

**BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC MỎ - ĐỊA CHẤT**

**BÁO CÁO HỌC THUẬT HỌC KỲ 1
NĂM HỌC 2025 – 2026**

**XÂY DỰNG HỆ THỐNG HUMG CODELAB PHỤC VỤ
CÁC HỌC PHẦN LẬP TRÌNH**

Người báo cáo: Ths. Nguyễn Văn Thắng

Đơn vị: Bộ môn Tin học Trắc địa

Khoa Công nghệ Thông tin

Hà Nội, 2025

**BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC MỎ - ĐỊA CHẤT**

**BÁO CÁO HỌC THUẬT HỌC KỲ 2
NĂM HỌC 2025 – 2026**

**XÂY DỰNG HỆ THỐNG HUMG CODELAB PHỤC VỤ CÁC
HỌC PHẦN LẬP TRÌNH**

Phòng KHCN

Bộ môn Tin học Trắc địa

Người báo cáo

Ths. Nguyễn Văn Thắng

Hà Nội - 2025

MỤC LỤC

Mở đầu	2
1. Giới thiệu chung	3
2. Cơ sở lý thuyết và tổng quan nghiên cứu.....	4
2.1. Dạy học lập trình và yêu cầu công cụ hỗ trợ.....	4
2.2. Hệ thống chấm tự động (Auto-grading) và đánh giá chương trình	5
2.3. Thực thi mã nguồn an toàn (Sandboxing) và quản trị tài nguyên	6
2.4. Nền tảng học tập xã hội (Social learning)	7
3. Bài toán, mục tiêu và yêu cầu hệ thống.....	8
3.1. Mô tả bài toán và phạm vi áp dụng	8
3.2. Mục tiêu nghiên cứu và mục tiêu triển khai	9
3.3 Đối tượng sử dụng và kịch bản nghiệp vụ (Use cases)	10
3.4 Yêu cầu chức năng	11
4. Xây dựng và triển khai hệ thống HUMG Codelab	14
4.1 Công nghệ và môi trường triển khai.....	14
4.2 Hiện thực các mô-đun backend	15
4.3 Cơ chế thực thi mã nguồn và chấm tự động.....	19
4.4 Frontend và trải nghiệm người dùng	21
5. Kết luận và hướng phát triển	25
5.1. Kết luận	25
5.2. Hướng phát triển.....	26
TÀI LIỆU THAM KHẢO	29

Mở đầu

HUMG Codelab là nền tảng web hỗ trợ dạy–học và đánh giá trong các học phần lập trình, hướng tới chuẩn hóa môi trường thực hành, rút ngắn vòng phản hồi học tập và giảm tải vận hành cho giảng viên. Nghiên cứu này trình bày quá trình xây dựng hệ thống theo mô hình client–server, trong đó frontend cung cấp trình soạn thảo mã nguồn trực tuyến và các giao diện quản lý lớp học/bài thi, còn backend đảm nhiệm xác thực–phân quyền, lưu trữ dữ liệu, thực thi chương trình và chấm tự động theo bộ kiểm thử. Hệ thống hỗ trợ biên dịch và chạy chương trình trên máy chủ, trả kết quả theo thời gian thực; đồng thời cung cấp cơ chế giám sát tiến độ và một số tín hiệu chống gian lận (như phát hiện thao tác copy–paste, chuyển tab) nhằm tăng tính minh bạch trong kiểm tra lập trình. Bên cạnh ngôn ngữ Java (trọng tâm cho học phần OOP), nền tảng mở rộng hỗ trợ Python và Jupyter Notebook để phục vụ các nội dung học tập đa dạng. Kết quả triển khai cho thấy hệ thống đáp ứng các chức năng cốt lõi của một codelab học thuật: quản lý ngân hàng bài tập/bài thi, chạy thử–nộp bài, tự động chấm, theo dõi realtime và hỗ trợ tương tác học tập thông qua không gian chia sẻ nội dung. Báo cáo cũng thảo luận các yêu cầu quan trọng về an toàn thực thi, giới hạn tài nguyên và khả năng mở rộng khi phục vụ lớp học đông người.

1. Giới thiệu chung

Trong bối cảnh chuyển đổi số giáo dục đại học, các học phần lập trình đặt ra yêu cầu ngày càng cao về môi trường thực hành, đánh giá thường xuyên và quản lý lớp học theo dữ liệu. Ở mô hình truyền thống, hoạt động luyện tập và kiểm tra lập trình thường phụ thuộc vào phòng máy, việc cài đặt công cụ phát triển không đồng nhất giữa các thiết bị, quy trình nộp bài rời rạc và thiếu cơ chế chấm tự động theo tiêu chí chuẩn hóa. Những hạn chế này làm giảm tính kịp thời của phản hồi (feedback), tăng tải vận hành cho giảng viên và khó đảm bảo công bằng trong đánh giá khi số lượng người học lớn.

Đề tài “Xây dựng hệ thống HUMG Codelab phục vụ các học phần lập trình” tập trung thiết kế và triển khai một nền tảng web hỗ trợ toàn bộ vòng đời của hoạt động học lập trình: từ cung cấp ngân hàng bài tập/bài thi, soạn thảo và chạy chương trình trực tuyến, đến tự động chấm điểm theo bộ kiểm thử và giám sát tiến độ theo thời gian thực. Hệ thống hướng tới việc chuẩn hóa trải nghiệm thực hành, rút ngắn vòng phản hồi học tập, đồng thời tăng cường năng lực quản trị lớp học thông qua các chức năng theo dõi realtime, ghi nhận tiến độ làm bài và các cơ chế hạn chế gian lận (ví dụ: phát hiện hành vi copy–paste, chuyển tab trong quá trình làm bài).

Về phạm vi, hệ thống HUMG Codelab cung cấp các mô-đun cốt lõi gồm: (i) quản lý bài tập/bài thi và thư viện nội dung theo mức độ từ cơ bản đến nâng cao (đặc biệt phù hợp các chủ đề Java và lập trình hướng đối tượng); (ii) môi trường soạn thảo mã nguồn trực tuyến với khả năng biên dịch–thực thi trên máy chủ; (iii) chấm tự động dựa trên test case, trả kết quả/biểu hiện lỗi biên dịch–thời gian chạy; (iv) truyền thông thời gian thực phục vụ giám sát lớp học; và (v) mở rộng hỗ trợ đa ngôn ngữ lập trình (Java, Python, Jupyter Notebook) nhằm đáp ứng các học phần/hoạt động học tập đa dạng. Bên cạnh đó, nền tảng còn tích hợp không gian tương tác học thuật

dạng bảng tin (feeds) để khuyến khích chia sẻ tri thức và thảo luận trong cộng đồng học tập.

Đóng góp chính của đề tài thể hiện ở việc hiện thực hóa một kiến trúc hệ thống web có khả năng phục vụ đồng thời nhu cầu dạy–học–đánh giá trong các học phần lập trình: kết hợp giao diện người dùng hiện đại (trình soạn thảo mã nguồn) với dịch vụ thực thi mã nguồn phía máy chủ, cơ chế chấm tự động theo bộ kiểm thử, và lớp hạ tầng realtime/giám sát nhằm đảm bảo trải nghiệm học tập liền mạch, minh bạch và có thể mở rộng. Trên phương diện triển khai, hệ thống được xây dựng theo mô hình client–server, sử dụng Node.js/Express cho backend, React cho frontend, cơ sở dữ liệu MongoDB và kênh giao tiếp thời gian thực Socket.io; đồng thời đặt trọng tâm vào các yêu cầu phi chức năng như hiệu năng, an toàn thực thi và quản trị vận hành.

