScore || 0);
    const totalScore = total_maxScore + maxScore;

    if (totalScore <= 10) {
      const newRubric = await RubricItemModel.create(data.data);
      return res.status(201).json({ message: "Rubric item created successfully", data: newRubric });
    } else {
      return res.status(400).json({ message: "Failed to save: Total maxScore exceeds 10" });
    }
  } catch (error) {
    console.error('Error creating rubric item:', error);
    return res.status(500).json({ success: false, message: 'Server error' });
  }
},
```

Hàm checkScore

Hàm này bắt đầu bằng cách lấy dữ liệu từ yêu cầu và trích xuất rubric\_id từ đối tượng data. Tiếp theo, hàm truy vấn các mục rubric liên quan từ mô hình RubricItemModel, chỉ lấy các mục chưa bị xóa (isDelete: false).

Nếu có mục rubric, hàm sẽ tính tổng điểm tối đa (total\_maxScore) bằng cách sử dụng hàm SUM trong truy vấn thứ hai. Nếu không có mục nào, nó sẽ thiết lập giá trị mặc định cho total\_maxScore là 0. Sau đó, hàm tính tổng điểm bằng cách cộng total\_maxScore và maxScore từ yêu cầu.

Nếu tổng điểm không vượt quá 10, hàm sẽ tạo một mục rubric mới và trả về phản

hồi thành công với mã trạng thái 201. Ngược lại, nếu tổng điểm vượt quá 10, hàm sẽ trả về mã trạng thái 400 cùng với thông báo lỗi. Trong trường hợp xảy ra lỗi, hàm ghi lại lỗi và trả về mã trạng thái 500 với thông báo về sự cố trên máy chủ.

## 14.2. Lấy thông tin tất cả tiêu chí

API Endpoint: GET /api/admin/rubric-items

```
● ● ●  
index: async (req, res) => {  
    try {  
        const RubricsItem = await RubricItemModel.findAll();  
        res.status(200).json(RubricsItem);  
    } catch (error) {  
        console.error('Error getting all RubricsItem:', error);  
        res.status(500).json({ message: 'Internal server error' });  
    }  
},
```

Hàm index

## 14.3. Tạo mới 1 tiêu chí

API Endpoint: POST /api/admin/rubric-item

```
● ● ●  
create: async (req, res) => {  
    try {  
        const { data } = req.body;  
        const newrubric = await RubricItemModel.create(data);  
        res.status(201).json(newrubric);  
    } catch (error) {  
        console.error('Error creating rubrics_item:', error);  
        res.status(500).json({ message: 'Server error' });  
    }  
},
```

Hàm create

## 14.4. Lấy thông tin 1 tiêu chí

API Endpoint: GET /api/admin/rubric-item/:id

```
● ● ●  
getByID: async (req, res) => {  
    try {  
        const { id } = req.params;  
        const rubrics_item = await RubricItemModel.findOne({ where: {  
            rubricsitem_id: id } });  
        if (!rubrics_item) {  
            return res.status(404).json({ message: 'rubrics_item not  
            found' });  
        }  
  
        res.status(200).json(rubrics_item);  
    } catch (error) {  
        console.error('Error finding rubrics_item:', error);  
        res.status(500).json({ message: 'Server error' });  
    }  
},
```

Hàm getByID

## 14.5. Cập nhật 1 tiêu chí

API Endpoint: PUT /api/admin/rubric-item/:id

```
● ● ●

update: async (req, res) => {
  try {
    const { id } = req.params;
    const { data } = req.body;
    console.log(data);
    const rubrics_item = await RubricItemModel.findOne({ where: {
      rubricsitem_id: id
    } });
    if (!rubrics_item) {
      return res.status(404).json({ message: 'rubrics_item not found' });
    }
    await RubricItemModel.update(data, { where: { rubricsitem_id: id } });
    res.status(200).json({ message: `Successfully updated rubrics_item with ID: ${id}` });
  } catch (error) {
    console.error('Error updating rubrics_item:', error);
    res.status(500).json({ message: 'Server error' });
  }
}
```

*Hàm update*

## 14.6. Xóa mềm 1 tiêu chí

API Endpoint: PUT /api/admin/rubric-item/:id/softDelete

```
● ● ●

toggleSoftDeleteById: async (req, res) => {
  try {
    const { id } = req.params;
    const RubricItem = await RubricItemModel.findOne({ where: { rubricsitem_id: id } });
    if (!RubricItem) {
      return res.status(404).json({ message: 'RubricItem not found' });
    }

    if (RubricItem.isDelete) {
      const results = await RubricItemModel.findAll({
        attributes: ['rubric_id', [Sequelize.fn('SUM', Sequelize.col('maxScore')), 'total_maxScore']],
        where: { rubric_id: RubricItem.rubric_id, isDelete: false }
      });

      const rubricScores = results.map(result => ({
        total_maxScore: result.dataValues.total_maxScore
      }));

      const totalScore = parseFloat(rubricScores[0].total_maxScore) + parseFloat(RubricItem.maxScore);
      console.log('điem', RubricItem.rubric_id);
      if (totalScore <= 10) {
        const updatedIsDeleted = !RubricItem.isDelete;
        await RubricItemModel.update({ isDelete: updatedIsDeleted }, { where: { rubricsitem_id: id } });
        res.status(200).json({ message: `Toggled isDelete status to ${updatedIsDeleted}` });
      } else {
        res.status(400).json({ success: false, message: "Failed to save: Total maxScore exceeds 10" });
      }
    } else {
      const updatedIsDeleted = !RubricItem.isDelete;
      await RubricItemModel.update({ isDelete: updatedIsDeleted }, { where: { rubricsitem_id: id } });
      res.status(200).json({ message: `Toggled isDelete status to ${updatedIsDeleted}` });
    }
  } catch (error) {
    console.error('Error toggling RubricItemModel delete statuses:', error);
    res.status(500).json({ message: 'Server error' });
  }
}.
```

### *Hàm toggleSoftDeleteById*

Hàm này thực hiện việc chuyển đổi trạng thái xóa mềm của một mục rubric dựa trên rubricsitem\_id được cung cấp. Đầu tiên, hàm tìm kiếm mục rubric trong cơ sở dữ liệu. Nếu không tìm thấy, nó sẽ trả về mã trạng thái 404 cùng thông báo “RubricItem not found”.

Nếu mục rubric tồn tại, hàm sẽ xác định trạng thái xóa mới bằng cách đảo ngược giá trị isDelete. Nếu mục đang được xóa (tức là isDelete là true), hàm sẽ tính tổng điểm tối đa cho các mục rubric khác trong cùng một rubric bằng cách sử dụng hàm SUM. Nếu tổng điểm (bao gồm cả điểm của mục hiện tại) vượt quá 10, hàm sẽ trả về mã trạng thái 400 cùng thông báo lỗi.

Nếu mọi điều kiện đều hợp lệ, hàm sẽ cập nhật trạng thái isDelete của mục rubric và trả về mã trạng thái 200 cùng thông báo xác nhận trạng thái đã được chuyển đổi. Nếu có lỗi xảy ra trong quá trình thực hiện, hàm sẽ ghi lại lỗi và trả về mã trạng thái 500 với thông báo “Server error”.

### **14.7. Xóa mềm nhiều tiêu chí**

API Endpoint: PUT /api/admin/rubric-items/softDelete

