TRƯỜNG KỸ THUẬT VÀ CÔNG NGHỆ
KHOA CÔNG NGHỆ THÔNG TIN

THỰC TẬP ĐỒ ÁN CHUYÊN NGÀNH
HỌC KỲ I, NĂM HỌC 2025-2026

PHÁT TRIỂN HỆ THỐNG QUẢN LÝ DỰ ÁN VÀ CÔNG VIỆC NHÓM TRỰC TUYẾN

Giảng viên hướng dẫn:
ThS.Nguyễn Ngọc Đan Thanh

Sinh viên thực hiện:
Họ tên: Trần Quốc Đạm
MSSV:110122045
Lớp: DA22TTA

Vĩnh Long, tháng 12 năm 2025

TRƯỜNG KỸ THUẬT VÀ CÔNG NGHỆ
KHOA CÔNG NGHỆ THÔNG TIN

THỰC TẬP ĐỒ ÁN CHUYÊN NGÀNH
HỌC KỲ I, NĂM HỌC 2025-2026

PHÁT TRIỂN HỆ THỐNG QUẢN LÝ DỰ ÁN VÀ CÔNG VIỆC NHÓM TRỰC TUYẾN

Giảng viên hướng dẫn:
ThS.Nguyễn Ngọc Đan Thanh

Sinh viên thực hiện:
Họ tên: Trần Quốc Đạm
MSSV:110122045
Lớp: DA22TTA

Vĩnh Long, tháng 12 năm 2025

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Vĩnh Long, ngày ... tháng ... năm 2025
Giảng viên hướng dẫn
(Ký tên và ghi rõ họ tên)

NHẬN XÉT CỦA THÀNH VIÊN HỘI ĐỒNG

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Vĩnh Long, ngày ... tháng ... năm 2025
Thành viên hội đồng
(Ký tên và ghi rõ họ tên)

LỜI CẢM ƠN

Trước hết, em xin bày tỏ lòng biết ơn sâu sắc đến ThS. Nguyễn Ngọc Đan Thanh – giảng viên hướng dẫn đồ án chuyên ngành, người đã luôn tận tình hướng dẫn, định hướng đề tài và hỗ trợ em trong suốt quá trình thực hiện đồ án. Cô đã dành nhiều thời gian xem xét, chỉnh sửa nội dung báo cáo, góp ý chuyên môn và giải đáp kịp thời những khó khăn phát sinh trong quá trình nghiên cứu và triển khai đề tài. Sự tận tâm, nghiêm túc và kinh nghiệm của cô là yếu tố quan trọng giúp em hoàn thành đồ án đúng tiến độ và đạt được mục tiêu đề ra.

Bên cạnh đó, em xin cảm ơn các công cụ trí tuệ nhân tạo như ChatGPT, Gemini và Claude đã hỗ trợ em trong quá trình tham khảo tài liệu, hệ thống hóa kiến thức, gợi ý ý tưởng và cải thiện cách trình bày nội dung báo cáo. Các công cụ này đã góp phần nâng cao hiệu quả học tập và nghiên cứu. Tuy nhiên, toàn bộ nội dung trong đồ án đều do em chủ động lựa chọn, chỉnh sửa, kiểm chứng và chịu trách nhiệm hoàn toàn.

Cuối cùng, em xin gửi lời cảm ơn đến quý thầy cô Khoa Công nghệ Thông tin đã trang bị cho em những kiến thức nền tảng và kỹ năng cần thiết trong suốt quá trình học tập, tạo tiền đề để em hoàn thành tốt đồ án chuyên ngành này.

Em xin chân thành cảm ơn!

MỤC LỤC

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG BIỂU

TÓM TẮT ĐỒ ÁN CHUYÊN NGÀNH

Đề tài "Phát triển hệ thống quản lý dự án và công việc nhóm trực tuyến" nhằm giải quyết khó khăn trong việc quản lý, phân công và theo dõi tiến độ công việc của các nhóm nhỏ hoặc sinh viên khi các công cụ như Jira hay Trello còn phức tạp và khó sử dụng.

Đề tài tiếp cận bằng cách kết hợp nghiên cứu lý thuyết và thực nghiệm: tìm hiểu các nền tảng quản lý hiện có, nghiên cứu kiến trúc Microservices và các công nghệ ReactJS, NodeJS, MongoDB, Docker để xây dựng hệ thống web có tính mở rộng và hiệu quả cao.

Hệ thống được thiết kế theo mô hình Microservices với các chức năng quản lý người dùng, nhóm, dự án, công việc và thông báo. Giao diện thân thiện, dễ sử dụng; backend xử lý logic nghiệp vụ, giao tiếp với cơ sở dữ liệu qua API RESTful; toàn bộ hệ thống được triển khai thử nghiệm trên Docker.

Kết quả là xây dựng thành công hệ thống quản lý dự án và công việc nhóm trực tuyến với đầy đủ các chức năng như đăng ký, đăng nhập, tạo nhóm, phân công công việc, cập nhật tiến độ, bình luận và thống kê. Mã nguồn được quản lý rõ ràng trên GitHub, dễ dàng bảo trì và mở rộng. Giao diện hệ thống thân thiện, dễ sử dụng và hoạt động ổn định trong môi trường thử nghiệm.

MỞ ĐẦU

1. Lý do chọn đề tài

Trong bối cảnh chuyển đổi số đang diễn ra mạnh mẽ, hình thức làm việc nhóm và quản lý dự án trực tuyến đã trở thành xu hướng tất yếu, không chỉ tại các doanh nghiệp mà còn trong môi trường giáo dục đại học. Đối với sinh viên, đặc biệt là khối ngành công nghệ thông tin, việc thực hiện các đồ án nhóm yêu cầu sự phối hợp chặt chẽ, chia sẻ tài liệu và cập nhật tiến độ liên tục.

Tuy nhiên, thực trạng quản lý công việc nhóm của sinh viên hiện nay còn gặp nhiều bất cập. Đa số các nhóm vẫn sử dụng các công cụ giao tiếp tức thời như Zalo, Messenger để giao việc. Cách làm này dẫn đến việc thông tin bị trôi, khó theo dõi lịch sử chỉnh sửa và không có cái nhìn tổng quan về tiến độ dự án. Trong khi đó, các công cụ chuyên nghiệp như Jira hay Trello dù mạnh mẽ nhưng lại tạo ra rào cản lớn về độ phức tạp khi thiết lập và chi phí sử dụng các tính năng nâng cao.

Từ thực tế đó, đề tài "Phát triển hệ thống quản lý dự án và công việc nhóm trực tuyến" được lựa chọn nghiên cứu. Đề tài không chỉ giải quyết bài toán nghiệp vụ về một không gian làm việc tinh gọn, hiệu quả mà còn là cơ hội để nhóm sinh viên tiếp cận và làm chủ các công nghệ tiên tiến nhất hiện nay như kiến trúc Microservices, Docker và nền tảng NodeJS.

2. Mục đích nghiên cứu

Đề tài tập trung vào hai mục đích chính:

Về mặt khoa học và công nghệ: Nghiên cứu và ứng dụng kiến trúc vi dịch vụ để khắc phục nhược điểm của kiến trúc nguyên khối, giúp hệ thống dễ dàng mở rộng và bảo trì. Đồng thời, làm chủ quy trình phát triển phần mềm hiện đại với các công nghệ Docker.

Về mặt thực tiễn: Xây dựng thành công một ứng dụng web hoàn chỉnh, cho phép người dùng tạo nhóm, lập kế hoạch, phân công nhiệm vụ và theo dõi tiến độ trực quan, thay thế cho các phương pháp quản lý thủ công kém hiệu quả.

3. Đối tượng và phạm vi nghiên cứu

Đối tượng nghiên cứu:

Quy trình tổ chức và quản lý công việc nhóm trực tuyến: từ khâu khởi tạo dự án, lập kế hoạch, phân chia tác vụ đến theo dõi tiến độ.

Cơ chế tương tác và cộng tác giữa các thành viên trong cùng một không gian làm việc.

Các công nghệ phát triển ứng dụng web hiện đại và kiến trúc hệ thống vi dịch vụ.

Phạm vi nghiên cứu:

Hệ thống được xây dựng dưới dạng website.

Tập trung vào các chức năng cốt lõi cho nhóm quy mô nhỏ và vừa: quản lý người dùng, quản lý dự án, quản lý công việc theo trạng thái, bình luận và thông báo.

Cơ sở dữ liệu sử dụng MongoDB và triển khai thử nghiệm trên môi trường Docker.

4. Phương pháp nghiên cứu

Đề tài sử dụng phương pháp kết hợp giữa nghiên cứu lý thuyết và thực nghiệm:

Nghiên cứu lý thuyết: Tìm hiểu tài liệu về công nghệ ReactJS, NodeJS, kiến trúc vi dịch vụ và cơ sở dữ liệu NoSQL.

Phương pháp thực nghiệm: Phân tích yêu cầu, thiết kế hệ thống, lập trình xây dựng chức năng và kiểm thử hệ thống.

5. Cấu trúc báo cáo

Báo cáo đồ án được trình bày trong 5 chương, bao gồm:

Chương 1. Tổng quan nghiên cứu: Phân tích bối cảnh bài toán, đánh giá các giải pháp hiện có và đề xuất giải pháp của đề tài.

Chương 2. Nghiên cứu lý thuyết: Trình bày cơ sở lý thuyết về các công nghệ được sử dụng.

Chương 3. Hiện thực hóa nghiên cứu: Phân tích yêu cầu, thiết kế hệ thống và quy trình xây dựng ứng dụng.

Chương 4. Kết quả nghiên cứu: Trình bày các kết quả đạt được, hình ảnh giao diện và đánh giá hệ thống.

Chương 5. Kết luận và hướng phát triển: Tổng kết ưu, nhược điểm và định hướng mở rộng trong tương lai.

CHƯƠNG 1 TỔNG QUAN NGHIÊN CỨU

1.1 BỐI CẢNH VÀ BÀI TOÁN ĐẶT RA

Trong xu thế chuyển đổi số đang diễn ra sâu rộng, công nghệ thông tin ngày càng được ứng dụng mạnh mẽ trong nhiều lĩnh vực, đặc biệt là giáo dục đại học. Các mô hình học tập hiện đại không chỉ chú trọng đến việc tiếp thu kiến thức mà còn đề cao khả năng làm việc nhóm, tư duy dự án và kỹ năng cộng tác. Đối với sinh viên, đặc biệt là sinh viên các ngành kỹ thuật và công nghệ, việc tham gia các dự án học tập theo nhóm đã trở thành hình thức học tập phổ biến và mang tính bắt buộc trong nhiều học phần.

Cùng với sự gia tăng của các hoạt động học tập mang tính dự án, nhu cầu tổ chức, quản lý và theo dõi quá trình thực hiện công việc nhóm cũng ngày càng trở nên cấp thiết. Việc phân chia nhiệm vụ, giám sát tiến độ và phối hợp giữa các thành viên đòi hỏi phải có những phương thức quản lý khoa học và hiệu quả. Trong bối cảnh đó, các hệ thống quản lý dự án và công việc trực tuyến được xem là giải pháp quan trọng, giúp hỗ trợ tổ chức công việc một cách có cấu trúc, đảm bảo tính minh bạch và nâng cao hiệu quả cộng tác.

Tuy nhiên, việc áp dụng các hệ thống quản lý công việc vào môi trường sinh viên vẫn còn nhiều thách thức. Các giải pháp hiện có thường được thiết kế hướng tới doanh nghiệp hoặc các tổ chức chuyên nghiệp, dẫn đến sự không phù hợp về mức độ phức tạp, chi phí và cách thức sử dụng đối với sinh viên. Do đó, bài toán đặt ra là cần nghiên cứu và xây dựng một hệ thống quản lý dự án và công việc nhóm trực tuyến phù hợp với bối cảnh giáo dục đại học, vừa đáp ứng được nhu cầu tổ chức công việc, vừa đảm bảo tính đơn giản và dễ tiếp cận cho người học.

1.2 THỰC TRẠNG QUẢN LÝ CÔNG VIỆC NHÓM CỦA SINH VIÊN HIỆN NÀY

Thực tế cho thấy, việc quản lý công việc nhóm của sinh viên hiện nay chủ yếu dựa trên các phương thức mang tính thủ công và phân tán. Trong quá trình thực hiện các bài tập lớn hoặc đồ án, sinh viên thường tự tổ chức nhóm và phân công nhiệm vụ thông qua các công cụ giao tiếp tức thời. Mặc dù các công cụ này hỗ trợ trao đổi thông tin nhanh chóng nhưng lại không được thiết kế để phục vụ cho việc quản lý công việc có hệ thống trong thời gian dài.

Thông tin về nhiệm vụ, thời hạn hoàn thành và tiến độ thực hiện thường được trao đổi rời rạc, thiếu sự tổng hợp và lưu trữ tập trung. Điều này dẫn đến tình trạng khó theo dõi lịch sử công việc, khó xác định mức độ hoàn thành của từng thành viên cũng như tiến độ chung của dự án. Trong nhiều trường hợp, việc cập nhật trạng thái công việc phụ thuộc vào ý thức cá nhân, gây ra sự thiếu nhất quán và làm giảm hiệu quả phối hợp trong nhóm.