2. Cơ sở lý thuyết và tổng quan nghiên cứu

2.1. Dạy học lập trình và yêu cầu công cụ hỗ trợ

Học lập trình là quá trình hình thành năng lực giải quyết vấn đề thông qua thiết kế thuật toán, hiện thực hóa bằng mã nguồn và kiểm chứng bằng các ca kiểm thử. Khác với một số học phần thiên về ghi nhớ, tiến bộ trong học lập trình phụ thuộc mạnh vào tần suất luyện tập và chất lượng phản hồi. Do đó, một môi trường hỗ trợ học lập trình cần đáp ứng tối thiểu các yêu cầu: (i) cung cấp bài tập/bài thi theo tiến trình năng lực; (ii) cho phép thực hành liền mạch trên nhiều thiết bị mà không phụ thuộc cấu hình cài đặt; (iii) đưa ra phản hồi nhanh, rõ ràng, có thể lặp lại; và (iv) hỗ trợ giảng viên quan sát trạng thái lớp học, phát hiện sớm người học gặp khó khăn và tổng hợp minh chứng đánh giá.

Trong thực tiễn đào tạo, hoạt động đánh giá lập trình có thể diễn ra theo nhiều hình thức: bài tập về nhà, bài kiểm tra trên lớp, thi kết thúc học phần hoặc đánh giá quá trình. Các cách tiếp cận thường gặp gồm: (i) chấm theo kết quả chạy đúng (correctness) thông qua test case; (ii) chấm theo rubric (tiêu chí) cho các khía cạnh như cấu trúc chương trình, phong cách mã, hiệu quả; và (iii) đánh giá theo quá trình (process-based) dựa trên lịch sử sửa bài, tần suất chạy thử, mức độ tiến bộ. Trong khuôn khổ nền tảng codelab, chấm theo test case là nền tảng kỹ thuật quan trọng vì cho phép tự động hóa phản hồi ở quy mô lớn; tuy nhiên để đảm bảo tính sư phạm, hệ thống cần thiết kế cơ chế phản hồi phù hợp (ví dụ: phân loại lỗi, gợi ý từng phần) và cung cấp dữ liệu để giảng viên can thiệp khi cần.

2.2. Hệ thống chấm tự động (Auto-grading) và đánh giá chương trình

Chấm tự động (auto-grading) trong bài toán lập trình thường dựa trên nguyên lý thực thi chương trình người học với tập đầu vào chuẩn hóa và so sánh đầu ra thu được với đầu ra kỳ vọng [1]. Về mặt kỹ thuật, quy trình chấm gồm các bước: tiếp nhận mã nguồn, kiểm tra định dạng/cấu trúc tối thiểu (ví dụ: lớp main, hàm main), biên dịch (đối với ngôn ngữ biên dịch), thực thi trong môi trường giới hạn, thu thập chuẩn ra/chuẩn lỗi (stdout/stderr), sau đó đối chiếu với đáp án theo quy tắc so khớp (exact match, normalize whitespace, hoặc so khớp có dung sai). Điểm số có thể được tính theo tỉ lệ test case vượt qua, hoặc theo trọng số từng nhóm test (cơ bản/biên/hiệu năng).

Một thành phần không thể tách rời của auto-grading là quản trị tài nguyên: giới hạn thời gian chạy, số tiến trình con, bộ nhớ, và kích thước dữ liệu I/O. Các giới hạn này vừa giúp phòng tránh hành vi vô tình (vòng lặp vô hạn) vừa giảm rủi ro lạm dụng tài nguyên. Ngoài ra, hệ thống cần cơ chế ghi nhận và diễn giải lỗi để phản hồi có giá trị học tập: phân biệt lỗi biên dịch, lỗi runtime, sai kết quả, quá thời gian, hoặc vi phạm chính sách an toàn.

Về ưu điểm, auto-grading cung cấp phản hồi nhanh và nhất quán, đặc biệt phù hợp với lớp học đông, giúp giảng viên giảm tải chấm tay và người học có thể tự điều chỉnh bài làm thông qua chạy thử nhiều lần. Tuy nhiên, chấm theo test case có các hạn chế: (i) phụ thuộc độ bao phủ của bộ kiểm thử; (ii) khó đánh giá chất lượng mã và tư duy thiết kế (đặc biệt ở nội dung OOP); (iii) nguy cơ “hard-code” đầu ra khi người học đoán được test; và (iv) khó xử lý các bài toán có nhiều đáp án hợp lệ hoặc yêu cầu trình bày. Vì vậy, trong thiết kế hệ thống học thuật, auto-grading nên được xem là cơ chế “đánh giá đúng/sai” cốt lõi, đồng thời cần mở đường cho bổ sung rubric, kiểm tra đạo văn và đánh giá theo quá trình ở các giai đoạn phát triển tiếp theo.

2.3. Thực thi mã nguồn an toàn (Sandboxing) và quản trị tài nguyên

Việc cho phép người học gửi và thực thi mã nguồn trên máy chủ đặt ra yêu cầu an toàn đặc thù vì mã nguồn được xem như dữ liệu không tin cậy (untrusted). Các rủi ro phổ biến gồm: truy cập hệ thống tệp, gọi lệnh hệ điều hành, truy cập mạng, khai thác thư viện nguy hiểm, tiêu thụ tài nguyên bất thường (CPU/RAM), hoặc tạo tiến trình con để né giới hạn. Do đó, thiết kế sandbox cần kết hợp nhiều lớp biện pháp:

- Ràng buộc hành vi: chặn các API/statement nguy hiểm (ví dụ: các lời gọi thực thi hệ thống, thao tác tệp, thao tác mạng) bằng kiểm tra tĩnh (lọc chuỗi/từ khóa) hoặc kiểm tra ở runtime.

- Giới hạn tài nguyên: áp đặt timeout cho biên dịch và chạy; giới hạn bộ nhớ; giới hạn số tiến trình; giới hạn kích thước output để tránh “spam stdout”.

- Cô lập môi trường: chạy mã trong thư mục tạm/không gian cô lập; tách tiến trình thực thi khỏi tiến trình web; đảm bảo dọn dẹp tài nguyên sau khi chạy.

- Ghi log và truy vết: lưu thông tin yêu cầu (người dùng, bài, ngôn ngữ, thời gian chạy), trạng thái lỗi và các sự kiện vi phạm để phục vụ giám sát và điều tra.

Trong bối cảnh hệ thống HUMG Codelab, sandbox còn cần tương thích đa ngôn ngữ: Java (biên dịch bằng javac và chạy bằng java), Python (thực thi trực tiếp bằng python3) và Jupyter Notebook (trích xuất mã từ các cell và chạy tuần tự). Điều này làm tăng yêu cầu chuẩn hóa pipeline thực thi, đồng thời tăng độ phức tạp trong kiểm soát phụ thuộc và kiểm soát output (đặc biệt với notebook có thể sinh biểu đồ/đầu ra phức hợp). Vì vậy, chiến lược thiết kế cần ưu tiên “an toàn mặc định” (secure-by-default), chỉ cho phép tập tính năng tối thiểu phục vụ học tập và mở rộng dần theo nhu cầu.

2.4. Nền tảng học tập xã hội (Social learning)

Học tập xã hội (social learning) nhấn mạnh vai trò của tương tác, thảo luận và chia sẻ kinh nghiệm trong việc hình thành tri thức và kỹ năng. Đối với học lập trình, người học thường tiến bộ nhanh hơn khi được tham khảo cách tiếp cận đa dạng, nhận phản hồi từ bạn học/giảng viên và thảo luận về lỗi sai. Vì vậy, việc tích hợp một không gian trao đổi (ví dụ: bảng tin, bài đăng, bình luận, chia sẻ đoạn mã có tô màu cú pháp) có thể nâng cao động lực và hỗ trợ học tập ngoài giờ học chính khóa [1].

