

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



**ISO 9001:2015**

**LÂM NGỌC TÀI**

**XÂY DỰNG HỆ THỐNG TRỰC QUAN HÓA KẾT  
QUẢ HỌC TẬP 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 HỆ THỐNG TRỰC QUAN HÓA KẾT  
QUẢ HỌC TẬP SINH VIÊN CNTT**

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

Sinh viên: Lâm Ngọc Tài

Lớp: DA20TTB

MSSV: 110120152

GVHD: TS. Nguyễn Bảo An

TRÀ VINH, NĂM 2024

## LỜI MỞ ĐẦU

Trong quá trình lựa chọn đề tài, tôi quyết định chọn xây dựng hệ thống trực quan hóa kết quả học tập sinh viên công nghệ thông tin. Lý do tôi chọn đề tài này là vì hiện nay, việc đánh giá và theo dõi kết quả học tập của sinh viên thường gặp nhiều khó khăn, đặc biệt trong việc trình bày dữ liệu một cách trực quan và dễ hiểu. Hệ thống trực quan hóa kết quả học tập sẽ giúp cho giảng viên và sinh viên dễ dàng nhận biết được những điểm mạnh và yếu trong quá trình học tập, từ đó có những biện pháp cải thiện kịp thời.

Trong đề tài này, tôi sẽ vận dụng những kiến thức đã học về lập trình, cơ sở dữ liệu, và trực quan hóa dữ liệu để xây dựng một hệ thống hoàn chỉnh, đáp ứng được nhu cầu thực tế của nhà trường và sinh viên.

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, mục tiêu nghiên cứu, nội dung nghiên cứu, đối tượng, phạm vi và phương pháp tiếp cận.

### **Chương 2: Cơ sở lý thuyết**

Trình bày các kiến thức cơ bản và các công nghệ sẽ sử dụng trong đề tài, bao gồm lập trình, cơ sở dữ liệu, và trực quan hóa dữ liệu.

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

Mô tả quá trình triển khai hệ thống, các công cụ và ngôn ngữ lập trình sử dụng, cùng với các bước kiểm thử hệ thống.

### **Chương 4: Kết quả nghiên cứu**

Chương này mô tả quá trình triển khai hệ thống, bao gồm việc phát triển, kiểm thử và cài đặt hệ thống trực quan hóa kết quả học tập.

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

Đánh giá hiệu quả của hệ thống đã xây dựng, nêu ra những ưu điểm và nhược điểm, cùng với những hướng phát triển trong tương lai.

## LỜI CẢM ƠN

Trước hết, em xin gửi lời cảm ơn đến toàn thể quý thầy cô, giảng viên 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 điều kiện tốt nhất để em hoàn thành trọn vẹn bài đồ án cơ sở ngành này.

Tiếp theo, em xin tỏ lòng biết ơn sâu sắc đến thầy Nguyễn Bảo Ân – Giảng viên Khoa Kỹ thuật & Công nghệ, Trường Đại học Trà Vinh, trong quá trình hướng dẫn đã vô cùng tâm huyết trong việc truyền đạt kiến thức đến em.

Trong bài báo cáo, do lượng kiến thức và kinh nghiệm còn khiêm tốn, thời gian nghiên cứu ngắn nên vẫn còn một số sai sót nhỏ không đáng kể. Do đó, em kính mong quý thầy cô thông cảm, góp ý để em có thể tiếp thu và cải thiện cho những nghiên cứu trong tương lai.

Sau tất cả, kính chúc các thầy cô luôn dồi dào sức khoẻ.

Em xin chân thành cảm ơn!

Trà Vinh, ngày ..... tháng 6 năm 2024

Sinh viên thực hiện

Lâm Ngọc Tài

## NHẬN XÉT

## **Giảng viên hướng dẫn**

# BẢN NHẬN XÉT ĐỒ ÁN, KHÓA LUẬN TỐT NGHIỆP

Ho và tên sinh viên: ..... MSSV: .....

Ngành: ..... Khóa: .....

Tên đề tài: .....

Họ và tên Giáo viên hướng dẫn: .....

Chức danh: ..... Học vị: .....

## NHÂN XÉT

- ## 1. Nội dung đề tài:

- ## 2. Ưu điểm:

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

- ### 3. Khuyết điểm:

#### 4. Điểm mới đề tài:

### 5. Giá trị thực trên đèn 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)

# NHẬN XÉT

## Giảng viên chấm

(ký và ghi rõ họ tên)

## BẢN NHẬN XÉT ĐỒ ÁN, KHÓA LUẬN TỐT NGHIỆP

Họ và tên người nhận xét: .....

Chức danh: ..... Học vị: .....

Chuyên ngành: .....

Cơ quan công tác: .....

Tên sinh viên: .....

Tên đề tài đồ án, khóa luận tốt nghiệp: .....

.....  
.....

### I. Ý KIẾN NHẬN XÉT

1. Nội dung:

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

2. Điểm mới các kết quả của đồ án, khóa luận:

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

3. Ứng dụng thực tế:

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

## II. CÁC VẤN ĐỀ CẦN LÀM RÕ

### III. KẾT LUẬN

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

....., ngày ..... tháng ..... năm 2024

### Người nhận xét

(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 .....	3
1.3. Nội dung.....	3
1.4. Đối tượng và phạm vi nghiên cứu .....	3
1.5. Phương pháp nghiên cứu .....	3
<b>CHƯƠNG 2. CƠ SỞ LÝ THUYẾT .....</b>	<b>4</b>
2.1. Phân tích kết quả học tập .....	4
2.1.1. Thu thập dữ liệu .....	4
2.1.2. Chỉ số .....	5
2.1.3. Phân tích dữ liệu .....	5
2.1.4. Trực quan hóa dữ liệu .....	6
2.1.5. Áp dụng vào kết quả học tập .....	6
2.2. Dashboard thể hiện kết quả học tập .....	6
2.2.1. Thành phần chính của dashboard.....	7
2.2.2. Thiết kế giao diện người dùng (UI) của Dashboard .....	8
2.3. Các công nghệ cần thiết để thực hiện đồ án .....	8
2.3.1. Công nghệ backend.....	8
2.3.2. Công nghệ Frontend.....	11
2.3.3. Hệ quản trị cơ sở dữ liệu MySQL.....	16
2.3.4. Môi trường triển khai ứng dụng (VPS).....	17
2.4. Kiến trúc ứng dụng .....	19
2.4.1. Kiến trúc client-server .....	19
2.4.2. Giao tiếp giữa front-end và back-end RESTful API .....	21
2.4.3. Quản lý và làm tài liệu API với Swagger .....	23

### **CHƯƠNG 3. HIỆN THỰC HÓA NGHIÊN CỨU ..... 26**

3.1. Mô tả bài toán .....	26
3.2. Yêu cầu chức năng .....	28
3.3. Đặt tả hệ thống .....	29
3.4. Lược đồ dữ liệu .....	31
3.5. Kiến trúc hệ thống.....	44
3.5.1. Backend.....	44
3.5.2. Kiến trúc Frontend .....	47
3.6. Xây dựng Backend.....	49
3.6.1. Kết nối cơ sở dữ liệu .....	49
3.6.2. Định Nghĩa Mô Hình với Sequelize .....	50
3.6.3. Xác thực người dùng.....	53
3.6.4. Tạo và Quản Lý API Routes với Express.js .....	55
3.6.5. Các API.....	56
3.7. Xây dựng Frontend .....	158
3.7.1. Cấu hình kết nối với thư viện, framework .....	158
3.7.2. Tạo Giao Diện Người Dùng .....	159
3.7.3. Cấu Hình Axios .....	159

### **CHƯƠNG 4. KẾT QUẢ NGHIÊN CỨU ..... 162**

4.1. Kết quả nghiên cứu Backend .....	162
4.1.1. Khởi chạy .....	162
4.1.2. Xác thực người dùng.....	163
4.1.3. Các API.....	165
4.1.4. Swagger.....	166
4.2. Kết quả nghiên cứu Frontend.....	167
4.2.1. Kết quả trực quan hóa kết quả học tập.....	167

4.4.1. Giao diện đăng nhập .....	173
4.4.2. Giao diện trang chủ.....	173
4.4.3. Giao diện quản lý giáo viên .....	174
4.4.4. Giao diện quản lý sinh viên .....	177
4.4.5. Giao diện quản lý lớp học .....	178
4.4.6. Giao diện quản lý khóa học .....	180
<b>CHƯƠNG 5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN .....</b>	<b>182</b>
5.1. Kết luận.....	182
5.2. Hướng phát triển .....	182
<b>DANH MỤC TÀI LIỆU THAM KHẢO.....</b>	<b>184</b>

## MỤC LỤC ẢNH

Hình 2.1. Mô tả quá trình phân tích kết quả học tập. [1] .....	4
Hình 2.2. Giao diện dashboard điện hình. ....	7
Hình 2.3. Mô tả kiến trúc client-server.....	19
Hình 2.4. Cách thức hoạt động của RESTful API .....	21
Hình 2.5. Ví dụ về API thao tác trên swagger .....	23
Hình 3.1 Sơ đồ use case .....	29
Hình 3.2. Lược đồ cơ sở dữ liệu .....	31
Hình 3.3. Kiến trúc hệ thống backend .....	44
Hình 3.4. Kiến trúc frontend .....	47
Hình 3.5. Import các thư viện cần thiết kết nối cơ sở dữ liệu.....	49
Hình 3.6. Tạo instance của Sequelize .....	49
Hình 3.7. Kiểm tra kết nối với dữ liệu .....	50
Hình 3.8. Import các thư viên tạo modal .....	50
Hình 3.9. Định nghĩa modal.....	51
Hình 3.10. Thiết lập quan hệ mô hình .....	52
Hình 3.11. Xuất mô hình.....	52
Hình 3.12. Cấu hình JWT Strategy .....	53
Hình 3.13. Sử dụng JWT Strategy với Passport .....	53
Hình 3.14. Kiểm tra Access Token.....	54
Hình 3.15. Xác thực Access Token .....	54
Hình 3.16. Kiểm tra Refresh Token.....	55
Hình 3.17. Định nghĩa các tuyến đường cho môn học với middleware xác thực .....	55
Hình 3.18. API register.....	56
Hình 3.19. API đăng nhập.....	58
Hình 3.20. API đổi mật khẩu .....	59
Hình 3.21. API đăng xuất .....	61
Hình 3.22. API lấy thông tin người dùng. ....	62
Hình 3.23. API làm mới token.....	63
Hình 3.24. Thu hồi token .....	65
Hình 3.25. API tỉ lệ đạt được của chuẩn đầu ra môn học. ....	66
Hình 3.26. API tỉ lệ đạt được câu Plo .....	68

Hình 3.27. API điểm trung bình của subject.....	69
Hình 3.28. API lấy điểm của một sinh viên.....	71
Hình 3.29. API điểm trung bình môn học.....	73
Hình 3.30. API lấy tất cả các lớp học. ....	74
Hình 3.31. API lấy tất cả thông tin lớp học và giáo viên.....	75
Hình 3.32. Lấy thông tin lớp học bởi id.....	77
Hình 3.33. API xuất file excel lớp học. ....	78
Hình 3.34. API tạo lớp học mới.....	79
Hình 3.35. API cập nhật lớp học.....	80
Hình 3.36. API xóa khóa học.....	81
Hình 3.37. API thay đổi trạng thái lớp học.....	82
Hình 3.38. API đảo ngược trạng thái lớp học .....	83
Hình 3.39. API lấy thông tin tất cả lớp học. ....	84
Hình 3.40. API tạo mới năm học. ....	85
Hình 3.41. API lấy thông tin một lớp học bởi id. ....	86
Hình 3.42.API cập nhật thông tin lớp học .....	87
Hình 3.43. API xóa lớp học. ....	88
Hình 3.44. API lấy lớp học trạng thái ẩn. ....	89
Hình 3.45. API lấy lớp học trạng thái không ẩn. ....	90
Hình 3.46. API đảo ngược trạng thái năm học. ....	90
Hình 3.47. API lấy tất cả học kì.....	91
Hình 3.48. API tạo học kì mới. ....	92
Hình 3.49. API lấy thông tin của một học kì. ....	93
Hình 3.50. API cập nhật học kì.....	93
Hình 3.51. API xóa một học kì. ....	94
Hình 3.52. API lấy danh sách học kì bị ẩn .....	95
Hình 3.53. API lấy danh sách học kì không ẩn.....	95
Hình 3.54. Đảo ngược trạng thái học kì.....	96
Hình 3.55. API lấy tất cả các khóa học.....	97
Hình 3.56. API lấy tất cả thông tin khóa học.....	98
Hình 3.57. API tạo khóa học mới. ....	99
Hình 3.58. API lấy tất cả thông tin khóa học và số lượng sinh viên đăng kí .....	101

Hình 3.59. API lấy thông tin của một lớp học .....	103
Hình 3.60. API lấy thông tin khóa học bởi id giáo viên .....	105
Hình 3.61. Lấy thông tin chi tiết của một khóa học cụ thể .....	106
Hình 3.62. API cập nhật khóa học .....	108
Hình 3.63. API xóa một khóa học .....	109
Hình 3.64. API lấy danh sách khóa học bị ẩn .....	110
Hình 3.65. API đảo ngược khóa học .....	111
Hình 3.66. API lấy danh sách sinh viên đã đăng ký khóa học .....	112
Hình 3.67. API xuất file excel từ danh sách id sinh viên .....	114
Hình 3.68. API xuất file excel bởi id khóa học .....	116
Hình 3.69. API lưu danh sách sinh viên đăng ký khóa học từ file excel .....	118
Hình 3.70. API lấy danh sách sinh viên .....	120
Hình 3.71. API lấy danh sách sinh viên bởi id lớp học .....	121
Hình 3.72. API kết quả học tập của một sinh viên .....	123
Hình 3.73. API tạo sinh viên mới .....	124
Hình 3.74. API lấy danh sách lớp sinh viên có thông tin .....	125
Hình 3.75. API lấy thông tin sinh viên bằng id sinh viên .....	126
Hình 3.76. API cập nhật thông tin của một sinh viên .....	127
Hình 3.77. Xóa thông tin một sinh viên .....	128
Hình 3.78. API lấy danh sách sinh viên bị ẩn .....	129
Hình 3.79. API lấy danh sách sinh viên .....	130
Hình 3.80. API đảo ngược trạng thái của sinh viên .....	130
Hình 3.81. API xuất file excel mẫu điền thông tin sinh viên .....	131
Hình 3.82. API xuất file excel với thông tin sinh viên .....	133
Hình 3.83. API xuất excel danh sách sinh viên bởi id lớp học .....	135
Hình 3.84. API tạo sinh viên bằng file excel .....	137
Hình 3.85. Cập nhật danh sách sinh viên bằng file excel .....	139
Hình 3.86. API lấy danh sách sinh viên có phân trang .....	141
Hình 3.87. API lấy danh sách sinh viên bị chặn .....	143
Hình 3.88. API tạo mới một giáo viên .....	145
Hình 3.89. API lấy thông tin giáo viên bởi id giáo viên .....	146
Hình 3.90. API cập nhật giáo viên bởi id giáo viên .....	147

Hình 3.91. API chặn một giáo viên .....	148
Hình 3.92. API chặn nhiều giáo viên.....	149
Hình 3.93. API bỏ chặn một giáo viên .....	150
Hình 3.94. API bỏ chặn nhiều giáo viên.....	151
Hình 3.95. API ẩn một giáo viên .....	151
Hình 3.96. API ẩn nhiều giáo viên.....	152
Hình 3.97. API xuất file excel mẫu điền thông tin giáo viên .....	153
Hình 3.98. API xuất file excel với thông tin giáo viên .....	154
Hình 3.99. API lưu giáo viên bằng file excel. ....	156
Hình 3.100. Cấu hình tệp tailwind.config.js .....	158
Hình 3.101. Cấu hình kết nối fire base .....	158
Hình 3.102. Khởi tạo root và render ứng dụng.....	159
Hình 3.103. Import thư viện Axios.....	159
Hình 3.104. Instance axios với cấu hình mặc định.....	160
Hình 3.105. Thiết lập interceptors cho Axios .....	160
Hình 3.106. Ví dụ về sử dụng axiosAdmin .....	161
Hình 4.1. Khởi chạy backend .....	162
Hình 4.2. API xử lý đăng nhập .....	163
Hình 4.3. Tìm người dùng trong cơ sở dữ liệu .....	163
Hình 4.4. Kiểm tra mật khẩu.....	164
Hình 4.5. khởi tạo các biến cần thiết. ....	164
Hình 4.6. Thu hồi và hết hạn các biến cần thiết.....	164
Hình 4.7. Lưu token vào cơ sở dữ liệu .....	164
Hình 4.8. Đặt mã thông báo trong cookies. ....	165
Hình 4.9. Phải hồi khi chưa đăng nhập khi gọi end point.....	165
Hình 4.10. Nội dung trả lời khi đăng nhập .....	166
Hình 4.11. Giao diện swagger .....	166
Hình 4.12. Tỉ lệ đạt được của chuẩn đầu ra môn học .....	167
Hình 4.13. Biểu đồ đường tỉ lệ đạt của chuẩn đầu ra chương trình. ....	168
Hình 4.14. Phân bố điểm của khóa học. ....	169
Hình 4.15. Biểu đồ cột thể hiện điểm của sinh viên .....	170
Hình 4.16. Chi tiết biểu đồ cột.....	171

Hình 4.17. Biểu đồ điểm trung bình khóa học.....	171
Hình 4.18. Filter câu biểu đồ điểm trung bình khóa học. ....	172
Hình 4.19. Giao diện đăng nhập. .....	173
Hình 4.20. Giao diện trang chủ.....	173
Hình 4.21. Danh sách giáo viên.....	174
Hình 4.22. Giao diện khi tương tác với bảng giáo viên.....	175
Hình 4.23. Giao diện danh sách giáo viên đã chặn.....	176
Hình 4.24. Giao diện tạo giáo viên mới.....	176
Hình 4.25. Giao diện quản lý sinh viên. ....	177
Hình 4.26. Giao diện quản lý lớp học. ....	178
Hình 4.27. Giao diện quản lý khóa học. ....	180

## MỤC LỤC BẢNG

Bảng 3.1. Mô tả Actor.....	29
Bảng 3.2 Mô tả use case .....	29
Bảng 3.3. Mô tả bảng Student.....	32
Bảng 3.4. Mô tả bảng Teacher .....	32
Bảng 3.5. Môn tả bảng Subjects .....	33
Bảng 3.6. Mô tả bảng Courses .....	34
Bảng 3.7. Mô tả bảng academic_year .....	34
Bảng 3.8. Mô tả bảng Semesters.....	35
Bảng 3.9. Mô tả bảng Semester_academic_years.....	35
Bảng 3.10. Mô tả bảng Assessments .....	36
Bảng 3.11. Mô tả bảng assessmentItems .....	37
Bảng 3.12. Mô tả bảng Chapters.....	37
Bảng 3.13. Mô tả bảng Classes .....	38
Bảng 3.14. Mô tả bảng Clos .....	38
Bảng 3.15. Mô tả bảng Plos .....	39
Bảng 3.16. Mô tả bảng Pos .....	39
Bảng 3.17. Mô tả bảng course_enrollment .....	40
Bảng 3.18. Mô tả bảng Programs .....	40
Bảng 3.19. Mô tả bảng Rubrics .....	41
Bảng 3.20. Mô tả bảng RubricItems .....	41
Bảng 3.21. Mô tả bảng map_clo_chapters.....	42
Bảng 3.22. Mô tả bảng map_plo_clo .....	42
Bảng 3.23. Mô tả bảng map_po_plo .....	43
Bảng 3.24. Mô tả bảng refresh_tokens .....	43

## **DANH MỤC TỪ VIẾT TẮT**

Từ viết tắt	Ý nghĩa
API	Application programming interface
CLO	Course learning objectives
CNTT	Công nghệ thông tin
CSS	Cascading style sheets
DOM	Document object model
HTML	Hypertext markup language
HTTP	Hypertext Transfer Protocol
ID	Identifier
JSON	Javascript Object Notation
JSX	JavaScript XML
MySQL	My structured query language
PO	Program objectives
ORM	Object-relational mapping
PLO	Program learning objectives
RDBMS	Relational database management system
REST	Representational state transfer
UI	User interface
URL	Uniform resource locator
npm	Node package manager

## CHƯƠNG 1. ĐẶT VẤN ĐỀ

### 1.1. Lý do chọn đề tài

Để đánh giá chất lượng của một chương trình đào tạo, mức độ đạt được của sinh viên đối với các chuẩn đầu ra (PLO - Program Learning Objectives) là yếu tố then chốt. Quy trình xây dựng một chương trình đào tạo bắt đầu từ việc xác định các mục tiêu chương trình (PO - Program Objectives). Từ các PO này, giáo viên thiết lập danh sách các chuẩn đầu ra PLO. Để sinh viên đạt được các PLO chương trình đào tạo bao gồm nhiều học phần, mỗi học phần đóng góp vào việc trang bị kiến thức, kỹ năng, và thái độ cho sinh viên. Mỗi học phần có các chuẩn đầu ra của môn học (CLO - Course Learning Objectives) và được chia thành nhiều chương học. Do đó, cần có một bảng ánh xạ từ CLO tới PLO. Khi giảng viên đánh giá môn học, họ sử dụng các rubric để xác định xem sinh viên có đạt được các CLO hay không. Từ đó, thông qua các thao tác thống kê, mức độ đạt được các PLO của sinh viên được tính toán.

Các thông tin về mức độ đạt chuẩn đầu ra của sinh viên cung cấp những thông tin quan trọng, giúp nhà trường, khoa, giảng viên và sinh viên thực hiện các điều chỉnh để nâng cao chất lượng đào tạo. Đối với nhà trường, thông tin này giúp đánh giá chất lượng tổng thể của chương trình đào tạo, từ đó lập kế hoạch chiến lược và cải thiện hiệu quả giáo dục. Ở cấp độ chương trình, khoa có thể đánh giá và điều chỉnh chương trình đào tạo, đảm bảo các mục tiêu giáo dục được đáp ứng và duy trì tiêu chuẩn cao. Các giảng viên có thể hiểu rõ chất lượng dạy học ở cấp độ môn học, nhận diện các khó khăn trong các học phần và hỗ trợ sinh viên gặp khó khăn. Đối với sinh viên, thông tin này giúp họ tự phản ánh về năng lực học tập của mình, so sánh với các bạn cùng lớp và đặt ra mục tiêu phấn đấu cụ thể hơn.

Trong bối cảnh này, một hệ thống trực quan hóa mức độ đạt chuẩn đầu ra của sinh viên sẽ mang lại nhiều lợi ích lớn. Hệ thống sẽ cung cấp cái nhìn tổng thể và chi tiết về mức độ đạt chuẩn đầu ra của sinh viên, giúp các bên liên quan dễ dàng nắm bắt tình hình. Các thông tin được trực quan hóa giúp phân tích chi tiết hơn về các điểm mạnh và điểm yếu trong quá trình học tập, từ đó đưa ra các biện pháp cải thiện cụ thể. Sinh viên và giảng viên có thể tương tác tốt hơn dựa trên các dữ liệu trực quan, thúc đẩy việc học tập và giảng dạy hiệu quả. Nhà trường và khoa có thể ra quyết định chính xác và kịp thời dựa trên các thông tin trực quan, nâng cao chất lượng và hiệu quả của chương trình đào tạo.

Hệ thống trực quan hóa kết quả học tập sẽ giúp phân tích dữ liệu học tập, sử dụng các kỹ thuật phân tích dữ liệu để trực quan hóa các kết quả học tập, từ đó nhận diện các xu hướng và mẫu trong dữ liệu học tập của sinh viên. Sinh viên có thể theo dõi tiến độ học tập của mình một cách dễ dàng và trực quan, giúp họ tự định hướng và điều chỉnh kế hoạch học tập. Giảng viên có thể sử dụng hệ thống để đánh giá hiệu quả của các phương pháp giảng dạy và điều chỉnh khi cần thiết. Nhà trường và khoa có thể sử dụng trang web để quản lý và cải thiện chương trình đào tạo, đảm bảo đáp ứng các chuẩn đầu ra và nâng cao chất lượng giáo dục.

Hệ thống trực quan hóa kết quả học tập mang lại nhiều lợi ích cho cả sinh viên, giảng viên. Đối với sinh viên, hệ thống này cung cấp một cái nhìn rõ ràng về tiến độ học tập và mức độ đạt chuẩn đầu ra của họ. Điều này không chỉ giúp sinh viên tự đánh giá năng lực của mình mà còn tạo động lực để họ phấn đấu đạt các mục tiêu học tập cụ thể. Hơn nữa, sinh viên có thể dễ dàng nhận diện những điểm mạnh và điểm yếu của mình, từ đó lập kế hoạch học tập phù hợp để cải thiện kết quả.

Đối với giảng viên, hệ thống trực quan hóa cung cấp các dữ liệu chi tiết về hiệu quả của các phương pháp giảng dạy và nội dung học tập. Giảng viên có thể sử dụng thông tin này để điều chỉnh và cải thiện phương pháp giảng dạy của mình, từ đó nâng cao chất lượng giảng dạy và giúp sinh viên đạt kết quả học tập tốt hơn. Hệ thống cũng giúp giảng viên nhận diện các sinh viên gặp khó khăn và cung cấp hỗ trợ kịp thời, góp phần giảm tỷ lệ bỏ học và tăng tỷ lệ hoàn thành khóa học.

Với nhà quản lý giáo dục, hệ thống trực quan hóa kết quả học tập cung cấp một công cụ mạnh mẽ để theo dõi và đánh giá chất lượng chương trình đào tạo. Nhà quản lý có thể sử dụng các dữ liệu từ hệ thống để ra quyết định chiến lược, điều chỉnh chương trình đào tạo và đảm bảo rằng các mục tiêu giáo dục được đạt được. Hệ thống cũng giúp nhà quản lý nhận diện các xu hướng và vấn đề trong giáo dục, từ đó đề xuất các biện pháp cải thiện hiệu quả và chất lượng giáo dục.

Việc xây dựng một hệ thống trực quan hóa kết quả học tập sinh viên công nghệ thông tin là rất cần thiết và cấp bách. Nó không chỉ giúp các bên liên quan đánh giá và cải thiện chất lượng giáo dục mà còn thúc đẩy sự tương tác và phát triển toàn diện của sinh viên. Hệ thống này sẽ là công cụ mạnh mẽ giúp nâng cao hiệu quả học tập và giảng dạy, đảm bảo sinh viên đạt được các chuẩn đầu ra và chuẩn bị tốt hơn cho tương lai.

## **1.2. Mục tiêu**

Phát triển một hệ thống trực quan hóa kết quả học tập của sinh viên công nghệ thông tin.

Cung cấp giao diện thân thiện và dễ sử dụng cho sinh viên và giảng viên.

Hỗ trợ giảng viên trong việc đánh giá và quản lý kết quả học tập của sinh viên.

Nâng cao khả năng đánh giá và theo dõi tiến độ học tập của sinh viên.

## **1.3. Nội dung**

Đề tài này bao gồm các nội dung chính sau:

- Tổng quan về đề tài và tầm quan trọng của việc trực quan hóa kết quả học tập.
- Cơ sở lý thuyết và các công nghệ liên quan đến việc xây dựng hệ thống trực quan hóa dữ liệu.
- Quá trình phân tích yêu cầu và thiết kế hệ thống.
- Quá trình triển khai, kiểm thử và cài đặt hệ thống trực quan hóa kết quả học tập.
- Đánh giá hiệu quả của hệ thống và đề xuất các hướng phát triển tiếp theo.

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

Đối tượng nghiên cứu của đề tài này là các sinh viên và giảng viên thuộc ngành công nghệ thông tin. Phạm vi nghiên cứu bao gồm:

- Các phương pháp và công nghệ sử dụng trong việc trực quan hóa dữ liệu.
- Thiết kế và triển khai hệ thống trực quan hóa kết quả học tập.
- Đánh giá hiệu quả của hệ thống trong môi trường giáo dục thực tế.

## **1.5. Phương pháp nghiên cứu**

Nghiên cứu tài liệu: Tìm hiểu và tổng hợp các tài liệu liên quan đến trực quan hóa dữ liệu và các hệ thống quản lý kết quả học tập.

Phân tích yêu cầu: Khảo sát và thu thập ý kiến từ sinh viên và giảng viên để xác định các yêu cầu của hệ thống.

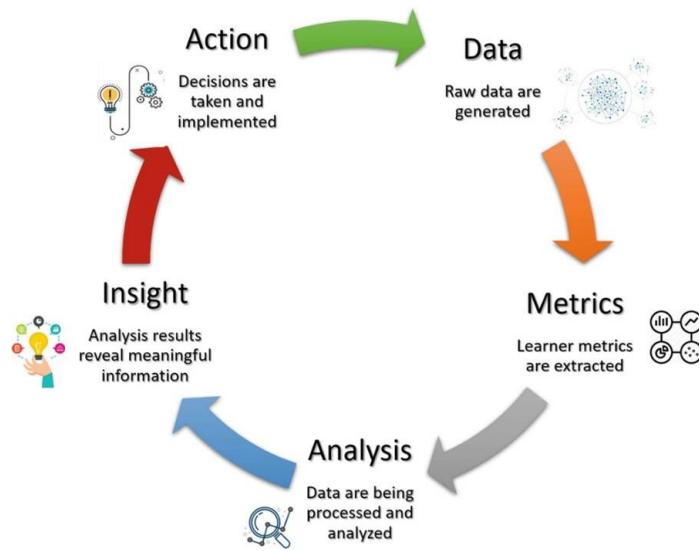
Thiết kế hệ thống: Thiết kế kiến trúc tổng thể và các thành phần chi tiết của hệ thống.

Triển khai và kiểm thử: Phát triển, kiểm thử và cài đặt hệ thống trong môi trường thực tế.

## CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

### 2.1. Phân tích kết quả học tập

Phân tích kết quả học tập là một bước quan trọng trong việc hiểu và cải thiện quá trình học tập của sinh viên. Dưới đây là các khía cạnh cần xem xét trong quá trình phân tích.



Hình 2.1. Mô tả quá trình phân tích kết quả học tập. [1]

#### 2.1.1. Thu thập dữ liệu

Việc thu thập dữ liệu là bước đầu tiên và cần thiết để có thể tiến hành phân tích. Các nguồn dữ liệu thường bao gồm:

Điểm số: Điểm thi, điểm bài tập, điểm tham gia lớp học.

Tham gia: Tỷ lệ tham gia các hoạt động học tập, lớp học, và các buổi thảo luận.

Phản hồi: Phản hồi từ sinh viên về khóa học và giảng viên.

Thông tin cá nhân: Thông tin về sinh viên như giới tính, độ tuổi, nền tảng học vấn, và các yếu tố cá nhân khác có thể ảnh hưởng đến kết quả học tập.

Các phương pháp thu thập dữ liệu có thể bao gồm:

Bảng điểm và báo cáo học tập: Được cung cấp bởi hệ thống quản lý học tập (LMS).

Khảo sát và phỏng vấn: Sử dụng các bảng câu hỏi trực tuyến hoặc trực tiếp để thu thập ý kiến từ sinh viên. [1]

**Hệ thống theo dõi:** Sử dụng các công cụ kỹ thuật số để theo dõi hoạt động của sinh viên trên các nền tảng học tập.

Kết hợp các nguồn dữ liệu này sẽ cung cấp một cái nhìn toàn diện về quá trình học tập và kết quả của sinh viên, giúp xác định các điểm mạnh và điểm yếu cũng như đề xuất các phương pháp cải thiện. [2]

### **2.1.2. Chỉ số**

**Định nghĩa chỉ số:** Chỉ số là các biện pháp cụ thể định lượng các khía cạnh của quá trình học tập. Ví dụ, chỉ số có thể bao gồm tỷ lệ hoàn thành khóa học, thời gian hoàn thành các nhiệm vụ, mức độ tham gia vào các hoạt động học tập, và điểm trung bình của sinh viên.

**Công cụ và Kỹ thuật:** Các công cụ phân tích dữ liệu và kỹ thuật khai thác dữ liệu được sử dụng để xử lý và chuyển đổi dữ liệu thô thành các chỉ số. Ví dụ, sử dụng SQL để truy vấn dữ liệu từ cơ sở dữ liệu, hoặc sử dụng các công cụ phân tích dữ liệu như R hoặc Python để tính toán và hiển thị các chỉ số.

Ví dụ: Một hệ thống có thể tính toán thời gian trung bình sinh viên dành cho mỗi tài liệu học tập và so sánh với thời gian dự kiến để xác định mức độ cam kết và hiệu quả của tài liệu đó.

### **2.1.3. Phân tích dữ liệu**

Sau khi thu thập dữ liệu, bước tiếp theo là phân tích các thông tin thu thập được để đưa ra những kết luận và đề xuất cụ thể:

**Phân tích định lượng:** Sử dụng các phương pháp thống kê để phân tích điểm số, tỷ lệ tham gia và các dữ liệu định lượng khác. Các công cụ như Excel, SPSS, hoặc R có thể được sử dụng để thực hiện các phân tích này.

**Phân tích định tính:** Đánh giá các phản hồi từ khảo sát và phỏng vấn để tìm ra những yếu tố ảnh hưởng đến kết quả học tập. Các kỹ thuật phân tích định tính như phân tích nội dung (content analysis) hoặc phân tích chủ đề (thematic analysis) có thể được áp dụng.

**Phân tích xu hướng:** Xem xét các xu hướng trong dữ liệu qua các học kỳ hoặc năm học để nhận diện những thay đổi trong kết quả học tập của sinh viên.

#### **2.1.4. Trực quan hóa dữ liệu**

Trực quan hóa dữ liệu là quá trình chuyển đổi dữ liệu phân tích thành các biểu đồ và đồ thị dễ hiểu, giúp người dùng dễ dàng nhận diện các mẫu và xu hướng. Các công cụ thường dùng bao gồm:

Biểu đồ cột và biểu đồ thanh: So sánh kết quả học tập giữa các nhóm sinh viên hoặc giữa các môn học khác nhau.

Biểu đồ đường: Theo dõi sự thay đổi của điểm số qua thời gian.

Biểu đồ tròn: Minh họa tỷ lệ phân bố của các hạng mục dữ liệu khác nhau, như tỷ lệ điểm số giữa các mức điểm.

Biểu đồ radar: còn được gọi là biểu đồ mạng nhện, là một công cụ trực quan hóa dữ liệu mạnh mẽ giúp thể hiện đa chiều thông tin một cách rõ ràng và trực quan.

Các công cụ như rechart, chartjs có thể được sử dụng để tạo các biểu đồ và báo cáo trực quan, cung cấp thông tin dễ hiểu và hữu ích cho việc ra quyết định. [3]

#### **2.1.5. Áp dụng vào kết quả học tập**

Kết quả phân tích dữ liệu học tập có thể được áp dụng vào nhiều khía cạnh của quá trình giáo dục để nâng cao chất lượng học tập:

Cải tiến phương pháp giảng dạy: Dựa trên phản hồi và phân tích dữ liệu, giảng viên có thể điều chỉnh phương pháp giảng dạy để đáp ứng tốt hơn nhu cầu của sinh viên.

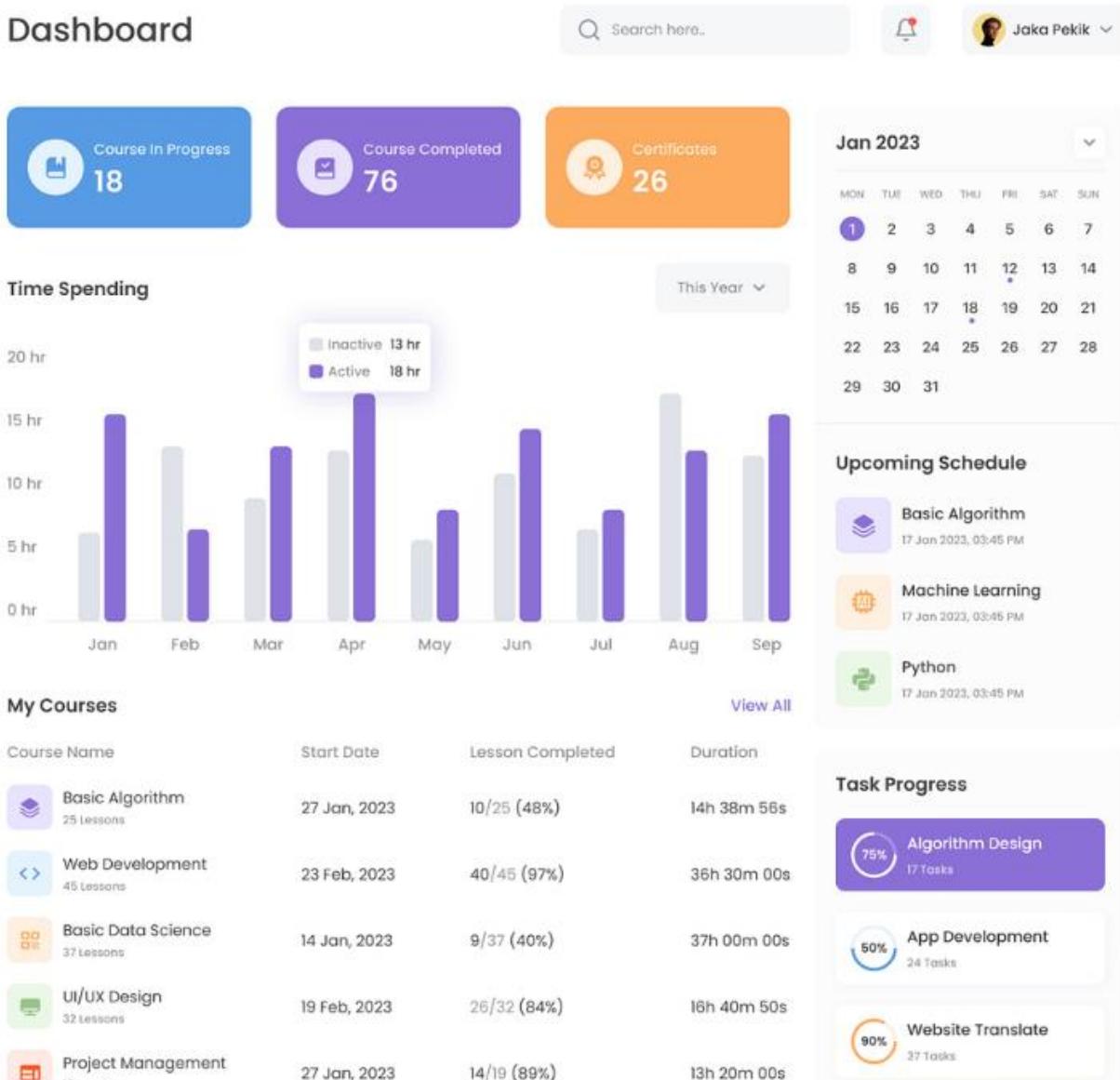
Phát hiện và hỗ trợ sinh viên gặp khó khăn: Nhận diện những sinh viên có kết quả học tập kém để cung cấp hỗ trợ kịp thời, như tư vấn học tập hoặc các khóa học bổ trợ.

Thiết kế lại chương trình học: Dựa trên phân tích xu hướng và phản hồi, các nhà quản lý giáo dục có thể điều chỉnh nội dung và cấu trúc chương trình học để nâng cao hiệu quả giảng dạy. [4]

### **2.2. Dashboard thể hiện kết quả học tập**

Dashboard là một công cụ quan trọng trong việc trực quan hóa kết quả học tập, cho phép người dùng dễ dàng theo dõi và phân tích dữ liệu thông qua các biểu đồ và báo cáo tương tác. Dưới đây là các thành phần cơ bản và hướng dẫn xây dựng một dashboard hiệu quả để thể hiện kết quả học tập của sinh viên.

## Dashboard



Hình 2.2. Giao diện dashboard điển hình.

### 2.2.1. Thành phần chính của dashboard

Một dashboard hiệu quả để thể hiện kết quả học tập của sinh viên thường bao gồm các thành phần sau:

**Tổng quan:** Một cái nhìn tổng quát về kết quả học tập hiện tại, bao gồm các chỉ số chính như điểm trung bình chung, tỷ lệ hoàn thành môn học, và tỷ lệ sinh viên đạt và không đạt.

**Biểu đồ xu hướng:** Biểu đồ đường hoặc biểu đồ cột hiển thị xu hướng điểm số hoặc tỷ lệ hoàn thành theo thời gian, giúp nhận diện các mô hình thay đổi và các yếu tố ảnh hưởng.

**Phân tích theo nhóm:** Biểu đồ phân tích kết quả học tập theo các nhóm khác nhau

như giới tính, độ tuổi, ngành học, hoặc lớp học. Điều này giúp phát hiện các điểm khác biệt và đề xuất các biện pháp hỗ trợ cụ thể.

**Phân tích môn học:** Biểu đồ cột hoặc biểu đồ thanh thể hiện kết quả học tập của sinh viên theo từng môn học, giúp nhận diện các môn học có tỷ lệ sinh viên gấp khó khăn cao.

**Biểu đồ phân phối:** Biểu đồ tròn hoặc bản đồ nhiệt thể hiện phân phối điểm số, giúp nhận diện các khu vực có tỷ lệ điểm số cao hoặc thấp. [2]

### **2.2.2. Thiết kế giao diện người dùng (UI) của Dashboard**

Thiết kế giao diện người dùng là một yếu tố quan trọng trong việc xây dựng dashboard, đảm bảo rằng thông tin được trình bày một cách rõ ràng và dễ hiểu:

**Bố cục hợp lý:** Sắp xếp các thành phần trên dashboard theo một cấu trúc hợp lý, giúp người dùng dễ dàng tìm kiếm và phân tích thông tin.

**Sử dụng màu sắc:** Sử dụng màu sắc hợp lý để phân biệt các loại dữ liệu và nhấn mạnh các thông tin quan trọng, nhưng tránh sử dụng quá nhiều màu sắc gây rối mắt.

**Tương tác:** Cho phép người dùng tương tác với các biểu đồ, như chọn khoảng thời gian, lọc dữ liệu theo nhóm, hoặc xem chi tiết thông tin khi di chuột qua.

**Tối ưu hóa hiển thị:** Đảm bảo rằng dashboard có thể hiển thị tốt trên các thiết bị khác nhau, từ máy tính để bàn đến điện thoại di động. [2]

## **2.3. Các công nghệ cần thiết để thực hiện đồ án**

Để xây dựng một hệ thống trực quan hóa kết quả học tập hiệu quả cho sinh viên công nghệ thông tin, cần sử dụng nhiều công nghệ khác nhau, từ backend, frontend cho đến cơ sở dữ liệu và các công cụ trực quan hóa. Dưới đây là các công nghệ cần thiết cho từng phần của hệ thống.

### **2.3.1. Công nghệ backend**

Backend là một thành phần phía server, chịu trách nhiệm xử lý dữ liệu, giao tiếp với cơ sở dữ liệu và cung cấp các API cho frontend.

### **2.3.1.1. NodeJS**

Node.js là một nền tảng chạy trên môi trường JavaScript, được xây dựng trên nền tảng V8 JavaScript Engine của Google Chrome. Được phát triển bởi Ryan Dahl vào năm 2009, Node.js đã trở thành một công cụ mạnh mẽ và phổ biến cho việc phát triển ứng dụng web và các dịch vụ máy chủ.



#### **Lịch sử phát triển**

Node.js ra đời: Node.js được phát triển bởi Ryan Dahl vào năm 2009. Mục tiêu ban đầu là tạo ra một nền tảng cho phép JavaScript chạy trên phía server.

Sự phát triển: Node.js nhanh chóng trở thành một trong những nền tảng phổ biến nhất cho phát triển web server-side, được hỗ trợ bởi cộng đồng mã nguồn mở lớn mạnh.

Các phiên bản chính: Các phiên bản LTS (Long-Term Support) của Node.js luôn được cập nhật và cải tiến để đáp ứng nhu cầu của cộng đồng và các công ty.

#### **Kiến trúc và thành phần**

Kiến trúc sự kiện không đồng bộ (Event-driven architecture): Node.js sử dụng mô hình sự kiện không đồng bộ giúp xử lý nhiều kết nối đồng thời mà không cần tạo ra nhiều thread.

V8 Engine: Node.js được xây dựng trên V8 JavaScript Engine của Google, cung cấp hiệu suất cao và thực thi mã JavaScript nhanh chóng.

Thư viện lõi: Bao gồm các module cơ bản như HTTP, File System (fs), Path, Events, và nhiều module khác hỗ trợ phát triển ứng dụng. [5]

#### **Các tính năng nổi bật**

Hiệu suất cao: Nhờ vào mô hình I/O không đồng bộ và không có blocking, Node.js có thể xử lý nhiều kết nối cùng lúc với hiệu suất cao.

Khả năng mở rộng: Dễ dàng mở rộng ứng dụng bằng cách thêm các worker process

để xử lý tải công việc cao.

Ecosystem rộng lớn: Với npm các nhà phát triển có thể truy cập hàng nghìn gói (packages) và module để tích hợp vào dự án của mình.

Dễ học và sử dụng: Sử dụng cùng một ngôn ngữ (JavaScript) cho cả front-end và back-end, giúp giảm bớt khó khăn trong việc học và triển khai.

### Các ứng dụng của Node.js

Web Server: Tạo các web server hiệu quả, có thể xử lý hàng triệu kết nối đồng thời.

RESTful API: Phát triển các API cho các ứng dụng web và mobile.

Ứng dụng thời gian thực: Đặc biệt hiệu quả trong các ứng dụng cần cập nhật thời gian thực như chat, game online, và các công cụ cộng tác.

Microservices: Thích hợp cho kiến trúc microservices, giúp chia nhỏ ứng dụng thành các dịch vụ nhỏ dễ quản lý.

#### 2.3.1.2. ExpressJs

Express là một framework web tối giản cho NodeJS, giúp xây dựng các ứng dụng web và API một cách nhanh chóng và dễ dàng. Express cung cấp các tính năng mạnh mẽ như định tuyến, middleware, và hỗ trợ các phương thức HTTP.



#### Lịch sử phát triển

Ra đời: Express.js được phát triển bởi TJ Holowaychuk và phát hành lần đầu tiên vào tháng 11 năm 2010.

Mục tiêu: Được tạo ra như một framework tối giản, giúp đơn giản hóa việc xây dựng các ứng dụng web và API bằng Node.js.

Sự phát triển: Express.js nhanh chóng trở thành một trong những framework phổ biến nhất cho Node.js nhờ sự đơn giản, linh hoạt và hiệu quả.

## Kiến trúc và thành phần

Middleware: Kiến trúc của Express.js dựa trên khái niệm middleware, các hàm trung gian xử lý yêu cầu (request) và phản hồi (response) theo từng bước.

Router: Express.js có hệ thống routing mạnh mẽ, cho phép định nghĩa các route và xử lý các phương thức HTTP khác nhau như GET, POST, PUT, DELETE.

Template Engines: Hỗ trợ nhiều template engines như Pug, EJS, và Handlebars, giúp tạo ra các trang web động dễ dàng.

## Các tính năng nổi bật

Đơn giản và linh hoạt: Express.js cung cấp một cách tiếp cận đơn giản và linh hoạt để xây dựng các ứng dụng web và API.

Hiệu suất cao: Tối ưu hóa cho hiệu suất cao, đặc biệt khi kết hợp với các công cụ và thư viện của Node.js.

Hệ sinh thái mạnh mẽ: Tích hợp tốt với nhiều thư viện và module khác trong hệ sinh thái của Node.js, giúp mở rộng chức năng dễ dàng.

Hỗ trợ middleware phong phú: Có nhiều middleware có sẵn để xử lý các tác vụ thông thường như xác thực (authentication), logging, parsing body request, v.v. [6]

## Các ứng dụng của Express.js

RESTful API: Rất phổ biến trong việc xây dựng các API RESTful, cung cấp các endpoint để tương tác với các ứng dụng khác.