Bên cạnh đó, việc lưu trữ và chia sẻ tài liệu phục vụ học tập thường được thực hiện thông qua các nền tảng lưu trữ trực tuyến riêng lẻ. Tuy các nền tảng này hỗ trợ tốt cho việc chia sẻ dữ liệu, nhưng lại không gắn kết chặt chẽ với quá trình quản lý nhiệm vụ và tiến độ thực hiện công việc. Sự tách biệt giữa công cụ giao tiếp, công cụ lưu trữ tài liệu và công cụ theo dõi tiến độ khiến cho quá trình làm việc nhóm trở nên kém hiệu quả và khó kiểm soát.

Từ thực trạng trên có thể nhận thấy rằng, sinh viên hiện nay vẫn thiếu một hệ thống quản lý công việc nhóm mang tính tập trung, phù hợp với nhu cầu học tập và quy mô dự án trong môi trường giáo dục đại học. Việc chưa có một công cụ hỗ trợ chuyên biệt không chỉ ảnh hưởng đến hiệu quả thực hiện các dự án học tập mà còn hạn chế khả năng rèn luyện kỹ năng quản lý công việc và làm việc nhóm của sinh viên. Đây chính là cơ sở thực tiễn quan trọng để tiếp tục phân tích những tồn tại và hạn chế cụ thể trong quá trình quản lý công việc nhóm, được trình bày ở các mục tiếp theo.

1.3 NHỮNG TỒN TẠI VÀ HẠN CHẾ CỦA PHƯƠNG PHÁP HIỆN TẠI

Từ thực trạng quản lý công việc nhóm đã phân tích, có thể nhận thấy nhiều tồn tại và hạn chế ảnh hưởng trực tiếp đến hiệu quả học tập cũng như chất lượng thực hiện các dự án của sinh viên. Một trong những hạn chế lớn nhất là sự thiếu vắng một cơ chế quản lý công việc mang tính tập trung và có hệ thống. Khi thông tin liên quan đến nhiệm vụ, tiến độ và tài liệu bị phân tán trên nhiều công cụ khác nhau, nhóm sinh viên gặp khó khăn trong việc tổng hợp và nắm bắt bức tranh tổng thể của dự án.

Bên cạnh đó, việc thiếu công cụ theo dõi tiến độ một cách trực quan khiến cho quá trình đánh giá mức độ hoàn thành công việc trở nên kém chính xác. Trong nhiều trường hợp, nhóm chỉ nhận ra sự chậm trễ hoặc sai lệch trong phân công nhiệm vụ khi thời hạn hoàn thành đã đến gần. Điều này không chỉ gây áp lực về thời gian mà còn ảnh hưởng

đến chất lượng sản phẩm cuối cùng của dự án, đồng thời làm giảm hiệu quả phối hợp giữa các thành viên.

Một hạn chế khác là tính minh bạch và trách nhiệm cá nhân trong làm việc nhóm chưa được đảm bảo. Khi nhiệm vụ không được ghi nhận rõ ràng về người thực hiện, thời hạn và trạng thái, việc xác định trách nhiệm của từng thành viên trở nên mơ hồ. Điều này dễ dẫn đến tình trạng phân công công việc không đồng đều, một số thành viên phải đảm nhận khối lượng công việc lớn hơn trong khi những thành viên khác tham gia chưa tích cực, gây ảnh hưởng đến tinh thần làm việc chung của nhóm.

Ngoài ra, các phương pháp quản lý thủ công còn hạn chế khả năng lưu trữ và theo dõi lịch sử thực hiện công việc. Việc không có dữ liệu ghi nhận quá trình thay đổi, chỉnh sửa và hoàn thành nhiệm vụ khiến nhóm sinh viên khó rút ra kinh nghiệm cho các dự án sau. Đồng thời, giảng viên hướng dẫn cũng gặp khó khăn trong việc theo dõi và đánh giá quá trình làm việc của từng cá nhân trong nhóm, dẫn đến việc đánh giá kết quả học tập chưa phản ánh đầy đủ mức độ đóng góp thực tế.

Những tồn tại và hạn chế nêu trên cho thấy rằng, các phương pháp quản lý công việc nhóm hiện nay của sinh viên chưa đáp ứng được yêu cầu về tính hiệu quả, minh bạch và khả năng kiểm soát trong môi trường học tập theo dự án. Đây chính là cơ sở quan trọng để đặt ra yêu cầu cần có một giải pháp quản lý công việc nhóm trực tuyến phù hợp, nhằm khắc phục các hạn chế hiện tại và nâng cao chất lượng làm việc nhóm trong môi trường giáo dục đại học.

1.4 CÁC HỆ THỐNG QUẢN LÝ DỰ ÁN VÀ CÔNG VIỆC NHÓM HIỆN NAY

Trong bối cảnh nhu cầu quản lý công việc và dự án ngày càng gia tăng, nhiều hệ thống quản lý dự án và công việc nhóm đã được phát triển và đưa vào sử dụng rộng rãi. Các hệ thống này chủ yếu hướng đến việc hỗ trợ tổ chức công việc theo cấu trúc, cho phép người dùng lập kế hoạch, phân công nhiệm vụ, theo dõi tiến độ và phối hợp làm việc trong môi trường trực tuyến. Một số hệ thống tiêu biểu có thể kể đến như Trello, Jira, Asana hay Notion, vốn được sử dụng phổ biến trong các doanh nghiệp và tổ chức chuyên nghiệp.

Nhìn chung, các hệ thống quản lý dự án hiện có đều cung cấp các chức năng cốt lõi phục vụ quản lý công việc nhóm, bao gồm tạo dự án, phân chia nhiệm vụ theo từng giai đoạn, cập nhật trạng thái thực hiện và hỗ trợ trao đổi thông tin giữa các thành viên.

Nhờ đó, quá trình làm việc nhóm được tổ chức một cách rõ ràng, giảm thiểu sự phụ thuộc vào các phương thức quản lý thủ công và góp phần nâng cao hiệu quả phối hợp trong nhóm.

1.5 ĐÁNH GIÁ ƯU VÀ NHƯỢC ĐIỂM CỦA CÁC HỆ THỐNG

Về mặt ưu điểm, các hệ thống quản lý dự án và công việc nhóm hiện nay có tính hoàn thiện cao, được phát triển dựa trên kinh nghiệm thực tiễn và nhu cầu sử dụng của các tổ chức chuyên nghiệp. Giao diện trực quan, khả năng theo dõi tiến độ theo thời gian thực và các công cụ báo cáo, thống kê giúp người quản lý dễ dàng nắm bắt tình hình thực hiện dự án. Ngoài ra, nhiều hệ thống còn hỗ trợ làm việc đa nền tảng, cho phép người dùng truy cập và cập nhật công việc mọi lúc, mọi nơi.

Tuy nhiên, bên cạnh những ưu điểm đó, các hệ thống hiện có cũng bộc lộ một số hạn chế khi áp dụng vào môi trường sinh viên. Phần lớn các nền tảng được thiết kế với mục tiêu phục vụ doanh nghiệp, dẫn đến quy trình sử dụng tương đối phức tạp và đòi hỏi người dùng phải có thời gian làm quen. Một số tính năng nâng cao chỉ được cung cấp trong các gói trả phí, gây khó khăn cho sinh viên trong quá trình tiếp cận và sử dụng lâu dài.

Bên cạnh đó, mức độ tùy chỉnh của các hệ thống hiện có đối với nhu cầu học tập và quản lý dự án quy mô nhỏ còn hạn chế. Các chức năng đôi khi vượt quá yêu cầu thực tế của sinh viên, trong khi một số nhu cầu đặc thù trong môi trường giáo dục chưa được hỗ trợ đầy đủ. Điều này khiến việc áp dụng các hệ thống quản lý dự án chuyên nghiệp vào hoạt động học tập chưa đạt được hiệu quả tối ưu.

Từ những phân tích trên có thể thấy rằng, mặc dù các hệ thống quản lý dự án và công việc nhóm hiện có mang lại nhiều lợi ích trong việc tổ chức và điều phối công việc nhưng vẫn chưa thực sự phù hợp với đặc thù và điều kiện sử dụng của sinh viên. Đây chính là cơ sở để đề tài tiếp tục nghiên cứu và đề xuất một giải pháp quản lý công việc nhóm trực tuyến đơn giản, linh hoạt và phù hợp hơn với môi trường giáo dục đại học.

1.6 ĐỊNH HƯỚNG GIẢI PHÁP CỦA ĐỀ TÀI

Từ việc phân tích bối cảnh, thực trạng cũng như đánh giá các hệ thống quản lý dự án và công việc nhóm hiện có, có thể nhận thấy nhu cầu cấp thiết về một giải pháp quản lý công việc nhóm trực tuyến phù hợp với môi trường sinh viên. Giải pháp được đề xuất

trong đề tài này hướng tới việc xây dựng một hệ thống có khả năng hỗ trợ tổ chức và quản lý công việc một cách hiệu quả, đồng thời đảm bảo tính đơn giản, dễ sử dụng và phù hợp với quy mô các dự án học tập trong giáo dục đại học.

Hệ thống được định hướng tập trung vào các chức năng cốt lõi phục vụ làm việc nhóm, bao gồm quản lý người dùng, quản lý dự án, phân công và theo dõi công việc theo trạng thái, hỗ trợ trao đổi thông tin giữa các thành viên và cung cấp thông báo kịp thời về tiến độ thực hiện. Việc tối giản các chức năng không cần thiết nhằm giảm độ phức tạp trong quá trình sử dụng, giúp sinh viên dễ dàng tiếp cận và áp dụng vào thực tế học tập.

Bên cạnh đó, hệ thống cũng được định hướng phát triển theo kiến trúc hiện đại, cho phép mở rộng và bảo trì thuận tiện trong tương lai. Mặc dù các yếu tố công nghệ cụ thể sẽ được trình bày chi tiết ở các chương tiếp theo, nhưng định hướng chung của đề tài là xây dựng một nền tảng linh hoạt, có khả năng thích ứng với sự thay đổi của nhu cầu người dùng và hỗ trợ tốt cho việc nghiên cứu, học tập cũng như thực hành phát triển phần mềm của sinh viên.

1.7 KẾT LUẬN CHƯƠNG

Chương 1 đã trình bày tổng quan về bối cảnh và bài toán đặt ra trong việc quản lý công việc nhóm trực tuyến trong môi trường sinh viên. Thông qua việc phân tích thực trạng, chỉ ra những tồn tại và hạn chế của các phương pháp quản lý hiện nay, chương này đã làm rõ sự cần thiết của việc xây dựng một hệ thống quản lý dự án và công việc nhóm phù hợp với đặc thù của giáo dục đại học.

Bên cạnh đó, việc khảo sát và đánh giá các hệ thống quản lý dự án hiện có đã cho thấy những ưu điểm cũng như những hạn chế khi áp dụng vào môi trường sinh viên, từ đó tạo cơ sở khoa học và thực tiễn cho việc đề xuất giải pháp của đề tài. Những nội dung đã trình bày trong chương này đóng vai trò là nền tảng để triển khai các nghiên cứu tiếp theo.

Trên cơ sở đó, Chương 2 sẽ tập trung trình bày các cơ sở lý thuyết và công nghệ liên quan đến việc xây dựng hệ thống quản lý dự án và công việc nhóm trực tuyến, làm tiền đề cho quá trình phân tích, thiết kế và hiện thực hóa hệ thống trong các chương tiếp theo.

CHƯƠNG 2 NGHIÊN CỨU LÝ THUYẾT

2.1 TỔNG QUAN PHÁT TRIỂN ỨNG DỤNG WEB

2.1.1 Giới thiệu về website và hệ thống web động

2.1.1.1 Khái niệm website

Website là một tập hợp các trang web được xây dựng bằng ngôn ngữ HTML và các công nghệ liên quan, có thể được truy cập thông qua trình duyệt web bằng một địa chỉ cụ thể trên mạng toàn cầu.

Một website sử dụng giao thức HTTP để truyền và chia sẻ dữ liệu, cho phép người dùng truy cập, xem nội dung và tương tác thông qua các liên kết siêu văn bản.

Nội dung của website có thể bao gồm văn bản, hình ảnh, âm thanh, video và nhiều định dạng thông tin khác, tất cả được lưu trữ trên máy chủ và kết nối với Internet.[1]

2.1.1.2 Phân loại website

Web tĩnh là loại website không có hệ thống quản lý nội dung, nội dung được lưu cố định trong các tệp HTML, CSS và JavaScript. Người dùng thông thường không thể chỉnh sửa nội dung mà phải thay đổi trực tiếp trong mã nguồn. Web tĩnh thường có tốc độ truy cập nhanh, chi phí xây dựng thấp và thân thiện với công cụ tìm kiếm. Tuy nhiên, loại web này khó quản lý, khó cập nhật và mở rộng, nên hiện nay chủ yếu được sử dụng cho các trang giới thiệu nhỏ, ít thay đổi nội dung hoặc trang one-page đơn giản.