Trong hệ thống HUMG Codelab, mô-đun feeds được xem như lớp bổ trợ sự phạm: tạo kênh chia sẻ kiến thức, đặt câu hỏi, thảo luận lời giải, đồng thời cung cấp khả năng kiểm duyệt theo vai trò để duy trì môi trường học thuật an toàn. Khi kết hợp với auto-grading và dữ liệu tiến độ, nền tảng có thể phát triển theo hướng học tập dựa trên cộng đồng (community-based learning), trong đó tri thức được lan tỏa không chỉ từ giảng viên mà còn từ chính người học.

3. Bài toán, mục tiêu và yêu cầu hệ thống

3.1. Mô tả bài toán và phạm vi áp dụng

Các học phần lập trình tại HUMG (đặc biệt các học phần Java và lập trình hướng đối tượng) thường có quy mô lớp lớn, nhu cầu luyện tập theo tuần và kiểm tra/thi định kỳ. Bài toán đặt ra là xây dựng một hệ thống codelab phục vụ đồng thời hai mục tiêu: (i) hỗ trợ học tập thực hành với phản hồi nhanh thông qua chạy thử và chấm tự động; (ii) hỗ trợ kiểm tra/thi lập trình với cơ chế quản trị lớp học và giám sát theo thời gian thực nhằm đảm bảo minh bạch và giảm tải vận hành.

Trong thực tiễn, các khó khăn nổi bật gồm: môi trường cài đặt không đồng nhất giữa các máy; quy trình nộp bài thủ công, khó tổng hợp; thời gian chấm kéo dài; thiếu kênh quan sát tiến độ khi thi; và thiếu công cụ giao tiếp/chia sẻ tri thức gắn với ngữ cảnh môn học. Do đó, hệ thống HUMG Codelab cần cung cấp một “vòng lặp thực hành” khép kín: giao bài → làm bài trên web editor → chạy thử → nộp bài → chấm theo test case → phản hồi và lưu vết, đồng thời cho phép theo dõi tiến độ realtime và ghi nhận một số tín hiệu hành vi để hỗ trợ chống gian lận.

Phạm vi áp dụng của hệ thống trong báo cáo này gồm:

- Đối tượng: sinh viên, giảng viên (và vai trò quản trị/điều phối nếu cần).
- Hoạt động hỗ trợ: học tập (luyện tập, làm bài, trao đổi) và đánh giá (kiểm tra/thi lập trình).
- Ngôn ngữ: Java là trọng tâm; mở rộng Python và Jupyter Notebook để phù hợp các học phần/kịch bản học tập khác nhau.
- Hình thức triển khai: hệ thống web theo mô hình client–server; tích hợp kênh realtime bằng Socket.io; lưu trữ dữ liệu trên MongoDB.

Các giả định/giới hạn chính:

- Hệ thống chấm tự động ưu tiên đánh giá correctness theo test case; các tiêu chí chất lượng mã/rubric có thể bổ sung về sau.
- Tín hiệu anti-cheat mang tính hỗ trợ ra quyết định của giảng viên, không tự động kết luận vi phạm.
- Thực thi mã nguồn là vùng rủi ro cao nên cần áp dụng chính sách an toàn mặc định và giới hạn tài nguyên.

3.2. Mục tiêu nghiên cứu và mục tiêu triển khai

Mục tiêu nghiên cứu của đề tài là đề xuất và hiện thực hóa một mô hình hệ thống codelab phục vụ học phần lập trình theo hướng dữ liệu và phản hồi nhanh, kết hợp các thành phần: (i) editor và trải nghiệm thực hành liền mạch; (ii) pipeline thực thi và chấm tự động theo test case; (iii) giám sát realtime trong bối cảnh lớp học/thi; và (iv) cơ chế an toàn thực thi và quản trị tài nguyên, đảm bảo hệ thống vận hành ổn định khi phục vụ nhiều người học đồng thời.

Mục tiêu triển khai được cụ thể hóa theo hai nhóm:

(1) Mục tiêu chức năng (Functional goals):

- Quản lý người dùng và phân quyền theo vai trò (giảng viên/sinh viên).
- Quản lý lớp học, nội dung bài tập/bài thi, test case và cấu hình chấm.
- Cung cấp web editor hỗ trợ viết mã, chạy thử (run) và nộp bài (submit).
- Tự động chấm theo test case, lưu kết quả và phản hồi lỗi (compile/runtime/wrong answer/timeout).
- Theo dõi tiến độ và một số hoạt động theo thời gian thực (realtime monitoring), bao gồm điểm danh realtime theo lớp.
- Thu thập tín hiệu anti-cheat ở mức hỗ trợ (copy–paste, chuyển tab) và lưu vết sự kiện.

- Mở rộng hỗ trợ đa ngôn ngữ (Java/Python/Jupyter Notebook).
- Tăng cường tương tác học tập: bảng tin chia sẻ (feeds) và hệ thống nhắn tin realtime (messaging), hỗ trợ Markdown chia sẻ code.

(2) Mục tiêu phi chức năng (Non-functional goals):

- Bảo mật và riêng tư (xác thực/ủy quyền, giới hạn truy cập dữ liệu theo lớp).
- An toàn thực thi (sandbox), kiểm soát timeout/bộ nhớ và dọn dẹp tài nguyên.
- Hiệu năng và khả năng chịu tải (độ trễ chấp nhận được cho run/submit, cơ chế hạn chế quá tải).
- Độ ổn định và khả năng vận hành (logging/monitoring, sao lưu/khôi phục, triển khai production).

3.3 Đối tượng sử dụng và kịch bản nghiệp vụ (Use cases)

Hệ thống xác định các tác nhân (actors) chính và các kịch bản nghiệp vụ tương ứng như sau.

Vai trò	Mã Use Case	Tên Use Case	Mô tả chi tiết
Sinh viên	UC-S01	Tham gia lớp/học phần	Xem danh sách lớp, truy cập nội dung học tập theo từng lớp.
	UC-S02	Làm bài trên web editor	Mở đề bài, viết mã, nhận gợi ý và hiển thị lỗi cú pháp (editor support).
	UC-S03	Chạy thử (Run)	Gửi mã nguồn lên server, nhận kết quả (output) hoặc lỗi theo thời gian thực.
	UC-S04	Nộp bài (Submit)	Nộp mã nguồn để chấm theo test case, nhận kết quả tổng hợp và chi tiết từng test.
	UC-S05	Xem lịch sử và phản hồi	Xem các lần chạy/nộp, trạng thái chấm điểm, lỗi biên dịch hoặc runtime.

	UC-S06	Điểm danh realtime	Tham gia phiên điểm danh của lớp và cập nhật trạng thái có mặt.
	UC-S07	Tương tác học tập	Đăng bài/bình luận trên feeds, nhắn tin trao đổi với giảng viên và sinh viên khác.
Giảng viên	UC-T01	Quản lý lớp	Tạo lớp, quản lý danh sách sinh viên, tổ chức các hoạt động theo lớp.
	UC-T02	Quản lý bài tập/bài thi	Tạo, sửa, xóa đề bài; thiết lập cấu hình (thời gian, ngôn ngữ, hướng dẫn).
	UC-T03	Quản lý test case	Nhập test case, thiết lập quy tắc so khớp output và trọng số chấm điểm.
	UC-T04	Giám sát realtime	Theo dõi tiến độ làm bài, trạng thái run/submit và tổng quan tình hình lớp.
	UC-T05	Điểm danh realtime	Bắt đầu/kết thúc phiên điểm danh, theo dõi cập nhật tức thời theo lớp.
	UC-T06	Xử lý anti-cheat	Xem log sự kiện (copy–paste, chuyển tab) và đối chiếu khi cần thiết.
	UC-T07	Trao đổi và hỗ trợ	Nhắn tin realtime, phản hồi trên feeds và kiểm duyệt nội dung.
	UC-T08	Tổng hợp kết quả	Xem kết quả bài làm theo sinh viên/lớp, trích xuất báo cáo đánh giá.

