NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN

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

Vĩnh Long, ngày ..... tháng …… năm ……

Giảng viên hướng dẫn

(Ký tên và ghi rõ họ tên)

NHẬN XÉT CỦA THÀNH VIÊN HỘI ĐỒNG

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

Vĩnh Long, ngày ….. tháng …… năm ……

Thành viên hội đồng

(Ký tên và ghi rõ họ tên)

LỜI CẢM ƠN

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

Dưới sự hướng dẫn của cô Phan Thị Phương Nam đã hoàn thành đề tài "Tìm hiểu về API và RESTful Architecture" . Để hoàn thành đồ án này và em xin chân thành gửi đến thêm một lời cảm ơn đến từ các thầy cô đã giảng dạy em trong suốt quá trình học tập , đặc biệt em xin gửi lời cảm ơn chân thành tới Cô giáo hướng dẫn Phan Thị Phương Nam đã tận tình , chu đáo hướng dẫn em thực hiện đồ án chuyên ngành này.

Mặc dù đã có nhiều cố gắng để thực hiện để tài một cách hoàn chỉnh nhất. do thời gian có hạn, trình độ hiểu biết và nhận thức còn chưa cao cho nên trong đồ án không thể tránh những thiếu sót, em rất mong nhận được sự đóng góp ý kiến của các thầy cô và bạn bè để em có thể hoàn thiện đồ án này tốt hơn.

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

Em xin chân thành cảm ơn!

MỤC LỤC

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG BIỂU

TÓM TẮT ĐỒ ÁN CHUYÊN NGÀNH

Dự án xây dựng hệ thống quản lý sinh viên theo kiến trúc RESTful, tách biệt Frontend (React) và Backend (Node.js/Express) với cơ sở dữ liệu MongoDB (Mongoose). Vấn đề nghiên cứu là tạo một nền tảng quản lý dữ liệu sinh viên, lớp, chuyên ngành, môn học và điểm số vừa dễ dùng, an toàn, mở rộng tốt, đồng thời hỗ trợ thống kê trực quan. Hướng tiếp cận dùng MERN stack, JWT cho xác thực, Multer cho upload ảnh, Chart.js cho biểu đồ, cùng mô hình MVC để rõ ràng và dễ bảo trì; API thiết kế theo resource (users, classes, majors, courses, grades) với các phương thức GET/POST/PUT/DELETE, có phân trang, tìm kiếm, lọc, và chuẩn hóa phản hồi JSON. Cách giải quyết tập trung vào giao diện responsive, form có validation, xử lý lỗi/notify thân thiện; backend với middleware bảo mật, validation dữ liệu, tính GPA tự động, chống trùng điểm theo (userId, courseId), và API thống kê tổng quan.

Áp dụng thực tế và kết quả đạt được: Dự án đã triển khai đầy đủ các nhóm chức năng quản lý: sinh viên, lớp học, chuyên ngành, môn học, điểm số; có trang tổng quan hiển thị thống kê bằng biểu đồ; có biểu mẫu nhập liệu kèm kiểm tra lỗi; xử lý điểm tự động tính GPA theo thang 4.0, xếp loại từ A+ tới F, chặn trùng điểm theo cặp sinh viên và môn học,và cơ chế đăng nhập bảo mật. Tổng cộng có hơn hai chục điểm cuối (endpoint) hoạt động ổn định, giao diện gồm nhiều trang chức năng (đăng nhập, quản lý, hồ sơ cá nhân, bảng điểm, thống kê), dữ liệu trả về theo định dạng chuẩn hóa giúp dễ tích hợp. API này có thể dùng lại cho trang web khác hoặc ứng dụng di động vì chỉ cần cấp quyền truy cập (token) và cho phép miền truy cập (CORS),ví dụ một ứng dụng ngoài có thể gọi “lấy danh sách sinh viên”, “bảng điểm của một sinh viên”, hoặc “thống kê tổng quan” để hiển thị theo nhu cầu của họ. Nhờ tuân thủ chuẩn REST và tổ chức mã nguồn theo mô hình rõ ràng, hệ thống dễ bảo trì, mở rộng, và đủ cơ sở để trình bày trước thầy cô về nguyên lý, cách thiết kế, cũng như minh chứng vận hành thực tế.

MỞ ĐẦU

1. Lý do chọn đề tài

Trong thời đại công nghệ số, hầu hết các hệ thống phần mềm hiện nay đều cần giao tiếp với nhau để trao đổi dữ liệu, chia sẻ chức năng hoặc tích hợp các dịch vụ từ bên ngoài. API (Application Programming Interface) chính là cầu nối cho phép các ứng dụng khác nhau nói chuyện với nhau một cách có tổ chức và an toàn. Trong số các kiến trúc thiết kế API, RESTful là tiêu chuẩn phổ biến nhất hiện nay, được sử dụng bởi các công ty công nghệ lớn như Google, Facebook, Amazon, Netflix và nhiều dịch vụ web khác. Tuy nhiên, hiểu biết sâu sắc về API và RESTful không phải là điều hiển nhiên nhiều lập trình viên thường chỉ biết cách dùng API mà không hiểu rõ các nguyên tắc thiết kế đằng sau. Vì vậy, em muốn chọn đề tài này không chỉ để học cách sử dụng, mà để nắm vững những nguyên lý cốt lõi, từ đó thiết kế và xây dựng API của riêng mình một cách chuyên nghiệp và chuẩn mực.

2. Mục đích nghiên cứu

Mục tiêu chính của đề tài là làm rõ hoàn toàn các khái niệm API và kiến trúc RESTful, để từ đó có thể áp dụng vào thực tế xây dựng dịch vụ web. Cụ thể hơn, em muốn hiểu rõ: một là API là gì, tại sao lại cần API, cách API hoạt động, hai là REST là gì, tại sao nó trở thành tiêu chuẩn phổ biến, ba là cách ánh xạ các thao tác cơ bản CRUD (Create, Read, Update, Delete) sang các phương thức HTTP (POST, GET, PUT, DELETE), bốn là cách sử dụng mã trạng thái HTTP (200 OK, 201 Created, 204 No Content, 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found, 500 Internal Server Error…) đúng ngữ nghĩa để giao tiếp trạng thái phản hồi với client, năm là cách chuẩn hóa dữ liệu trao đổi bằng định dạng JSON, sáu là các nguyên tắc thiết kế URL theo tài nguyên không phải hành động, bảy là cách xử lý phân trang, tìm kiếm, lọc và sắp xếp dữ liệu, tám là cấu trúc phản hồi nhất quán để dễ dàng tích hợp, chín là những vấn đề bảo mật cơ bản khi xây dựng API, cuối cùng là cách sử dụng và tiêu thụ API từ phía client để hiểu được luồng hoạt động từ đầu đến cuối.

3. Đối tượng nghiên cứu

Đối tượng chính của đề tài là các nguyên lý, quy tắc và thực hành trong thiết kế RESTful API cho dịch vụ web. Em sẽ tập trung vào việc phân tích cách thức API RESTful hoạt động, từ cách xây dựng các endpoint (điểm cuối) cho tới cách client

gọi các endpoint đó và xử lý phản hồi. Đối tượng cụ thể bao gồm: thiết kế tài nguyên và URL, ánh xạ HTTP methods với CRUD, sử dụng mã trạng thái HTTP, định dạng dữ liệu JSON, chuẩn hóa cấu trúc request/response, xử lý tham số truy vấn, cơ chế phân trang và lọc dữ liệu, xử lý lỗi nhất quán, bảo mật cơ bản (JWT, CORS), và quy trình triển khai một API thực tế.

4. Phạm vi nghiên cứu

Phạm vi của đề tài tập trung chủ yếu vào nền tảng lý thuyết về API và RESTful architecture, cùng với những ví dụ thực hành minh họa. Cụ thể, em sẽ trình bày: Các khái niệm cơ bản (API, REST, HTTP), Các nguyên tắc thiết kế RESTful (stateless, client-server separation, resource-based URLs), HTTP methods (GET, POST, PUT, PATCH, DELETE) và ánh xạ với CRUD, HTTP status codes phổ biến và cách sử dụng đúng, JSON format và cấu trúc response chuẩn hóa , Query parameters cho phân trang, tìm kiếm, lọc (page, limit, search, filter, sort), Ví dụ endpoint cụ thể (ví dụ: GET /api/users, POST /api/users, PUT /api/users/123, DELETE /api/users/123), Cách gọi API từ và xử lý response, Các vấn đề bảo mật cơ bản (HTTPS, JWT token, CORS), Xử lý lỗi nhất quán và ghi log. Tuy nhiên, phạm vi này KHÔNG bao gồm những chủ đề nâng cao như GraphQL (một thay thế cho REST), microservices architecture (kiến trúc dịch vụ nhỏ), API versioning quá chi tiết, caching strategy chuyên sâu, rate limiting nâng cao, các chuẩn bảo mật phức tạp như OAuth 2.0 chi tiết, hoặc tối ưu hiệu năng ở quy mô siêu lớn.

5. Phương pháp nghiên cứu

Trước hết, tiến hành nghiên cứu tài liệu liên quan đến API, RESTful architecture, giao thức HTTP, HTTP methods, HTTP status codes, định dạng dữ liệu JSON và các nguyên tắc thiết kế RESTful thông qua sách chuyên ngành và tài liệu kỹ thuật uy tín. Trên cơ sở đó, sinh viên phân tích và tổng hợp các kiến thức để xây dựng nền tảng lý thuyết cho đề tài.

Tiếp theo, áp dụng các nguyên lý RESTful để thiết kế và xây dựng API quản lý sinh viên theo mô hình Client – Server. Hệ thống được cài đặt bằng Node.js và Express, triển khai các chức năng CRUD thông qua các phương thức HTTP (GET, POST, PUT, DELETE), dữ liệu được quản lý bằng MongoDB.

Cuối cùng, hệ thống API được kiểm thử bằng công cụ Postman nhằm đánh giá tính đúng đắn, tính nhất quán của dữ liệu trả về, việc sử dụng mã trạng thái HTTP và mức độ đáp ứng yêu cầu của đề tài.

CHƯƠNG 1: TỔNG QUAN

1.1. Giới thiệu chung về API

API (Application Programming Interface) là một tập hợp các quy tắc và giao thức cho phép các ứng dụng phần mềm giao tiếp với nhau. Nói một cách đơn giản, API là một cửa sổ mà thông qua đó các ứng dụng có thể yêu cầu dữ liệu hoặc dịch vụ từ một ứng dụng khác. Thay vì phải chạm trực tiếp vào mã nguồn hay cơ sở dữ liệu, các nhà phát triển có thể gọi các hàm hoặc điểm cuối (endpoints) được cung cấp bởi API để lấy dữ liệu hoặc thực hiện các hành động cụ thể. Ví dụ, khi sử dụng ứng dụng dự báo thời tiết trên điện thoại, ứng dụng đó không lưu trữ tất cả dữ liệu thời tiết toàn bộ thế giới mà thay vào đó nó gọi một API từ một máy chủ thời tiết để lấy thông tin cần thiết. API đã trở thành một phần không thể thiếu trong phát triển phần mềm hiện đại vì nó cho phép các nhà phát triển tập trung vào các tính năng ứng dụng của họ thay vì phải xây dựng lại từ đầu những công cụ mà những người khác đã xây dựng.

1.2 Vấn đề cần giải quyết

Mặc dù API rất phổ biến, nhưng không phải lúc nào các nhà phát triển đều thiết kế API đúng theo các nguyên tắc.

Các vấn đề thường gặp bao gồm:

- Thiết kế không nhất quán: Các endpoint không tuân theo một chuẩn chung, khiến người sử dụng khó hiểu.
- HTTP status code: Trả về 200 OK cho tất cả trường hợp, thậm chí cả khi có lỗi.
- Response không chuẩn hóa: Mỗi endpoint trả về format JSON khác nhau.
- Thiếu tài liệu: API không được ghi chép rõ ràng.
- Không hỗ trợ phân trang/lọc: Endpoint trả về toàn bộ dữ liệu, gây ảnh hưởng đến hiệu suất.
- Bảo mật yếu: Không kiểm tra quyền truy cập hoặc không xác thực người dùng.

Để giải quyết những vấn đề này, em sử dụng RESTful architecture, một tiêu chuẩn thiết kế API được công nhân rộng rãi,

1.3 RESTful Architecture

REST (Representational State Transfer) là một phong cách thiết kế API được Roy Fielding đề xuất vào năm 2000. RESTful API tuân theo các nguyên tắc: sử dụng HTTP method phù hợp (GET, POST, PUT, DELETE) cho mỗi tác vụ, sử dụng URL để đại diện cho tài nguyên (resource), không phải hành động, sử dụng HTTP status code để thể hiện kết quả của yêu cầu, dữ liệu được truyền dưới định dạng JSON.

1.4 Phương pháp tiếp cận

Đề tài của em bắt đầu từ các khái niệm cơ bản về HTTP, URL, và JSON. Tiếp theo, em trình bày chi tiết các nguyên tắc của REST, bao gồm HTTP methods (GET, POST, PUT, DELETE), HTTP status codes, và cách thiết kế resource-based URLs. Em cũng trình bày các vấn đề thực tế như phân trang, tìm kiếm, lọc dữ liệu, và bảo mật (JWT authentication).

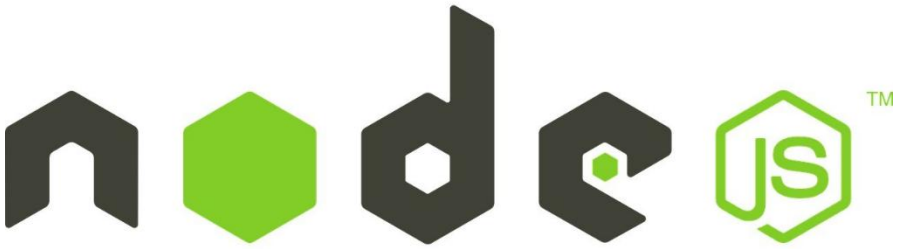
Phần thực hành, em xây dựng một hệ thống quản lý sinh viên hoàn chỉnh với backend sử dụng Express.js và Node.js, database MongoDB với Mongoose, frontend sử dụng React, và gọi API bằng Axios. Qua ví dụ thực tế này, các nguyên tắc RESTful được minh họa rõ ràng.

CHƯƠNG 2: NGHIÊN CỨU LÝ THUYẾT

2.1 Công nghệ backend

2.1.1 Tổng quan về Node.js

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



Hình 2.1 Tổng quan Node.js

Một số câu lệnh cơ bản trong Node.js:

- node <filename>: Thực thi chương trình từ tệp chỉ định.
- npm init: Tạo dự án mới và tệp package.json.
- npm install <packageName>: Cài đặt gói từ npm.
- npm install-g <packageName>: Cài đặt gói toàn cầu.
- require('<moduleName>*): Nhập mô-đun trong tệp Node.js.
- console.log(): In ra màn hình console.
- process.env: Truy cập biến môi trường.
- module.exports: Xuất mô-đun cho các tệp khác.
- dirname: Lấy đường dẫn thư mục hiện tại.
- fs.readFile(): Đọc dữ liệu từ tệp không đồng bộ.