Web động là website có hệ thống quản lý nội dung cho phép người dùng dễ dàng cập nhật và chỉnh sửa thông tin trên trang. Loại website này được xây dựng bằng HTML, CSS, JavaScript kết hợp với ngôn ngữ lập trình phía server như PHP, NodeJS cùng cơ sở dữ liệu như MySQL hoặc MongoDB. Web động dễ quản lý, mở rộng và hỗ trợ tương tác người dùng, phù hợp cho các website thương mại điện tử, tin tức, blog hoặc hệ thống quản lý thông tin. Tuy nhiên, chi phí và yêu cầu kỹ thuật cao hơn so với web tĩnh.[1]

2.1.1.3 Vai trò của website

Website giữ vai trò là cầu nối giữa hệ thống và người dùng, giúp truy cập, quản lý và trao đổi thông tin một cách nhanh chóng, trực quan và hiệu quả. Trong một hệ thống hiện đại, website hoạt động như giao diện tương tác chính, nơi người dùng thực hiện các chức năng như đăng nhập, quản lý dữ liệu hoặc trao đổi thông tin.[1]

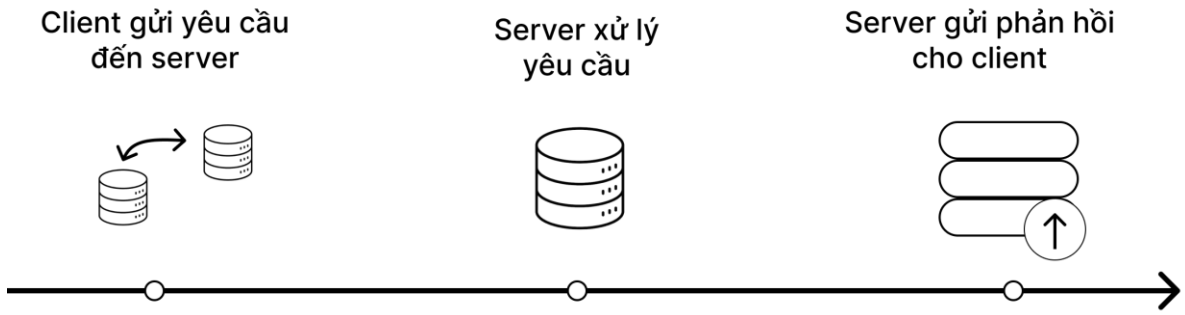
Bên cạnh đó, website còn là kênh truyền thông và quảng bá thông tin của tổ chức, giúp giới thiệu dịch vụ, sản phẩm và hình ảnh đến người dùng trên Internet. Nhờ khả năng truy cập mọi lúc, mọi nơi, website góp phần tự động hóa quy trình làm việc, tăng hiệu quả quản lý, giảm chi phí vận hành và nâng cao trải nghiệm người dùng.

Trong hệ thống quản lý dự án và công việc nhóm, website đóng vai trò trung tâm, giúp kết nối các thành viên, quản lý nhiệm vụ, theo dõi tiến độ và lưu trữ dữ liệu một cách tập trung, đảm bảo hoạt động nhóm diễn ra hiệu quả và đồng bộ.

2.1.2 Kiến trúc Client – Server

2.1.2.1 Tổng quan kiến trúc Client – Server

Kiến trúc Client–Server là mô hình kiến trúc trong đó hệ thống được chia thành hai thành phần chính là client và server. Client chịu trách nhiệm hiển thị giao diện và tương tác với người dùng, trong khi server đảm nhiệm việc xử lý nghiệp vụ, quản lý dữ liệu và cung cấp dịch vụ cho client thông qua các giao thức mạng. Mô hình này được sử dụng rộng rãi trong các ứng dụng web hiện đại nhờ khả năng mở rộng, dễ bảo trì và tách biệt rõ ràng giữa các thành phần.[2]



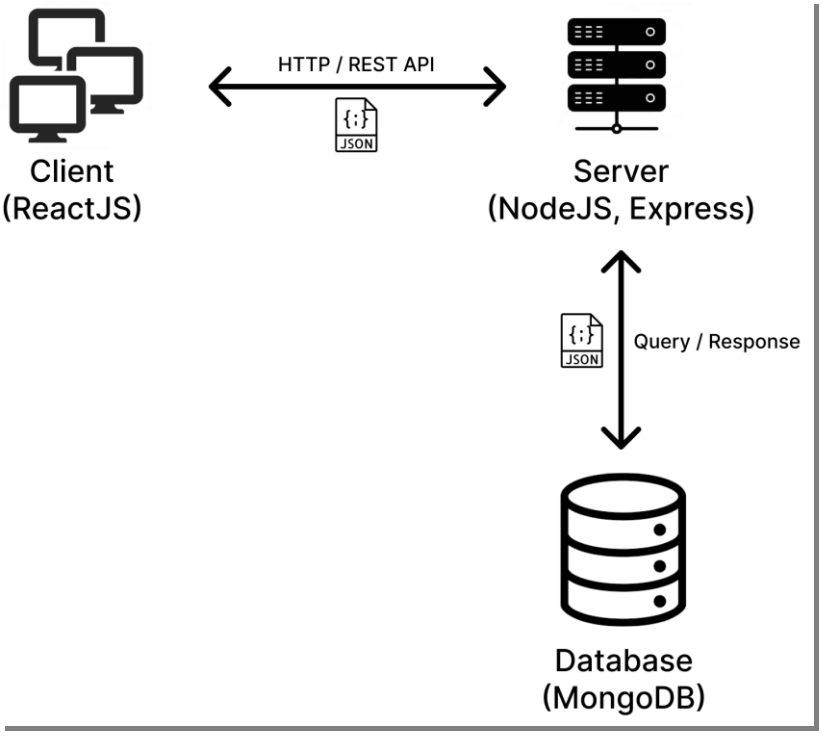
Hình 2.1 Cách hoạt động của mô hình Client – Server

2.1.2.2 Mô hình Client – Server trong hệ thống

Trong hệ thống quản lý dự án và công việc nhóm trực tuyến, phía client được triển khai dưới dạng ứng dụng web sử dụng ReactJS. Client có nhiệm vụ tiếp nhận thao tác của người dùng, hiển thị thông tin dự án, công việc và gửi các yêu cầu xử lý đến server thông qua giao thức HTTP. Dữ liệu trao đổi giữa client và server được định dạng dưới dạng JSON.

Phía server được xây dựng trên nền tảng NodeJS và ExpressJS, chịu trách nhiệm xử lý logic nghiệp vụ, xác thực người dùng, truy vấn cơ sở dữ liệu MongoDB và trả kết

quả về cho Client. Các chức năng của server được cung cấp dưới dạng các RESTful API, cho phép client và các dịch vụ khác trong hệ thống có thể giao tiếp một cách thống nhất và hiệu quả.



Hình 2.2 Mô hình client - server trong hệ thống

2.1.2.3 Lợi ích của kiến trúc Client – Server đối với đề tài

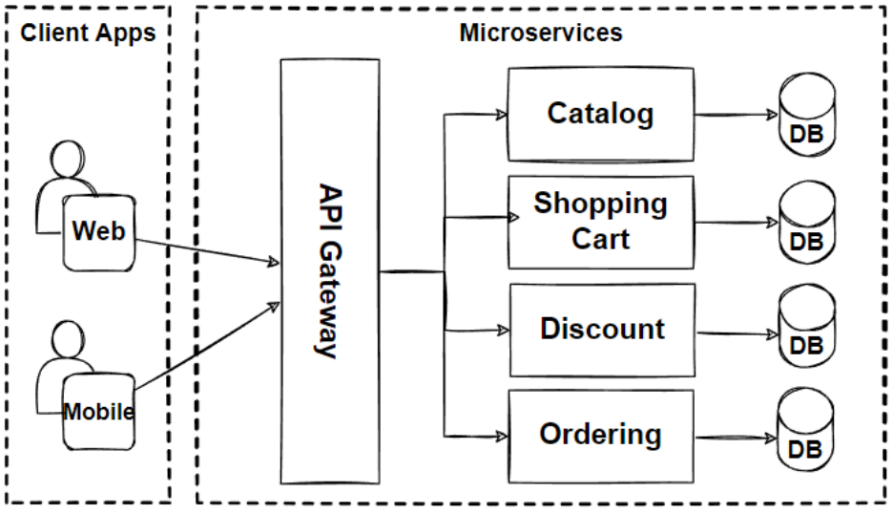
Việc áp dụng kiến trúc Client – Server giúp hệ thống tách biệt rõ ràng giữa giao diện người dùng và xử lý nghiệp vụ, từ đó đơn giản hóa quá trình phát triển và bảo trì. Kiến trúc này cho phép mở rộng linh hoạt cả phía client lẫn server mà không ảnh hưởng đến toàn bộ hệ thống. Đồng thời, mô hình cũng tạo nền tảng phù hợp để triển khai RESTful API và kiến trúc vi dịch vụ, đáp ứng tốt yêu cầu xây dựng hệ thống quản lý công việc nhóm trực tuyến trong môi trường học tập và làm việc hiện đại.[2]

2.1.3 Kiến trúc vi dịch vụ

2.1.3.1 Khái niệm

Kiến trúc vi dịch vụ (Microservices) là một phong cách thiết kế phần mềm, trong đó hệ thống được chia thành nhiều dịch vụ nhỏ, độc lập, mỗi dịch vụ đảm nhiệm một chức năng nghiệp vụ riêng biệt. Mỗi dịch vụ thường có mã nguồn riêng, cơ sở dữ liệu riêng và có thể được phát triển, triển khai, vận hành độc lập với các dịch vụ khác.[3]

Các dịch vụ trong kiến trúc cần trao đổi thông tin với nhau để phối hợp hoàn thành các chức năng của ứng dụng. Một số cách phổ biến như là REST API, gRPC và Message/Event.



Hình 2.3 Ví dụ về kiến trúc vi dịch vụ

2.1.3.2 Lợi ích và hạn chế của kiến trúc vi dịch vụ

A. Lợi ích

Dễ mở rộng quy mô: Mỗi dịch vụ có thể được phát triển, triển khai và nâng cấp độc lập, giúp việc tăng hoặc giảm sức mạnh của máy chủ (scale up, scale down) theo nhu cầu trở nên linh hoạt và nhanh chóng mà không ảnh hưởng đến toàn hệ thống.

Khả năng chịu lỗi cao: Khi một dịch vụ gặp sự cố, các dịch vụ khác vẫn hoạt động bình thường, giảm thiểu nguy cơ sập toàn bộ ứng dụng.

Dễ hiểu và quản lý mã nguồn: Mỗi microservice đảm nhận một chức năng riêng biệt, làm cho mã nguồn và logic nghiệp vụ trở nên đơn giản, dễ bảo trì hơn so với kiến trúc nguyên khối.

Cho phép thử nghiệm và sử dụng công nghệ đa dạng: Các dịch vụ có thể được phát triển bằng các ngôn ngữ lập trình khác nhau, sử dụng cơ sở dữ liệu riêng biệt, hỗ trợ thử nghiệm nhanh và áp dụng công nghệ mới.

Triển khai độc lập: Các dịch vụ có thể được cập nhật hoặc thay thế mà không cần ảnh hưởng đến các phần còn lại của hệ thống, giúp đẩy nhanh quá trình phát triển và triển khai.

Hỗ trợ tự động hóa quy trình: Thích hợp để áp dụng các công cụ tự động hóa trong quá trình build, deploy và giám sát các dịch vụ.[3]

B. Hạn chế:

Giao tiếp phức tạp giữa các dịch vụ: Việc phân tách thành nhiều dịch vụ nhỏ tạo ra sự phụ thuộc qua các giao thức mạng, làm tăng độ trễ và rủi ro xảy ra lỗi kết nối, cũng như khó khăn trong đồng bộ dữ liệu.

Tăng yêu cầu tài nguyên: Nhiều dịch vụ chạy độc lập yêu cầu phân bổ tài nguyên nhiều hơn so với hệ thống đơn khối, đồng thời tăng lượng cơ sở dữ liệu và hệ thống log cần quản lý.

Khó khăn trong kiểm thử và sửa lỗi toàn cục: Việc kiểm thử và gỡ lỗi phải thực hiện đồng thời trên nhiều dịch vụ riêng biệt, càng làm tăng sự phức tạp và thời gian phát triển.

2.1.3.3 API Gateway trong kiến trúc vi dịch vụ

Trong kiến trúc vi dịch vụ, mỗi ứng dụng thường gồm nhiều dịch vụ nhỏ, mỗi dịch vụ có API riêng để giao tiếp. Khi người dùng hoặc ứng dụng khác muốn truy cập dữ liệu hoặc chức năng, nếu gọi trực tiếp từng dịch vụ sẽ rất phức tạp và khó quản lý. Đây là lý do xuất hiện API Gateway.