```

softDeleteMultiple: async (req, res) => {
  try {
    const { data } = req.body;
    const { rubricsitem_id } = data;

    if (!Array.isArray(rubricsitem_id) || rubricsitem_id.length === 0) {
      return res.status(400).json({ message: 'No RubricItemModel ids provided' });
    }

    const RubricItems = await RubricItemModel.findAll({ where: { rubricsitem_id } });

    if (RubricItems.length !== rubricsitem_id.length) {
      return res.status(404).json({ message: 'One or more RubricItemModels not found' });
    }

    const rubricId = RubricItems[0].rubric_id;
    const anyItemIsDelete = RubricItems.some(item => item.isDelete);

    const totalArrayScore = RubricItems.reduce((total, rubricItem) => total + parseFloat(rubricItem.maxScore), 0);

    if (anyItemIsDelete) {
      const result = await RubricItemModel.findOne({
        attributes: [[Sequelize.fn('SUM', Sequelize.col('maxScore')), 'total_maxScore']],
        where: { rubric_id: rubricId, isDelete: false }
      });

      const currentTotalScore = parseFloat(result.dataValues.total_maxScore || 0);
      const newTotalScore = currentTotalScore + totalArrayScore;

      if (newTotalScore <= 10) {
        const updatedRubricItems = await Promise.all(RubricItems.map(async (RubricItem) => {
          const updatedIsDeleted = !RubricItem.isDelete;
          await RubricItem.update({ isDelete: updatedIsDeleted });
          return { rubricsitem_id: RubricItem.rubricsitem_id, isDelete: updatedIsDeleted };
        }));
        return res.status(200).json({ message: 'RubricItemModel delete statuses toggled', updatedRubricItems });
      } else {
        return res.status(400).json({ success: false, message: "Failed to save: Total maxScore exceeds 10" });
      }
    }

    const updatedRubricItems = await Promise.all(RubricItems.map(async (RubricItem) => {
      const updatedIsDeleted = !RubricItem.isDelete;
      await RubricItem.update({ isDelete: updatedIsDeleted });
      return { rubricsitem_id: RubricItem.rubricsitem_id, isDelete: updatedIsDeleted };
    }));

    return res.status(200).json({ message: 'RubricItemModel delete statuses toggled', updatedRubricItems });
  } catch (error) {
    console.error('Error toggling RubricItemModel delete status:', error);
    return res.status(500).json({ message: 'Server error' });
  }
},

```

### *Hàm softDeleteMultiple*

Hàm này thực hiện việc chuyển đổi trạng thái xóa mềm cho nhiều mục rubric được chỉ định thông qua một mảng rubricsitem\_id. Đầu tiên, hàm kiểm tra xem dữ liệu đầu vào có phải là một mảng và có độ dài lớn hơn 0 hay không. Nếu không, nó sẽ trả về mã trạng thái 400 cùng thông báo “No RubricItemModel ids provided”.

Tiếp theo, hàm tìm kiếm tất cả các mục rubric dựa trên các ID đã cung cấp. Nếu số lượng mục tìm thấy không khớp với số lượng ID đã cung cấp, hàm sẽ trả về mã trạng thái 404 cùng thông báo “One or more RubricItemModels not found”.

Hàm sẽ xác định rubric\_id của các mục rubric và kiểm tra xem có bất kỳ mục nào

đã bị xóa hay không. Nếu có, nó sẽ tính tổng điểm tối đa từ các mục rubric và so sánh với tổng điểm hiện tại trong cùng rubric. Nếu tổng điểm mới không vượt quá 10, hàm sẽ tiến hành cập nhật trạng thái isDelete cho từng mục rubric và trả về mã trạng thái 200 cùng thông báo xác nhận.

Nếu không có mục nào đã bị xóa, hàm sẽ thực hiện cập nhật trạng thái isDelete cho tất cả các mục và trả về kết quả tương tự. Trong trường hợp có lỗi xảy ra trong quá trình thực hiện, hàm sẽ ghi lại lỗi và trả về mã trạng thái 500 với thông báo “Server error”.

#### 14.8. Xóa 1 tiêu chí

API Endpoint: DELETE /api/admin/rubric-item/:id

```
● ● ●

delete: async (req, res) => {
  try {
    const { id } = req.params;
    await RubricItemModel.destroy({ where: { rubricsitem_id: id } });
    res.status(200).json({ message: 'Successfully deleted rubrics_item' });
  } catch (error) {
    console.error('Error deleting rubrics_item:', error);
    res.status(500).json({ message: 'Server error' });
  }
},
```

Hàm delete

### PHỤ LỤC 15. API của mục tiêu CT (PO)

#### 15.1. Lấy thông tin tất cả mục tiêu CT

API Endpoint: GET /api/admin/pos

```
● ● ●

index: async (req, res) => {
  try {
    const pos = await PoModel.findAll({
      include: [
        { model: ProgramModel,
          attributes: ['program_id', 'programName']
        }
      ];
    });
    res.json(pos);
  } catch (error) {
    console.error('Error getting all PoModels:', error);
    res.status(500).json({ message: 'Internal server error' });
  }
},
```

*Hàm index*

## 15.2. Tạo mới mục tiêu CT

API Endpoint: POST /api/admin/po

```
● ○ ●  
create: async (req, res) => {  
    try {  
        const { data } = req.body;  
        const newPoModel = await PoModel.create(data);  
        res.status(201).json({ message: 'PoModel created  
successfully', data: newPoModel });  
    } catch (error) {  
        console.error('Error creating PoModel:', error);  
        res.status(500).json({ message: 'Server error' });  
    }  
},
```

*Hàm create*

## 15.3. Lấy thông tin 1 mục tiêu CT

API Endpoint: GET /api/admin/po/:id

```
● ○ ●  
getByID: async (req, res) => {  
    try {  
        const { id } = req.params;  
        const po = await PoModel.findOne({ where: { po_id: id } });  
        if (!po) {  
            return res.status(404).json({ message: 'PoModel not found' });  
        }  
        res.json(po);  
    } catch (error) {  
        console.error('Error finding PoModel:', error);  
        res.status(500).json({ message: 'Server error' });  
    }  
},
```

*Hàm getByID*

## 15.4. Cập nhật thông tin 1 mục tiêu CT

API Endpoint: PUT /api/admin/po/:id

```
update: async (req, res) => {
    try {
        const { id } = req.params;
        const { data } = req.body;
        const [updated] = await PoModel.update(data, { where: {
            po_id: id
        } });
        if (!updated) {
            return res.status(404).json({ message: 'PoModel not found' });
        }
        res.json({ message: 'PoModel updated successfully' });
    } catch (error) {
        console.error('Error updating PoModel:', error);
        res.status(500).json({ message: 'Server error' });
    }
},
```

*Hàm update*

#### 15.4. Xóa 1 mục tiêu CT

API Endpoint: DELETE /api/admin/po/:id