Ưu điểm:

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

Nhược điểm:

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

Các ứng dụng của Node.js:

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

2.1.2 Tổng quan về Express.js

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

Hình 2.2 Tổng quan Express.js

Ưu điểm:

Đơn giản và dễ sử dụng: Cú pháp dễ hiểu giúp lập trình viên nhanh chóng triển khai các tính năng.

Hỗ trợ Middleware: Cung cấp hệ thống Middleware linh hoạt cho xác thực, ghi log, nén dữ liệu và xử lý lỗi.

Hiệu suất cao: Xây dựng trên Node.js, xử lý nhanh các yêu cầu web đồng thời và mở rộng tốt.

Khả năng mở rộng: Hỗ trợ nhiều kết nối đồng thời nhờ mô hình I/O không chặn và Event-driven.

Nhược điểm:

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

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

Các ứng dụng của Express.js

Ứng dụng web: Xây dựng các trang web và ứng dụng web động.

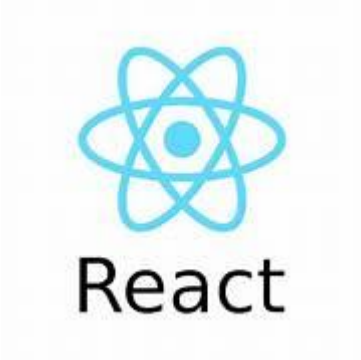
API RESTful: Tạo và quản lý các API cho giao tiếp dữ liệu.

Ứng dụng thời gian thực: Hỗ trợ các ứng dụng cần tính năng thời gian thực như chat hoặc thông báo.

2.2 Công nghệ Frontend

2.2.1 Tổng quan React.js

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



Hình 2.3 Tổng quan về React.js

Cơ chế hoạt động của React.js:

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

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

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

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

2.3.2 Tổng quan về Javascript

Javascript chính là một ngôn ngữ lập trình web rất phổ biến ngày nay. Javascript được tích hợp đồng thời nhúng vào HTML để hỗ trợ cho website trở nên sống động hơn. Chúng cũng đóng vai trò tương tự như một phần của website, cho phép Client-side Script từ người dùng tương tự máy chủ (Nodejs) để tạo ra những website động.

Hình 2.4 Tổng qua về Javascript

Cơ chế hoạt động của JavaScript:

JavaScript là ngôn ngữ kịch bản được thực thi chủ yếu trên trình duyệt nhằm tạo ra các trang web có tính tương tác cao. Khi trang web được tải, trình duyệt sẽ phân tích mã HTML để tạo cấu trúc DOM, sau đó thực thi mã JavaScript để thao tác, cập nhật và phản hồi các sự kiện xảy ra trên trang. JavaScript hoạt động theo cơ chế event-driven và single-thread, nghĩa là các tác vụ được xử lý tuần tự thông qua Event Loop, trong đó các sự kiện, callback và promise được đưa vào hàng đợi và lần lượt thực thi mà không làm gián đoạn giao diện người dùng. Nhờ cơ chế này, JavaScript có thể xử lý bất đồng bộ hiệu quả, đảm bảo trang web hoạt động mượt mà và phản hồi nhanh.

2.3.3 Tổng quan về MongoDB

MongoDB là một chương trình cơ sở dữ liệu mã nguồn mở được thiết kế theo kiểu hướng đối tượng trong đó các bảng được cấu trúc một cách linh hoạt cho phép các dữ liệu lưu trên bảng không cần phải tuân theo một dạng cấu trúc nhất định nào. Chính do cấu trúc linh hoạt này nên MongoDB có thể được dùng để lưu trữ các dữ liệu có cấu trúc phức tạp và đa dạng và không cố định (hay còn gọi là Big Data).

Hình 2.5 Tổng quan về MongoDB

Một số đặc điểm của MongoDB

- Kho lưu định hướng Document: Dữ liệu được lưu trong các tài liệu kiểu JSON.
- Lập chỉ mục trên bất kỳ thuộc tính nào.
- Các truy vấn đa dạng.
- Cập nhật nhanh hơn.

Hoạt động của MongoDB

Nguyên tắc hoạt động của MongoDB là dưới một tiến trình dịch vụ ngầm và mở một cổng (mặc định là cổng 27017), để có thể tiếp nhận các yêu cầu truy vấn, thao tác; sau đó tiến hành xử lý.

Mỗi bản ghi của MongoDB (document) được gắn một trường có tên "_id" nhằm xác định tính duy nhất của bản ghi. Có thể hiểu id này như tên gọi của một bản ghi và dùng phân biệt chúng với các bản ghi khác. Đồng thời, nó còn được sử dụng cho mục đích truy vấn hoặc tìm kiếm thông tin. Trường dữ liệu "_id" được tự động đánh chỉ mục (index) để đảm bảo tốc độ truy vấn đạt hiệu suất tối ưu.

Mỗi truy vấn dữ liệu đều được ghi đệm lên bộ nhớ RAM nên các truy vấn sau đó sẽ diễn ra nhanh hơn. Bởi nó không cần đọc dữ liệu từ ổ cứng.

Khi thực hiện thêm, xóa hay sửa bản ghi thì MongoDB đều mất 60s để ghi các dữ liệu được thay đổi từ RAM xuống ổ cứng. Điều này nhằm mục đích đảm bảo hiệu suất mặc định của chương trình.

2.3.4 Tổng quan về RESTful API

Khái niệm

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

Hình 2.6 Tổng quan về RESTful API

Nguyên tắc cơ bản:

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

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

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

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

Cấu trúc của RESTful API:

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

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

Định dạng Dữ liệu (Data Format): Dữ liệu trao đổi giữa client và server thường được định dạng bằng JSON (JavaScript Object Notation) hoặc XML (eXtensible

Markup Language). JSON là định dạng phổ biến nhất trong RESTful API nhờ tính đơn giản và dễ doc.

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

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

Thành phần của RESTful API:

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

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

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

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

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

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

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

URL dựa trên tài nguyên

Là cách thiết kế URL trong kiến trúc RESTful, trong đó mỗi URL được xây dựng dựa trên tài nguyên mà hệ thống quản lý, thay vì dựa trên hành động. Mỗi tài nguyên được biểu diễn bằng một danh từ số nhiều như students, classes, subjects,… và các thao tác trên tài nguyên sẽ được xác định thông qua các HTTP methods như GET, POST, PUT, DELETE. Cách tổ chức này giúp API trở nên rõ ràng, nhất quán, dễ mở rộng và tuân thủ chuẩn RESTful.

Ví dụ:

Bảng 2.1: Thiết kế URLs dựa trên tài nguyên trong RESTful API.

Hành động	HTTP Method	URL	Ý nghĩa
Lấy danh sách users	GET	/api/users	Lấy tất cả users (collection)
Lấy user cụ thể	GET	/api/users/123	Lấy user cụ thể có ID = 123
Tạo user mới	POST	/api/users	Tạo user mới trong collection
Cập nhật toàn bộ user	PUT	/api/users/123	Thay thế hoàn toàn user 123
Cập nhật một phần	PATCH	/api/users/123	Cập nhật chỉ một số trường của user 123
Xóa	DELETE	/api/users/123	Xóa user 123

CHƯƠNG 3: HIỆN THỰC HÓA NGHIÊN CỨU

3.1 Mô tả bài toán

Trong môi trường giáo dục đại học, việc quản lý thông tin sinh viên, môn học, lớp học và kết quả học tập đòi hỏi hệ thống tin cậy, dễ sử dụng và dễ tích hợp. Nhiều đơn vị vẫn quản lý rời rạc hoặc dùng sổ sách, gây khó khăn khi tra cứu, tổng hợp báo cáo và đảm bảo tính toàn vẹn dữ liệu. Dự án này nhằm xây dựng một dịch vụ quản lý sinh viên dưới dạng giao diện lập trình để phục vụ cho việc quản lý tập trung.

Vấn đề cần giải quyết

Hệ thống hiện tại thiếu chuẩn hóa dữ liệu, khó mở rộng và không có cơ chế xác thực phân quyền chặt chẽ. Cụ thể cần giải quyết: lưu trữ thông tin sinh viên, chuyên ngành, môn học và lớp học đồng nhất; ghi nhận và truy vấn điểm theo học kỳ, ngăn chặn trùng lặp và sai lệch dữ liệu, cung cấp API để giao tiếp với giao diện người dùng và hệ thống khác, và đảm bảo an toàn, sao lưu và khôi phục dữ liệu khi cần.

Mục tiêu nghiên cứu

Mục tiêu chính: thiết kế và triển khai hệ thống API hoàn chỉnh cho QLSV gồm các chức năng cơ bản (quản lý người dùng, chuyên ngành, môn học, lớp, điểm), kèm theo cơ chế xác thực, phân quyền đơn giản, khả năng phân trang lọc, và báo cáo tổng hợp. Yêu cầu đầu ra: mã nguồn có cấu trúc, mô tả dữ liệu, hồ sơ API, tập kiểm thử cơ bản.

Vai trò của Admin:

Vai trò của admin trong hệ thống Quản lý Sinh viên là người đảm nhiệm việc vận hành, bảo trì và quản trị dữ liệu toàn diện của hệ, bao gồm cả trách nhiệm về phân quyền, duy trì tính chính xác của thông tin và đảm bảo an toàn hoạt động. Admin có quyền quản lý các thực thể cốt lõi như tài khoản người dùng (sinh viên, giảng viên), lớp học, môn học, ngành và điểm, thực hiện các thao tác tạo/sửa/xóa khi cần, đồng thời chịu trách nhiệm phê duyệt và nhập liệu hàng loạt (import/export) để đảm bảo dữ liệu đồng bộ.

Về vận hành, admin theo dõi dashboard tổng quan (KPI), xử lý các job nặng như import file hoặc tính toán lại điểm qua hàng đợi công việc, và can thiệp khi phát

hiện xung đột dữ liệu hoặc lỗi đồng bộ; tất cả thay đổi quan trọng phải được ghi lại trong audit log để truy xuất.

Tóm tắt cho các chức năng:

Auth: Đăng nhập/đăng ký, quản lý token, xác thực người dùng.

Profile: Hiển thị và chỉnh sửa thông tin cá nhân, avatar.

Dashboard: Tổng quan KPI, thống kê tổng số SV/lớp/môn và GPA trung bình.

Admin: Giao diện quản trị để quản lý người dùng, cấu hình hệ thống và audit.

Class: Quản lý lớp học (CRUD, danh sách dropdown).

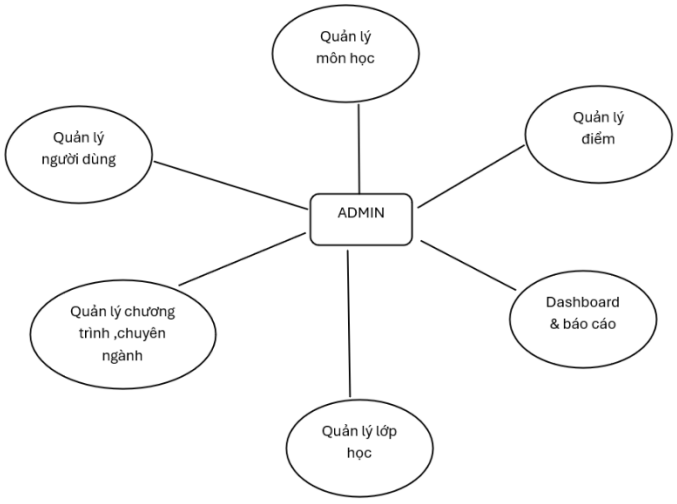
Course: Quản lý môn học (CRUD, tìm kiếm, dropdown).

Major: Quản lý chuyên ngành (CRUD, liên kết lớp).

Grade: Quản lý điểm, bảng điểm cá nhân, import/export và workflow chấm.

3.2 Yêu cầu chức năng

API hệ thống QLSV được thiết kế theo phong cách RESTful để cung cấp các thao tác CRUD cho các thực thể chính (Xác thực, Người dùng, Lớp học, Chuyên ngành, Khóa học, Điểm số, Bảng điều khiển, Quản trị viên), với các endpoint điển hình như /api/auth/, /api/users, /api/classes/, /api/majors/, /api/courses/, /api/grades/, /api/dashboard/summary, namespace quản trị /api/admin/ cho import/job/role/logs. Mọi route nhạy cảm được bảo vệ bằng JWT và middleware phân quyền responses chuẩn hóa theo envelope { success, message, data, pagination? } và sử dụng HTTP status phù hợp.



Hình 3.1 Sơ đồ use case Admin

Bảng 3.1 Mô tả chức năng

STT	Tên	Ý nghĩa
1	Tổng quan	Hiển thị KPI và thống kê tổng quan hệ thống (số SV, số lớp, số môn, GPA trung bình), cảnh báo và truy cập nhanh.
2	Quản lý sinh viên	CRUD sinh viên, tìm kiếm/lọc, import/export danh sách, upload ảnh, xem/ chỉnh sửa thông tin cá nhân.
3	Quản lý lớp học	Tạo/sửa/xóa lớp, quản lý danh sách lớp, liên kết lớp chuyên ngành, cung cấp dữ liệu cho dropdown.
4	Quản lý ngành học	CRUD chuyên ngành, quản lý mã ngành, thống kê số lớp/ sinh viên theo ngành.
5	Quản lý môn học	CRUD môn học, phiên bản môn, tìm kiếm, gán giảng viên, thông tin tín chỉ/học kỳ và dropdown.
6	Quản lý điểm	Tạo/cập nhật/xóa điểm, bảng điểm sinh viên, tính GPA, import/export điểm, workflow chấm và audit lịch sử.

3.3 Thiết kế API

3.3.1 Định tuyến API trong Express.js

Định tuyến API trong Express.js nên ngắn gọn và module hóa, tách mỗi resource thành một express.Router() trong routes/ (ví dụ users, courses, grades) và mount chúng trong routes/index.js. Cấu hình middleware chung, còn middleware chuyên biệt (xác thực, phân quyền, validate) gắn ở cấp router/route để kiểm soát truy cập và hợp lệ hóa dữ liệu. Dùng quy ước RESTful (GET, POST, PUT/PATCH, DELETE), hỗ trợ query params cho lọc/pagination, và giữ response envelope chuẩn ({ success, data, error, meta }). Cuối cùng, dùng global error handler để trả lỗi chuẩn, không lộ stack trace, và cân nhắc nested routes (ví dụ /courses/:courseId/grades) khi cần biểu diễn quan hệ giữa tài nguyên.

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

```
const app = express();
app.use(express.json());
app.use(cors());

const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);
app.use('/uploads', express.static(path.join(__dirname, 'uploads')));

mongoose.connect(process.env.MONGODB_URI)
  .then(() => {
    const PORT = process.env.PORT || 7000;
    app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
  })
  .catch(err => console.error('DB connection error:', err));

app.use('/api', userRoute);
app.use('/api/auth', authRoute);
app.use('/api/class', classRoute);
app.use('/api/major', majorRoute);
app.use('/api/course', courseRoute);
app.use('/api/grade', gradeRoute)
app.use('/api/dashboard', dashboardRoute);
```

Hình 3.2 Định tuyến API trong Express.js

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

Các định tuyến:

- **/api (user):** chịu trách nhiệm CRUD tài khoản người dùng và upload avatar. Các route xử lý tạo, đọc, cập nhật, xóa người dùng; thường kèm validate, kiểm tra quyền và xử lý multipart cho upload.