API Gateway là một lớp trung gian đứng giữa client và các dịch vụ. Nó có vai trò như cổng chính để tất cả các yêu cầu đi vào hệ thống. Thay vì gọi từng dịch vụ riêng lẻ, người dùng chỉ cần gọi API Gateway, gateway sẽ chuyển tiếp yêu cầu đến đúng dịch vụ và trả kết quả về.

Nó còn đảm nhận nhiều nhiệm vụ quan trọng giúp hệ thống Microservices hoạt động hiệu quả và dễ quản lý:

Xác thực và phân quyền: API Gateway kiểm tra xem người dùng có quyền truy cập vào dịch vụ hay không, đảm bảo chỉ những yêu cầu hợp lệ mới được xử lý.

Tích hợp dữ liệu: Khi một người dùng cần thông tin từ nhiều dịch vụ, API Gateway có thể kết hợp dữ liệu từ các dịch vụ khác nhau và trả về chỉ trong một lần gọi, giúp client không phải gọi nhiều lần.

Giảm tải và tối ưu hiệu năng: API Gateway có thể lưu tạm dữ liệu để trả lời nhanh hơn và giới hạn số lượng yêu cầu từ người dùng, tránh tình trạng quá tải.

Ghi log và giám sát: Theo dõi mọi yêu cầu đi qua, giúp quản lý, phát hiện lỗi và phân tích hiệu suất của hệ thống dễ dàng hơn.

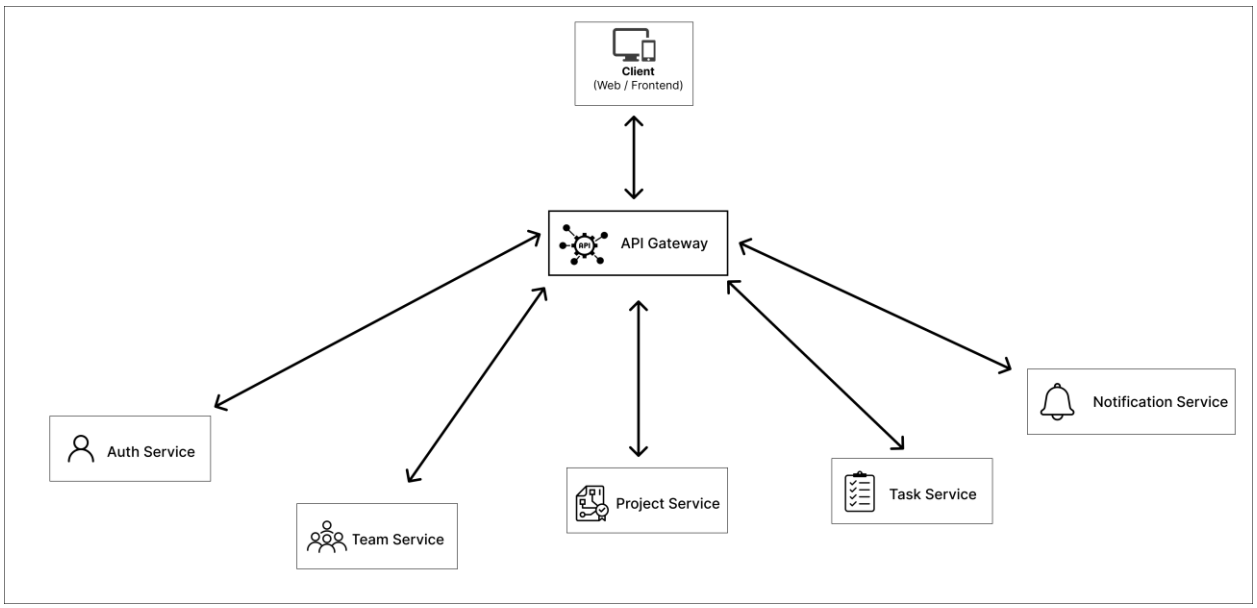
2.1.3.4 Giao tiếp giữa các dịch vụ trong kiến trúc

Trong kiến trúc vi dịch vụ, mỗi dịch vụ được xây dựng và triển khai một cách độc lập, đảm nhiệm một chức năng nghiệp vụ riêng biệt. Do đó, để hệ thống hoạt động như một thể thống nhất, các dịch vụ cần có cơ chế giao tiếp với nhau nhằm trao đổi dữ liệu và phối hợp xử lý nghiệp vụ. Cách thức giao tiếp giữa các dịch vụ là một trong những yếu tố quan trọng, ảnh hưởng trực tiếp đến hiệu năng, khả năng mở rộng và tính linh hoạt của toàn bộ hệ thống.

Về mặt lý thuyết, giao tiếp giữa các microservice có thể được thực hiện theo hai hình thức chính là giao tiếp đồng bộ và giao tiếp bất đồng bộ. Trong giao tiếp đồng bộ, một dịch vụ gửi yêu cầu và chờ phản hồi từ dịch vụ khác trước khi tiếp tục xử lý. Ngược lại, giao tiếp bất đồng bộ cho phép các dịch vụ trao đổi thông tin thông qua các cơ chế trung gian như hàng đợi hoặc hệ thống sự kiện, giúp giảm sự phụ thuộc trực tiếp giữa các dịch vụ. Việc lựa chọn hình thức giao tiếp phụ thuộc vào đặc thù nghiệp vụ và quy mô của hệ thống.

Trong phạm vi đề tài này, giao tiếp giữa các dịch vụ được triển khai theo mô hình giao tiếp đồng bộ thông qua RESTful API. Nó là một kiểu kiến trúc giao tiếp dựa trên giao thức HTTP, trong đó mỗi dịch vụ cung cấp các tài nguyên dưới dạng các endpoint và sử dụng các phương thức HTTP tiêu chuẩn như GET, POST, PUT và DELETE để thao tác dữ liệu. Dữ liệu trao đổi giữa các dịch vụ thường được biểu diễn dưới dạng JSON, giúp đảm bảo tính nhẹ, dễ đọc và dễ tích hợp.

Việc lựa chọn RESTful API trong kiến trúc vi dịch vụ của đề tài xuất phát từ nhiều yếu tố. Trước hết, RESTful API có cách triển khai đơn giản, dễ hiểu và phù hợp với các hệ thống có quy mô vừa và nhỏ như môi trường học tập của sinh viên. Bên cạnh đó, RESTful API tương thích tốt với nền tảng web và các công nghệ phát triển hiện đại, đồng thời hỗ trợ khả năng mở rộng và bảo trì hệ thống trong tương lai. Do đó, việc sử dụng RESTful API làm phương thức giao tiếp giữa các dịch vụ giúp hệ thống đảm bảo được tính linh hoạt, hiệu quả và phù hợp với mục tiêu nghiên cứu của đề tài.



Hình 2.4 Giao tiếp giữa các dịch vụ trong hệ thống

2.2 CÔNG NGHỆ VÀ FRAMEWORK BACK-END

2.2.1 NodeJS

2.2.1.1 Giới thiệu về NodeJS

Node.js là một công cụ giúp chạy JavaScript trên máy chủ thay vì chỉ chạy trên trình duyệt như bình thường. Nhờ Node.js có thể viết chương trình để làm website, gửi dữ liệu, tạo các ứng dụng chat, trò chơi trực tuyến hay API cho điện thoại và web bằng JavaScript. Điều này có nghĩa là cùng một ngôn ngữ JavaScript có thể dùng cho cả phần hiển thị trên web (front-end) và phần xử lý phía máy chủ (back-end), giúp việc lập trình dễ dàng và đồng nhất hơn.[4]

2.2.1.2 Những điểm quan trọng của NodeJS

Node.js làm việc theo kiểu bất đồng bộ (asynchronous) nghĩa là khi nhận một yêu cầu từ người dùng, nó sẽ không ngồi chờ xử lý xong mà tiếp tục nhận các yêu cầu khác, sau đó trả kết quả khi đã xong. Điều này giúp Node.js xử lý được nhiều người dùng cùng lúc mà không bị chậm.

Node.js chạy trên một luồng duy nhất nhưng vẫn có thể xử lý nhiều việc cùng lúc nhờ cách quản lý thông minh. Nó đi kèm với npm (Node Package Manager) là một kho thư viện chứa hàng nghìn công cụ có sẵn, giúp lập trình viên tái sử dụng code và phát triển nhanh hơn.

Node.js phù hợp với các ứng dụng cần nhiều kết nối cùng lúc hoặc thời gian thực như chat trực tuyến, xem video trực tuyến hoặc các API trả dữ liệu cho ứng dụng khác.

2.2.1.3 Cách hoạt động cơ bản

Khi một người dùng mở website hoặc gửi yêu cầu, Node.js sẽ ghi lại yêu cầu đó, xử lý các công việc cần thiết, rồi gửi lại kết quả. Nó làm việc theo cơ chế vòng lặp nghĩa là cứ khi có kết quả xong, Node.js sẽ gửi dữ liệu về người dùng mà không làm gián đoạn các yêu cầu khác.

Ví dụ:

```
const { createServer } = require('node:http');
const hostname = '127.0.0.1';
const port = 3000;
const server = createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World');
});
server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
``` :contentReference[oaicite:12]{index=12}
```

*2.2.1.4 Lợi ích và hạn chế*

Node.js giúp trang web và ứng dụng chạy nhanh, xử lý nhiều người dùng cùng lúc mà không bị chậm. Chỉ cần học JavaScript là có thể lập trình cả phần hiển thị và phần xử lý trên máy chủ, giúp việc lập trình đơn giản hơn.

Thư viện npm có sẵn rất nhiều công cụ, tiết kiệm thời gian phát triển và dễ xây dựng các ứng dụng hiện đại như chat, game trực tuyến, API.

Tuy nhiên, nó không phù hợp cho những công việc nặng tính toán như xử lý hình ảnh lớn hoặc trí tuệ nhân tạo vì chỉ dùng một luồng chính. Nếu lập trình viên không cẩn thận, code bất đồng bộ có thể khó đọc hoặc gây rối.

Ngoài ra, vì có quá nhiều thư viện, việc chọn những thư viện ổn định để sử dụng lâu dài cũng cần cân nhắc kỹ.
```

2.2.2 ExpressJS

2.2.2.1 Giới thiệu về ExpressJS

ExpressJS là một framework nhẹ cho Node.js, được phát triển nhằm giúp lập trình viên xây dựng các ứng dụng web và API một cách nhanh chóng và hiệu quả. Nó cung cấp các công cụ cơ bản để quản lý server HTTP, xử lý các yêu cầu và phản hồi, định tuyến và sử dụng middleware để thêm các chức năng mở rộng như xác thực, xử lý dữ liệu từ người dùng hay quản lý các tập tin tĩnh.

ExpressJS hoạt động như một lớp trung gian giữa Node.js và ứng dụng, giúp giảm bớt lượng code cần viết từ đầu và tăng tính linh hoạt trong việc tổ chức dự án. Nhờ đó có thể tập trung vào logic nghiệp vụ thay vì các chi tiết phức tạp của server.

2.2.2.2 Đặc điểm nổi bật của ExpressJS

Một trong những ưu điểm của ExpressJS là hệ thống định tuyến rất linh hoạt, giúp lập trình viên dễ dàng tạo các đường dẫn khác nhau cho từng chức năng trong ứng dụng. ExpressJS còn cho phép xử lý các dữ liệu được gửi kèm theo đường dẫn, như các tham số trong URL hoặc tham số truy vấn, để ứng dụng biết cách xử lý yêu cầu của người dùng một cách chính xác. Nói một cách đơn giản, hệ thống định tuyến trong ExpressJS giống như một người hướng dẫn, phân loại các yêu cầu và gửi chúng đến đúng phần của ứng dụng để xử lý.

Hệ thống middleware của ExpressJS cho phép thêm các chức năng bổ sung vào quá trình xử lý của ứng dụng. Nhờ middleware, ứng dụng có thể thực hiện các công việc như xác thực người dùng, kiểm tra dữ liệu, quản lý lỗi hoặc ghi nhật ký hoạt động. ExpressJS rất nhẹ và tối giản, không bắt buộc lập trình viên phải theo một cấu trúc cố định, giúp tự do sắp xếp mã nguồn theo ý muốn và dễ dàng kết hợp với các thư viện bên ngoài để mở rộng chức năng.

2.2.2.3 Cách hoạt động cơ bản

Cấu hình một ứng dụng Express thường bắt đầu bằng việc thêm thư viện, tạo app, định tuyến và lắng nghe cổng:

```
const express = require('express');
const app = express();
app.get('/', (req, res) => {
```

```
  res.send('Hello World');
});
app.listen(3000, () => {
  console.log('Server is running on http://localhost:3000');
});
```

Khi có request HTTP tới, Express sẽ tìm middleware và route phù hợp, thực thi các middleware đó, rồi tới hàm xử lý cuối cùng và gửi response về client.

2.2.2.4 Ứng dụng của ExpressJS

ExpressJS thích hợp để xây dựng các dịch vụ web API hoặc các ứng dụng web động. Nó cũng rất phù hợp khi làm các ứng dụng một trang (SPA – Single Page Application) kết hợp với các framework front-end như ReactJS, Angular hay VueJS.