```
delete: async (req, res) => {
    try {
        const { id } = req.params;
        await PoPloModel.destroy({ where: { po_id: id } });
        const deletedCount = await PoModel.destroy({ where: { po_id: id } });
        if (!deletedCount) {
            return res.status(404).json({ message: 'PoModel not found' });
        }
        res.status(200).json({ message: 'PoModel deleted successfully' });
    } catch (error) {
        console.error('Error deleting PoModel:', error);
        res.status(500).json({ message: 'Server error' });
    }
},
```

*Hàm delete*

#### 15.5. Xóa nhiều mục tiêu CT

API Endpoint: DELETE /api/admin//pos/multiple

```
deleteMultiple: async (req, res) => {
    const { po_id } = req.query;
    try {
        await PoPloModel.destroy({ where: { po_id: po_id } });
        const deletedCount = await PoModel.destroy({ where: {
            po_id: po_id
        } });
        if (!deletedCount) {
            return res.status(404).json({ message: 'PoModel not found' });
        }
        res.json({ message: 'PoModel deleted successfully' });
    } catch (error) {
        console.error('Error deleting PoModel:', error);
        res.status(500).json({ message: 'Server error' });
    }
},
```

*Hàm deleteMultiple*

## 15.6. Lấy tất cả thông tin mục tiêu CT chưa ẩn

API Endpoint: GET /api/admin/pos/isDelete/false

```
isDeleteToFalse: async (req, res) => {
    try {
        const pos = await PoModel.findAll({ where: { isDelete: false } });
        res.json(pos);
    } catch (error) {
        console.error('Error finding active PoModels:', error);
        res.status(500).json({ message: 'Server error' });
    }
},
```

*Hàm isDeleteToFalse*

Hàm này được thiết kế để truy xuất tất cả các đối tượng PoModel chưa bị xóa, tức là có thuộc tính isDelete là false. Nó sử dụng phương thức findAll với điều kiện where: { isDelete: false }.

Nếu truy vấn thành công, hàm trả về danh sách đối tượng dưới dạng JSON với mã trạng thái HTTP 200. Nếu có lỗi xảy ra, hàm ghi lại lỗi vào console và trả về mã trạng thái HTTP 500 cùng với thông báo “Server error”.

## 15.7. Lấy tất cả thông tin mục tiêu CT đã ẩn

API Endpoint: GET /api/admin/pos/isDelete/true

```
isDeleteToTrue: async (req, res) => {
    try {
        const pos = await PoModel.findAll({ where: { isDelete: true } });
        res.json(pos);
    } catch (error) {
        console.error('Error finding deleted PoModels:', error);
        res.status(500).json({ message: 'Server error' });
    }
},|
```

*Hàm isDeleteToTrue*

## 15.8. Xóa mềm 1 mục tiêu CT

API Endpoint: PUT /api/admin/po/:id/softDelete

```
toggleSoftDeleteById: async (req, res) => {
    try {
        const { id } = req.params;
        const po = await PoModel.findOne({ where: { po_id: id } });
        if (!po) {
            return res.status(404).json({ message: 'po not found' });
        }
        const updatedIsDeleted = !po.isDelete;
        await PoModel.update({ isDelete: updatedIsDeleted }, {
            where: { po_id: id } });

        res.status(200).json({ message: `Toggled isDelete status to ${updatedIsDeleted}` });

    } catch (error) {
        console.error('Error toggling PoModel delete statuses:', error);
        res.status(500).json({ message: 'Server error' });
    }
},|
```

*Hàm toggleSoftDeleteById*

Hàm này được thiết kế để quản lý trạng thái xóa mềm (isDelete) của các bản ghi trong mô hình PoModel. Khi một yêu cầu được gửi đến hàm với ID của bản ghi, hàm sẽ

đầu tiên xác thực xem bản ghi đó có tồn tại trong cơ sở dữ liệu hay không. Nếu không tìm thấy bản ghi, hàm sẽ trả về mã lỗi 404 cùng với thông báo rằng bản ghi không tồn tại.

Ngược lại, nếu bản ghi được tìm thấy, hàm sẽ tiến hành đảo ngược giá trị của thuộc tính isDelete (từ true thành false hoặc ngược lại). Sau đó, hàm cập nhật trạng thái mới này vào cơ sở dữ liệu. Nếu quá trình cập nhật diễn ra thành công, hàm sẽ phản hồi với mã trạng thái 200 và thông báo xác nhận rằng trạng thái đã được chuyển đổi.

Trong trường hợp xảy ra bất kỳ lỗi nào, hàm sẽ ghi lại thông tin lỗi vào console và trả về mã trạng thái 500 với thông báo lỗi “Server error”.

### 15.9. Xóa mềm nhiều mục tiêu CT

API Endpoint: PUT /api/admin/pos/softDelete

```
● ● ●

softDeleteMultiple: async (req, res) => {
  try {
    const { data } = req.body;
    const { po_id } = data;
    if (!Array.isArray(po_id) || po_id.length === 0) {
      return res.status(400).json({ message: 'No PoModel ids provided' });
    }

    const pos = await PoModel.findAll({ where: { po_id: po_id } });
    if (pos.length !== po_id.length) {
      return res.status(404).json({ message: 'One or more PoModels not found' });
    }

    const updatedPos = await Promise.all(pos.map(async (po) => {
      const updatedIsDeleted = !po.isDelete;
      await po.update({ isDelete: updatedIsDeleted });
      return { po_id: po.po_id, isDelete: updatedIsDeleted };
    }));

    res.json({ message: 'PoModel delete statuses toggled', updatedPos });
  } catch (error) {
    console.error('Error toggling PoModel delete status:', error);
    res.status(500).json({ message: 'Server error' });
  }
},
```

Hàm softDeleteMultiple

## PHỤ LỤC 16. API của CDR CT (PLO)

### 16.1. Lấy thông tin tất CDR CT

API Endpoint: GET /api/admin/plos

```
index: async (req, res) => {
  try {
    const plos = await PloModel.findAll();
    res.status(200).json(plos);
  } catch (error) {
    console.error('Error getting all PL0s:', error);
    res.status(500).json({ message: 'Internal server error' });
  }
},
```

*Hàm index*

### 16.2. Tạo mới CDR CT

API Endpoint: POST /api/admin/plo

```
create: async (req, res) => {
  try {
    const { data } = req.body;
    if (!data) {
      return res.status(400).json({ message: 'No data provided' });
    }
    const newPL0 = await PloModel.create(data);
    res.status(201).json(newPL0);
  } catch (error) {
    console.error('Error creating PL0:', error);
    res.status(500).json({ message: 'Internal server error' });
  }
},
```

*Hàm create*

### 16.3. Lấy thông tin 1 CDR CT

API Endpoint: GET /api/admin/plo/:id