Ứng dụng web: Sử dụng để xây dựng các ứng dụng web từ đơn giản đến phức tạp, với khả năng tạo ra các trang web động.

Ứng dụng thời gian thực: Khi kết hợp với các công nghệ như WebSockets, Express.js có thể tạo ra các ứng dụng thời gian thực như chat, game online.

Microservices: Được sử dụng rộng rãi trong kiến trúc microservices, giúp chia nhỏ ứng dụng thành các dịch vụ nhỏ gọn và dễ quản lý.

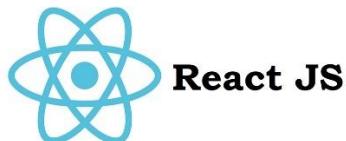
### 2.3.2. Công nghệ Frontend

Frontend là phần giao diện người dùng của hệ thống, nơi mà người dùng tương tác

trực tiếp. Các công nghệ chính sử dụng trong phần frontend bao gồm React, TailwindCSS, Firebase, NextUI, Ant Design (AntD), Recharts, và Chart.js.

### 2.3.2.1. ReactJS

ReactJS là một thư viện JavaScript dùng để xây dựng giao diện người dùng, được phát triển bởi Facebook. React giúp xây dựng các ứng dụng web phức tạp thông qua việc tạo ra các component UI có thể tái sử dụng.



Dưới đây là một số đặc điểm chính của React:

Component-based Architecture: React cho phép xây dựng các ứng dụng bằng cách tạo ra các component nhỏ, có thể tái sử dụng. Mỗi component quản lý trạng thái và logic riêng, giúp tăng tính module hóa và dễ bảo trì.

Virtual DOM: React sử dụng một bản sao ảo của DOM thật (Virtual DOM) để tối ưu hóa hiệu suất. Khi trạng thái của một component thay đổi, React sẽ cập nhật Virtual DOM trước và chỉ thay đổi các phần cần thiết của DOM thật, giảm thiểu thao tác trên DOM và tăng tốc độ ứng dụng.

One-way data binding: dữ liệu trong React chảy theo một hướng từ cha xuống con, giúp dễ dàng theo dõi và quản lý trạng thái ứng dụng.

JSX: JSX là một cú pháp mở rộng cho phép viết HTML trong JavaScript, làm cho việc phát triển giao diện người dùng trực quan hơn.

Ecosystem and community: react có một hệ sinh thái phong phú và cộng đồng phát triển lớn, cung cấp nhiều thư viện và công cụ hỗ trợ như Redux, React Router, và nhiều thư viện UI khác. [7]

### 2.3.2.2. TailwindCSS

TailwindCSS là một framework CSS tiện dụng cho phép tùy biến giao diện người dùng một cách linh hoạt mà không cần phải viết nhiều CSS. TailwindCSS giúp các nhà

phát triển dễ dàng thiết kế các trang web đẹp và nhất quán.



Dưới đây là một số đặc điểm chính của TailwindCSS:

Utility-first CSS: TailwindCSS cung cấp các lớp CSS nhỏ (utility classes) để xây dựng giao diện trực tiếp trong HTML. Điều này giúp tăng tốc quá trình phát triển và giảm bớt việc viết CSS tùy chỉnh.

Customization: tailwindCSS dễ dàng tùy biến thông qua tệp cấu hình, cho phép thay đổi màu sắc, kích thước, và các giá trị khác để phù hợp với thiết kế của dự án.

Responsive design: tailwindCSS hỗ trợ thiết kế đáp ứng (responsive) một cách dễ dàng với các lớp tiện ích cho các breakpoint khác nhau.

Integration: tailwindCSS tích hợp tốt với các framework và thư viện JavaScript như React, Vue, và Angular, giúp dễ dàng áp dụng vào các dự án hiện có.

Performance: tailwindCSS tối ưu hóa hiệu suất bằng cách loại bỏ các lớp không sử dụng khi build, giảm kích thước tệp CSS cuối cùng.

### 2.3.2.3. Firebase

Firebase là một nền tảng phát triển ứng dụng web và di động của Google, cung cấp nhiều dịch vụ như cơ sở dữ liệu, xác thực, lưu trữ tệp, và hosting.



Các dịch vụ chính của Firebase bao gồm:

Firebase realtime database và firestore: Cung cấp cơ sở dữ liệu NoSQL với khả năng đồng bộ dữ liệu thời gian thực.

Firebase authentication: cung cấp các phương thức xác thực người dùng dễ dàng tích hợp như email/password, OAuth, và xác thực qua tài khoản mạng xã hội.

Firebase storage: lưu trữ và phục vụ tệp tin, đặc biệt hữu ích cho việc quản lý các tệp phương tiện.

Firebase hosting: cung cấp dịch vụ hosting tĩnh an toàn và nhanh chóng cho các ứng dụng web.

Firebase analytics: cung cấp các công cụ phân tích mạnh mẽ để theo dõi hành vi người dùng và hiệu suất ứng dụng.

#### 2.3.2.4. NextUI

NextUI là một thư viện component UI dành cho React, cung cấp các component sẵn sàng sử dụng với thiết kế đẹp và dễ tùy biến.



NextUI giúp tăng tốc quá trình phát triển giao diện người dùng với các đặc điểm sau:

Component đa dạng: Cung cấp nhiều component UI cơ bản và nâng cao, giúp xây dựng giao diện người dùng nhanh chóng.

Tùy biến dễ dàng: Hỗ trợ tùy biến giao diện dễ dàng với các props và theme.

Hiệu suất cao: Được tối ưu hóa cho hiệu suất, giúp các ứng dụng React chạy mượt mà.

#### 2.3.2.5. Ant Design (AntD)

Ant Design (AntD) là một thư viện component UI dành cho React, được phát triển bởi Alibaba. AntD cung cấp một hệ thống component phong phú và thiết kế chuyên nghiệp.



Các đặc điểm chính của AntD bao gồm:

Component phong phú: Cung cấp hàng trăm component sẵn sàng sử dụng cho các nhu cầu khác nhau, từ các form điều khiển đến các bảng dữ liệu phức tạp.

Thiết kế nhất quán: Tuân theo các quy tắc thiết kế nhất quán, giúp tạo ra các giao diện người dùng đẹp và nhất quán.

Tích hợp tốt: Dễ dàng tích hợp với các thư viện và công cụ khác như Redux, React Router.

### **2.3.2.6. Recharts**

Recharts là một thư viện vẽ biểu đồ dành cho React, được xây dựng trên D3.js. Recharts cung cấp các biểu đồ đơn giản và dễ sử dụng, giúp trực quan hóa dữ liệu một cách hiệu quả.



Các đặc điểm chính của Recharts bao gồm:

Component biểu đồ: Cung cấp nhiều loại biểu đồ như biểu đồ cột, biểu đồ đường, biểu đồ tròn, biểu đồ radar, và nhiều loại khác.

Dễ sử dụng: API thân thiện và dễ sử dụng cho các nhà phát triển React.

Tùy biến cao: Cho phép tùy biến các component biểu đồ để phù hợp với yêu cầu cụ thể của ứng dụng.

### **2.3.2.6. Chart.js**

Chart.js là một thư viện JavaScript đơn giản và linh hoạt để vẽ biểu đồ. Chart.js hỗ trợ nhiều loại biểu đồ khác nhau và có thể dễ dàng tích hợp vào các ứng dụng React.



Các đặc điểm chính của Chart.js bao gồm:

Biểu đồ đa dạng: Hỗ trợ nhiều loại biểu đồ như biểu đồ đường, biểu đồ cột, biểu đồ tròn, biểu đồ radar, biểu đồ bong bóng, và nhiều loại khác.

Dễ tích hợp: Dễ dàng tích hợp với các ứng dụng React thông qua các wrapper như react-chartjs-2.

Tùy biến: Cung cấp nhiều tùy chọn để tùy biến các biểu đồ, từ màu sắc, kích thước đến các hiệu ứng.

### **2.3.3. Hệ quản trị cơ sở dữ liệu MySql**

MySQL là một hệ quản trị cơ sở dữ liệu quan hệ (RDBMS) mã nguồn mở, phổ biến nhất hiện nay. Được phát triển bởi công ty MySQL AB, sau đó được Sun Microsystems mua lại và hiện nay thuộc sở hữu của Oracle Corporation. MySQL được sử dụng rộng rãi trong các ứng dụng web, đặc biệt là các ứng dụng dựa trên nền tảng LAMP (Linux, Apache, MySQL, PHP/Python/Perl). [8]

#### **Lịch sử phát triển**

MySQL được phát triển lần đầu tiên vào năm 1995 bởi Michael Widenius (Monty), David Axmark và Allan Larsson. Ban đầu, nó được thiết kế như một công cụ quản lý cơ sở dữ liệu nhẹ, dễ sử dụng và có hiệu suất cao. Qua nhiều năm, MySQL đã phát triển mạnh mẽ và trở thành một trong những hệ quản trị cơ sở dữ liệu phổ biến nhất trên thế giới.

#### **Kiến trúc và thành phần**

MySQL có kiến trúc client-server, trong đó MySQL server là thành phần chính, chịu trách nhiệm quản lý các cơ sở dữ liệu, xử lý các truy vấn SQL và tương tác với các ứng dụng client.

- MySQL Server: Đây là thành phần chính của MySQL, chịu trách nhiệm quản lý dữ liệu, xử lý các lệnh SQL, và duy trì tính toàn vẹn của dữ liệu.
- MySQL Client: Đây là các công cụ hoặc ứng dụng kết nối với MySQL server để thực hiện các thao tác trên cơ sở dữ liệu.
- Storage Engines: MySQL hỗ trợ nhiều loại storage engines khác nhau như InnoDB, MyISAM, MEMORY, và nhiều loại khác. Mỗi loại storage engine có những đặc điểm và ứng dụng riêng. [8]

#### **Các tính năng nổi bật**

MySQL cung cấp nhiều tính năng nổi bật như:

- Hiệu suất cao: MySQL được tối ưu hóa cho các ứng dụng web và các hệ thống cần xử lý nhiều truy vấn cùng lúc.
- Bảo mật: MySQL cung cấp nhiều cơ chế bảo mật như quản lý người dùng, phân quyền truy cập, và mã hóa dữ liệu.

- Độ tin cậy: MySQL hỗ trợ tính năng backup và recovery, giúp đảm bảo tính toàn vẹn và an toàn của dữ liệu.

- Mã nguồn mở: MySQL là phần mềm mã nguồn mở, cho phép người dùng tự do tải về, sử dụng, và chỉnh sửa mã nguồn. [8]

## Ứng dụng của MySQL

MySQL được sử dụng rộng rãi trong nhiều ứng dụng khác nhau, từ các website nhỏ đến các hệ thống lớn của các công ty công nghệ hàng đầu như Facebook, Twitter, và YouTube. Nó cũng là thành phần chính trong nhiều hệ thống quản lý nội dung (CMS) như WordPress, Joomla, và Drupal.

### 2.3.4. Môi trường triển khai ứng dụng (VPS)

VPS, viết tắt của Virtual Private Server, là một dịch vụ lưu trữ web sử dụng công nghệ ảo hóa để cung cấp các tài nguyên máy chủ riêng biệt trên một máy chủ vật lý. VPS là một lựa chọn phổ biến cho các trang web có lượng truy cập trung bình đến cao, cần nhiều tài nguyên hơn so với dịch vụ lưu trữ chia sẻ nhưng không yêu cầu một máy chủ vật lý riêng.

### Lịch sử phát triển

Dịch vụ VPS xuất hiện như một giải pháp trung gian giữa lưu trữ chia sẻ (shared hosting) và máy chủ riêng (dedicated server). Với sự phát triển của công nghệ ảo hóa, VPS trở nên phổ biến vào những năm 2000, mang lại sự linh hoạt và hiệu suất cao cho người dùng mà không phải chịu chi phí lớn như máy chủ riêng.

### Cách hoạt động của VPS

VPS hoạt động dựa trên công nghệ ảo hóa, chia một máy chủ vật lý thành nhiều máy chủ ảo độc lập. Mỗi VPS có hệ điều hành riêng và có quyền truy cập vào một phần tài nguyên của máy chủ vật lý, bao gồm CPU, RAM, và dung lượng đĩa.

Máy chủ vật lý (Host Machine): Đây là máy chủ thực tế, chứa các tài nguyên vật lý như CPU, RAM, và ổ cứng.

Công nghệ ảo hóa: Phần mềm ảo hóa (như KVM, VMware, hoặc Hyper-V) tạo ra các máy chủ ảo trên máy chủ vật lý.

Máy chủ ảo (VPS): Mỗi VPS hoạt động như một máy chủ riêng biệt, với hệ điều

hành và tài nguyên riêng, không bị ảnh hưởng bởi các VPS khác trên cùng máy chủ vật lý.

## Các loại VPS

Có nhiều loại VPS khác nhau, tùy thuộc vào công nghệ ảo hóa và cách thức quản lý:

**Managed VPS:** Nhà cung cấp dịch vụ quản lý hoàn toàn VPS, bao gồm bảo mật, cập nhật phần mềm, và hỗ trợ kỹ thuật.

**Unmanaged VPS:** Người dùng tự quản lý VPS, từ cài đặt phần mềm đến bảo mật và bảo trì hệ thống.

**Cloud VPS:** VPS được triển khai trên một nền tảng đám mây, cho phép mở rộng tài nguyên linh hoạt và tính sẵn sàng cao.

## Lợi ích của việc sử dụng VPS

Sử dụng VPS mang lại nhiều lợi ích so với lưu trữ chia sẻ và máy chủ riêng:

- Hiệu suất cao: VPS cung cấp tài nguyên riêng biệt, giúp đảm bảo hiệu suất ổn định cho trang web hoặc ứng dụng.
- Tự do tùy chỉnh: Người dùng có toàn quyền truy cập root và có thể cài đặt bất kỳ phần mềm hoặc hệ điều hành nào họ muốn.
- Bảo mật tốt hơn: VPS cung cấp môi trường riêng biệt, giảm nguy cơ bị ảnh hưởng bởi các trang web hoặc ứng dụng khác trên cùng máy chủ.
- Khả năng mở rộng: Tài nguyên của VPS có thể được mở rộng dễ dàng mà không cần di chuyển dữ liệu hoặc cấu hình lại hệ thống.

## Các ứng dụng của VPS

VPS được sử dụng rộng rãi trong nhiều trường hợp khác nhau, bao gồm:

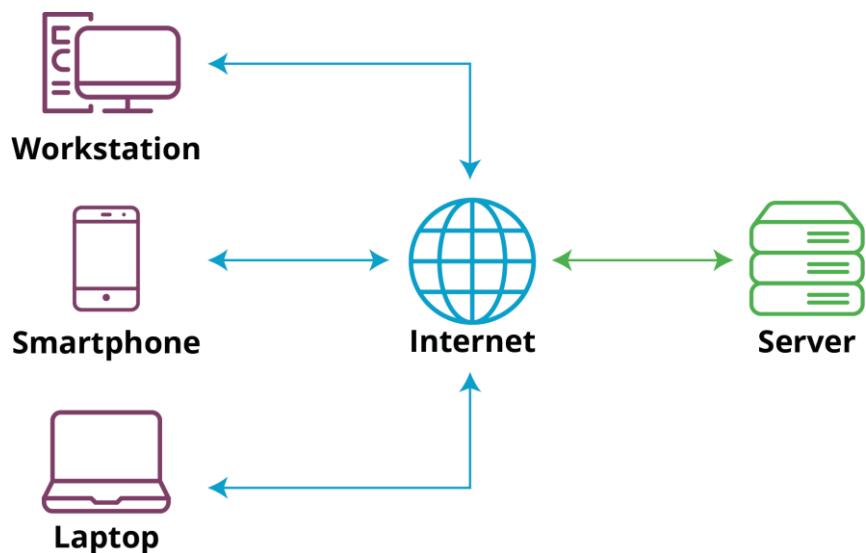
- Lưu trữ website: VPS phù hợp cho các trang web có lượng truy cập lớn hoặc yêu cầu tài nguyên cao.
- Lưu trữ ứng dụng: VPS cung cấp môi trường lý tưởng để chạy các ứng dụng web hoặc phần mềm doanh nghiệp.
- Phát triển và thử nghiệm: VPS cung cấp môi trường linh hoạt cho việc phát triển và thử nghiệm các ứng dụng phần mềm.

- Máy chủ game: VPS có thể được sử dụng để chạy các máy chủ game, cung cấp hiệu suất và độ ổn định cao cho người chơi.

## 2.4. Kiến trúc ứng dụng

### 2.4.1. Kiến trúc client-server

Kiến trúc client-server là một mô hình phổ biến trong phát triển ứng dụng web, cho phép phân chia rõ ràng giữa giao diện người dùng và logic xử lý nghiệp vụ. Trong mô hình này, client và server là hai thành phần chính hoạt động tương tác với nhau thông qua mạng.



Hình 2.3. Mô tả kiến trúc client-server

Dưới đây là chi tiết về kiến trúc client-server và các khía cạnh liên quan:

#### Tổng quan về kiến trúc client-server

Trong kiến trúc client-server, client (máy khách) và server (máy chủ) là hai thực thể độc lập, mỗi thực thể đảm nhận những vai trò khác nhau:

- Client: Là các ứng dụng chạy trên máy của người dùng cuối (như trình duyệt web, ứng dụng di động). Client chịu trách nhiệm về giao diện người dùng, xử lý tương tác của người dùng, và gửi yêu cầu tới server.
- Server: Là các ứng dụng hoặc dịch vụ chạy trên máy chủ, xử lý các yêu cầu từ client, thực hiện logic nghiệp vụ, truy xuất dữ liệu từ cơ sở dữ liệu, và gửi phản hồi lại cho client. [6]

#### Cách thức hoạt động của kiến trúc client-server

**Client gửi yêu cầu (Request):** Người dùng tương tác với ứng dụng client (ví dụ: nhấn nút, điền vào biểu mẫu) và client gửi yêu cầu HTTP đến server.

**Server nhận yêu cầu:** Server nhận yêu cầu và phân tích nó để xác định hành động cần thực hiện.

**Xử lý yêu cầu:** Server thực hiện các logic nghiệp vụ cần thiết, có thể bao gồm truy xuất hoặc cập nhật dữ liệu trong cơ sở dữ liệu.

**Gửi phản hồi (Response):** Sau khi xử lý xong, server gửi phản hồi (thường dưới dạng HTML, JSON hoặc XML) trở lại cho client.

**Hiển thị phản hồi:** Client nhận phản hồi từ server và cập nhật giao diện người dùng dựa trên dữ liệu nhận được.

### **Lợi ích của kiến trúc client-server**

**Phân tách nhiệm vụ:** Client và server có thể được phát triển, triển khai, và nâng cấp độc lập. Điều này giúp cải thiện khả năng quản lý và bảo trì ứng dụng.

**Khả năng mở rộng:** Server có thể được mở rộng (scale) để xử lý nhiều yêu cầu hơn bằng cách thêm nhiều máy chủ hơn hoặc sử dụng các kỹ thuật cân bằng tải (load balancing).

**Bảo mật:** Kiến trúc client-server giúp bảo vệ dữ liệu nhạy cảm bằng cách lưu trữ và xử lý nó trên server, nơi mà các biện pháp bảo mật mạnh mẽ hơn có thể được áp dụng.

**Hiệu suất:** Các tác vụ nặng về tính toán và xử lý dữ liệu có thể được thực hiện trên server, giúp giảm tải cho client và cải thiện hiệu suất ứng dụng.

### **Thách thức của kiến trúc client-server**

**Độ trễ mạng:** Giao tiếp giữa client và server phụ thuộc vào mạng, có thể gây ra độ trễ trong phản hồi nếu kết nối mạng không ổn định hoặc chậm.

**Phụ thuộc vào server:** Nếu server gặp sự cố hoặc ngừng hoạt động, client sẽ không thể thực hiện được các chức năng yêu cầu truy cập server.

**Bảo mật truyền thông:** Dữ liệu truyền qua mạng giữa client và server có thể bị tấn công nếu không được mã hóa đúng cách.

## Các thành phần chính trong kiến trúc client-server

### Client-side (Phía client):

Giao diện người dùng (UI): Được xây dựng bằng HTML, CSS, và JavaScript (hoặc các framework như React, Angular, Vue).

Logic xử lý client: Xử lý sự kiện, xác thực dữ liệu trước khi gửi yêu cầu đến server.

Giao tiếp với server: Sử dụng AJAX, Fetch API hoặc thư viện như Axios để gửi yêu cầu HTTP đến server.

### Server-side (Phía server):

Web server: Quản lý các yêu cầu HTTP từ client và định tuyến chúng đến các dịch vụ phù hợp (ví dụ: Apache, Nginx).

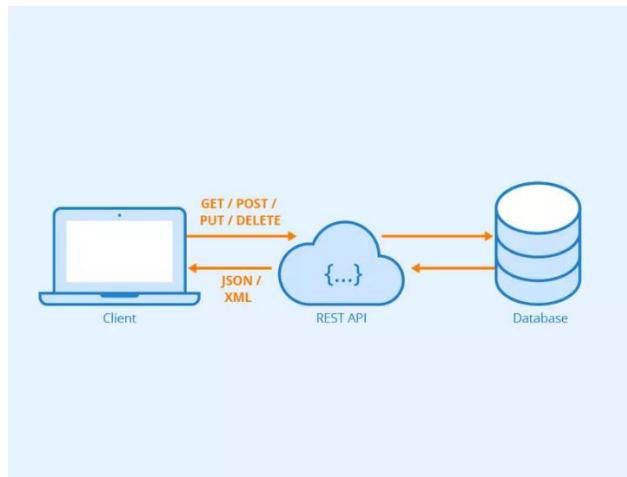
Ứng dụng server: Xử lý logic nghiệp vụ, có thể được xây dựng bằng các ngôn ngữ và framework như NodeJS với Express, Django, hoặc Ruby on Rails.

Cơ sở dữ liệu: Lưu trữ dữ liệu và cung cấp các dịch vụ truy xuất dữ liệu (ví dụ: MySQL, PostgreSQL, MongoDB).

### 2.4.2. Giao tiếp giữa front-end và back-end RESTful API

#### Tìm hiểu về RESTful API

Giao tiếp giữa front-end và back-end thông qua RESTful API là một phương pháp phổ biến để xây dựng các ứng dụng web hiện đại. RESTful API sử dụng các phương thức HTTP để truyền tải dữ liệu giữa client (front-end) và server (back-end).



Hình 2.4. Cách thức hoạt động của RESTful API

## Các nguyên tắc của RESTful API

Giao diện thống nhất (Uniform Interface): Giao diện đồng nhất là yếu tố cốt lõi của REST. Điều này đảm bảo rằng các tương tác giữa client và server diễn ra thông qua một giao diện duy nhất.

Stateless: Mỗi yêu cầu từ client đến server phải chứa tất cả thông tin cần thiết để server hiểu và xử lý yêu cầu. Server không lưu trữ trạng thái client giữa các yêu cầu.

Khả năng lưu trữ (Cacheable): Các phản hồi từ server có thể được đánh dấu là có thể lưu trữ hoặc không. Nếu một phản hồi có thể lưu trữ, client có thể lưu trữ nó để sử dụng lại trong tương lai.

Hệ thống phân tầng (Layered System): Kiến trúc REST có thể được tổ chức thành các lớp, mỗi lớp có trách nhiệm riêng, và client không biết rằng nó đang giao tiếp với server trực tiếp hay thông qua một lớp trung gian. [5]

## Các thành phần chính của RESTful API

Resources: Mọi thứ trong REST đều là resources. Một resource có thể là một đối tượng dữ liệu, một tập hợp các đối tượng, hoặc bất kỳ thứ gì có thể được đại diện bằng một URL.

URL: Mỗi resource trong REST được định danh bằng một URL duy nhất. URL đại diện cho đường dẫn tới resource trên server.

HTTP Methods: RESTful API sử dụng các phương thức HTTP để thực hiện các thao tác khác nhau trên resource. Các phương thức chính bao gồm:

GET: Lấy dữ liệu từ server.

POST: Gửi dữ liệu mới tới server để tạo resource mới.

PUT: Cập nhật dữ liệu của một resource hiện có.

DELETE: Xóa một resource.

## Các bước xây dựng RESTful API

Xác định resources: Bắt đầu bằng việc xác định các resources mà API sẽ quản lý. Mỗi resource nên đại diện cho một thực thể cụ thể.

Định nghĩa URL cho mỗi resource: Tạo URL duy nhất cho mỗi resource. URL nên

có cấu trúc rõ ràng và dễ hiểu.

Xác định HTTP methods cho từng thao tác: Gán các phương thức HTTP phù hợp cho từng thao tác trên resources. Ví dụ, sử dụng GET để lấy thông tin resource, POST để tạo resource mới, v.v.

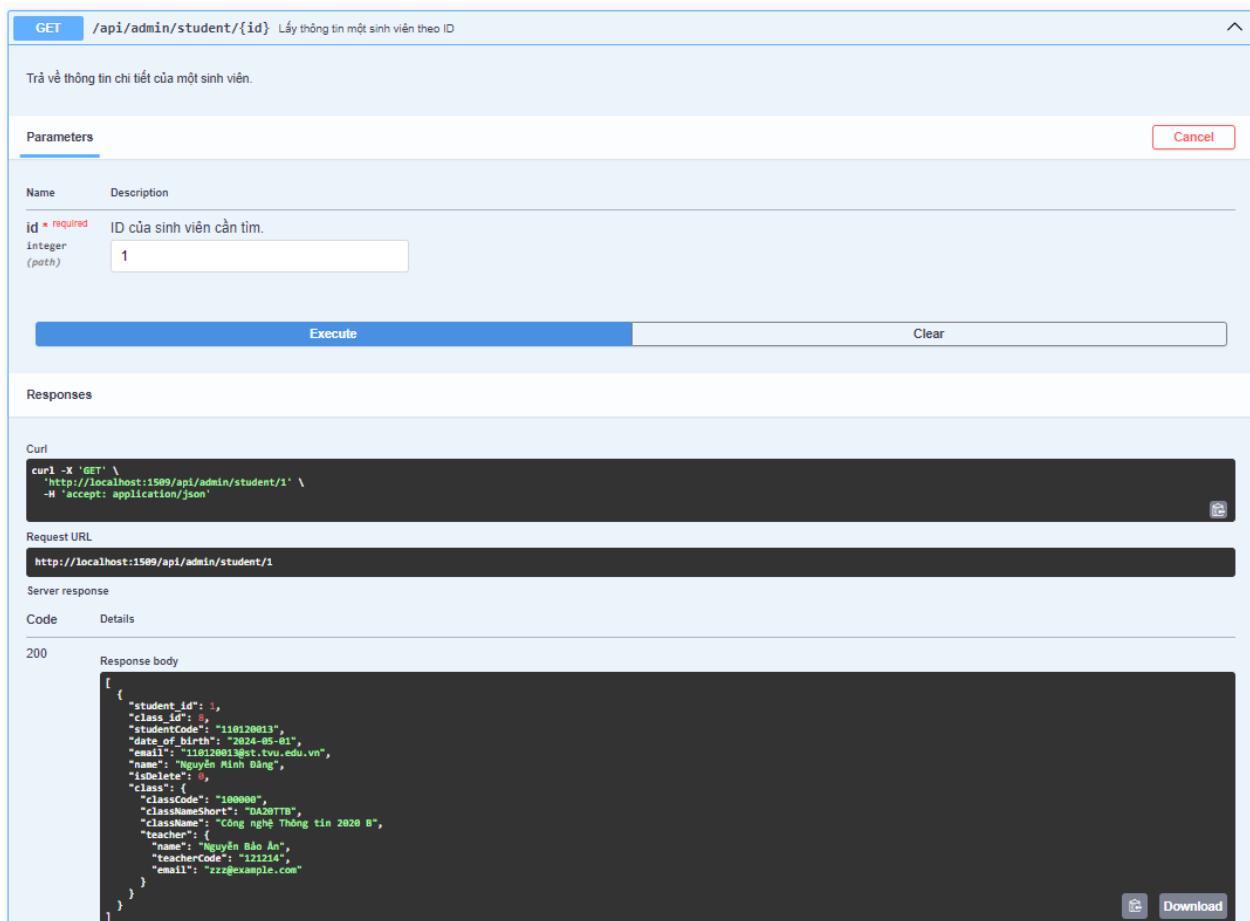
Thiết kế phản hồi của server: Xác định định dạng phản hồi của server (ví dụ: JSON, XML) và đảm bảo rằng phản hồi chứa tất cả thông tin cần thiết.

Bảo mật API: Áp dụng các biện pháp bảo mật như xác thực, phân quyền, và mã hóa để bảo vệ API khỏi các mối đe dọa tiềm ẩn. [6]

### 2.4.3. Quản lý và làm tài liệu API với Swagger

## Tìm hiểu về Swagger

Swagger là một bộ công cụ mã nguồn mở giúp thiết kế, xây dựng, và tài liệu hóa các API RESTful. Nó cung cấp các công cụ để tự động tạo ra tài liệu API, giúp cho việc tương tác với API trở nên dễ dàng và nhất quán. Swagger hiện là một phần của OpenAPI Specification (OAS), tiêu chuẩn được chấp nhận rộng rãi trong việc mô tả các API RESTful.



Hình 2.5. Ví dụ về API thao tác trên swagger

## Lịch sử phát triển

Swagger được tạo ra vào năm 2011 bởi Tony Tam khi làm việc tại Wordnik. Mục đích ban đầu của Swagger là để giúp tự động hóa việc tạo tài liệu cho các API. Với sự phát triển của các API và nhu cầu về việc tài liệu hóa rõ ràng và nhất quán, Swagger nhanh chóng trở nên phổ biến và được tích hợp vào nhiều dự án lớn. Năm 2015, Swagger trở thành một phần của OpenAPI Initiative, một tổ chức dưới sự quản lý của Linux Foundation.

## Cách hoạt động của Swagger

Swagger sử dụng các định nghĩa dựa trên JSON hoặc YAML để mô tả cấu trúc và hành vi của các API RESTful. Các định nghĩa này sau đó có thể được sử dụng để tạo ra tài liệu API động, tạo mã nguồn máy khách (client) và máy chủ (server), và cung cấp giao diện người dùng cho việc thử nghiệm API.

**Định nghĩa API:** Các định nghĩa API được viết dưới dạng file JSON hoặc YAML, mô tả các endpoint, phương thức HTTP, kiểu dữ liệu, và các thông tin liên quan khác.

**Công cụ Swagger:** Sử dụng các công cụ như Swagger Editor, Swagger UI, và Swagger Codegen để tạo tài liệu, giao diện thử nghiệm, và mã nguồn từ các định nghĩa API.

**Giao diện người dùng:** Swagger UI cung cấp một giao diện người dùng trực quan để tương tác và thử nghiệm các API trực tiếp từ trình duyệt.

## Các tính năng nổi bật

Swagger cung cấp nhiều tính năng mạnh mẽ và hữu ích, bao gồm:

**Tài liệu API động:** Tự động tạo ra tài liệu API từ các định nghĩa, giúp dễ dàng chia sẻ và duy trì.

**Giao diện thử nghiệm:** Swagger UI cung cấp một giao diện thử nghiệm tương tác, cho phép người dùng thử nghiệm các endpoint API trực tiếp từ trình duyệt.

**Tạo mã nguồn:** Swagger Codegen tạo mã nguồn cho các ứng dụng máy khách và máy chủ từ các định nghĩa API, giúp giảm thời gian phát triển.

**Tích hợp dễ dàng:** Swagger có thể được tích hợp với nhiều ngôn ngữ lập trình và framework khác nhau, bao gồm Node.js, Java, Python, và nhiều ngôn ngữ khác.

## Ứng dụng của Swagger

Swagger được sử dụng rộng rãi trong việc phát triển và tài liệu hóa các API RESTful. Một số ứng dụng phổ biến của Swagger bao gồm:

Phát triển API: Tạo và duy trì tài liệu API nhất quán, giúp giảm thiểu lỗi và tăng cường sự hiểu biết giữa các thành viên trong nhóm phát triển.

Thử nghiệm API: Sử dụng Swagger UI để thử nghiệm các endpoint API một cách dễ dàng và hiệu quả.

Tạo mã nguồn tự động: Sử dụng Swagger Codegen để tự động tạo mã nguồn cho các ứng dụng máy khách và máy chủ, giúp tiết kiệm thời gian và công sức trong quá trình phát triển.

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

### **3.1. Mô tả bài toán**

Đánh giá chất lượng của một chương trình đào tạo thường xoay quanh việc xem xét mức độ đạt được của sinh viên đối với các chuẩn đầu ra (PLO - Program Learning Outcomes). Đây là tiêu chí quan trọng giúp các đơn vị giáo viên và nhà quản lý giáo dục xác định được hiệu quả của chương trình đào tạo, từ đó đưa ra những điều chỉnh cần thiết để nâng cao chất lượng giáo dục.

Để xây dựng một chương trình đào tạo chất lượng, các đơn vị giáo viên cần lập danh sách các mục tiêu chương trình (PO - Program Objectives). Những mục tiêu này sẽ đặt nền tảng cho việc xác định các chuẩn đầu ra của chương trình đào tạo (PLO). Từ danh sách các PLO, một loạt các học phần sẽ được thiết kế để đóng góp vào việc đào tạo kiến thức, kỹ năng và thái độ cho sinh viên nhằm đạt được các PLO này.

Mỗi học phần trong chương trình đào tạo có một danh sách các chuẩn đầu ra của môn học (CLO - Course Learning Objectives) và danh sách các chương (bài) học. Do đó, cần có một bảng ánh xạ giữa CLO và PLO để đảm bảo rằng các CLO đều hướng tới việc thỏa mãn các PLO. Khi đánh giá môn học, giảng viên sẽ sử dụng các rubric để xác định liệu sinh viên có đạt được các CLO hay không. Việc này cho phép thực hiện các thao tác thống kê để tính toán mức độ thỏa mãn PLO của sinh viên.

Thông tin về mức độ thỏa mãn chuẩn đầu ra của sinh viên mang lại nhiều tín hiệu quý giá, giúp cả nhà trường, khoa, giảng viên và sinh viên thực hiện những điều chỉnh cần thiết để nâng cao chất lượng đào tạo.

Nhà trường có thể sử dụng dữ liệu này để đánh giá chất lượng tổng thể của chương trình đào tạo, từ đó lập kế hoạch chiến lược phát triển và cải tiến chương trình giảng dạy.

Các khoa có thể đánh giá chất lượng ở mức chương trình, điều chỉnh chương trình đào tạo để đáp ứng tốt hơn nhu cầu của sinh viên và xã hội. Thông tin này cũng giúp khoa có căn cứ khoa học để chứng minh chất lượng đào tạo với các bên liên quan.

Giảng viên có thể nhận được thông tin chi tiết về chất lượng giảng dạy ở cấp độ môn học, hiểu được những khó khăn mà sinh viên gặp phải, từ đó điều chỉnh phương pháp

giảng dạy để hỗ trợ sinh viên tốt hơn.

Sinh viên có thể tự phản ánh về năng lực học tập của mình, so sánh với các bạn cùng lớp, từ đó đặt ra những mục tiêu cụ thể để cố gắng hơn. Điều này giúp sinh viên chủ động hơn trong quá trình học tập và phát triển bản thân.

Nếu có một hệ thống trực quan hóa mức độ đạt chuẩn đầu ra của sinh viên, sẽ mang lại hiệu quả rất lớn trong việc nâng cao chất lượng giáo dục. Hệ thống này có thể hiển thị các dữ liệu về mức độ đạt được của sinh viên đối với các CLO và PLO dưới dạng biểu đồ, bảng biểu, giúp các bên liên quan dễ dàng theo dõi và phân tích.

Hệ thống trực quan hóa kết quả học tập cần có những đặc điểm sau:

Giao diện thân thiện: Dễ sử dụng cho tất cả các bên liên quan.

Dữ liệu cập nhật: Thông tin được cập nhật liên tục, phản ánh chính xác quá trình học tập của sinh viên.

Báo cáo chi tiết: Cung cấp các báo cáo chi tiết về từng CLO và PLO, giúp nhận diện những khu vực cần cải thiện.

Tính năng so sánh: Cho phép so sánh kết quả học tập giữa các sinh viên, các lớp học và các năm học.

Lợi ích của hệ thống trực quan hóa kết quả học tập:

Nhà trường: Có thể nhanh chóng đánh giá chất lượng chương trình đào tạo và lập kế hoạch cải tiến.

Khoa: Dễ dàng theo dõi và điều chỉnh chương trình đào tạo theo thời gian thực.

Giảng viên: Hiểu rõ hơn về hiệu quả giảng dạy của mình và những khó khăn mà sinh viên gặp phải.

Sinh viên: Năm bắt được tiến độ học tập của mình, từ đó điều chỉnh phương pháp học tập cho phù hợp.

Đánh giá chất lượng đào tạo thông qua các chuẩn đầu ra là một quy trình quan trọng và cần thiết trong giáo dục. Việc xây dựng một hệ thống trực quan hóa kết quả học tập của sinh viên không chỉ giúp nâng cao chất lượng giáo dục mà còn hỗ trợ các bên liên quan trong việc điều chỉnh và cải tiến chương trình đào tạo. Hệ thống này sẽ là công cụ hữu hiệu để nâng

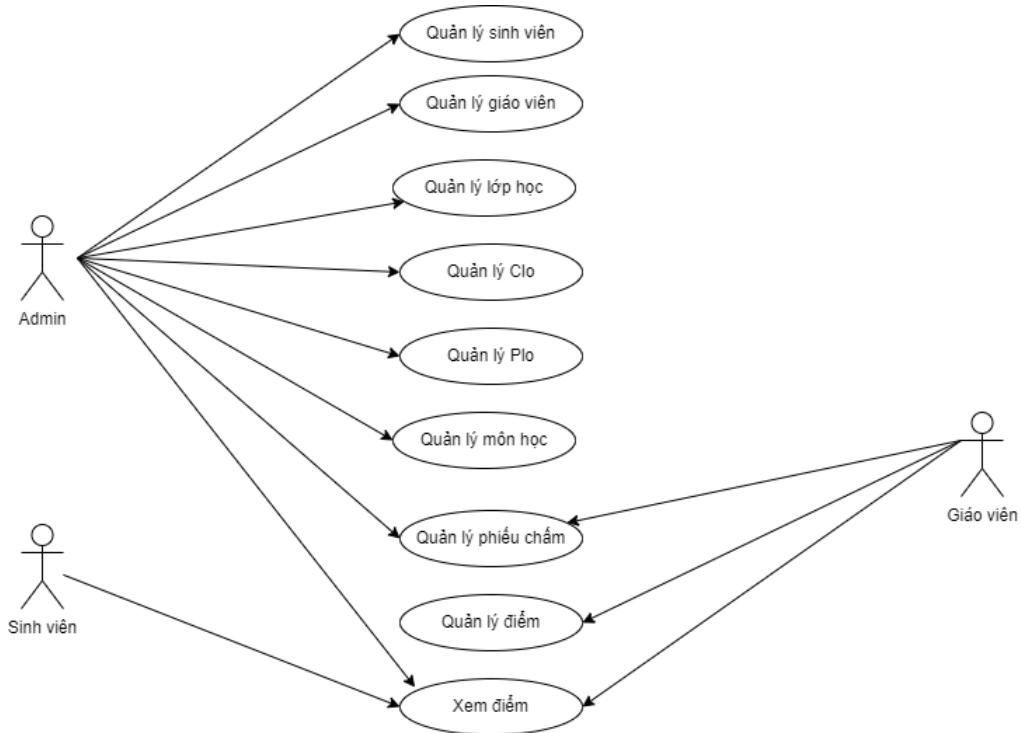
cao hiệu quả giảng dạy và học tập, góp phần vào sự phát triển bền vững của nền giáo dục.

### **3.2. Yêu cầu chức năng**

Để xây dựng hệ thống trực quan hóa kết quả học tập sinh viên CNTT, cần một số yêu cầu chức năng cụ thể để đáp ứng nhu cầu người dùng:

- Hiển thị kết quả học tập: Hiển thị toàn bộ kết quả học tập của sinh viên theo từng môn học, học kỳ.
- Lọc kết quả học tập theo tiêu chí: Cho phép người dùng lọc kết quả học tập theo môn học, học kỳ, hoặc điểm số.
- Tìm kiếm sinh viên theo tên: Người dùng có thể tìm kiếm thông tin kết quả học tập của sinh viên dựa trên tên hoặc mã sinh viên.
- Tìm kiếm kết quả theo môn học: Người dùng có thể tìm kiếm kết quả học tập theo tên môn học hoặc mã môn học.
- Xem thông tin chi tiết: Cung cấp thông tin chi tiết về kết quả học tập của từng sinh viên, bao gồm: mã sinh viên, tên sinh viên, môn học, học kỳ, điểm số, và mô tả chi tiết về kết quả.
- Biểu đồ và đồ thị trực quan: Hiển thị biểu đồ và đồ thị trực quan về tiến độ và hiệu suất học tập của sinh viên, bao gồm các loại biểu đồ như biểu đồ đường, biểu đồ cột, biểu đồ tròn, và biểu đồ radar.
- Báo cáo tổng hợp: Cung cấp các báo cáo tổng hợp về kết quả học tập của sinh viên theo từng lớp học, học kỳ, năm học.
- Chức năng xuất dữ liệu: Cho phép xuất dữ liệu kết quả học tập dưới dạng các file như Excel, PDF để tiện lưu trữ và chia sẻ.

### 3.3. Đặt tả hệ thống



Hình 3.1 Sơ đồ use case

Bảng 3.1. Mô tả Actor

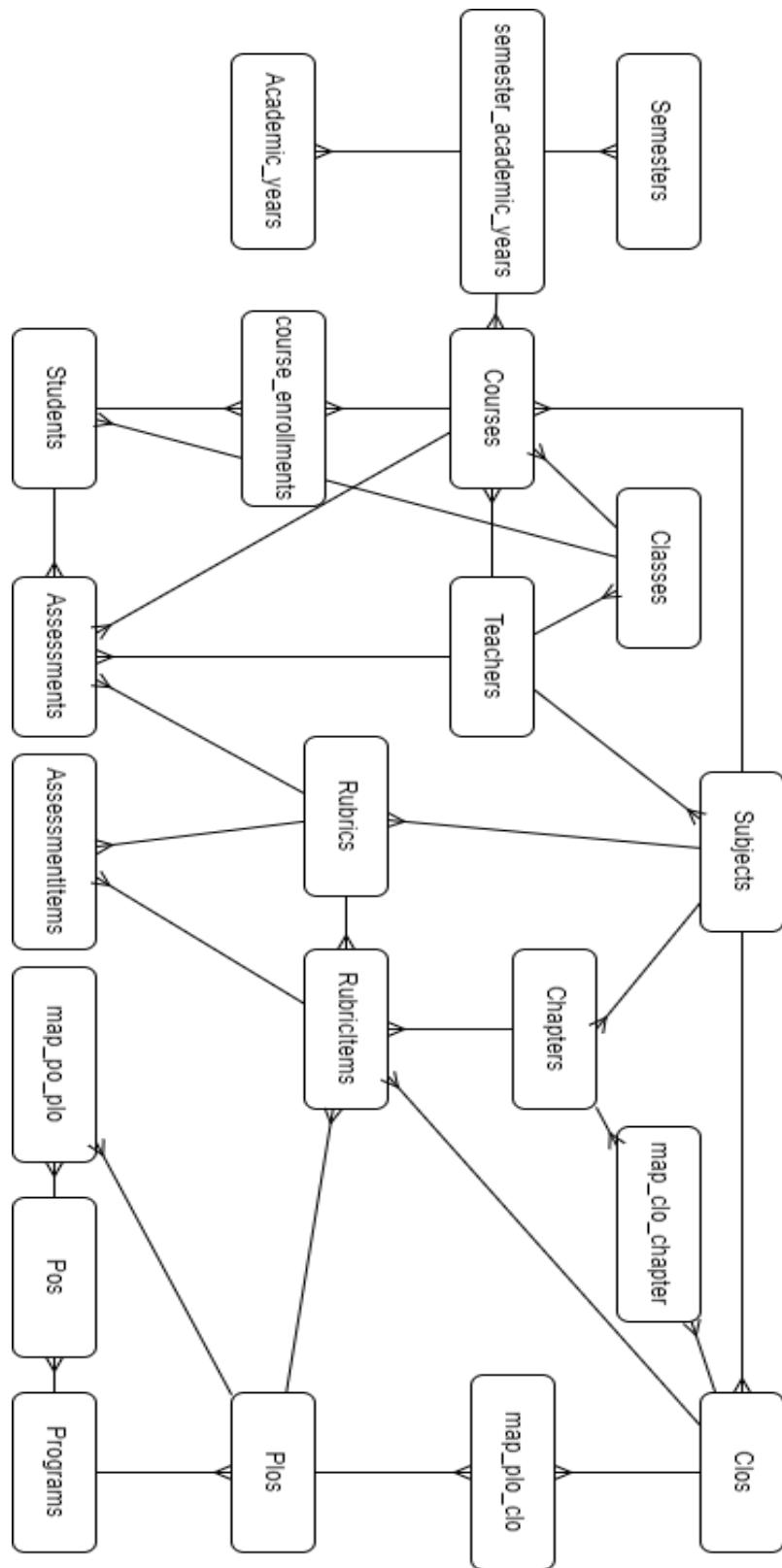
STT	Tên Actor	Ý nghĩa
1	Admin	Người quản trị hệ thống
2	Giáo viên	Giáo viên
3	Sinh viên	Sinh viên

Bảng 3.2 Mô tả use case

STT	Tên Use-case	Ý Nghĩa
1	Quản lý sinh viên	Quản trị viên có thể quản lý thông tin của các sinh viên thêm, sửa, xoá nếu cần thiết.
2	Quản lý giáo viên	Quản trị viên có thể quản lý thông tin của giáo viên trên trang web và thêm, sửa, xoá nếu cần thiết.
3	Quản lý lớp học	Quản trị viên có thể quản lý thông tin của lớp web và thêm, sửa, xoá nếu cần thiết.
4	Quản lý Clo	Quản trị viên có thể quản lý chuẩn đầu ra học phần và thêm, sửa, xoá nếu cần thiết.