3.4 Yêu cầu chức năng

Yêu cầu chức năng được mô tả chi tiết theo bảng sau:

Mã FR	Tên yêu cầu chức năng	Nội dung đặc tả chi tiết
FR-01	Xác thực người dùng	Hệ thống cho phép đăng ký, đăng nhập và duy trì phiên làm việc (session) an toàn.

FR-02	Phân quyền theo vai trò	Phân biệt tối thiểu vai trò Giáo viên và Sinh viên; kiểm soát quyền truy cập chức năng và dữ liệu tương ứng.
FR-03	Quản lý lớp học	Giáo viên tạo và quản lý lớp; sinh viên truy cập lớp học theo quyền được cấp.
FR-04	Quản lý ngân hàng bài	Giáo viên tạo/sửa/xóa đề; hệ thống lưu trữ và hiển thị đề theo lớp, chương hoặc chủ đề.
FR-05	Quản lý cấu hình bài	Cho phép thiết lập các thuộc tính: thời gian, trạng thái kích hoạt, ngôn ngữ lập trình hỗ trợ, hướng dẫn làm bài.
FR-06	Quản lý test case	Giáo viên quản lý (tạo/sửa/xóa) test case; hệ thống gắn test case với bài để chấm điểm tự động.
FR-07	Web Editor trực tuyến	Cung cấp trình soạn thảo hỗ trợ tô màu cú pháp theo ngôn ngữ và các thao tác cơ bản (copy, paste, format).
FR-08	Chạy thử mã nguồn (Run)	Sinh viên gửi mã nguồn và input; hệ thống thực thi và trả về output/error (stdout/stderr) theo thời gian thực.
FR-09	Nộp bài (Submit)	Hệ thống chạy toàn bộ test case khi sinh viên nộp bài, tổng hợp trạng thái đạt/không đạt và lưu kết quả.
FR-10	Phản hồi lỗi có cấu trúc	Phân loại và trả về các nhóm lỗi: Compilation, Runtime, Wrong Answer, Timeout, Vi phạm chính sách.
FR-11	Lưu vết và lịch sử	Lưu trữ lịch sử Run/Submit của từng người học theo từng bài; cho phép xem lại kết quả chi tiết.

FR-12	Hỗ trợ đa ngôn ngữ	Hỗ trợ tối thiểu Java; mở rộng Python, Jupyter Notebook; cho phép chọn ngôn ngữ theo cấu hình bài bài tập.
FR-13	Giám sát Realtime	Cập nhật tức thời các sự kiện (run, submit, trạng thái tiến độ) của sinh viên lên giao diện quản lý của giảng viên.
FR-14	Điểm danh Realtime	Hỗ trợ bắt đầu/kết thúc phiên điểm danh; sinh viên tham gia và giảng viên nhận cập nhật trạng thái có mặt ngay lập tức.
FR-15	Thu thập tín hiệu Anti-cheat	Ghi nhận hành vi (copy-paste, chuyển tab) trong bối cảnh làm bài thi/kiểm tra để giảng viên đối chiếu.
FR-16	Bảng tin học tập (Feeds)	Cho phép đăng bài, bình luận, thả tim; hỗ trợ định dạng Markdown, khối mã code và cơ chế kiểm duyệt nội dung.
FR-17	Nhắn tin Realtime	Hỗ trợ nhắn tin 1-1, định dạng Markdown, hiển thị trạng thái đã đọc và tín hiệu "đang soạn bài".
FR-18	Thông báo (Push/PWA)	Gửi thông báo đẩy khi có tin nhắn mới hoặc sự kiện quan trọng ngay cả khi người dùng không online.
FR-19	Tải lên nội dung số	Hỗ trợ tải tệp (ví dụ: ảnh trong feeds) có giới hạn dung lượng và kiểm soát định dạng tệp.
FR-20	Báo cáo và tổng hợp	Cho phép giáo viên xem tổng hợp kết quả theo bài/lớp/sinh viên và xuất dữ liệu báo cáo (ví dụ: file CSV).

4. Xây dựng và triển khai hệ thống HUMG Codelab

4.1 Công nghệ và môi trường triển khai

Hệ thống HUMG Codelab được hiện thực theo kiến trúc web client–server, trong đó backend cung cấp REST API và kênh realtime, còn frontend cung cấp giao diện codelab cho người học và giao diện quản trị cho giảng viên.

(1) Backend:

- Nền tảng: Node.js + Express.
- Cơ sở dữ liệu: MongoDB thông qua Mongoose. Kết nối được tối ưu hóa bằng pool (ví dụ: maxPoolSize, minPoolSize, timeout và tham số chờ kết nối) để phục vụ số lượng người dùng đồng thời lớn trong giờ học/thi.
- Bảo mật và tối ưu truyền tải: helmet (CSP tùy môi trường), cors, compression, và express-rate-limit cho nhóm endpoint /api/* (kèm giới hạn nghiêm ngặt hơn cho /api/auth/*).
- Realtime: Socket.io, cấu hình hỗ trợ polling và websocket, giới hạn kích thước buffer, timeout ping, giới hạn số kết nối đồng thời và cơ chế dọn dẹp kết nối “stale”.
- Thực thi mã nguồn: mô hình “worker pool” dựa trên worker_threads, cho phép thực thi đa ngôn ngữ (Java/Python/Jupyter) có hàng đợi (queue), timeout và giới hạn tài nguyên; đồng thời quản lý thư mục tạm để tránh rò rỉ tài nguyên.
- Quan sát hệ thống: endpoint /api/health và /api/metrics cung cấp thông tin sức khỏe và số liệu vận hành phục vụ giám sát.

(2) Frontend:

- Nền tảng: React.

- Thư viện UI: kết hợp Ant Design (một số component) và TailwindCSS cho bố cục/giao diện hiện đại; hiển thị nội dung Markdown bằng react-markdown + remark-gfm.
- Trình soạn thảo mã nguồn: Monaco Editor (@monaco-editor/react).
- Quản lý truy vấn và trạng thái dữ liệu: react-query cho fetch/caching; Axios cho giao tiếp API với interceptor tự gắn JWT; có lớp “request optimizer” hỗ trợ throttle/deduplicate một số request nhạy cảm (đặc biệt là chạy code).
- Tích hợp realtime: Socket.io client dùng cho giám sát thi, điểm danh realtime, nhắn tin realtime; kèm cơ chế tránh tạo nhiều kết nối trùng lặp

Về môi trường triển khai, hệ thống sử dụng biến môi trường (.env/.env.production) để cấu hình URI MongoDB, JWT secret, origin hợp lệ, cổng dịch vụ, tham số giới hạn thực thi, và khóa VAPID (phục vụ web push). Trong chế độ phát triển, frontend thường chạy riêng (cổng 3000) và proxy về backend API; trong triển khai production có thể đặt phía sau reverse proxy (nginx/PM2) tùy hạ tầng.

4.2 Hiện thực các mô-đun backend

Backend được tổ chức theo cấu trúc điển hình routes/–controllers/–models/, cho phép tách biệt rõ: định tuyến API, xử lý nghiệp vụ và mô hình dữ liệu.

(1) Xác thực và phân quyền

- Xác thực dựa trên JWT: middleware authenticateToken kiểm tra token, tải thông tin người dùng (trừ mật khẩu) và gắn vào req.user.

- Phân quyền theo vai trò: requireTeacher/requireStudent đảm bảo người dùng chỉ truy cập chức năng đúng vai trò; đồng thời hỗ trợ kiểm tra quyền truy cập tài nguyên (ví dụ bài thi, bài nộp) theo nguyên tắc “chỉ chủ sở hữu/giảng viên tạo bài được truy cập”.