- **/api/auth:** xử lý xác thực (đăng ký, đăng nhập, refresh token, logout). Đây là khu vực nhạy cảm về bảo mật cần validate chặt, rate limiting và trả token an toàn.

- **/api/class**: quản lý thông tin lớp học và danh sách sinh viên trong lớp. Thực hiện các thao tác CRUD lớp, hỗ trợ filter/pagination cho danh sách lớn và kiểm soát truy cập theo vai trò.

- **/api/major:** quản lý chuyên ngành (mã, tên, mô tả). Dùng làm tham chiếu cho course/class và hỗ trợ các thao tác dữ liệu danh mục.
- **/api/course:** quản lý học phần tạo/sửa/xóa/tra cứu môn học, hỗ trợ lọc theo chuyên ngành, sắp xếp và phân trang.
- **/api/grade:** quản lý điểm endpoints tạo bản ghi điểm, lấy danh sách (phân trang), lấy bảng điểm sinh viên, và CRUD theo id. Các thao tác ghi/sửa thường yêu cầu phân quyền (instructor/admin).
- **/api/dashboard:** cung cấp dữ liệu tổng hợp và báo cáo (aggregation). Kết quả trả thường là số liệu đã tính toán và có thể cần cache/giới hạn truy vấn.

Một số lưu ý

Luôn kiểm tra và xác thực dữ liệu đầu vào (body, params, query) trước khi xử lý; dùng bộ xử lý lỗi toàn cục để trả về cấu trúc lỗi thống nhất và không để lộ chi tiết nội bộ; khởi động ứng dụng chỉ sau khi kết nối cơ sở dữ liệu thành công. Áp dụng cơ chế xác thực và phân quyền cho các đường dẫn nhạy cảm, giới hạn tần suất truy cập cho các endpoint công khai, và ghi log có cấu trúc kèm mã yêu cầu để dễ truy vết. Với chức năng tải lên tệp, kiểm tra loại tệp và kích thước, lưu trữ an toàn và phục vụ tệp qua đường dẫn tĩnh; theo dõi hiệu năng và cảnh báo bằng hệ thống giám sát.

Định tuyến trong Router

Tách mỗi miền nghiệp vụ thành một router riêng để tách trách nhiệm rõ ràng; mount từng router vào một đường dẫn gốc có ý nghĩa (ví dụ /api, /api/grade, /api/auth). Gắn middleware ở cấp router hoặc route khi cần (xác thực, kiểm tra dữ liệu, xử lý upload), và tránh lồng route quá sâu; nếu cần phân cấp, dùng router phụ và truyền tham số đường dẫn để controller xử lý. Cấu trúc này giúp dễ bảo trì, dễ test và mở rộng module mới mà không làm ảnh hưởng phần còn lại.

Tổ chức Endpoint

Đặt tên tài nguyên theo danh từ số nhiều và sử dụng phương thức tương ứng cho hành vi (lấy, tạo, cập nhật, xóa). Dùng tham số đường dẫn để chỉ tài nguyên cụ thể và tham số truy vấn cho lọc, sắp xếp, phân trang. Trả kết quả theo mẫu thống nhất gồm trạng thái, dữ liệu, lỗi và thông tin phụ (ví dụ phân trang) để client xử lý nhất

quán. Tách các hành động đặc thù ra đường dẫn riêng (ví dụ bảng điểm, xuất báo cáo) để dễ đọc và bảo trì.

Mở rộng ứng dụng

Thiết kế theo lớp: router , điều khiển , dịch vụ , truy xuất dữ liệu, để tách rõ trách nhiệm và dễ viết kiểm thử đơn vị. Gom và quản lý tất cả router ở một chỗ để mount dễ dàng; đưa phiên bản hóa giao diện lập trình vào thiết kế để tránh phá vỡ client khi nâng cấp. Khi cần mở rộng quy mô, cân nhắc tách các bối cảnh nghiệp vụ thành dịch vụ độc lập, bổ sung bộ nhớ đệm cho các truy vấn đọc nhiều, thiết lập pipeline tự động triển khai và hệ thống giám sát/alert để vận hành ổn định.

3.3.2 Định tuyến API người dùng (user)

```
import express from 'express';
import { create, getAllUsers, getUserById, update, deleteUser } from '../controller/userController.js';
import upload from '../middleware/uploadMiddleware.js';
import { uploadAvatar } from '../controller/userController.js';

const route = express.Router();

route.post('/user', create);
route.get('/users', getAllUsers);
route.get('/user/:id', getUserById);
route.put('/update/user/:id', update);
route.delete('/delete/user/:id', deleteUser);

// Upload avatar
route.post('/user/:id/avatar', upload.single('avatar'), uploadAvatar);

export default route;
```

Hình 3.3 Định tuyến API người dùng

Middleware:

Định tuyến và tổ chức endpoint: Tách mỗi miền nghiệp vụ thành express.Router() riêng và mount dưới base path rõ ràng (ví dụ /api, /api/grade, /api/auth). Tuân theo chuẩn RESTful: dùng danh từ số nhiều cho resource, HTTP verbs phù hợp, path params cho tài nguyên (/:id) và query params cho lọc/pagination. Tách các action đặc thù để rõ mục đích và dễ bảo trì; gom routers vào một file index để mount chung.

Mở rộng & vận hành production: Thiết kế theo lớpvà dùng versioning (/api/v1) để tránh breaking changes. Khi mở rộng, cân nhắc tách bounded contexts thành dịch

vụ riêng, dùng cache cho read-heavy endpoints, chuyển lưu file lên object storage (S3) thay vì lưu local, và thiết lập CI/CD, logging có cấu trúc, tracing và monitoring (metrics/alerts) để vận hành an toàn và dễ mở rộng.

Các định tuyến:

- POST /api/user : Tạo tài khoản (create)
- GET /api/users: Lấy danh sách người dùng (list)
- GET /api/user/:id: Lấy thông tin chi tiết người dùng theo id
- PUT /api/update/user/:id: Cập nhật thông tin người dùng theo id
- DELETE /api/delete/user/:id: Xóa người dùng theo id
- POST /api/user/:id/avatar : Upload ảnh đại diện (middleware

3.3.3 Định tuyến API xác thực (auth)

```
import express from "express";
import { register, login, verifyToken, getCurrentUser, logout } from "../controller/authController.js";
import { protectRoute } from "../middleware/authMiddleware.js";

const router = express.Router();

// Public routes
router.post("/register", register);
router.post("/login", login);
router.post("/verify-token", verifyToken);

// Protected routes
router.get("/me", protectRoute, getCurrentUser);
router.post("/logout", protectRoute, logout);

export default router;
```

Hình 3.4 Định tuyến API xác thực (auth)

Các định tuyến:

- POST /api/auth/register : đăng ký tài khoản
- POST /api/auth/login : đăng nhập, trả token
- POST /api/auth/verify-token : kiểm tra/verify token

- GET /api/auth/me: lấy thông tin người dùng hiện tại (protected bằng protectRoute)
- POST /api/auth/logout : đăng xuất (protected bằng protectRoute)

3.4.4 Định tuyến API lớp học (class)

```
import express from "express";
import {
    createClass,
    getAllClasses,
    getClassById,
    updateClass,
    deleteClass,
    getAllClassesForDropdown
} from "../controller/classController.js";

const router = express.Router();

// Public routes (không cần auth)
router.get("/all", getAllClasses);
router.get("/dropdown", getAllClassesForDropdown);
router.get("/:id", getClassById);

// Các routes cần auth (có thể thêm sau)
router.post("/", createClass);
router.put("/:id", updateClass);
router.delete("/:id", deleteClass);

export default router;
```

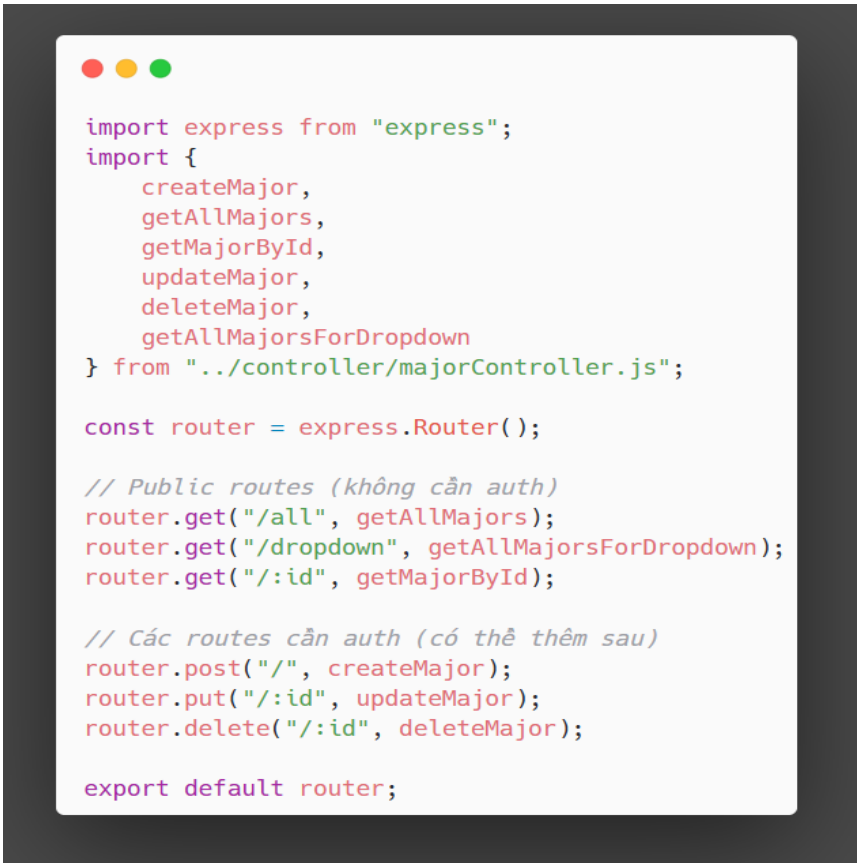
Hình 3.5 Định tuyến API lớp học

Các định tuyến

- GET /api/class/all: Lấy tất cả lớp (dùng cho danh sách, có thể phân trang)
- GET /api/class/dropdown: Lấy dữ liệu rút gọn cho dropdown
- GET /api/class/:id: Lấy chi tiết một lớp theo id
- POST /api/class: Tạo lớp mới
- PUT /api/class/:id: Cập nhật lớp theo id
- DELETE /api/class/:id: Xóa lớp theo id

Các route public (GET) có thể không cần auth, các route thay đổi dữ liệu (POST/PUT/DELETE) nên bảo vệ bằng middleware xác thực.

3.3.4 Định tuyến API chuyên ngành (major)



```
import express from "express";
import {
    createMajor,
    getAllMajors,
    getMajorById,
    updateMajor,
    deleteMajor,
    getAllMajorsForDropdown
} from "../controller/majorController.js";

const router = express.Router();

// Public routes (không cần auth)
router.get("/all", getAllMajors);
router.get("/dropdown", getAllMajorsForDropdown);
router.get("/:id", getMajorById);

// Các routes cần auth (có thể thêm sau)
router.post("/", createMajor);
router.put("/:id", updateMajor);
router.delete("/:id", deleteMajor);

export default router;
```

Hình 3.6 Định tuyến API Chuyên ngành

Các định tuyến:

- GET /api/major/all : Lấy tất cả chuyên ngành
- GET /api/major/dropdown: Lấy danh sách rút gọn cho dropdown
- GET /api/major/:id: Lấy chi tiết chuyên ngành theo id
- POST /api/major : Tạo chuyên ngành mới
- PUT /api/major/:id : Cập nhật chuyên ngành theo id
- DELETE /api/major/:id : Xóa chuyên ngành theo id

3.3.5 Định tuyến API môn học (cource)

```
import express from "express";
import {
    createCourse, getAllCourses, getCourseById, updateCourse, deleteCourse,getAllCoursesForDropdown
} from "../controller/courseController.js";
import protectRoute from "../middleware/authMiddleware.js";
const router = express.Router();
// POST Tạo môn học mới
router.post("/", protectRoute, createCourse);
// GET  Lấy tất cả môn học (phân trang + tìm kiếm)
router.get("/all", getAllCourses);
// GET Lấy tất cả môn học cho dropdown
router.get("/dropdown", getAllCoursesForDropdown);
// GET  Lấy chi tiết môn học
router.get("/:id", getCourseById);
// PUT  Cập nhật môn học
router.put("/:id", protectRoute, updateCourse);
// DELETE - Xóa môn học
router.delete("/:id", protectRoute, deleteCourse);
export default router;
```

Hình 3.7 Định tuyến API môn học

Các định tuyến:

- POST /api/course/ : tạo môn học
- GET /api/course/all : lấy tất cả môn học (phân trang/tìm kiếm)
- GET /api/course/dropdown : lấy cho dropdown
- GET /api/course/:id : lấy chi tiết môn học
- PUT /api/course/:id : cập nhật môn học
- DELETE /api/course/:id : xóa môn học

3.3.5 Định tuyến API điểm (grades)

```
import express from "express";
import {
    createGrade, getAllGrades, getGradeById, updateGrade, deleteGrade,getStudentTranscript
} from "../controller/gradeController.js";

const router = express.Router();
// POST - Tạo điểm mới
router.post("/", createGrade);
// GET - Lấy tất cả điểm (phân trang)
router.get("/all", getAllGrades);
// GET - Lấy bảng điểm của 1 sinh viên
router.get("/transcript/:userId", getStudentTranscript);
// GET - Lấy chi tiết 1 điểm
router.get("/:id", getGradeById);
// PUT - Cập nhật điểm
router.put("/:id", updateGrade);
// DELETE - Xóa điểm
router.delete("/:id", deleteGrade);
export default router;
```

Hình 3.8 Định tuyến API điểm

Các định tuyến:

- POST /api/grade/ : tạo bản ghi điểm
- GET /api/grade/all : lấy tất cả điểm (phân trang)
- GET /api/grade/transcript/:userId : lấy bảng điểm (transcript) của sinh viên
- GET /api/grade/:id : lấy chi tiết điểm
- PUT /api/grade/:id : cập nhật điểm
- DELETE /api/grade/:id : xóa điểm

3.3.6 Định tuyến API tổng quan (Dashboarh)

```
import express from 'express';
import { getDashboardSummary } from '../controller/dashboardController.js';
const router = express.Router();

router.get('/summary', getDashboardSummary);

export default router;
```

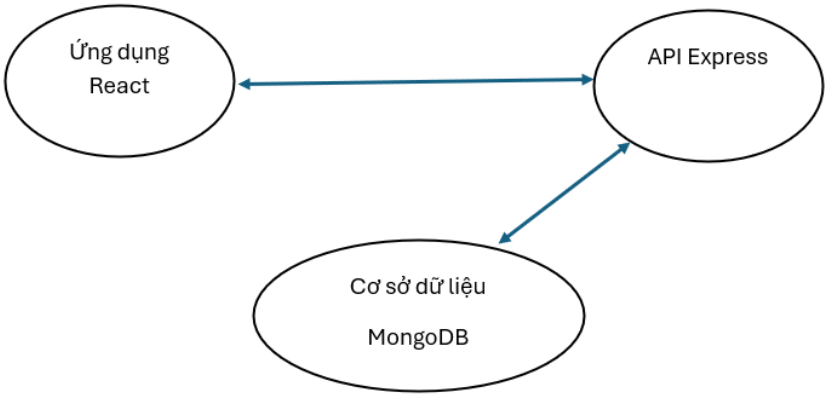
Hình 3.9 Định tuyến API tổng quan

Các định tuyến

- GET /api/dashboard/summary : lấy số liệu tóm tắt/ báo cáo

Định tuyến cho dashboard được triển khai trong dashboardRoute.js với endpoint chính là GET /api/dashboard/summary. Đây là route chỉ đọc (HTTP GET) trả về các số liệu tổng quan của hệ thống Route được xử lý bởi nơi tổng hợp dữ liệu từ nhiều model (User, Class, Course, Grade) và trả về JSON có cấu trúc rõ ràng (ví dụ: { users, classes, courses, avgGrade }). Vì thông tin tổng quan nhạy cảm nên cần bảo vệ route bằng middleware xác thực (protectRoute) để chỉ cho phép người dùng đã đăng nhập (hoặc role admin) truy cập; đồng thời nên có cơ chế cache (ví dụ Redis) để giảm tải cho các truy vấn tổng hợp tốn tài nguyên. Ngoài ra, controller phải xử lý lỗi và trả mã HTTP phù hợp (200 cho thành công, 401/403 nếu chưa xác thực/không đủ quyền, 500 cho lỗi server).