5	Quản lý Plo	Quản trị viên có thể quản lý thông tin chuẩn đầu ra đào tạo và thêm, sửa, xoá nếu cần thiết.
6	Quản lý môn học	Quản trị viên có thể quản lý thông tin của môn học trên trang web và thêm, sửa, xoá nếu cần thiết.
7	Quản lý phiếu chấm	Quản trị viên hoặc giáo viên cáo thê tạo phiếu chấm.
8	Quản lý điểm	Giáo viên có thể tìm kiếm, lọc các tiêu chí trên hệ thống để xem kết quả học tập của sinh viên
9	Xem điểm	Xem kết quả học tập của sinh viên

### 3.4. Lược đồ dữ liệu



Hình 3.2. Lược đồ cơ sở dữ liệu

## Mô tả các bảng:

Bảng 3.3. Mô tả bảng Student

STT	Tên thuộc tính	Loại	Ràng buộc	Ý nghĩa/ghi chú
1	student_id	int	khóa chính	mã khóa sinh viên
2	teacher_id	int	khóa ngoại	mã khóa giáo viên
3	studentCode	varchar(9)	duy nhất	mã sinh viên
4	email	varchar(255)		email của sinh viên
5	name	varchar(255)		tên sinh viên
6	date_of_birth	date		ngày sinh
7	isDelete	tinyint		ẩn sinh viên
8	createdAt	timestamp		ngày tạo
9	updatedAt	timestamp		ngày cập nhật

Bảng 3.4. Mô tả bảng Teacher

STT	Tên thuộc tính	Loại	Ràng buộc	Ý nghĩa/ghi chú
1	teacher_id	int	khóa ngoại	mã khóa giáo viên
2	name	varchar(255)		tên giáo viên
3	teacherCode	varchar(20)	duy nhất	mã giáo viên
4	email	varchar(255)	duy nhất	email của giáo viên
5	password	varchar(255)		mật khẩu
6	permission	tinyint		phân quyền

7	imgURL	text		link ảnh hình đại diện
8	isBlock	tityint		chặn giáo viên
9	typeTeacher	enum('GVCV', 'GVGD')		loại giáo viên
10	isDelete	tinyint		ẩn giáo viên
11	createdAt	timestamp		ngày tạo
12	updatedAt	timestamp		ngày cập nhật

Bảng 3.5. Môn tả bảng Subjects

STT	Tên thuộc tính	Loại	Ràng buộc	Ý nghĩa/ghi chú
1	subject_id	int	khóa chính	mã khóa môn học
2	teacher_id	int	khóa ngoại	mã khóa giáo viên
3	subjectName	text		tên môn học
4	subjectCode	varchar(6)	duy nhất	mã môn học
5	description	text		mô tả khóa học
6	munberCredits	int		số tính chỉ
7	numberCreditsTheory	int		số tính chỉ lý thuyết
8	numberCreditsPractice	int		số tính chỉ thực hành
9	typesubject	enum('Đại cương','Cơ sở ngành','Chuyên')		loại môn học

		'ngành','Thực tập và Đồ án')		
10	isDelete	tinyint		ẩn môn học
11	createdAt	timestamp		ngày tạo
12	updatedAt	timestamp		ngày cập nhật

Bảng 3.6. Mô tả bảng Courses

STT	Tên thuộc tính	Loại	Ràng buộc	Ý nghĩa/ghi chú
1	course_id	Int	khóa chính	mã khóa khóa học
2	teacher_id	int	khóa ngoại	mã khóa giáo viên
3	class_id	int	khóa ngoại	mã khóa lớp
4	subject_id	int	khóa ngoại	mã khóa môn học
5	id_semester_academic_year	int	khóa ngoại	mã khóa học kì năm học
6	courseName	text		tên khóa học
7	courseCode	varchar(20)	duy nhất	mã khoán học
8	isDelete	tinyint		ẩn khóa học
9	createdAt	timestamp		ngày tạo
10	updatedAt	timestamp		ngày cập nhật

Bảng 3.7. Mô tả bảng academic\_year

STT	Tên thuộc tính	Loại	Ràng buộc	Ý nghĩa/ghi chú
-----	----------------	------	-----------	-----------------

1	course_id	int	khóa chính	mã khóa khóa học
2	startDate	timestamp		năm bắt đầu
3	endDate	timestamp		năm kết thúc
4	description	text		mô tả năm học
5	isDelete	tinyint		ẩn năm học
6	createdAt	timestamp		ngày tạo
7	updatedAt	timestamp		ngày cập nhật

Bảng 3.8. Mô tả bảng Semesters

STT	Tên thuộc tính	Loại	Ràng buộc	Ý nghĩa/ghi chú
1	semester_id	int	khóa chính	mã khóa học kì
2	descriptionShort	text		mô tả ngắn
3	descriptionLong	text		mô tả dài
4	codeSemester	varchar(20)		mã năm học
5	isDelete	tinyint		ẩn năm học
6	createdAt	timestamp		ngày tạo
7	updatedAt	timestamp		ngày cập nhật

Bảng 3.9. Mô tả bảng Semester\_academic\_years

STT	Tên thuộc tính	Loại	Ràng buộc	Ý nghĩa/ghi chú
1	id_semester_academic_year	int	khóa chính	mã khóa năm học học kì

2	academic_year_id	int		mã khóa năm học
3	semester_id	int		mã khóa học kỳ
4	isDelete	tinyint		ẩn năm học học kỳ
5	createdAt	timestamp		ngày tạo
6	updatedAt	timestamp		ngày cập nhật

Bảng 3.10. Mô tả bảng Assessments

STT	Tên thuộc tính	Loại	Ràng buộc	Ý nghĩa/ghi chú
1	assessment_id	int	khóa chính	mã khóa lần đánh giá
2	teacher_id	int	khóa ngoại	mã khóa giáo viên
3	student_id	int	khóa ngoại	mã khóa lớp
4	rubric_id	int	khóa ngoại	mã khóa phiếu chấm
4	course_id	int	khóa ngoại	mã khóa khóa học
5	description	text		mô tả
6	totalScore	double(8,2)		điểm của lần chấm
7	date	date		ngày chấm
8	place	text		địa chỉ
9	isDelete	tinyint		ẩn khóa học
10	createdAt	timestamp		ngày tạo
11	updatedAt	timestamp		ngày cập nhật

Bảng 3.11. Mô tả bảng assessmentItems

STT	Tên thuộc tính	Loại	Ràng buộc	Ý nghĩa/ghi chú
1	assessmentItem_id	int	khóa chính	mã khóa mục đánh giá
2	assessment_id	int	khóa ngoại	mã khóa lần đánh giá
3	rubricsItem_id	int	khóa ngoại	mã khóa mục phiếu chấm
4	assessmentScore	double(8,2)		điểm mục đánh giá
5	isDelete	tinyint		ẩn mục đánh giá
6	createdAt	timestamp		ngày tạo
7	updatedAt	timestamp		ngày cập nhật

Bảng 3.12. Mô tả bảng Chapters

STT	Tên thuộc tính	Loại	Ràng buộc	Ý nghĩa/ghi chú
1	chapter_id	int	khóa chính	mã khóa chương
2	subject_id	int	khóa ngoại	mã khóa môn học
3	chapterName	varchar(100)		tên chương học
4	description	text		Mô tả chương
5	isDelete	tinyint		ẩn chương
6	createdAt	timestamp		ngày tạo

7	updatedAt	timestamp		ngày cập nhật
---	-----------	-----------	--	---------------

Bảng 3.13. Mô tả bảng Classes

STT	Tên thuộc tính	Loại	Ràng buộc	Ý nghĩa/ghi chú
1	class_id	int	khóa chính	mã khóa lớp
2	teacher_id	int	khóa ngoại	mã khóa giáo viên
3	className	varchar(255)		tên lớp
4	classNameShort	varchar(20)	duy nhất	tên lớp ngắn
5	classCode	varchar(10)	duy nhất	mã lớp
6	isDelete	tinyint		ẩn lớp
7	createdAt	timestamp		ngày tạo
8	updatedAt	timestamp		ngày cập nhật

Bảng 3.14. Mô tả bảng Clos

STT	Tên thuộc tính	Loại	Ràng buộc	Ý nghĩa/ghi chú
1	clo_id	int	khóa chính	mã khóa CDR môn học
2	subject_id	int	khóa ngoại	mã khóa môn học
3	cloName	varchar(20)		tên CDR môn học
4	description	text		Mô tả CDR môn học
5	isDelete	tinyint		ẩn CDR môn học
6	createdAt	timestamp		ngày tạo

7	updatedAt	timestamp		ngày cập nhật
---	-----------	-----------	--	---------------

Bảng 3.15. Mô tả bảng Plos

STT	Tên thuộc tính	Loại	Ràng buộc	Ý nghĩa/ghi chú
1	plo_id	int	khóa chính	mã khóa CDR chương trình
2	program_id	int	khóa ngoại	mã khóa chương trình
3	ploName	varchar(20)		tên CDR chương trình
4	description	text		Mô tả CDR chương trình
5	isDelete	tinyint		ẩn CDR chương trình
6	createdAt	timestamp		ngày tạo
7	updatedAt	timestamp		ngày cập nhật

Bảng 3.16. Mô tả bảng Pos

STT	Tên thuộc tính	Loại	Ràng buộc	Ý nghĩa/ghi chú
1	po_id	int	khóa chính	mã khóa mục tiêu chương trình đào tạo
2	program_id	int	khóa ngoại	mã khóa chương trình
3	poName	varchar(255)		tên mục tiêu chương trình đào tạo
4	description	text		Mô tả
5	isDelete	tinyint		ẩn mục tiêu chương trình đào tạo

6	createdAt	timestamp		ngày tạo
7	updatedAt	timestamp		ngày cập nhật

Bảng 3.17. Mô tả bảng course\_enrollment

STT	Tên thuộc tính	Loại	Ràng buộc	Ý nghĩa/ghi chú
1	id_detail_courses	int	khóa chính	mã khóa course_enrollment
2	student_id	int	khóa ngoại	mã khóa sinh viên
3	course_id	int	khóa ngoại	mã khóa khóa học
4	isDelete	tinyint		ẩn course_enrollment
5	createdAt	timestamp		ngày tạo
6	updatedAt	timestamp		ngày cập nhật

Bảng 3.18. Mô tả bảng Programs

STT	Tên thuộc tính	Loại	Ràng buộc	Ý nghĩa/ghi chú
1	program_id	int	khóa chính	mã khóa chương trình
2	programName	text		tên chương trình
3	description	text		mô tả chương trình
4	isDelete	tinyint		ẩn chương trình
5	createdAt	timestamp		ngày tạo
6	updatedAt	timestamp		ngày cập nhật

Bảng 3.19. Mô tả bảng Rubrics

STT	Tên thuộc tính	Loại	Ràng buộc	Ý nghĩa/ghi chú
1	rubric_id	int	khóa chính	mã khóa phiếu chấm
2	subject_id	int	khóa ngoại	mã khóa môn học
3	teacher_id	int	khóa ngoại	mã khóa giáo viên
4	rubricName	varchar(255)		tên phiếu chấm
5	comment	text		ghi chú
6	isDelete	tinyint		ẩn phiếu chấm
7	createdAt	timestamp		ngày tạo
8	updatedAt	timestamp		ngày cập nhật

Bảng 3.20. Mô tả bảng RubricItems

STT	Tên thuộc tính	Loại	Ràng buộc	Ý nghĩa/ghi chú
1	rubricsItem_id	int	khóa chính	mã khóa mục phiếu chấm
2	chapter_id	int	khóa ngoại	mã khóa sinh viên
3	clo_id	int	khóa ngoại	mã khóa khóa học
4	rubric_id	int	khóa ngoại	mã khóa phiếu chấm
5	plo_id	int	khóa ngoại	mã khóa CDR chương trình
6	description	text		mô tả
7	maxScore	double(8,2)		điểm mục phiếu chấm

8	isDelete	tinyint		ẩn mục phiếu chấm
9	createdAt	timestamp		ngày tạo
10	updatedAt	timestamp		ngày cập nhật

Bảng 3.21. Mô tả bảng map\_clo\_chapters

STT	Tên thuộc tính	Loại	Ràng buộc	Ý nghĩa/ghi chú
1	id_clo_chapter	int	khóa chính	mã khóa map_clo_chapter
2	clo_id	int	khóa ngoại	mã khóa CĐR môn học
3	chapter_id	int	khóa ngoại	mã khóa chương
4	isDelete	tinyint		ẩn
5	createdAt	timestamp		ngày tạo
6	updatedAt	timestamp		ngày cập nhật

Bảng 3.22. Mô tả bảng map\_plo\_clo

STT	Tên thuộc tính	Loại	Ràng buộc	Ý nghĩa/ghi chú
1	id_plo_clo	int	khóa chính	mã khóa map_plo_clo
2	clo_id	int	khóa ngoại	mã khóa CĐR môn học
3	plo_id	int	khóa ngoại	mã khóa CĐR chương trình
4	isDelete	tinyint		ẩn
5	createdAt	timestamp		ngày tạo
6	updatedAt	timestamp		ngày cập nhật

Bảng 3.23. Mô tả bảng map\_po\_plo

STT	Tên thuộc tính	Loại	Ràng buộc	Ý nghĩa/ghi chú
1	id_po_plo	int	khóa chính	mã khóa map_po_plo
2	clo_id	int	khóa ngoại	mã khóa CDR môn học
3	plo_id	int	khóa ngoại	mã khóa CDR chương trình
4	isDelete	tinyint		ẩn
5	createdAt	timestamp		ngày tạo
6	updatedAt	timestamp		ngày cập nhật

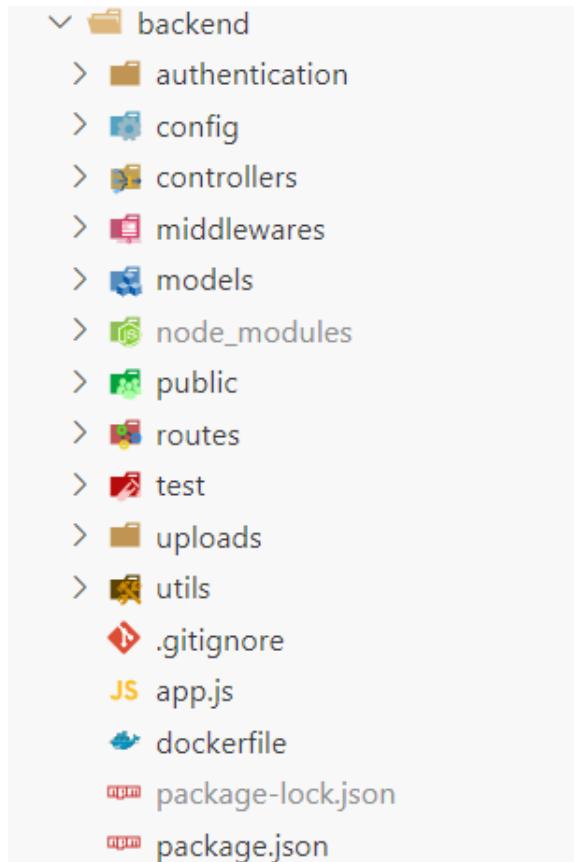
Bảng 3.24. Mô tả bảng refresh\_tokens

STT	Tên thuộc tính	Loại	Ràng buộc	Ý nghĩa/ghi chú
1	id	int	khóa chính	mã khóa
2	teacher_id	int	khóa ngoại	mã khóa giáo viên
2	token	varchar(255)		refresh token
3	expired	tinyint		có hết hạn
4	revoked	tinyint		có thu hồi
5	createdAt	timestamp		ngày tạo
6	updatedAt	timestamp		ngày cập nhật

## 3.5. Kiến trúc hệ thống

### 3.5.1. Backend

Backend là một thành phần phía server của hệ thống. Đối với hệ thống trực quan hóa kết quả học tập sinh viên CNTT sử dụng Express.js, backend được thiết kế để đảm bảo tính mở rộng, bảo mật, hiệu suất và dễ bảo trì.



Hình 3.3. Kiến trúc hệ thống backend

Dưới đây là một phân tích chi tiết về các thành phần chính của backend:

#### 3.5.1.1. Layered architecture

Backend được thiết kế theo kiến trúc lớp (layered architecture), với mỗi lớp có một vai trò cụ thể và rõ ràng:

- Presentation Layer: Xử lý các yêu cầu HTTP từ client, định tuyến các yêu cầu đến các controller tương ứng và trả về phản hồi. Lớp này bao gồm các routes và middleware.
- Business Logic Layer: Chứa các logic nghiệp vụ của ứng dụng. Lớp này được hiện

thực bởi các controller.

- Data Access Layer: Tương tác với cơ sở dữ liệu để truy xuất và lưu trữ dữ liệu. Lớp này được hiện thực bởi các models và các thao tác cơ sở dữ liệu.
- Utility Layer: Chứa các hàm và thư viện tiện ích dùng chung trong toàn bộ ứng dụng.

### **3.5.1.2. Thư mục chính và vai trò của chúng**

Dựa trên cấu trúc thư mục đã trình bày, dưới đây là mô tả chi tiết về từng thư mục và vai trò của chúng:

- authentication: Xử lý xác thực và phân quyền người dùng. Bao gồm các middleware như authMiddleware.js để bảo vệ các route.
- config: Chứa các file cấu hình như kết nối cơ sở dữ liệu, cấu hình môi trường (environment variables).
- controllers: Chứa các controller để xử lý logic nghiệp vụ cho các yêu cầu HTTP.
- middlewares: Chứa các middleware để xử lý các yêu cầu HTTP trước khi chúng đến controller.
- models: Chứa các mô hình dữ liệu (models) sử dụng ORM (Object-Relational Mapping) để tương tác với cơ sở dữ liệu.
- routes: Định nghĩa các route cho ứng dụng, mỗi route có thể liên kết với một controller cụ thể.
- utils: Chứa các hàm tiện ích dùng chung.
- app.js: File chính của ứng dụng, nơi cấu hình Express và kết nối các thành phần khác. Chứa các middleware chung, kết nối cơ sở dữ liệu, và khởi động server.

### **3.5.1.3. Quy trình xử lý yêu cầu HTTP**

Dưới đây là mô tả quy trình xử lý một yêu cầu HTTP từ khi nhận được đến khi phản hồi:

- Nhận yêu cầu: Khi một yêu cầu HTTP đến từ client, nó sẽ được tiếp nhận bởi app.js.
- Xử lý middleware: Yêu cầu sẽ đi qua các middleware toàn cục (như bodyParser, cors, authMiddleware) để xử lý các chức năng như phân tích body, xử lý CORS, và xác thực người dùng.

- Định tuyến: Sau khi qua các middleware, yêu cầu sẽ được định tuyến đến route tương ứng dựa trên URL và phương thức HTTP. Ví dụ: /api/users/:id sẽ được xử lý bởi userRoutes.js.
- Xử lý trong controller: Route sẽ gọi hàm tương ứng trong controller, ví dụ getUser trong userController.js, để thực hiện logic nghiệp vụ.
- Truy xuất dữ liệu: Controller có thể tương tác với model để truy xuất hoặc cập nhật dữ liệu trong cơ sở dữ liệu.
- Trả về phản hồi: Sau khi xử lý xong, controller sẽ gửi phản hồi về cho client qua HTTP response.

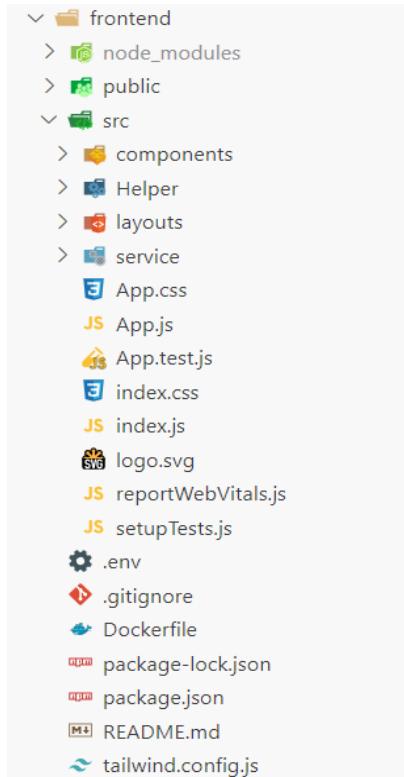
#### **3.5.1.4. Bảo mật và Quản lý lỗi**

- Bảo mật: sử dụng các middleware để xác thực và phân quyền người dùng, bảo vệ các tài nguyên nhạy cảm và đảm bảo rằng chỉ những người dùng có quyền mới có thể truy cập.
- Quản lý lỗi: sử dụng middleware để quản lý lỗi toàn cục, ghi log lỗi và trả về phản hồi lỗi có ý nghĩa cho client.

Backend được chọn với sự phân chia rõ ràng các thư mục và tệp tin theo chức năng, giúp dễ dàng quản lý, bảo trì và mở rộng. Các thành phần như authentication, config, controllers, middlewares, models, routes, và utils được tổ chức hợp lý, tối ưu hóa hiệu suất và đảm bảo an ninh. Điều này cho phép tái sử dụng mã nguồn, dễ dàng kiểm thử và quản lý các tài nguyên tĩnh, tệp tin tải lên. Các tệp tin như dockerfile và package.json hỗ trợ quản lý phiên bản và triển khai backend nhất quán trên mọi môi trường. Kiến trúc này cũng tăng tính linh hoạt trong phát triển, cho phép nhiều lập trình viên làm việc đồng thời mà không gây xung đột, đảm bảo sự ổn định và hiệu quả cho các dự án lớn.

### 3.5.2. Kiến trúc Frontend

Cấu trúc thư mục frontend của một ứng dụng React



Hình 3.4. Kiến trúc frontend

#### 3.5.2.1. Thư mục chính và vai trò của chúng

node\_modules: Chứa các gói npm được cài đặt. Thư mục này được tạo tự động khi chạy lệnh npm install hoặc yarn install.

public: Chứa các file tĩnh như hình ảnh, biểu tượng, và file index.html là điểm vào của ứng dụng.

src: Chứa mã nguồn chính của ứng dụng React.

Chi tiết các thư mục con trong src

components: Chứa các thành phần (components) của ứng dụng. Mỗi component thường đại diện cho một phần tử giao diện (UI) độc lập và có thể tái sử dụng.

Helper: Chứa các hàm tiện ích (utility functions) và các module hỗ trợ khác mà các component có thể sử dụng.

layouts: Chứa các layout components, đại diện cho bộ cục tổng thể của trang hoặc

các phần lớn của trang.

service: Chứa các service để làm việc với API hoặc các dịch vụ bên ngoài khác.

Các file quan trọng trong src

App.css: Chứa các style toàn cục cho ứng dụng.

App.js: Thành phần chính của ứng dụng, là nơi cấu hình các routes và điều hướng.

App.test.js: Chứa các test case cho thành phần App.js.

index.css: Chứa các style toàn cục khác có thể áp dụng cho ứng dụng.

index.js: Điểm vào của ứng dụng React. File này render component App vào DOM.

logo.svg: Logo của ứng dụng.

reportWebVitals.js: Đo lường hiệu suất của ứng dụng.

setupTests.js: Cấu hình cho các test case.

Các file cấu hình và môi trường

.env: Chứa các biến môi trường.

.gitignore: Liệt kê các file và thư mục sẽ bị Git bỏ qua.

Dockerfile: File cấu hình Docker để tạo container cho ứng dụng.

package-lock.json và package.json: Chứa các thông tin về các gói npm và các script của ứng dụng.

README.md: File hướng dẫn và tài liệu của dự án.

Kiến trúc này không chỉ giúp dễ dàng quản lý và bảo trì mã nguồn mà còn hỗ trợ tối ưu hóa hiệu suất và bảo mật. Việc phân chia rõ ràng theo chức năng giúp các lập trình viên dễ dàng làm việc đồng thời trên các phần khác nhau của ứng dụng mà không gây xung đột. Dockerfile và .env đảm bảo môi trường triển khai nhất quán, trong khi các tệp tin kiểm thử và cấu hình giúp duy trì chất lượng và khả năng mở rộng của ứng dụng.

## 3.6. Xây dựng Backend

### 3.6.1. Kết nối cơ sở dữ liệu

Quá trình kết nối dữ liệu trong Node.js với Sequelize và dotenv bao gồm các bước sau đây:

Import các thư viện cần thiết



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

Hình 3.5. Import các thư viện cần thiết kết nối cơ sở dữ liệu

Sequelize: Được import từ gói sequelize để sử dụng các tính năng ORM.

dotenv: Được import để quản lý các biến môi trường, giúp bảo mật thông tin nhạy cảm như thông tin đăng nhập cơ sở dữ liệu.

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



```
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,
  }
);
```

Hình 3.6. Tạo instance của Sequelize

Sequelize: Khởi tạo một instance của Sequelize với các tham số như tên cơ sở dữ liệu, tên người dùng, mật khẩu và các thông tin khác được lấy từ các biến môi trường.

process.env.DB\_NAME: Tên cơ sở dữ liệu.

process.env.DB\_USER: Tên người dùng cơ sở dữ liệu.

process.env.DB\_PASSWORD: Mật khẩu cơ sở dữ liệu.

process.env.DB\_HOST: Địa chỉ host của cơ sở dữ liệu.

process.env.DB\_DIALECT: Loại cơ sở dữ liệu (MySQL, PostgreSQL, SQLite, v.v.).

## Kiểm tra kết nối tới cơ sở dữ liệu

```
● ● ●  
async function testConnection() {  
  try {  
    await sequelize.authenticate();  
    console.log('Connection has been established successfully.');//  
  } catch (error) {  
    console.error('Unable to connect to the database:', error);  
  }  
}  
  
testConnection();
```

Hình 3.7. Kiểm tra kết nối với dữ liệu

`testConnection`: Hàm bắt đầu để kiểm tra kết nối tới cơ sở dữ liệu.

`sequelize.authenticate()`: Phương thức này kiểm tra xem kết nối tới cơ sở dữ liệu có thành công hay không.

`console.log`: In ra thông báo thành công nếu kết nối được thiết lập.

`console.error`: In ra lỗi nếu không thể kết nối tới cơ sở dữ liệu.

### 3.6.2. Định Nghĩa Mô Hình với Sequelize

Đoạn mã dưới đây là mô tả cách định nghĩa một mô hình sinh viên (`StudentModel`) sử dụng Sequelize ORM. Mô hình này bao gồm các thông tin chi tiết về sinh viên và mối quan hệ với một mô hình lớp học (`ClassModel`). Dưới đây là phân tích chi tiết từng phần của đoạn mã:

Import các thư viện cần thiết

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

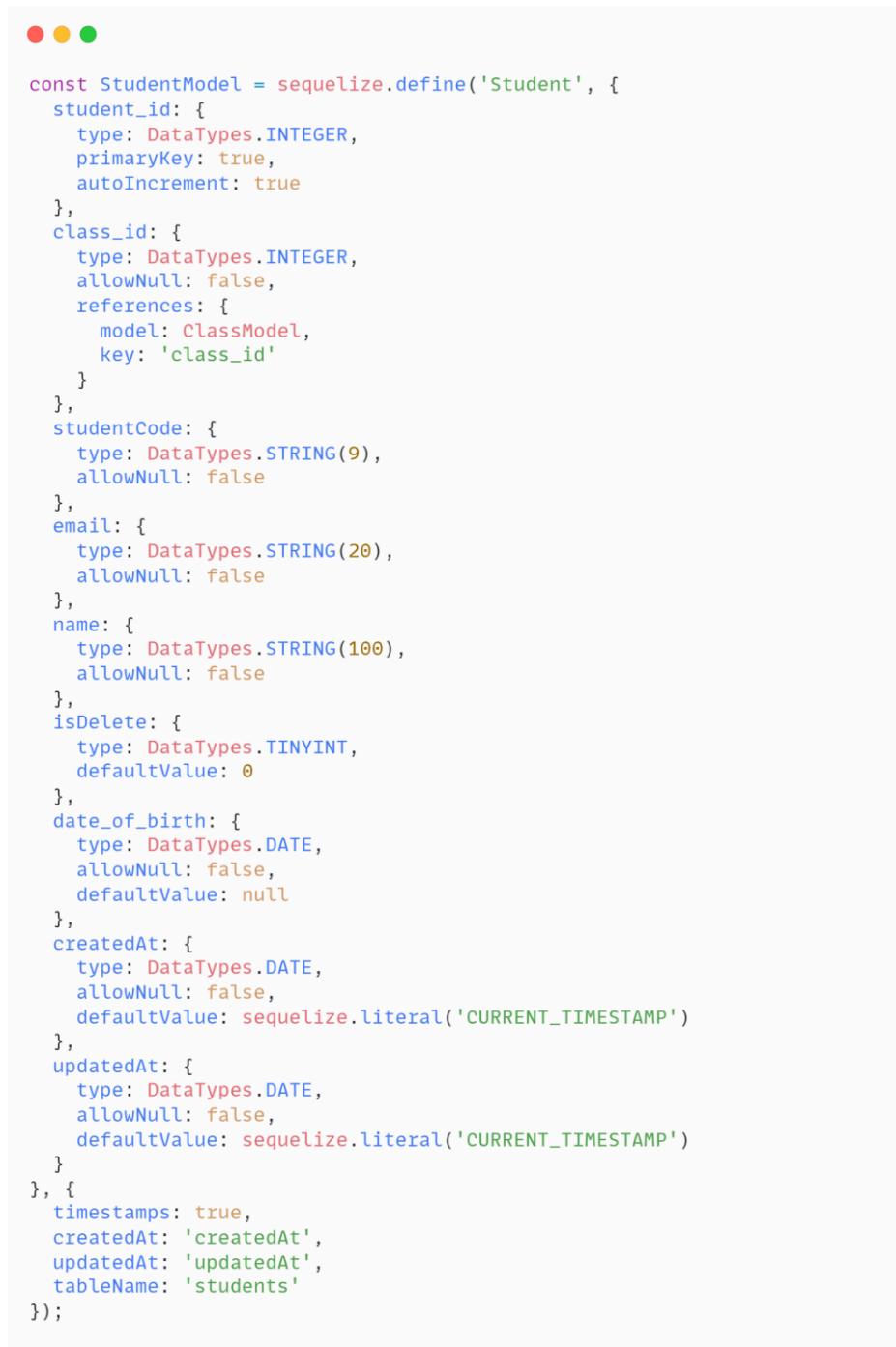
Hình 3.8. Import các thư viện tạo modal

**DataTypes**: Được import từ Sequelize để xác định kiểu dữ liệu của các thuộc tính trong mô hình.

**sequelize**: Instance của Sequelize, được cấu hình để kết nối tới cơ sở dữ liệu, được import từ file cấu hình (`../config/database`).

**ClassModel:** Mô hình lớp học, được import để thiết lập mối quan hệ với mô hình sinh viên.

## Định nghĩa mô hình



```
const StudentModel = sequelize.define('Student', {
  student_id: {
    type: DataTypes.INTEGER,
    primaryKey: true,
    autoIncrement: true
  },
  class_id: {
    type: DataTypes.INTEGER,
    allowNull: false,
    references: {
      model: ClassModel,
      key: 'class_id'
    }
  },
  studentCode: {
    type: DataTypes.STRING(9),
    allowNull: false
  },
  email: {
    type: DataTypes.STRING(20),
    allowNull: false
  },
  name: {
    type: DataTypes.STRING(100),
    allowNull: false
  },
  isDelete: {
    type: DataTypes.TINYINT,
    defaultValue: 0
  },
  date_of_birth: {
    type: DataTypes.DATE,
    allowNull: false,
    defaultValue: null
  },
  createdAt: {
    type: DataTypes.DATE,
    allowNull: false,
    defaultValue: sequelize.literal('CURRENT_TIMESTAMP')
  },
  updatedAt: {
    type: DataTypes.DATE,
    allowNull: false,
    defaultValue: sequelize.literal('CURRENT_TIMESTAMP')
  }
}, {
  timestamps: true,
  createdAt: 'createdAt',
  updatedAt: 'updatedAt',
  tableName: 'students'
});
```

Hình 3.9. Định nghĩa modal

sequelize.define('Student', {...}): Hàm định nghĩa mô hình với tên là Student.

student\_id: Khóa chính, tự động tăng.

class\_id: Khóa ngoại, tham chiếu tới class\_id trong ClassModel.

studentCode: Mã sinh viên, không được phép để trống.

email: Email sinh viên, không được phép để trống.

name: Tên sinh viên, không được phép để trống.

isDelete: Cờ đánh dấu xóa, mặc định là 0 (chưa bị xóa).

date\_of\_birth: Ngày sinh, không được phép để trống.

createdAt: Ngày tạo bản ghi, mặc định là thời gian hiện tại.

updatedAt: Ngày cập nhật bản ghi, mặc định là thời gian hiện tại.

timestamps: true: Tự động thêm các trường createdAt và updatedAt.

tableName: 'students': Tên bảng trong cơ sở dữ liệu là students.

## Thiết lập quan hệ giữa mô hình

```
● ● ●  
StudentModel.belongsTo(ClassModel, {  
  foreignKey: 'class_id'  
});
```

Hình 3.10. Thiết lập quan hệ mô hình

belongsTo: Thiết lập quan hệ belongs to giữa StudentModel và ClassModel qua khóa ngoại class\_id.

## Xuất mô hình

```
● ● ●  
module.exports = StudentModel;
```

Hình 3.11. Xuất mô hình

module.exports: Xuất khẩu mô hình StudentModel để có thể sử dụng trong các module khác của ứng dụng.

### 3.6.3. Xác thực người dùng

Cấu hình bảo mật:

Đoạn mã trên cấu hình passport.js sử dụng JWT (JSON Web Token) để xác thực người dùng thông qua passport-jwt strategy.

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

Hình 3.12. Cấu hình JWT Strategy

opts: Một đối tượng chứa các tùy chọn cho JwtStrategy.

jwtFromRequest: Xác định cách JWT được lấy từ request. Ở đây sử dụng fromAuthHeaderAsBearerToken() để lấy JWT từ tiêu đề Authorization dưới dạng Bearer token.

secretOrKey: Khóa bí mật để xác thực JWT, được lấy từ biến môi trường 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);  
    }  
}));
```

Hình 3.13. Sử dụng JWT Strategy với Passport

passport.use: Hàm này đăng ký JwtStrategy với Passport.

new JwtStrategy(opts, async (jwt\_payload, done) => {...}): Khởi tạo một strategy mới với các tùy chọn đã cấu hình.

jwt\_payload: Payload của JWT sau khi được giải mã.

done: Hàm callback được gọi sau khi hoàn thành quá trình xác thực.

TeacherModel.findByPk(jwt\_payload.id): Tìm người dùng trong cơ sở dữ liệu bằng ID từ payload của JWT.

Nếu tìm thấy người dùng, hàm done được gọi với người dùng đó.

Nếu không tìm thấy, hàm done được gọi với false.

Nếu có lỗi xảy ra trong quá trình tìm kiếm, hàm done được gọi với lỗi đó.

Xây dựng middleware xác thực

```
● ● ●  
const token = req.cookies.accessToken;  
if (!token) {  
    return res.status(401).json({ message: 'Access token is missing' });  
}
```

Hình 3.14. Kiểm tra Access Token

req.cookies.accessToken: Lấy Access Token từ cookie của yêu cầu.

Nếu không có Access Token, trả về mã trạng thái 401 cùng với thông báo lỗi.

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

Hình 3.15. Xác thực Access Token

jwt.verify(token, process.env.JWT\_SECRET): Xác thực Access Token bằng khóa bí mật (JWT\_SECRET).

TeacherModel.findByPk(decoded.id): Tìm người dùng bằng ID được giải mã từ Access Token.

Nếu không tìm thấy người dùng, trả về mã trạng thái 401 cùng với thông báo lỗi.

```

● ● ●

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' });
}

```

Hình 3.16. Kiểm tra Refresh Token

req.cookies.refreshToken: Lấy Refresh Token từ cookie của yêu cầu.

RefreshTokenModel.findOne: Tìm Refresh Token trong cơ sở dữ liệu với token và teacher\_id.

Kiểm tra xem Refresh Token có tồn tại, không bị thu hồi hoặc hết hạn. Nếu không hợp lệ, trả về mã trạng thái 401 cùng với thông báo lỗi.

### 3.6.4. Tạo và Quản Lý API Routes với Express.js

Router trong Express.js giúp quản lý các tuyến đường một cách có tổ chức và dễ bảo trì. Nó cho phép nhóm các tuyến đường liên quan lại với nhau và áp dụng middleware cho nhóm đó.

```

● ● ●

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

router.get('/subjects', ensureAuthenticated, SubjectController.index);
router.post('/subject', ensureAuthenticated, SubjectController.create);
router.get('/subject/:subject_id', ensureAuthenticated,
SubjectController.getByID);

```

Hình 3.17. Định nghĩa các tuyến đường cho môn học với middleware xác thực

router.get('/subjects', ensureAuthenticated, SubjectController.index):

Phương thức: GET

Đường dẫn: /subjects

Middleware: ensureAuthenticated để xác thực người dùng trước khi cho phép truy cập.

Hành động: Gọi phương thức index của SubjectController để lấy danh sách tất cả các môn học.

router.post('/subject', ensureAuthenticated, SubjectController.create):

Phương thức: POST

Đường dẫn: /subject

Middleware: ensureAuthenticated để xác thực người dùng trước khi cho phép truy cập.

Hành động: Gọi phương thức create của SubjectController để tạo một môn học mới.

router.get('/subject/', ensureAuthenticated, SubjectController.getByID):

Phương thức: GET

Đường dẫn: /subject/:subject\_id

Middleware: ensureAuthenticated để xác thực người dùng trước khi cho phép truy cập.

Hành động: Gọi phương thức getByID của SubjectController để lấy thông tin chi tiết của một môn học cụ thể dựa trên subject\_id.

### 3.6.5. Các API

#### 3.6.5.1. API của Authenticate

##### a) Đăng ký tài khoản teacher

```
● ● ●

register: async (req, res) => {
  const { email, password, name, teacherCode, typeTeacher } = req.body;
  try {
    const existingUser = await TeacherModel.findOne({ where: { teacherCode } });
    if (existingUser) {
      return res.status(400).json({ message: 'Teacher code đã được sử dụng' });
    }

    const data = {
      email,
      password,
      name,
      teacherCode,
      typeTeacher
    };
    const newUser = await TeacherModel.create(data);
    console.log(`Đã đăng ký người dùng mới: ${newUser.email}`);
    res.status(201).json(newUser);
  } catch (error) {
    console.error(`Lỗi đăng ký: ${error.message}`);
    res.status(500).json({ message: 'Lỗi server', error });
  }
}
```

Hình 3.18. API register.

Hàm register là một hàm không đồng bộ được thiết kế để xử lý quá trình đăng ký cho các giáo viên mới. Dưới đây là giải thích từng bước về hoạt động của nó:

Trích xuất dữ liệu từ yêu cầu: Hàm bắt đầu bằng việc trích xuất các trường email, password, name, teacherCode, và typeTeacher từ req.body. Những trường này được mong đợi sẽ được cung cấp bởi phía khách hàng trong nội dung yêu cầu.

Kiểm tra mã giáo viên hiện có: Hàm kiểm tra xem có giáo viên nào với teacherCode đã cung cấp có tồn tại trong cơ sở dữ liệu hay không bằng cách sử dụng TeacherModel.findOne({ where: { teacherCode } }). Nếu tìm thấy một người dùng đã tồn tại, nó sẽ trả về mã trạng thái 400 và thông báo rằng mã giáo viên đã được sử dụng.

Tạo người dùng mới: Nếu không tìm thấy người dùng hiện có, nó sẽ tiến hành tạo một người dùng mới. Nó xây dựng một đối tượng data chứa các trường đã trích xuất (email, password, name, teacherCode, typeTeacher).

Lưu người dùng mới: Hàm sau đó cố gắng lưu người dùng mới vào cơ sở dữ liệu bằng cách sử dụng TeacherModel.create(data). Khi tạo thành công, nó sẽ ghi lại email của người dùng mới đăng ký và trả về mã trạng thái 201 cùng với đối tượng người dùng mới được tạo ở định dạng JSON.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khôi catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo chỉ ra lỗi máy chủ, cùng với chi tiết lỗi.

## b) Đăng nhập

```
● ● ●

login: async (req, res) => {
  const { teacherCode, password } = req.body;
  try {
    const user = await TeacherModel.findOne({ where: { teacherCode } });
    if (!user) {
      return res.status(400).json({ message: 'Teacher code hoặc mật khẩu không đúng' });
    }

    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) {
      return res.status(400).json({ message: 'Teacher code hoặc mật khẩu không đúng' });
    }

    const payload = { id: user.teacher_id, permission: user.permission };
    const accessToken = jwt.sign(payload, process.env.JWT_SECRET, { expiresIn: '30m' });
    const refreshToken = jwt.sign(payload, process.env.JWT_SECRET, { expiresIn: '7d' });

    // Thu hồi và hết hạn tất cả các refresh token cũ của người dùng
    await RefreshTokenModel.update(
      { revoked: true, expired: true },
      { where: { teacher_id: user.teacher_id } }
    );

    // Lưu refresh token mới vào database
    await RefreshTokenModel.create({ token: refreshToken, teacher_id: user.teacher_id });

    // Đặt token trong cookies
    res.cookie('accessToken', accessToken, { httpOnly: false, secure: true, sameSite: 'Strict', maxAge: 30 * 60 * 1000 });
    res.cookie('refreshToken', refreshToken, { httpOnly: false, secure: true, sameSite: 'Strict', maxAge: 7 * 24 * 60 * 60 * 1000 });

    console.log(`Đăng nhập thành công cho người dùng: ${user.name}`);
    res.json({
      message: 'Đăng nhập thành công',
      accessToken,
      refreshToken
    });
  } catch (error) {
    console.error(`Lỗi đăng nhập: ${error.message}`);
    res.status(500).json({ message: 'Lỗi server', error });
  }
},
```

Hình 3.19. API đăng nhập

Hàm login là một hàm không đồng bộ được thiết kế để xử lý quá trình đăng nhập cho giáo viên. Dưới đây là giải thích từng bước về hoạt động của nó:

Trích xuất dữ liệu từ yêu cầu: Hàm bắt đầu bằng việc trích xuất các trường teacherCode và password từ req.body. Những trường này được mong đợi sẽ được cung cấp bởi phía khách hàng trong nội dung yêu cầu.

Kiểm tra mã giáo viên và mật khẩu: Hàm kiểm tra xem có giáo viên nào với teacherCode đã cung cấp có tồn tại trong cơ sở dữ liệu hay không bằng cách sử dụng TeacherModel.findOne({ where: { teacherCode } }). Nếu không tìm thấy người dùng, nó sẽ trả về mã trạng thái 400 và thông báo rằng mã giáo viên hoặc mật khẩu không đúng.

Xác thực mật khẩu: Nếu tìm thấy người dùng, hàm sẽ so sánh mật khẩu đã cung cấp với mật khẩu được lưu trữ trong cơ sở dữ liệu bằng cách sử dụng bcrypt.compare(password, user.password). Nếu mật khẩu không khớp, nó sẽ trả về mã trạng thái 400 và thông báo rằng mã giáo viên hoặc mật khẩu không đúng.

Tạo token truy cập và làm mới: Nếu mật khẩu khớp, hàm sẽ tạo ra một payload chứa ID người dùng và quyền hạn (permission). Sau đó, nó sẽ tạo ra token truy cập (access token) và token làm mới (refresh token) sử dụng jwt.sign với một thời hạn nhất định.

Thu hồi và hết hạn token cũ: Hàm sẽ thu hồi và đánh dấu hết hạn tất cả các token làm mới cũ của người dùng trong cơ sở dữ liệu bằng cách sử dụng RefreshTokenModel.update.

Lưu token mới vào database: Hàm sẽ lưu token làm mới vào cơ sở dữ liệu bằng cách sử dụng RefreshTokenModel.create.

Đặt token trong cookies: Token truy cập và token làm mới sẽ được đặt trong cookies với các tùy chọn bảo mật như httpOnly, secure, sameSite, và maxAge.

Phản hồi thành công: Cuối cùng, hàm sẽ ghi lại việc đăng nhập thành công và trả về thông báo cùng với token truy cập và token làm mới cho khách hàng.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo chỉ ra lỗi máy chủ, cùng với chi tiết lỗi.

### c) Đổi mật khẩu

```
● ● ●
changePassword: async (req, res) => {
  const { teacherCode, oldPassword, newPassword } = req.body;
  try {
    const user = await TeacherModel.findOne({ where: { teacherCode } });
    if (!user) {
      return res.status(404).json({ message: 'Người dùng không tồn tại' });
    }

    const passwordMatch = await bcrypt.compare(oldPassword, user.password);
    if (!passwordMatch) {
      return res.status(400).json({ message: 'Mật khẩu cũ không chính xác' });
    }

    user.password = newPassword;
    await user.save();

    console.log(`Đã thay đổi mật khẩu cho người dùng: ${user.email}`);
    res.status(200).json({ message: 'Đã thay đổi mật khẩu thành công' });
  } catch (error) {
    console.error(`Lỗi thay đổi mật khẩu: ${error.message}`);
    res.status(500).json({ message: 'Lỗi server', error });
  }
},
```

Hình 3.20. API đổi mật khẩu

Hàm changePassword là một hàm không đồng bộ được thiết kế để xử lý việc thay

đổi mật khẩu cho giáo viên. Dưới đây là giải thích từng bước về hoạt động của nó:

Trích xuất dữ liệu từ yêu cầu: Hàm bắt đầu bằng việc trích xuất các trường teacherCode, oldPassword, và newPassword từ req.body. Những trường này được mong đợi sẽ được cung cấp bởi phía khách hàng trong nội dung yêu cầu.

Kiểm tra người dùng tồn tại: Hàm kiểm tra xem có giáo viên nào với teacherCode đã cung cấp có tồn tại trong cơ sở dữ liệu hay không bằng cách sử dụng TeacherModel.findOne({ where: { teacherCode } }). Nếu không tìm thấy người dùng, nó sẽ trả về mã trạng thái 404 và thông báo rằng người dùng không tồn tại.

Xác thực mật khẩu cũ: Nếu tìm thấy người dùng, hàm sẽ so sánh mật khẩu cũ đã cung cấp với mật khẩu được lưu trữ trong cơ sở dữ liệu bằng cách sử dụng bcrypt.compare(oldPassword, user.password). Nếu mật khẩu không khớp, nó sẽ trả về mã trạng thái 400 và thông báo rằng mật khẩu cũ không chính xác.

Thay đổi mật khẩu: Nếu mật khẩu cũ khớp, hàm sẽ cập nhật mật khẩu mới cho người dùng và lưu thay đổi này vào cơ sở dữ liệu bằng cách sử dụng user.save().

Phản hồi thành công: Sau khi thay đổi mật khẩu thành công, hàm sẽ ghi lại việc thay đổi mật khẩu và trả về thông báo thành công với mã trạng thái 200.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo chỉ ra lỗi máy chủ, cùng với chi tiết lỗi.

## d) Đăng xuất

```
logout: async (req, res) => {
  try {
    const refreshToken = req.cookies.refreshToken;
    if (refreshToken) {
      // Thu hồi refresh token trong cơ sở dữ liệu
      await RefreshTokenModel.update(
        { revoked: true },
        { where: { token: refreshToken } }
      );
    }

    // Xóa cookies
    res.clearCookie('accessToken');
    res.clearCookie('refreshToken');

    res.json({ message: 'Đăng xuất thành công' });
  } catch (error) {
    console.error(`Lỗi đăng xuất: ${error.message}`);
    res.status(500).json({ message: 'Lỗi server', error });
  }
}
```

Hình 3.21. API đăng xuất

Hàm logout là một hàm không đồng bộ được thiết kế để xử lý việc đăng xuất của người dùng. Dưới đây là giải thích từng bước về hoạt động của nó:

Trích xuất refresh token từ cookies: Hàm bắt đầu bằng việc trích xuất refreshToken từ cookies của yêu cầu (req.cookies.refreshToken). Đây là token làm mới được lưu trữ trên trình duyệt của người dùng.

Thu hồi refresh token: Nếu tìm thấy refreshToken, hàm sẽ thu hồi nó trong cơ sở dữ liệu bằng cách sử dụng RefreshTokenModel.update, đánh dấu token là đã bị thu hồi (revoked: true) dựa trên giá trị token.

Xóa cookies: Sau khi thu hồi refreshToken, hàm sẽ xóa các cookies accessToken và refreshToken bằng cách sử dụng res.clearCookie.

Phản hồi thành công: Cuối cùng, hàm sẽ trả về thông báo thành công (Đăng xuất thành công) dưới dạng JSON.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo chỉ ra lỗi máy chủ, cùng với chi tiết lỗi.

## e) API lấy thông tin người đăng nhập

```
● ● ●

getUser: async (req, res) => {
  try {
    const user = await TeacherModel.findByPk(req.user.teacher_id, {
      attributes: ['teacher_id', 'email', 'permission', 'name', 'teacherCode', 'typeTeacher', 'imgURL']
    });

    if (!user) {
      return res.status(404).json({ message: 'Người dùng không tồn tại' });
    }

    res.json(user);
  } catch (error) {
    console.error(`Lỗi lấy thông tin người dùng: ${error.message}`);
    res.status(500).json({ message: 'Lỗi server', error });
  }
},
```

Hình 3.22. API lấy thông tin người dùng.

Hàm getUser là một hàm không đồng bộ được thiết kế để lấy thông tin người dùng từ cơ sở dữ liệu. Dưới đây là giải thích từng bước về hoạt động của nó:

Lấy thông tin người dùng: Hàm bắt đầu bằng việc sử dụng TeacherModel.findByPk để tìm người dùng trong cơ sở dữ liệu theo teacher\_id của người dùng (req.user.teacher\_id). Hàm này sẽ chỉ lấy các thuộc tính được chỉ định bao gồm: teacher\_id, email, permission, name, teacherCode, typeTeacher, và imgURL.

Kiểm tra người dùng tồn tại: Nếu không tìm thấy người dùng, hàm sẽ trả về mã trạng thái 404 và thông báo rằng người dùng không tồn tại.

Phản hồi thành công: Nếu tìm thấy người dùng, hàm sẽ trả về thông tin của người dùng dưới dạng JSON.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo chỉ ra lỗi máy chủ, cùng với chi tiết lỗi.

### 3.6.5.2. API của Token

#### a) Làm mới token