(2) Quản lý bài thi (Exam)

- API routes/exam.js hiện thực các chức năng: lấy danh sách bài thi theo quyền, xem chi tiết bài thi, tạo–sửa–xóa bài thi (giảng viên), thống kê và truy xuất bài nộp.

- Mô hình dữ liệu models/Exam.js hỗ trợ: đề bài, starter code theo ngôn ngữ, danh sách test case (có cờ isHidden), giới hạn thời gian (phút), số lần nộp tối đa, khoảng thời gian diễn ra, danh sách lớp/sinh viên được phép, và cờ bật/tắt chống gian lận. Các virtual field isOngoing/isUpcoming/isEnded giúp frontend hiển thị trạng thái bài thi.

- Trong bối cảnh thi, hệ thống che đáp án của test case ẩn để giảm rủi ro lộ đáp án; đồng thời kiểm tra quyền truy cập theo lớp (dựa vào classCode của sinh viên và allowedClasses của bài thi).

(3) Nộp bài, chạy thử và lưu nháp (Submissions & Drafts)

- API routes/submission.js tách rõ các hành vi: run (/submissions/run), submit (POST /submissions), draft (lưu/đọc nháp), truy xuất bài nộp (theo sinh viên, theo bài thi), chấm thử công (giảng viên), và ghi nhận vi phạm.

- controllers/submissionController.js hiện thực:

- Submit: tạo bản ghi Submission, kiểm tra quyền và trạng thái bài thi (đang diễn ra), kiểm tra giới hạn số lần nộp, sau đó chấm theo test case theo hướng bất đồng bộ để giảm thời gian chờ của request; trạng thái bài nộp được cập nhật theo vòng đời pending → running → completed/error.

- Run: thực thi code theo input tùy chọn hoặc theo test case của bài thi; trả về output/error và (nếu có) kết quả từng test.

- Draft & session: lưu nháp code theo bài thi (ExamDraft) và quản lý phiên làm bài (bắt đầu/kết thúc) để đồng bộ bộ đếm thời gian còn lại.

- Mô hình models/Submission.js lưu các trường: code, ngôn ngữ, trạng thái, kết quả từng test, lỗi biên dịch, thời gian thực thi, và cơ chế tính điểm tự động dựa trên tỉ lệ test case đạt; đồng thời hỗ trợ chấm thủ công với manualScore/teacherComments.

- Mô hình models/ExamDraft.js ghi nhận nhấp và thời gian làm bài theo session: startSession/endSession/getRemainingTime, phục vụ đồng bộ “thời gian còn lại” phía client và giảm sai lệch khi reload/mất kết nối.

(4) Điểm danh realtime theo lớp (Attendance)

- Mô-đun lớp học (routes/class.js, controllers/classController.js) hỗ trợ quản lý lớp, import danh sách sinh viên từ Excel/CSV, xuất danh sách và quản lý điểm danh.

- Điểm danh realtime dùng models/AttendanceSession.js lưu phiên điểm danh đang hoạt động; sinh viên điểm danh qua API và server phát sự kiện Socket.io student_attendance_updated/attendance_session_updated tới room attendance-{classId} để giao diện giảng viên cập nhật tức thì.

(5) Ghi nhận vi phạm và anti-cheat

- Vi phạm được lưu ở models/CheatingLog.js thông qua API /submissions/violation, bao gồm loại vi phạm, số lần, userAgent, IP và dữ liệu bổ sung.

- Backend cung cấp thống kê vi phạm theo bài thi (tổng, gần đây, theo loại, theo sinh viên) để giảng viên theo dõi.

(6) Realtime monitoring (Socket.io)

Hệ thống thiết kế các “room” theo ngữ cảnh:

- exam-`{examId}`: theo dõi tiến độ bài thi, cập nhật code (giới hạn payload) và sự kiện hoạt động.

- attendance-`{classId}`: cập nhật điểm danh realtime.

- user-`{userId}` và conversation-`{conversationId}`: phục vụ nhắn tin realtime.

Để tránh quá tải, backend kết hợp:

- Giới hạn số kết nối Socket.io đồng thời, timeout kết nối không hoạt động, dọn dẹp kết nối stale.

- Rate limiting cho từng loại sự kiện (code-update, activity, heartbeat, violation) thông qua “socket rate limiter”.

- “Smart monitoring”: trong production chỉ phát các hoạt động quan trọng, và tối ưu payload (ví dụ chỉ gửi hash/độ dài thay vì toàn bộ code).

(7) Nền tảng học tập xã hội (Feeds)

- API routes/feeds.js và controllers/feedController.js cung cấp CRUD bài đăng, bình luận lồng nhau (giới hạn mức), like/unlike, thống kê cộng đồng, và cơ chế kiểm duyệt (ẩn/hiện).

- Upload ảnh sử dụng multer với giới hạn dung lượng, giới hạn số file và whitelist định dạng; lưu ở thư mục uploads/feeds và public qua /uploads.

(8) Nhắn tin realtime và thông báo (Messaging & Push)

- Mô-đun routes/messages.js, controllers/messageController.js, models/Conversation.js và models/Message.js hiện thực nhắn tin 1-1, trạng thái đã đọc, đếm unread, gợi ý “đang soạn”, và ngữ cảnh hội thoại (general/class/exam/exercise).

- Khi gửi tin nhắn, backend có thể phát sự kiện realtime qua Socket.io, đồng thời tích hợp web push để gửi thông báo khi người nhận không online (PWA).

4.3 Cơ chế thực thi mã nguồn và chấm tự động

Hon Thực thi mã nguồn là lõi kỹ thuật của HUMG Codelab. Hệ thống hiện thực theo mô hình “pool thực thi” (universal execution pool) nhằm kiểm soát đồng thời, giới hạn tài nguyên và giảm chi phí spawn tiến trình tràn lan.

(1) Tầng điều phối: codeExecutor và universal pool

- `utils/codeExecutor.js` đóng vai trò façade, cung cấp `executeCode(language, code, input)` cho controller và các hàm `validateJavaCode/validatePythonCode/validateJupyterNotebook` để kiểm tra cấu trúc và chặn một số pattern nguy hiểm.

- `utils/universalExecutionPool.js` hiện thực hàng đợi và cụm worker theo từng ngôn ngữ bằng `worker_threads`. Mỗi ngôn ngữ có cấu hình số worker tối đa và timeout riêng; khi hết worker rảnh, request được đưa vào queue (có giới hạn kích thước và timeout chờ).

(2) Worker thực thi theo ngôn ngữ

- Java (`utils/java-execution-worker.js`):
 - Tạo thư mục tạm theo session; trích xuất tên lớp public class ... để đặt tên file `.java`.
 - Biên dịch bằng `javac` với giới hạn bộ nhớ compiler; thời gian biên dịch bị khống chế (compile timeout) và dùng `SIGKILL` khi quá hạn.
 - Thực thi bằng java với giới hạn bộ nhớ JVM; input được pipe qua `stdin`; thu thập `stdout/stderr`; dọn dẹp thư mục tạm “aggressive” để tránh rò rỉ file.

- Python (utils/python-execution-worker.js):

- Ghi code vào main.py trong thư mục tạm; kích hoạt môi trường venv và chạy python3.

- Ép backend đồ họa của matplotlib về Agg để chạy trên server không có GUI; giới hạn timeout và buffer output; dọn dẹp thư mục tạm theo chu kỳ.

- Jupyter Notebook (utils/jupyter-execution-worker.js):

- Nhận notebook JSON (.ipynb), lưu vào file tạm và thực thi bằng jupyter nbconvert --execute --stdout.

- Trích xuất output theo từng cell (text, HTML) và thu thập ảnh biểu đồ (PNG/SVG) từ display_data/output.data, trả về cellOutputs và (nếu có) plotImage phục vụ hiển thị trực quan trên frontend.

- Thiết kế này cho phép hỗ trợ các học phần thực hành phân tích dữ liệu/đồ thị, nơi output không chỉ là text.