3.4 Kiến trúc tổng thể



Hình 3.10 Kiến trúc tổng thể

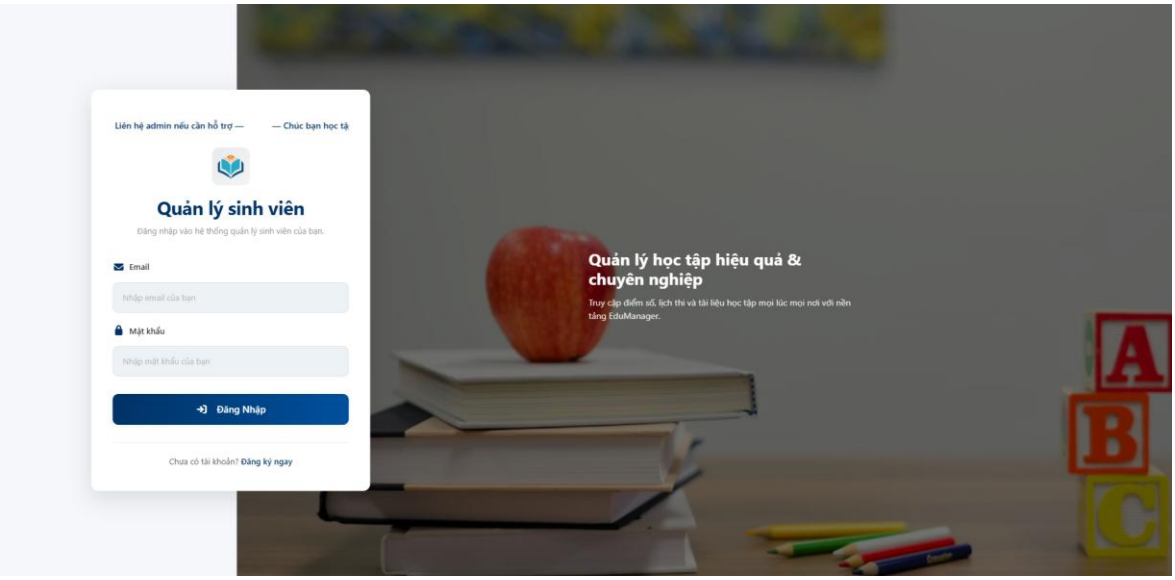
Ứng dụng gồm hai lớp chính: frontend là ứng dụng React chạy trên cổng 3005 và backend là API Express chạy trên cổng 8000, hai lớp giao tiếp qua HTTP/HTTPS (REST/JSON). React chịu trách nhiệm giao diện người dùng và gọi các endpoint của API để lấy/ghi dữ liệu; API Express xử lý nghiệp vụ, xác thực (JWT/session), xác thực quyền, kiểm tra dữ liệu, tải tệp và thực thi các truy vấn đến cơ sở dữ liệu MongoDB.

MongoDB lưu trữ dữ liệu chính và được truy cập bởi các controller/service trên backend. Ở môi trường thực tế, giao tiếp phải dùng HTTPS, backend nên bật CORS có cấu hình, dùng middleware bảo mật (helmet, rate-limit), và bảo vệ các route nhạy cảm bằng middleware xác thực, tệp tải lên tốt nhất lưu trữ trên object storage (S3) hoặc qua CDN thay vì chỉ lưu local. Để đáp ứng tải lớn, kiến trúc có thể đặt reverse proxy (nginx) hoặc load balancer phía trước, scale nhiều instance backend, dùng connection pool cho MongoDB, và cache các truy vấn tổng hợp (Redis). Cuối cùng cần giám sát, logging tập trung và biến môi trường cho cấu hình (port, DB URI, secrets) để đảm bảo an toàn và vận hành dễ dàng.

CHƯƠNG 4: KẾT QUẢ NGHIÊN CỨU

4.1 Kết quả thực nghiệm

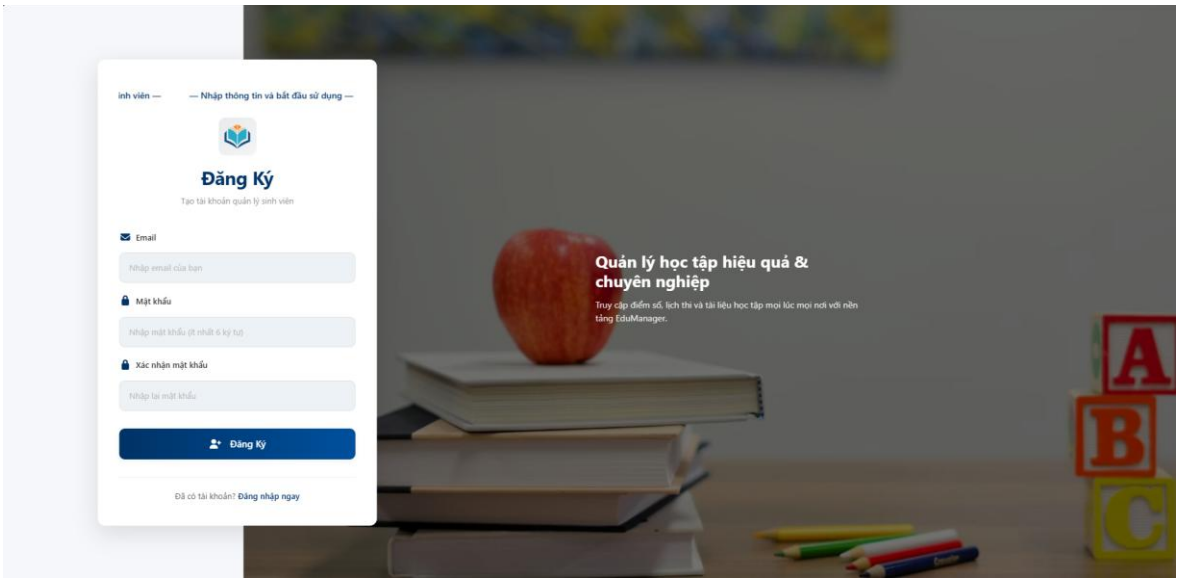
4.1.1 Chức năng đăng nhập hệ thống



Hình 4.1 Giao diện đăng nhập

Trang đăng nhập đóng vai trò cầu nối giữa người dùng và API**,** về triển khai form thực hiện kiểm tra đầu vào phía client (validate email, độ dài mật khẩu), hiển thị trạng thái loading và lỗi rõ ràng, sau đó gửi yêu cầu POST tới /api/auth/logi**n,** server trả về JWT (và có thể refresh token) — trên client nên lưu token an toàn **,**không lưu mật khẩu thô**,** cần hỗ trợ các tiện ích UX như hiện/ẩn mật khẩu, thông báo lỗi chi tiết nhưng không tiết lộ thông tin bảo mật, và tùy chọn “ghi nhớ đăng nhập” nếu dùng refresh token**,** về bảo mật, áp dụng HTTPS, rate limiting, khoá tạm thời sau nhiều lần sai mật khẩu, log sự kiện đăng nhập và (tuỳ yêu cầu) bổ sung xác thực hai lớp để tăng độ an toàn.

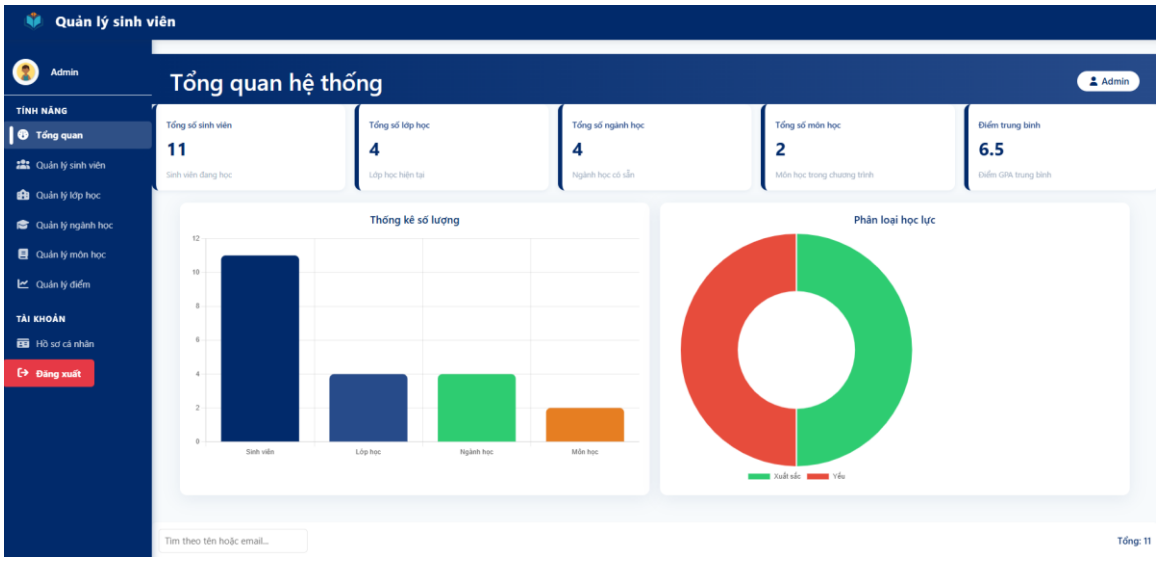
Đây là giao diện dành cho trang đăng ký:



Hình 4.2 Giao diện đăng ký

Giao diện trang đăng ký của QLSV ngắn gọn và trực quan: một form gồm các trường cần thiết (email, mật khẩu, xác nhận mật khẩu, tùy chọn thông tin như lớp), mỗi trường có label và placeholder rõ ràng cùng kiểm tra hợp lệ tức thì ở phía client, cung cấp nút hiện ẩn mật khẩu, chỉ báo độ mạnh mật khẩu và helper text cho yêu cầu định dạng, khi gửi hiển thị trạng thái loading và vô hiệu hóa nút để tránh gửi trùng, gửi dữ liệu an toàn qua HTTPS tới API POST /api/auth/register, server thực hiện validate, hash mật khẩu và trả thông báo phù hợp.

4.1.2 Chức năng tổng quan hệ thống



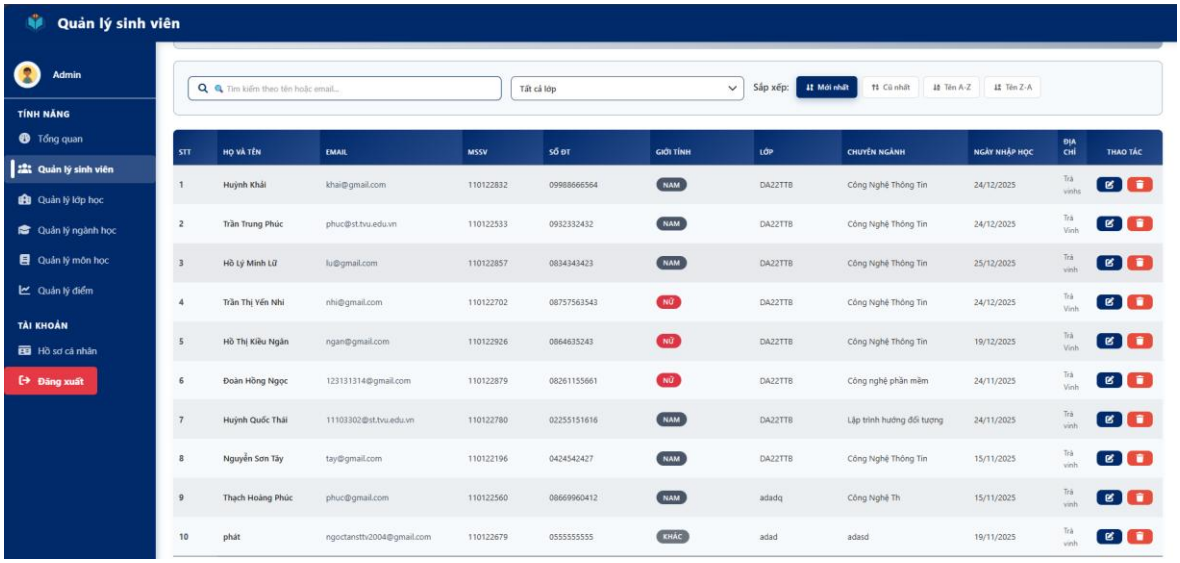
Hình 4.3 Giao diện tổng quan hệ thống

Trang tổng quan (dashboard) là màn hình chính mà người dùng nhìn thấy ngay sau khi đăng nhập vào hệ thống QLSV, nó đóng vai trò trung tâm hiển thị các thông tin tóm tắt quan trọng và cho phép truy cập nhanh tới các chức năng điều hành. Trên dashboard thường có các thẻ (cards) hiển thị các chỉ số chính (ví dụ: tổng số sinh viên, tổng số lớp, tổng môn học, điểm trung bình), biểu đồ và bảng thống kê cho phép phân tích nhanh tình hình học tập theo kỳ hoặc theo lớp.

Nội dung dashboard nên được lọc theo vai trò người dùng (admin, giảng viên, sinh viên) để chỉ hiện dữ liệu phù hợp, các phần thống kê cần hỗ trợ phân trang, lọc theo thời gian và export dữ liệu khi cần.

Về hiệu năng và vận hành, các báo cáo tổng hợp tốn tài nguyên nên được cache (ví dụ Redis) hoặc chạy dưới dạng job nền, và API trả dữ liệu cho dashboard cần có authentication chặt chẽ để tránh lộ thông tin nhạy cảm.