```
refreshToken: async (req, res) => {
    const { refreshToken } = req.cookies;

    if (!refreshToken) {
        return res.status(400).json({ message: 'Refresh token là bắt buộc' });
    }

    try {
        const storedToken = await RefreshTokenModel.findOne({ where: { token: refreshToken } });

        if (!storedToken || storedToken.revoked || storedToken.expired) {
            return res.status(400).json({ message: 'Refresh token không hợp lệ' });
        }

        const decoded = jwt.verify(refreshToken, process.env.JWT_SECRET);
        const user = await TeacherModel.findByPk(decoded.id);

        if (!user) {
            return res.status(400).json({ message: 'Người dùng không hợp lệ' });
        }

        const payload = { id: user.teacher_id, permission: user.permission };
        const newAccessToken = jwt.sign(payload, process.env.JWT_SECRET, { expiresIn: '30m' });
        const newRefreshToken = jwt.sign(payload, process.env.JWT_SECRET, { expiresIn: '7d' });

        // Đánh dấu refresh token cũ là đã thu hồi và hết hạn
        storedToken.revoked = true;
        storedToken.expired = true;
        await storedToken.save();

        // Lưu refresh token mới vào database
        await RefreshTokenModel.create({ token: newRefreshToken, teacher_id: user.teacher_id });

        // Đặt token mới trong HTTP-only cookies
        res.cookie('accessToken', newAccessToken, { httpOnly: false, secure: true, sameSite: 'Strict', maxAge: 30 * 60 * 1000 });
        res.cookie('refreshToken', newRefreshToken, { httpOnly: false, secure: true, sameSite: 'Strict', maxAge: 7 * 24 * 60 * 60 * 1000 });

        res.json({
            message: 'Làm mới token thành công',
            newAccessToken,
            newRefreshToken
        });
    } catch (error) {
        console.error(`Lỗi làm mới token: ${error.message}`);
        res.status(500).json({ message: 'Lỗi server', error });
    }
}
```

Hình 3.23. API làm mới token

Hàm refreshToken là một hàm không đồng bộ được thiết kế để làm mới token truy cập cho người dùng. Dưới đây là giải thích từng bước về hoạt động của nó:

Trích xuất refresh token từ cookies: Hàm bắt đầu bằng việc trích xuất refreshToken từ cookies của yêu cầu (req.cookies.refreshToken). Nếu không có refreshToken, hàm sẽ trả về mã trạng thái 400 và thông báo rằng refresh token là bắt buộc.

Kiểm tra tính hợp lệ của refresh token: Hàm kiểm tra xem refreshToken có tồn tại trong cơ sở dữ liệu hay không bằng cách sử dụng RefreshTokenModel.findOne. Nếu không tìm thấy token hoặc token đã bị thu hồi hoặc hết hạn, hàm sẽ trả về mã trạng thái 400 và

thông báo rằng refresh token không hợp lệ.

Giải mã refresh token: Nếu token hợp lệ, hàm sẽ giải mã refreshToken bằng cách sử dụng jwt.verify với khóa bí mật từ biến môi trường (process.env.JWT\_SECRET).

Kiểm tra người dùng tồn tại: Hàm kiểm tra xem người dùng có tồn tại trong cơ sở dữ liệu bằng cách sử dụng TeacherModel.findByPk với id được giải mã từ token. Nếu không tìm thấy người dùng, hàm sẽ trả về mã trạng thái 400 và thông báo rằng người dùng không hợp lệ.

Tạo token mới: Nếu người dùng hợp lệ, hàm sẽ tạo payload chứa id và permission của người dùng. Sau đó, hàm tạo token truy cập mới (newAccessToken) và token làm mới mới (newRefreshToken) sử dụng jwt.sign.

Thu hồi và hết hạn refresh token cũ: Hàm sẽ đánh dấu refresh token cũ là đã thu hồi và hết hạn, sau đó lưu thay đổi này vào cơ sở dữ liệu.

Lưu refresh token mới vào database: Hàm sẽ lưu token làm mới vào cơ sở dữ liệu bằng cách sử dụng RefreshTokenModel.create.

Đặt token mới trong cookies: Token truy cập và token làm mới mới sẽ được đặt trong cookies với các tùy chọn bảo mật như httpOnly, secure, sameSite, và maxAge.

Phản hồi thành công: Cuối cùng, hàm sẽ trả về thông báo thành công cùng với token truy cập và token làm mới mới cho khách hàng.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo chỉ ra lỗi máy chủ, cùng với chi tiết lỗi.

## b) Thu hồi token

```
revokeToken: async (req, res) => {
  const { refreshToken } = req.body;

  if (!refreshToken) {
    return res.status(400).json({ message: 'Refresh token là bắt buộc' });
  }

  try {
    const storedToken = await RefreshTokenModel.findOne({ where: { token: refreshToken } });

    if (!storedToken) {
      return res.status(400).json({ message: 'Refresh token không hợp lệ' });
    }

    storedToken.revoked = true;
    await storedToken.save();

    res.json({ message: 'Token đã bị thu hồi' });
  } catch (error) {
    console.error(`Lỗi thu hồi token: ${error.message}`);
    res.status(500).json({ message: 'Lỗi server', error });
  }
}
```

Hình 3.24. Thu hồi token

Hàm revokeToken là một hàm không đồng bộ được thiết kế để thu hồi một refresh token cụ thể. Dưới đây là giải thích từng bước về hoạt động của nó:

Trích xuất refresh token từ yêu cầu: Hàm bắt đầu bằng việc trích xuất refreshToken từ req.body. Nếu không có refreshToken, hàm sẽ trả về mã trạng thái 400 và thông báo rằng refresh token là bắt buộc.

Kiểm tra tính hợp lệ của refresh token: Hàm kiểm tra xem refreshToken có tồn tại trong cơ sở dữ liệu hay không bằng cách sử dụng RefreshTokenModel.findOne. Nếu không tìm thấy token, hàm sẽ trả về mã trạng thái 400 và thông báo rằng refresh token không hợp lệ.

Thu hồi refresh token: Nếu token hợp lệ, hàm sẽ đánh dấu token là đã bị thu hồi (revoked = true) và lưu thay đổi này vào cơ sở dữ liệu bằng cách sử dụng storedToken.save().

Phản hồi thành công: Cuối cùng, hàm sẽ trả về thông báo thành công (Token đã bị thu hồi) dưới dạng JSON.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo chỉ ra lỗi máy chủ, cùng với chi tiết lỗi.

### 3.6.5.3. API chart

#### a) Tỉ lệ đạt được clo trên subject

```
● ● ●

getCloPercentage: async (req, res) => {
  try {
    const results = await sequelize.query(
      `SELECT
        s.subject_id,
        s.subjectName,
        c.clo_id,
        c.cloName,
        SUM(ai.assessmentScore) AS totalScoreAchieved,
        SUM(ri.maxScore) AS totalMaxScore,
        ROUND((SUM(ai.assessmentScore) / SUM(ri.maxScore)),4) AS percentage_score
      FROM
        assessmentItems ai
      JOIN
        rubricsItems ri ON ai.rubricsItem_id = ri.rubricsItem_id
      JOIN
        rubrics r ON ri.rubric_id = r.rubric_id
      JOIN
        subjects s ON r.subject_id = s.subject_id
      JOIN
        clos c ON ri.clo_id = c.clo_id
      WHERE
        ai.isDelete = 0
        AND ri.isDelete = 0
        AND r.isDelete = 0
        AND s.isDelete = 0
        AND c.isDelete = 0
      GROUP BY
        s.subject_id, s.subjectName, c.clo_id, c.cloName
      ORDER BY
        s.subject_id, c.clo_id;
    `,
    {
      type: Sequelize.QueryTypes.SELECT,
    }
  );
  const formattedResults = results.reduce((acc, result) => {
    const { subject_id, subjectName, clo_id, cloName, description, percentage_score } = result;

    if (!acc[subject_id]) {
      acc[subject_id] = {
        subject_id,
        subjectName,
        clos: []
      };
    }

    acc[subject_id].clos.push({
      clo_id,
      cloName,
      description,
      percentage_score
    });
  });

  return acc;
}, {});

res.json(Object.values(formattedResults));
} catch (error) {
  console.error('Error:', error);
  res.status(500).json({ message: 'Internal Server Error' });
}
}
```

Hình 3.25. API tỉ lệ đạt được của chuẩn đầu ra môn học.

Hàm getCloPercentage là một hàm không đồng bộ được thiết kế để truy vấn và tính toán tỷ lệ phần trăm điểm đạt được của các CLO (Course Learning Objectives) cho từng

môn học. Dưới đây là giải thích từng bước về hoạt động của nó:

Thực hiện truy vấn cơ sở dữ liệu: Hàm sử dụng `sequelize.query` để thực hiện truy vấn SQL nhằm tính toán tỷ lệ phần trăm điểm đạt được cho từng CLO của mỗi môn học. Truy vấn này liên kết các bảng `assessmentItems`, `rubricsItems`, `rubrics`, `subjects` và `clos` để lấy tổng điểm đạt được (`totalScoreAchieved`) và tổng điểm tối đa (`totalMaxScore`), sau đó tính toán tỷ lệ phần trăm điểm đạt được (`percentage_score`).

Định dạng kết quả: Kết quả truy vấn được định dạng lại để dễ dàng sử dụng hơn. Hàm sử dụng `reduce` để gom nhóm kết quả theo `subject_id`, mỗi nhóm chứa các CLO tương ứng với thông tin như `clo_id`, `cloName`, `description`, và `percentage_score`.

Trả về kết quả: Kết quả được trả về dưới dạng JSON với cấu trúc rõ ràng, dễ đọc.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khôi `catch`. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo chỉ ra lỗi máy chủ, cùng với chi tiết lỗi.

## b) Tỉ lệ đạt của Plo

```
getPloPercentage: async (req, res) => {
  try {
    const results = await sequelize.query(
      `SELECT
        plo.plo_id,
        plo.ploName,
        SUM(ai.assessmentScore) AS totalScoreAchieved,
        SUM(ri.maxScore) AS totalMaxScore,
        ROUND((SUM(ai.assessmentScore) / SUM(ri.maxScore)), 4) AS percentage_score
      FROM
        assessmentItems ai
      JOIN rubricsItems ri ON ai.rubricsItem_id = ri.rubricsItem_id
      JOIN rubrics r ON ri.rubric_id = r.rubric_id
      JOIN subjects s ON r.subject_id = s.subject_id
      JOIN plos plo ON ri.plo_id = plo.plo_id
      WHERE
        ai.isDelete = 0 AND ri.isDelete = 0 AND r.isDelete = 0 AND s.isDelete = 0 AND plo.isDelete = 0
      GROUP BY
        plo.plo_id,
        plo.ploName
      ORDER BY
        plo.plo_id,plo.ploName;`,
    {
      type: Sequelize.QueryTypes.SELECT,
    }
  );
  const formattedResults = results.reduce((acc, result) => {
    const { subject_id, subjectName, plo_id, ploName, percentage_score } = result;
    if (!acc[subject_id]) {
      acc[subject_id] = {
        subject_id,
        subjectName,
        plos: []
      };
    }
    acc[subject_id].plos.push({
      plo_id,
      ploName,
      percentage_score
    });
    return acc;
  }, {});
  res.json(Object.values(formattedResults));
} catch (error) {
  console.error('Error:', error);
  res.status(500).json({ message: 'Internal Server Error' });
}
}
```

Hình 3.26. API tỉ lệ đạt được câu Plo

Hàm getPloPercentage là một hàm không đồng bộ được thiết kế để truy vấn và tính toán tỷ lệ phần trăm điểm đạt được của các PLO (Program Learning Outcomes) cho từng môn học. Dưới đây là giải thích từng bước về hoạt động của nó:

Thực hiện truy vấn cơ sở dữ liệu: Hàm sử dụng sequelize.query để thực hiện truy vấn SQL nhằm tính toán tỷ lệ phần trăm điểm đạt được cho từng PLO. Truy vấn này liên

kết các bảng assessmentItems, rubricsItems, rubrics, subjects, và plos để lấy tổng điểm đạt được (totalScoreAchieved) và tổng điểm tối đa (totalMaxScore), sau đó tính toán tỷ lệ phần trăm điểm đạt được (percentage\_score).

Định dạng kết quả: Kết quả truy vấn được định dạng lại để dễ dàng sử dụng hơn. Hàm sử dụng reduce để gom nhóm kết quả theo subject\_id, mỗi nhóm chứa các PLO tương ứng với thông tin như plo\_id, ploName, và percentage\_score.

Trả về kết quả: Kết quả được trả về dưới dạng JSON với cấu trúc rõ ràng, dễ đọc.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo chỉ ra lỗi máy chủ, cùng với chi tiết lỗi.

### c) Điểm trung bình của subject

```
● ● ●  
averageScoresPerSubject: async (req, res) => {  
  try {  
    const results = await sequelize.query(  
      `SELECT  
        ROUND(AVG(a.totalScore), 2) AS averageScore,  
        s.subject_id AS subject_id,  
        s.subjectName AS subjectName  
      FROM  
        assessments AS a  
      LEFT JOIN courses AS c  
        ON a.course_id = c.course_id  
      LEFT JOIN subjects AS s  
        ON c.subject_id = s.subject_id  
      GROUP BY  
        s.subject_id, s.subjectName;  
    `,  
    {  
      type: Sequelize.QueryTypes.SELECT  
    }  
  );  
  res.json(results);  
} catch (error) {  
  console.error('Error fetching average scores per subject:', error);  
  res.status(500).json({ message: 'Internal Server Error' });  
}  
}
```

Hình 3.27. API điểm trung bình của subject

Hàm averageScoresPerSubject là một hàm không đồng bộ được thiết kế để truy vấn và tính toán điểm trung bình của các môn học. Dưới đây là giải thích từng bước về hoạt động của nó:

Thực hiện truy vấn cơ sở dữ liệu: Hàm sử dụng sequelize.query để thực hiện truy vấn SQL nhằm tính toán điểm trung bình (averageScore) cho từng môn học (subject). Truy vấn này liên kết các bảng assessments, courses, và subjects để lấy dữ liệu cần thiết. Điểm

trung bình được tính bằng cách lấy trung bình cộng tổng điểm (totalScore) từ bảng assessments.

Định dạng kết quả: Kết quả truy vấn bao gồm điểm trung bình (averageScore), mã môn học (subject\_id), và tên môn học (subjectName). Kết quả được trả về dưới dạng JSON.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo chỉ ra lỗi máy chủ, cùng với chi tiết lỗi.

#### d) Lấy điểm các môn của một sinh viên



```
getStudentPerformanceByCourse: async (req, res) => {
  try {
    const { student_id } = req.params;
    const results = await sequelize.query(
      `SELECT
        a.assessment_id,
        a.totalScore,
        s.student_id AS student_id,
        s.studentCode AS studentCode,
        s.name AS studentName,
        s.date_of_birth AS studentDOB,
        c.class_id AS class_id,
        c.className AS className,
        c.classCode AS classCode,
        c.classNameShort AS classNameShort,
        cr.course_id AS course_id,
        cr.courseName AS courseName,
        sub.subject_id AS subject_id,
        sub.subjectName AS subjectName,
        sem.semester_id AS semester_id,
        sem.descriptionShort AS semesterDescriptionShort,
        ay.academic_year_id AS academic_year_id,
        ay.startDate AS academicYearStartDate,
        ay.endDate AS academicYearEndDate,
        ay.description AS academicYearDescription
      FROM
        assessments AS a
      LEFT JOIN students AS s
        ON a.student_id = s.student_id
      LEFT JOIN classes AS c
        ON s.class_id = c.class_id
      LEFT JOIN courses AS cr
        ON a.course_id = cr.course_id
      LEFT JOIN subjects AS sub
        ON cr.subject_id = sub.subject_id
      LEFT JOIN semester_academic_years AS say
        ON cr.id_semester_academic_year = say.id_semester_academic_year
      LEFT JOIN semesters AS sem
        ON say.semester_id = sem.semester_id
      LEFT JOIN academic_years AS ay
        ON say.academic_year_id = ay.academic_year_id
      WHERE
        a.student_id = :student_id
        AND a.isDelete = false
      ORDER BY
        ay.startDate ASC;`,
      {
        type: Sequelize.QueryTypes.SELECT,
        replacements: { student_id }
      }
    );
    res.json(results);
  } catch (error) {
    console.error('Error fetching student performance by course:', error);
    res.status(500).json({ message: 'Internal Server Error' });
  }
}
```

Hình 3.28. API lấy điểm của một sinh viên

Hàm getStudentPerformanceByCourse là một hàm không đồng bộ được thiết kế để lấy dữ liệu hiệu suất học tập của một sinh viên cụ thể theo khóa học. Dưới đây là giải thích từng bước về hoạt động của nó:

Lấy thông tin sinh viên từ yêu cầu: Hàm bắt đầu bằng việc lấy student\_id từ các tham số của yêu cầu (req.params).

Thực hiện truy vấn cơ sở dữ liệu: Hàm sử dụng sequelize.query để thực hiện truy vấn SQL nhằm lấy dữ liệu hiệu suất học tập của sinh viên. Truy vấn này liên kết nhiều bảng như assessments, students, classes, courses, subjects, semester\_academic\_years, semesters, và academic\_years để lấy dữ liệu cần thiết. Truy vấn này bao gồm các thông tin về điểm số (totalScore), thông tin sinh viên (student\_id, studentCode, studentName, studentDOB), thông tin lớp học (class\_id, className, classCode, classNameShort), thông tin khóa học (course\_id, courseName), thông tin môn học (subject\_id, subjectName), thông tin học kỳ (semester\_id, semesterDescriptionShort), và thông tin năm học (academic\_year\_id, academicYearStartDate, academicYearEndDate, academicYearDescription).

Trả về kết quả: Kết quả của truy vấn được trả về dưới dạng JSON.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo chỉ ra lỗi máy chủ, cùng với chi tiết lỗi.

## e) Điểm trung bình khóa học

```

● ● ●

getAverageCourseScores: async (req, res) => {
  try {
    const {
      course_id_list,
      subject_id_list,
      academic_year_id_list,
      semester_id_list,
      class_id_list
    } = req.body.processedFilters;

    // Construct dynamic query filters
    const courseIdFilter = course_id_list && course_id_list.length > 0 ? 'AND c.course_id IN (:course_id_list)' : '';
    const subjectIdFilter = subject_id_list && subject_id_list.length > 0 ? 'AND sub.subject_id IN (:subject_id_list)' : '';
    const academicYearIdFilter = academic_year_id_list && academic_year_id_list.length > 0 ? 'AND ay.academic_year_id IN (:academic_year_id_list)' : '';
    const semesterIdFilter = semester_id_list && semester_id_list.length > 0 ? 'AND s.semester_id IN (:semester_id_list)' : '';
    const classIdFilter = class_id_list && class_id_list.length > 0 ? 'AND cl.class_id IN (:class_id_list)' : '';

    const query = `

      SELECT
        c.course_id,
        c.courseName,
        ay.academic_year_id,
        ay.description AS academic_year,
        s.semester_id,
        s.descriptionShort AS semester,
        t.teacher_id,
        t.name AS teacherName,
        cl.class_id,
        cl.className,
        AVG(a.totalScore) AS averageScore
      FROM
        courses c
      JOIN
        semester_academic_years say ON c.id_semester_academic_year = say.id_semester_academic_year
      JOIN
        academic_years ay ON say.academic_year_id = ay.academic_year_id
      JOIN
        semesters s ON say.semester_id = s.semester_id
      JOIN
        assessments a ON c.course_id = a.course_id
      JOIN
        teachers t ON c.teacher_id = t.teacher_id
      JOIN
        classes cl ON c.class_id = cl.class_id
      JOIN
        subjects sub ON c.subject_id = sub.subject_id
      WHERE
        c.isDelete = 0
        AND ay.isDelete = 0
        AND s.isDelete = 0
        AND a.isDelete = 0
        ${courseIdFilter}
        ${subjectIdFilter}
        ${academicYearIdFilter}
        ${semesterIdFilter}
        ${classIdFilter}
      GROUP BY
        c.course_id, c.courseName, ay.academic_year_id, ay.description, s.semester_id, s.descriptionShort,
        t.teacher_id, t.name, cl.class_id, cl.className
      ORDER BY
        ay.academic_year_id, s.semester_id, c.course_id;
    `;

    const replacements = {
      ...(course_id_list && course_id_list.length > 0 && { course_id_list }),
      ...(subject_id_list && subject_id_list.length > 0 && { subject_id_list }),
      ...(academic_year_id_list && academic_year_id_list.length > 0 && { academic_year_id_list }),
      ...(semester_id_list && semester_id_list.length > 0 && { semester_id_list }),
      ...(class_id_list && class_id_list.length > 0 && { class_id_list })
    };

    const results = await sequelize.query(query, {
      type: Sequelize.QueryTypes.SELECT,
      replacements,
    });

    res.json(results);
  } catch (error) {
    console.error('Error fetching average course scores:', error);
    res.status(500).json({ message: 'Internal Server Error' });
  }
}

```

Hình 3.29. API điểm trung bình môn học

Hàm getAverageCourseScores là một hàm không đồng bộ được thiết kế để lấy điểm trung bình của các khóa học dựa trên các bộ lọc động được cung cấp trong yêu cầu. Dưới đây là giải thích từng bước về hoạt động của nó:

Lấy và xử lý bộ lọc từ yêu cầu: Hàm bắt đầu bằng việc lấy các bộ lọc từ req.body.processedFilters. Các bộ lọc bao gồm course\_id\_list, subject\_id\_list, academic\_year\_id\_list, semester\_id\_list, và class\_id\_list.

Xây dựng các bộ lọc truy vấn động: Hàm kiểm tra từng bộ lọc và xây dựng các điều kiện bổ sung cho truy vấn SQL dựa trên các giá trị được cung cấp. Nếu một bộ lọc có giá trị, nó sẽ thêm điều kiện tương ứng vào truy vấn.

Tạo truy vấn SQL: Hàm sử dụng các bộ lọc động để xây dựng truy vấn SQL. Truy vấn này lấy các thông tin như course\_id, courseName, academic\_year\_id, academic\_year, semester\_id, semester, teacher\_id, teacherName, class\_id, className, và tính điểm trung bình (averageScore) từ bảng assessments.

Thực hiện truy vấn cơ sở dữ liệu: Hàm thực hiện truy vấn SQL bằng cách sử dụng sequelize.query, truyền các giá trị thay thế cho các bộ lọc động.

Trả về kết quả: Kết quả của truy vấn được trả về dưới dạng JSON.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo chỉ ra lỗi máy chủ, cùng với chi tiết lỗi.

#### 3.6.5.4. API của lớp học

##### a) Lấy tất cả các thông tin lớp học



```
● ● ●

index: async (req, res) => {
  try {
    const classes = await ClassModel.findAll({ where: { isDelete: false } });
    res.json(classes);
  } catch (error) {
    console.error('Lỗi khi lấy tất cả các lớp học:', error);
    res.status(500).json({ message: 'Lỗi máy chủ nội bộ' });
  }
}
```

Hình 3.30. API lấy tất cả các lớp học.

Hàm index là một hàm không đồng bộ được thiết kế để lấy danh sách tất cả các lớp

học từ cơ sở dữ liệu. Dưới đây là giải thích từng bước về hoạt động của nó:

Truy vấn cơ sở dữ liệu: Hàm sử dụng ClassModel.findAll để tìm tất cả các lớp học trong cơ sở dữ liệu mà có thuộc tính isDelete bằng false. Điều này đảm bảo rằng chỉ các lớp học chưa bị xóa mới được trả về.

Trả về kết quả: Kết quả của truy vấn, tức là danh sách các lớp học, được trả về dưới dạng JSON.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo chỉ ra lỗi máy chủ nội bộ, cùng với chi tiết lỗi.

### b) Lấy lớp học cùng với thông tin giáo viên

```
getAllWithTeacher: async (req, res) => {
  try {
    const { page, size } = req.query;

    const attributes = ['class_id', 'teacher_id', 'className', 'classNameShort', 'classCode', 'isDelete'];
    const whereClause = { isDelete: false };

    if (page && size) {
      const offset = (page - 1) * size;
      const limit = parseInt(size, 10);

      const { count, rows: classes } = await ClassModel.findAndCountAll({
        include: [
          { model: TeacherModel,
            attributes: ['name']
          },
          attributes: attributes,
          where: whereClause,
          offset: offset,
          limit: limit
        });
      return res.json({
        total: count,
        classes: classes
      });
    } else {
      const classes = await ClassModel.findAll({
        include: [
          { model: TeacherModel,
            attributes: ['name']
          },
          attributes: attributes,
          where: whereClause
        });
      return res.json(classes);
    }
  } catch (error) {
    console.error('Error:', error);
    res.status(500).json({ message: 'Internal Server Error' });
  }
}
```

Hình 3.31. API lấy tất cả thông tin lớp học và giáo viên

Hàm getAllWithTeacher là một hàm không đồng bộ được thiết kế để lấy danh sách

tất cả các lớp học cùng với thông tin của giáo viên phụ trách, có hỗ trợ phân trang. Dưới đây là giải thích từng bước về hoạt động của nó:

Lấy thông tin phân trang từ yêu cầu: Hàm bắt đầu bằng việc lấy các tham số page và size từ req.query. Đây là các tham số dùng để xác định trang hiện tại và kích thước của mỗi trang.

Xác định các thuộc tính và điều kiện lọc: Hàm xác định các thuộc tính cần lấy từ bảng ClassModel và điều kiện lọc để chỉ lấy các lớp học chưa bị xóa (isDelete: false).

Kiểm tra và xử lý phân trang:

Nếu có thông tin về phân trang (page và size), hàm tính toán offset và limit để lấy dữ liệu theo trang:

offset được tính dựa trên trang hiện tại và kích thước trang: (page - 1) \* size.

limit là số lượng bản ghi cần lấy trong mỗi trang.

Hàm sử dụng ClassModel.findAndCountAll để lấy danh sách các lớp học cùng với số lượng tổng cộng các lớp học thỏa mãn điều kiện.

Kết quả trả về bao gồm tổng số bản ghi (count) và danh sách các lớp học (classes).

Xử lý khi không có phân trang: Nếu không có thông tin phân trang, hàm sử dụng ClassModel.findAll để lấy tất cả các lớp học thỏa mãn điều kiện.

Bao gồm thông tin giáo viên: Trong cả hai trường hợp, hàm đều bao gồm thông tin giáo viên bằng cách sử dụng include với TeacherModel, chỉ lấy thuộc tính name của giáo viên.

Trả về kết quả: Kết quả được trả về dưới dạng JSON, bao gồm tổng số bản ghi và danh sách các lớp học khi có phân trang, hoặc chỉ danh sách các lớp học khi không có phân trang.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo chỉ ra lỗi máy chủ, cùng với chi tiết lỗi.

### c) Lấy thông tin lớp bởi id



```
getByID: async (req, res) => {
  try {
    const { id } = req.params;
    const classes = await ClassModel.findAll({
      include: [
        {
          model: TeacherModel,
          attributes: ['name']
        },
        {
          attributes: ['class_id', 'teacher_id', 'className', 'classCode', 'isDelete'],
          where: {
            isDelete: false,
            class_id: id
          }
        }
      ];
    if (!classes) {
      return res.status(404).json({ message: 'Không tìm thấy lớp học' });
    }
    res.json(classes);
  } catch (error) {
    console.error('Lỗi khi tìm kiếm lớp học:', error);
    res.status(500).json({ message: 'Lỗi máy chủ' });
  }
}
```

Hình 3.32. Lấy thông tin lớp học bởi id

Hàm `getByID` là một hàm không đồng bộ được thiết kế để lấy thông tin chi tiết về một lớp học dựa trên ID của lớp. Dưới đây là giải thích từng bước về hoạt động của nó:

Lấy ID từ yêu cầu: Hàm bắt đầu bằng việc lấy id từ các tham số của yêu cầu (`req.params`).

Thực hiện truy vấn cơ sở dữ liệu: Hàm sử dụng `ClassModel.findAll` để tìm lớp học trong cơ sở dữ liệu dựa trên `class_id` đã cung cấp. Truy vấn này bao gồm:

Thông tin của giáo viên (`TeacherModel`) với thuộc tính `name`.

Các thuộc tính của lớp học: `class_id`, `teacher_id`, `className`, `classCode`, `isDelete`.

Điều kiện lọc để chỉ lấy các lớp học chưa bị xóa (`isDelete: false`) và có `class_id` khớp với id đã cung cấp.

Kiểm tra kết quả: Sau khi thực hiện truy vấn, hàm kiểm tra xem có lớp học nào được tìm thấy hay không. Nếu không có lớp học nào khớp, hàm sẽ trả về mã trạng thái 404 cùng với thông báo rằng không tìm thấy lớp học.

Trả về kết quả: Nếu tìm thấy lớp học, kết quả sẽ được trả về dưới dạng JSON.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi `catch`. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo chỉ ra lỗi máy chủ, cùng với chi tiết lỗi.

#### d) Xuất một tệp Excel chứa thông tin về các lớp học dựa trên danh sách ID

```
getExcelWithData: async (req, res) => {
  try {
    const { data } = req.body;

    if (!data || !Array.isArray(data.id) || data.id.length === 0) {
      return res.status(400).json({ error: 'Invalid or missing id array' });
    }

    const { id } = data;

    const workbook = new ExcelJS.Workbook();
    const worksheet = workbook.addWorksheet('Students Form');

    const classes = await ClassModel.findAll({
      include: [
        {
          model: TeacherModel,
          attributes: ['name']
        },
        attributes: ['class_id', 'teacher_id', 'className', 'classCode', 'isDelete'],
        where: {
          isDelete: false,
          class_id: id
        }
      ],
    });

    worksheet.columns = [
      { header: 'id', key: 'id', width: 15 },
      { header: 'Mã lớp', key: 'classCode', width: 15 },
      { header: 'Tên lớp', key: 'className', width: 32 },
      { header: 'GVCV', key: 'nameTeacher', width: 20 },
    ];

    classes.forEach(item => {
      worksheet.addRow({
        id: item.class_id,
        classCode: item.classCode,
        className: item.className,
        nameTeacher: item.teacher.name,
        email: item.email
      });
    });

    await worksheet.protect('yourpassword', {
      selectLockedCells: true,
      selectUnlockedCells: true
    });

    worksheet.eachRow((row, rowNum) => {
      row.eachCell((cell, colNumber) => {
        if (colNumber === 1) {
          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' });
  }
}
```

Hình 3.33. API xuất file excel lớp học.

Hàm getExcelWithData là một hàm không đồng bộ được thiết kế để tạo và xuất một tệp Excel chứa thông tin về các lớp học dựa trên danh sách ID được cung cấp trong yêu cầu. Dưới đây là giải thích từng bước về hoạt động của nó:

Kiểm tra dữ liệu đầu vào: Hàm bắt đầu bằng việc lấy dữ liệu từ req.body. Nếu dữ liệu không hợp lệ hoặc không chứa một mảng id, hàm sẽ trả về mã trạng thái 400 và thông báo lỗi rằng mảng ID không hợp lệ hoặc thiếu.

Tạo tệp Excel mới: Hàm sử dụng thư viện ExcelJS để tạo một workbook mới và thêm một worksheet với tên "Students Form".

Truy vấn cơ sở dữ liệu: Hàm sử dụng ClassModel.findAll để lấy thông tin về các lớp học dựa trên danh sách ID được cung cấp. Truy vấn bao gồm thông tin giáo viên (TeacherModel) với thuộc tính name, và các thuộc tính của lớp học như class\_id, teacher\_id, className, classCode, isDelete.

Định dạng các cột trong worksheet: Hàm định nghĩa các cột trong worksheet bao gồm id, Mã lớp, Tên lớp, và GVCV.

Thêm dữ liệu vào worksheet: Hàm thêm từng lớp học vào worksheet với các thông tin tương ứng.

Bảo vệ worksheet: Hàm bảo vệ worksheet với mật khẩu, cho phép chọn các ô đã khóa và chưa khóa. Các ô trong cột id sẽ được khóa để bảo vệ.

Xuất tệp Excel: Hàm thiết lập các tiêu đề phản hồi để chỉ định loại tệp và tên tệp. Sau đó, hàm ghi workbook vào phản hồi và kết thúc phản hồi.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi máy chủ.

#### e) Tạo lớp học mới



```
● ● ●

create: async (req, res) => {
  try {
    const { data } = req.body;
    const newClass = await ClassModel.create(data);
    res.json(newClass);
  } catch (error) {
    console.error('Lỗi khi tạo lớp học:', error);
    res.status(500).json({ message: 'Lỗi máy chủ' });
  }
},
```

Hình 3.34. API tạo lớp học mới.

Hàm create là một hàm không đồng bộ được thiết kế để tạo một lớp học mới dựa trên dữ liệu được cung cấp trong yêu cầu. Dưới đây là giải thích từng bước về hoạt động

của nó:

Lấy dữ liệu từ yêu cầu: Hàm bắt đầu bằng việc lấy dữ liệu từ req.body. Dữ liệu này chứa thông tin cần thiết để tạo một lớp học mới.

Tạo lớp học mới: Hàm sử dụng ClassModel.create để tạo một bản ghi mới trong cơ sở dữ liệu với dữ liệu đã được cung cấp.

Trả về kết quả: Nếu lớp học được tạo thành công, hàm sẽ trả về thông tin của lớp học mới dưới dạng JSON.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi máy chủ.

## f) Cập nhật lớp học

```
● ● ●

update: async (req, res) => {
  try {
    const { id } = req.params;
    const { data } = req.body;
    console.log("data", data)
    const classObj = await ClassModel.findOne({ where: { class_id: id, isDelete: false } });
    if (!classObj) {
      return res.status(404).json({ message: 'Không tìm thấy lớp học' });
    }
    await ClassModel.update(data, { where: { class_id: id } });
    res.json({ message: `Cập nhật thành công lớp học có id: ${id}` });
  } catch (error) {
    console.error('Lỗi khi cập nhật lớp học:', error);
    res.status(500).json({ message: 'Lỗi máy chủ' });
  }
}
```

Hình 3.35. API cập nhật lớp học

Hàm update là một hàm không đồng bộ được thiết kế để cập nhật thông tin của một lớp học dựa trên ID của lớp. Dưới đây là giải thích từng bước về hoạt động của nó:

Lấy ID từ yêu cầu: Hàm bắt đầu bằng việc lấy id từ các tham số của yêu cầu (req.params).

Lấy dữ liệu cập nhật từ yêu cầu: Hàm lấy dữ liệu cần cập nhật từ req.body.

Tìm lớp học theo ID: Hàm sử dụng ClassModel.findOne để tìm lớp học trong cơ sở dữ liệu dựa trên class\_id và điều kiện isDelete: false. Nếu không tìm thấy lớp học, hàm sẽ trả về mã trạng thái 404 và thông báo rằng không tìm thấy lớp học.

Cập nhật thông tin lớp học: Nếu tìm thấy lớp học, hàm sử dụng ClassModel.update để cập nhật dữ liệu mới cho lớp học dựa trên class\_id.

Trả về kết quả: Nếu cập nhật thành công, hàm sẽ trả về thông báo thành công với mã trạng thái 200 và thông tin về lớp học đã được cập nhật.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi máy chủ.

### g) Xóa lớp học

```
● ● ●

delete: async (req, res) => {
  try {
    const { id } = req.params;
    const classObj = await ClassModel.findOne({ where: { class_id: id, isDelete: false } });
    if (!classObj) {
      return res.status(404).json({ message: 'Không tìm thấy lớp học' });
    }
    await ClassModel.update({ isDelete: true }, { where: { class_id: id } });
    res.json({ message: 'Xóa lớp học thành công' });
  } catch (error) {
    console.error('Lỗi khi xóa lớp học:', error);
    res.status(500).json({ message: 'Lỗi máy chủ' });
  }
}
```

Hình 3.36. API xóa khóa học.

Hàm delete là một hàm không đồng bộ được thiết kế để xóa logic một lớp học dựa trên ID của lớp. Dưới đây là giải thích từng bước về hoạt động của nó:

Lấy ID từ yêu cầu: Hàm bắt đầu bằng việc lấy id từ các tham số của yêu cầu (req.params).

Tìm lớp học theo ID: Hàm sử dụng ClassModel.findOne để tìm lớp học trong cơ sở dữ liệu dựa trên class\_id và điều kiện isDelete: false. Nếu không tìm thấy lớp học, hàm sẽ trả về mã trạng thái 404 và thông báo rằng không tìm thấy lớp học.

Cập nhật trạng thái xóa: Nếu tìm thấy lớp học, hàm sẽ sử dụng ClassModel.update để cập nhật thuộc tính isDelete thành true cho lớp học dựa trên class\_id. Đây là cách xóa logic, tức là lớp học sẽ không bị xóa vĩnh viễn khỏi cơ sở dữ liệu mà chỉ bị ẩn đi.

Trả về kết quả: Nếu cập nhật trạng thái xóa thành công, hàm sẽ trả về thông báo thành công với mã trạng thái 200.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khôi catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi máy chủ.

### h) Thay đổi trạng thái lớp

```
● ● ●

isDeleteToTrue: async (req, res) => {
  try {
    const { page, size } = req.query;

    const attributes = ['class_id', 'teacher_id', 'className', 'classCode', 'isDelete'];
    const whereClause = { isDelete: true };

    if (page && size) {
      const offset = (page - 1) * size;
      const limit = parseInt(size, 10);

      const { count, rows: classes } = await ClassModel.findAndCountAll({
        include: [
          {
            model: TeacherModel,
            attributes: ['name']
          }
        ],
        attributes: attributes,
        where: whereClause,
        offset: offset,
        limit: limit
      });

      return res.json({
        total: count,
        classes: classes
      });
    } else {
      const classes = await ClassModel.findAll({
        include: [
          {
            model: TeacherModel,
            attributes: ['name']
          }
        ],
        attributes: attributes,
        where: whereClause
      });

      return res.json(classes);
    }
  } catch (error) {
    console.error('Error:', error);
    res.status(500).json({ message: 'Internal Server Error' });
  }
}
```

Hình 3.37. API thay đổi trạng thái lớp học.

Hàm isDeleteToTrue là một hàm không đồng bộ được thiết kế để lấy danh sách các lớp học đã bị xóa (có thuộc tính isDelete là true). Dưới đây là giải thích từng bước về hoạt động của nó:

Lấy thông tin phân trang từ yêu cầu: Hàm bắt đầu bằng việc lấy các tham số page và size từ req.query. Đây là các tham số dùng để xác định trang hiện tại và kích thước của mỗi trang.

Xác định các thuộc tính và điều kiện lọc: Hàm xác định các thuộc tính cần lấy từ bảng ClassModel và điều kiện lọc để chỉ lấy các lớp học đã bị xóa (isDelete: true).

Kiểm tra và xử lý phân trang:

Nếu có thông tin về phân trang (page và size), hàm tính toán offset và limit để lấy dữ liệu theo trang:

offset được tính dựa trên trang hiện tại và kích thước trang: (page - 1) \* size.

limit là số lượng bản ghi cần lấy trong mỗi trang.

Hàm sử dụng ClassModel.findAndCountAll để lấy danh sách các lớp học cùng với số lượng tổng cộng các lớp học thỏa mãn điều kiện.

Kết quả trả về bao gồm tổng số bản ghi (count) và danh sách các lớp học (classes).

Xử lý khi không có phân trang: Nếu không có thông tin phân trang, hàm sử dụng ClassModel.findAll để lấy tất cả các lớp học thỏa mãn điều kiện.

Bao gồm thông tin giáo viên: Trong cả hai trường hợp, hàm đều bao gồm thông tin giáo viên bằng cách sử dụng include với TeacherModel, chỉ lấy thuộc tính name của giáo viên.

Trả về kết quả: Kết quả được trả về dưới dạng JSON, bao gồm tổng số bản ghi và danh sách các lớp học khi có phân trang, hoặc chỉ danh sách các lớp học khi không có phân trang.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo chỉ ra lỗi máy chủ, cùng với chi tiết lỗi.

### i) Đảo ngược trạng thái lớp học

```
IsDelete: async (req, res) => {
  try {
    const { id } = req.params;
    const classes = await ClassModel.findOne({ where: { class_id: id } });
    if (!classes) {
      return res.status(404).json({ message: 'Không tìm thấy lớp học' });
    }
    const updatedIsDeleted = !classes.isDelete;
    await ClassModel.update({ isDelete: updatedIsDeleted }, { where: { class_id: id } });
    res.json({ message: `Đã đảo ngược trạng thái isDelete thành ${updatedIsDeleted}` });
  } catch (error) {
    console.error('Lỗi khi đảo ngược trạng thái isDelete của lớp học:', error);
    res.status(500).json({ message: 'Lỗi máy chủ' });
  }
}
```

Hình 3.38. API đảo ngược trạng thái lớp học

Hàm IsDelete là một hàm không đồng bộ được thiết kế để đảo ngược trạng thái

isDelete của một lớp học dựa trên ID của lớp. Dưới đây là giải thích từng bước về hoạt động của nó:

Lấy ID từ yêu cầu: Hàm bắt đầu bằng việc lấy id từ các tham số của yêu cầu (req.params).

Tìm lớp học theo ID: Hàm sử dụng ClassModel.findOne để tìm lớp học trong cơ sở dữ liệu dựa trên class\_id. Nếu không tìm thấy lớp học, hàm sẽ trả về mã trạng thái 404 và thông báo rằng không tìm thấy lớp học.

Đảo ngược trạng thái isDelete: Nếu tìm thấy lớp học, hàm sẽ đảo ngược trạng thái isDelete của lớp học. Giá trị mới của isDelete được tính bằng cách lấy giá trị hiện tại và đảo ngược nó (từ false thành true hoặc từ true thành false).

Cập nhật trạng thái isDelete: Hàm sẽ sử dụng ClassModel.update để cập nhật giá trị mới của isDelete cho lớp học dựa trên class\_id.

Trả về kết quả: Nếu cập nhật trạng thái isDelete thành công, hàm sẽ trả về thông báo thành công với mã trạng thái 200, thông báo rằng trạng thái isDelete đã được đảo ngược và hiển thị giá trị mới.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi máy chủ.

### 3.6.5.5. API của năm học

#### a) Lấy thông tin tất cả lớp học.



```
index: async (req, res) => {
  try {
    const academicYears = await AcademicYearModel.findAll();
    res.json(academicYears);
  } catch (error) {
    console.error('Lỗi khi lấy tất cả các năm học:', error);
    res.status(500).json({ message: 'Lỗi server' });
  }
}
```

Hình 3.39. API lấy thông tin tất cả lớp học.

Hàm index là một hàm không đồng bộ được thiết kế để lấy danh sách tất cả các năm học từ cơ sở dữ liệu. Dưới đây là giải thích từng bước về hoạt động của nó:

Truy vấn cơ sở dữ liệu: Hàm sử dụng AcademicYearModel.findAll để tìm và lấy danh sách tất cả các năm học trong cơ sở dữ liệu.

Trả về kết quả: Kết quả của truy vấn, tức là danh sách các năm học, được trả về dưới dạng JSON.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi máy chủ.

### b) Tạo mới một năm học



```
● ● ●

create: async (req, res) => {
  try {
    const { data } = req.body;
    const year = parseInt(data, 10);

    if (isNaN(year)) {
      return res.status(400).json({ message: 'Dữ liệu năm không hợp lệ' });
    }

    const startDate = new Date(Date.UTC(year, 0, 1, 0, 0, 0));
    const endDate = new Date(Date.UTC(year, 11, 31, 23, 59, 59));
    const description = `Năm học ${year}-${year + 1}`;

    const existingAcademicYear = await AcademicYearModel.findOne({
      where: { startDate, endDate },
    });

    if (existingAcademicYear) {
      return res.status(200).json(existingAcademicYear);
    }

    const newAcademicYear = await AcademicYearModel.create({
      startDate,
      endDate,
      description,
      isDelete: 0,
      createdAt: new Date(),
      updatedAt: new Date(),
    });

    res.status(201).json(newAcademicYear);
  } catch (error) {
    console.error('Lỗi khi tạo năm học mới:', error);
    res.status(500).json({ message: 'Lỗi server' });
  }
}
```

Hình 3.40. API tạo mới năm học.

Hàm create là một hàm không đồng bộ được thiết kế để tạo một năm học mới dựa trên dữ liệu được cung cấp trong yêu cầu. Dưới đây là giải thích từng bước về hoạt động của nó:

Lấy dữ liệu từ yêu cầu: Hàm bắt đầu bằng việc lấy dữ liệu từ req.body. Dữ liệu này chưa năm học cần tạo.

Chuyển đổi dữ liệu năm học: Hàm chuyển đổi dữ liệu năm học thành kiểu số nguyên bằng parseInt. Nếu dữ liệu không hợp lệ (không phải là số), hàm sẽ trả về mã trạng thái 400 và thông báo rằng dữ liệu năm không hợp lệ.

Tạo các ngày bắt đầu và kết thúc năm học: Hàm sử dụng Date.UTC để tạo các ngày bắt đầu (startDate) và kết thúc (endDate) cho năm học dựa trên năm đã chuyển đổi. Mô tả năm học cũng được tạo từ năm này.

Kiểm tra năm học đã tồn tại: Hàm kiểm tra xem năm học đã tồn tại trong cơ sở dữ liệu hay chưa bằng cách sử dụng AcademicYearModel.findOne với điều kiện startDate và endDate. Nếu năm học đã tồn tại, hàm trả về thông tin năm học hiện tại với mã trạng thái 200.

Tạo năm học mới: Nếu năm học chưa tồn tại, hàm tạo một năm học mới bằng cách sử dụng AcademicYearModel.create với các thông tin startDate, endDate, description, và các trường khác. Ngày tạo và cập nhật được thiết lập bằng new Date().

Trả về kết quả: Nếu tạo năm học mới thành công, hàm trả về thông tin của năm học mới với mã trạng thái 201.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi máy chủ.

### c) Lấy thông tin bởi ID

```
● ● ●  
getByID: async (req, res) => {  
    try {  
        const { id } = req.params;  
        const academicYear = await AcademicYearModel.findOne({ where: {  
            academic_year_id: id } });  
        if (!academicYear) {  
            return res.status(404).json({ message: 'Không tìm thấy năm học' });  
        }  
        res.json(academicYear);  
    } catch (error) {  
        console.error('Lỗi khi tìm kiếm năm học:', error);  
        res.status(500).json({ message: 'Lỗi server' });  
    }  
}
```

Hình 3.41. API lấy thông tin một lớp học bởi id.

Hàm getByID là một hàm không đồng bộ được thiết kế để lấy thông tin của một

năm học cụ thể dựa trên ID của năm học. Dưới đây là giải thích từng bước về hoạt động của nó:

Lấy ID từ yêu cầu: Hàm bắt đầu bằng việc lấy id từ các tham số của yêu cầu (req.params).

Truy vấn cơ sở dữ liệu: Hàm sử dụng AcademicYearModel.findOne để tìm năm học trong cơ sở dữ liệu dựa trên academic\_year\_id. Điều này tìm kiếm một bản ghi năm học có ID khớp với id đã cung cấp.

Kiểm tra kết quả: Sau khi thực hiện truy vấn, hàm kiểm tra xem có năm học nào được tìm thấy hay không. Nếu không có năm học nào khớp, hàm sẽ trả về mã trạng thái 404 cùng với thông báo rằng không tìm thấy năm học.

Trả về kết quả: Nếu tìm thấy năm học, hàm sẽ trả về thông tin của năm học dưới dạng JSON.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi máy chủ.

#### d) Cập nhật thông tin năm học



```
update: async (req, res) => {
  try {
    const { id } = req.params;
    const { data } = req.body;
    const academicYear = await AcademicYearModel.findOne({ where: {
      academic_year_id: id
    } });
    if (!academicYear) {
      return res.status(404).json({ message: 'Không tìm thấy năm học' });
    }
    await AcademicYearModel.update(data, { where: { academic_year_id: id } });
    res.json({ message: `Cập nhật thành công năm học có ID: ${id}` });
  } catch (error) {
    console.error('Lỗi khi cập nhật năm học:', error);
    res.status(500).json({ message: 'Lỗi server' });
  }
}
```

Hình 3.42.API cập nhật thông tin lớp học

Hàm update là một hàm không đồng bộ được thiết kế để cập nhật thông tin của một năm học dựa trên ID của năm học. Dưới đây là giải thích từng bước về hoạt động của nó:

Lấy ID từ yêu cầu: Hàm bắt đầu bằng việc lấy id từ các tham số của yêu cầu (req.params).

Lấy dữ liệu cập nhật từ yêu cầu: Hàm lấy dữ liệu cần cập nhật từ req.body.

Tìm năm học theo ID: Hàm sử dụng AcademicYearModel.findOne để tìm năm học trong cơ sở dữ liệu dựa trên academic\_year\_id. Nếu không tìm thấy năm học, hàm sẽ trả về mã trạng thái 404 và thông báo rằng không tìm thấy năm học.

Cập nhật thông tin năm học: Nếu tìm thấy năm học, hàm sẽ sử dụng AcademicYearModel.update để cập nhật dữ liệu mới cho năm học dựa trên academic\_year\_id.

Trả về kết quả: Nếu cập nhật thành công, hàm sẽ trả về thông báo thành công với mã trạng thái 200 và thông tin về năm học đã được cập nhật.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi máy chủ.

#### e) Xóa một lớp học

```
● ● ●  
delete: async (req, res) => {  
    try {  
        const { id } = req.params;  
        await AcademicYearModel.destroy({ where: { academic_year_id: id } });  
        res.json({ message: 'Xóa năm học thành công' });  
    } catch (error) {  
        console.error('Lỗi khi xóa năm học:', error);  
        res.status(500).json({ message: 'Lỗi server' });  
    }  
}
```

Hình 3.43. API xóa lớp học.

Hàm delete là một hàm không đồng bộ được thiết kế để xóa một năm học dựa trên ID của năm học. Dưới đây là giải thích từng bước về hoạt động của nó:

Lấy ID từ yêu cầu: Hàm bắt đầu bằng việc lấy id từ các tham số của yêu cầu (req.params).

Xóa năm học: Hàm sử dụng AcademicYearModel.destroy để xóa năm học trong cơ sở dữ liệu dựa trên academic\_year\_id. Điều này sẽ xóa vĩnh viễn bản ghi năm học có ID khớp với id đã cung cấp.

Trả về kết quả: Nếu xóa thành công, hàm sẽ trả về thông báo thành công với mã

trạng thái 200.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi máy chủ.

#### f) Lấy năm học bị ẩn

```
● ● ●  
isDeleteToTrue: async (req, res) => {  
    try {  
        const academicYear = await AcademicYearModel.findAll({ where: { isDelete: true } });  
        if (!academicYear) {  
            return res.status(404).json({ message: 'Không tìm thấy academic year' });  
        }  
        res.json(academicYear);  
    } catch (error) {  
        console.error('Lỗi tìm kiếm:', error);  
        res.status(500).json({ message: 'Lỗi server' });  
    }  
}
```

Hình 3.44. API lấy lớp học trạng thái ẩn.

Hàm isDeleteToTrue là một hàm không đồng bộ được thiết kế để lấy danh sách tất cả các năm học có thuộc tính isDelete là true. Dưới đây là giải thích từng bước về hoạt động của nó:

Truy vấn cơ sở dữ liệu: Hàm sử dụng AcademicYearModel.findAll để tìm tất cả các năm học trong cơ sở dữ liệu mà có thuộc tính isDelete bằng true.

Kiểm tra kết quả: Sau khi thực hiện truy vấn, hàm kiểm tra xem có năm học nào được tìm thấy hay không. Nếu không có năm học nào khớp, hàm sẽ trả về mã trạng thái 404 cùng với thông báo rằng không tìm thấy năm học.

Trả về kết quả: Nếu tìm thấy năm học, kết quả sẽ được trả về dưới dạng JSON.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi máy chủ.

### g) Lấy danh sách năm học được ẩn.