```
getByID: async (req, res) => {
    try {
        const { id } = req.params;
        if (!id) {
            return res.status(400).json({ message: 'No ID provided' });
        }
        const plo = await PloModel.findOne({ where: { plo_id: id } });
    );
    if (!plo) {
        return res.status(404).json({ message: 'PLO not found' });
    }
    res.status(200).json(plo);
} catch (error) {
    console.error('Error fetching PLO:', error);
    res.status(500).json({ message: 'Internal server error' });
}
},
```

Hàm *getByID*

#### 16.4. Cập nhật thông tin 1 CDR CT

API Endpoint: PUT /api/admin/plo/:id

```
update: async (req, res) => {
    try {
        const { id } = req.params;
        const { data } = req.body;
        if (!id) {
            return res.status(400).json({ message: 'No ID provided' });
        }
        if (!data) {
            return res.status(400).json({ message: 'No data provided' });
        }
        const [updated] = await PloModel.update(data, { where: { plo_id: id } });
        if (!updated) {
            return res.status(404).json({ message: 'PLO not found' });
        }
        res.status(200).json({ message: 'PLO updated successfully' });
    } catch (error) {
        console.error('Error updating PLO:', error);
        res.status(500).json({ message: 'Internal server error' });
    }
},
```

Hàm *update*

#### 16.5. Xóa 1 mục tiêu CT

API Endpoint: DELETE /api/admin/plo/:id



```
delete: async (req, res) => {
  try {
    const { id } = req.params;
    if (!id) {
      return res.status(400).json({ message: 'No ID provided' });
    }
    await PoPloModel.destroy({ where: { plo_id: id } });
    await PloCloModel.destroy({ where: { plo_id: id } });
    const deleted = await PloModel.destroy({ where: { plo_id: id } });
    if (!deleted) {
      return res.status(404).json({ message: 'PL0 not found' });
    }
    res.status(200).json({ message: 'PL0 deleted successfully' });
  } catch (error) {
    console.error('Error deleting PL0:', error);
    res.status(500).json({ message: 'Internal server error' });
  }
},
```

*Hàm delete*

## 16.6. Xóa nhiều mục tiêu CT

API Endpoint: DELETE /api/admin/plos/multiple



```
deleteMultiple: async (req, res) => {
  const { plo_id } = req.query;
  try {
    await PloCloModel.destroy({ where: { plo_id: plo_id } });
    await PoPloModel.destroy({ where: { plo_id: plo_id } });
    const deletedCount = await PloModel.destroy({ where: { plo_id: plo_id } });
    if (!deletedCount) {
      return res.status(404).json({ message: 'PL0 not found' });
    }
    res.json({ message: 'PL0 deleted successfully' });
  } catch (error) {
    console.error('Error deleting PL0:', error);
    res.status(500).json({ message: 'Server error' });
  }
},
```

*Hàm deleteMultiple*

## 16.7. Lấy tất cả thông tin mục tiêu CT chưa ẩn

API Endpoint: GET /api/admin/plos/isDelete/false

```
isDeleteTofalse: async (req, res) => {
    try {
        const plos = await PloModel.findAll({ where: { isDelete: false } });
        res.status(200).json(plos);
    } catch (error) {
        console.error('Error fetching PL0s:', error);
        res.status(500).json({ message: 'Internal server error' });
    }
},
```

Hàm isDeleteTofalse

## 16.8. Lấy tất cả thông tin mục tiêu CT đã ẩn

API Endpoint: GET /api/admin/plos/isDelete>true

```
isDeleteTotrue: async (req, res) => {
    try {
        const plos = await PloModel.findAll({ where: { isDelete: true } });
        res.status(200).json(plos);
    } catch (error) {
        console.error('Error fetching PL0s:', error);
        res.status(500).json({ message: 'Internal server error' });
    }
},
```

Hàm isDeleteTotrue

## 16.9. Xóa mềm 1 mục tiêu CT

API Endpoint: PUT /api/admin/plo/:id/softDelete

```

toggleSoftDeleteById: async (req, res) => {
    try {
        const { id } = req.params;
        const plo = await PloModel.findOne({ where: { plo_id: id } });
    };
    if (!plo) {
        return res.status(404).json({ message: 'plo not found' });
    }
    const updatedIsDeleted = !plo.isDelete;
    await PloModel.update({ isDelete: updatedIsDeleted }, {
        where: { plo_id: id } });
    res.status(200).json({ message: `Toggled isDelete status to ${updatedIsDeleted}` });
}

} catch (error) {
    console.error('Error toggling PLO delete statuses:', error);
    res.status(500).json({ message: 'Server error' });
}
},

```

Hàm toggleSoftDeleteById

## 16.10. Xóa mềm nhiều mục tiêu CT

API Endpoint: PUT /api/admin/plos/softDelete

```

softDeleteMultiple: async (req, res) => {
    try {
        const { data } = req.body;
        const { plo_id } = data;
        if (!Array.isArray(plo_id) || plo_id.length === 0) {
            return res.status(400).json({ message: 'No PLO ids provided' });
        }

        const plos = await PloModel.findAll({ where: { plo_id: plo_id } });
        if (plos.length !== plo_id.length) {
            return res.status(404).json({ message: 'One or more PLOs not found' });
        }

        const updatedPlos = await Promise.all(plos.map(async (plo)
=> {
            const updatedIsDeleted = !plo.isDelete;
            await plo.update({ isDelete: updatedIsDeleted });
            return { plo_id: plo.plo_id, isDelete: updatedIsDeleted
        });
    }));

    res.json({ message: 'PLO delete statuses toggled',
updatedPlos });
    } catch (error) {
        console.error('Error toggling PLO delete status:', error);
        res.status(500).json({ message: 'Server error' });
    }
},

```

Hàm softDeleteMultiple

## PHỤ LỤC 17. API của CDR môn học (CLO)

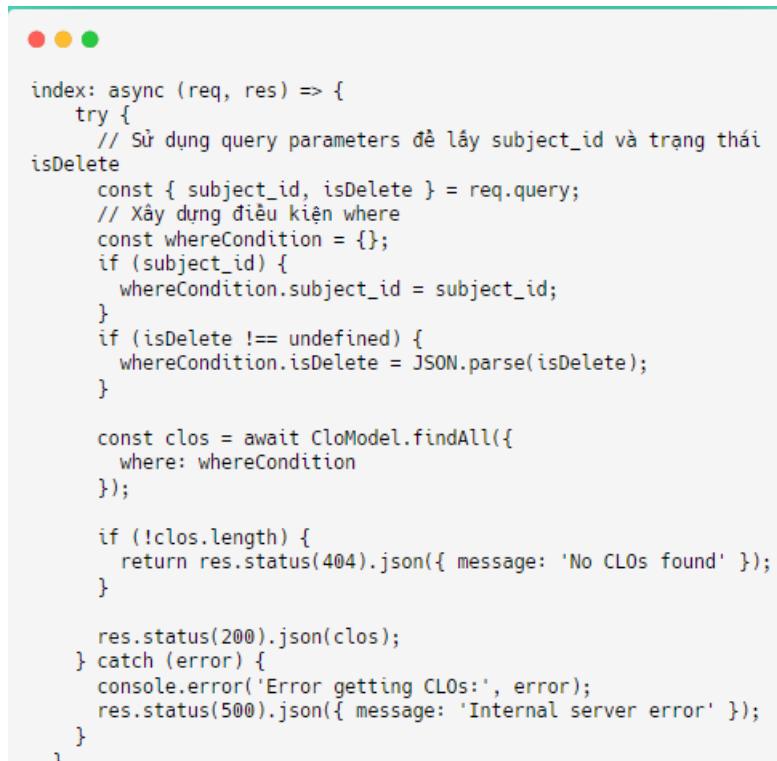
### 17.1. Lấy thông tin tất cả CDR của môn học

API Endpoint: GET /api/admin/clos

**Tham số truy vấn:**

**subject\_id:** Tham số này cho phép lọc các CDR dựa trên ID của môn học. Khi tham số này được cung cấp, API sẽ chỉ trả về các đánh giá liên quan đến môn học có ID tương ứng.

**isDelete:** Tham số này được sử dụng để lọc các CDR dựa trên trạng thái bị xóa hay không.