Trong các dự án microservice, ExpressJS thường được sử dụng để tạo các dịch vụ backend nhẹ, chịu trách nhiệm xử lý dữ liệu, giao tiếp với cơ sở dữ liệu và cung cấp API cho ứng dụng phía người dùng.

2.3 CÔNG NGHỆ VÀ FRAMEWORK FRONT-END

2.3.1 Tailwind CSS

2.3.1.1 Giới thiệu về Tailwind CSS

Tailwind CSS là một framework CSS thuộc dạng utility-first, nghĩa là nó cung cấp rất nhiều lớp CSS nhỏ, mỗi lớp thực hiện một chức năng cụ thể như màu nền, padding, margin, border, font hay shadow. Thay vì viết CSS riêng cho từng thành phần như các framework truyền thống, với Tailwind chúng ta chỉ cần kết hợp các lớp này trực tiếp trên HTML để tạo giao diện.

2.3.1.2 Đặc điểm của Tailwind CSS

Tailwind CSS giúp tăng tốc phát triển giao diện vì chúng ta có thể xây dựng giao diện trực tiếp trên HTML mà không phải viết CSS tùy chỉnh quá nhiều.

Framework này cũng rất linh hoạt và dễ tùy chỉnh. Các lớp CSS sẵn có có thể kết hợp với nhau để tạo giao diện độc đáo, đồng thời người dùng có thể định nghĩa thêm các class mới hoặc chỉnh sửa trực tiếp trong tệp cấu hình Tailwind.

Hiệu suất của Tailwind cũng được tối ưu tốt nhờ khả năng build ra file CSS chỉ chứa những class đã sử dụng. Điều này giúp giảm kích thước file và tăng tốc độ tải trang, đặc biệt hữu ích cho các dự án lớn.

Một ưu điểm khác là khả năng tái sử dụng và tối ưu hóa cấu trúc HTML. Việc sử dụng lại các class sẵn giúp giảm trùng lặp code, giữ HTML gọn gàng hơn và dễ bảo trì.

Nhược điểm lớn nhất của Tailwind là khi sử dụng, các thẻ HTML thường chứa rất nhiều class, khiến code dài và đôi khi khó nhìn.

2.3.2 Vite

2.3.2.1 Giới thiệu Vite

Vite là một công cụ build và phát triển ứng dụng web hiện đại. Vite được thiết kế để giải quyết các vấn đề về tốc độ, hiệu suất và trải nghiệm lập trình khi phát triển các ứng dụng web hiện đại, bao gồm cả React, Vue và nhiều framework khác. Khi kết hợp với React, Vite giúp tạo ra các ứng dụng nhanh, mượt mà và dễ bảo trì.

2.3.2.2 Cách Vite tối ưu hiệu suất

Vite khởi động rất nhanh vì nó sử dụng cơ chế ES Modules (ESM) của trình duyệt. Thay vì phải gộp toàn bộ code của ứng dụng thành một file lớn trước khi chạy, Vite chỉ tải những phần code cần thiết khi trình duyệt yêu cầu, nhờ đó thời gian mở ứng dụng được rút ngắn đáng kể.

Vite còn có tính năng Hot Module Replacement (HMR) mượt mà, nghĩa là khi bạn sửa code, Vite chỉ cập nhật những phần thay đổi mà không phải tải lại toàn bộ trang. Điều này giúp bạn thấy kết quả ngay lập tức và giữ nguyên trạng thái của ứng dụng.

Khi chuẩn bị đưa ứng dụng lên bản chính thức, Vite dùng công cụ Rollup để loại bỏ code không cần thiết và chia nhỏ các phần của ứng dụng. Nhờ đó, dung lượng file nhỏ hơn, trang web tải nhanh hơn và hiệu suất tổng thể tốt hơn.

Cấu hình của Vite cũng rất đơn giản, có thể bắt đầu dự án ngay mà không cần nhiều thiết lập phức tạp. Nếu muốn thì vẫn mở rộng và thêm các tính năng khác nhờ hệ thống plugin, hỗ trợ TypeScript, Tailwind CSS, React, Vue và nhiều công nghệ frontend khác một cách dễ dàng.

2.3.3 ReactJS

2.3.3.1 ReactJS là gì

ReactJS là một thư viện JavaScript miễn phí giúp xây dựng giao diện cho website. Thay vì phát triển cả trang web một lần, ReactJS chia giao diện thành các phần nhỏ gọi là component, mỗi component có thể sử dụng lại nhiều lần. Khi có thay đổi dữ liệu, ReactJS chỉ cập nhật các phần giao diện liên quan, giúp website phản hồi nhanh và mượt mà hơn.[5]

2.3.3.2 Các tính năng nổi bật của ReactJS

ReactJS áp dụng kiến trúc hướng thành phần (Component-Based), cho phép chia nhỏ giao diện thành các thành phần độc lập, mỗi thành phần có trạng thái và thuộc tính riêng. Các thành phần có thể được tái sử dụng nhiều lần, giúp quản lý và phát triển ứng dụng lớn trở nên dễ dàng hơn.

Để viết component, ReactJS sử dụng JSX, một cách viết JavaScript gần giống HTML, giúp code dễ đọc và trực quan hơn.

ReactJS quản lý giao diện theo từng phần nhỏ. Khi dữ liệu thay đổi, nó chỉ cập nhật những phần liên quan, giúp website phản hồi nhanh và mượt mà.

Trong ReactJS, dữ liệu chỉ di chuyển theo một hướng, từ thành phần cha xuống thành phần con thông qua props. Thành phần con nhận dữ liệu này để hiển thị hoặc sử dụng, nhưng không được phép thay đổi trực tiếp dữ liệu của thành phần cha.

Nếu thành phần con muốn thay đổi dữ liệu, nó sẽ gửi thông tin lên cha (thường bằng một hàm callback do thành phần cha cung cấp). Thành phần cha sau đó sẽ cập nhật trạng thái của mình và React sẽ tự động truyền dữ liệu mới xuống component con.

ReactJS rất phù hợp để xây dựng Single-Page Applications (SPA), cho phép cập nhật nội dung mượt mà mà không cần tải lại trang. Việc sử dụng props giúp các component trở nên năng động và dễ tái sử dụng.[5]

2.3.3.3 Cách hoạt động

Khi tải trang web, trình duyệt tạo ra một bản đồ cấu trúc của các phần tử trên trang. Mỗi thẻ HTML, hình ảnh hay đoạn văn bản đều được lưu trong bản đồ này. Khi dữ liệu thay đổi, nếu cập nhật toàn bộ trang, trình duyệt sẽ phải vẽ lại mọi thứ, làm chậm tốc độ.

ReactJS giải quyết vấn đề này bằng cách so sánh trạng thái hiện tại với trạng thái mới của giao diện và chỉ thay đổi những phần cần thiết. Nhờ vậy, trang web chạy nhanh, mượt mà và không phải tải lại toàn bộ.

Ngoài ra, ReactJS cũng có thể chạy trên máy chủ Node.js, nhờ thiết kế linh hoạt, cho phép tạo giao diện phía server nếu cần.[5]

2.3.3.4 Vòng đời của ReactJS

Mỗi component trong React đều có một vòng đời, tức là các bước từ khi tạo ra, hiển thị, cập nhật đến khi bị xóa khỏi trang.

Khởi tạo: Component được tạo ra với dữ liệu ban đầu, bao gồm các thông tin nhận từ component cha và trạng thái bên trong component.

Hiển thị: Component xuất hiện trên giao diện. Ở giai đoạn này, bạn có thể thực hiện các công việc như tải dữ liệu từ server hoặc thiết lập bộ đếm thời gian.

Cập nhật: Khi dữ liệu bên trong component hoặc thông tin nhận từ component cha thay đổi, component sẽ được cập nhật. React sẽ tự động hiển thị các thay đổi trên giao diện.

Xóa bỏ: Khi component bị gỡ khỏi trang, React sẽ dọn dẹp những thứ còn tồn tại như bộ đếm thời gian hay các sự kiện lắng nghe, để tránh lãng phí bộ nhớ.

2.4 HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU

2.4.1 Cơ sở dữ liệu phi quan hệ

2.4.1.1 Giới thiệu về cơ sở dữ liệu phi quan hệ

Cơ sở dữ liệu phi quan hệ (NoSQL) là loại cơ sở dữ liệu không dựa trên mô hình bảng với hàng và cột như cơ sở dữ liệu quan hệ. Thay vào đó, dữ liệu được lưu trữ dưới nhiều dạng khác nhau.

Cơ sở dữ liệu phi quan hệ thường được thiết kế để xử lý lượng dữ liệu lớn, mở rộng linh hoạt và hiệu năng cao, đặc biệt trong các ứng dụng web hiện đại, dịch vụ lưu trữ dữ liệu phi cấu trúc hoặc dữ liệu thay đổi liên tục.[6]

2.4.1.2 Các dạng cơ sở dữ liệu phi quan hệ

Cơ sở dữ liệu phi quan hệ gồm nhiều loại khác nhau, trong đó phổ biến nhất là bốn loại.

Thứ nhất là cơ sở dữ liệu tài liệu (Document Databases), lưu trữ dữ liệu dưới dạng tài liệu như JSON, BSON hay XML, mỗi tài liệu có thể chứa cấu trúc dữ liệu phức tạp và linh hoạt.

Thứ hai là cơ sở dữ liệu cặp khóa – giá trị, dữ liệu được lưu dưới dạng các cặp khóa và giá trị, đơn giản và nhanh chóng khi truy xuất.

Thứ ba là cơ sở dữ liệu đồ thị, chuyên xử lý và truy vấn các mối quan hệ phức tạp giữa dữ liệu, phù hợp với các ứng dụng cần mô tả mối quan hệ mạng xã hội hay phân cấp.

Thứ tư là cơ sở dữ liệu cột (Column-Family Stores), dữ liệu được tổ chức theo cột thay vì theo hàng, thích hợp với các ứng dụng cần xử lý khối lượng lớn dữ liệu.

2.4.2 MongoDB

2.4.2.1 Giới thiệu MongoDB

MongoDB là một hệ quản trị cơ sở dữ liệu phi quan hệ hướng tài liệu (document-oriented), mã nguồn mở, được phát triển bởi MongoDB Inc và phát hành lần đầu vào năm 2009.

Thay vì sử dụng bảng và hàng như cơ sở dữ liệu quan hệ, MongoDB lưu trữ dữ liệu dưới dạng collections và documents. Mỗi document là một đơn vị dữ liệu cơ bản, gồm các cặp key-value, trong khi collection là tập hợp các document, tương tự như table trong cơ sở dữ liệu quan hệ.

Dạng document trong MongoDB:

```
{
  "_id": "1",
  "name": "Nguyễn Văn A",
  "email": "nguyenvana@example.com",
  "age": 25,
  "address": {
    "city": "Hà Nội",
    "district": "Cầu Giấy",
    "street": "123 Đường ABC"},
  "hobbies": ["đọc sách", "bơi lội", "đi du lịch"]}
```

2.4.2.2 Các thao tác cơ bản trong MongoDB sử dụng NodeJS

Kết nối MongoDB

```
import mongoose from 'mongoose';
const connectDB = async () => {
    try {
        await mongoose.connect(process.env.MONGO_URI);
        console.log('MongoDB kết nối thành công');
    }catch (error) {
        console.error('Lỗi kết nối MongoDB:', error);
        process.exit(1);
    }};
export default connectDB;
```

INSERT: thêm document mới vào collection

```
const team = await Team.create({
  team_name: "Team A",
  description: "Mô tả team",
  created_by: userId
});
```

UPDATE: cập nhật dữ liệu trong document

```
const team = await Team.findByIdAndUpdate(
  teamId,
  {
    team_name: "Tên mới",
    description: "Mô tả mới"
  },
  { new: true }
);
```

DELETE: xóa document

```
await Team.findByIdAndDelete(teamId);
```

SELECT (find): truy vấn và lấy dữ liệu từ document

```
const teams = await Team.find();
```

2.5 BẢO MẬT VÀ HIỆU NĂNG CƠ BẢN TRONG ỨNG DỤNG WEB

2.5.1 Xác thực và phân quyền người dùng

Ứng dụng sử dụng JWT (Mã thông báo Web dạng JSON) để xác thực người dùng. JWT là một chuỗi ký hiệu gồm các thông tin được mã hóa, giúp hệ thống nhận biết người dùng mà không cần phải lưu trữ thông tin đăng nhập trên máy chủ.

Khi người dùng đăng nhập thành công, hệ thống tạo ra một mã thông báo chứa thông tin nhận dạng của người dùng và gửi về thiết bị của họ. Mã này được lưu trữ an toàn, thường trong Cookie chỉ đọc hoặc trong bộ nhớ cục bộ trên thiết bị.

Mỗi khi người dùng gửi yêu cầu tới máy chủ, mã thông báo này sẽ được gửi kèm theo để máy chủ xác nhận. Máy chủ sẽ kiểm tra tính hợp lệ của mã thông báo bằng một lớp xử lý gọi là trung gian. Nếu mã thông báo hợp lệ, người dùng được phép truy cập vào các chức năng tương ứng.