4.1.3 Chức năng quản lý sinh viên

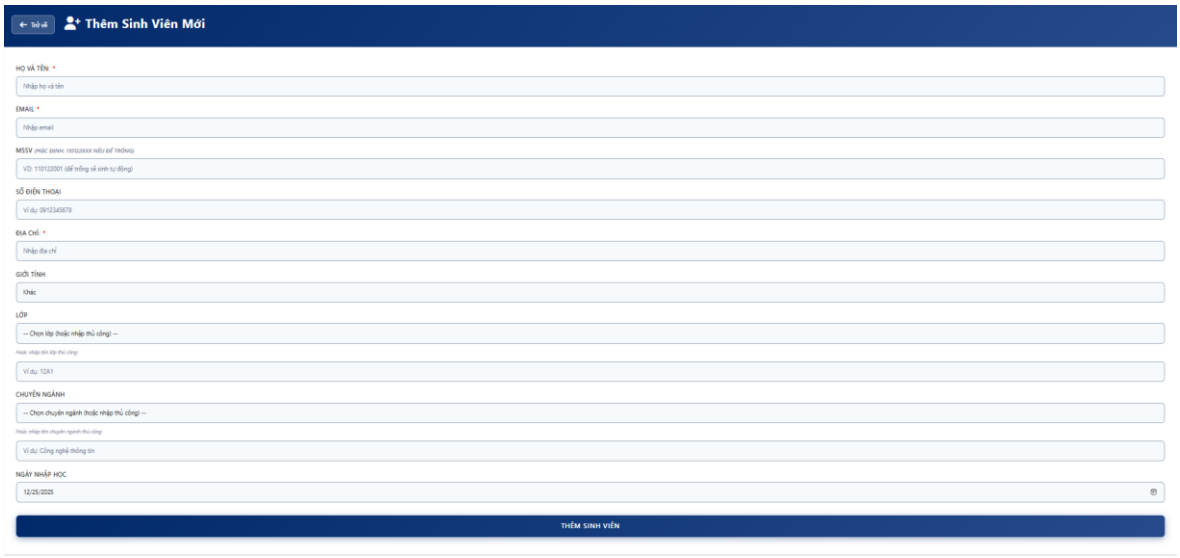


Hình 4.4 Giao diện quản lý sinh viên

Chức năng quản lý sinh viên là một thành phần cốt lõi của hệ thống, đảm nhiệm toàn bộ vòng đời thông tin của sinh viên từ khi nhập học đến khi tốt nghiệp, nó bao gồm việc thêm thông tin sinh viên bằng cách nhập đầy đủ các trường dữ liệu tên, email, sđt, địa chỉ, giới tính,lớp,chuyên ngành, ngày nhập học, lưu trữ và hiển thị thông tin cá nhân, thông tin học tập, cập nhật khi có thay đổi, xóa hoặc đánh dấu tạm khóa hồ sơ khi cần và liên kết hồ sơ với dữ liệu học tập như điểm số và môn học. Chức năng này phải đảm bảo tính toàn vẹn của dữ liệu bằng các kiểm tra ràng buộc và xác thực ở cả phía giao diện người dùng và phía máy chủ, ví dụ kiểm tra định dạng

địa chỉ thư điện tử, kiểm tra không trùng mã sinh viên, và xác thực các trường bắt buộc trước khi lưu. Về phân quyền và bảo mật, mọi thao tác nhạy cảm như tạo mới, sửa hoặc xóa hồ sơ phải yêu cầu người dùng đã xác thực và được ủy quyền theo vai trò công việc, đồng thời hệ thống cần ghi nhật ký hành động để phục vụ kiểm tra và truy vết khi cần. Về trải nghiệm người dùng, giao diện quản lý cần cung cấp công cụ tìm kiếm nhanh, lọc theo nhiều tiêu chí, phân trang, sắp xếp và các chức năng chỉnh sửa nhanh để tăng hiệu quả vận hành.

4.1.3.1 Thêm sinh viên



Hình 4.5 Giao diện thêm sinh viên

Chức năng Thêm sinh viên mới cho phép quản trị viên nhập thông tin sinh viên (tên, email, địa chỉ, số điện thoại, lớp, chuyên ngành, ngày nhập học server sẽ kiểm tra tính hợp lệ của dữ liệu (định dạng email, trường bắt buộc, không trùng email) và xử lý upload file (chỉ cho phép ảnh, giới hạn kích thước), sau đó tạo bản ghi sinh viên trong cơ sở dữ liệu và trả về phản hồi JSON chứa thông tin sinh viên vừa tạo cùng mã trạng thái 201.

4.1.3.2 Sửa thông tin sinh viên

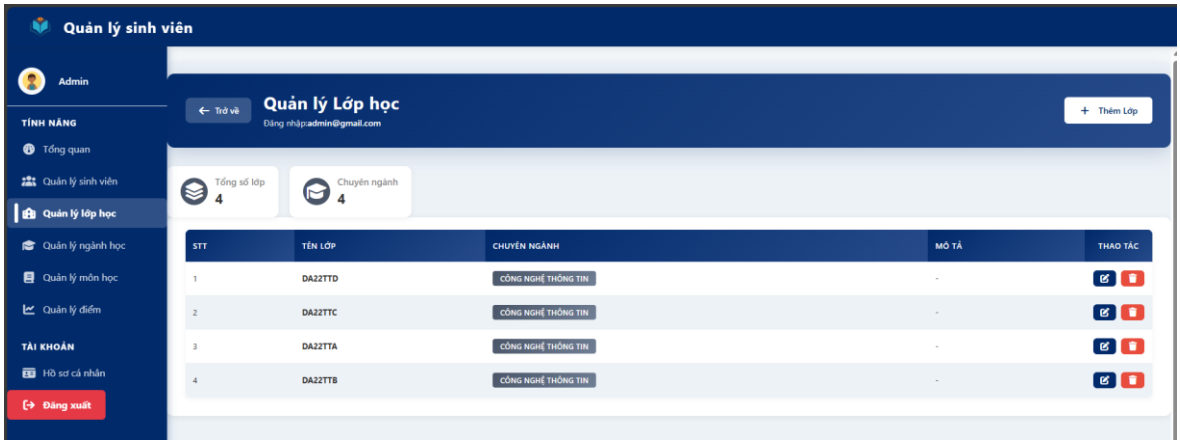
Hình 4.6 Giao diện sửa sinh viên

Chức năng cập nhật sinh viên cho phép quản trị viên sửa thông tin của một sinh viên hiện có bằng cách gửi yêu cầu PUT/PATCH kèm id của sinh viên, server sẽ xác thực token và quyền, kiểm tra tính hợp lệ của dữ liệu (email, số điện thoại, ngày nhập học, các trường bắt buộc), xử lý upload ảnh mới nếu có (kiểm tra loại ảnh và kích thước), tìm bản ghi theo id, áp dụng các thay đổi vào cơ sở dữ liệu và trả về phản hồi JSON chứa thông tin sinh viên đã được cập nhật cùng mã trạng thái 200; nếu không tìm thấy id trả về 404, dữ liệu không hợp lệ trả về 400.

4.1.3.3 Xóa sinh viên

Người quản trị có thể xóa thông tin về sinh viên khỏi dữ liêu nêu sinh viên không còn được quản lý nữa.

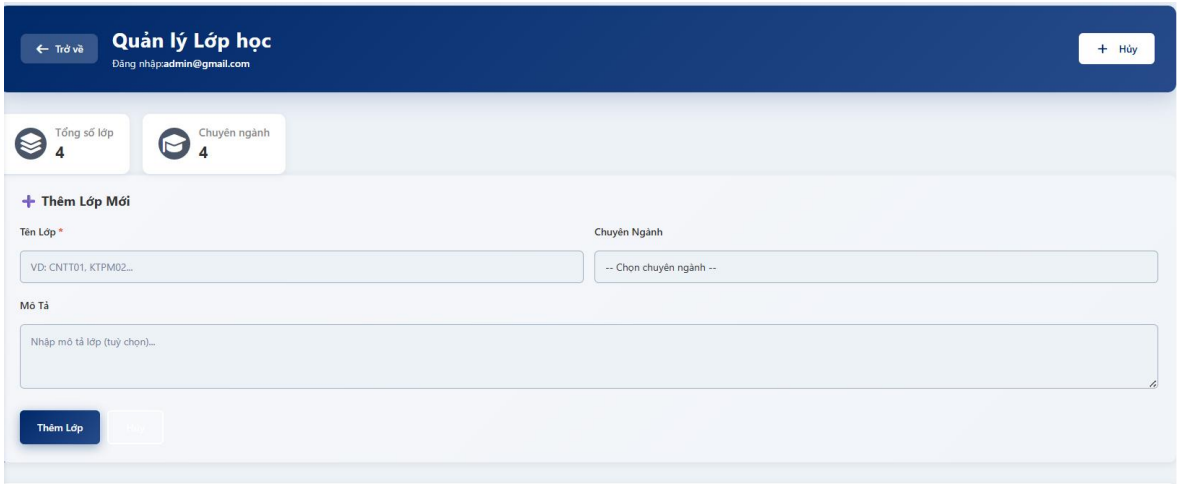
4.1.4 Chức năng quản lý lớp học



Hình 4.7 Giao diện quản lý lớp học

Chức năng "Quản lý lớp học" cho phép người quản trị xem, tạo, sửa và xóa các lớp học trong hệ thống; bao gồm hiển thị danh sách lớp với thông tin (tên lớp, chuyên ngành, mô tả, danh sách sinh viên), kiểm tra tính hợp lệ dữ liệu khi tạo và cập nhật, tất cả thao tác nhạy cảm đều yêu cầu xác thực và phân quyền, và hệ thống trả về kết quả bằng JSON cùng mã trạng thái HTTP phù hợp (200/201/400/404/401).

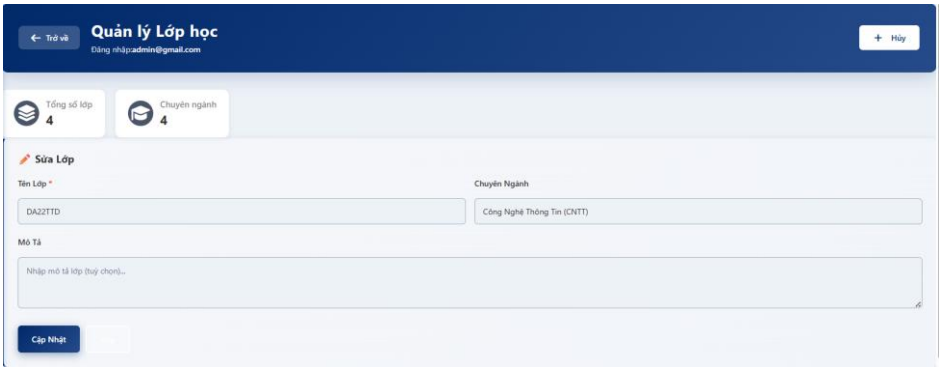
4.1.4.1 Thêm lớp



Hình 4.8 Giao diện thêm lớp

Chức năng "Thêm lớp học" trên web cho phép quản trị viên nhập tên lớp, chọn chuyên ngành và (tùy chọn) mô tả qua form, sau khi nhấn "Thêm Lớp" hệ thống sẽ xác thực dữ liệu (tên lớp không rỗng, chuyên ngành hợp lệ), tạo bản ghi lớp mới trong cơ sở dữ liệu và trả về phản hồi JSON cùng cập nhật danh sách lớp hiển thị trên giao diện (vị trí, bộ lọc và phân trang vẫn được giữ nguyên); thao tác này yêu cầu xác thực người dùng và xử lý lỗi rõ ràng (trùng tên, dữ liệu sai định dạng) để người dùng nhận thông báo phù hợp.

4.1.4.2 Sửa lớp



Hình 4.9 Giao diện sửa lớp

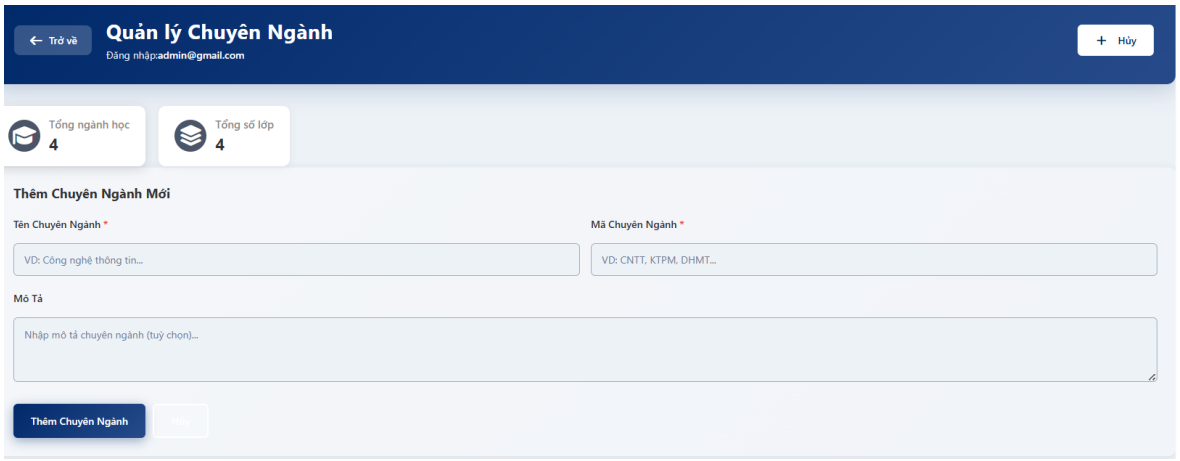
Chức năng "Sửa lớp học" trên web cho phép quản trị viên chọn một lớp từ danh sách, mở form sửa sẵn dữ liệu (tên lớp, chuyên ngành, mô tả), chỉnh thông tin và gửi yêu cầu cập nhật; hệ thống sẽ xác thực đầu vào (tên không rỗng, chuyên ngành hợp lệ, tránh trùng tên), kiểm tra quyền người dùng, cập nhật bản ghi trong cơ sở dữ liệu và trả về JSON (200 OK) giao diện sau đó cập nhật danh sách lớp và hiển thị thông báo thành công hoặc lỗi tương ứng (ví dụ 400/404/401).

4.1.5 Chức năng quản lý ngành học

Hình 4.10 Giao diện quản lý ngành học

Chức năng "Quản lý ngành học" trên web cho phép quản trị viên xem danh sách ngành (mã, tên, mô tả, số lớp liên kết), tạo ngành mới, chỉnh sửa hoặc xóa ngành hiện có thông qua form/ modal; giao diện hỗ trợ tìm kiếm, lọc và hiển thị số lượng lớp cho mỗi ngành, còn phía server thực hiện xác thực người dùng, kiểm tra hợp lệ dữ liệu (mã/ngành không trùng, định dạng hợp lệ), lưu thay đổi vào cơ sở dữ liệu và trả về phản hồi JSON (200/201/400/404) để UI cập nhật trạng thái tương ứng.