```
index: async (req, res) => {
    try {
        // Sử dụng query parameters để lấy subject_id và trạng thái
        const { subject_id, isDelete } = req.query;
        // Xây dựng điều kiện where
        const whereCondition = {};
        if (subject_id) {
            whereCondition.subject_id = subject_id;
        }
        if (isDelete !== undefined) {
            whereCondition.isDelete = JSON.parse(isDelete);
        }

        const clos = await CloModel.findAll({
            where: whereCondition
        });

        if (!clos.length) {
            return res.status(404).json({ message: 'No CLOs found' });
        }

        res.status(200).json(clos);
    } catch (error) {
        console.error('Error getting CLOs:', error);
        res.status(500).json({ message: 'Internal server error' });
    }
}
```

Hàm index

Hàm này sử dụng tham số truy vấn subject\_id và isDelete để xây dựng điều kiện lọc cho truy vấn cơ sở dữ liệu. Nếu các tham số này được cung cấp, hàm sẽ tìm kiếm các đối tượng CLO phù hợp. Kết quả trả về bao gồm danh sách CLOs nếu tìm thấy, hoặc mã lỗi 404 nếu không có đối tượng nào thỏa mãn điều kiện. Trường hợp xảy ra lỗi trong quá trình xử lý, hàm trả về mã lỗi 500 kèm thông báo lỗi.

## 17.2. Tạo mới CDR môn học

API Endpoint: POST /api/admin/clo

```
● ● ●
create: async (req, res) => {
  try {
    const { data } = req.body;
    const newCLO = await CloModel.create(data);
    res.status(201).json(newCLO);
  } catch (error) {
    console.error('Error creating CLO:', error);
    res.status(500).json({ message: 'Internal server error' });
  }
},
```

*Hàm create*

## 17.3. Lấy thông tin 1 CDR môn học

API Endpoint: GET /api/admin/clo/:id

```
● ● ●
getByID: async (req, res) => {
  try {
    const { id } = req.params;
    const clo = await CloModel.findOne({ where: { clo_id: id } });
    if (!clo) {
      return res.status(404).json({ message: 'CLO not found' });
    }
    res.status(200).json(clo);
  } catch (error) {
    console.error('Error getting CLO by ID:', error);
    res.status(500).json({ message: 'Internal server error' });
  }
},
```

*Hàm getByID*

## 17.4. Cập nhật 1 CDR môn học

API Endpoint: PUT /api/admin/clo/:id

```
● ● ●
update: async (req, res) => {
  try {
    const { id } = req.params;
    const { data } = req.body;
    const clo = await CloModel.findOne({ where: { clo_id: id } });
    if (!clo) {
      return res.status(404).json({ message: 'CLO not found' });
    }
    await CloModel.update(data, { where: { clo_id: id } });
    res.status(200).json({ message: `Successfully updated CLO with
ID: ${id}` });
  } catch (error) {
    console.error('Error updating CLO:', error);
    res.status(500).json({ message: 'Internal server error' });
  }
},
```

*Hàm update*

## 17.5. Xóa 1 CDR môn học

API Endpoint: DELETE /api/admin/clo/:id

```
delete: async (req, res) => {
  try {
    const { id } = req.params;
    const clo = await CloModel.findOne({ where: { clo_id: id } });
    await CloChapterModel.destroy({ where: { clo_id: id } });
    await PloCloModel.destroy({ where: { clo_id: id } });
    if (!clo) {
      return res.status(404).json({ message: 'CLO not found' });
    }
    await CloModel.destroy({ where: { clo_id: id } });
    res.status(200).json({ message: 'Successfully deleted CLO' });
  } catch (error) {
    console.error('Error deleting CLO:', error);
    res.status(500).json({ message: 'Internal server error' });
  }
},
```

*Hàm delete*

## 17.6. Xóa nhiều CDR môn học

API Endpoint: DELETE /api/admin/clo/multiple

```
deleteMultiple: async (req, res) => {
  const { clo_id } = req.query;
  try {
    const cloIds = clo_id.map(id => parseInt(id));
    for (const id of cloIds) {
      await CloChapterModel.destroy({ where: { clo_id: id } });
      await PloCloModel.destroy({ where: { clo_id: id } });
    }

    await CloModel.destroy({ where: { clo_id: clo_id } });

    res.status(200).json({ message: 'Xóa nhiều CLO thành công' });
  } catch (error) {
    console.error('Lỗi khi xóa nhiều CLO:', error);
    res.status(500).json({ message: 'Lỗi server nội bộ' });
  }
},
```

*Hàm deleteMultiple*

## 17.7. Xóa mềm 1 CDR môn học

API Endpoint: PUT /api/admin/clo/:id/softDelete

```

toggleSoftDeleteById: async (req, res) => {
  try {
    const { id } = req.params;
    const clo = await CloModel.findOne({ where: { clo_id: id } });
    if (!clo) {
      return res.status(404).json({ message: 'clo not found' });
    }
    const updatedIsDeleted = !clo.isDelete;
    await CloModel.update({ isDelete: updatedIsDeleted }, { where: { clo_id: id } });

    res.status(200).json({ message: `Toggled isDelete status to ${updatedIsDeleted}` });

  } catch (error) {
    console.error('Error toggling CloModel delete statuses:', error);
    res.status(500).json({ message: 'Server error' });
  }
}.

```

*Hàm toggleSoftDeleteById*

## 17.8. Xóa mềm nhiều CDR môn học

API Endpoint: PUT /api/admin/clos/softDelete