```
● ● ●  
isDeleteToFalse: async (req, res) => {  
    try {  
        const academicYear = await AcademicYearModel.findAll({ where: { isDelete: false } });  
        if (!academicYear) {  
            return res.status(404).json({ message: 'Không tìm thấy academic year' });  
        }  
        res.json(academicYear);  
    } catch (error) {  
        console.error('Lỗi tìm kiếm academic year:', error);  
        res.status(500).json({ message: 'Lỗi server' });  
    }  
}
```

Hình 3.45. API lấy lớp học trạng thái không ẩn.

Hàm isDeleteToFalse là một hàm không đồng bộ được thiết kế để lấy danh sách tất cả các năm học có thuộc tính isDelete là false. Dưới đây là giải thích từng bước về hoạt động của nó:

Truy vấn cơ sở dữ liệu: Hàm sử dụng AcademicYearModel.findAll để tìm tất cả các năm học trong cơ sở dữ liệu mà có thuộc tính isDelete bằng false.

Kiểm tra kết quả: Sau khi thực hiện truy vấn, hàm kiểm tra xem có năm học nào được tìm thấy hay không. Nếu không có năm học nào khớp, hàm sẽ trả về mã trạng thái 404 cùng với thông báo rằng không tìm thấy năm học.

Trả về kết quả: Nếu tìm thấy năm học, kết quả sẽ được trả về dưới dạng JSON.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi máy chủ.

### h) Đảo ngược trạng thái lớp học

```
● ● ●  
IsDelete: async (req, res) => {  
    try {  
        const { id } = req.params;  
        const academicYear = await AcademicYearModel.findOne({ where: { class_id: id } });  
        if (!academicYear) {  
            return res.status(404).json({ message: 'Không tìm thấy academic year' });  
        }  
        const updatedIsDeleted = !academicYear.isDelete;  
        await AcademicYearModel.update({ isDelete: updatedIsDeleted }, { where: { class_id: id } });  
        res.json({ message: `Đã đảo ngược trạng thái isDelete thành ${updatedIsDeleted}` });  
    } catch (error) {  
        console.error('Lỗi khi đảo ngược trạng thái isDelete của academic year:', error);  
        res.status(500).json({ message: 'Lỗi máy chủ' });  
    }  
}
```

Hình 3.46. API đảo ngược trạng thái năm học.

Hàm IsDelete là một hàm không đồng bộ được thiết kế để đảo ngược trạng thái isDelete của một năm học dựa trên ID của năm học. Dưới đây là giải thích từng bước về hoạt động của nó:

Lấy ID từ yêu cầu: Hàm bắt đầu bằng việc lấy id từ các tham số của yêu cầu (req.params).

Tìm năm học theo ID: Hàm sử dụng AcademicYearModel.findOne để tìm năm học trong cơ sở dữ liệu dựa trên class\_id. Nếu không tìm thấy năm học, hàm sẽ trả về mã trạng thái 404 và thông báo rằng không tìm thấy năm học.

Đảo ngược trạng thái isDelete: Nếu tìm thấy năm học, hàm sẽ đảo ngược trạng thái isDelete của năm học. Giá trị mới của isDelete được tính bằng cách lấy giá trị hiện tại và đảo ngược nó (từ false thành true hoặc từ true thành false).

Cập nhật trạng thái isDelete: Hàm sẽ sử dụng AcademicYearModel.update để cập nhật giá trị mới của isDelete cho năm học dựa trên class\_id.

Trả về kết quả: Nếu cập nhật trạng thái isDelete thành công, hàm sẽ trả về thông báo thành công với mã trạng thái 200, thông báo rằng trạng thái isDelete đã được đảo ngược và hiển thị giá trị mới.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi máy chủ.

### 3.6.5.6. API của học kì

#### a) Lấy tất cả học kì

```
index: async (req, res) => {
  try {
    const semesters = await SemesterModel.findAll();
    res.json(semesters);
  } catch (error) {
    console.error('Lỗi khi lấy tất cả các học kỳ:', error);
    res.status(500).json({ message: 'Lỗi server' });
  }
}
```

Hình 3.47. API lấy tất cả học kì.

Hàm index là một hàm không đồng bộ được thiết kế để lấy tất cả các học kỳ từ cơ sở dữ liệu. Dưới đây là giải thích từng bước về hoạt động của nó:

Tìm tất cả các học kỳ: Hàm sử dụng SemesterModel.findAll để lấy tất cả các học kỳ từ cơ sở dữ liệu.

Trả về kết quả: Nếu thành công, hàm sẽ trả về danh sách các học kỳ dưới dạng JSON.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khôi catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi server.

### b) Tạo học kì mới

```
● ● ●  
create: async (req, res) => {  
    try {  
        const { data } = req.body;  
        const newSemester = await SemesterModel.create(data);  
        res.json(newSemester);  
    } catch (error) {  
        console.error('Lỗi khi tạo học kỳ mới:', error);  
        res.status(500).json({ message: 'Lỗi server' });  
    }  
}
```

Hình 3.48. API tạo học kì mới.

Hàm create là một hàm không đồng bộ được thiết kế để tạo một học kỳ mới trong cơ sở dữ liệu. Dưới đây là giải thích từng bước về hoạt động của nó:

Nhận dữ liệu từ yêu cầu: Hàm bắt đầu bằng việc lấy dữ liệu từ thân yêu cầu (req.body).

Tạo học kỳ mới: Hàm sử dụng SemesterModel.create để tạo một học kỳ mới trong cơ sở dữ liệu dựa trên dữ liệu đã nhận.

Trả về kết quả: Nếu thành công, hàm sẽ trả về đối tượng học kỳ mới được tạo dưới dạng JSON.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khôi catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi server.

### c) Lấy thông tin học kỳ

```
● ● ●

getByID: async (req, res) => {
  try {
    const { id } = req.params;
    const semester = await SemesterModel.findOne({ where: { semester_id: id } });
    if (!semester) {
      return res.status(404).json({ message: 'Không tìm thấy học kỳ' });
    }
    res.json(semester);
  } catch (error) {
    console.error('Lỗi khi tìm kiếm học kỳ:', error);
    res.status(500).json({ message: 'Lỗi server' });
  }
}
```

Hình 3.49. API lấy thông tin của một học kì.

Hàm getByID là một hàm không đồng bộ được thiết kế để lấy thông tin của một học kỳ dựa trên ID của học kỳ đó. Dưới đây là giải thích từng bước về hoạt động của nó:

Lấy ID từ yêu cầu: Hàm bắt đầu bằng việc lấy id từ các tham số của yêu cầu (req.params).

Tìm học kỳ theo ID: Hàm sử dụng SemesterModel.findOne để tìm học kỳ trong cơ sở dữ liệu dựa trên semester\_id. Nếu không tìm thấy học kỳ, hàm sẽ trả về mã trạng thái 404 và thông báo rằng không tìm thấy học kỳ.

Trả về kết quả: Nếu tìm thấy học kỳ, hàm sẽ trả về đối tượng học kỳ dưới dạng JSON.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi server.

### d) Cập nhật học kì

```
● ● ●

update: async (req, res) => {
  try {
    const { id } = req.params;
    const { data } = req.body;
    const semester = await SemesterModel.findOne({ where: { semester_id: id } });
    if (!semester) {
      return res.status(404).json({ message: 'Không tìm thấy học kỳ' });
    }
    await SemesterModel.update(data, { where: { semester_id: id } });
    res.json({ message: 'Cập nhật thành công học kỳ có ID: ${id}' });
  } catch (error) {
    console.error('Lỗi khi cập nhật học kỳ:', error);
    res.status(500).json({ message: 'Lỗi server' });
  }
}
```

Hình 3.50. API cập nhật học kì.

Hàm update là một hàm không đồng bộ được thiết kế để cập nhật thông tin của một học kỳ dựa trên ID của học kỳ đó. Dưới đây là giải thích từng bước về hoạt động của nó:

Lấy ID và dữ liệu từ yêu cầu: Hàm bắt đầu bằng việc lấy id từ các tham số của yêu cầu (req.params) và lấy dữ liệu từ thân yêu cầu (req.body).

Tìm học kỳ theo ID: Hàm sử dụng SemesterModel.findOne để tìm học kỳ trong cơ sở dữ liệu dựa trên semester\_id. Nếu không tìm thấy học kỳ, hàm sẽ trả về mã trạng thái 404 và thông báo rằng không tìm thấy học kỳ.

Cập nhật thông tin học kỳ: Nếu tìm thấy học kỳ, hàm sẽ sử dụng SemesterModel.update để cập nhật thông tin của học kỳ dựa trên dữ liệu mới cung cấp và semester\_id.

Trả về kết quả: Nếu cập nhật thành công, hàm sẽ trả về thông báo thành công với mã trạng thái 200, thông báo rằng học kỳ có ID đã được cập nhật.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi server.

### e) Xóa học kỳ

```
delete: async (req, res) => {
  try {
    const { id } = req.params;
    await SemesterModel.destroy({ where: { semester_id: id } });
    res.json({ message: 'Xóa học kỳ thành công' });
  } catch (error) {
    console.error('Lỗi khi xóa học kỳ:', error);
    res.status(500).json({ message: 'Lỗi server' });
  }
}
```

Hình 3.51. API xóa một học kỳ.

Hàm delete là một hàm không đồng bộ được thiết kế để xóa một học kỳ dựa trên ID của học kỳ đó. Dưới đây là giải thích từng bước về hoạt động của nó:

Lấy ID từ yêu cầu: Hàm bắt đầu bằng việc lấy id từ các tham số của yêu cầu (req.params).

Xóa học kỳ: Hàm sử dụng SemesterModel.destroy để xóa học kỳ trong cơ sở dữ liệu dựa trên semester\_id.

Trả về kết quả: Nếu xóa thành công, hàm sẽ trả về thông báo thành công với mã trạng thái 200, thông báo rằng học kỳ đã được xóa.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi server.

## f) Lấy danh sách học kì bị ẩn

```
● ● ●

isDeleteToTrue: async (req, res) => {
  try {
    const semester = await SemesterModel.findAll({ where: { isDelete: true } });
    if (!semester) {
      return res.status(404).json({ message: 'Không tìm thấy semester' });
    }
    res.json(semester);
  } catch (error) {
    console.error('Lỗi tìm kiếm semester:', error);
    res.status(500).json({ message: 'Lỗi server' });
  }
}
```

Hình 3.52. API lấy danh sách học kì bị ẩn

Hàm isDeleteToTrue là một hàm không đồng bộ được thiết kế để lấy tất cả các học kỳ có thuộc tính isDelete là true. Dưới đây là giải thích từng bước về hoạt động của nó:

Tìm tất cả các học kỳ có isDelete là true: Hàm sử dụng SemesterModel.findAll để lấy tất cả các học kỳ trong cơ sở dữ liệu có thuộc tính isDelete là true.

Trả về kết quả: Nếu tìm thấy bất kỳ học kỳ nào, hàm sẽ trả về danh sách các học kỳ đó dưới dạng JSON. Nếu không tìm thấy, hàm sẽ trả về mã trạng thái 404 và thông báo rằng không tìm thấy học kỳ.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi server.

## g) Lấy danh sách học kì không bị ẩn.

```
● ● ●

isDeleteToFalse: async (req, res) => {
  try {
    const semester = await SemesterModel.findAll({ where: { isDelete: false } });
    if (!semester) {
      return res.status(404).json({ message: 'Không tìm thấy semester' });
    }
    res.json(semester);
  } catch (error) {
    console.error('Lỗi tìm kiếm semester:', error);
    res.status(500).json({ message: 'Lỗi server' });
  }
}
```

Hình 3.53. API lấy danh sách học kì không ẩn.

Hàm isDeleteToFalse là một hàm không đồng bộ được thiết kế để lấy tất cả các học kỳ có thuộc tính isDelete là false. Dưới đây là giải thích từng bước về hoạt động của nó:

Tìm tất cả các học kỳ có isDelete là false: Hàm sử dụng SemesterModel.findAll để

lấy tất cả các học kỳ trong cơ sở dữ liệu có thuộc tính isDelete là false.

Trả về kết quả: Nếu tìm thấy bất kỳ học kỳ nào, hàm sẽ trả về danh sách các học kỳ đó dưới dạng JSON. Nếu không tìm thấy, hàm sẽ trả về mã trạng thái 404 và thông báo rằng không tìm thấy học kỳ.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi server.

#### **h) Đảo ngược trạng thái học kì**



```
● ● ●

isDelete: async (req, res) => {
  try {
    const { id } = req.params;
    const semester = await SemesterModel.findOne({ where: { class_id: id } });
    if (!semester) {
      return res.status(404).json({ message: 'Không tìm thấy semester' });
    }
    const updatedIsDeleted = !semester.isDelete;
    await SemesterModel.update({ isDelete: updatedIsDeleted }, { where: { class_id: id } });
    res.json({ message: `Đã đảo ngược trạng thái isDelete thành ${updatedIsDeleted}` });
  } catch (error) {
    console.error('Lỗi khi đảo ngược trạng thái isDelete của semester:', error);
    res.status(500).json({ message: 'Lỗi máy chủ' });
  }
}
```

Hình 3.54. Đảo ngược trạng thái học kì.

Hàm isDelete là một hàm không đồng bộ được thiết kế để đảo ngược trạng thái isDelete của một học kỳ dựa trên ID của học kỳ đó. Dưới đây là giải thích từng bước về hoạt động của nó:

Lấy ID từ yêu cầu: Hàm bắt đầu bằng việc lấy id từ các tham số của yêu cầu (req.params).

Tìm học kỳ theo ID: Hàm sử dụng SemesterModel.findOne để tìm học kỳ trong cơ sở dữ liệu dựa trên class\_id. Nếu không tìm thấy học kỳ, hàm sẽ trả về mã trạng thái 404 và thông báo rằng không tìm thấy học kỳ.

Đảo ngược trạng thái isDelete: Nếu tìm thấy học kỳ, hàm sẽ đảo ngược trạng thái isDelete của học kỳ đó. Giá trị mới của isDelete được tính bằng cách lấy giá trị hiện tại và đảo ngược nó (từ false thành true hoặc từ true thành false).

Cập nhật trạng thái isDelete: Hàm sẽ sử dụng SemesterModel.update để cập nhật giá trị mới của isDelete cho học kỳ dựa trên class\_id.

Trả về kết quả: Nếu cập nhật trạng thái isDelete thành công, hàm sẽ trả về thông báo thành công với mã trạng thái 200, thông báo rằng trạng thái isDelete đã được đảo ngược và hiển thị giá trị mới.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi máy chủ.

### 3.6.5.7. API của khóa học

#### a) Lấy tất cả khóa học và các thông tin liên quan



```
index: async (req, res) => {
  try {
    const courses = await CourseModel.findAll({
      include: [
        {
          model: ClassModel,
          where: { isDelete: false }
        },
        {
          model: TeacherModel,
          where: { isDelete: false },
          attributes: ['teacher_id', 'name', 'teacherCode', 'email', 'permission', 'typeTeacher', 'imgURL'],
        },
        {
          model: SubjectModel,
          where: { isDelete: false }
        },
        {
          model: SemesterAcademicYearModel,
          where: { isDelete: false }
        },
      ],
      where: { isDelete: false }
    });
    res.json(courses);
  } catch (error) {
    console.error('Lỗi khi lấy tất cả các khóa học:', error);
    res.status(500).json({ message: 'Lỗi máy chủ nội bộ' });
  }
}
```

Hình 3.55. API lấy tất cả các khóa học.

Hàm index là một hàm không đồng bộ được thiết kế để lấy tất cả các khóa học từ cơ sở dữ liệu, kèm theo thông tin liên quan từ các mô hình khác như Class, Teacher, Subject và SemesterAcademicYear. Dưới đây là giải thích từng bước về hoạt động của nó:

Tìm tất cả các khóa học không bị xóa: Hàm sử dụng CourseModel.findAll để lấy tất cả các khóa học từ cơ sở dữ liệu với điều kiện isDelete là false.

Bao gồm thông tin từ các mô hình liên quan: Hàm sẽ bao gồm thông tin từ các mô hình liên quan như:

ClassModel: Chỉ bao gồm các lớp học không bị xóa (isDelete: false).

TeacherModel: Chỉ bao gồm các giáo viên không bị xóa (isDelete: false) và chỉ lấy

các thuộc tính cụ thể như teacher\_id, name, teacherCode, email, permission, typeTeacher, và imgURL.

SubjectModel: Chỉ bao gồm các môn học không bị xóa (isDelete: false).

SemesterAcademicYearModel: Chỉ bao gồm các kỳ học không bị xóa (isDelete: false).

Trả về kết quả: Nếu thành công, hàm sẽ trả về danh sách các khóa học kèm theo thông tin liên quan dưới dạng JSON.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khôi catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi máy chủ nội bộ.

### b) Lấy thông tin tất cả khóa học.

```
● ● ●  
getAll: async (req, res) => {  
    try {  
        const courses = await CourseModel.findAll({  
            where: { isDelete: false }  
        });  
        res.json(courses);  
    } catch (error) {  
        console.error('Lỗi khi lấy tất cả các khóa học:', error);  
        res.status(500).json({ message: 'Lỗi máy chủ nội bộ' });  
    }  
}
```

Hình 3.56. API lấy tất cả thông tin khóa học.

Hàm getAll là một hàm không đồng bộ được thiết kế để lấy tất cả các khóa học từ cơ sở dữ liệu mà không bị xóa. Dưới đây là giải thích từng bước về hoạt động của nó:

Tìm tất cả các khóa học không bị xóa: Hàm sử dụng CourseModel.findAll để lấy tất cả các khóa học từ cơ sở dữ liệu với điều kiện isDelete là false.

Trả về kết quả: Nếu thành công, hàm sẽ trả về danh sách các khóa học dưới dạng JSON.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khôi catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi máy chủ nội bộ.

c) Tạo khóa học mới

```
create: async (req, res) => {
  const {
    academic_year_id,
    class_id,
    courseCode,
    courseName,
    semester_id,
    subject_id,
    teacher_id
  } = req.body.data;

  console.log(req.body);
  try {
    let semesterAcademicYear = await SemesterAcademicYearModel.findOne({
      where: {
        semester_id,
        academic_year_id
      }
    });

    if (!semesterAcademicYear) {
      semesterAcademicYear = await SemesterAcademicYearModel.create({
        semester_id,
        academic_year_id,
        isDelete: 0
      });
    }

    await CourseModel.create({
      academic_year_id,
      class_id,
      courseCode,
      courseName,
      id_semester_academic_year: semesterAcademicYear.id_semester_academic_year,
      semester_id,
      subject_id,
      teacher_id
    });
  }

  res.json({ message: `Tạo thành công khóa học` });
} catch (error) {
  console.error(`Lỗi khi cập nhật khóa học:`, error);
  res.status(500).json({ message: 'Lỗi máy chủ' });
}
}
```

Hình 3.57. API tạo khóa học mới.

Hàm create là một hàm không đồng bộ được thiết kế để tạo một khóa học mới trong cơ sở dữ liệu. Dưới đây là giải thích từng bước về hoạt động của nó:

Nhận dữ liệu từ yêu cầu: Hàm bắt đầu bằng việc lấy các thông tin cần thiết từ thân yêu cầu (req.body.data), bao gồm academic\_year\_id, class\_id, courseCode, columnName, semester\_id, subject\_id, và teacher\_id.

Kiểm tra sự tồn tại của cặp semester\_id và academic\_year\_id: Hàm sử dụng SemesterAcademicYearModel.findOne để kiểm tra xem cặp semester\_id và academic year id đã tồn tại trong cơ sở dữ liệu chưa.

Tạo mới nếu không tồn tại: Nếu cặp này chưa tồn tại, hàm sẽ tạo mới một bản ghi trong bảng SemesterAcademicYearModel với các thông tin semester\_id, academic\_year\_id, và isDelete là 0.

Tạo khóa học mới: Sau khi xác định hoặc tạo mới semesterAcademicYear, hàm sẽ sử dụng CourseModel.create để tạo một khóa học mới trong cơ sở dữ liệu với các thông tin đã nhận và id\_semester\_academic\_year từ bản ghi vừa được xác định hoặc tạo mới.

Trả về kết quả: Nếu thành công, hàm sẽ trả về thông báo thành công với mã trạng thái 200, thông báo rằng khóa học đã được tạo thành công.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi máy chủ.

#### d) Lấy tất cả khóa học kèm thông tin sinh viên đăng ký

```
● ● ●  
getAllWithCourseEnrollment: async (req, res) => {  
    try {  
        const courses = await CourseModel.findAll({  
            attributes: [  
                include: [  
                    [Sequelize.literal(`  
                        SELECT COUNT(*)  
                        FROM course_enrollments AS ce  
                        WHERE ce.course_id = course.course_id  
                        AND ce.isDelete = FALSE  
                    `)], 'enrollmentCount'  
                ]  
            ],  
            include: [  
                {  
                    model: ClassModel,  
                    where: { isDelete: false },  
                },  
                {  
                    model: TeacherModel,  
                    attributes: ['teacher_id', 'name', 'email', 'typeTeacher', 'teacherCode', 'permission', 'imgURL'],  
                    where: { isDelete: false },  
                },  
                {  
                    model: SubjectModel,  
                    where: { isDelete: false },  
                },  
                {  
                    model: SemesterAcademicYearModel,  
                    where: { isDelete: false },  
                    include: [  
                        {  
                            model: SemesterModel,  
                            where: { isDelete: false },  
                        },  
                        {  
                            model: AcademicYearModel,  
                            where: { isDelete: false },  
                        }  
                    ]  
                },  
                where: { isDelete: false },  
                order: [['course_id', 'DESC']]  
            );  
  
            res.json(courses);  
        } catch (error) {  
            console.error('Lỗi khi lấy tất cả các khóa học:', error);  
            res.status(500).json({ message: 'Lỗi máy chủ nội bộ' });  
        }  
    }  
}
```

Hình 3.58. API lấy tất cả thông tin khóa học và số lượng sinh viên đăng ký

Hàm getAllWithCourseEnrollment là một hàm không đồng bộ được thiết kế để lấy tất cả các khóa học từ cơ sở dữ liệu, kèm theo thông tin liên quan từ các mô hình khác như Class, Teacher, Subject, SemesterAcademicYear, và số lượng đăng ký khóa học. Dưới đây là giải thích từng bước về hoạt động của nó:

Lấy tất cả các khóa học không bị xóa: Hàm sử dụng CourseModel.findAll để lấy tất cả các khóa học từ cơ sở dữ liệu với điều kiện isDelete là false.

Bao gồm thông tin từ các mô hình liên quan:

ClassModel: Bao gồm thông tin lớp học, chỉ lấy các lớp không bị xóa (isDelete: false).

TeacherModel: Bao gồm thông tin giáo viên, chỉ lấy các giáo viên không bị xóa (isDelete: false) và các thuộc tính cụ thể như teacher\_id, name, email, typeTeacher, teacherCode, permission, và imgURL.

SubjectModel: Bao gồm thông tin môn học, chỉ lấy các môn không bị xóa (isDelete: false).

SemesterAcademicYearModel: Bao gồm thông tin kỳ học, chỉ lấy các kỳ không bị xóa (isDelete: false), kèm theo thông tin từ SemesterModel và AcademicYearModel, cũng chỉ lấy các mục không bị xóa.

Tính số lượng đăng ký khóa học: Sử dụng Sequelize.literal để thêm thuộc tính enrollmentCount, tính tổng số đăng ký không bị xóa trong bảng course\_enrollments cho từng khóa học.

Sắp xếp theo ID khóa học: Sắp xếp kết quả theo course\_id theo thứ tự giảm dần.

Trả về kết quả: Nếu thành công, hàm sẽ trả về danh sách các khóa học kèm theo thông tin liên quan và số lượng đăng ký dưới dạng JSON.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi máy chủ nội bộ.

### e) Lấy thông tin một khóa học

```
● ● ●

getByID: async (req, res) => {
  try {
    console.log("aaaaa");
    const { id } = req.params;
    const course = await CourseModel.findAll({
      attributes: [
        include: [
          [Sequelize.literal(`(
            SELECT COUNT(*)
            FROM course_enrollments AS ce
            WHERE ce.course_id = course.course_id
            AND ce.isDelete = FALSE
          )`), 'enrollmentCount']
        ],
        include: [
          {
            model: ClassModel,
            where: { isDelete: false },
            required: true
          },
          {
            model: TeacherModel,
            where: { isDelete: false },
            required: true
          },
          {
            model: SubjectModel,
            where: { isDelete: false },
            required: true
          },
          {
            model: SemesterAcademicYearModel,
            where: { isDelete: false },
            required: true,
            include: [
              {
                model: SemesterModel,
                where: { isDelete: false },
                required: true
              },
              {
                model: AcademicYearModel,
                where: { isDelete: false },
                required: true
              }
            ]
          },
          {
            where: {
              isDelete: false,
              course_id: id
            }
          }
        ];
        if (!course) {
          return res.status(404).json({ message: 'Không tìm thấy khóa học' });
        }
        res.json(course);
      } catch (error) {
        console.error('Lỗi khi tìm kiếm khóa học:', error);
        res.status(500).json({ message: 'Lỗi máy chủ' });
      }
    })
  }
}
```

Hình 3.59. API lấy thông tin của một lớp học.

Hàm getByID là một hàm không đồng bộ được thiết kế để lấy thông tin chi tiết của một khóa học dựa trên ID của khóa học đó. Dưới đây là giải thích từng bước về hoạt động của nó:

Lấy ID từ yêu cầu: Hàm bắt đầu bằng việc lấy id từ các tham số của yêu cầu (req.params).

Tìm kiếm khóa học: Hàm sử dụng CourseModel.findAll để tìm khóa học trong cơ sở dữ liệu dựa trên course\_id và điều kiện isDelete là false.

Bao gồm thông tin từ các mô hình liên quan:

ClassModel: Bao gồm thông tin lớp học, chỉ lấy các lớp không bị xóa (isDelete: false).

TeacherModel: Bao gồm thông tin giáo viên, chỉ lấy các giáo viên không bị xóa (isDelete: false).

SubjectModel: Bao gồm thông tin môn học, chỉ lấy các môn không bị xóa (isDelete: false).

SemesterAcademicYearModel: Bao gồm thông tin kỳ học, chỉ lấy các kỳ không bị xóa (isDelete: false), kèm theo thông tin từ SemesterModel và AcademicYearModel, cũng chỉ lấy các mục không bị xóa.

Tính số lượng đăng ký khóa học: Sử dụng Sequelize.literal để thêm thuộc tính enrollmentCount, tính tổng số đăng ký không bị xóa trong bảng course\_enrollments cho khóa học.

Kiểm tra kết quả: Nếu không tìm thấy khóa học, hàm sẽ trả về mã trạng thái 404 và thông báo rằng không tìm thấy khóa học.

Trả về kết quả: Nếu tìm thấy khóa học, hàm sẽ trả về thông tin chi tiết của khóa học dưới dạng JSON.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi máy chủ.

## f) Lấy thông tin khóa học bởi id giáo viên

```
● ● ●  
getByIDTeacher: async (req, res) => {  
    try {  
        const { id_teacher } = req.params;  
        const courses = await CourseModel.findAll({ where: { teacher_id: id_teacher, isDelete: false } });  
        if (!courses || courses.length === 0) {  
            return res.status(404).json({ message: 'Không tìm thấy khóa học' });  
        }  
  
        res.json({ course: courses });  
    } catch (error) {  
        console.error('Lỗi khi tìm kiếm khóa học:', error);  
        res.status(500).json({ message: 'Lỗi máy chủ' });  
    }  
}
```

Hình 3.60. API lấy thông tin khóa học bởi id giáo viên.

Hàm getByIDTeacher là một hàm không đồng bộ được thiết kế để lấy tất cả các khóa học do một giáo viên cụ thể giảng dạy dựa trên ID của giáo viên đó. Dưới đây là giải thích từng bước về hoạt động của nó:

Lấy ID của giáo viên từ yêu cầu: Hàm bắt đầu bằng việc lấy id\_teacher từ các tham số của yêu cầu (req.params).

Tìm các khóa học của giáo viên: Hàm sử dụng CourseModel.findAll để tìm tất cả các khóa học trong cơ sở dữ liệu có teacher\_id khớp với id\_teacher và isDelete là false.

Kiểm tra kết quả: Nếu không tìm thấy bất kỳ khóa học nào hoặc danh sách khóa học trống, hàm sẽ trả về mã trạng thái 404 và thông báo rằng không tìm thấy khóa học.

Trả về kết quả: Nếu tìm thấy các khóa học, hàm sẽ trả về danh sách các khóa học dưới dạng JSON.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi máy chủ.

**g) Lấy thông tin chi tiết của một khóa học cụ thể, kèm theo số lượng đăng ký khóa học**



```
getByIdWithCourseEnrollment: async (req, res) => {
  try {
    const { id } = req.params;
    const course = await CourseModel.findAll({
      include: [
        {
          model: ClassModel,
          where: { isDelete: false }
        },
        {
          model: TeacherModel,
          where: { isDelete: false }
        },
        {
          model: SubjectModel,
          where: { isDelete: false }
        },
        {
          model: SemesterAcademicYearModel,
          where: { isDelete: false }
        }
      ],
      where: {
        isDelete: false,
        course_id: id
      }
    });

    if (!course) {
      return res.status(404).json({ message: 'Không tìm thấy khóa học' });
    }

    const enrollmentCount = await CourseEnrollmentModel.count({
      where: {
        course_id: id,
        isDelete: false
      }
    });

    res.json({ course, enrollmentCount });
  } catch (error) {
    console.error('Lỗi khi tìm kiếm khóa học:', error);
    res.status(500).json({ message: 'Lỗi máy chủ' });
  }
}
```

Hình 3.61. Lấy thông tin chi tiết của một khóa học cụ thể.

Hàm getByIdWithCourseEnrollment là một hàm không đồng bộ được thiết kế để lấy thông tin chi tiết của một khóa học cụ thể, kèm theo số lượng đăng ký khóa học. Dưới đây là giải thích từng bước về hoạt động của nó:

Lấy ID từ yêu cầu: Hàm bắt đầu bằng việc lấy id từ các tham số của yêu cầu (req.params).

Tìm kiếm khóa học: Hàm sử dụng CourseModel.findAll để tìm khóa học trong cơ sở dữ liệu dựa trên course\_id và điều kiện isDelete là false. Khóa học được lấy kèm theo thông tin từ các mô hình liên quan như ClassModel, TeacherModel, SubjectModel, và SemesterAcademicYearModel, tất cả đều phải có isDelete là false.

Kiểm tra kết quả: Nếu không tìm thấy khóa học, hàm sẽ trả về mã trạng thái 404 và thông báo rằng không tìm thấy khóa học.

Tính số lượng đăng ký khóa học: Hàm sử dụng CourseEnrollmentModel.count để đếm số lượng đăng ký khóa học không bị xóa (isDelete là false) cho khóa học với course\_id đã cho.

Trả về kết quả: Nếu tìm thấy khóa học và tính được số lượng đăng ký, hàm sẽ trả về thông tin chi tiết của khóa học và số lượng đăng ký dưới dạng JSON.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi máy chủ.

## h) Cập nhật khóa học



```
update: async (req, res) => {
  const { id } = req.params;
  const {
    academic_year_id,
    class_id,
    courseCode,
    courseName,
    semester_id,
    subject_id,
    teacher_id
  } = req.body.data;

  console.log(req.body);
  try {
    const course = await CourseModel.findOne({ where: { course_id: id } });
    if (!course) {
      return res.status(404).json({ message: 'Không tìm thấy khóa học' });
    }

    // Kiểm tra sự tồn tại của cặp semester_id và academic_year_id
    let semesterAcademicYear = await SemesterAcademicYearModel.findOne({
      where: {
        semester_id,
        academic_year_id
      }
    });

    // Nếu không tồn tại, tạo mới
    if (!semesterAcademicYear) {
      semesterAcademicYear = await SemesterAcademicYearModel.create({
        semester_id,
        academic_year_id,
        isDelete: 0
      });
    }

    // Cập nhật khóa học với id_semester_academic_year mới
    await CourseModel.update({
      academic_year_id,
      class_id,
      courseCode,
      courseName,
      id_semester_academic_year: semesterAcademicYear.id_semester_academic_year,
      semester_id,
      subject_id,
      teacher_id
    }, { where: { course_id: id } });

    res.json({ message: `Cập nhật thành công khóa học có id: ${id}` });
  } catch (error) {
    console.error('Lỗi khi cập nhật khóa học:', error);
    res.status(500).json({ message: 'Lỗi máy chủ' });
  }
}
```

Hình 3.62. API cập nhật khóa học

Hàm update là một hàm không đồng bộ được thiết kế để cập nhật thông tin của một khóa học dựa trên ID của khóa học đó. Dưới đây là giải thích từng bước về hoạt động của nó:

Lấy ID và dữ liệu từ yêu cầu: Hàm bắt đầu bằng việc lấy id từ các tham số của yêu

cầu (req.params) và lấy dữ liệu từ thân yêu cầu (req.body.data), bao gồm academic\_year\_id, class\_id, courseCode, courseName, semester\_id, subject\_id, và teacher\_id.

Tìm khóa học theo ID: Hàm sử dụng CourseModel.findOne để tìm khóa học trong cơ sở dữ liệu dựa trên course\_id. Nếu không tìm thấy khóa học, hàm sẽ trả về mã trạng thái 404 và thông báo rằng không tìm thấy khóa học.

Kiểm tra sự tồn tại của cặp semester\_id và academic\_year\_id: Hàm sử dụng SemesterAcademicYearModel.findOne để kiểm tra xem cặp semester\_id và academic\_year\_id đã tồn tại trong cơ sở dữ liệu chưa.

Tạo mới nếu không tồn tại: Nếu cặp này chưa tồn tại, hàm sẽ tạo mới một bản ghi trong bảng SemesterAcademicYearModel với các thông tin semester\_id, academic\_year\_id, và isDelete là 0.

Cập nhật thông tin khóa học: Sau khi xác định hoặc tạo mới semesterAcademicYear, hàm sẽ sử dụng CourseModel.update để cập nhật thông tin của khóa học dựa trên các thông tin đã nhận và id\_semester\_academic\_year từ bản ghi vừa được xác định hoặc tạo mới.

Trả về kết quả: Nếu cập nhật thành công, hàm sẽ trả về thông báo thành công với mã trạng thái 200, thông báo rằng khóa học có ID đã được cập nhật.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi máy chủ.

### i) Xóa một khóa học



```
delete: async (req, res) => {
  try {
    const { id } = req.params;
    await CourseModel.destroy({ where: { course_id: id } });
    res.json({ message: 'Successfully deleted course' });
  } catch (error) {
    console.error('Error deleting course:', error);
    res.status(500).json({ message: 'Server error' });
  }
}
```

Hình 3.63. API xóa một khóa học

Hàm delete là một hàm không đồng bộ được thiết kế để xóa một khóa học dựa trên ID của khóa học đó. Dưới đây là giải thích từng bước về hoạt động của nó:

Lấy ID từ yêu cầu: Hàm bắt đầu bằng việc lấy id từ các tham số của yêu cầu (req.params).

Xóa khóa học: Hàm sử dụng CourseModel.destroy để xóa khóa học trong cơ sở dữ liệu dựa trên course\_id.

Trả về kết quả: Nếu xóa thành công, hàm sẽ trả về thông báo thành công với mã trạng thái 200, thông báo rằng khóa học đã được xóa.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi máy chủ.

#### j) Lấy danh sách khóa học bị ẩn



```
isDeleteToTrue: async (req, res) => {
  try {
    const courses = await CourseModel.findAll({ where: { isDelete: true } });
    if (!courses) {
      return res.status(404).json({ message: 'No courses found' });
    }
    res.json(courses);
  } catch (error) {
    console.error('Error finding courses with isDelete true:', error);
    res.status(500).json({ message: 'Server error' });
  }
}
```

Hình 3.64. API lấy danh sách khóa học bị ẩn

Hàm isDeleteToTrue là một hàm không đồng bộ được thiết kế để lấy tất cả các khóa học có thuộc tính isDelete là true. Dưới đây là giải thích từng bước về hoạt động của nó:

Tìm tất cả các khóa học có isDelete là true: Hàm sử dụng CourseModel.findAll để lấy tất cả các khóa học từ cơ sở dữ liệu với điều kiện isDelete là true.

Kiểm tra kết quả: Nếu không tìm thấy bất kỳ khóa học nào, hàm sẽ trả về mã trạng thái 404 và thông báo rằng không tìm thấy khóa học.

Trả về kết quả: Nếu tìm thấy các khóa học, hàm sẽ trả về danh sách các khóa học dưới dạng JSON.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi máy chủ.

## k) Đảo ngược khóa học

```
● ● ●  
isDelete: async (req, res) => {  
    try {  
        const { id } = req.params;  
        const course = await CourseModel.findOne({ where: { course_id: id } });  
        if (!course) {  
            return res.status(404).json({ message: 'Course not found' });  
        }  
        const updatedIsDeleted = !course.isDelete;  
        await CourseModel.update({ isDelete: updatedIsDeleted }, { where: { course_id: id } });  
        res.json({ message: `Successfully toggled isDelete status to ${updatedIsDeleted}` });  
    } catch (error) {  
        console.error('Error updating isDelete status:', error);  
        res.status(500).json({ message: 'Server error' });  
    }  
}
```

Hình 3.65. API đảo ngược khóa học

Hàm isDelete là một hàm không đồng bộ được thiết kế để đảo ngược trạng thái isDelete của một khóa học dựa trên ID của khóa học đó. Dưới đây là giải thích từng bước về hoạt động của nó:

Lấy ID từ yêu cầu: Hàm bắt đầu bằng việc lấy id từ các tham số của yêu cầu (req.params).

Tìm khóa học theo ID: Hàm sử dụng CourseModel.findOne để tìm khóa học trong cơ sở dữ liệu dựa trên course\_id. Nếu không tìm thấy khóa học, hàm sẽ trả về mã trạng thái 404 và thông báo rằng không tìm thấy khóa học.

Đảo ngược trạng thái isDelete: Nếu tìm thấy khóa học, hàm sẽ đảo ngược trạng thái isDelete của khóa học đó. Giá trị mới của isDelete được tính bằng cách lấy giá trị hiện tại và đảo ngược nó (từ false thành true hoặc từ true thành false).

Cập nhật trạng thái isDelete: Hàm sẽ sử dụng CourseModel.update để cập nhật giá trị mới của isDelete cho khóa học dựa trên course\_id.

Trả về kết quả: Nếu cập nhật trạng thái isDelete thành công, hàm sẽ trả về thông báo thành công với mã trạng thái 200, thông báo rằng trạng thái isDelete đã được đảo ngược và hiển thị giá trị mới.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khôi catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi máy chủ.

### 3.6.5.8. API của đăng ký khóa học

#### a) Lấy danh sách sinh viên đăng ký khóa học



```
getByID: async (req, res) => {
  try {
    const { id } = req.params;
    const course = await CourseEnrollmentModel.findAll({
      include: [
        {
          model: StudentModel,
          where: { isDelete: false }
        },
        {
          model: CourseModel,
          where: { isDelete: false }
        }
      ],
      where: {
        isDelete: false,
        course_id: id
      }
    });
    if (!course) {
      return res.status(404).json({ message: 'Không tìm thấy khóa học' });
    }
    res.json(course);
  } catch (error) {
    console.error('Lỗi khi tìm kiếm khóa học:', error);
    res.status(500).json({ message: 'Lỗi máy chủ' });
  }
}
```

Hình 3.66. API lấy danh sách sinh viên đã đăng ký khóa học

Hàm getByID là một hàm không đồng bộ được thiết kế để lấy thông tin chi tiết của một khóa học cùng với thông tin về các sinh viên đăng ký khóa học đó dựa trên ID của khóa học. Dưới đây là giải thích từng bước về hoạt động của nó:

Lấy ID từ yêu cầu: Hàm bắt đầu bằng việc lấy id từ các tham số của yêu cầu (req.params).

Tìm kiếm khóa học và sinh viên đăng ký: Hàm sử dụng CourseEnrollmentModel.findAll để tìm khóa học trong cơ sở dữ liệu dựa trên course\_id và điều kiện isDelete là false. Kết quả được lấy kèm theo thông tin từ các mô hình liên quan như StudentModel và CourseModel, tất cả đều phải có isDelete là false.

Kiểm tra kết quả: Nếu không tìm thấy khóa học, hàm sẽ trả về mã trạng thái 404 và thông báo rằng không tìm thấy khóa học.

Trả về kết quả: Nếu tìm thấy khóa học, hàm sẽ trả về thông tin chi tiết của khóa học cùng với thông tin về các sinh viên đăng ký dưới dạng JSON.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi máy chủ.

## b) Xuất file excel danh sách sinh viên bằng mảng id sinh viên

```
● ● ●

getExcelCourseEnrollmentWithData: async (req, res) => {
  try {
    const { data } = req.body;
    const { id } = data;

    const workbook = new ExcelJS.Workbook();
    const worksheet = workbook.addWorksheet('Students Form');

    // Lấy danh sách student_id từ CourseEnrollmentModel dựa trên id_detail_courses
    const enrollments = await CourseEnrollmentModel.findAll({
      attributes: ['student_id'],
      where: {
        course_id: id,
        isDelete: false
      }
    });

    // Trích xuất danh sách student_id
    const studentIds = enrollments.map(enrollment => enrollment.student_id);

    // Lấy thông tin học sinh từ StudentModel dựa trên danh sách student_id
    const students = await StudentModel.findAll({
      include: [
        {
          model: ClassModel,
          attributes: ['classCode']
        },
        attributes: ['student_id', 'class_id', 'studentCode', 'email', 'name', 'isDelete'],
        where: {
          isDelete: false,
          student_id: studentIds
        }
      ];
    });

    worksheet.columns = [
      { header: 'id', key: 'id', width: 15 },
      { header: 'Mã lớp', key: 'classCode', width: 15 },
      { header: 'Tên SV', key: 'name', width: 32 },
      { header: 'MSSV', key: 'studentCode', width: 20 },
      { header: 'Email', key: 'email', width: 30 }
    ];

    students.forEach(student => {
      worksheet.addRow({
        id: student.student_id,
        classCode: student.class.classCode,
        name: student.name,
        studentCode: student.studentCode,
        email: student.email
      });
    });

    await worksheet.protect('yourpassword', {
      selectLockedCells: true,
      selectUnlockedCells: true
    });

    worksheet.eachRow((row, pageNumber) => {
      row.eachCell((cell, colNumber) => {
        if (colNumber === 1) {
          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' });
  }
}
```

Hình 3.67. API xuất file excel từ danh sách id sinh viên

Hàm getExcelCourseEnrollmentWithData là một hàm không đồng bộ được thiết kế để tạo tệp Excel chứa thông tin về các sinh viên đăng ký một khóa học dựa trên ID của khóa học đó. Dưới đây là giải thích từng bước về hoạt động của nó:

Lấy dữ liệu từ yêu cầu: Hàm bắt đầu bằng việc lấy data từ thân yêu cầu (req.body) và trích xuất id từ data.

Tạo workbook và worksheet: Sử dụng thư viện ExcelJS để tạo một workbook mới và một worksheet mới có tên "Students Form".

Lấy danh sách sinh viên đăng ký: Sử dụng CourseEnrollmentModel.findAll để lấy danh sách student\_id từ bảng CourseEnrollmentModel dựa trên course\_id và điều kiện isDelete là false.

Lấy thông tin chi tiết của các sinh viên: Sử dụng StudentModel.findAll để lấy thông tin chi tiết của các sinh viên từ bảng StudentModel dựa trên danh sách student\_id đã lấy được ở bước trước và điều kiện isDelete là false. Kết quả bao gồm thông tin về lớp học từ bảng ClassModel.

Định nghĩa cột cho worksheet: Định nghĩa các cột cho worksheet bao gồm id, classCode, name, studentCode, và email.

Thêm thông tin sinh viên vào worksheet: Duyệt qua danh sách sinh viên và thêm từng sinh viên vào worksheet với các thông tin tương ứng.

Bảo vệ worksheet: Bảo vệ worksheet bằng cách khóa các ô ở cột id và mở khóa các ô ở các cột khác.

Thiết lập header cho phản hồi: Thiết lập header cho phản hồi để chỉ định rằng nội dung phản hồi là một tệp Excel.

Ghi workbook vào phản hồi và kết thúc: Ghi workbook vào phản hồi và kết thúc phản hồi.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi máy chủ.

### c) Xuất file excel danh sách sinh viên bằng id khóa học