(3) Cơ chế chấm tự động theo test case

- Dữ liệu test case nằm trong Exam.testCases, mỗi test case có input, expectedOutput, isHidden, và weight (mở đường cho chấm theo trọng số).

- Khi submit, hệ thống chạy tuần tự các test case, so sánh output chuẩn hóa (trim) và ghi kết quả vào Submission.testResults. Điểm tự động được tính theo tỷ lệ test case đạt so với tổng số test và được lưu vào Submission.score.

- Khi run, hệ thống ưu tiên trả phản hồi nhanh: chạy theo input tùy chọn (console input) hoặc chạy một phần test case; phản hồi được cấu trúc hóa theo nhóm lỗi (validation, compilation, runtime).

(4) Quản trị tài nguyên và chống quá tải

Ngoài timeout theo worker, hệ thống còn áp dụng:

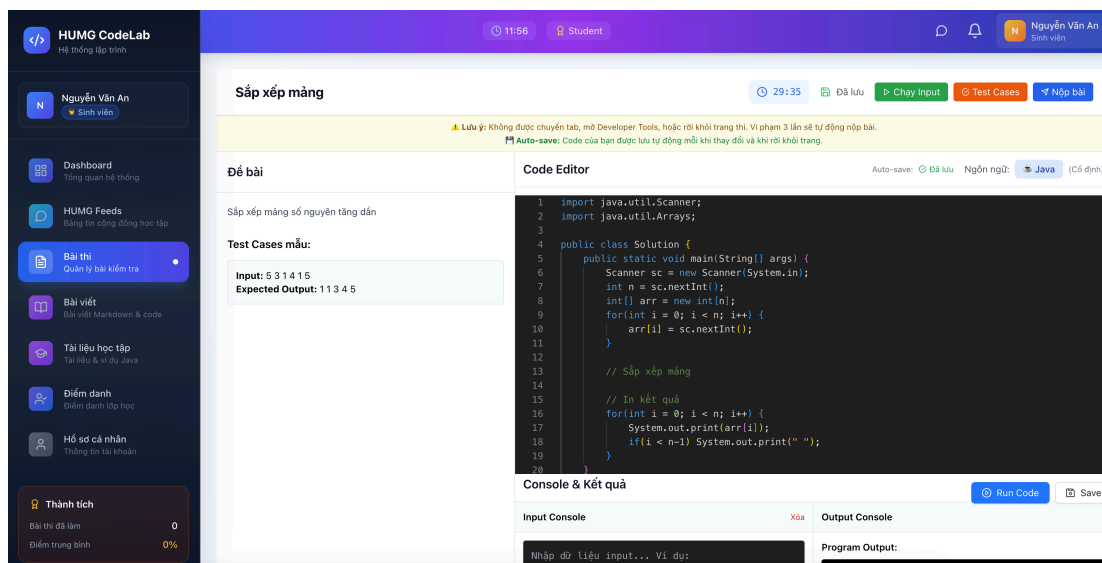
- Giới hạn kích thước output (buffer), giới hạn hàng đợi, và cơ chế từ chối yêu cầu khi hệ thống “under heavy load”.
- Tối ưu frontend bằng throttling khi người học bấm chạy code liên tục; đồng thời hạn chế phát realtime monitoring trong production để giảm tải Socket.io.

4.4 Frontend và trải nghiệm người dùng

Frontend được thiết kế theo hướng “tác vụ theo vai trò”, tối ưu vòng lặp học tập (viết–chạy–sửa) và tối ưu bối cảnh thi (giới hạn thao tác rủi ro, lưu nháp, đồng bộ)

(1) Trải nghiệm sinh viên khi làm bài/thi

- Trang làm bài (`pages/ExamTaking.js`) bố cục hai vùng: đề bài (Markdown) và editor + console.

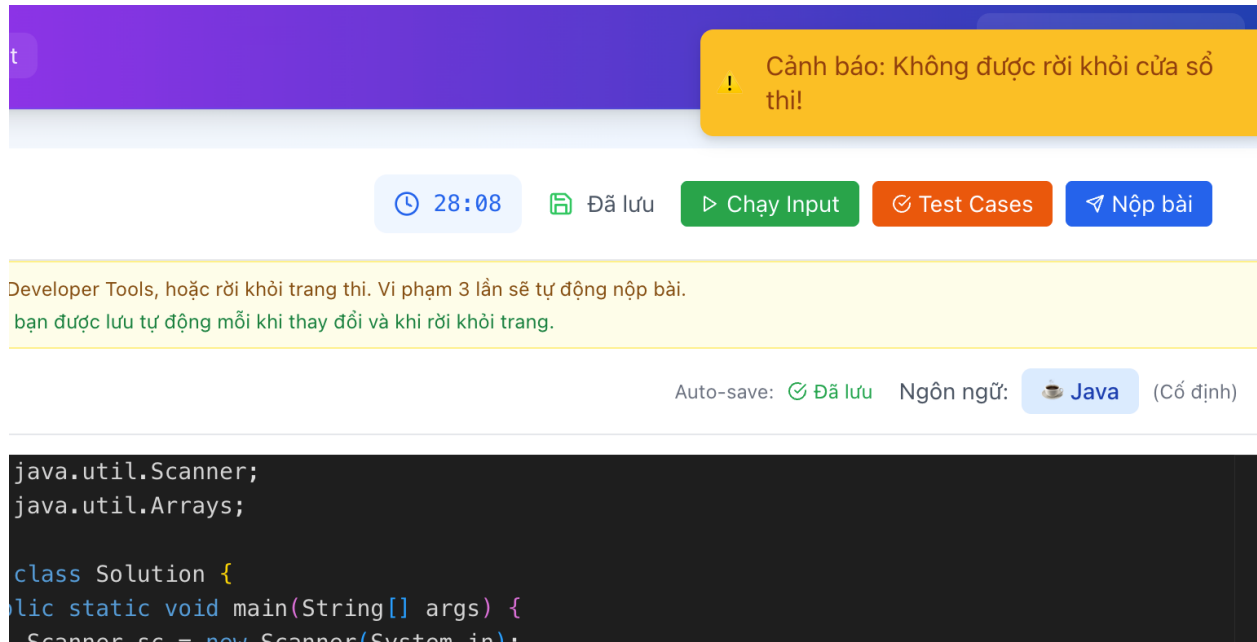


- Editor dùng `components/CodeEditor.js`:
- Với Java/Python: Monaco Editor chuẩn, hỗ trợ `onRun` và `onSave`.
- Với Jupyter: chế độ “notebook” cho phép tạo nhiều cell code/markdown, chạy từng cell hoặc chạy tất cả; output hiển thị ngay dưới cell, hỗ trợ ảnh plot (base64).

- Console hỗ trợ:
 - Input console (nhập stdin) và Output console (stdout/stderr).
 - Hiển thị kết quả test case (passed/failed) và hiển thị ảnh plot/cell outputs cho Jupyter.
- Tự động lưu nháp:
 - Lưu cục bộ (localStorage) để chống mất dữ liệu khi reload.
 - Lưu lên server theo cơ chế debounce (ví dụ 2 giây) qua API `/submissions/draft``.
- Đồng bộ thời gian làm bài:
 - Khi vào bài thi, client gọi `/submissions/exam/:examId/session/start``, định kỳ sync `/session`` để giảm sai lệch; khi rời trang, gọi `/session/end``.
 - Khi hết thời gian, hệ thống tự động nộp bài và điều hướng về danh sách bài thi.