```

softDeleteMultiple: async (req, res) => {
  try {
    const { data } = req.body;
    const { clo_id } = data;
    if (!Array.isArray(clo_id) || clo_id.length === 0) {
      return res.status(400).json({ message: 'No CloModel ids provided' });
    }

    const clos = await CloModel.findAll({ where: { clo_id: clo_id } });
    if (clos.length !== clo_id.length) {
      return res.status(404).json({ message: 'One or more CloModels not found' });
    }

    const updatedClos = await Promise.all(clos.map(async (clo) =>
{
      const updatedIsDeleted = !clo.isDelete;
      await clo.update({ isDelete: updatedIsDeleted });
      return { clo_id: clo.clo_id, isDelete: updatedIsDeleted };
}));

    res.json({ message: 'CloModel delete statuses toggled', updatedClos });
  } catch (error) {
    console.error('Error toggling CloModel delete status:', error);
    res.status(500).json({ message: 'Server error' });
  }
}.

```

*Hàm softDeleteMultiple*

## PHỤ LỤC 18. API của chương môn học

### 18.1. Lấy thông tin tất cả chương của môn học

API Endpoint: GET /api/admin/chapters

Tham số truy vấn:

**subject\_id**: Tham số này cho phép lọc các chương dựa trên ID của môn học. Khi tham số này được cung cấp, API sẽ chỉ trả về các đánh giá liên quan đến môn học có ID tương ứng.

**isDelete**: Tham số này được sử dụng để lọc các chương dựa trên trạng thái bị xóa hay không.

```
● ○ ●

index: async (req, res) => {
  try {
    const { subject_id, isDelete } = req.query;

    if (subject_id && isDelete !== undefined) {
      const chapters = await ChapterModel.findAll({
        where: { subject_id: subject_id, isDelete: JSON.parse(isDelete) },
        include: [{ model: SubjectModel, attributes: ['subject_id', 'subjectName'] }]
      });

      if (chapters.length === 0) {
        return res.status(404).json({ message: 'No chapters found' });
      }

      return res.status(200).json(chapters);
    } else if (subject_id) {
      const chapters = await ChapterModel.findAll({
        where: { subject_id: subject_id, isDelete: false },
        include: [{ model: SubjectModel, attributes: ['subject_id', 'subjectName'] }]
      });

      if (chapters.length === 0) {
        return res.status(404).json({ message: 'No chapters found' });
      }

      return res.status(200).json(chapters);
    } else {
      const chapters = await ChapterModel.findAll();
      return res.status(200).json(chapters);
    }
  } catch (error) {
    console.error('Error handling chapter requests:', error);
    res.status(500).json({ message: 'Internal server error' });
  }
},
```

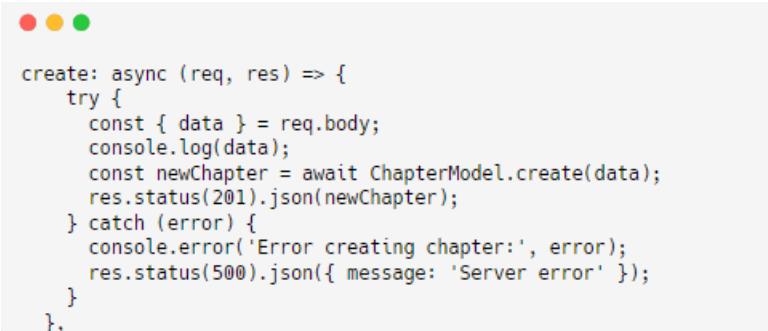
*Hàm index*

Hàm này xử lý yêu cầu truy vấn thông tin các chương dựa trên tham số `subject_id` và `isDelete`. Nếu cả hai tham số được cung cấp, hàm sẽ tìm kiếm các chương liên quan đến môn học cụ thể, có trạng thái xóa mềm tương ứng. Nếu không tìm thấy chương nào, mã lỗi

404 sẽ được trả về. Nếu chỉ có subject\_id, hàm sẽ tìm kiếm các chương không bị xóa. Nếu không có tham số nào, hàm sẽ trả về tất cả các chương. Trong trường hợp có lỗi trong quá trình thực thi, mã lỗi 500 cùng thông báo lỗi sẽ được gửi đến người dùng.

## 18.2. Tạo mới chương môn học

API Endpoint: POST /api/admin/chapter

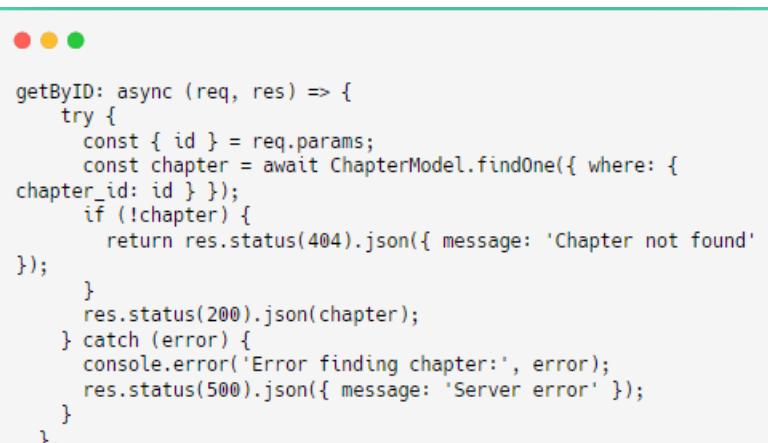


```
create: async (req, res) => {
  try {
    const { data } = req.body;
    console.log(data);
    const newChapter = await ChapterModel.create(data);
    res.status(201).json(newChapter);
  } catch (error) {
    console.error('Error creating chapter:', error);
    res.status(500).json({ message: 'Server error' });
  }
}.
```

*Hàm create*

## 18.3. Lấy thông tin 1 chương môn học

API Endpoint: GET /api/admin/chapter/:id



```
getByID: async (req, res) => {
  try {
    const { id } = req.params;
    const chapter = await ChapterModel.findOne({ where: { chapter_id: id } });
    if (!chapter) {
      return res.status(404).json({ message: 'Chapter not found' });
    }
    res.status(200).json(chapter);
  } catch (error) {
    console.error('Error finding chapter:', error);
    res.status(500).json({ message: 'Server error' });
  }
}.
```

*Hàm getByID*

## 18.4. Cập nhật 1 chương môn học

API Endpoint: PUT /api/admin/chapter/:id

```
● ● ●
update: async (req, res) => {
  try {
    const { id } = req.params;
    const { data } = req.body;
    const chapter = await ChapterModel.findOne({ where: {
      chapter_id: id } });
    if (!chapter) {
      return res.status(404).json({ message: 'Chapter not found' });
    }
    const updatedChapter = await ChapterModel.update(data, {
      where: { chapter_id: id } });
    res.status(200).json({ message: `Successfully updated chapter
with ID: ${id}` });
  } catch (error) {
    console.error('Error updating chapter:', error);
    res.status(500).json({ message: 'Server error' });
  }
}
```

*Hàm update*

## 18.5. Xóa 1 chương môn học

API Endpoint: DELETE /api/admin/chapter/:id