Cơ chế này giúp tách biệt việc xác thực và quyền truy cập, đảm bảo chỉ những người dùng hợp lệ mới có thể sử dụng các chức năng quan trọng của hệ thống, đồng thời tăng tính bảo mật và giảm tải cho máy chủ.

2.5.2 Bảo mật cơ bản

Phòng chống tấn công trên cơ sở dữ liệu: Hệ thống sử dụng thư viện Mongoose để làm việc với cơ sở dữ liệu MongoDB, giúp tự động lọc và ngăn chặn các truy vấn nguy hiểm. Bên cạnh đó, mọi dữ liệu từ người dùng trước khi lưu vào cơ sở dữ liệu đều được kiểm tra và xác thực bằng các công cụ như express-validator hoặc Joi, đảm bảo dữ liệu hợp lệ và an toàn.

Phòng chống tấn công chèn mã độc trên giao diện (XSS): Ứng dụng lọc và mã hóa dữ liệu nhập từ người dùng trước khi hiển thị. Đồng thời, ReactJS tự động xử lý các ký tự đặc biệt trong HTML, nhờ đó ngăn chặn việc người dùng chèn mã độc gây hại trên trang.

Phòng chống tấn công giả mạo yêu cầu (CSRF): Vì hệ thống sử dụng JWT thay cho cơ chế session truyền thống, nguy cơ tấn công CSRF được giảm đáng kể. Token chỉ được gửi khi yêu cầu hợp lệ từ người dùng đã đăng nhập, giúp bảo vệ các chức năng quan trọng khỏi bị lợi dụng.

2.5.3 Quản lý phiên làm việc với JWT

JSON Web Token (JWT) là một tiêu chuẩn mở được sử dụng để tạo ra các chuỗi token dạng JSON đã được ký số, cho phép truyền tải thông tin giữa các thành phần trong hệ thống một cách an toàn, gọn nhẹ và có thể kiểm tra tính xác thực. JWT thường được áp dụng trong cơ chế xác thực và phân quyền truy cập của các ứng dụng web và dịch vụ API hiện đại, đặc biệt phù hợp với các hệ thống được xây dựng theo kiến trúc Microservices.

Một JWT thông thường bao gồm ba phần chính: Header, Payload và Signature. Header chứa thông tin về thuật toán ký và loại token; Payload lưu trữ các thông tin định danh và dữ liệu cần thiết của người dùng; Signature được sử dụng để đảm bảo tính toàn vẹn của token, giúp phát hiện các trường hợp token bị thay đổi trong quá trình truyền tải. Cấu trúc này cho phép JWT vừa đảm bảo tính bảo mật vừa giữ được sự gọn nhẹ khi trao đổi dữ liệu giữa client và server.

Trong hệ thống của đề tài, JWT được sử dụng để quản lý trạng thái đăng nhập thay cho cơ chế session truyền thống. Sau khi người dùng xác thực thành công, hệ thống sẽ cấp một JWT chứa các thông tin nhận dạng cơ bản như mã người dùng, vai trò và thời gian hết hạn. Token này được sử dụng để xác thực các yêu cầu tiếp theo gửi đến các dịch vụ API, giúp hệ thống không cần lưu trữ trạng thái phiên làm việc trên máy chủ, từ đó giảm tải cho server và tăng khả năng mở rộng khi triển khai theo kiến trúc vi dịch vụ.

JWT trong hệ thống được lưu trữ tại phía client dưới dạng local storage. Cách tiếp cận này cho phép ứng dụng phía client chủ động đính kèm token vào phần tiêu đề của các yêu cầu HTTP khi giao tiếp với các dịch vụ API. Tuy nhiên, để hạn chế các rủi ro bảo mật như tấn công Cross-Site Scripting (XSS), hệ thống cần áp dụng các biện pháp bảo mật phù hợp, bao gồm kiểm soát chặt chẽ dữ liệu đầu vào, hạn chế chèn mã độc và tuân thủ các nguyên tắc phát triển ứng dụng web an toàn.[7]

2.6 CÔNG CỤ HỖ TRỢ PHÁT TRIỂN VÀ TRIỂN KHAI

2.6.1 Postman

Postman là công cụ quan trọng giúp lập trình viên kiểm thử, phát triển và quản lý API một cách hiệu quả. Không chỉ gửi yêu cầu và nhận phản hồi, Postman còn cho phép

tạo các kịch bản kiểm thử tự động, giúp kiểm tra API theo nhiều tình huống khác nhau mà không cần thao tác thủ công.

Với Postman, lập trình viên có thể tổ chức các yêu cầu thành bộ sưu tập, tạo biến môi trường để dễ dàng chuyển đổi giữa các môi trường phát triển, thử nghiệm và sản xuất. Ngoài ra, Postman còn ghi lại lịch sử các yêu cầu, giúp dễ dàng theo dõi, so sánh và gỡ lỗi khi cần.

Một điểm mạnh khác là Postman hỗ trợ làm việc nhóm, cho phép chia sẻ bộ sưu tập, tài liệu API và kịch bản kiểm thử, từ đó nâng cao hiệu quả hợp tác giữa các thành viên trong dự án. Nhờ giao diện thân thiện, hỗ trợ nhiều định dạng dữ liệu như JSON, XML, HTML và khả năng tích hợp với các công cụ quản lý mã nguồn như GitHub, Postman trở thành một công cụ không thể thiếu trong phát triển và kiểm thử API hiện nay.

2.6.2 GitHub

GitHub là một nền tảng lưu trữ và quản lý mã nguồn dựa trên hệ thống kiểm soát phiên bản Git, được sử dụng rộng rãi trong lĩnh vực phát triển phần mềm. Nền tảng này cho phép lưu trữ tập trung mã nguồn dự án, theo dõi lịch sử thay đổi và hỗ trợ phối hợp làm việc giữa nhiều cá nhân hoặc nhóm phát triển.

Bên cạnh chức năng quản lý phiên bản, GitHub còn cung cấp môi trường cộng tác trực tuyến, giúp kiểm soát quá trình phát triển phần mềm một cách có tổ chức, minh bạch và hiệu quả trong suốt vòng đời của dự án.

2.6.2.1 Chức năng chính của GitHub

GitHub cho phép người dùng tạo và quản lý các repository nhằm lưu trữ toàn bộ mã nguồn của dự án. Hệ thống branch hỗ trợ tách biệt các nhánh phát triển, giúp lập trình viên triển khai hoặc thử nghiệm tính năng mới mà không ảnh hưởng đến nhánh chính.

Thông qua cơ chế commit, GitHub ghi nhận chi tiết từng thay đổi của mã nguồn, cho phép theo dõi lịch sử chỉnh sửa và khôi phục lại các phiên bản trước khi cần thiết. Ngoài ra, tính năng pull request hỗ trợ việc hợp nhất mã nguồn giữa các nhánh, đồng thời tạo điều kiện cho việc kiểm tra và đánh giá mã trước khi tích hợp vào hệ thống chính.

GitHub cũng cung cấp chức năng issues để quản lý lỗi, yêu cầu cải tiến hoặc các nhiệm vụ cần thực hiện trong dự án. Cơ chế phân quyền người dùng giúp kiểm soát

quyền truy cập, đảm bảo tính an toàn và nhất quán trong quá trình cộng tác phát triển phần mềm.

2.6.2.2 Ưu điểm của GitHub

GitHub mang lại nhiều lợi ích trong việc quản lý mã nguồn tập trung và duy trì tính nhất quán giữa các thành viên tham gia dự án. Nền tảng này hỗ trợ hiệu quả cho làm việc nhóm thông qua các chức năng như quản lý nhánh, theo dõi lịch sử thay đổi, kiểm soát phiên bản và quản lý công việc.

Toàn bộ lịch sử chỉnh sửa mã nguồn được lưu trữ đầy đủ, giúp việc truy vết, đánh giá và khôi phục dữ liệu trở nên thuận tiện. Bên cạnh đó, GitHub có khả năng tích hợp với nhiều công cụ phát triển phổ biến như Visual Studio Code, Postman và Docker, góp phần nâng cao hiệu quả và chất lượng của quá trình phát triển phần mềm.

2.6.3 Docker

Docker là một nền tảng mã nguồn mở dùng để đóng gói, triển khai và chạy ứng dụng trong các container. Container là các môi trường độc lập, chứa toàn bộ mã nguồn, thư viện, cấu hình và các phụ thuộc cần thiết để ứng dụng có thể chạy ổn định trên mọi hệ thống.

Nhờ Docker, quá trình phát triển và triển khai phần mềm trở nên nhanh hơn, đồng nhất hơn và ít xảy ra lỗi do khác biệt môi trường (giữa máy lập trình viên, máy chủ hoặc hệ thống thật).

2.6.3.1 Chức năng chính của Docker

Docker là công cụ giúp đóng gói ứng dụng cùng các thành phần phụ thuộc vào một đơn vị, nhờ đó việc triển khai ứng dụng trở nên nhanh chóng và nhất quán trên nhiều môi trường.

Docker cung cấp Docker Engine để chạy và quản lý container, cùng với Docker Image (bản mẫu của container) có thể tái sử dụng nhiều lần. Người dùng có thể thao tác thông qua Docker CLI để quản lý container, image, mạng và bộ lưu trữ (volume).

Ngoài ra, Docker Compose cho phép định nghĩa và chạy nhiều container cùng lúc, còn Docker có thể tích hợp với các công cụ DevOps như GitHub Actions hoặc Jenkins để tự động hóa quá trình xây dựng và triển khai ứng dụng.

2.6.3.2 Một số khái niệm cơ bản trong Docker

Docker Image: Docker Image là bản mẫu dùng để tạo ra container. Một image chứa toàn bộ mã nguồn, thư viện, môi trường và cấu hình của ứng dụng. Image thường được tạo ra từ Dockerfile, có thể lưu trữ trên Docker Hub hoặc registry riêng.

Docker Container: Container là phiên bản đang chạy của một Docker Image. Mỗi container là một môi trường độc lập, cách ly với các container khác, giúp đảm bảo ứng dụng chạy ổn định và an toàn. Nếu ví Docker Image là "khuôn mẫu", thì Docker Container chính là "sản phẩm thật" được tạo ra từ khuôn đó.

Dockerfile: Dockerfile là tệp cấu hình chứa các lệnh để Docker biết cách tạo ra một image. Nó mô tả môi trường chạy, mã nguồn cần sao chép, thư viện cần cài đặt và lệnh khởi động ứng dụng.

```
FROM node:18-alpine
WORKDIR /app
COPY . .
RUN npm install
CMD ["npm", "start"]
```

Docker Compose: Docker Compose là công cụ giúp chạy và quản lý nhiều container cùng lúc. Thông qua tệp docker-compose.yml, ta có thể định nghĩa cấu hình cho các dịch vụ (như backend, frontend, database) và kết nối chúng lại với nhau.

Docker Hub: Docker Hub là kho lưu trữ và chia sẻ Docker Image trực tuyến. Người dùng có thể tải xuống các image phổ hoặc tải lên image tự tạo để chia sẻ với người khác. Nó tương tự như GitHub nhưng dành riêng cho việc lưu trữ image.

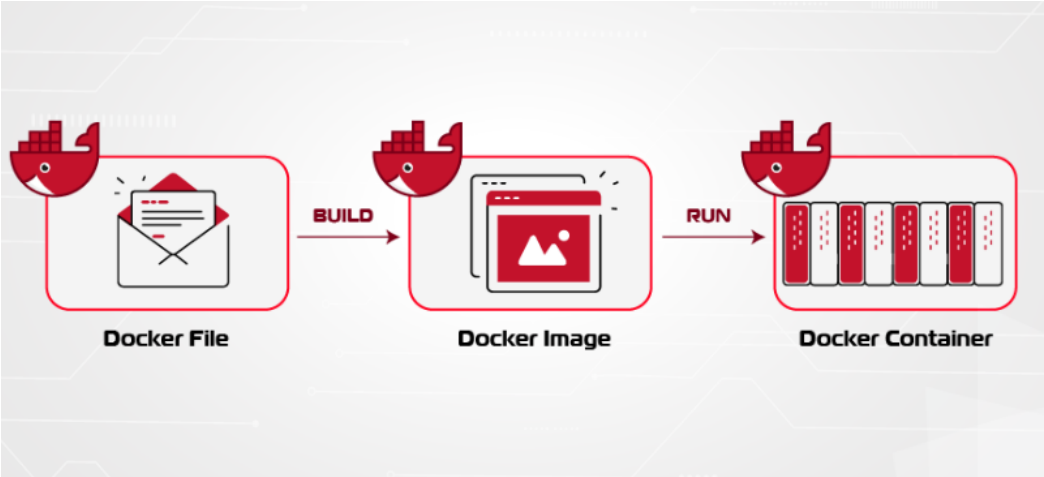
2.6.3.3 Cách hoạt động của Docker

Quy trình hoạt động của Docker gồm ba bước chính:

Dockerfile → Image: Nhà phát triển viết Dockerfile để mô tả cách tạo ra môi trường cho ứng dụng. Docker sẽ dựa trên đó để build thành một image.