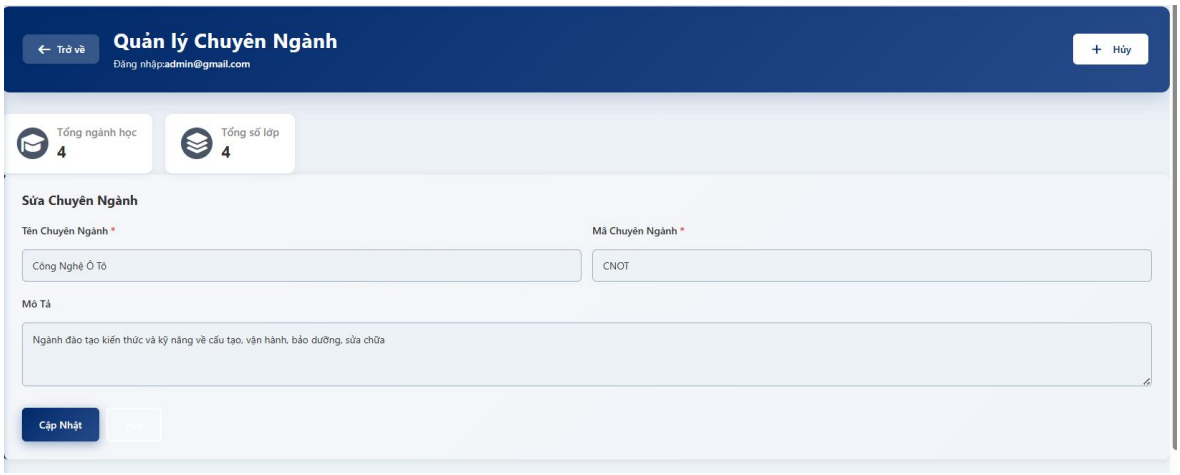
4.1.5.1 Thêm ngành học



Hình 4.11 Thêm ngành học

Chức năng thêm ngành học trên web cho phép quản trị viên nhập thông tin ngành mới (tên ngành, mã ngành, mô tả) qua form, hệ thống sẽ kiểm tra tính hợp lệ (mã/ngành không được bỏ trống và không trùng), yêu cầu người dùng đã xác thực, sau đó lưu bản ghi ngành vào cơ sở dữ liệu và trả về phản hồi (201 Created). Giao diện sẽ tự động cập nhật danh sách ngành, hiển thị số lượng ngành/lớp liên quan và thông báo thành công hoặc lỗi (ví dụ: mã đã tồn tại, dữ liệu không hợp lệ) để người dùng biết kết quả thao tác.

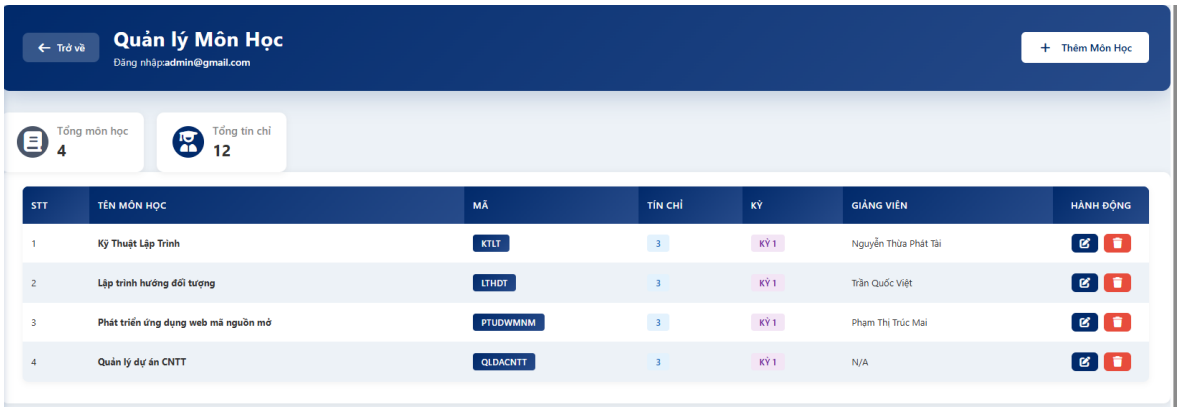
4.1.5.2 Sửa ngành học



Hình 4.12 Giao diện sửa ngành học

Chức năng "Sửa ngành học" cho phép quản trị viên chọn một ngành từ danh sách, mở form chứa dữ liệu hiện tại (tên ngành, mã ngành, mô tả), chỉnh sửa và gửi yêu cầu cập nhật; hệ thống sẽ xác thực quyền truy cập, kiểm tra tính hợp lệ và tránh trùng mã/ngành, cập nhật bản ghi trong cơ sở dữ liệu và trả về phản hồi JSON (200 OK) để giao diện cập nhật danh sách và hiển thị thông báo thành công hoặc lỗi tương ứng (404 nếu không tìm thấy, 400 nếu dữ liệu sai).

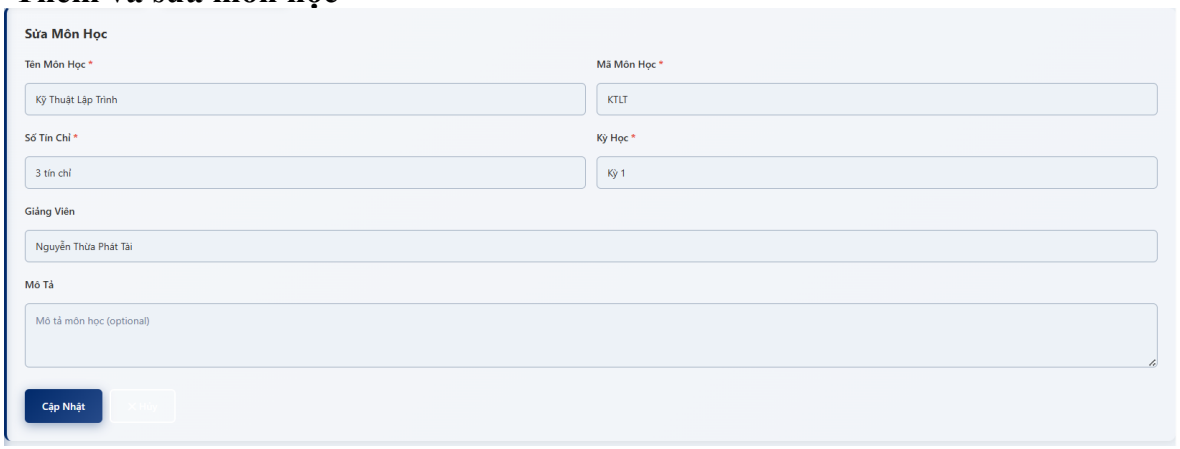
4.1.6 Chức năng quản lý môn học



Hình 4.13 Giao diện quản lý môn học

Trang này cho phép quản trị viên xem, tạo, sửa và xóa các môn học (tên môn, mã môn, số tín chỉ, kỳ, giảng viên, mô tả), hỗ trợ tìm kiếm/lọc và phân trang danh sách; khi thêm hoặc cập nhật, hệ thống sẽ xác thực dữ liệu (mã không trống, tín chỉ hợp lệ), kiểm tra quyền người dùng, lưu thay đổi vào cơ sở dữ liệu và trả về phản hồi JSON để UI cập nhật giao diện và hiển thị thông báo thành công hoặc lỗi tương ứng.

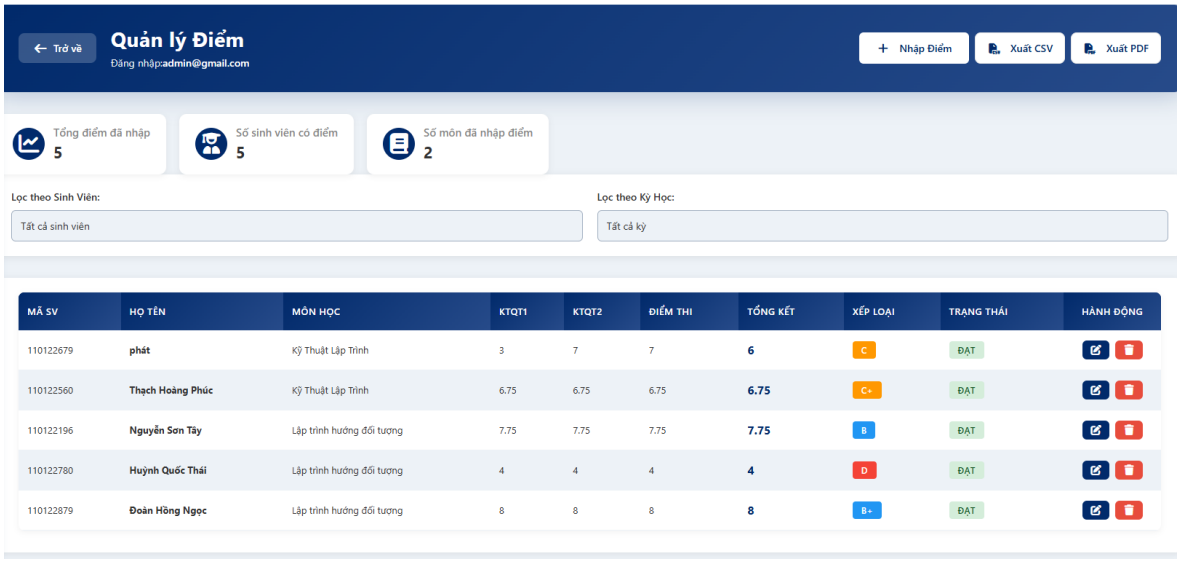
Thêm và sửa môn học



Hình 4.14 Giao diện thêm môn học

Cho phép quản trị viên nhập hoặc chỉnh sửa thông tin môn (tên môn, mã môn, số tín chỉ, kỳ học, giảng viên, mô tả) qua form trên giao diện; khi lưu hệ thống sẽ xác thực dữ liệu (mã/môn không rỗng, tín chỉ hợp lệ), yêu cầu người dùng đã xác thực, gửi yêu cầu tới API (POST /courses để tạo, PUT /courses/:id để cập nhật), lưu thay đổi vào cơ sở dữ liệu và trả về JSON (201/200). Giao diện tự động cập nhật danh sách môn, giữ bộ lọc/phân trang hiện tại và hiển thị thông báo thành công hoặc lỗi (ví dụ: mã trùng, dữ liệu không hợp

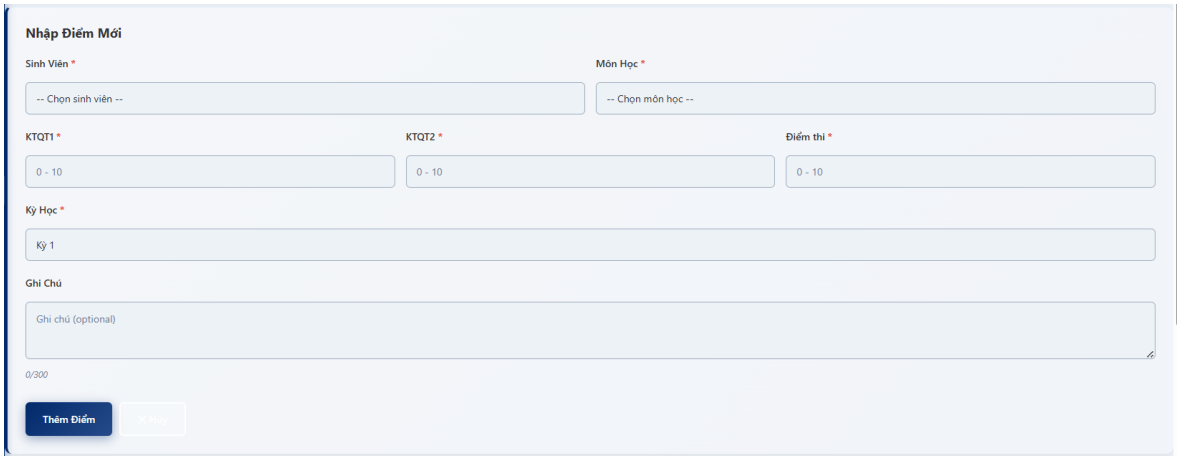
4.1.7 Chức năng quản lý điểm



Hình 4.15 Giao diện quản lý điểm

Tính năng Quản lý điểm cho phép admin quản lý toàn bộ dữ liệu điểm của sinh viên gồm danh sách điểm, thêm,sửa,xóa điểm và xem bảng điểm theo sinh viên hoặc kỳ học. Giao diện hiển thị các thẻ thống kê (tổng số điểm đã nhập, số sinh viên có điểm, số môn đã nhập), kèm bộ lọc theo sinh viên và kỳ học, cùng bảng liệt kê chi tiết (môn học, điểm số, điểm chữ, GPA, trạng thái) với các nút hành động để chỉnh sửa hoặc xóa. Trên backend, các API RESTful (/grades, /grades/transcript/:userId, ...) được bảo vệ bằng JWT để đảm bảo chỉ người dùng đã xác thực mới có thể thay đổi dữ liệu.

Thêm và sửa điểm



Hình 4.16 Giao diện thêm điểm

Cho phép người dùng đã xác thực dễ dàng nhập hoặc điều chỉnh điểm cho từng sinh viên: giao diện cung cấp form chọn Sinh viên, Môn học, nhập Điểm (phạm vi 0–10), chọn Kỳ học và thêm Ghi chú; khi lưu, client gọi API POST /grades để thêm hoặc PUT /grades/:id để cập nhật, backend kiểm tra hợp lệ (số điểm trong khoảng, tham chiếu student/course tồn tại), cập nhật bảng điểm và tính lại GPA/trạng thái tương ứng. Giao diện hiện tại hiển thị form rõ ràng, validation tại chỗ, và thông báo thành công/lỗi để người dùng biết kết quả ngay lập tức.

CHƯƠNG 5: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

5.1 Kết luận

Trong quá trình thực hiện đề tài "Tìm hiểu về API và RESTful Architecture Hệ thống Quản lý Sinh viên", em đã hoàn thành việc thiết kế và triển khai một hệ thống client và server tích hợp, đáp ứng các yêu cầu chức năng cơ bản của một hệ thống quản lý sinh viên: xác thực người dùng (đăng ký, đăng nhập bằng JWT), quản lý sinh viên (CRUD, upload ảnh), quản lý lớp, ngành, môn học, quản lý điểm và cung cấp các API thống kê cho dashboard. Về mặt kỹ thuật, em đã áp dụng kiến trúc RESTful cho việc tổ chức endpoint, phân tách rõ ràng giữa router , controller , model, sử dụng Express.js kết hợp Mongoose để tương tác với MongoDB, và dùng Multer để xử lý upload file. Việc tổ chức mã nguồn theo mô-đun đã giúp giảm độ phức tạp, thuận tiện cho bảo trì và mở rộng sau này.

Kết quả thực nghiệm cho thấy các API chính hoạt động ổn định trong môi trường phát triển: các thao tác CRUD trả về mã trạng thái và payload phù hợp, xác thực JWT bảo vệ các route nhạy cảm, và giao diện client có thể tương tác mượt mà với backend thông qua JSON API.

Trong quá trình phát triển, em đã tích lũy được nhiều kỹ năng thực tế: thiết kế API theo chuẩn REST, xử lý lỗi và trả mã trạng thái hợp lý, lập trình middleware cho xác thực, và phối hợp client – server. Thực hành test bằng Postman và viết tài liệu API giúp nâng cao kỹ năng triển khai quy trình kiểm thử cơ bản và đảm bảo chất lượng đầu ra. Bên cạnh đó, việc đối mặt với các tình huống lỗi và xung đột trong mã nguồn đã rèn luyện khả năng phân tích, debug và đưa ra quyết định kỹ thuật hợp lý.