```
● ● ●
delete: async (req, res) => {
  try {
    const { id } = req.params;
    await ChapterModel.destroy({ where: { chapter_id: id } });
    res.status(200).json({ message: 'Successfully deleted chapter' });
  } catch (error) {
    console.error('Error deleting chapter:', error);
    res.status(500).json({ message: 'Server error' });
  }
}
```

*Hàm delete*

## 18.6. Xóa nhiều chương môn học

API Endpoint: DELETE /api/admin/chapter/multiple

```
deleteMultiple: async (req, res) => {
    const { chapter_id } = req.query;
    try {
        const chapterIds = chapter_id.map(id => parseInt(id));
        for (const id of chapterIds) {
            await CloChapterModel.destroy({ where: { chapter_id: id } });
            await RubricItemModel.destroy({ where: { chapter_id: id } });
        }
        await ChapterModel.destroy({ where: { chapter_id: chapter_id } });
    }
    res.status(200).json({ message: 'Successfully deleted multiple chapters' });
} catch (error) {
    console.error('Error deleting multiple chapters:', error);
    res.status(500).json({ message: 'Server error' });
}
}.
```

*Hàm deleteMultiple*

## 18.7. Xóa mềm 1 chương môn học

API Endpoint: PUT /api/admin/chapter/:id/softDelete

```
toggleSoftDeleteById: async (req, res) => {
    try {
        const { id } = req.params;
        const chapter = await ChapterModel.findOne({ where: { chapter_id: id } });
        if (!chapter) {
            return res.status(404).json({ message: 'Chapter not found' });
        }
        const updatedIsDeleted = !chapter.isDelete;
        await ChapterModel.update({ isDelete: updatedIsDeleted }, {
            where: { chapter_id: id } });
        res.status(200).json({ message: `Successfully toggled isDelete status to ${updatedIsDeleted}` });
    } catch (error) {
        console.error('Error updating isDelete status:', error);
        res.status(500).json({ message: 'Server error' });
    }
}
```

*Hàm toggleSoftDeleteById*

## 18.8. Xóa mềm nhiều chương môn học

API Endpoint: PUT /api/admin/chapters/softDelete

```

softDeleteMultiple: async (req, res) => {
  try {
    const { data } = req.body;
    const { chapter_id } = data;
    if (!Array.isArray(chapter_id) || chapter_id.length === 0) {
      return res.status(400).json({ message: 'No chapter ids provided' });
    }

    const chapters = await ChapterModel.findAll({ where: { chapter_id: chapter_id } });
    if (chapters.length !== chapter_id.length) {
      return res.status(404).json({ message: 'One or more chapters not found' });
    }

    const updatedChapters = await Promise.all(chapters.map(async (chapter) => {
      const updatedIsDeleted = !chapter.isDelete;
      await chapter.update({ isDelete: updatedIsDeleted });
      return { chapter_id: chapter.chapter_id, isDelete: updatedIsDeleted };
    }));
  }

  res.status(200).json({ message: 'Chapter delete statuses toggled', updatedChapters });
} catch (error) {
  console.error('Error toggling chapter delete status:', error);
  res.status(500).json({ message: 'Server error' });
}
}

```

*Hàm softDeleteMultiple*

## PHỤ LỤC 19. API của assessment item

### 19.1. Tạo mới assessment item

API Endpoint: POST /api/admin/assessment-item

```

create: async (req, res) => {
  try {
    const { data } = req.body;
    console.log(data);
    const AssessmentItem = await AssessmentItemModel.bulkCreate(data);
    res.status(201).json(AssessmentItem);
  } catch (error) {
    console.error('Lỗi tạo AssessmentItem:', error);
    res.status(500).json({ message: 'Lỗi server' });
  }
}

```

*Hàm create*

### 19.2. Cập nhật 1 assessment item

API Endpoint: PUT /api/admin/assessment-item/:id

```

update: async (req, res) => {
    try {
        const { id } = req.params;
        const { data } = req.body;
        const updated = await AssessmentItemModel.update(data, {
            where: { assessmentItem_id: id } });
        if (updated[0] === 0) {
            return res.status(404).json({ message: 'items not found' });
        }
        res.json(updated);
    } catch (error) {
        console.error('Lỗi cập nhật items:', error);
        res.status(500).json({ message: 'Lỗi server' });
    }
},

```

*Hàm update*

## PHỤ LỤC 20. API của ánh xạ CDR CT với mục tiêu CT

### 20.1. Nhận thông tin tất cả ánh xạ

API Endpoint: GET /api/admin/po-plo

```

getAll: async (req, res) => {
    try {
        const PoPlo = await PoPloModel.findAll();
        res.json(PoPlo);
    } catch (error) {
        console.error('Error getting all PoPlo:', error);
        res.status(500).json({ message: 'Internal server error' });
    }
},

```

*Hàm getAll*

### 20.1. Tạo mới ánh xạ

API Endpoint: POST /api/admin/po-plo

```

SavePoPlo: async (req, res) => {
    try {
        const { dataSave } = req.body;

        if (dataSave && dataSave.length > 0) {
            await PoPloModel.bulkCreate(dataSave);
            res.status(200).json({ message: 'Data saved successfully',
                status: 'success' });
        } else {
            res.status(400).json({ message: 'No data provided for
saving', status: 'failure' });
        }
    } catch (error) {
        console.error('Error in SavePoPlo:', error);
        res.status(500).json({ message: 'Internal server error',
            status: 'error' });
    }
},

```

*Hàm SavePoPlo*

## 20.2. Xóa 1 ánh xạ

API Endpoint: DELETE /api/admin/po-plo

```
DeletePoPlo: async (req, res) => {
    try {
        const { dataDelete } = req.body;

        if (dataDelete && dataDelete.length > 0) {
            await PoPloModel.destroy({
                where: { id_po_plo: dataDelete.map(item => item.id_po_plo)
            }
            });
            res.status(200).json({ message: 'Data deleted successfully',
status: 'success' });
        } else {
            res.status(400).json({ message: 'No data provided for
deletion', status: 'failure' });
        }
    } catch (error) {
        console.error('Error in DeletePoPlo:', error);
        res.status(500).json({ message: 'Internal server error',
status: 'error' });
    }
}
```

*Hàm DeletePoPlo*

## PHỤ LỤC 21. API của ánh xạ CDR CT với CDR môn học

### 21.1. Nhận thông tin tất cả ánh xạ

API Endpoint: GET /api/admin/plo-clo