(2) Cơ chế anti-cheat phía client

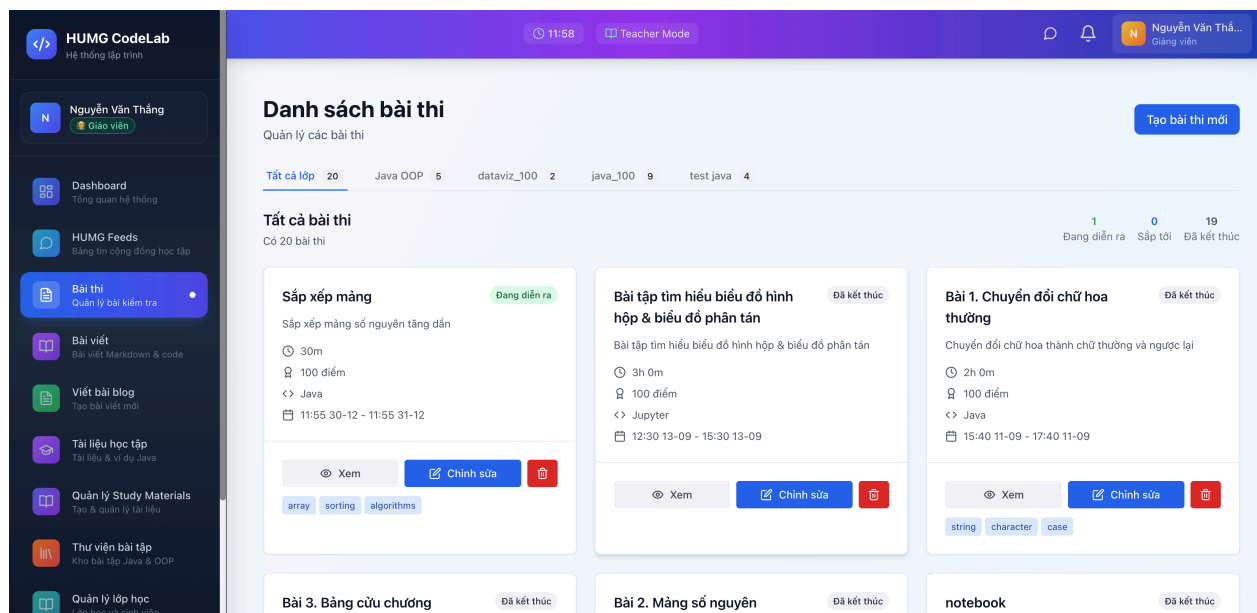
Trang thi giám sát các hành vi như chuyển tab/ẩn cửa sổ, mở dev tools (phím tắt), click chuột phải; các sự kiện được ghi nhận lên server qua API `/submissions/violation``. Đồng thời UI hiển thị cảnh báo theo mức độ và (khi đạt ngưỡng) tự động nộp bài.



(3) Trải nghiệm giảng viên

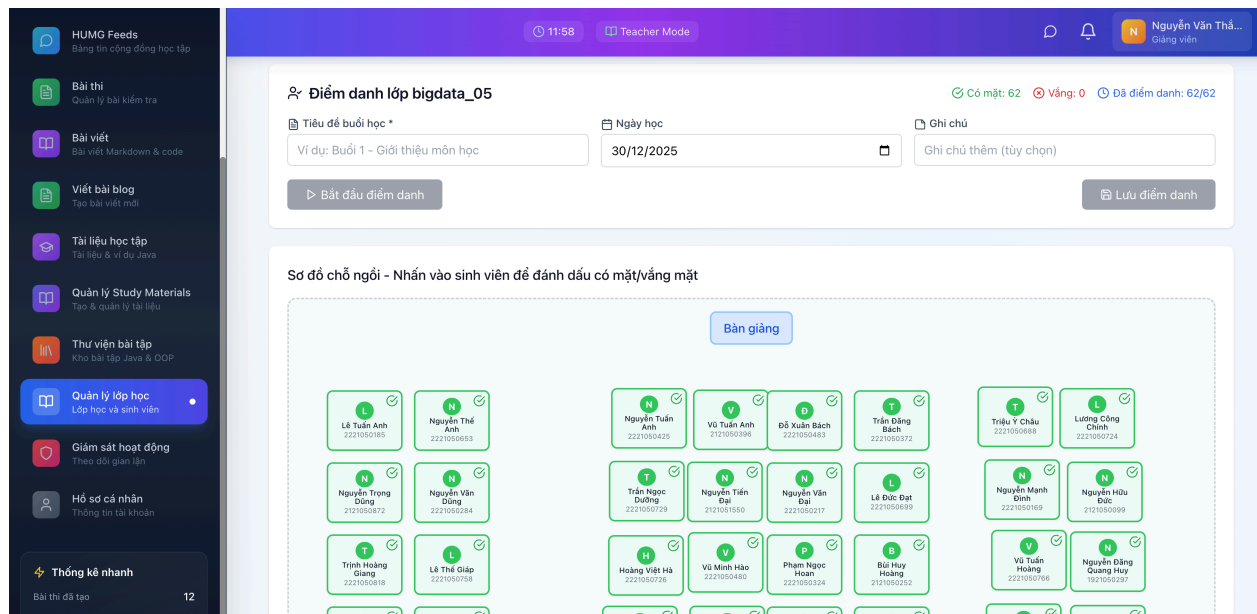
Các trang quản trị hỗ trợ:

- Tạo/sửa bài thi, cấu hình test case, cấu hình lớp/sinh viên được phép.



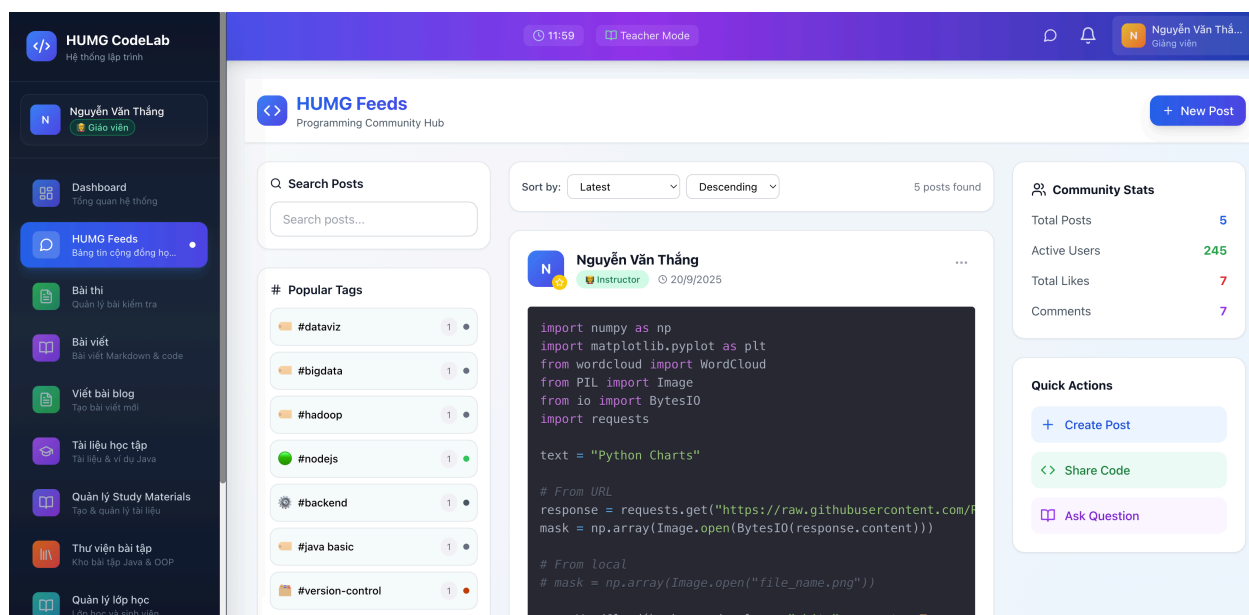
- Theo dõi bài nộp và thống kê; hỗ trợ xem sinh viên “đang làm” dựa trên draft và hoạt động; và xem tín hiệu vi phạm theo bài thi.