Image → Container: Khi chạy image, Docker sẽ khởi tạo container – là môi trường thực thi độc lập của ứng dụng.

Container → Deploy: Container có thể được triển khai lên bất kỳ máy chủ hoặc dịch vụ cloud nào có Docker Engine, đảm bảo ứng dụng chạy nhất quán ở mọi nơi.



Hình 2.5 Cách hoạt động của Docker

2.7 TỔNG KẾT

Đề tài đã lựa chọn và áp dụng một ngăn xếp công nghệ hiện đại, tập trung vào tính mở rộng và hiệu quả, cùng với các công cụ mạnh mẽ để hỗ trợ quy trình phát triển và triển khai.

Hệ thống được thiết kế dựa trên kiến trúc vi dịch vụ, cho phép các dịch vụ hoạt động độc lập, dễ dàng mở rộng và có khả năng chịu lỗi cao. API Gateway đóng vai trò là lớp trung gian, làm cổng truy cập thống nhất cho toàn bộ hệ thống, quản lý việc xác thực và tích hợp dữ liệu từ nhiều dịch vụ.

Hệ thống được xây dựng bằng bộ công nghệ JavaScript hiện đại. Phần xử lý phía backend sử dụng Node.js và framework Express.js, với khả năng xử lý không đồng bộ giúp tạo ra các API hiệu suất cao. Phần frontend sử dụng thư viện ReactJS để xây dựng giao diện theo mô hình thành phần, đồng thời tối ưu hóa việc hiển thị, chỉ cập nhật những phần thay đổi cần thiết. Quá trình phát triển được hỗ trợ bởi công cụ Vite, giúp tăng tốc độ làm việc và tối ưu hóa việc đóng gói ứng dụng, trong khi giao diện được thiết kế để tự điều chỉnh linh hoạt trên nhiều kích thước màn hình nhờ Tailwind CSS.

Đối với quản lý dữ liệu, hệ thống lựa chọn MongoDB vì sự linh hoạt về cấu trúc, phù hợp với kiến trúc vi dịch vụ và các loại dữ liệu dự án đa dạng. Về bảo mật, cơ chế xác thực dựa trên JWT được triển khai để quản lý phiên và phân quyền người dùng, giúp tăng tính an toàn và giảm tải cho máy chủ.

Cuối cùng, quy trình triển khai được tối ưu hóa bằng các công cụ hiện đại. Nền tảng Docker được sử dụng để container hóa toàn bộ ứng dụng, đảm bảo môi trường chạy nhất quán từ phát triển đến triển khai. GitHub được sử dụng để quản lý mã nguồn. Đồng thời, công cụ Postman được sử dụng để kiểm thử, phát triển và quản lý các API một cách hiệu quả.

CHƯƠNG 3 HIỆN THỰC HÓA NGHIÊN CỨU

3.1 MÔ TẢ BÀI TOÁN

3.1.1 Mục tiêu hệ thống

Hệ thống được xây dựng nhằm thiết lập một nền tảng quản lý dự án và công việc nhóm trực tuyến tập trung, giúp tối ưu hóa quy trình cộng tác trong môi trường số. Mục tiêu trọng tâm bao gồm:

Khắc phục hạn chế truyền thống: Thay thế các phương thức quản lý thủ công, phân tán qua các ứng dụng nhắn tin tức thời (như Zalo, Messenger) vốn dễ dẫn đến tình trạng trôi thông tin và thiếu cái nhìn tổng quan về tiến độ.

Ứng dụng công nghệ hiện đại: Triển khai hệ thống dựa trên kiến trúc vi dịch vụ để khắc phục nhược điểm của kiến trúc nguyên khối, đảm bảo tính linh hoạt, khả năng chịu lỗi và dễ dàng bảo trì.

Tối ưu hóa quản lý: Cung cấp các công cụ trực quan để phân công nhiệm vụ, theo dõi trạng thái công việc và lưu trữ tài liệu tập trung, từ đó nâng cao tính minh bạch và trách nhiệm cá nhân của từng thành viên trong nhóm.

3.1.2 Quy trình nghiệp vụ tổng quát

Quy trình nghiệp vụ của hệ thống được thiết kế theo hướng tinh gọn, tập trung vào các giai đoạn cốt lõi của một dự án học tập hoặc làm việc nhóm:

Khởi tạo và quản trị người dùng: Người dùng đăng ký tài khoản và được xác thực qua hệ thống để đảm bảo tính an toàn dữ liệu.

Thiết lập không gian làm việc: Người dùng thực hiện tạo nhóm và khởi tạo các dự án tương ứng với nhu cầu thực tế.

Lập kế hoạch và phân công: Trong mỗi dự án, các tác vụ được tạo lập với đầy đủ thông tin về mô tả, thời hạn và người thực hiện.

Vận hành và tương tác: Thành viên cập nhật trạng thái công việc, tham gia thảo luận qua hệ thống bình luận và nhận thông báo theo thời gian thực về các thay đổi quan trọng.

Giám sát và kết thúc: Trưởng nhóm hoặc người quản lý theo dõi tiến độ tổng thể qua các báo cáo thống kê để có những điều chỉnh phù hợp cho đến khi dự án hoàn thành.

3.1.3 Đối tượng sử dụng

Hệ thống hướng tới các nhóm người dùng cụ thể trong môi trường giáo dục và làm việc quy mô nhỏ:

Sinh viên các trường: Đặc biệt là sinh viên khối ngành kỹ thuật và công nghệ thông tin cần một công cụ hỗ trợ thực hiện đồ án, bài tập lớn một cách chuyên nghiệp nhưng dễ tiếp cận.

Nhóm nghiên cứu và làm việc quy mô nhỏ: Các nhóm có nhu cầu cộng tác trực tuyến nhưng đòi hỏi sự đơn giản, tối giản hóa các quy trình phức tạp của những công cụ doanh nghiệp như Jira hay Trello.

Giảng viên hướng dẫn: Có thể sử dụng hệ thống như một công cụ hỗ trợ theo dõi, đánh giá mức độ đóng góp và quá trình làm việc thực tế của từng sinh viên trong nhóm một cách chính xác hơn.

3.2 ĐẶC TẢ YÊU CẦU HỆ THỐNG

Trên cơ sở phân tích thực trạng quản lý công việc của sinh viên và những hạn chế từ các công cụ hiện có, hệ thống được đặc tả với các yêu cầu chức năng và phi chức năng nhằm đảm bảo tính hiệu quả, an toàn và khả năng mở rộng.

3.2.1 Yêu cầu chức năng

Hệ thống tập trung vào các nghiệp vụ cốt lõi hỗ trợ quá trình cộng tác nhóm, bao gồm các nhóm chức năng chính sau:

Quản lý người dùng: Hệ thống cho phép người dùng đăng ký tài khoản mới, đăng nhập để xác thực danh tính và quản lý thông tin hồ sơ cá nhân. Quy trình này phải được tích hợp cơ chế bảo mật để bảo vệ thông tin người dùng.

Quản lý nhóm: Người dùng có thể khởi tạo các nhóm làm việc, thực hiện mời thành viên tham gia vào không gian chung để phối hợp thực hiện dự án.

Quản lý dự án: Hỗ trợ việc tạo lập dự án, thiết lập các kế hoạch tổng quát và theo dõi tiến độ thực hiện trong phạm vi từng nhóm cụ thể.

Quản lý công việc: Đây là chức năng trung tâm, cho phép khởi tạo tác vụ, mô tả chi tiết công việc, phân công thực hiện, thiết lập thời hạn và cập nhật trạng thái tiến độ.

Tương tác và cộng tác: Hệ thống hỗ trợ tính năng bình luận trực tiếp trong từng đầu việc để trao đổi thông tin chuyên sâu và lưu lại lịch sử chỉnh sửa.

Hệ thống thông báo: Tự động gửi thông báo đến các thành viên liên quan khi có sự thay đổi về trạng thái công việc, phân công nhiệm vụ mới hoặc có bình luận mới.

Báo cáo và thống kê: Cung cấp số liệu tổng quát về tiến độ hoàn thành của dự án và công việc.

3.2.2 Yêu cầu phi chức năng

Bên cạnh các tính năng nghiệp vụ, hệ thống cần đáp ứng các tiêu chuẩn kỹ thuật để vận hành ổn định trong môi trường thực tế:

Tính mở rộng: Với kiến trúc Microservices, hệ thống phải có khả năng mở rộng linh hoạt từng dịch vụ độc lập mà không gây ảnh hưởng đến cấu trúc tổng thể, đáp ứng nhu cầu gia tăng người dùng trong tương lai.

Hiệu năng xử lý: Ứng dụng phải đảm bảo tốc độ phản hồi nhanh, cập nhật dữ liệu mượt mà nhờ cơ chế xử lý bất đồng bộ của NodeJS và khả năng cập nhật giao diện cục bộ của ReactJS.

Tính bảo mật: Xác thực người dùng thông qua mã thông báo JWT để quản lý phiên làm việc an toàn và giảm tải cho máy chủ. Phòng chống các lỗ hổng bảo mật phổ biến như tấn công chèn mã độc, giả mạo yêu cầu và tấn công vào cơ sở dữ liệu.

Khả năng chịu lỗi: Kiến trúc vi dịch vụ giúp hệ thống duy trì hoạt động phần lớn các chức năng ngay cả khi một dịch vụ cụ thể gặp sự cố.

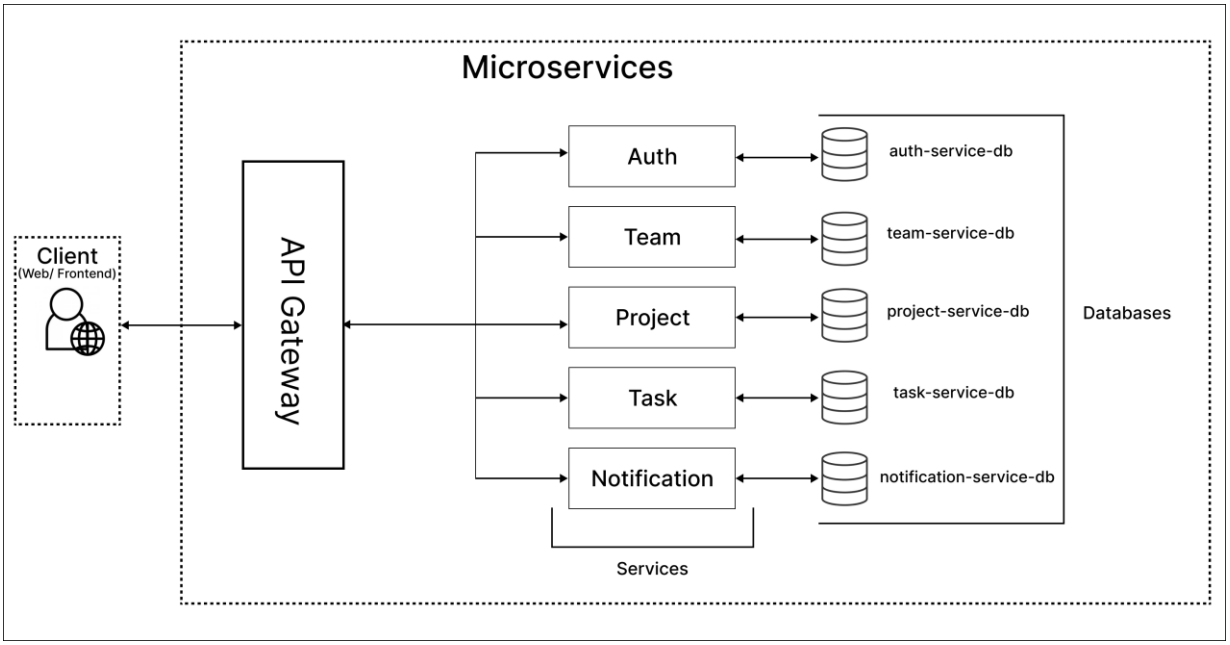
Tính nhất quán môi trường: Toàn bộ hệ thống được container hóa bằng Docker nhằm đảm bảo sự đồng bộ tuyệt đối về môi trường từ giai đoạn phát triển đến khi triển khai thực tế.

Trải nghiệm người dùng: Giao diện phải được thiết kế tinh gọn, trực quan, dễ tiếp cận đối với sinh viên và có khả năng hiển thị tương thích trên nhiều kích thước màn hình.

3.3 PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

3.3.1 Sơ đồ kiến trúc tổng thể

Hệ thống được xây dựng dựa trên phong cách thiết kế kiến trúc vi dịch vụ, trong đó các chức năng nghiệp vụ được chia tách thành những dịch vụ độc lập. Cấu trúc tổng thể bao gồm các thành phần chính sau:



Hình 3.1 Sơ đồ kiến trúc tổng thể hệ thống

Lớp giao diện: Được xây dựng bằng thư viện ReactJS, chịu trách nhiệm tiếp nhận thao tác người dùng và hiển thị thông tin trực quan.