5.2 Hướng phát triển

- Triển khai phân quyền chi tiết (RBAC) và cơ chế refresh-token để nâng cao bảo mật phiên người dùng.
- Chuyển lưu trữ file lên dịch vụ đám mây (AWS S3, Azure Blob Storage) để tăng độ bền và khả năng mở rộng.
- Sử dụng PATCH cho các cập nhật một phần, chuẩn hóa HTTP methods theo REST semantics.
- Thêm validation đầu vào mạnh mẽ hơn (schema validation, sanitization) và áp dụng rate limiting, Helmet để tăng cường bảo mật.

- Tối ưu frontend bằng chia code-splitting, lazy-loading và giảm phụ thuộc vào thư viện nặng.
- Thêm test tự động (unit, integration API tests) và thiết lập pipeline CI/CD để tự động hóa kiểm thử và deploy.
- Cân nhắc caching cho các endpoint báo cáo nặng (Redis) và pagination cho các danh sách lớn để cải thiện hiệu năng.

DANH MỤC TÀI LIỆU THAM KHẢO

[1] M. Cantelon, M. Harter, T. Holowaychuk, and N. Rajlich, *Node.js in Action*

, Second Edition, Manning Publications, 2017.

[2] E. Hahn, *Express in Action: Writing, Building, and Testing Node.js Applications*, Manning Publications, 2016.

[3] A. Banks and E. Porcello, *Learning React: Functional Web Development with React and Redux*, O'Reilly Media, Inc., 2020.

[4] K. Chodorow, *MongoDB: The Definitive Guide*, Third Edition, O'Reilly Media, Inc., 2019.

[4] K. Chodorow, *MongoDB: The Definitive Guide*, Third Edition, O'Reilly Media, Inc., 2019.

[5] L. Richardson and M. Amundsen, *RESTful Web APIs*, O'Reilly Media, Inc., 2013.

[6] I. Grigorik, *High Performance Browser Networking*, O'Reilly Media, Inc., 2013.

[7] Mozilla Developer Network (MDN), *HTTP Protocol Documentation*, Available: https://developer.mozilla.org/, Accessed 2025.

[8] JSON Web Token (JWT), *JSON Web Token Introduction*, Available: https://jwt.io/, Accessed 2025.

[9] MongoDB Inc., *Mongoose ODM Documentation*, Available: https://mongoosejs.com/, Accessed 2025.

[10] OpenJS Foundation, *Node.js Documentation*, Available: https://nodejs.org/, Accessed 2025.

PHỤ LỤC

PHỤ LỤC 1: Yêu cầu môi trường

- Node.js: 16+ (hoặc LTS hiện tại)
- npm / yarn: tương ứng với Node
- MongoDB: 4.4+ (server hoặc MongoDB Atlas)
- Port mặc định: backend 8000, frontend 3000

Cài đặt & khởi chạy

Cài đặt phụ thuộc:

Backend:

- cd server
- npm install

Frontend:

- cd client
- npm install

Khởi động:

Backend: cd server - npm start

Frontend: cd client - npm start

Chạy đồng thời : npm run dev tại workspace gốc

Biến môi trường

Kết nối mongodb

Biến PORT=8000 xác định cổng mà server lắng nghe (ví dụ truy cập http://localhost:8000), còn MONGODB_URI=mongodb://127.0.0.1:27017/tan là chuỗi kết nối đến cơ sở dữ liệu MongoDB (host 127.0.0.1, cổng 27017, database tên tan), khi triển khai lên môi trường khác chỉ cần chỉnh hai biến này (hoặc đặt biến môi trường tương ứng) mà không phải sửa mã nguồn.

Cài đặt các thư viện:

Express: Express là framework web cho Node.js giúp xây dựng server, định nghĩa route và middleware một cách đơn giản. Trong dự án nó chịu trách nhiệm nhận request từ client, gọi controller xử lý logic và trả response theo chuẩn RESTful.

Mongoose: Mongoose là Object Document Mapper (ODM) cho MongoDB, cho phép định nghĩa schema, model và thao tác với cơ sở dữ liệu bằng API JavaScript có cấu trúc. Dự án dùng Mongoose để định nghĩa các schema (User, Class, Course, Grade…) và thực hiện các thao tác CRUD.

jsonwebtoken: Thư viện jsonwebtoken hỗ trợ tạo, ký và xác thực JWT (JSON Web Token). Ứng dụng dùng JWT để cấp token khi người dùng đăng nhập và bảo vệ các route cần xác thực bằng middleware kiểm tra token.

bcryptjs: bcryptjs dùng để băm (hash) mật khẩu trước khi lưu vào database và so sánh mật khẩu khi đăng nhập. Băm mật khẩu giúp bảo mật thông tin đăng nhập tránh lưu mật khẩu thuần.

Multer: Multer là middleware xử lý multipart/form-data, dùng để upload file (hình ảnh) từ client lên server. Nó cấu hình nơi lưu trữ, tên file và kiểm soát kiểu/kích thước file trước khi lưu.

dotenv: dotenv dùng để load biến môi trường từ file .env vào process.env. Giúp tách cấu hình (port, DB URI, JWT_SECRET) khỏi mã nguồn và dễ thay đổi theo môi trường (dev/staging/prod).

CORS: cors là middleware cấu hình Cross-Origin Resource Sharing, cho phép hoặc hạn chế các nguồn (origin) frontend truy cập API. Dùng để tránh lỗi trình duyệt chặn request giữa client và server khi chạy trên domain/port khác nhau.

Axios (server/client): Axios là HTTP client promise-based cho cả server và client, dùng để gọi các API từ server hoặc từ frontend. Nó hỗ trợ request/response interceptors, timeout và xử lý JSON thuận tiện.

React: React là thư viện UI cho frontend, giúp xây dựng giao diện dạng component, quản lý state và rendering nhanh. Project dùng React để xây dựng giao diện quản trị, form tạo/sửa sinh viên, bảng điểm, dashboard…

React Router DOM: react-router-dom quản lý điều hướng (routing) phía client cho SPA, cho phép định nghĩa các route, chuyển trang mà không reload, và bảo vệ route cần xác thực.

Bootstrap: Bootstrap là framework CSS giúp dựng giao diện responsive nhanh với hệ thống grid, component sẵn có (button, form, modal). Dự án dùng Bootstrap để tạo layout và style cơ bản cho dashboard.

Chart.js / Recharts: Thư viện vẽ biểu đồ (Chart.js và/hoặc Recharts) dùng để hiển thị thống kê, biểu đồ GPA, phân bố điểm trên dashboard, giúp trực quan hóa dữ liệu cho người dùng.

PHỤ LỤC 2: Cấu hình Axios

```
import axios from 'axios';

const api = axios.create({
  baseURL: process.env.REACT_APP_API_BASE_URL || 'http://localhost:8000/api',
  timeout: 10000
});

// Thêm interceptor để đính token tự động
api.interceptors.request.use(config => {
  const token = localStorage.getItem('token');
  if (token) config.headers.Authorization = token.startsWith('Bearer') ? token : `Bearer ${token}`;
  return config;
}, error => Promise.reject(error));

export default api;
```

Cấu hình Axios

Đoạn mã tạo một instance Axios cấu hình sẵn dùng chung cho toàn bộ frontend: baseURL lấy từ biến môi trường REACT_APP_API_BASE_URL (nếu không có thì mặc định http://localhost:8000/api), timeout giới hạn thời gian chờ 10 giây, đồng thời đăng ký một request interceptor để tự động đính Authorization header với giá trị token lấy từ localStorage (nếu token chưa có tiền tố Bearer thì thêm vào) trước khi gửi request. Việc này giúp tập trung cấu hình URL, timeout và xử lý authentication ở một chỗ, tránh lặp mã khi gọi API và tạo điều kiện dễ dàng thay đổi cấu hình cho các môi trường khác nhau.

PHỤ LỤC 3: API xác thực

3.1 Lưu thông tin đăng ký

API Endpoint : POST /auth/register

```
export const register = async (req, res) => {
    try {
        const { email, password, passwordConfirm } = req.body;
        if (!email || !password || !passwordConfirm) {
            return res.status(400).json({ success: false, message: "Email, mật khẩu và xác nhận mật
khẩu là bắt buộc" });
        }
        if (password !== passwordConfirm) {
            return res.status(400).json({ success: false, message: "Mật khẩu không trùng khớp" });
        }
        const existingUser = await Auth.findOne({ email: email.toLowerCase() });
        if (existingUser) return res.status(400).json({ success: false, message: "Email này đã được
đăng ký" });
        const newAuth = new Auth({ email: email.toLowerCase(), password, role: "user" });
        const saveData = await newAuth.save();
        const token = generateToken(saveData._id, saveData.email);
        res.status(201).json({
            success: true,
            message: "Đăng ký tài khoản thành công",
            token,
            user: { id: saveData._id, email: saveData.email, role: saveData.role }
        });
    } catch (error) {
        const errorResponse = handleValidationError(error);
        res.status(errorResponse.status).json({ success: false, message: errorResponse.message });
    }
};
```

Hàm Controller

POST /auth/register nhận email, password, passwordConfirm; kiểm tra hợp lệ, kiểm tra trùng email, lưu user (mật khẩu được hash trong model), rồi trả về JWT token và thông tin user. Đây là bước tạo tài khoản (stateless) client lưu token để gọi các API bảo vệ sau này.

3.2 Lưu thông tin đăng nhập

API Endpoint : POST /auth/login

```
export const login = async (req, res) => {
    try {
        const { email, password } = req.body;
        if (!email || !password) return res.status(400).json({ success: false, message: "Email và mật
khẩu là bắt buộc" });
        const user = await Auth.findOne({ email: email.toLowerCase() }).select("+password");
        if (!user) return res.status(401).json({ success: false, message: "Email hoặc mật khẩu không
đúng" });
        const isPasswordMatch = await user.comparePassword(password);
        if (!isPasswordMatch) return res.status(401).json({ success: false, message: "Email hoặc mật
khẩu không đúng" });
        const token = generateToken(user._id, user.email);
        res.status(200).json({
            success: true,
            message: "Đăng nhập thành công",
            token,
            user: { id: user._id, email: user.email, role: user.role }
        });
    } catch (error) {
        res.status(500).json({ success: false, message: error.message });
    }
};
```

Hàm login

Hàm login trong authController.js xử lý endpoint POST /auth/login: nhận email và password từ req.body, kiểm tra hợp lệ, tìm user trong cơ sở dữ liệu (kèm trường password), so sánh mật khẩu bằng phương thức hash (ví dụ comparePassword), nếu khớp tạo JWT bằng generateToken và trả về HTTP 200 với token và thông tin user; nếu thiếu trường trả 400, nếu email/mật khẩu sai trả 401, lỗi server trả 500. Client lưu token (ví dụ localStorage) và gửi Authorization: Bearer {token} cho các API bảo vệ; nên bổ sung cơ chế bảo mật như chính sách mật khẩu mạnh, refresh token, rate limiting và khóa tài khoản sau nhiều lần đăng nhập thất bại.

PHỤ LỤC 4: API sinh viên

4.1 Lấy danh sách sinh viên

API entpoin: GET /getUser

```
export const getAllUsers = async (req, res) => {
    try {
        const page = parseInt(req.query.page) || 1;
        const limit = parseInt(req.query.limit) || 10;
        const search = req.query.search || '';
        const filter = req.query.filter || '';
        const sort = req.query.sort || '-createdAt';
        if (page < 1) return res.status(400).json({ success: false, message: "Page phải lớn hơn 0" });
        if (limit < 1 || limit > 1000) return res.status(400).json({ success: false, message: "Limit
phải từ 1 đến 1000" });
        let query = {};
        if (search) {
            query.$or = [
                { name: { $regex: search, $options: 'i' } },
                { email: { $regex: search, $options: 'i' } }
            ];
        }
       ...
        const totalPages = Math.ceil(total / limit);
        res.status(200).json({
            success: true,
            message: "Lấy danh sách sinh viên thành công",
            data: userData,
            pagination: { currentPage: page, pageSize: limit, total: total, totalPages, hasNextPage:
page < totalPages, hasPrevPage: page > 1 }
        });
    } catch (error) {
        res.status(500).json({ success: false, message: error.message });
    }
};
```

Hàm getAllUsers

Hàm trả về danh sách sinh viên với hỗ trợ tìm kiếm, lọc (filter), sắp xếp (sort) và phân trang (page, limit). Trả về dữ liệu và thông tin phân trang.

Hàm getAllUsers trả về danh sách sinh viên với khả năng tìm kiếm, lọc, sắp xếp và phân trang, giúp client lấy dữ liệu lớn một cách hiệu quả. Hàm đọc các query parameters như page, limit, search, filter, sort, xây dựng truy vấn MongoDB tương ứng (tìm theo tên/email, lọc theo lớp, sắp xếp theo trường mong muốn), rồi tính total

và trả data kèm pagination (currentPage, pageSize, total, totalPages, hasNextPage, hasPrevPage). Hàm cũng kiểm tra hợp lệ cho page và limit để tránh yêu cầu quá lớn, và xử lý trường hợp không có dữ liệu bằng phản hồi hợp lệ. Về lỗi, trả mã 400 cho tham số không hợp lệ và 500 cho lỗi server, khuyến nghị thêm caching (Redis) và giới hạn limit tối đa để bảo toàn hiệu năng khi sản xuất.

4.2 Lấy thông tin sinh viên

API entpoin:GET /getUser/:id

```
export const getUserById = async (req, res) => {
    try {
        const id = req.params.id;
        const userExists = await User.findById(id);
        if (!userExists) {
            return res.status(404).json({ success: false, message: "Sinh viên không tồn tại" });
        }
        res.status(200).json({
            success: true,
            message: "Lấy thông tin sinh viên thành công",
            data: userExists
        });
    } catch (error) {
        res.status(500).json({ success: false, message: error.message });
    }
}
```

Hàm getUserById

Hàm getUserById lấy thông tin chi tiết của một sinh viên theo id (MongoDB ObjectId). Luồng xử lý: đọc req.params.id, truy vấn User.findById(id), nếu không tìm thấy trả 404 với thông báo phù hợp, nếu tìm thấy trả 200 kèm đối tượng user. Hàm nên validate định dạng id trước khi truy vấn để tránh lỗi truy vấn không mong muốn; cũng có thể lựa chọn projection để chỉ trả các trường cần thiết (tránh lộ dữ liệu nhạy cảm). Trong production, cần kiểm tra quyền truy cập (ví dụ chỉ admin hoặc owner được xem một số trường).

4.3 Tạo user

API entpoin:POST /addUser

```
export const create = async (req, res) => {
    try {
        const { name, email, phone, address, gender, className, majorName, joinDate } = req.body;
        if (!name || !email || !address) {
            return res.status(400).json({ success: false, message: "Tên, email và địa chỉ là bắt buộc"
});
        }
        const newUser = new User(req.body);
        const saveData = await newUser.save();
        res.status(201).json({
            success: true,
            message: "Thêm sinh viên thành công",
            data: saveData
        });
    } catch (error) {
        const errorResponse = handleValidationError(error);
        res.status(errorResponse.status).json({ success: false, message: errorResponse.message });
    }
}
```

Hàm tạo user

Hàm này nhận dữ liệu từ req.body, kiểm tra các trường bắt buộc (ví dụ name, email, address), khởi tạo new User(req.body) và lưu vào DB. Khi lưu thành công trả 201 với data là tài liệu mới; khi có lỗi validation hoặc duplicate (index unique) sẽ trả mã 400 với thông điệp rõ ràng. Lưu ý: mã trong repo tách việc upload file ra riêng nếu muốn upload ảnh khi tạo, cần dùng multipart/form-data và middleware multer để xử lý file trước khi gọi logic tạo.