- Quản lý lớp học: import danh sách sinh viên bằng Excel/CSV; xuất danh sách; cấu hình sơ đồ chỗ ngồi (seating chart) phục vụ giám sát.
- Điểm danh realtime: giảng viên mở phiên điểm danh và xem cập nhật tức thời từ sinh viên.



(4) Tương tác cộng đồng (Feeds & Messaging)

- Feeds cung cấp trang chia sẻ nội dung học tập, hỗ trợ Markdown và chèn code; bình luận lẫn nhau và like.



- Messaging cung cấp nhấn tin realtime với trạng thái “đang soạn”, đã đọc/chưa đọc; tích hợp thông báo trình duyệt (khi được cấp quyền) và hỗ trợ push/PWA theo cấu hình hệ thống.

5. Kết luận và hướng phát triển

5.1. Kết luận

Nghiên cứu “Xây dựng hệ thống HUMG Codelab phục vụ các học phần lập trình” đã hoàn thành mục tiêu thiết kế và hiện thực một nền tảng web hỗ trợ dạy–học và đánh giá lập trình theo hướng chuẩn hóa môi trường thực hành và tăng tốc vòng phản hồi học tập. Trên phương diện chức năng, hệ thống cung cấp đầy đủ các thành phần cốt lõi của một codelab học thuật: quản lý bài thi/bài tập và test case; trình soạn thảo mã nguồn trực tuyến; chạy thử (run) và nộp bài (submit) với cơ chế chấm tự động theo bộ kiểm thử; lưu vết bài làm và kết quả; cùng các chức năng hỗ trợ quản trị lớp học và trao đổi học thuật.

Trên phương diện đánh giá và quản trị lớp học, nền tảng tích hợp kênh realtime và các cơ chế hỗ trợ minh bạch trong kiểm tra/thi lập trình, bao gồm theo dõi tiến độ, điểm danh realtime theo lớp và ghi nhận một số tín hiệu hành vi phục vụ chống

gian lận (anti-cheat). Các tín hiệu này được thiết kế với mục đích hỗ trợ giảng viên ra quyết định, đồng thời nhấn mạnh nguyên tắc minh bạch và thu thập tối thiểu dữ liệu cần thiết.

Một đóng góp nổi bật của hệ thống là khả năng mở rộng đa ngôn ngữ ở mức thực thi–phản hồi: bên cạnh Java (trọng tâm cho học phần OOP), hệ thống hỗ trợ Python và Jupyter Notebook với cơ chế trích xuất output theo cell và xử lý ảnh biểu đồ, qua đó mở rộng đáng kể phạm vi ứng dụng sang các học phần thực hành dữ liệu. Mặt khác, việc hiện thực mô hình worker pool và hàng đợi thực thi đã tạo nền tảng kỹ thuật quan trọng để kiểm soát đồng thời, giới hạn tài nguyên và giảm rủi ro quá tải khi phục vụ lớp đông người.

Tổng hợp lại, HUMG Codelab có giá trị ứng dụng trực tiếp trong tổ chức học phần lập trình tại HUMG: giúp giảm tải chấm bài, tăng tính kịp thời của phản hồi cho sinh viên, hỗ trợ giảng viên quan sát trạng thái lớp học theo thời gian thực và chuẩn hóa quy trình kiểm tra/thi lập trình. Đồng thời, hệ thống mở ra cơ hội tiếp tục hoàn thiện theo hướng nâng cao chất lượng đánh giá, tăng cường an toàn thực thi và mở rộng khả năng vận hành ở quy mô lớn.

5.2. Hướng phát triển

Dựa trên kết quả triển khai và các yêu cầu thực tiễn trong giảng dạy, hướng phát triển của hệ thống có thể tập trung theo các nhóm sau:

- Mở rộng ngôn ngữ và mô hình chấm: bổ sung thêm các ngôn ngữ phổ biến trong đào tạo (C/C++, JavaScript/TypeScript, SQL, Go...), đồng thời hoàn thiện cơ chế chấm theo trọng số test case và hỗ trợ các dạng so khớp linh hoạt (tolerance cho số thực, so khớp theo tập/định dạng). Với nội dung OOP, có thể bổ sung bài chấm theo “interface/contract” và bộ test nâng cao để kiểm tra thiết kế.

- Chấm theo rubric và đánh giá chất lượng mã: tích hợp công cụ phân tích tĩnh (linters, checkstyle/pylint), đo độ phức tạp, phát hiện code smell, và chấm theo rubric để đánh giá các khía cạnh ngoài correctness (phong cách, cấu trúc, hiệu quả). Đây là bước quan trọng để nâng chất lượng đánh giá trong các học phần hướng đối tượng và dự án.

- Tăng cường sandbox và an toàn thực thi: chuyển từ cơ chế hạn chế theo pattern và giới hạn tiến trình sang sandbox mạnh hơn bằng container (Docker) hoặc cơ chế cô lập cấp hệ điều hành, áp dụng giới hạn CPU/RAM/IO chặt chẽ, quản lý filesystem theo whitelist, và chuẩn hóa quy trình dọn dẹp tài nguyên. Điều này giúp giảm rủi ro bảo mật và tăng độ ổn định khi chạy code phức tạp.

- Nâng cao khả năng chịu tải và vận hành production: tối ưu pipeline run/submit theo hướng tách worker service, bổ sung cache cho dữ liệu đọc nhiều (đề thi/bài tập), triển khai multi-instance và adapter cho Socket.io khi cần mở rộng ngang; hoàn thiện giám sát (metrics, tracing), cảnh báo tự động và cơ chế “graceful degradation” khi quá tải.

- Phân tích học tập (Learning analytics) và dashboard: phát triển dashboard theo lớp/bài thi cho giảng viên dựa trên dữ liệu run/submit/draft/session time và vi phạm; cung cấp chỉ báo sự phạm như tiến độ, nhóm lỗi phổ biến, mức độ luyện tập, và gợi ý can thiệp. Với sinh viên, cung cấp bảng tiến độ cá nhân và gợi ý bài luyện tập phù hợp.

- Tăng cường chống gian lận và tính liêm chính học thuật: bổ sung phát hiện tương đồng mã nguồn (code plagiarism), phân tích hành vi theo chuỗi thời gian, đối chiếu đa nguồn (log hệ thống, sự kiện realtime) và cơ chế xuất báo cáo vi phạm theo minh chứng. Đồng thời cần xây dựng chính sách sử dụng dữ liệu rõ ràng nhằm đảm bảo quyền riêng tư.

- Cải thiện trải nghiệm học tập và mở rộng kênh học tập: hoàn thiện PWA/offline-friendly cho một số chức năng, tối ưu giao diện mobile, bổ sung “gợi ý theo bước” (scaffolded hints), thư viện lời giải mẫu có kiểm soát, và tích hợp hoạt động học tập xã hội (feeds/messaging) theo ngữ cảnh bài học.

Các hướng phát triển trên có thể được ưu tiên theo nhu cầu thực tế của từng học phần và hạ tầng triển khai. Trong ngắn hạn, ưu tiên các hạng mục liên quan đến an toàn thực thi, giới hạn tài nguyên và giám sát vận hành; trong trung hạn, mở rộng ngôn ngữ và hoàn thiện cơ chế chấm; trong dài hạn, phát triển analytics và các chức năng hỗ trợ sư phạm nâng cao.

TÀI LIỆU THAM KHẢO

- [1] T. T. M. Tiên, "HỆ THỐNG TỰ ĐỘNG CHẤM BÀI TẬP LẬP TRÌNH," *Journal of Transportation Science and Technology*, 2019.
- [2] E. Hagene, "What is social learning theory?," 2025. [Online]. Available: <https://frontcore.com/blog/what-is-social-learning-theory/>.