```
getFormStudentWithDataByCourse: async (req, res) => {
  try {
    const { id } = req.params;

    const workbook = new ExcelJS.Workbook();
    const worksheet = workbook.addWorksheet('Students Form');

    const students = await CourseEnrollmentModel.findAll({
      include: [
        {
          model: StudentModel,
          attributes: ['student_id', 'class_id', 'studentCode', 'email', 'name', 'isDelete'],
          where: {
            isDelete: false,
          },
          include: [
            {
              model: ClassModel,
              attributes: ['classCode', 'classNameShort', 'className'],
              where: {
                isDelete: false,
              }
            }
          ]
        },
        {
          model: CourseModel,
          attributes: ['courseName'],
          where: {
            course_id: id,
            isDelete: false,
          },
        },
        {
          attributes: [],
          where: {
            isDelete: false,
          }
        }
      );
    });

    worksheet.columns = [
      { header: 'Mã lớp', key: 'classCode', width: 15 },
      { header: 'Tên SV', key: 'name', width: 32 },
      { header: 'MSSV', key: 'studentCode', width: 20 },
      { header: 'Email', key: 'email', width: 30 }
    ];
    const index = 0;
    students.forEach(student => {
      worksheet.addRow({
        classCode: student.Student.class.classNameShort,
        name: student.Student.name,
        studentCode: student.Student.studentCode,
        email: student.Student.email
      });
    });
    res.setHeader('Content-Type', 'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet');
    res.setHeader('Content-Disposition', 'attachment; filename="Student.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' });
  }
}
```

Hình 3.68. API xuất file excel bởi id khóa học

Hàm getFormStudentWithDataByCourse là một hàm không đồng bộ được thiết kế để tạo tệp Excel chứa thông tin về các sinh viên đăng ký một khóa học dựa trên ID của khóa học đó. Dưới đây là giải thích từng bước về hoạt động của nó:

Lấy ID từ yêu cầu: Hàm bắt đầu bằng việc lấy id từ các tham số của yêu cầu (req.params).

Tạo workbook và worksheet: Sử dụng thư viện ExcelJS để tạo một workbook mới và một worksheet mới có tên "Students Form".

Lấy danh sách sinh viên đăng ký: Sử dụng CourseEnrollmentModel.findAll để lấy thông tin về các sinh viên đăng ký khóa học từ bảng CourseEnrollmentModel, kết hợp với thông tin chi tiết của sinh viên từ bảng StudentModel và lớp học từ bảng ClassModel. Khóa học và sinh viên chỉ được lấy nếu không bị xóa (isDelete: false).

Định nghĩa cột cho worksheet: Định nghĩa các cột cho worksheet bao gồm classCode, name, studentCode, và email.

Thêm thông tin sinh viên vào worksheet: Duyệt qua danh sách sinh viên và thêm từng sinh viên vào worksheet với các thông tin tương ứng.

Thiết lập header cho phản hồi: Thiết lập header cho phản hồi để chỉ định rằng nội dung phản hồi là một tệp Excel.

Ghi workbook vào phản hồi và kết thúc: Ghi workbook vào phản hồi và kết thúc phản hồi.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi máy chủ.

#### d) API lưu lấy dữ liệu từ excel lưu vào database

```

saveExcel: async (req, res) => {
  const courseId = req.body.data; // course_id

  if (!courseId) {
    return res.status(400).json({ message: "Course ID is required." });
  }

  if (!req.files || req.files.length === 0) {
    return res.status(400).send('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(1);
    const emailsSet = new Set();
    const studentCodesSet = new Set();
    const insertPromises = [];

    const processRow = async (row, rowNumber) => {
      if (rowNumber !== 1) { // Skip the header row
        let email = row.getCell(4).value;
        let studentCode = row.getCell(3).value;

        if (email && typeof email === 'object' && email.hasOwnProperty('text')) {
          email = email.text;
        }

        let insertStudent = false;
        if (!email || emailsSet.has(email)) {
          console.error(`Duplicate or invalid email found at row ${rowNumber}: ${email}`);
        } else if (!studentCode || studentCodesSet.has(studentCode)) {
          console.error(`Duplicate or invalid studentCode found at row ${rowNumber}: ${studentCode}`);
        } else {
          const [student] = await sequelize.query(
            'SELECT * FROM students WHERE studentCode = ? OR email = ?',
            { replacements: [studentCode, email], type: sequelize.QueryTypes.SELECT }
          );
          if (!student) {
            emailsSet.add(email);
            studentCodesSet.add(studentCode);
            insertStudent = true;
            const insertStudentSQL = `INSERT INTO students (class_id, name, studentCode, email)
              VALUES ((SELECT class_id FROM classes WHERE classNameShort = ?), ?, ?, ?)`;
            const studentValues = [
              row.getCell(1).value,
              row.getCell(2).value,
              studentCode,
              email,
            ];
            insertPromises.push(sequelize.query(insertStudentSQL, { replacements: studentValues }));
            .catch(error => {
              console.error(`Error inserting row ${rowNumber}: ${error.message}`);
            }));
          }
        }
        if (insertStudent || !emailsSet.has(email) && !studentCodesSet.has(studentCode)) {
          const insertEnrollmentSQL = `INSERT INTO course_enrollments (student_id, course_id)
            VALUES ((SELECT student_id FROM students WHERE studentCode = ?), ?)`;
          const enrollmentValues = [
            studentCode,
            courseId
          ];
          insertPromises.push(sequelize.query(insertEnrollmentSQL, { replacements: enrollmentValues }));
          .catch(error => {
            console.error(`Error inserting enrollment for row ${rowNumber}: ${error.message}`);
          }));
        }
      }
    };

    const processAllRows = async () => {
      for (let i = 2; i <= worksheet.rowCount; i++) {
        const row = worksheet.getRow(i);
        await processRow(row, i);
      }
    };

    await processAllRows();
    await Promise.all(insertPromises);
    fs.unlinkSync(filePath); // Remove the uploaded file after processing
    return res.status(200).json({ message: "Data has been successfully saved to the database." });
  } catch (error) {
    console.error('Error processing file:', error);
    return res.status(500).json({ message: "An error occurred while processing the file." });
  }
}

```

Hình 3.69. API lưu danh sách sinh viên đăng ký khóa học từ file excel.

Hàm saveExcel là một hàm không đồng bộ được thiết kế để xử lý tệp Excel được tải lên, kiểm tra dữ liệu trong tệp, và lưu thông tin sinh viên cùng với đăng ký khóa học vào cơ sở dữ liệu. Dưới đây là giải thích từng bước về hoạt động của nó:

Kiểm tra dữ liệu yêu cầu:

- Kiểm tra xem courseId có tồn tại trong yêu cầu không. Nếu không, trả về mã trạng thái 400 với thông báo rằng Course ID is required.
- Kiểm tra xem tệp đã được tải lên hay chưa. Nếu không, trả về mã trạng thái 400 với thông báo rằng No file uploaded.

Đọc tệp Excel:

- Sử dụng thư viện ExcelJS để đọc tệp Excel từ đường dẫn được tải lên.
- Lấy worksheet đầu tiên từ workbook.

Xử lý dữ liệu từ tệp Excel:

- Tạo các tập hợp emailsSet và studentCodesSet để theo dõi các email và mã sinh viên đã được xử lý.
- Tạo một danh sách insertPromises để lưu trữ các promises cho các thao tác chèn vào cơ sở dữ liệu.

Xử lý từng hàng dữ liệu:

- Duyệt qua từng hàng trong worksheet, bắt đầu từ hàng thứ 2 (bỏ qua hàng tiêu đề).
- Kiểm tra và trích xuất email và mã sinh viên từ mỗi hàng.
- Kiểm tra tính hợp lệ và sự trùng lặp của email và mã sinh viên.
- Nếu sinh viên chưa tồn tại trong cơ sở dữ liệu, thêm thông tin sinh viên vào cơ sở dữ liệu.
- Sau đó, thêm đăng ký khóa học vào cơ sở dữ liệu.

Xử lý tất cả các hàng:

- Duyệt qua tất cả các hàng và gọi hàm processRow cho mỗi hàng để xử lý dữ liệu.

Chờ tất cả các promises hoàn thành:

- Chờ tất cả các promises trong insertPromises hoàn thành.

Xóa tệp đã tải lên:

- Xóa tệp đã tải lên sau khi xử lý xong.

Trả về kết quả:

- Nếu tất cả đều thành công, trả về mã trạng thái 200 với thông báo rằng dữ liệu đã được lưu thành công vào cơ sở dữ liệu.
- Nếu có lỗi xảy ra trong quá trình này, trả về mã trạng thái 500 với thông báo lỗi máy chủ.

### 3.6.5.9. API của sinh viên

#### a) Lấy danh sách sinh viên chứa tất cả thông tin cần thiết.

```
● ● ●

index: async (req, res) => {
  try {
    const { page, size } = req.query;

    const attributes = ['student_id', 'class_id', 'studentCode', 'email', 'name', 'createdAt', 'updatedAt', 'isDelete'];
    const whereClause = { isDelete: false };

    if (page && size) {
      const offset = (page - 1) * size;
      const limit = parseInt(size, 10);

      const { count, rows: students } = await StudentModel.findAndCountAll({
        include: [
          { model: ClassModel,
            attributes: ['classCode', 'classNameShort', 'className'],
            where: {
              isDelete: false
            }
          },
          attributes: attributes,
          where: whereClause,
          offset: offset,
          limit: limit
        ],
        attributes: attributes,
        where: whereClause,
        offset: offset,
        limit: limit
      });
      return res.json({
        total: count,
        students: students
      });
    } else {
      const students = await StudentModel.findAll({
        include: [
          { model: ClassModel,
            attributes: ['classCode']
          },
          attributes: attributes,
          where: whereClause
        ],
        attributes: attributes
      });

      return res.json(students);
    }
  } catch (error) {
    console.error('Lỗi khi lấy tất cả sinh viên:', error);
    res.status(500).json({ message: 'Lỗi server' });
  }
}
```

Hình 3.70. API lấy danh sách sinh viên.

Hàm index là một hàm không đồng bộ được thiết kế để lấy danh sách tất cả các sinh viên từ cơ sở dữ liệu, hỗ trợ phân trang nếu có yêu cầu. Dưới đây là giải thích chi tiết về

hoạt động của hàm:

Lấy thông tin phân trang từ yêu cầu: Hàm bắt đầu bằng việc lấy các tham số page và size từ truy vấn yêu cầu (req.query).

Định nghĩa các thuộc tính và điều kiện lọc:

- attributes: Chứa danh sách các thuộc tính của sinh viên cần lấy.
- whereClause: Điều kiện lọc để chỉ lấy các sinh viên chưa bị xóa (isDelete: false).

Kiểm tra phân trang:

Nếu có thông tin phân trang (page và size):

- Tính toán offset và limit để xác định phạm vi dữ liệu cần lấy.

Sử dụng StudentModel.findAndCountAll để lấy tổng số lượng sinh viên (count) và danh sách sinh viên (rows: students) với phân trang.

- Trả về phản hồi JSON chứa tổng số lượng sinh viên và danh sách sinh viên theo trang.

Nếu không có thông tin phân trang:

- Sử dụng StudentModel.findAll để lấy tất cả các sinh viên phù hợp với điều kiện lọc mà không áp dụng phân trang.
- Trả về phản hồi JSON chứa danh sách sinh viên.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi server.

## b) Lấy danh sách sinh viên bằng id lớp học



```
getAllById: async (req, res) => {
  try {
    const { id } = req.params;
    const students = await StudentModel.findAll({ where: { class_id: id, isDelete: false } });
    if (!students) {
      return res.status(404).json({ message: 'Không tìm thấy students' });
    }
    res.json(students);
  } catch (error) {
    console.error('Lỗi tìm kiếm students:', error);
    res.status(500).json({ message: 'Lỗi server' });
  }
}
```

Hình 3.71. API lấy danh sách sinh viên bởi id lớp học

Hàm getAllById là một hàm không đồng bộ được thiết kế để lấy danh sách tất cả các sinh viên thuộc một lớp học cụ thể dựa trên class\_id. Dưới đây là giải thích chi tiết về hoạt động của hàm:

Lấy class\_id từ yêu cầu: Hàm bắt đầu bằng việc lấy id từ các tham số của yêu cầu (req.params).

Tìm kiếm danh sách sinh viên: sử dụng StudentModel.findAll để tìm kiếm tất cả các sinh viên có class\_id trùng với id và chưa bị xóa (isDelete: false).

Kiểm tra kết quả: nếu không tìm thấy sinh viên nào, hàm sẽ trả về mã trạng thái 404 cùng với thông báo "Không tìm thấy students".

Nếu tìm thấy sinh viên, hàm sẽ trả về danh sách sinh viên dưới dạng JSON.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi server.

### c) Kết quả học tập của một sinh viên



```
learningOutcomes: async (req, res) => {
  try {
    const { id } = req.params;
    const results = await sequelize.query(
      `SELECT
        s.student_id,
        s.name AS studentName,
        sub.subject_id,
        sub.subjectName,
        c.course_id,
        c.courseName,
        a.totalScore AS score,
        ay.academic_year_id,
        ay.description AS academic_year,
        sem.semester_id,
        sem.descriptionShort AS semester,
        t.teacher_id,
        t.name AS teacherName,
        cl.class_id,
        cl.className,
        cl.classNameShort
      FROM
        students s
      JOIN
        course_enrollments ce ON s.student_id = ce.student_id
      JOIN
        courses c ON ce.course_id = c.course_id
      JOIN
        subjects sub ON c.subject_id = sub.subject_id
      JOIN
        semester_academic_years say ON c.id_semester_academic_year = say.id_semester_academic_year
      JOIN
        academic_years ay ON say.academic_year_id = ay.academic_year_id
      JOIN
        semesters sem ON say.semester_id = sem.semester_id
      JOIN
        assessments a ON c.course_id = a.course_id AND a.student_id = s.student_id
      JOIN
        teachers t ON c.teacher_id = t.teacher_id
      JOIN
        classes cl ON c.class_id = cl.class_id
      WHERE
        s.student_id = ${id}
        AND s.isDelete = 0
        AND c.isDelete = 0
        AND sub.isDelete = 0
        AND ce.isDelete = 0
        AND a.isDelete = 0
        AND ay.isDelete = 0
        AND sem.isDelete = 0
        AND t.isDelete = 0
        AND cl.isDelete = 0
      ORDER BY
        ay.academic_year_id, sem.semester_id, c.course_id;`,
      {
        type: Sequelize.QueryTypes.SELECT,
      }
    );
    res.json(results);
  } catch (error) {
    console.error('Error fetching average scores per subject:', error);
    res.status(500).json({ message: 'Internal Server Error' });
  }
}
```

Hình 3.72. API kết quả học tập của một sinh viên

Hàm learningOutcomes là một hàm không đồng bộ được thiết kế để lấy thông tin về kết quả học tập của một sinh viên cụ thể dựa trên ID của sinh viên đó. Dưới đây là giải thích chi tiết về hoạt động của hàm:

Lấy student\_id từ yêu cầu: Hàm bắt đầu bằng việc lấy id từ các tham số của yêu cầu (req.params).

Truy vấn cơ sở dữ liệu:

- Sử dụng sequelize.query để thực hiện một truy vấn SQL. Truy vấn này sẽ lấy thông tin về sinh viên, môn học, khóa học, điểm số, năm học, học kỳ, giáo viên và lớp học.
- Truy vấn bao gồm nhiều bảng được kết hợp (JOIN) để thu thập các thông tin cần thiết từ các bảng students, course\_enrollments, courses, subjects, semester\_academic\_years, academic\_years, semesters, assessments, teachers, và classes.
- Điều kiện WHERE được sử dụng để chỉ lấy các bản ghi mà sinh viên, khóa học, môn học, ghi danh khóa học, đánh giá, năm học, học kỳ, giáo viên và lớp học không bị xóa (isDelete = 0).
- Kết quả được sắp xếp theo academic\_year\_id, semester\_id và course\_id.

Trả về kết quả dưới dạng JSON: Kết quả của truy vấn được trả về dưới dạng JSON.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi server.

#### d) Tạo một sinh viên mới

```
● ● ●  
create: async (req, res) => {  
  try {  
    const { data } = req.body;  
    const newStudent = await StudentModel.create(data);  
    res.json(newStudent);  
  } catch (error) {  
    console.error('Lỗi khi tạo sinh viên mới:', error);  
    res.status(500).json({ message: 'Lỗi server' });  
  }  
}
```

Hình 3.73. API tạo sinh viên mới

Hàm create là một hàm không đồng bộ được thiết kế để tạo một sinh viên mới trong cơ sở dữ liệu dựa trên dữ liệu được cung cấp trong yêu cầu. Dưới đây là giải thích chi tiết về hoạt động của hàm:

Lấy dữ liệu từ yêu cầu: Hàm bắt đầu bằng việc lấy data từ phần thân của yêu cầu (req.body).

Tạo sinh viên mới: Sử dụng StudentModel.create(data) để tạo một bản ghi sinh viên mới trong cơ sở dữ liệu với các thông tin được cung cấp trong data.

Trả về kết quả dưới dạng JSON: Sau khi tạo thành công, hàm sẽ trả về đối tượng sinh viên mới dưới dạng JSON.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi server.

### e) Lấy danh sách sinh viên có thông tin lớp học

```
● ● ●

getAllWithClass: async (req, res) => {
  try {
    const students = await StudentModel.findAll({
      include: [
        {
          model: ClassModel,
          attributes: ['classCode', 'classNameShort'],
          where: { isDelete: false }
        },
        {
          attributes: ['student_id', 'class_id', 'studentCode', 'email', 'name', 'isDelete'],
          where: { isDelete: false }
        }
      ],
      res.json(students);
    } catch (error) {
      console.error('Error:', error);
      res.status(500).json({ message: 'Internal Server Error' });
    }
}
```

Hình 3.74. API lấy danh sách lớp sinh viên có thông tin

Hàm getAllWithClass là một hàm không đồng bộ được thiết kế để lấy danh sách tất cả các sinh viên cùng với thông tin về lớp học mà họ thuộc về. Dưới đây là giải thích chi tiết về hoạt động của hàm:

Tìm kiếm danh sách sinh viên:

- Sử dụng StudentModel.findAll để tìm kiếm tất cả các sinh viên chưa bị xóa (isDelete: false).
- Bao gồm thông tin về lớp học của mỗi sinh viên từ bảng ClassModel, với các thuộc tính classCode và classNameShort. Chỉ lấy các lớp học chưa bị xóa (isDelete: false).

Trả về kết quả dưới dạng JSON: Sau khi lấy được danh sách sinh viên cùng với

thông tin lớp học, hàm sẽ trả về kết quả dưới dạng JSON.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi server.

#### f) Lấy thông tin sinh viên bằng id sinh viên



```
getByID: async (req, res) => {
  try {
    const { id } = req.params;
    const students = await StudentModel.findAll({
      include: [
        {
          model: ClassModel,
          attributes: ['classCode', 'classNameShort', 'className'],
          include: [
            {
              model: TeacherModel,
              attributes: ['name', 'teacherCode', 'email']
            }
          ],
          attributes: ['student_id', 'class_id', 'studentCode', 'date_of_birth', 'email', 'name', 'isDelete'],
          where: {
            student_id: id, isDelete: false
          }
        };
      if (!students) {
        return res.status(404).json({ message: 'Không tìm thấy students' });
      }
      res.json(students);
    } catch (error) {
      console.error('Lỗi tìm kiếm students:', error);
      res.status(500).json({ message: 'Lỗi server' });
    }
  }
}
```

Hình 3.75. API lấy thông tin sinh viên bằng id sinh viên

Hàm getID là một hàm không đồng bộ được thiết kế để lấy thông tin chi tiết của một sinh viên dựa trên student\_id. Hàm này bao gồm thông tin về lớp học và giáo viên liên quan đến sinh viên đó. Dưới đây là giải thích chi tiết về hoạt động của hàm:

Lấy student\_id từ yêu cầu: Hàm bắt đầu bằng việc lấy id từ các tham số của yêu cầu (req.params).

Tìm kiếm thông tin sinh viên:

- Sử dụng StudentModel.findAll để tìm kiếm sinh viên có student\_id trùng với id và chưa bị xóa (isDelete: false).
- Bao gồm thông tin về lớp học của sinh viên từ bảng ClassModel, với các thuộc tính classCode, classNameShort, và className.
- Bao gồm thông tin về giáo viên của lớp học từ bảng TeacherModel, với các thuộc tính name, teacherCode, và email.

Kiểm tra kết quả:

- Nếu không tìm thấy sinh viên nào, hàm sẽ trả về mã trạng thái 404 cùng với thông báo "Không tìm thấy students".
- Nếu tìm thấy sinh viên, hàm sẽ trả về danh sách sinh viên dưới dạng JSON.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi server.

### g) Cập nhật thông tin một sinh viên

```
● ● ●

update: async (req, res) => {
  try {
    const { id } = req.params;
    const { data } = req.body;
    const studentDetails = await StudentModel.findOne({ where: { student_id: id } });
    if (!studentDetails) {
      return res.status(404).json({ message: 'Không tìm thấy sinh viên' });
    }
    await StudentModel.update(data, { where: { student_id: id } });
    res.json({ message: 'Cập nhật thành công thông tin sinh viên có ID: ${id}' });
  } catch (error) {
    console.error('Lỗi khi cập nhật sinh viên:', error);
    res.status(500).json({ message: 'Lỗi server' });
  }
}
```

Hình 3.76. API cập nhật thông tin của một sinh viên

Hàm update là một hàm không đồng bộ được thiết kế để cập nhật thông tin của một sinh viên dựa trên student\_id. Dưới đây là giải thích chi tiết về hoạt động của hàm:

Lấy student\_id từ yêu cầu: Hàm bắt đầu bằng việc lấy id từ các tham số của yêu cầu (req.params).

Lấy dữ liệu từ yêu cầu: Hàm lấy data từ phần thân của yêu cầu (req.body).

Kiểm tra sự tồn tại của sinh viên:

- Sử dụng StudentModel.findOne để tìm sinh viên có student\_id trùng với id.
- Nếu không tìm thấy sinh viên, hàm sẽ trả về mã trạng thái 404 cùng với thông báo "Không tìm thấy sinh viên".

Cập nhật thông tin sinh viên:

- Sử dụng StudentModel.update để cập nhật thông tin của sinh viên với data được cung cấp.

- Nếu cập nhật thành công, hàm sẽ trả về thông báo thành công cùng với student\_id.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi server.

#### **h) Xóa một sinh viên**



```
delete: async (req, res) => {
  try {
    const { id } = req.params;
    await StudentModel.destroy({ where: { student_id: id } });
    res.json({ message: 'Xóa sinh viên thành công' });
  } catch (error) {
    console.error('Lỗi khi xóa sinh viên:', error);
    res.status(500).json({ message: 'Lỗi server' });
  }
}
```

*Hình 3.77. Xóa thông tin một sinh viên*

Hàm delete là một hàm không đồng bộ được thiết kế để xóa một sinh viên dựa trên student\_id. Dưới đây là giải thích chi tiết về hoạt động của hàm:

Lấy student\_id từ yêu cầu: Hàm bắt đầu bằng việc lấy id từ các tham số của yêu cầu (req.params).

Xóa sinh viên: sử dụng StudentModel.destroy để xóa sinh viên có student\_id trùng với id.

Trả về thông báo thành công: Sau khi xóa thành công, hàm sẽ trả về thông báo JSON xác nhận rằng sinh viên đã được xóa thành công.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi server.

### i) Lấy danh sách sinh viên bị ẩn



```
isDeleteToTrue: async (req, res) => {
  try {
    const students = await StudentModel.findAll(
      {
        include: [
          {
            model: ClassModel,
            attributes: ['classCode']
          }],
        where: { isDelete: true }
      });
    if (!students) {
      return res.status(404).json({ message: 'Không tìm thấy students' });
    }
    res.json(students);
  } catch (error) {
    console.error('Lỗi tìm kiếm students:', error);
    res.status(500).json({ message: 'Lỗi server' });
  }
}
```

Hình 3.78. API lấy danh sách sinh viên bị ẩn

Hàm isDeleteToTrue là một hàm không đồng bộ được thiết kế để lấy danh sách tất cả các sinh viên đã bị đánh dấu xóa (isDelete: true) và bao gồm thông tin về lớp học mà họ thuộc về. Dưới đây là giải thích chi tiết về hoạt động của hàm:

Tìm kiếm danh sách sinh viên đã bị xóa:

- Sử dụng StudentModel.findAll để tìm kiếm tất cả các sinh viên có isDelete: true.
- Bao gồm thông tin về lớp học của mỗi sinh viên từ bảng ClassModel, với thuộc tính classCode.

Kiểm tra kết quả:

- Nếu không tìm thấy sinh viên nào, hàm sẽ trả về mã trạng thái 404 cùng với thông báo "Không tìm thấy students".
- Nếu tìm thấy sinh viên, hàm sẽ trả về danh sách sinh viên dưới dạng JSON.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi server.

## j) Lấy danh sách sinh viên



```
isDeleteToFalse: async (req, res) => {
  try {
    const students = await StudentModel.findAll({ where: { isDelete: false } });
    if (!students) {
      return res.status(404).json({ message: 'Không tìm thấy students' });
    }
    res.json(students);
  } catch (error) {
    console.error('Lỗi tìm kiếm students:', error);
    res.status(500).json({ message: 'Lỗi server' });
  }
}
```

Hình 3.79. API lấy danh sách sinh viên

Hàm isDeleteToFalse là một hàm không đồng bộ được thiết kế để lấy danh sách tất cả các sinh viên chưa bị đánh dấu xóa (isDelete: false). Dưới đây là giải thích chi tiết về hoạt động của hàm:

Tìm kiếm danh sách sinh viên chưa bị xóa: sử dụng StudentModel.findAll để tìm kiếm tất cả các sinh viên có isDelete: false.

Kiểm tra kết quả:

- Nếu không tìm thấy sinh viên nào, hàm sẽ trả về mã trạng thái 404 cùng với thông báo "Không tìm thấy students".
- Nếu tìm thấy sinh viên, hàm sẽ trả về danh sách sinh viên dưới dạng JSON.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi server.

## k) Đảo ngược trạng thái sinh viên



```
isDelete: async (req, res) => {
  try {
    const { id } = req.params;
    const student = await StudentModel.findOne({ where: { student_id: id } });
    if (!student) {
      return res.status(404).json({ message: 'Không tìm thấy students' });
    }
    const updatedIsDeleted = !student.isDelete;
    await StudentModel.update({ isDelete: updatedIsDeleted }, { where: { student_id: id } });
    res.json({ message: `Đã đảo ngược trạng thái isDelete thành ${updatedIsDeleted}` });
  } catch (error) {
    console.error('Lỗi khi đảo ngược trạng thái isDelete của students:', error);
    res.status(500).json({ message: 'Lỗi máy chủ' });
  }
}
```

Hình 3.80. API đảo ngược trạng thái của sinh viên

Hàm isDelete là một hàm không đồng bộ được thiết kế để đảo ngược trạng thái isDelete của một sinh viên dựa trên student\_id. Dưới đây là giải thích chi tiết về hoạt động của hàm:

Lấy student\_id từ yêu cầu: Hàm bắt đầu bằng việc lấy id từ các tham số của yêu cầu (req.params).

Tìm kiếm thông tin sinh viên:

- Sử dụng StudentModel.findOne để tìm sinh viên có student\_id trùng với id.
- Nếu không tìm thấy sinh viên, hàm sẽ trả về mã trạng thái 404 cùng với thông báo "Không tìm thấy students".

Đảo ngược trạng thái isDelete:

- Đảo ngược giá trị của thuộc tính isDelete hiện tại của sinh viên (!students.isDelete).
- Sử dụng StudentModel.update để cập nhật giá trị mới của isDelete cho sinh viên có student\_id trùng với id.

Trả về thông báo thành công: Sau khi cập nhật thành công, hàm sẽ trả về thông báo JSON xác nhận rằng trạng thái isDelete đã được đảo ngược thành công với giá trị mới.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi máy chủ.

## I) Xuất file excel mẫu điền thông tin sinh viên

```
getFormStudent: async (req, res) => {
  const workbook = new ExcelJS.Workbook();
  const worksheet = workbook.addWorksheet('Students Form');

  worksheet.columns = [
    { header: 'Mã lớp', key: 'classCode', width: 15 },
    { header: 'Tên SV', key: 'name', width: 32 },
    { header: 'MSSV', key: 'studentCode', width: 20 },
    { header: 'Email', key: 'email', width: 30 },
  ];

  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();
}
```

Hình 3.81. API xuất file excel mẫu điền thông tin sinh viên

Hàm getFormStudent là một hàm không đồng bộ được thiết kế để tạo và trả về một

file Excel chứa biểu mẫu thông tin sinh viên. Dưới đây là giải thích chi tiết về hoạt động của hàm:

Tạo workbook và worksheet:

- Sử dụng ExcelJS.Workbook() để tạo một workbook mới.
- Tạo một worksheet mới có tên "Students Form" trong workbook.

Định nghĩa các cột cho worksheet:

- Thiết lập các cột trong worksheet với các thuộc tính: classCode, name, studentCode, và email.
- Đặt tiêu đề cho mỗi cột và thiết lập độ rộng cho chúng.

Thiết lập header cho phản hồi:

- Thiết lập Content-Type trong header của phản hồi để chỉ định rằng nội dung phản hồi là một tệp Excel.
- Thiết lập Content-Disposition trong header của phản hồi để chỉ định tên tệp là "StudentsForm.xlsx".

Ghi workbook vào phản hồi và kết thúc:

- Sử dụng workbook.xlsx.write(res) để ghi nội dung của workbook vào phản hồi.
- Gọi res.end() để kết thúc phản hồi.

### m) Xuất file excel với thông tin danh sách sinh viên

```
getFormStudentWithData: async (req, res) => {
  try {
    const { data } = req.body;

    if (!data || !Array.isArray(data.id) || data.id.length === 0) {
      return res.status(400).json({ error: 'Invalid or missing id array' });
    }
    const { id } = data;
    const workbook = new ExcelJS.Workbook();
    const worksheet = workbook.addWorksheet('Students Form');

    const students = await StudentModel.findAll({
      include: [
        { model: ClassModel,
          attributes: ['classCode']
        },
        attributes: ['student_id', 'class_id', 'studentCode', 'email', 'name', 'isDelete'],
        where: {
          isDelete: false,
          student_id: id
        }
      });
    worksheet.columns = [
      { header: 'id', key: 'id', width: 15 },
      { header: 'Mã lớp', key: 'classCode', width: 15 },
      { header: 'Tên SV', key: 'name', width: 32 },
      { header: 'MSSV', key: 'studentCode', width: 20 },
      { header: 'Email', key: 'email', width: 30 }
    ];
    students.forEach(student => {
      worksheet.addRow({
        id: student.student_id,
        classCode: student.class.classCode,
        name: student.name,
        studentCode: student.studentCode,
        email: student.email
      });
    });

    await worksheet.protect('yourpassword', {
      selectLockedCells: true,
      selectUnlockedCells: true
    });

    worksheet.eachRow((row, rowNum) => {
      row.eachCell((cell, colNumber) => {
        if (colNumber === 1) {
          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' });
  }
}
```

Hình 3.82. API xuất file excel với thông tin sinh viên

Hàm getFormStudentWithData là một hàm không đồng bộ được thiết kế để tạo và

trả về một file Excel chứa thông tin về các sinh viên dựa trên danh sách ID được cung cấp trong yêu cầu. Dưới đây là giải thích chi tiết về hoạt động của hàm:

Kiểm tra dữ liệu yêu cầu:

- Hàm bắt đầu bằng việc lấy data từ phần thân của yêu cầu (req.body).
- Kiểm tra tính hợp lệ của data để đảm bảo rằng id là một mảng và không rỗng. Nếu không hợp lệ, trả về mã trạng thái 400 với thông báo lỗi.

Tạo workbook và worksheet:

- Sử dụng ExcelJS.Workbook() để tạo một workbook mới.
- Tạo một worksheet mới có tên "Students Form" trong workbook.

Tìm kiếm thông tin sinh viên:

- Sử dụng StudentModel.findAll để tìm kiếm tất cả các sinh viên có student\_id nằm trong mảng id và chưa bị xóa (isDelete: false).
- Bao gồm thông tin về lớp học của mỗi sinh viên từ bảng ClassModel, với thuộc tính classCode.

Định nghĩa các cột cho worksheet:

- Thiết lập các cột trong worksheet với các thuộc tính: id, classCode, name, studentCode, và email.
- Đặt tiêu đề cho mỗi cột và thiết lập độ rộng cho chúng.

Thêm thông tin sinh viên vào worksheet: duyệt qua danh sách sinh viên và thêm từng sinh viên vào worksheet với các thông tin tương ứng.

Bảo vệ worksheet:

- Bảo vệ worksheet bằng mật khẩu.
- Đặt thuộc tính bảo vệ cho các ô để cột id được khóa (không chỉnh sửa được), còn các cột khác thì không khóa (có thể chỉnh sửa được).

Thiết lập header cho phản hồi:

- Thiết lập Content-Type trong header của phản hồi để chỉ định rằng nội dung

phản hồi là một tệp Excel.

- Thiết lập Content-Disposition trong header của phản hồi để chỉ định tên tệp là "StudentsForm.xlsx".

Ghi workbook vào phản hồi và kết thúc:

- Sử dụng workbook.xlsx.write(res) để ghi nội dung của workbook vào phản hồi.
- Gọi res.end() để kết thúc phản hồi.

## n) Xuất danh sách sinh viên với id lớp học

```
● ● ●

getFormStudentByClass: async (req, res) => {
  try {
    const { id } = req.params;
    const workbook = new ExcelJS.Workbook();
    const worksheet = workbook.addWorksheet('Students Form');
    const students = await StudentModel.findAll({
      include: [
        {
          model: ClassModel,
          attributes: ['classCode', 'classNameShort', 'className'],
          where: {
            class_id: id,
          }
        },
        {
          attributes: ['student_id', 'class_id', 'studentCode', 'email', 'name', 'isDelete'],
          where: {
            isDelete: false,
          }
        }
      ];
    });

    worksheet.columns = [
      { header: 'Mã lớp', key: 'classNameShort', width: 15 },
      { header: 'Tên SV', key: 'name', width: 32 },
      { header: 'MSSV', key: 'studentCode', width: 20 },
      { header: 'Email', key: 'email', width: 30 }
    ];

    students.forEach(student => {
      worksheet.addRow({
        classNameShort: student.class.classNameShort,
        name: student.name,
        studentCode: student.studentCode,
        email: student.email
      });
    });

    res.setHeader('Content-Type', 'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet');
    res.setHeader('Content-Disposition', 'attachment; filename="Students.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' });
  }
}
```

Hình 3.83. API xuất excel danh sách sinh viên bởi id lớp học

Hàm getFormStudentByClass là một hàm không đồng bộ được thiết kế để tạo và trả về một file Excel chứa thông tin về các sinh viên thuộc một lớp cụ thể dựa trên class\_id. Dưới đây là giải thích chi tiết về hoạt động của hàm:

Lấy class\_id từ yêu cầu: Hàm bắt đầu bằng việc lấy id từ các tham số của yêu cầu (req.params).

Tạo workbook và worksheet:

- Sử dụng ExcelJS.Workbook() để tạo một workbook mới.
- Tạo một worksheet mới có tên "Students Form" trong workbook.

Tìm kiếm thông tin sinh viên:

- Sử dụng StudentModel.findAll để tìm kiếm tất cả các sinh viên có class\_id trùng với id và chưa bị xóa (isDelete: false).
- Bao gồm thông tin về lớp học của mỗi sinh viên từ bảng ClassModel, với các thuộc tính classCode, classNameShort, và className.

Định nghĩa các cột cho worksheet:

- Thiết lập các cột trong worksheet với các thuộc tính: classNameShort, name, studentCode, và email.
- Đặt tiêu đề cho mỗi cột và thiết lập độ rộng cho chúng.

Thêm thông tin sinh viên vào worksheet: duyệt qua danh sách sinh viên và thêm từng sinh viên vào worksheet với các thông tin tương ứng.

Thiết lập header cho phản hồi:

- Thiết lập Content-Type trong header của phản hồi để chỉ định rằng nội dung phản hồi là một tệp Excel.
- Thiết lập Content-Disposition trong header của phản hồi để chỉ định tên tệp là "Students.xlsx".

Ghi workbook vào phản hồi và kết thúc:

- Sử dụng workbook.xlsx.write(res) để ghi nội dung của workbook vào phản hồi.
- Gọi res.end() để kết thúc phản hồi.

## o) Tạo sinh viên bằng file excel

```
● ○ ●

saveStudentExcel: async (req, res) => {
  if (req.files) {
    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(1);
    const insertPromises = [];
    const emailsSet = new Set();
    const studentCodesSet = new Set();

    worksheet.eachRow({ includeEmpty: false }, (row, rowNum) => {
      if (rowNum !== 1) { // bỏ dòng đầu
        let email = row.getCell(4).value;
        let studentCode = row.getCell(3).value;

        if (email && typeof email === 'object' && email.hasOwnProperty('text')) {
          email = email.text;
        }

        if (!email || emailsSet.has(email)) {
          console.error(`Duplicate or invalid email found at row ${rowNum}: ${email}`);
          return; // Skip this row
        }

        if (!studentCode || studentCodesSet.has(studentCode)) {
          console.error(`Duplicate or invalid studentCode found at row ${rowNum}: ${studentCode}`);
          return; // Skip this row
        }

        emailsSet.add(email);
        studentCodesSet.add(studentCode);

        const sql = `INSERT INTO students (class_id, name, studentCode, email)
                    VALUES ((SELECT class_id FROM classes WHERE classNameShort = ?), ?, ?, ?)`;
        const values = [
          row.getCell(1).value,
          row.getCell(2).value,
          studentCode,
          email,
        ];
        insertPromises.push(sequelize.query(sql, { replacements: values })
          .catch(error => {
            console.error(`Error inserting row ${rowNum}: ${error.message}`);
          }));
      }
    });

    Promise.all(insertPromises)
      .then(() => {
        console.log('All rows inserted successfully');
      })
      .catch(error => {
        console.error('Error inserting rows:', error);
      });

    await Promise.all(insertPromises);
    console.log('All data has been inserted into the database!');
    fs.unlinkSync(filePath);
    res.status(200).json({ message: "Dữ liệu đã được lưu thành công vào cơ sở dữ liệu." });
  } else {
    res.status(400).send('No file uploaded.');
  }
}
```

Hình 3.84. API tạo sinh viên bằng file excel

Hàm saveStudentExcel là một hàm không đồng bộ được thiết kế để xử lý file Excel

tải lên, đọc dữ liệu từ file, và chèn dữ liệu sinh viên vào cơ sở dữ liệu. Dưới đây là giải thích chi tiết về hoạt động của hàm:

Kiểm tra file tải lên: hàm bắt đầu bằng việc kiểm tra xem file có được tải lên hay không (req.files).

Thiết lập đường dẫn và đọc file Excel:

- Xác định đường dẫn tới thư mục tải lên và đường dẫn tới file Excel.
- Tạo một workbook mới bằng cách sử dụng ExcelJS.Workbook() và đọc nội dung file Excel từ đường dẫn.

Khởi tạo các tập hợp để kiểm tra email và mã sinh viên trùng lặp: khởi tạo các tập hợp (Set) để lưu trữ các email và mã sinh viên đã được xử lý để tránh trùng lặp.

Duyệt qua các hàng trong worksheet:

- Sử dụng worksheet.eachRow để duyệt qua từng hàng trong worksheet.
- Bỏ qua hàng đầu tiên vì nó chứa tiêu đề.

Kiểm tra và xử lý email và mã sinh viên trùng lặp:

- Lấy giá trị của email và mã sinh viên từ các cột tương ứng.
- Kiểm tra và loại bỏ các email và mã sinh viên trùng lặp hoặc không hợp lệ.
- Thêm các email và mã sinh viên hợp lệ vào các tập hợp tương ứng.

Chuẩn bị và thực thi các câu lệnh SQL để chèn dữ liệu:

- Tạo câu lệnh SQL để chèn dữ liệu vào bảng students.
- Sử dụng sequelize.query để thực thi câu lệnh SQL với các giá trị thay thế.
- Sử dụng Promise.all để thực thi tất cả các câu lệnh SQL song song.

Xử lý hoàn tất:

- Xóa file Excel sau khi xử lý xong (fs.unlinkSync).
- Trả về phản hồi xác nhận rằng dữ liệu đã được lưu thành công vào cơ sở dữ liệu.

## p) Cập nhật danh sách sinh viên bằng file excel

```
updateStudentsFromExcel: 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('Students Form');
    const studentUpdates = [];

    worksheet.eachRow((row, rowNum) => {
      if (rowNum > 1) { // Skip header row
        const studentData = {
          student_id: row.getCell('A').value,
          classCode: row.getCell('B').value,
          name: row.getCell('C').value,
          studentCode: row.getCell('D').value,
          email: row.getCell('E').value
        };
        studentUpdates.push(studentData);
      }
    });
    for (const studentData of studentUpdates) {
      const student = await StudentModel.findById(studentData.student_id);
      if (student) {
        const classModel = await ClassModel.findOne({ where: { classCode: studentData.classCode } });
        if (classModel) {
          student.class_id = classModel.class_id;
        }
        student.name = studentData.name;
        student.studentCode = studentData.studentCode;
        student.email = studentData.email;
        await student.save();
      }
    }
    fs.unlinkSync(filePath); // Clean up the uploaded file
    res.status(200).json({ message: 'Students updated successfully' });
  } catch (error) {
    console.error('Error updating students from Excel file:', error);
    res.status(500).json({ error: 'Internal server error' });
  }
}
```

Hình 3.85. Cập nhật danh sách sinh viên bằng file excel

Hàm updateStudentsFromExcel là một hàm không đồng bộ được thiết kế để cập nhật thông tin sinh viên từ một file Excel. Dưới đây là giải thích chi tiết về hoạt động của hàm:

Kiểm tra file tải lên:

- Hàm bắt đầu bằng việc kiểm tra xem file có được tải lên hay không (req.files).
- Nếu không có file nào được tải lên, hàm sẽ trả về mã trạng thái 400 với thông báo "No file uploaded".

Xử lý và đọc file Excel:

- Xác định đường dẫn tới thư mục tải lên và đường dẫn tới file Excel.
- Tạo một workbook mới bằng cách sử dụng ExcelJS.Workbook() và đọc nội dung file Excel từ đường dẫn.
- Lấy worksheet có tên "Students Form".

Duyệt qua các hàng trong worksheet và thu thập dữ liệu:

- Sử dụng worksheet.eachRow để duyệt qua từng hàng trong worksheet.
- Bỏ qua hàng đầu tiên vì nó chứa tiêu đề.
- Thu thập dữ liệu từ các cột tương ứng và lưu trữ vào mảng studentUpdates.

Cập nhật thông tin sinh viên trong cơ sở dữ liệu:

- Duyệt qua từng đối tượng sinh viên trong mảng studentUpdates.
- Tìm sinh viên trong cơ sở dữ liệu bằng StudentModel.findByPk.
- Nếu sinh viên tồn tại, tìm lớp học tương ứng bằng ClassModel.findOne với classCode.
- Cập nhật thông tin sinh viên với các dữ liệu từ file Excel và lưu thay đổi vào cơ sở dữ liệu.

Xóa file Excel sau khi xử lý xong: sử dụng fs.unlinkSync để xóa file Excel sau khi đã xử lý xong.

Trả về thông báo thành công hoặc lỗi:

- Nếu không có lỗi xảy ra, trả về mã trạng thái 200 với thông báo "Students updated successfully".
- Nếu có lỗi xảy ra, ghi lại thông báo lỗi và trả về mã trạng thái 500 với thông báo lỗi máy chủ.

### 3.6.5.10. API của giáo viên

#### a) Lấy danh sách sinh viên có phân trang



```
index: async (req, res) => {
  try {
    const { page, size } = req.query;
    const attributes = ['teacher_id', 'name', 'teacherCode', 'email', 'permission', 'typeTeacher', 'imgURL'];
    const whereClause = {
      isDelete: false,
      isBlock: false
    };
    if (page && size) {
      const offset = (page - 1) * size;
      const limit = parseInt(size, 10);
      const { count, rows: teachers } = await TeacherModel.findAndCountAll({
        attributes: attributes,
        where: whereClause,
        offset: offset,
        limit: limit
      });
      const teachersWithPermissionName = teachers.map(teacher => ({
        ...teacher.dataValues,
        permissionName: getPermissionName(teacher.permission)
      }));
      return res.json({
        total: count,
        teachers: teachersWithPermissionName
      });
    } else {
      const teachers = await TeacherModel.findAll({
        attributes: attributes,
        where: whereClause
      });
      const teachersWithPermissionName = teachers.map(teacher => ({
        ...teacher.dataValues,
        permissionName: getPermissionName(teacher.permission)
      }));
      return res.json(teachersWithPermissionName);
    }
  } catch (error) {
    console.error('Lỗi khi lấy tất cả các giáo viên:', error);
    res.status(500).json({ message: 'Lỗi server' });
  }
}
```

Hình 3.86. API lấy danh sách sinh viên có phân trang

Hàm index là một hàm không đồng bộ được thiết kế để lấy danh sách tất cả các giáo viên từ cơ sở dữ liệu, hỗ trợ phân trang nếu có yêu cầu. Hàm này cũng bổ sung thêm tên quyền (permission name) dựa trên giá trị quyền của giáo viên. Dưới đây là giải thích chi tiết về hoạt động của hàm:

Lấy thông tin phân trang từ yêu cầu: hàm bắt đầu bằng việc lấy các tham số page và size từ truy vấn yêu cầu (req.query).

Định nghĩa các thuộc tính và điều kiện lọc:

- attributes: Chứa danh sách các thuộc tính của giáo viên cần lấy.

- whereClause: Điều kiện lọc để chỉ lấy các giáo viên chưa bị xóa (isDelete: false) và không bị chặn (isBlock: false).

Nếu có thông tin phân trang (page và size):

- Tính toán offset và limit để xác định phạm vi dữ liệu cần lấy.
- Sử dụng TeacherModel.findAndCountAll để lấy tổng số lượng giáo viên (count) và danh sách giáo viên (rows: teachers) với phân trang.
- Bổ sung thêm tên quyền (permission name) cho từng giáo viên bằng cách sử dụng hàm getPermissionName.
- Trả về phản hồi JSON chứa tổng số lượng giáo viên và danh sách giáo viên theo trang.

Nếu không có thông tin phân trang:

- Sử dụng TeacherModel.findAll để lấy tất cả các giáo viên phù hợp với điều kiện lọc mà không áp dụng phân trang.
- Bổ sung thêm tên quyền cho từng giáo viên.
- Trả về phản hồi JSON chứa danh sách giáo viên.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi server.

## b) Lấy danh sách giáo viên bị chặn