```
index: async (req, res) => {
  try {
    const { id_clos, clo_id } = req.query;

    if (id_clos) {
      const ids = JSON.parse(id_clos);
      const ploClos = await PloCloModel.findAll({ where: { clo_id: ids } });

      if (ploClos.length === 0) {
        return res.status(404).json({ message: 'No PLO-CLOs found for the given CLO IDs' });
      }
      return res.status(200).json(ploClos);
    } else if (clo_id) {
      const poPlo = await PloCloModel.findAll({ where: { clo_id: clo_id } });
      const ploIds = poPlo.map(item => item.plo_id);

      if (ploIds.length === 0) {
        return res.status(404).json({ message: 'No PLOs found for the given CLO ID' });
      }

      const plos = await PloModel.findAll({ where: { plo_id: ploIds } });
      return res.status(200).json(plos);
    } else {
      const poPlo = await PloCloModel.findAll();
      return res.status(200).json(poPlo);
    }
  } catch (error) {
    console.error('Error handling PLO-CLO requests:', error);
    res.status(500).json({ message: 'Internalserver error' });
  }
},
```

### Hàm index

Hàm này có nhiệm vụ truy vấn thông tin PLO-CLO dựa trên các tham số id\_clos và clo\_id.

Đầu tiên, nếu tham số id\_clos được cung cấp, hàm sẽ phân tích và lấy danh sách ID CLO, sau đó truy xuất các bản ghi PLO-CLO tương ứng. Nếu không tìm thấy bản ghi nào, hàm sẽ trả về mã trạng thái 404 cùng thông báo “No PLO-CLOs found for the given CLO IDs.”

Tiếp theo, trong trường hợp chỉ có clo\_id, hàm sẽ tìm kiếm các bản ghi PLO-CLO liên quan và trích xuất các ID PLO. Nếu không có bản ghi nào, mã 404 cũng sẽ được trả về.

Cuối cùng, nếu không có tham số nào được cung cấp, hàm sẽ truy vấn tất cả các bản ghi PLO-CLO có sẵn.

Trong mọi tình huống, nếu có lỗi xảy ra, mã trạng thái 500 và thông báo “Internal server error” sẽ được gửi tới người dùng, cùng với việc ghi lại thông tin chi tiết về lỗi trong console để hỗ trợ xử lý sự cố.

## 21.2. Tạo mới ánh xạ

API Endpoint: POST /api/admin/plo-clo

```
● ● ●

SaveCloPlo: async (req, res) => {
  try {
    const { dataSave } = req.body;

    if (dataSave && dataSave.length > 0) {
      await PloCloModel.bulkCreate(dataSave);
      res.status(200).json({ message: 'Data saved successfully',
status: 'success' });
    } else {
      res.status(400).json({ message: 'No data provided for saving',
status: 'failure' });
    }
  } catch (error) {
    console.error('Error in SavePoPlo:', error);
    res.status(500).json({ message: 'Internal server error', status:
'error' });
  }
}.
```

Hàm SaveCloPlo

## 21.3. Xóa 1 ánh xạ

API Endpoint: DELETE /api/admin/plo-clo

```
● ● ●

DeleteCloPlo: async (req, res) => {
  try {
    const { dataDelete } = req.body;

    if (dataDelete && dataDelete.length > 0) {
      await PloCloModel.destroy({
        where: { id_plo_clo: dataDelete.map(item => item.id_plo_clo)
      });
      res.status(200).json({ message: 'Data deleted successfully',
status: 'success' });
    } else {
      res.status(400).json({ message: 'No data provided for
deletion', status: 'failure' });
    }
  } catch (error) {
    console.error('Error in DeletePoPlo:', error);
    res.status(500).json({ message: 'Internal server error', status:
'error' });
  }
}.
```

Hàm DeleteCloPlo

## PHỤ LỤC 22. API của ánh xạ chương môn học với CDR môn học

### 22.1. Nhận thông tin tất cả ánh xạ

API Endpoint: GET /api/admin/clo-chapter

```
getCloChapter: async (req, res) => {
  try {
    const { clo_id, chapter_ids } = req.query;

    if (clo_id) {
      const cloChapters = await CloChapterModel.findAll({ where: { clo_id: clo_id } });

      if (!cloChapters.length) {
        return res.status(404).json({ message: 'No chapters found for the given CLO ID' });
      }

      const chapterIds = cloChapters.map(item => item.chapter_id);
      const chapters = await ChapterModel.findAll({ where: { chapter_id: chapterIds } });
      return res.status(200).json(chapters);
    }
    if (chapter_ids) {
      const ids = JSON.parse(chapter_ids);
      const cloChapters = await CloChapterModel.findAll({ where: { chapter_id: ids } });

      if (!cloChapters.length) {
        return res.status(404).json({ message: 'No CLO-Chapters found for the given Chapter IDs' });
      }
      return res.status(200).json(cloChapters);
    }

    const allCloChapters = await CloChapterModel.findAll();
    return res.status(200).json(allCloChapters);
  } catch (error) {
    console.error('Error handling CLO-CHAPTER request:', error);
    res.status(500).json({ message: 'Internal server error' });
  }
},
```

### Hàm *getCloChapter*

Hàm này được thiết kế để truy xuất thông tin liên quan đến các chương (chapter) của CLO (Course Learning Outcomes) dựa trên các tham số *clo\_id* và *chapter\_ids*.

Đầu tiên, nếu *clo\_id* được cung cấp, hàm sẽ tìm tất cả các bản ghi trong mô hình CloChapterModel dựa trên *clo\_id*. Nếu không tìm thấy bản ghi nào, hàm sẽ trả về mã 404 cùng thông báo “No chapters found for the given CLO ID”. Ngược lại, nó sẽ trích xuất các *chapter\_id* và tiếp tục truy vấn mô hình ChapterModel để lấy thông tin chi tiết về các chương đó.

Trong trường hợp *chapter\_ids* được cung cấp, hàm sẽ phân tích JSON để lấy danh sách ID chương và tìm các bản ghi CLO-Chapter tương ứng trong CloChapterModel. Nếu không có bản ghi nào, mã 404 sẽ được trả về cùng với thông báo “No CLO-Chapters found for the given Chapter IDs”.

Cuối cùng, nếu không có tham số nào được cung cấp, hàm sẽ truy vấn và trả về tất cả các bản ghi CLO-Chapter có sẵn.

Nếu xảy ra lỗi trong quá trình thực hiện, hàm sẽ ghi lại thông tin chi tiết về lỗi và

trả về mã trạng thái 500 với thông báo “Internal server error” cho người dùng.

## 22.2. Tạo mới ánh xạ

API Endpoint: POST /api/admin/clo-chapter

```
● ● ●

SaveCloChapter: async (req, res) => {
  try {
    const { dataSave } = req.body;
    if (dataSave && dataSave.length > 0) {
      await CloChapterModel.bulkCreate(dataSave);
    }
    res.json({ message: 'Data saved successfully' });
  } catch (error) {
    console.error('Error SaveOrDelete:', error);
    res.status(500).json({ message: 'Internal server error' });
  }
},
```

*Hàm SaveCloChapter*

## 22.3. Xóa 1 ánh xạ

API Endpoint: DELETE /api/admin/clo-chapter

```
● ● ●

DeleteCloChapter: async (req, res) => {
  try {
    const { dataDelete } = req.body;

    console.log(dataDelete);
    if (dataDelete && dataDelete.length > 0) {
      await CloChapterModel.destroy({ where: { id_clo_chapter : dataDelete.map(item => item.id_clo_chapter) } });
    }

    res.json({ message: 'Data deleted successfully' });
  } catch (error) {
    console.error('Error SaveOrDelete:', error);
    res.status(500).json({ message: 'Internal server error' });
  }
},
```

*Hàm DeleteCloChapter*