4.4 Cập nhật user

API entpoin: PUT /updateUser/:id

```
export const update = async (req, res) => {
    try {
        const id = req.params.id;
        const userExists = await User.findById(id);
        if (!userExists) {
            return res.status(404).json({ success: false, message: "Sinh viên không tồn tại" });
        }
        const updateData = await User.findByIdAndUpdate(id, req.body, { new: true, runValidators: true
});
        res.status(200).json({
            success: true,
            message: "Cập nhật sinh viên thành công",
            data: updateData
        });
    } catch (error) {
        const errorResponse = handleValidationError(error);
        res.status(errorResponse.status).json({ success: false, message: errorResponse.message });
    }
};
```

Hàm update

Hàm update cập nhật thông tin sinh viên theo id với dữ liệu từ req.body (có thể gửi form-data nếu kết hợp upload). Trước khi cập nhật hàm kiểm tra user tồn tại, sau đó gọi findByIdAndUpdate(..., { new: true, runValidators: true }) để trả về document mới và chạy validator của Mongoose. Nên giới hạn các trường được phép cập nhật (whitelist) để tránh cập nhật các trường nhạy cảm hoặc hệ thống (như _id, createdAt), và xử lý lỗi validation trả về thông tin hữu ích cho client.

4.5 Xóa user

API entpoin: DELETE /deleteUser/:id

```
export const deleteUser = async (req, res) => {
    try {
        const id = req.params.id;
        const userExists = await User.findById(id);
        if (!userExists) {
            return res.status(404).json({ success: false, message: "Sinh viên không tồn tại" });
        }
        await User.findByIdAndDelete(id);
        res.status(200).json({ success: true, message: "Xóa sinh viên thành công" });
    } catch (error) {
        res.status(500).json({ success: false, message: error.message });
    }
}
```

Hàm delete

Hàm deleteUser xóa sinh viên theo id. Luồng: kiểm tra tồn tại bằng findById, nếu không có trả 404, nếu có thực hiện findByIdAndDelete và trả 200 khi xóa thành công. Nên cân nhắc delete mềm (soft delete) thay vì xóa cứng nếu cần giữ lịch sử (thêm trường deletedAt/isDeleted) hoặc thực hiện xoá liên quan (cascade) với dữ liệu phụ thuộc (điểm, ảnh, hồ sơ) để tránh dữ liệu rác. Đồng thời kiểm tra quyền xóa và log hoạt động xóa để truy vết.

PHỤ LỤC 5: API Lớp học

- API Endpoint: GET /api/classes/all: Lấy danh sách lớp (hỗ trợ phân trang và tìm kiếm
- API Endpoint: GET /api/classes/dropdown: Lấy toàn bộ lớp để dùng cho dropdown
- API Endpoint: GET /api/classes/:id: Lấy chi tiết một lớp
- API Endpoint: POST /api/classes: Tạo lớp mới,kiểm tra hợp lệ và cập nhật
- API Endpoint: PUT /api/classes/:id: Cập nhật lớp
- API Endpoint: DELETE /api/classes/:id: Xóa lớp

Ví dụ hàm tạo lớp:

```
export const createClass = async (req, res) => {
    try {
        const { className, description, majorId } = req.body;
        if (!className) return res.status(400).json({ success: false, message: "Tên lớp là bắt buộc" });
        if (majorId) { const majorExists = await Major.findById(majorId); if (!majorExists) return
res.status(404).json({ success: false, message: "Chuyên ngành không tồn tại" }); }
        const newClass = new Class({ className, description: description || "", majorId: majorId || null,
totalStudents: 0 });
        const saveData = await newClass.save();
        if (majorId) await Major.findByIdAndUpdate(majorId, { $inc: { totalClasses: 1 } });
        await createAudit({ actorId: req.userId || null, action: 'create', entity: 'Class', entityId:
saveData._1d, entityName: saveData.className, details: { createdFields: { className, description, majorId
} } });
        res.status(201).json({ success: true, message: "Tạo lớp thành công", data: saveData });
    } catch (error) { /* xử lý lỗi */ }
};
```

Hàm tạo lớp

Endpoint này được sử dụng để tạo mới một lớp học trong hệ thống. Server nhận dữ liệu từ client bao gồm tên lớp (className), mô tả (description) và mã chuyên ngành (majorId). Trước khi lưu dữ liệu, hệ thống thực hiện kiểm tra tính hợp lệ của thông tin đầu vào, đảm bảo tên lớp không được để trống và chuyên ngành (nếu có) phải tồn tại trong hệ thống. Sau khi tạo lớp thành công, API sẽ cập nhật lại số lượng lớp (totalClasses) của chuyên ngành liên quan và ghi nhận lịch sử thao tác (audit log). Kết quả trả về là thông tin chi tiết của lớp vừa được tạo kèm thông báo tạo thành công.

PHỤ LỤC 6: API Môn học

- API Endpoint: POST /api/courses: Tạo mới một môn học trong hệ thống

- API Endpoint: GET /api/courses/all: lấy danh sách toàn bộ môn học trong hệ thống.

- API Endpoint: GET /api/courses/dropdown : cung cấp danh sách rút gọn các môn học

- API Endpoint: GET /api/courses/:id: lấy thông tin chi tiết của một môn học cụ thể dựa trên mã định danh

- API Endpoint: PUT /api/courses/:id: cập nhật thông tin của một môn học đã tồn tại và yêu cầu người dùng phải được xác thực

- API Endpoint: DELETE /api/courses/:id: xóa một môn học khỏi hệ thống và chỉ cho phép người dùng có quyền truy cập hợp lệ thực hiện

Ví dụ hàm tạo môn học:

```
export const createCourse = async (req, res) => {
    try {
        const { courseName, code, credits, semester, description, instructor } = req.body;
        if (!courseName || !code || !credits || !semester) return res.status(400).json({ success: false,
message: "Vui lòng điền tất cả trường bắt buộc" });
        const existingCourse = await Course.findOne({ $or: [{ courseName }, { code: code.toUpperCase() }]
});
        if (existingCourse) return res.status(400).json({ success: false, message: "Tên môn học hoặc mã
môn học đã tồn tại" });
        const newCourse = new Course({ courseName, code: code.toUpperCase(), credits, semester,
description: description || "", instructor: instructor || "" });
        await newCourse.save();
        res.status(201).json({ success: true, message: "Tạo môn học thành công", data: newCourse });
    } catch (error) { /* xử lý lỗi */ }
};
```

Hàm tạo môn học

Endpoint này dùng để tạo mới một môn học trong hệ thống và chỉ cho phép người dùng đã xác thực truy cập. Server tiếp nhận các thông tin gồm tên môn học (courseName), mã môn (code), số tín chỉ (credits), học kỳ (semester), mô tả và giảng viên phụ trách. Trước khi lưu dữ liệu, hệ thống kiểm tra các trường bắt buộc và đảm bảo rằng tên môn học hoặc mã môn học chưa tồn tại nhằm tránh trùng lặp. Sau khi hợp lệ, mã môn được chuẩn hóa về dạng chữ in hoa, dữ liệu được lưu vào cơ sở dữ liệu MongoDB và API trả về thông tin chi tiết của môn học vừa được tạo cùng thông báo thành công.

PHỤ LỤC 7: API điểm

- API Endpoint: POST /api/grades: Tạo bản ghi điểm cho sinh viên theo từng môn học và học kỳ.

- API Endpoint: GET /api/grades/all: truy vấn danh sách điểm của toàn bộ sinh viên trong hệ thống.

- API Endpoint: GET /api/grades/transcript/:userId: lấy bảng điểm của một sinh viên cụ thể dựa trên mã định danh sinh viên.

- API Endpoint: GET /api/grades/:id: lấy thông tin chi tiết của một bản ghi điểm theo mã.

- API Endpoint: PUT /api/grades/:id: cập nhật thông tin điểm của sinh viên.

- API Endpoint: DELETE /api/grades/:id: xóa một bản ghi điểm khỏi hệ thống.

- Migration endpoints:

- POST /api/grades/backfill-components: được sử dụng một lần để đồng bộ dữ liệu cũ, tự động điền các thành phần điểm (KTQT, điểm thi) cho những bản ghi chỉ có tổng điểm, đồng thời tính lại các giá trị liên quan.

- POST /api/grades/recalculate-derived: dùng để tính toán lại toàn bộ các trường dẫn xuất như điểm tổng kết, GPA, điểm chữ, xếp loại và trạng thái học tập cho tất cả bản ghi điểm trong hệ thống, đảm bảo dữ liệu luôn chính xác và nhất quán.

- POST /api/grades/fix-misclassified: rà soát và sửa các bản ghi điểm bị phân loại sai (ví dụ điểm đủ điều kiện đạt nhưng bị xếp loại rớt), sau đó cập nhật lại điểm chữ, GPA, xếp loại và trạng thái học tập cho đúng với quy định.

Ví dụ hàm tạo điểm:

```
export const createGrade = async (req, res) => {
    try {
        const { userId, courseId, midterm, final, assignment, ktqt1, ktqt2, exam, score, semester, notes } = req.body;
        if (!userId || !courseId || !semester) return res.status(400).json({ success: false, message: "Vui lòng điền các trường bắt
buộc: Sinh viên, Môn học, Kỳ" });
        const user = await User.findById(userId); if (!user) return res.status(404).json({ success: false, message: "Không tìm thấy
sinh viên" });
        const course = await Course.findById(courseId); if (!course) return res.status(404).json({ success: false, message: "Không tìm
thấy môn học" });
        const parseNum = (v) => (v === null || v === undefined || v === '') ? undefined : Number(v);
        const m = parseNum(midterm ?? ktqt1);
        const f = parseNum(final ?? ktqt2);
        const a = parseNum(assignment ?? exam);
        let finalScore;
        if (m !== undefined && f !== undefined && a !== undefined) {
            finalScore = +((((m + f) / 2) + a) / 2).toFixed(2);
        } else if (score !== undefined && score !== null && score !== '') {
            finalScore = Number(score);
        } else {
            return res.status(400).json({ success: false, message: 'Vui lòng cung cấp KTQT1, KTQT2 và Điểm thi hoặc tổng điểm' });
        }
        const gpa = calculateGPA(finalScore);
        const grade = getGradeLetter(finalScore);
        const classification = getClassification(grade);
        const status = getStatus(finalScore);
        const newGrade = new Grade({ userId, courseId, score: finalScore, ktqt1: m ?? null, ktqt2: f ?? null, exam: a ?? null,
semester, credits: course.credits, gpa, grade, classification, status, notes: notes || "", mssv: user.mssv || null });
        await newGrade.save();
        await newGrade.populate("userId", "name email mssv");
        await newGrade.populate("courseId", "courseName code credits");
        res.status(201).json({ success: true, message: "Tạo điểm thành công", data: newGrade });
    } catch (error) { /* xử lý lỗi */ }
};
```

Hàm tạo điểm

Endpoint này dùng để tạo bản ghi điểm cho sinh viên theo từng môn học và học kỳ. Hệ thống tiếp nhận thông tin sinh viên (userId), môn học (courseId), học kỳ (semester) cùng các thành phần điểm như kiểm tra quá trình (KTQT1, KTQT2), điểm thi hoặc tổng điểm. Server kiểm tra sự tồn tại của sinh viên và môn học trước khi xử lý. Nếu đầy đủ các thành phần điểm, hệ thống sẽ tự động tính điểm tổng kết theo công thức quy định; nếu chỉ có tổng điểm, hệ thống sử dụng trực tiếp giá trị này. Sau đó, API tự động suy ra GPA, điểm chữ, xếp loại học lực và trạng thái đạt/không đạt, lưu

dữ liệu vào cơ sở dữ liệu và trả về thông tin điểm chi tiết đã được liên kết với sinh viên và môn học.

PHỤ LỤC 8: API tổng quan

API Endpoint: GET /api/dashboard/summary

Mô tả:

Endpoint này dùng để tổng hợp và cung cấp các số liệu thống kê tổng quan của hệ thống quản lý sinh viên phục vụ cho trang Dashboard. API thực hiện truy vấn và tính toán các thông tin như tổng số sinh viên, tổng số lớp học, chuyên ngành, môn học, GPA trung bình, số lượng sinh viên theo từng mức học lực (xuất sắc, khá, trung bình, yếu) hoặc theo trạng thái học tập. Dữ liệu trả về ở dạng JSON, giúp giao diện quản trị hiển thị biểu đồ, thẻ thống kê và các báo cáo trực quan, hỗ trợ cán bộ quản lý theo dõi nhanh tình hình học tập và vận hành của hệ thống.

PHỤ LỤC 9: API ngành học

API Endpoint: POST /api/major: cho phép tạo mới một chuyên ngành

API Endpoint: GET /api/major/all: hỗ trợ lấy danh sách chuyên ngành kèm phân trang và tìm kiếm theo tên hoặc mã, phục vụ hiển thị bảng dữ liệu trên giao diện quản trị.

API Endpoint: GET /api/major/dropdown: trả về danh sách rút gọn chuyên ngành.

API Endpoint: GET /api/major/:id: cho phép lấy chi tiết thông tin của một chuyên ngành cụ thể.

API Endpoint: PUT /api/major/:id dùng để cập nhật thông tin chuyên ngành.

API Endpoint: DELETE /api/major/:id: cho phép xóa chuyên ngành.

Ví dụ hàm tạo ngành học:

```
export const createMajor = async (req, res) => {
    try {
        const { majorName, code, description } = req.body;
        if (!majorName || !code) return res.status(400).json({ success: false, message: "Tên chuyên ngành và mã là bắt buộc" });
        const newMajor = new Major({ majorName, code: code.toUpperCase(), description: description || "", totalClasses: 0 });
        const saveData = await newMajor.save();
        await createAudit({ actorId: req.userId || null, actorEmail: req.userEmail || null, action: 'create', entity: 'Major',
entityId: saveData._id, entityName: saveData.majorName, details: { createdFields: { majorName, code, description } } });
        res.status(201).json({ success: true, message: "Tạo chuyên ngành thành công", data: saveData });
    } catch (error) { /* xử lý lỗi */ }
};
```

Hàm tạo ngành học

Hàm createMajor được sử dụng để tạo mới một ngành/chuyên ngành trong hệ thống quản lý sinh viên. Hàm nhận dữ liệu từ request body bao gồm tên ngành (majorName), mã ngành (code) và mô tả (description). Trước khi lưu dữ liệu, hệ

thống kiểm tra các trường bắt buộc nhằm đảm bảo tính hợp lệ của thông tin đầu vào. Mã ngành được chuẩn hóa về dạng chữ in hoa để tránh trùng lặp không nhất quán. Sau khi tạo mới, ngành học được khởi tạo với số lượng lớp bằng 0 và lưu vào cơ sở dữ liệu MongoDB.