```
getAllStore: async (req, res) => {
  try {
    const { page, size } = req.query;

    const attributes = ['teacher_id', 'name', 'teacherCode', 'email', 'permission', 'typeTeacher', 'imgURL'];
    const whereClause = {
      isBlock: true,
      isDelete: false,
    };
    if (page && size) {
      const offset = (page - 1) * size;
      const limit = parseInt(size, 10);
      const { count, rows: teachers } = await TeacherModel.findAndCountAll({
        attributes: attributes,
        where: whereClause,
        offset: offset,
        limit: limit
      });
      const teachersWithPermissionName = teachers.map(teacher => ({
        ...teacher.dataValues,
        permissionName: getPermissionName(teacher.permission)
      }));
      return res.json({
        total: count,
        teachers: teachersWithPermissionName
      });
    } else {
      const teachers = await TeacherModel.findAll({
        attributes: attributes,
        where: whereClause
      });
      const teachersWithPermissionName = teachers.map(teacher => ({
        ...teacher.dataValues,
        permissionName: getPermissionName(teacher.permission)
      }));
      return res.json(teachersWithPermissionName);
    }
  } catch (error) {
    console.error('Lỗi khi lấy tất cả các giáo viên:', error);
    res.status(500).json({ message: 'Lỗi server' });
  }
}
```

Hình 3.87. API lấy danh sách sinh viên bị chặn

Hàm getAllStore là một hàm không đồng bộ được thiết kế để lấy danh sách tất cả các giáo viên bị chặn (blocked) từ cơ sở dữ liệu, hỗ trợ phân trang nếu có yêu cầu. Hàm này cũng bổ sung thêm tên quyền (permission name) dựa trên giá trị quyền của giáo viên. Dưới đây là giải thích chi tiết về hoạt động của hàm:

Lấy thông tin phân trang từ yêu cầu: hàm bắt đầu bằng việc lấy các tham số page và size từ truy vấn yêu cầu (req.query).

Định nghĩa các thuộc tính và điều kiện lọc:

- attributes: Chứa danh sách các thuộc tính của giáo viên cần lấy.
- whereClause: Điều kiện lọc để chỉ lấy các giáo viên bị chặn (isBlock: true)

và chưa bị xóa (isDelete: false).

Nếu có thông tin phân trang (page và size):

- Tính toán offset và limit để xác định phạm vi dữ liệu cần lấy.
- Sử dụng TeacherModel.findAndCountAll để lấy tổng số lượng giáo viên (count) và danh sách giáo viên (rows: teachers) với phân trang.
- Bổ sung thêm tên quyền (permission name) cho từng giáo viên bằng cách sử dụng hàm getPermissionName.
- Trả về phản hồi JSON chứa tổng số lượng giáo viên và danh sách giáo viên theo trang.

Nếu không có thông tin phân trang:

- Sử dụng TeacherModel.findAll để lấy tất cả các giáo viên phù hợp với điều kiện lọc mà không áp dụng phân trang.
- Bổ sung thêm tên quyền cho từng giáo viên.
- Trả về phản hồi JSON chứa danh sách giáo viên.

Xử lý lỗi: Nếu có bất kỳ lỗi nào xảy ra trong quá trình này, chúng sẽ bị bắt bởi khối catch. Hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 cùng với thông báo lỗi server.

c) Tạo một giáo viên mới

```
create: async (req, res) => {
  try {
    const { email, password, name, typeTeacher, teacherCode } = req.body;
    const existingEmail = await TeacherModel.findOne({ where: { email: email } });
    if (existingEmail) {
      return res.status(400).json({ message: 'Email đã tồn tại' });
    }

    let uniqueTeacherCode = teacherCode;

    if (!teacherCode) {
      uniqueTeacherCode = await generateUniqueTeacherCode();
    } else {
      const existingCode = await TeacherModel.findOne({ where: { teacherCode: teacherCode } });
      if (existingCode) {
        return res.status(400).json({ message: 'Teacher code already exists' });
      }
    }

    const data = {
      email,
      password,
      name,
      teacherCode: uniqueTeacherCode,
      typeTeacher
    };

    const newTeacher = await TeacherModel.create(data);
    res.json({
      message: "Tạo giáo viên thành công",
      teacher_id: newTeacher.teacher_id,
      email: newTeacher.email,
      name: newTeacher.name,
      permission: newTeacher.permission,
      typeTeacher: newTeacher.typeTeacher,
    });
  } catch (error) {
    console.error('Lỗi khi tạo giáo viên mới:', error);
    res.status(500).json({ message: 'Lỗi server' });
  }
}
```

Hình 3.88. API tạo mới một giáo viên

Hàm create là một hàm không đồng bộ được thiết kế để tạo một giáo viên mới trong cơ sở dữ liệu. Hàm này kiểm tra tính duy nhất của email và mã giáo viên (teacherCode) trước khi tạo giáo viên mới. Dưới đây là giải thích chi tiết về hoạt động của hàm:

Lấy dữ liệu từ yêu cầu: đảm bảo đầu bằng việc lấy các thuộc tính email, password, name, typeTeacher, và teacherCode từ phần thân của yêu cầu (req.body).

Kiểm tra tính duy nhất của email:

- Sử dụng TeacherModel.findOne để kiểm tra xem email đã tồn tại trong cơ sở dữ liệu hay chưa.
  - Nếu email đã tồn tại, trả về mã trang thái 400 với thông báo "Email đã tồn tại".

Kiểm tra và tạo mã giáo viên duy nhất:

- Nếu không có mã giáo viên (teacherCode) được cung cấp trong yêu cầu, tạo một mã giáo viên duy nhất bằng cách sử dụng hàm generateUniqueTeacherCode.
- Nếu mã giáo viên được cung cấp, kiểm tra xem mã này đã tồn tại trong cơ sở dữ liệu hay chưa.
- Nếu mã giáo viên đã tồn tại, trả về mã trạng thái 400 với thông báo "Teacher code already exists".

Tạo giáo viên mới:

- Tạo đối tượng dữ liệu giáo viên mới với các thuộc tính đã lấy từ yêu cầu và mã giáo viên duy nhất.
- Sử dụng TeacherModel.create để tạo giáo viên mới trong cơ sở dữ liệu.

Trả về phản hồi thành công: trả về phản hồi JSON chứa thông tin giáo viên mới được tạo, bao gồm teacher\_id, email, name, permission, và typeTeacher.

Xử lý lỗi: nếu có bất kỳ lỗi nào xảy ra trong quá trình này, hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 với thông báo lỗi server.

#### d) Lấy danh sách giáo viên bởi ID giáo viên

```
getByID: async (req, res) => {
  try {
    const { id } = req.params;
    const attributes = ['teacher_id', 'name', 'teacherCode', 'email', 'permission', 'typeTeacher', 'imgURL'];
    const whereClause = {
      isDelete: false,
      isBlock: false,
      teacher_id: id
    };
    const teachers = await TeacherModel.findOne({
      attributes: attributes,
      where: whereClause,
    });

    return res.json({
      teachers,
      permissionName: getPermissionName(teachers.permission)
    });
  } catch (error) {
    console.error('Lỗi khi tìm kiếm giáo viên:', error);
    res.status(500).json({ message: 'Lỗi server' });
  }
}
```

Hình 3.89. API lấy thông tin giáo viên bởi id giáo viên

Hàm getByID là một hàm không đồng bộ được thiết kế để lấy thông tin chi tiết của

một giáo viên dựa trên teacher\_id. Hàm này kiểm tra xem giáo viên có tồn tại và không bị xóa hoặc chặn trước khi trả về thông tin của giáo viên đó cùng với tên quyền (permission name). Dưới đây là giải thích chi tiết về hoạt động của hàm:

Lấy teacher\_id từ yêu cầu: hàm bắt đầu bằng việc lấy id từ các tham số của yêu cầu (req.params).

Định nghĩa các thuộc tính và điều kiện lọc:

- attributes: Chứa danh sách các thuộc tính của giáo viên cần lấy.
- whereClause: Điều kiện lọc để chỉ lấy giáo viên có teacher\_id trùng với id, chưa bị xóa (isDelete: false), và không bị chặn (isBlock: false).

Tìm kiếm thông tin giáo viên: sử dụng TeacherModel.findOne để tìm kiếm giáo viên phù hợp với điều kiện lọc.

Trả về thông tin giáo viên:

- Nếu tìm thấy giáo viên, trả về phản hồi JSON chứa thông tin giáo viên và tên quyền (permission name).
- Nếu không tìm thấy giáo viên, trả về phản hồi với thông báo lỗi.

Xử lý lỗi: nếu có bất kỳ lỗi nào xảy ra trong quá trình này, hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 với thông báo lỗi server.

### e) Cập nhật giáo viên bởi ID giáo viên

```
update: async (req, res) => {
  try {
    const { id } = req.params;
    const { data } = req.body;

    console.log("vcvc", data)
    const teacherDetail = await TeacherModel.findOne({ where: { teacher_id: id } });
    if (!teacherDetail) {
      return res.status(404).json({ message: 'Không tìm thấy giáo viên' });
    }

    await teacherDetail.update(data);

    res.json({ message: `Cập nhật thành công thông tin giáo viên thành công` });
  } catch (error) {
    console.error('Lỗi khi cập nhật giáo viên:', error);
    res.status(500).json({ message: 'Lỗi server' });
  }
}
```

Hình 3.90. API cập nhật giáo viên bởi id giáo viên

Hàm update là một hàm không đồng bộ được thiết kế để cập nhật thông tin của một giáo viên dựa trên teacher\_id. Dưới đây là giải thích chi tiết về hoạt động của hàm:

Lấy teacher\_id từ yêu cầu:

- Hàm bắt đầu bằng việc lấy id từ các tham số của yêu cầu (req.params).
- Lấy dữ liệu cập nhật từ phần thân của yêu cầu (req.body.data).

Tìm kiếm giáo viên theo ID:

- Sử dụng TeacherModel.findOne để tìm giáo viên có teacher\_id trùng với id.
- Nếu không tìm thấy giáo viên, trả về mã trạng thái 404 với thông báo "Không tìm thấy giáo viên".

Cập nhật thông tin giáo viên: sử dụng phương thức update của đối tượng teacherDetail để cập nhật thông tin của giáo viên với dữ liệu mới.

Trả về phản hồi thành công: nếu cập nhật thành công, trả về phản hồi JSON với thông báo "Cập nhật thành công thông tin giáo viên thành công".

Xử lý lỗi: nếu có bất kỳ lỗi nào xảy ra trong quá trình này, hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 với thông báo lỗi server.

## f) Chặn một giáo viên

```
blockTeacher: async (req, res) => {
  try {
    const teacherId = req.params.id;
    await TeacherModel.update({ isBlock: 1 }, { where: { teacher_id: teacherId } });
    res.status(200).json({ message: 'Teacher has been blocked.' });
  } catch (error) {
    console.error('Error blocking teacher:', error);
    res.status(500).json({ message: 'Server error' });
  }
}
```

Hình 3.91. API chặn một giáo viên.

Hàm blockTeacher là một hàm không đồng bộ được thiết kế để chặn một giáo viên dựa trên teacher\_id. Dưới đây là giải thích chi tiết về hoạt động của hàm:

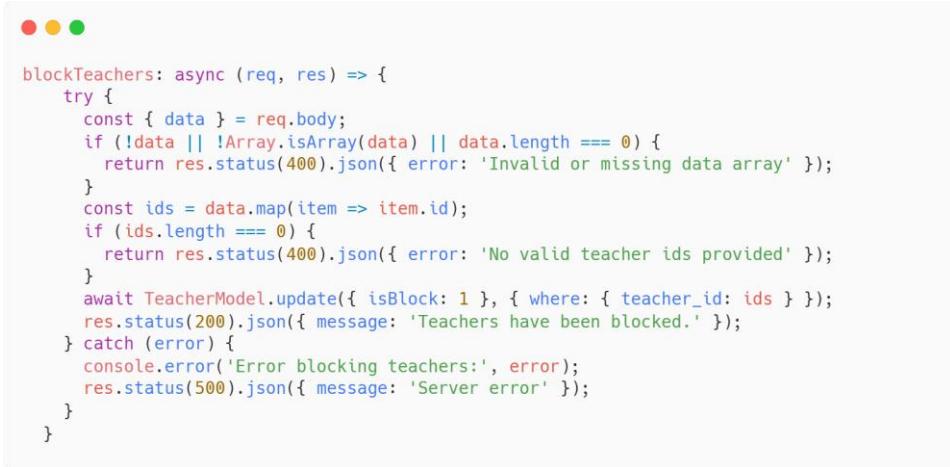
Lấy teacher\_id từ yêu cầu: hàm bắt đầu bằng việc lấy id từ các tham số của yêu cầu (req.params).

Cập nhật trạng thái chặn của giáo viên: sử dụng TeacherModel.update để cập nhật thuộc tính isBlock của giáo viên thành 1 (chặn) với điều kiện teacher\_id trùng với id.

Trả về phản hồi thành công: nếu cập nhật thành công, trả về phản hồi JSON với thông báo "Teacher has been blocked".

Xử lý lỗi: nếu có bất kỳ lỗi nào xảy ra trong quá trình này, hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 với thông báo lỗi server.

### g) Chặn nhiều giáo viên



```
blockTeachers: async (req, res) => {
    try {
        const { data } = req.body;
        if (!data || !Array.isArray(data) || data.length === 0) {
            return res.status(400).json({ error: 'Invalid or missing data array' });
        }
        const ids = data.map(item => item.id);
        if (ids.length === 0) {
            return res.status(400).json({ error: 'No valid teacher ids provided' });
        }
        await TeacherModel.update({ isBlock: 1 }, { where: { teacher_id: ids } });
        res.status(200).json({ message: 'Teachers have been blocked.' });
    } catch (error) {
        console.error('Error blocking teachers:', error);
        res.status(500).json({ message: 'Server error' });
    }
}
```

Hình 3.92. API chặn nhiều giáo viên

Hàm blockTeachers là một hàm không đồng bộ được thiết kế để chặn nhiều giáo viên cùng một lúc dựa trên một danh sách teacher\_id. Dưới đây là giải thích chi tiết về hoạt động của hàm:

Kiểm tra dữ liệu yêu cầu:

- Hàm bắt đầu bằng việc lấy data từ phần thân của yêu cầu (req.body).
- Kiểm tra xem data có tồn tại, có phải là một mảng và không rỗng. Nếu không hợp lệ, trả về mã trạng thái 400 với thông báo lỗi.

Lấy danh sách teacher\_id:

- Duyệt qua mảng data để lấy danh sách các teacher\_id.
- Nếu danh sách teacher\_id trống, trả về mã trạng thái 400 với thông báo "No valid teacher ids provided".

Cập nhật trạng thái chặn của giáo viên: sử dụng TeacherModel.update để cập nhật thuộc tính isBlock của các giáo viên thành 1 (chặn) với điều kiện teacher\_id nằm trong danh sách ids.

Trả về phản hồi thành công: nếu cập nhật thành công, trả về phản hồi JSON với thông báo "Teachers have been blocked".

Xử lý lỗi: nếu có bất kỳ lỗi nào xảy ra trong quá trình này, hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 với thông báo lỗi server.

#### **h) Bỏ chặn một giáo viên**

```
● ● ●  
unblockTeacher: async (req, res) => {  
  try {  
    const teacherId = req.params.id;  
    await TeacherModel.update({ isBlock: 0 }, { where: { teacher_id: teacherId } });  
    res.status(200).json({ message: 'Teacher has been unblocked.' });  
  } catch (error) {  
    console.error('Error unblocking teacher:', error);  
    res.status(500).json({ message: 'Server error' });  
  }  
}
```

Hình 3.93. API bỏ chặn một giáo viên

Hàm unblockTeacher là một hàm không đồng bộ được thiết kế để mở khóa (unblock) một giáo viên dựa trên teacher\_id. Dưới đây là giải thích chi tiết về hoạt động của hàm:

Lấy teacher\_id từ yêu cầu: hàm bắt đầu bằng việc lấy id từ các tham số của yêu cầu (req.params).

Cập nhật trạng thái mở khóa của giáo viên: sử dụng TeacherModel.update để cập nhật thuộc tính isBlock của giáo viên thành 0 (mở khóa) với điều kiện teacher\_id trùng với id.

Trả về phản hồi thành công: nếu cập nhật thành công, trả về phản hồi JSON với thông báo "Teacher has been unblocked".

Xử lý lỗi: nếu có bất kỳ lỗi nào xảy ra trong quá trình này, hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 với thông báo lỗi server.

### i) Bỏ chặn nhiều giáo viên

```
●●●  
unblockTeachers: async (req, res) => {  
    try {  
        const { data } = req.body;  
        const teacherIds = data.map(item => item.id);  
  
        await TeacherModel.update({ isBlock: 0 }, { where: { teacher_id: teacherIds } });  
        res.status(200).json({ message: 'Teacher has been unblocked.' });  
    } catch (error) {  
        console.error('Error unblocking teacher:', error);  
        res.status(500).json({ message: 'Server error' });  
    }  
}
```

Hình 3.94. API bỏ chặn nhiều giáo viên

Hàm unblockTeachers là một hàm không đồng bộ được thiết kế để mở khóa nhiều giáo viên cùng một lúc dựa trên một danh sách teacher\_id. Dưới đây là giải thích chi tiết về hoạt động của hàm:

Lấy dữ liệu yêu cầu: hàm bắt đầu bằng việc lấy data từ phần thân của yêu cầu (req.body).

Lấy danh sách teacher\_id: duyệt qua mảng data để lấy danh sách các teacher\_id.

Cập nhật trạng thái mở khóa của giáo viên: sử dụng TeacherModel.update để cập nhật thuộc tính isBlock của các giáo viên thành 0 (mở khóa) với điều kiện teacher\_id nằm trong danh sách teacherIds.

Trả về phản hồi thành công: nếu cập nhật thành công, trả về phản hồi JSON với thông báo "Teachers have been unblocked".

Xử lý lỗi: nếu có bất kỳ lỗi nào xảy ra trong quá trình này, hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 với thông báo lỗi server.

### j) Án một giáo viên

```
●●●  
deleteTeacher: async (req, res) => {  
    try {  
        const teacherId = req.params.id;  
        await TeacherModel.update({ isDelete: 1 }, { where: { teacher_id: teacherId } });  
        res.status(200).json({ message: 'Teacher has been deleted.' });  
    } catch (error) {  
        console.error('Error deleting teacher:', error);  
        res.status(500).json({ message: 'Server error' });  
    }  
}
```

Hình 3.95. API án một giáo viên

Hàm deleteTeacher là một hàm không đồng bộ được thiết kế để đánh dấu một giáo viên là đã xóa trong cơ sở dữ liệu dựa trên teacher\_id. Dưới đây là giải thích chi tiết về hoạt động của hàm:

Lấy teacher\_id từ yêu cầu: hàm bắt đầu bằng việc lấy id từ các tham số của yêu cầu (req.params).

Cập nhật trạng thái xóa của giáo viên: sử dụng TeacherModel.update để cập nhật thuộc tính isDelete của giáo viên thành 1 (đã xóa) với điều kiện teacher\_id trùng với id.

Trả về phản hồi thành công: nếu cập nhật thành công, trả về phản hồi JSON với thông báo "Teacher has been deleted."

Xử lý lỗi: nếu có bất kỳ lỗi nào xảy ra trong quá trình này, hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 với thông báo lỗi server.

### k) Ẩn nhiều giáo viên

```
deleteTeachers: async (req, res) => {
  try {
    const { data } = req.body;
    const teacherIds = data.map(item => item.id);
    if (!data || !Array.isArray(data) || data.length === 0) {
      return res.status(400).json({ error: 'Invalid or missing data array' });
    }
    await TeacherModel.update(
      { isDelete: 1 },
      { where: { teacher_id: teacherIds } }
    );
    res.status(200).json({ message: 'Teachers have been deleted.' });
  } catch (error) {
    console.error('Error deleting teachers:', error);
    res.status(500).json({ message: 'Server error' });
  }
}
```

Hình 3.96. API ẩn nhiều giáo viên

Hàm deleteTeachers là một hàm không đồng bộ được thiết kế để đánh dấu nhiều giáo viên là đã xóa trong cơ sở dữ liệu dựa trên danh sách teacher\_id. Dưới đây là giải thích chi tiết về hoạt động của hàm:

Lấy dữ liệu yêu cầu: hàm bắt đầu bằng việc lấy data từ phần thân của yêu cầu (req.body).

Kiểm tra dữ liệu và lấy danh sách teacher\_id:

- Kiểm tra xem data có tồn tại, có phải là một mảng và không rỗng. Nếu không hợp lệ, trả về mã trạng thái 400 với thông báo lỗi.

- Duyệt qua mảng data để lấy danh sách các teacher\_id.

Cập nhật trạng thái xóa của giáo viên: sử dụng TeacherModel.update để cập nhật thuộc tính isDelete của các giáo viên thành 1 (đã xóa) với điều kiện teacher\_id nằm trong danh sách teacherIds.

Trả về phản hồi thành công: nếu cập nhật thành công, trả về phản hồi JSON với thông báo "Teachers have been deleted."

Xử lý lỗi: nếu có bất kỳ lỗi nào xảy ra trong quá trình này, hàm sẽ ghi lại thông báo lỗi và trả về mã trạng thái 500 với thông báo lỗi server.

## I) Xuất file excel mẫu điền thông tin giáo viên

```
● ● ●

getFormTeacher: async (req, res) => {
  console.log("vao")
  const workbook = new ExcelJS.Workbook();
  const worksheet = workbook.addWorksheet('Danh sách giáo viên');

  worksheet.columns = [
    { header: 'Mã giáo viên', key: 'teacherCode', width: 15 },
    { header: 'Tên giáo viên', key: 'name', width: 32 },
    { header: 'Email', key: 'email', width: 30 },
  ];

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

Hình 3.97. API xuất file excel mẫu điền thông tin giáo viên

Hàm getFormTeacher là một hàm không đồng bộ được thiết kế để tạo và trả về một file Excel chứa biểu mẫu thông tin giáo viên. Dưới đây là giải thích chi tiết về hoạt động của hàm:

Tạo workbook và worksheet:

- Sử dụng ExcelJS.Workbook() để tạo một workbook mới.
- Tạo một worksheet mới có tên "Danh sách giáo viên" trong workbook.

Định nghĩa các cột cho worksheet:

- Thiết lập các cột trong worksheet với các thuộc tính: teacherCode, name, và email.
- Đặt tiêu đề cho mỗi cột và thiết lập độ rộng cho chúng.

Thiết lập header cho phản hồi:

- Thiết lập Content-Type trong header của phản hồi để chỉ định rằng nội dung

phản hồi là một tệp Excel.

- Thiết lập Content-Disposition trong header của phản hồi để chỉ định tên tệp là "Teacher.xlsx".

Ghi workbook vào phản hồi và kết thúc:

- Sử dụng workbook.xlsx.write(res) để ghi nội dung của workbook vào phản hồi.
- Gọi res.end() để kết thúc phản hồi.

### m) Xuất file excel với thông tin giáo viên

```
● ● ●

getFormTeacherWithData: async (req, res) => {
  try {
    const { data } = req.body;
    console.log(data);

    if (!data || !Array.isArray(data) || data.length === 0) {
      return res.status(400).json({ error: 'Invalid or missing data array' });
    }
    const ids = data.map(item => item.id);
    if (ids.length === 0) {
      return res.status(400).json({ error: 'No valid teacher codes provided' });
    }

    const workbook = new ExcelJS.Workbook();
    const worksheet = workbook.addWorksheet('Teacher');

    const teachers = await TeacherModel.findAll({
      attributes: ['teacher_id', 'teacherCode', 'email', 'name', 'typeTeacher'],
      where: {
        isDelete: false,
        isBlock: false,
        teacher_id: ids
      }
    });

    worksheet.columns = [
      { header: 'Mã giáo viên', key: 'teacherCode', width: 15 },
      { header: 'Tên giáo viên', key: 'name', width: 32 },
      { header: 'Email', key: 'email', width: 30 },
      { header: 'Loại giáo viên', key: 'typeTeacher', width: 20 },
    ];

    teachers.forEach(teacher => {
      worksheet.addRow({
        teacherCode: teacher.teacherCode.toString(),
        name: teacher.name,
        email: teacher.email,
        typeTeacher: teacher.typeTeacher,
      });
    });

    res.setHeader('Content-Type', 'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet');
    res.setHeader('Content-Disposition', 'attachment; filename="Teacher.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' });
  }
}
```

Hình 3.98. API xuất file excel với thông tin giáo viên

Hàm getFormTeacherWithData là một hàm không đồng bộ được thiết kế để tạo và trả về một file Excel chứa thông tin về các giáo viên dựa trên danh sách teacher\_id được cung cấp trong yêu cầu. Dưới đây là giải thích chi tiết về hoạt động của hàm:

Lấy và kiểm tra dữ liệu yêu cầu:

- Hàm bắt đầu bằng việc lấy data từ phần thân của yêu cầu (req.body).
- Kiểm tra tính hợp lệ của data để đảm bảo rằng nó là một mảng và không rỗng. Nếu không hợp lệ, trả về mã trạng thái 400 với thông báo lỗi.

Lấy danh sách teacher\_id: duyệt qua mảng data để lấy danh sách các teacher\_id.

Tạo workbook và worksheet:

- Sử dụng ExcelJS.Workbook() để tạo một workbook mới.
- Tạo một worksheet mới có tên "Teacher" trong workbook.

Tìm kiếm thông tin giáo viên: sử dụng TeacherModel.findAll để tìm kiếm tất cả các giáo viên có teacher\_id nằm trong danh sách ids và chưa bị xóa hoặc chặn.

Định nghĩa các cột cho worksheet:

- Thiết lập các cột trong worksheet với các thuộc tính: teacherCode, name, email, và typeTeacher.
- Đặt tiêu đề cho mỗi cột và thiết lập độ rộng cho chúng.

Thêm thông tin giáo viên vào worksheet: duyệt qua danh sách giáo viên và thêm từng giáo viên vào worksheet với các thông tin tương ứng.

Thiết lập header cho phản hồi:

- Thiết lập Content-Type trong header của phản hồi để chỉ định rằng nội dung phản hồi là một tệp Excel.
- Thiết lập Content-Disposition trong header của phản hồi để chỉ định tên tệp là "Teacher.xlsx".

Ghi workbook vào phản hồi và kết thúc:

- Sử dụng workbook.xlsx.write(res) để ghi nội dung của workbook vào phản hồi.

- Gọi res.end() để kết thúc phản hồi.

### n) Lưu giáo viên bằng file excel

```

● ● ●

saveTeacherExcel: async (req, res) => {
  if (req.files) {
    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(1);
    const insertPromises = [];
    const emailsSet = new Set();
    const idsSet = new Set();

    worksheet.eachRow({ includeEmpty: false }, async (row, rowNumber) => {
      if (rowNumber !== 1) { // Skip the header row
        let email = row.getCell(3).value;
        let teacherCode = row.getCell(1).value;

        if (email && typeof email === 'object' && email.hasOwnProperty('text')) {
          email = email.text;
        }

        if (!email || emailsSet.has(email)) {
          console.error(`Duplicate or invalid email found at row ${rowNumber}: ${email}`);
          return; // Skip this row
        }

        if (!teacherCode || idsSet.has(teacherCode)) {
          console.error(`Duplicate or invalid teacherCode found at row ${rowNumber}: ${teacherCode}`);
          teacherCode = await generateUniqueTeacherCode();
        }
        emailsSet.add(email);
        idsSet.add(teacherCode);
        const salt = await bcrypt.genSalt(10);
        const hashedPassword = await bcrypt.hash(teacherCode.toString(), salt);
        const sql = `INSERT INTO teachers (teacherCode, name, email, password)
                    VALUES (?, ?, ?, ?)`;
        const values = [
          teacherCode,
          row.getCell(2).value,
          email,
          hashedPassword
        ];

        insertPromises.push(sequence.query(sql, { replacements: values })
          .catch(error => {
            console.error(`Error inserting row ${rowNumber}: ${error.message}`);
          }));
      }
    });
    await Promise.all(insertPromises);
    console.log('All data has been inserted into the database!');
    fs.unlinkSync(filePath); // Remove the uploaded file after processing
    res.status(200).json({ message: "Dữ liệu đã được lưu thành công vào cơ sở dữ liệu." });
  } else {
    res.status(400).send('No file uploaded.');
  }
}

```

Hình 3.99. API lưu giáo viên bằng file excel.

Hàm saveTeacherExcel là một hàm không đồng bộ được thiết kế để xử lý file Excel tải lên, đọc dữ liệu từ file, và chèn dữ liệu giáo viên vào cơ sở dữ liệu. Dưới đây là giải thích chi tiết về hoạt động của hàm:

Kiểm tra file tải lên: hàm bắt đầu bằng việc kiểm tra xem file có được tải lên hay không (req.files).

Đọc file Excel: xác định đường dẫn tới file Excel và đọc nội dung của nó bằng cách sử dụng ExcelJS.Workbook().

Khởi tạo các tập hợp để kiểm tra trùng lặp email và mã giáo viên: sử dụng các tập hợp (Set) để lưu trữ các email và mã giáo viên đã được xử lý để tránh trùng lặp.

Duyệt qua các hàng trong worksheet: sử dụng worksheet.eachRow để duyệt qua từng hàng trong worksheet, bỏ qua hàng đầu tiên vì nó chứa tiêu đề.

Kiểm tra và xử lý trùng lặp email và mã giáo viên:

- Lấy giá trị của email và mã giáo viên từ các cột tương ứng.
- Kiểm tra và loại bỏ các email và mã giáo viên trùng lặp hoặc không hợp lệ.
- Nếu mã giáo viên trùng lặp, tạo một mã giáo viên mới duy nhất bằng cách sử dụng hàm generateUniqueTeacherCode.

Hash mật khẩu và chèn dữ liệu vào cơ sở dữ liệu:

- Sử dụng bcrypt để tạo hash mật khẩu từ mã giáo viên.
- Tạo câu lệnh SQL để chèn dữ liệu vào bảng teachers.
- Sử dụng sequelize.query để thực thi câu lệnh SQL với các giá trị thay thế.
- Lưu các promise chèn dữ liệu vào mảng insertPromises.

Chờ tất cả các promise hoàn thành và xử lý hoàn tất:

- Sử dụng Promise.all để đợi tất cả các promise chèn dữ liệu hoàn thành.
- Xóa file Excel sau khi xử lý xong (fs.unlinkSync).

Trả về phản hồi thành công hoặc lỗi:

- Nếu không có lỗi xảy ra, trả về mã trạng thái 200 với thông báo thành công.
- Nếu có lỗi, ghi lại thông báo lỗi và trả về mã trạng thái 500 với thông báo lỗi server.

## 3.7. Xây dựng Frontend

### 3.7.1. Cấu hình kết nối với thư viện, framework

#### 3.7.1.1. Tailwind



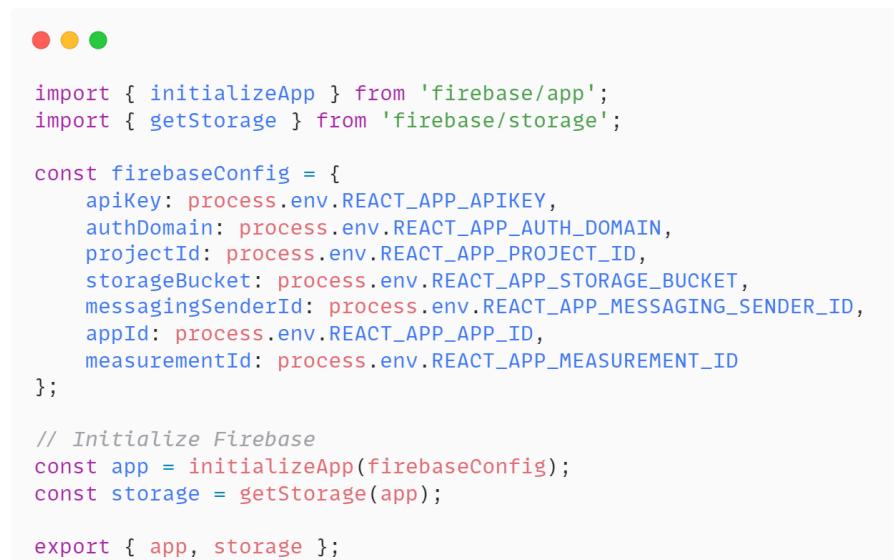
```
/* @type {import('tailwindcss').Config} */
const { nextui } = require("@nextui-org/react");

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

Hình 3.100. Cấu hình tệp tailwind.config.js

Tệp cấu hình này giúp tích hợp Tailwind CSS với NextUI trong dự án React. Nó xác định các tệp cần quét để tìm các lớp Tailwind, mở rộng chủ đề nếu cần, kích hoạt chế độ tối thông qua lớp class, và tích hợp các thành phần giao diện từ NextUI. Điều này giúp dễ dàng sử dụng các thành phần giao diện hiện đại và tùy chỉnh chúng theo ý muốn trong dự án React của mình.

#### 3.7.1.2. FireBase



```
import { initializeApp } from 'firebase/app';
import { getStorage } from 'firebase/storage';

const firebaseConfig = {
  apiKey: process.env.REACT_APP_APIKEY,
  authDomain: process.env.REACT_APP_AUTH_DOMAIN,
  projectId: process.env.REACT_APP_PROJECT_ID,
  storageBucket: process.env.REACT_APP_STORAGE_BUCKET,
  messagingSenderId: process.env.REACT_APP_MESSAGING_SENDER_ID,
  appId: process.env.REACT_APP_APP_ID,
  measurementId: process.env.REACT_APP_MEASUREMENT_ID
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);
const storage = getStorage(app);

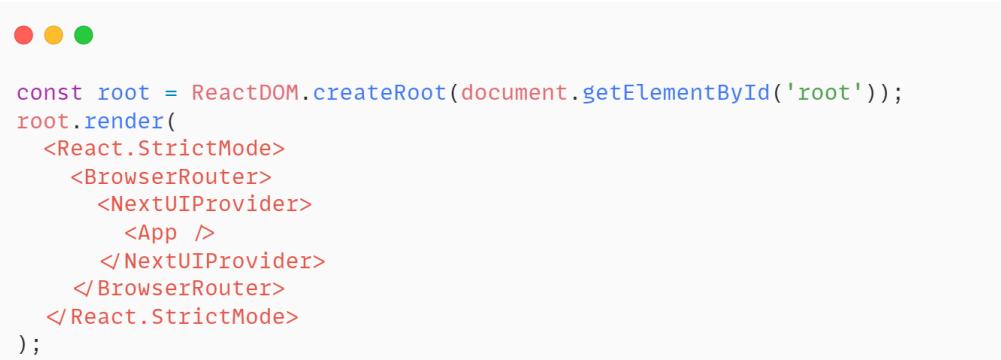
export { app, storage };
```

Hình 3.101. Cấu hình kết nối fire base

### 3.7.2. Tạo Giao Diện Người Dùng

Mục tiêu: Phát triển giao diện người dùng (UI) thân thiện và dễ sử dụng.

Công cụ: Sử dụng React và các thư viện UI như NextUI, Ant Design, hoặc Tailwind CSS.



```
● ● ●

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <BrowserRouter>
      <NextUIProvider>
        <App />
      </NextUIProvider>
    </BrowserRouter>
  </React.StrictMode>
);
```

Hình 3.102. Khởi tạo root và render ứng dụng

ReactDOM.createRoot: Tạo một gốc (root) mới để render ứng dụng React.

React.StrictMode: Chế độ kiểm tra nghiêm ngặt để phát hiện các vấn đề tiềm ẩn trong ứng dụng.

BrowserRouter: Cung cấp các tính năng điều hướng cho ứng dụng.

NextUIProvider: Cung cấp các thành phần UI từ NextUI.

App: Thành phần gốc được render bên trong các provider.

### 3.7.3. Cấu Hình Axios

#### Import thư viện Axios



```
● ● ●

import axios from "axios";
```

Hình 3.103. Import thư viện Axios

Axios là thư viện giúp thực hiện các yêu cầu HTTP dễ dàng.

Tạo Instance Axios với Cấu Hình Mặc Định

```

● ● ●

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

```

Hình 3.104. Instance axios với cấu hình mặc định

`axios.create`: Tạo một instance Axios mới với các cấu hình mặc định.

`withCredentials`: Cho phép gửi cookie trong các yêu cầu cross-origin.

`baseURL`: URL cơ bản cho tất cả các yêu cầu được cấu hình từ biến môi trường.

### Thiết Lập Interceptors cho Axios

```

● ● ●

axiosAdmin.interceptors.response.use(
  response => response,
  async error => {
    const originalRequest = error.config;
    if (error.response && error.response.status === 401 && !originalRequest._retry) {
      originalRequest._retry = true;

      try {
        // Gọi endpoint /refresh-token để lấy token mới
        const response = await
          axios.post(`${process.env.REACT_APP_API_DOMAIN_CLIENT}/refresh-token`, {}, {
            withCredentials: true
          });

        if (response.status === 200) {
          // Lấy token mới từ response
          const newAccessToken = response.data.accessToken;

          // Cập nhật header Authorization cho request gốc và thử lại request
          originalRequest.headers['Authorization'] = `Bearer ${newAccessToken}`;
          return axiosAdmin(originalRequest);
        }
      } catch (refreshError) {
        return Promise.reject(refreshError);
      }
    }
    return Promise.reject(error);
  }
);

```

Hình 3.105. Thiết lập interceptors cho Axios

`interceptors.response.use`: Thiết lập interceptor cho phản hồi từ Axios.

`response => response`: Trả về phản hồi nếu không có lỗi.

`async error => {...}`: Hàm bắt đồng bộ để xử lý lỗi.

originalRequest: Lưu trữ yêu cầu ban đầu để thử lại sau khi lấy token mới.

error.response.status === 401: Kiểm tra lỗi 401 (Unauthorized).

!originalRequest.\_retry: Đảm bảo yêu cầu không được thử lại nhiều lần.

originalRequest.\_retry = true: Đánh dấu yêu cầu là đã thử lại.

axios.post(\${process.env.REACT\_APP\_API\_DOMAIN\_CLIENT}/refresh-token, {}, { withCredentials: true }): Gửi yêu cầu đến endpoint /refresh-token để lấy token mới.

response.data.accessToken: Lấy token mới từ phản hồi.

originalRequest.headers['Authorization'] = Bearer \${newAccessToken}: Cập nhật header Authorization với token mới.

axiosAdmin(originalRequest): Thử lại yêu cầu ban đầu với token mới.

### Sử dụng



```
const response = await axiosAdmin.get(`/students?page=${page}&size=${size}`);
```

Hình 3.106. Ví dụ về sử dụng axiosAdmin

axiosAdmin.get: Sử dụng phương thức GET của instance axiosAdmin để gửi yêu cầu.

/students?page=\${page}&size=\${size}: Endpoint API.

## CHƯƠNG 4. KẾT QUẢ NGHIÊN CỨU

### 4.1. Kết quả nghiên cứu Backend

#### 4.1.1. Khởi chạy

Khởi chạy bằng cách chạy câu lệnh npm start

```
PS D:\DoAnTotNghiep\Source\datn\backend> npm start

> scr@1.0.0 start
> nodemon --inspect app.js

[nodemon] 3.1.3
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node --inspect app.js`
Debugger listening on ws://127.0.0.1:9229/72051353-9fbf-406d-8585-9a2a520f59d6
For help, see: https://nodejs.org/en/docs/inspector
App listening http://localhost:1509
Executing (default): SELECT 1+1 AS result
Connection has been established successfully.
```

Hình 4.1. Khởi chạy backend

Khi chạy lệnh npm start, nodemon sẽ giám sát tất cả các tệp trong dự án và tự động khởi động lại server nếu có bất kỳ thay đổi nào. Ta sẽ thấy thông báo trong console cho biết server đang chạy và kết nối cơ sở dữ liệu thành công.

#### 4.1.2. Xác thực người dùng

Đăng nhập người dùng bằng teacherCode và password



```
login: async (req, res) => {
  const { teacherCode, password } = req.body;
  try {
    const user = await TeacherModel.findOne({ where: { teacherCode } });
    if (!user) {
      return res.status(400).json({ message: 'Teacher code hoặc mật khẩu không đúng' });
    }

    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) {
      return res.status(400).json({ message: 'Teacher code hoặc mật khẩu không đúng' });
    }

    const payload = { id: user.teacher_id, permission: user.permission };
    const accessToken = jwt.sign(payload, process.env.JWT_SECRET, { expiresIn: '30m' });
    const refreshToken = jwt.sign(payload, process.env.JWT_SECRET, { expiresIn: '7d' });

    // Thu hồi và hết hạn tất cả các refresh token cũ của người dùng
    await RefreshTokenModel.update(
      { revoked: true, expired: true },
      { where: { teacher_id: user.teacher_id } }
    );

    // Lưu refresh token mới vào database
    await RefreshTokenModel.create({ token: refreshToken, teacher_id: user.teacher_id });

    // Đặt token trong cookies
    res.cookie('accessToken', accessToken, { httpOnly: false, secure: true, sameSite: 'Strict', maxAge: 30 * 60 * 1000 });
    res.cookie('refreshToken', refreshToken, { httpOnly: false, secure: true, sameSite: 'Strict', maxAge: 7 * 24 * 60 * 60 * 1000 });

    console.log(`Đăng nhập thành công cho người dùng: ${user.name}`);
    res.json({
      message: 'Đăng nhập thành công',
      accessToken,
      refreshToken
    });
  } catch (error) {
    console.error(`Lỗi đăng nhập: ${error.message}`);
    res.status(500).json({ message: 'Lỗi server', error });
  }
},
```

Hình 4.2. API xử lý đăng nhập

Dữ liệu đầu vào vào teacherCode, password từ body của yêu cầu HTTP POST do người dùng gửi đến.



```
const user = await TeacherModel.findOne({ where: { teacherCode } });
if (!user) {
  return res.status(400).json({ message: 'Teacher code hoặc mật khẩu không đúng' });
}
```

Hình 4.3. Tìm người dùng trong cơ sở dữ liệu

Sử dụng hàm findOne của mô hình TeacherModel để tìm kiếm người dùng trong cơ sở dữ liệu dựa trên teacherCode.

Nếu không tìm thấy người dùng nào khớp với teacherCode, trả về mã trạng thái 400

và thông báo lỗi.

```
● ● ●  
const isMatch = await bcrypt.compare(password, user.password);  
if (!isMatch) {  
    return res.status(400).json({ message: 'Teacher code hoặc mật khẩu không đúng' });  
}
```

Hình 4.4. Kiểm tra mật khẩu

Dòng 1: Sử dụng bcrypt.compare để so sánh mật khẩu do người dùng nhập với mật khẩu đã mã hóa được lưu trữ trong cơ sở dữ liệu.

Dòng 2-4: Nếu mật khẩu không khớp, trả về mã trạng thái 400 và thông báo lỗi.

```
● ● ●  
const payload = { id: user.teacher_id, permission: user.permission };  
const accessToken = jwt.sign(payload, process.env.JWT_SECRET, { expiresIn: '30m' });  
const refreshToken = jwt.sign(payload, process.env.JWT_SECRET, { expiresIn: '7d' });
```

Hình 4.5. khởi tạo các biến cần thiết.

Dòng 1: Tạo đối tượng payload chứa thông tin người dùng cần thiết cho mã thông báo.

Dòng 2: Sử dụng jwt.sign để tạo mã thông báo truy cập (access token) với thời gian hết hạn là 30 phút.

Dòng 3: Tạo mã thông báo làm mới (refresh token) với thời gian hết hạn là 7 ngày.

```
● ● ●  
await RefreshTokenModel.update(  
    { revoked: true, expired: true },  
    { where: { teacher_id: user.teacher_id } }  
)
```

Hình 4.6. Thu hồi và hết hạn các biến cần thiết.

Sử dụng hàm update của mô hình RefreshTokenModel để cập nhật trạng thái của các mã thông báo làm mới cũ, đánh dấu chúng là đã thu hồi và hết hạn, dựa trên teacher\_id của người dùng.

```
● ● ●  
await RefreshTokenModel.create({ token: refreshToken, teacher_id: user.teacher_id });
```

Hình 4.7. Lưu token vào cơ sở dữ liệu

Sử dụng hàm create của mô hình RefreshTokenModel để lưu mã thông báo làm mới mới vào cơ sở dữ liệu, gắn với teacher\_id của người dùng.

```
● ● ●

res.cookie('accessToken', accessToken, { httpOnly: false, secure: true,
sameSite: 'Strict', maxAge: 30 * 60 * 1000 });

res.cookie('refreshToken', refreshToken, { httpOnly: false, secure: true,
sameSite: 'Strict', maxAge: 7 * 24 * 60 * 60 * 1000 })
```

Hình 4.8. Đặt mã thông báo trong cookies.

Đặt mã thông báo truy cập vào cookie với các tùy chọn bảo mật như httpOnly, secure, sameSite và thời gian hết hạn là 30 phút đối với accessToken, 7 ngày đối với refreshToken.

#### 4.1.3. Các API

Khi chưa đăng nhập

The screenshot shows a POSTMAN interface. The URL is `http://localhost:1509/api/user`. The response status is `401 Unauthorized` with a time of `9 ms` and a size of `388 B`. The response body is a JSON object with the message `"message": "Access token is missing"`.

Khi chưa đăng nhập và thực hiện yêu cầu GET đến đường dẫn `http://localhost:1509/api/user`, hệ thống sẽ trả về một thông báo lỗi với mã trạng thái 401 Unauthorized. Thông báo lỗi chi tiết sẽ là một đối tượng JSON với nội dung như sau:

```
● ● ●

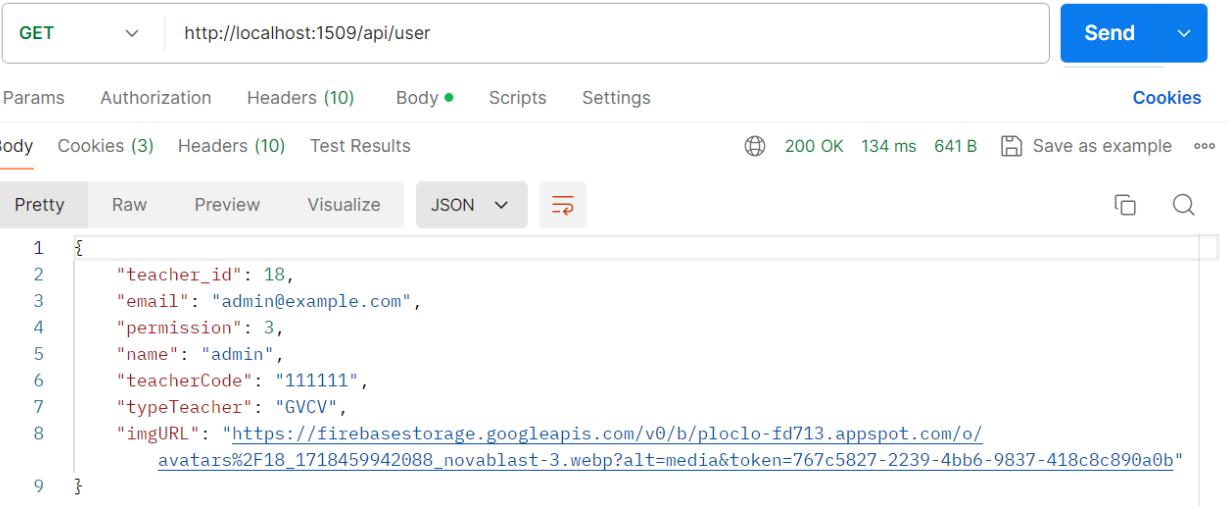
{
  "message": "Access token is missing"
}
```

Hình 4.9. Phải hồi khi chưa đăng nhập khi gọi end point

Điều này cho biết rằng yêu cầu đã thiếu mã thông báo truy cập (access token) cần thiết để xác thực và ủy quyền người dùng. Để khắc phục lỗi này, cần đảm bảo rằng mã

thông báo truy cập hợp lệ được gửi kèm theo yêu cầu.

## Khi đã đăng nhập



GET http://localhost:1509/api/user

Params Authorization Headers (10) Body Scripts Settings Cookies

Body Cookies (3) Headers (10) Test Results 200 OK 134 ms 641 B Save as example ...

Pretty Raw Preview Visualize JSON ↻

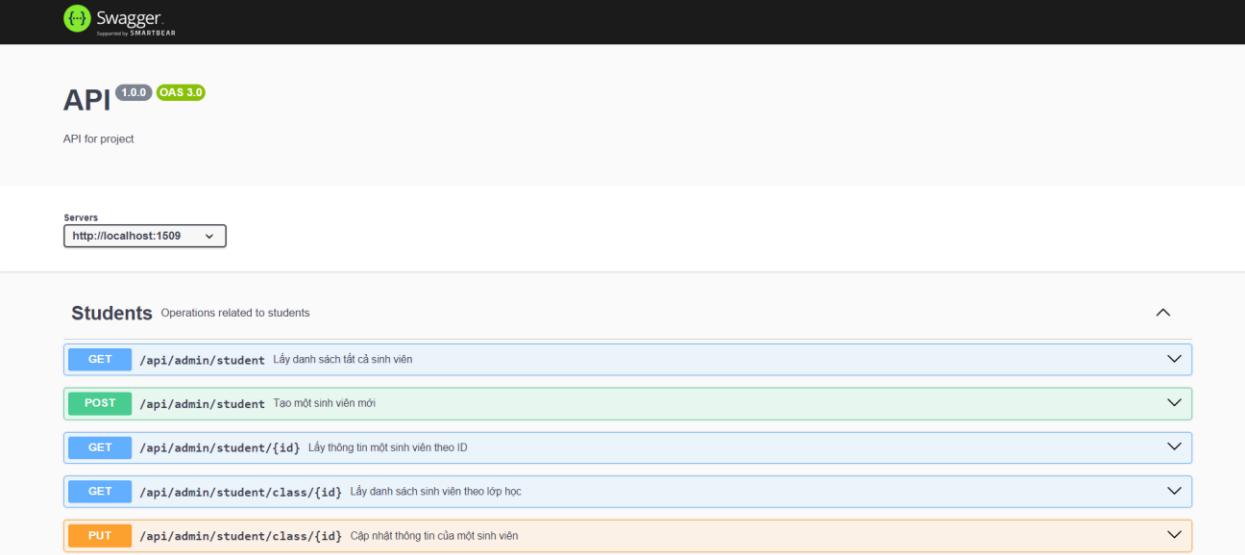
```
1 {  
2   "teacher_id": 18,  
3   "email": "admin@example.com",  
4   "permission": 3,  
5   "name": "admin",  
6   "teacherCode": "111111",  
7   "typeTeacher": "GVCV",  
8   "imgURL": "https://firebasestorage.googleapis.com/v0/b/ploclo-fd713.appspot.com/o/avatars%2F18_1718459942088_novablast-3.webp?alt=media&token=767c5827-2239-4bb6-9837-418c8c890a0b"  
9 }
```

Hình 4.10. Nội dung trả lời khi đăng nhập

Khi có token khi đăng nhập thì sẽ lấy accessToken trên cookie

### 4.1.4. Swagger

Các API cũng được triển khai trên Swagger cho người dùng thao tác dễ dàng



Swagger Supported by SMARTBEAR

API 1.0.0 OAS 3.0

API for project

Servers http://localhost:1509

**Students** Operations related to students

GET /api/admin/student Lấy danh sách tất cả sinh viên

POST /api/admin/student Tạo một sinh viên mới

GET /api/admin/student/{id} Lấy thông tin một sinh viên theo ID

GET /api/admin/student/class/{id} Lấy danh sách sinh viên theo lớp học

PUT /api/admin/student/class/{id} Cập nhật thông tin của một sinh viên

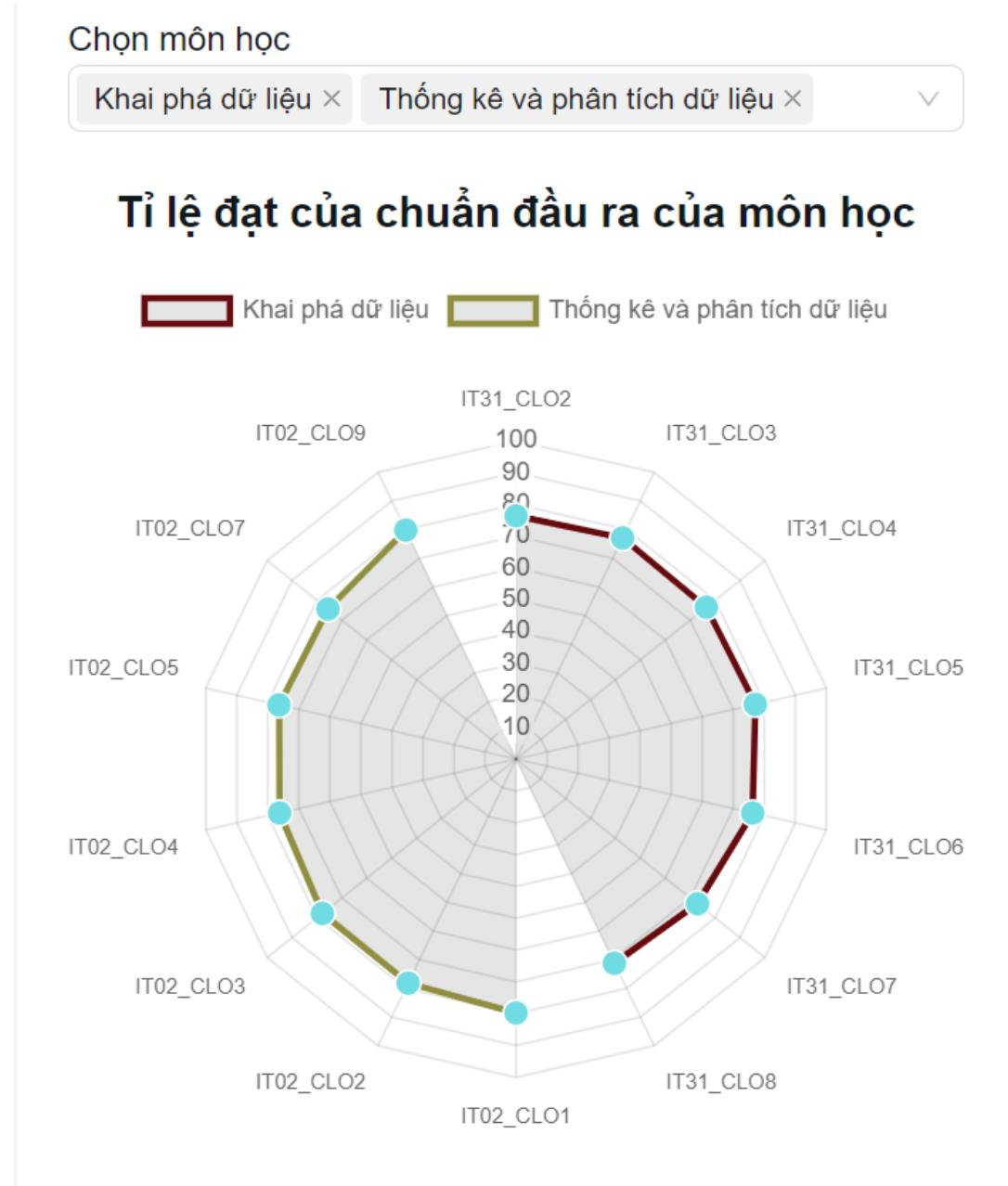
Hình 4.11. Giao diện swagger

Đây là nơi hiển thị cũng như các API và cũng là nơi tương tác.

## 4.2. Kết quả nghiên cứu Frontend

### 4.2.1. Kết quả trực quan hóa kết quả học tập

#### 4.2.1.1. Tỉ lệ đạt được của chuẩn đầu ra môn học



Hình 4.12. Tỉ lệ đạt được của chuẩn đầu ra môn học

Biểu đồ radar trong hình ảnh cung cấp là một công cụ trực quan hóa kết quả học tập cho hai môn học: "Khai phá dữ liệu" và "Thống kê và phân tích dữ liệu".

Ta có thể lựa chọn hiển thị nhiều môn hoặc ít môn trên thanh chọn môn học.

Biểu đồ hiển thị các điểm dữ liệu dọc theo mỗi trục tương ứng với tỷ lệ đạt được

cho từng CLO.

Thang đo tỷ lệ phần trăm từ 0 đến 100, với các khoảng cách được đánh dấu ở 10, 20, 30, v.v. lên đến 100.

Biểu đồ radar được sử dụng để so sánh hiệu suất trên các chuẩn đầu ra khác nhau cho nhiều môn học. Nó giúp các nhà giáo dục và sinh viên xác định các điểm mạnh và các khu vực cần cải thiện trong các chuẩn đầu ra học tập cụ thể cho từng môn học.

#### 4.2.1.2. Tỉ lệ đạt của chuẩn đầu ra chương trình



Hình 4.13. Biểu đồ đường tỉ lệ đạt của chuẩn đầu ra chương trình.

Hình ảnh cung cấp là một biểu đồ đường thể hiện tỉ lệ đạt của các chuẩn đầu ra chương trình (Program Learning Outcomes - PLO) khác nhau.

Đánh giá chất lượng chương trình: Biểu đồ này giúp các nhà quản lý giáo dục đánh giá hiệu quả của chương trình học dựa trên tỷ lệ đạt các chuẩn đầu ra.

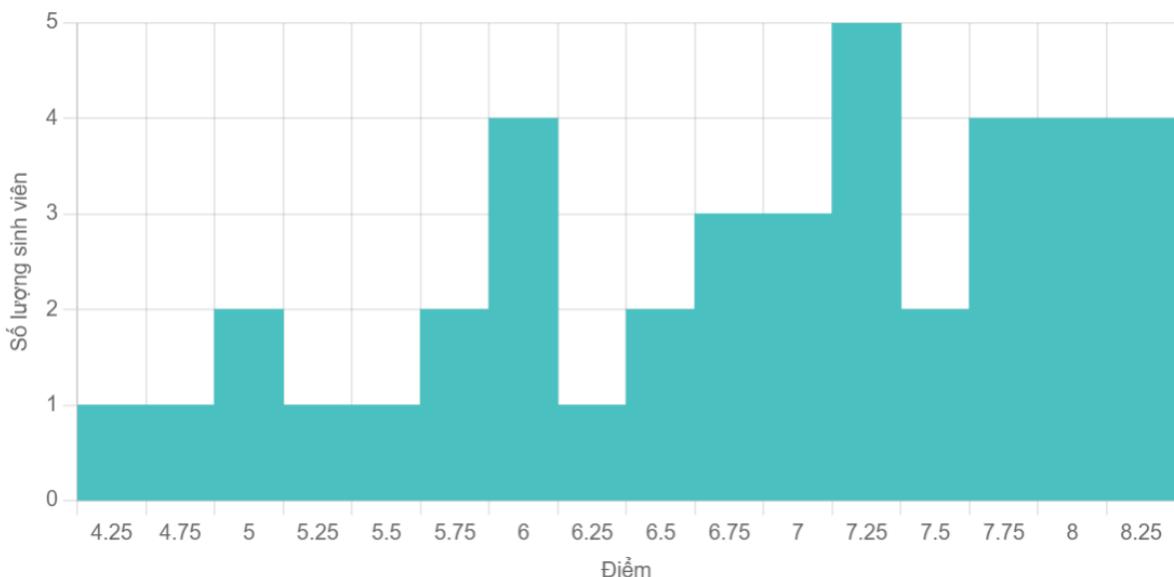
Phát hiện điểm mạnh và yếu: Giúp xác định các PLO nào đạt được tỷ lệ cao và PLO nào cần cải thiện. Ví dụ, IT\_PLO1 và IT\_PLO10 có tỷ lệ đạt cao hơn các PLO khác, trong khi IT\_PLO2 và IT\_PLO8 có tỷ lệ đạt thấp hơn.

Quản lý và cải tiến chương trình: Dữ liệu từ biểu đồ này có thể được sử dụng để cải tiến các khía cạnh của chương trình học, tập trung vào các lĩnh vực cần được cải thiện để nâng cao chất lượng giáo dục tổng thể.

Biểu đồ này không chỉ là công cụ để đánh giá hiệu quả hiện tại của chương trình học mà còn là cơ sở để cải thiện và nâng cao chất lượng giáo dục trong tương lai. Nó cung cấp một cái nhìn tổng quan về mức độ đạt được của các chuẩn đầu ra, giúp nhà quản lý giáo dục và giáo viên đưa ra các quyết định chiến lược để nâng cao kết quả học tập của sinh viên.

#### 4.2.1.3. Biểu đồ phân bố điểm

**Phân bố điểm của lớp Công nghệ phần mềm DA20TTB**



*Hình 4.14. Phân bố điểm của khóa học.*

Biểu đồ phân tán trong hình ảnh cung cấp là một công cụ trực quan hóa phân bố điểm số của lớp Công nghệ phần mềm DA20TTB.

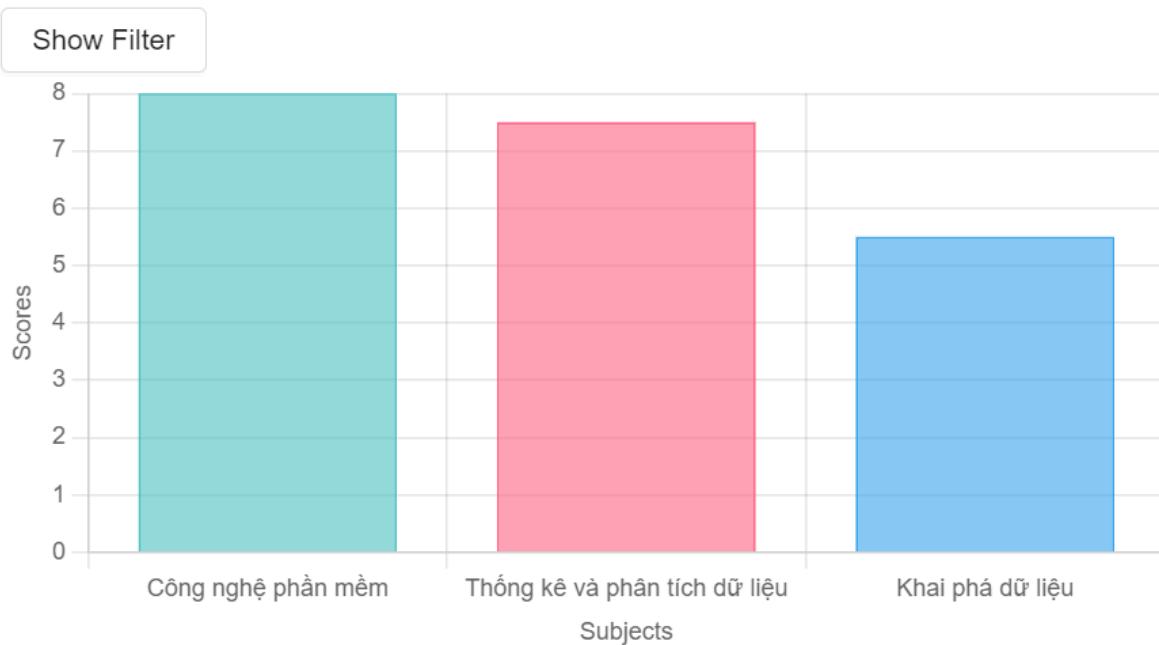
Mỗi điểm trên biểu đồ đại diện cho điểm số của một hoặc nhiều sinh viên.

Màu đỏ nhạt được sử dụng để biểu thị các điểm dữ liệu.

Khi di chuột qua một điểm, thông tin chi tiết về sinh viên sẽ xuất hiện. Ví dụ, tại điểm có tọa độ (7.5, 2), có thông tin: "Students: Nguyễn Thị Mỹ Yên, Nguyễn Thái Diên", nghĩa là có hai sinh viên đạt điểm 7.5 và có số lượng là 2.

Biểu đồ cho thấy sự phân bố điểm số trong lớp học, giúp nhận diện các khoảng điểm phổ biến và số lượng sinh viên đạt được điểm số đó. Biểu đồ phân tán này giúp giảng viên và sinh viên thấy được sự phân bố điểm số trong lớp học. Nó giúp xác định các điểm mạnh và yếu trong việc học tập, từ đó có thể điều chỉnh phương pháp giảng dạy, học tập phù hợp.

#### 4.2.1.4. Thể hiện điểm đạt được của sinh viên



Hình 4.15. Biểu đồ cột thể hiện điểm của sinh viên

Biểu đồ cột trong hình ảnh cung cấp là một công cụ trực quan hóa điểm số của ba môn sinh viên gần nhất.

Sự khác biệt về điểm số trung bình giữa các môn học có thể cho thấy mức độ khó khăn hoặc hiệu quả học tập của sinh viên trong từng môn học.

Biểu đồ cột này giúp giảng viên và sinh viên so sánh điểm số trung bình giữa các môn học khác nhau. Nó cung cấp thông tin hữu ích để đánh giá và cải thiện chất lượng giảng dạy và học tập.

Dưới biểu đồ nội dung chi tiết từng cột:

## Công nghệ phần mềm

Công nghệ phần mềm DA20TTB

Năm học 2019-2020

HKI

Score: 8.25

Teacher: Nguyễn Trần Diễm Hạnh Class: Công nghệ Thông tin 2020 B

## Thống kê và phân tích dữ liệu

Thống kê và phân tích dữ liệu DA20TTB

Năm học 2019-2020

HKII

Score: 6.75

Teacher: Nguyễn Bảo Ân Class: Công nghệ Thông tin 2020 B

## Khai phá dữ liệu

Khai phá dữ liệu DA20TTB

Năm học 2022-2023

HKII

Score: 5.25

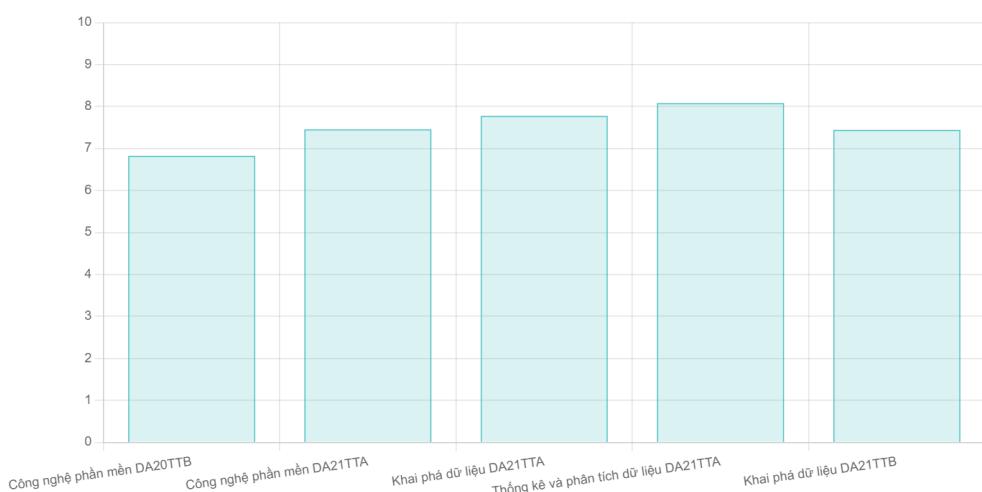
Teacher: Nguyễn Bảo Ân Class: Công nghệ Thông tin 2020 B

Hình 4.16. Chi tiết biểu đồ cột.

Cung cấp một cái nhìn tổng quan về kết quả học tập của các sinh viên trong các môn học khác nhau. Nó giúp giáo viên và sinh viên so sánh điểm số giữa các môn học và đánh giá hiệu quả học tập.

### 4.2.1.5. Biểu đồ thể hiện điểm trung bình của khóa học

Điểm trung bình của khóa học



Hình 4.17. Biểu đồ điểm trung bình khóa học

Hình ảnh này cung cấp một biểu đồ cột thể hiện điểm trung bình của các khóa học. Được lọc bởi filter sau:

The screenshot shows a 'Filters' section with three main categories: Năm học (Academic Year), Học kỳ (Semester), and Lớp học (Class). Each category has a dropdown menu labeled 'Chọn năm học' (Select academic year), 'Chọn học kỳ' (Select semester), and 'Chọn lớp' (Select class) respectively. Below these, there are two more dropdown menus: Subject (Chọn môn học) and Course (Chọn học phần).

*Hình 4.18. Filter cấubiểu đồđiểm trung bình khóa học.*

Khi di chuột vào cột của từng khóa học, thông tin chi tiết sẽ xuất hiện, bao gồm:

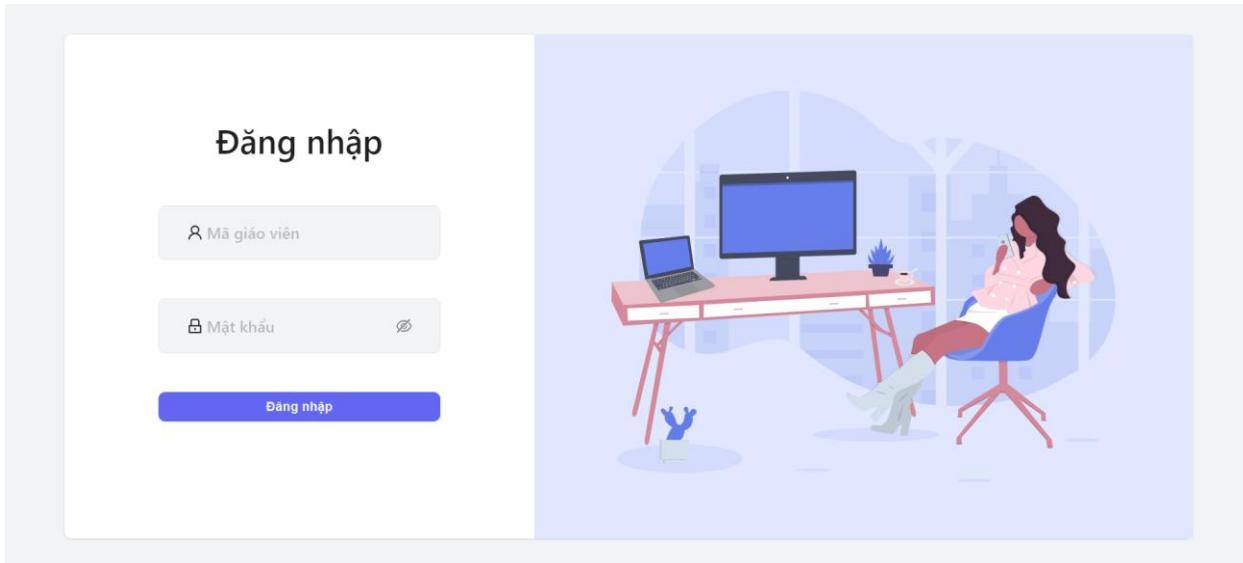
- Điểm trung bình (Average Score): Điểm số trung bình của khóa học.
- Giáo viên (Teacher): Giáo viên dạy khóa học.
- Học kỳ (Semester): Học kỳ mà khóa học diễn ra.
- Năm học (Academic Year): Năm học của khóa học.
- Lớp (Class): Lớp học của khóa học.

Phát hiện điểm mạnh và yếu: Giúp phát hiện ra điểm mạnh và yếu trong chương trình giảng dạy của từng khóa học.

Hướng dẫn cải thiện giáo dục: Cung cấp thông tin hữu ích để hướng dẫn cải thiện chất lượng giáo dục, tập trung vào các khóa học cần được cải thiện.

Định hướng phát triển học viên: Giúp học viên và giảng viên hiểu rõ hơn về mức độ đạt được của từng khóa học, từ đó có thể định hướng phát triển và cải thiện học tập phù hợp.

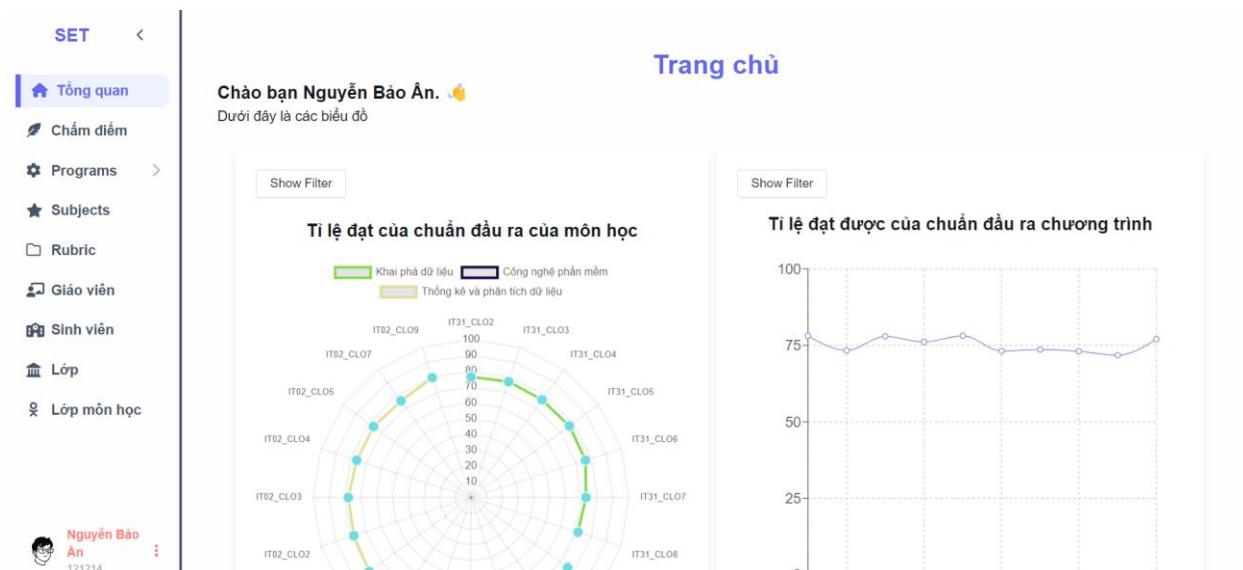
#### 4.4.1. Giao diện đăng nhập



Hình 4.19. Giao diện đăng nhập.

Người dùng đăng nhập bằng tài khoản được cấp bao gồm mã giáo viên và mật khẩu.

#### 4.4.2. Giao diện trang chủ



Hình 4.20. Giao diện trang chủ

Trang chủ là nơi có biểu đồ hiển thị trực quan kết quả học tập của các sinh viên. Tại đây, người dùng có thể dễ dàng theo dõi và so sánh thành tích học tập của các sinh viên thông qua các biểu đồ tương tác. Các biểu đồ này cung cấp cái nhìn tổng quan về điểm số, tiến bộ học tập theo thời gian, và sự phân bố thành tích giữa các sinh viên. Giao diện trực quan và sinh động giúp người dùng nhanh chóng nắm bắt thông tin và đưa ra các quyết định phù hợp dựa trên dữ liệu trực quan và phân tích chi tiết.

#### 4.4.3. Giao diện quản lý giáo viên

The screenshot shows a user interface for managing teachers. On the left, there is a sidebar with navigation links: Tổng quan, Chấm điểm, Programs, Subjects, Rubric, Giáo viên (selected), Sinh viên, Lớp, and Lớp môn học. Below the sidebar, a user profile is displayed: Nguyễn Bảo An, 121214. The main area is titled 'Danh sách giáo viên' and shows a table with 13 rows of teacher data. The columns are: Name, Teacher Code, Permission Name, and Action. The data includes:

Name	Teacher Code	Permission Name	Action
Phạm Minh Dương duongminh@tvu.edu.vn	123456	Giáo Viên GVCV	
Trầm Hoàng Nam tramhoangnam@tvu.edu.vn	100006	Giáo Viên GVGD	
Phan Thị Phương Nam ptpnam@tvu.edu.vn	123457	Giáo Viên GVCV	
Phạm Thị Trúc Mai pttmai@tvu.edu.vn	100007	Giáo Viên GVGD	
Võ Thành C vothanhc@tvu.edu.vn	111111	Siêu Quản Trị Viên GVCV	

At the bottom, there is a pagination control with pages 1, 2, 3, and Next.

Hình 4.21. Danh sách giáo viên.

Trang quản lý giáo viên là một phần của hệ thống quản lý học tập, nơi cung cấp giao diện trực quan để quản lý thông tin và quyền truy cập của giáo viên. Trang này bao gồm các thành phần chính như sau:

**Thanh tìm kiếm:** Nằm ở phía trên cùng của trang, thanh tìm kiếm cho phép người dùng tìm kiếm giáo viên theo tên hoặc mã số giáo viên. Người dùng chỉ cần nhập tên hoặc mã số vào ô tìm kiếm và hệ thống sẽ hiển thị danh sách các giáo viên tương ứng.

**Danh sách giáo viên:** Phần trung tâm của trang hiển thị danh sách giáo viên với các thông tin cơ bản bao gồm tên, mã số giáo viên, quyền truy cập và các hành động có thể thực hiện. Mỗi giáo viên được hiển thị dưới dạng một hàng trong bảng, với các cột:

**Name:** Tên của giáo viên.

**Teacher Code:** Mã số giáo viên.

**Permission Name:** Quyền truy cập của giáo viên (ví dụ: Giáo viên, Quản trị viên, Siêu quản trị viên).

**Action:** Các hành động có thể thực hiện, chẳng hạn như xem chi tiết, chỉnh sửa hoặc xóa giáo viên.

**Bộ lọc:** Người dùng có thể sử dụng các bộ lọc để lọc danh sách giáo viên theo quyền truy cập hoặc các tiêu chí khác. Các bộ lọc này giúp người dùng dễ dàng tìm kiếm và quản

lý giáo viên theo các nhóm cụ thể.

Nút Columns có thể lựa chọn các cột nào được hiển thị trong bảng

Nút thêm mới và quản lý khóa: Nút "Add New" cho phép người dùng thêm giáo viên mới vào hệ thống. Nút "Manage Blocked" cho phép quản lý các giáo viên bị khóa, chẳng hạn như mở khóa hoặc xem chi tiết lý do khóa.

Điều hướng phân trang: Ở phía dưới của danh sách giáo viên, hệ thống cung cấp các nút điều hướng phân trang để người dùng có thể di chuyển qua các trang khác nhau của danh sách giáo viên, giúp dễ dàng quản lý khi số lượng giáo viên lớn.

Menu điều hướng bên trái: Bao gồm các mục chính của hệ thống quản lý học tập như Tổng quan, Chấm điểm, Programs, Subjects, Rubric, Giáo viên, Sinh viên, Lớp, và Lớp môn học. Người dùng có thể dễ dàng chuyển đổi giữa các chức năng khác nhau của hệ thống thông qua menu này.

Khi chọn một vài giáo viên

Danh sách giáo viên				
Total 13 teachers Rows per page: 5				
	Name	Teacher Code	Permission Name	Action
<input type="checkbox"/>	 Nguyễn Bảo Ân annb@tvu.edu.vn	121214	Siêu Quản Trị Viên GVCV	<span>⋮</span>
<input checked="" type="checkbox"/>	 Nguyễn Nhứt Lam lamnn@tvu.edu.vn	100003	Giáo Viên GVGD	<span>⋮</span>
<input checked="" type="checkbox"/>	 Nguyễn Trần Diễm Hạnh diemhanh_tvu@tvu.edu.vn	100002	Giáo Viên GVGD	<span>⋮</span>
<input type="checkbox"/>	 Trần Văn Nam namtv@tvu.edu.vn	100004	Giáo Viên GVGD	<span>⋮</span>
<input checked="" type="checkbox"/>	 Trịnh Quốc Việt trinhtv@tvu.edu.vn	100005	Giáo Viên GVGD	<span>⋮</span>

Hình 4.22. Giao diện khi tương tác với bảng giáo viên.

Khi nhấn vào giáo viên bất kỳ sẽ hiển thị thanh chữ n Gang ở trên hiển thị số lượng giáo viên được chọn. Khi ta nhấn vào Download sẽ tả file excel chứa thông tin của giáo viên ta đã chọn. Hoặc nhấn Block sẽ chặn các giáo viên đã chọn.

## Khôi phục giáo viên đã chặn

The screenshot shows a list of blocked teachers with the following details:

Name	Teacher Code	Permission Name	Action
Trần Văn Nam namtv@tvu.edu.vn	100004	Giáo Viên GVGD	⋮
Trịnh Quốc Việt tqviettv@tvu.edu.vn	100005	Giáo Viên GVGD	⋮
Võ Thành C vothanhc@tvu.edu.vn	111111	Siêu Quản Trị Viên GVCV	⋮

Total 3 blocked teachers

Rows per page: 5

2 of 3 selected

Previous Next

Hình 4.23. Giao diện danh sách giáo viên đã chặn

Khi nhấp vào “Manage Blocked” sẽ mở danh sách chặn. Nhấn những giáo viên ta muốn bỏ chặn nhấn “Unlock”.

## Tạo giáo viên mới

The screenshot shows the 'Create New Teacher' interface with two main sections:

- Thêm giáo viên bằng file excel**: Includes 'Tải Mẫu CSV' (Load Sample CSV), 'Upload File' (Upload File), and 'Lưu file' (Save File).
- Nhập thông tin**: Includes fields for 'Tên giáo viên' (Teacher Name), 'Mã giáo viên' (Teacher Code), 'Email', and 'Quyền truy cập' (Access Rights). A dropdown menu for 'Quyền truy cập' is open, showing the following options:
 

Permission Name	Action
Siêu Quản Trị Viên GVCV	⋮
Giáo Viên GVGD	⋮

Cancel Add

Hình 4.24. Giao diện tạo giáo viên mới

Khi nhấp vào Add new thì giao diện tạo giáo viên mới sẽ xuất hiện. Có hai lựa chọn để tạo giáo viên mới đó là nhập giáo viên mới bằng excel, hoặc là nhập giáo viên thông qua form bên với.

#### 4.4.4. Giao diện quản lý sinh viên

The screenshot shows a user interface for managing students. On the left, there's a sidebar with navigation links: Tổng quan, Chấm điểm, Programs, Subjects, Rubric, Giáo viên, Sinh viên (which is selected and highlighted in blue), Lớp, and Lớp môn học. Below the sidebar is a user profile for 'Nguyễn Bảo An'. The main area is titled 'Danh sách sinh viên' and displays a table with 118 students. The table has columns for Mã sinh viên, Email, Tên sinh viên, Mã lớp, and Hành động. Each row contains a checkbox, the student's ID and email, their name, their class (e.g., DA20TTB), and a set of four icons for actions like edit, delete, and more. At the bottom of the table, it says '0 of 5 selected' and shows a page navigation from 1 to 24.

	Mã sinh viên	Email	Tên sinh viên	Mã lớp	Hành động
<input type="checkbox"/>	110120016	110120016@st.tvu.edu.vn	Huỳnh Phúc Đạt	DA20TTB	
<input type="checkbox"/>	110120013	110120013@st.tvu.edu.vn	Nguyễn Minh Đăng	DA20TTB	
<input type="checkbox"/>	110120037	110120037@st.tvu.edu.vn	Phan Vĩ Khang	DA20TTB	
<input type="checkbox"/>	110120019	110120019@st.tvu.edu.vn	Phạm Quyền Định	DA20TTB	
<input type="checkbox"/>	110120031	110120031@st.tvu.edu.vn	Trần Thái Hưng	DA20TTB	

Hình 4.25. Giao diện quản lý sinh viên.

Trang quản lý sinh viên là một phần quan trọng trong hệ thống quản lý học tập, cung cấp một giao diện trực quan và thân thiện cho việc quản lý thông tin sinh viên. Dưới đây là các thành phần và chức năng chính của trang này:

Thanh tìm kiếm: Nằm ở phía trên cùng của trang, thanh tìm kiếm cho phép người dùng tìm kiếm sinh viên theo tên, mã số sinh viên hoặc email. Người dùng chỉ cần nhập thông tin vào ô tìm kiếm và hệ thống sẽ hiển thị danh sách các sinh viên tương ứng.

Danh sách sinh viên: Phần trung tâm của trang hiển thị danh sách sinh viên với các thông tin cơ bản bao gồm mã số sinh viên, email, tên sinh viên, mã lớp và các hành động có thể thực hiện. Mỗi sinh viên được hiển thị dưới dạng một hàng trong bảng, với các cột:

Mã sinh viên: Mã số sinh viên.

Email: Địa chỉ email của sinh viên.

Tên sinh viên: Tên đầy đủ của sinh viên.

Mã lớp: Mã lớp mà sinh viên đang theo học.

Hành động: Các hành động có thể thực hiện, bao gồm xem chi tiết, chỉnh sửa và xóa sinh viên. Các biểu tượng hành động được hiển thị với các màu sắc khác nhau để dễ dàng nhận biết:

Bộ lọc và tùy chọn hiển thị: Người dùng có thể sử dụng các bộ lọc để lọc danh sách

sinh viên theo các tiêu chí khác nhau, hoặc thay đổi các cột hiển thị trong bảng bằng cách sử dụng nút "Columns".

Nút tạo mới và quản lý khóa: Nút "Tạo mới" cho phép người dùng thêm sinh viên mới vào hệ thống. Nút "Manage Blocked" cho phép quản lý các sinh viên bị khóa, chẳng hạn như mở khóa hoặc xem chi tiết lý do khóa.

Điều hướng phân trang: Ở phía dưới của danh sách sinh viên, hệ thống cung cấp các nút điều hướng phân trang để người dùng có thể di chuyển qua các trang khác nhau của danh sách sinh viên, giúp dễ dàng quản lý khi số lượng sinh viên lớn.

Menu điều hướng bên trái: Bao gồm các mục chính của hệ thống quản lý học tập như Tổng quan, Chấm điểm, Programs, Subjects, Rubric, Giáo viên, Sinh viên, Lớp và Lớp môn học. Người dùng có thể dễ dàng chuyển đổi giữa các chức năng khác nhau của hệ thống thông qua menu này.

#### 4.4.5. Giao diện quản lý lớp học

Mã lớp	Tên lớp	Tên giáo viên chủ nhiệm	Hành động
600000	Công nghệ Thông tin 2016 A	Nguyễn Trần Diễm Hạnh	
700000	Công nghệ Thông tin 2017 A	Võ Thành C	
800000	Công nghệ Thông tin 2018 A	Phan Thị Phương Nam	
900000	Công nghệ Thông tin 2018 B	Nguyễn Nhứt Lam	
100002	Công nghệ Thông tin 2019 B	Trịnh Quốc Việt	

Hình 4.26. Giao diện quản lý lớp học.

Trang quản lý các lớp trong hệ thống quản lý học tập cung cấp một giao diện trực quan và dễ sử dụng để quản lý thông tin về các lớp học. Trang này bao gồm các thành phần chính như sau:

Thanh tìm kiếm: Nằm ở phía trên cùng của trang, thanh tìm kiếm cho phép người dùng tìm kiếm lớp học theo tên lớp, mã lớp hoặc tên giáo viên. Người dùng chỉ cần nhập

thông tin vào ô tìm kiếm và hệ thống sẽ hiển thị danh sách các lớp học tương ứng.

Danh sách các lớp: Phần trung tâm của trang hiển thị danh sách các lớp với các thông tin cơ bản bao gồm mã lớp, tên lớp, tên giáo viên cố vấn và các hành động có thể thực hiện. Mỗi lớp học được hiển thị dưới dạng một hàng trong bảng, với các cột:

Mã lớp: Mã số của lớp học.

Tên lớp: Tên của lớp học.

Tên giáo viên cố vấn: Tên của giáo viên cố vấn của lớp học.

Hành động: Các hành động có thể thực hiện, bao gồm chỉnh sửa, tải xuống và xóa lớp học. Các biểu tượng hành động được hiển thị với các màu sắc khác nhau để dễ dàng nhận biết:

Bộ lọc và tùy chọn hiển thị: Người dùng có thể sử dụng các bộ lọc để lọc danh sách lớp theo các tiêu chí khác nhau, hoặc thay đổi các cột hiển thị trong bảng bằng cách sử dụng nút "Columns".

Nút tạo mới và quản lý khóa: Nút "Tạo mới" cho phép người dùng thêm lớp mới vào hệ thống. Nút "Manage Blocked" cho phép quản lý các lớp bị khóa, chẳng hạn như mở khóa hoặc xem chi tiết lý do khóa.

Điều hướng phân trang: Ở phía dưới của danh sách lớp học, hệ thống cung cấp các nút điều hướng phân trang để người dùng có thể di chuyển qua các trang khác nhau của danh sách lớp học, giúp dễ dàng quản lý khi số lượng lớp học lớn.

Menu điều hướng bên trái: Bao gồm các mục chính của hệ thống quản lý học tập như Tổng quan, Chấm điểm, Programs, Subjects, Rubric, Giáo viên, Sinh viên, Lớp và Lớp môn học. Người dùng có thể dễ dàng chuyển đổi giữa các chức năng khác nhau của hệ thống thông qua menu này.

#### 4.4.6. Giao diện quản lý khóa học

The screenshot displays the 'Danh sách khóa học' (List of Courses) page. The main content area shows four course entries, each with a large letter icon (K, C, T, L), course code, name, semester, teacher, and student count. Below each entry are three buttons: 'Sinh viên' (Student), 'Chỉnh sửa' (Edit), and 'Án môn học' (Grade). The left sidebar shows a navigation menu with 'SET' at the top, followed by 'Tổng quan', 'Chấm điểm', 'Programs', 'Subjects', 'Rubric', 'Giáo viên', 'Sinh viên', 'Lớp', and 'Lớp môn học' (selected). At the bottom left, there is a user profile for 'Nguyễn Bảo Ân' with the number '121214'.

Hình 4.27. Giao diện quản lý khóa học.

Trang quản lý lớp môn học là một phần quan trọng của hệ thống quản lý học tập, cung cấp giao diện trực quan để quản lý thông tin các lớp học và môn học. Dưới đây là các thành phần chính của trang này:

**Thanh tìm kiếm:** Nằm ở phía trên cùng của trang, thanh tìm kiếm cho phép người dùng tìm kiếm các lớp môn học theo tên khóa học, mã khóa học hoặc tên giáo viên. Người dùng chỉ cần nhập thông tin vào ô tìm kiếm và hệ thống sẽ hiển thị danh sách các lớp môn học tương ứng.

**Danh sách các khóa học:** Phần trung tâm của trang hiển thị danh sách các khóa học dưới dạng các ô thông tin. Mỗi ô thông tin đại diện cho một khóa học và chứa các thông tin chi tiết bao gồm:

**Mã khóa học:** Mã số của khóa học.

**Tên khóa học:** Tên của khóa học.

**Tên lớp:** Tên của lớp liên quan đến khóa học.

**Tên giáo viên:** Tên của giáo viên phụ trách khóa học.

**Học kỳ và năm học:** Thông tin về học kỳ và năm học của khóa học.

**Số sinh viên:** Số lượng sinh viên tham gia khóa học.

**Các nút hành động:** Mỗi ô thông tin của khóa học có các nút hành động để người

dùng thực hiện các chức năng khác nhau, bao gồm:

Sinh viên: Hiển thị danh sách sinh viên tham gia khóa học.

Chỉnh sửa: Chính sửa thông tin của khóa học.

Ẩn môn học: Ẩn khóa học khỏi danh sách hiển thị.

Nút tạo mới: Nút "Tạo mới" cho phép người dùng thêm một lớp môn học mới.

Điều hướng phân trang: Ở phía dưới của danh sách khóa học, hệ thống cung cấp các nút điều hướng phân trang để người dùng có thể di chuyển qua các trang khác nhau của danh sách khóa học, giúp dễ dàng quản lý khi số lượng khóa học lớn.

Menu điều hướng bên trái: Bao gồm các mục chính của hệ thống quản lý học tập như Tổng quan, Chấm điểm, Programs, Subjects, Rubric, Giáo viên, Sinh viên, Lớp và Lớp môn học. Người dùng có thể dễ dàng chuyển đổi giữa các chức năng khác nhau của hệ thống thông qua menu này.

## CHƯƠNG 5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

### 5.1. Kết luận

Trong quá trình xây dựng hệ thống trực quan hóa kết quả học tập sinh viên CNTT, chúng tôi đã đạt được một số kết quả quan trọng và đóng góp mới như sau:

Hệ thống trực quan hóa: tôi đã phát triển một hệ thống trực quan hóa kết quả học tập với giao diện thân thiện, dễ sử dụng, giúp người dùng có thể dễ dàng theo dõi và phân tích kết quả học tập của sinh viên. Các biểu đồ và đồ thị được thiết kế trực quan, cung cấp cái nhìn tổng quan về tiến trình học tập và hiệu suất của sinh viên.

Cơ sở dữ liệu đồng bộ: Hệ thống cơ sở dữ liệu được thiết kế và triển khai trên VPS với khả năng lưu trữ và truy xuất dữ liệu hiệu quả, đảm bảo tính chính xác và toàn vẹn của thông tin. Việc sử dụng MySQL cùng với ORM giúp dễ dàng quản lý và mở rộng hệ thống trong tương lai.

Quản lý người dùng: Chức năng quản lý người dùng bao gồm việc xác thực, phân quyền và quản lý thông tin người dùng đã được tích hợp, đảm bảo an toàn và bảo mật cho hệ thống.

Tích hợp và mở rộng: Hệ thống được xây dựng với khả năng tích hợp linh hoạt, cho phép kết nối với các hệ thống quản lý học tập khác và mở rộng chức năng theo nhu cầu của người dùng.

Những kết quả trên đã chứng minh tính khả thi và hiệu quả của hệ thống trong việc hỗ trợ công tác quản lý và phân tích kết quả học tập của sinh viên CNTT.

### 5.2. Hướng phát triển

Để tiếp tục nâng cao và mở rộng hệ thống, chúng tôi đề xuất một số hướng nghiên cứu và phát triển tiếp theo như sau:

Phân tích dữ liệu nâng cao: Nghiên cứu và phát triển các mô hình phân tích dữ liệu nâng cao, áp dụng các thuật toán học máy (machine learning) để dự đoán xu hướng học tập và phát hiện sớm các vấn đề tiềm ẩn.

Tích hợp trí tuệ nhân tạo (AI): Ứng dụng AI để cung cấp các gợi ý cá nhân hóa cho sinh viên, giúp họ cải thiện kết quả học tập dựa trên dữ liệu và phân tích cá nhân.

**Phát triển ứng dụng di động:** Xây dựng phiên bản ứng dụng di động của hệ thống để người dùng có thể truy cập và quản lý kết quả học tập mọi lúc, mọi nơi.

**Cải thiện giao diện người dùng:** Liên tục cải thiện giao diện người dùng, tăng cường trải nghiệm người dùng (UX/UI), đảm bảo hệ thống luôn thân thiện và dễ sử dụng.

**Mở rộng phạm vi ứng dụng:** Nghiên cứu và mở rộng hệ thống để áp dụng cho các ngành học khác ngoài CNTT, từ đó cung cấp một công cụ hữu ích cho nhiều lĩnh vực giáo dục khác nhau.

## **DANH MỤC TÀI LIỆU THAM KHẢO**

- [1] S. Mougiakou, D. Vinatsella, D. Sampson, Z. Papamitsiou, M. Giannakos and D. Ifenthaler, Educational Data Analytics for Teachers and School Leaders, 2023.
- [2] C. Yin and G.-J. Hwang, Roles and Strategies of Learning Analytics in the e-Publication Era, 2018.
- [3] I. F. Saida Ulfा, Predicting Factors that Influence Students' Learning Outcomes Using Learning Analytics in Online Learning Environment, International Journal of Emerging Technology in Learning, Kassel, Germany, 2021.
- [4] A. Johri, J. Lester, H. Rangwala and C. Klein, Learning Analytics in Higher Education, New York, 2018.
- [5] H. Sun, C. Humer, D. Bonetta and W. Binder, Efficient dynamic analysis for Node.js, 2018.
- [6] A. Mardan, Express.js Guide, 2014.
- [7] A. Fedosejev, React.js Essentials, 2015.
- [8] S. K. Vinicius M. Grippa, Learning MySQL, 2021.