Cổng API: Đóng vai trò là lớp trung gian và cổng truy cập duy nhất cho toàn bộ yêu cầu từ phía Client. API Gateway thực hiện các nhiệm vụ quan trọng như xác thực người dùng qua JWT, phân quyền và điều phối yêu cầu đến đúng dịch vụ xử lý.

Các dịch vụ nghiệp vụ: Bao gồm user service (quản lý người dùng), team service (quản lý nhóm), project service (quản lý dự án), task service (quản lý công việc) và notification service (quản lý thông báo). Mỗi dịch vụ vận hành độc lập, có mã nguồn và cơ sở dữ liệu riêng biệt, giúp tăng khả năng chịu lỗi và dễ dàng nâng cấp.

Lớp cơ sở dữ liệu: Hệ thống sử dụng MongoDB làm hệ quản trị cơ sở dữ liệu phi quan hệ, lưu trữ dữ liệu dưới dạng tài liệu linh hoạt, phù hợp với đặc thù thay đổi nhanh của các dự án.

Cơ chế giao tiếp: Các dịch vụ trao đổi thông tin với nhau thông qua giao thức HTTP dựa trên mô hình RESTful API, sử dụng định dạng dữ liệu JSON để đảm bảo tính nhẹ và tương thích cao.

3.3.2 Sơ đồ use case

Sơ đồ use case được xây dựng nhằm mô hình hóa các yêu cầu về chức năng và xác định ranh giới tương tác giữa các tác nhân với hệ thống quản lý dự án trực tuyến. Qua đó, hệ thống phân định rõ trách nhiệm và quyền hạn của từng đối tượng tham gia vào quy trình quản lý và thực hiện công việc nhóm.

A. Các tác nhân hệ thống

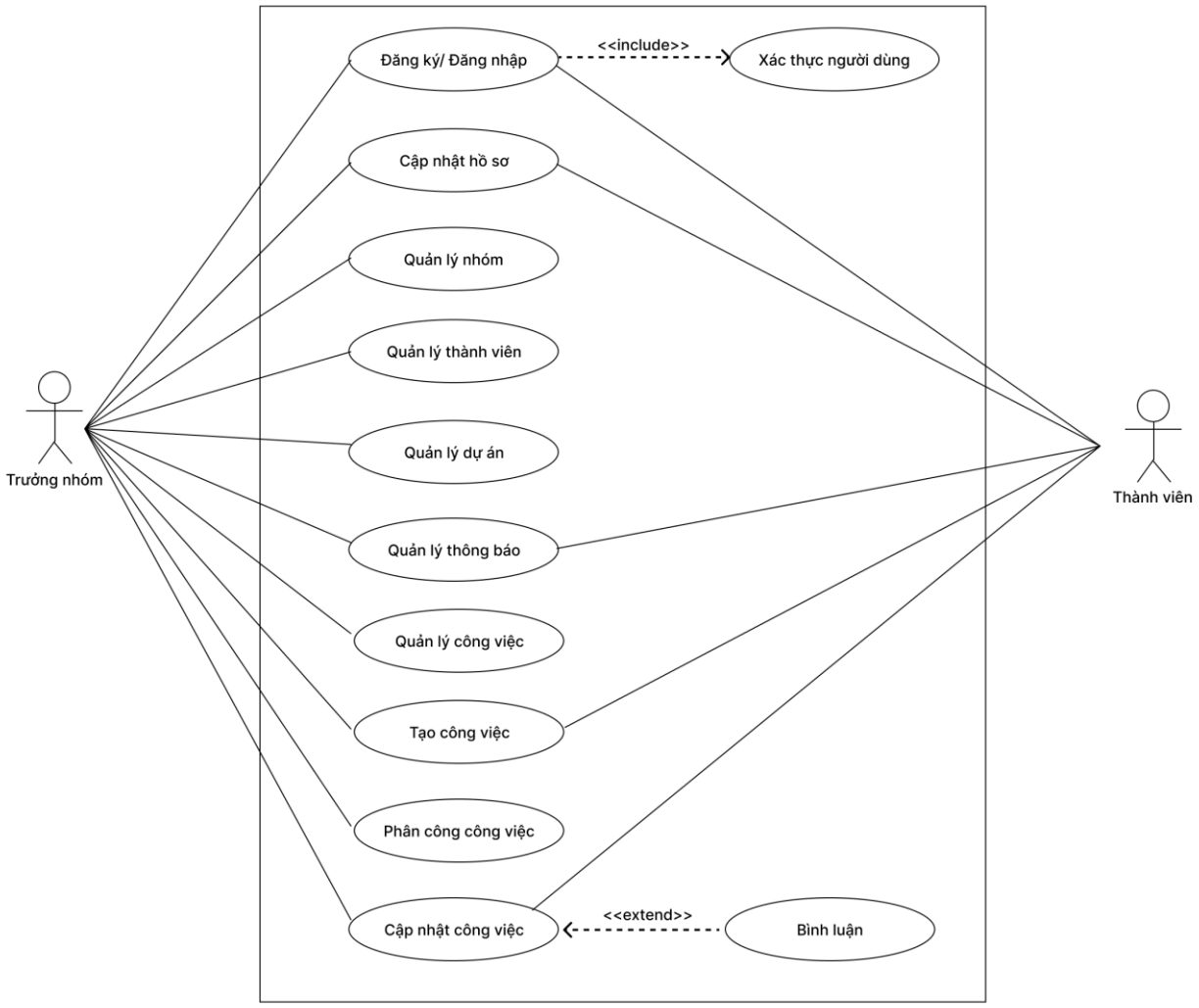
Người dùng: Là tác nhân tổng quát đại diện cho mọi thành viên đã đăng ký tài khoản trong hệ thống, có quyền thực hiện các chức năng cơ bản về tài khoản và tương tác chung.

Trưởng nhóm: Là người khởi tạo không gian làm việc, có quyền quản trị cao nhất đối với dự án và thành viên trong nhóm do mình quản lý.

Thành viên: Là nhân sự tham gia vào nhóm để thực hiện các nhiệm vụ được phân công và cộng tác với các thành viên khác.

B. Các nhóm use case chính

Dựa trên yêu cầu nghiệp vụ thực tế, các chức năng được phân loại thành các nhóm chính sau:



Hình 3.2 Sơ đồ use case với các chức năng của tác nhân trưởng nhóm và thành viên

3.3.2.1 Nhóm quản lý tài khoản

Nhóm dùng chung gồm:

Đăng ký, đăng nhập: Thực hiện xác thực danh tính để truy cập hệ thống.

Cập nhật hồ sơ: Thay đổi thông tin cá nhân và cài đặt tài khoản người dùng.

3.3.2.2 Nhóm quản lý dự án và nhân sự

Quản trị dự án: Thực hiện các quyền hạn khởi tạo, điều chỉnh nội dung (sửa) hoặc xóa dự án khỏi hệ thống.

Quản lý thành viên: Trưởng nhóm có quyền loại bỏ thành viên ra khỏi nhóm để đảm bảo cơ cấu nhân sự phù hợp với tiến độ dự án.

3.3.2.3 Nhóm quản lý tác vụ và nghiệp vụ công việc:

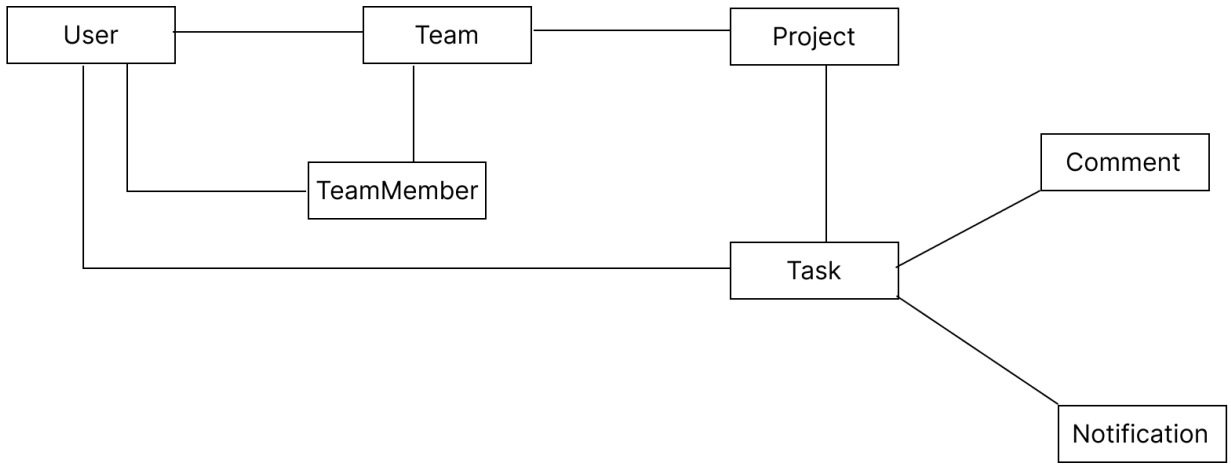
Đối với trưởng nhóm: Toàn quyền thực hiện các thao tác thêm, sửa, xóa các đầu việc; trực tiếp phân công nhiệm vụ cho các thành viên cụ thể trong nhóm.

Đối với thành viên: Thực hiện cập nhật tiến độ công việc và có quyền khởi tạo các công việc mới phát sinh trong quá trình thực hiện dự án.[8]

3.4 THIẾT KẾ CƠ SỞ DỮ LIỆU

3.4.1 Mô hình dữ liệu tổng quát

Mô hình dữ liệu tổng quát mô tả cấu trúc logic và các luồng liên kết giữa các thực thể hạt nhân trong hệ thống. Mặc dù vận hành trên nền tảng NoSQL, hệ thống vẫn duy trì tính toàn vẹn nghiệp vụ thông qua cơ chế tham chiếu bằng mã định danh ObjectID giữa các collection.



Hình 3.3 Mối quan hệ giữa các collection trong hệ thống

Quản lý định danh và quyền hạn: Collection User đóng vai trò là chủ thể của mọi tương tác trong hệ thống. Mối quan hệ giữa người dùng và nhóm được thiết lập thông qua thực thể trung gian TeamMember. Cấu trúc này cho phép một người dùng có thể tham gia vào nhiều nhóm khác nhau với các vai trò riêng biệt như trưởng nhóm hoặc thành viên.

Phân cấp không gian làm việc: Dữ liệu được tổ chức theo cấp bậc từ vĩ mô đến vi mô để tối ưu hóa việc quản lý. Mỗi nhóm đại diện cho một không gian cộng tác, chứa đựng các dự án cụ thể. Bên trong dự án, các công việc đóng vai trò là đơn vị nhỏ nhất của quy trình nghiệp vụ, nơi lưu trữ thông tin về người thực hiện, trạng thái và thời hạn hoàn thành.

Hệ thống tương tác và lưu vết: Để hỗ trợ tính cộng tác và minh bạch, các thực thể Comment và Notification được liên kết trực tiếp với Task nhằm ghi nhận nội dung thảo luận và đẩy thông báo tiến độ theo thời gian thực.

3.4.2 Đặc tả chi tiết các collection

Cơ sở dữ liệu của hệ thống được cấu trúc thành các collection riêng biệt để phục vụ kiến trúc vi dịch vụ, đảm bảo tính độc lập và hiệu quả trong việc truy xuất dữ liệu. Chi tiết các trường dữ liệu và kiểu dữ liệu của từng collection được mô tả cụ thể như sau:

Collection User

Lưu trữ thông tin định danh và trạng thái tài khoản của người dùng trong hệ thống.

Bảng 3.1 Mô tả collection User

Trường	**Kiểu**	**Mô tả**
id	ObjectId	Khóa chính của người dùng
full_name	String	Họ và tên người dùng
email	String	Email đăng nhập, duy nhất
password	String	Mật khẩu đã được mã hóa
avatar	String	Đường dẫn ảnh đại diện
create_at	Date	Ngày tạo tài khoản
last_login	Date	Thời điểm đăng nhập gần nhất

Collection Teams

Quản lý các không gian làm việc chung của sinh viên.

Bảng 3.2 Mô tả collection Teams

Trường	**Kiểu**	**Mô tả**
id	ObjectId	Khóa chính
team_name	String	Tên nhóm
description	String	Mô tả nhóm
create_by	ObjectId	Người tạo nhóm (liên kết Users)
create_at	Date	Thời gian tạo nhóm

Collection TeamMembers

Mô tả mối quan hệ giữa người dùng và các nhóm tương ứng, xác định vai trò cụ thể trong từng nhóm.

Bảng 3.3 Mô tả collection TeamMembers

Trường	**Kiểu**	**Mô tả**
id	ObjectId	Khóa chính
team_id	ObjectId	Tham chiếu đến nhóm cụ thể
user_id	ObjectId	Tham chiếu đến người dùng thành viên
role	String	Vai trò trong nhóm (leader/member)
joined_at	Date	Ngày tham gia nhóm

Collection Projects

Lưu trữ thông tin chi tiết các dự án được triển khai trong phạm vi từng nhóm.

Bảng 3.4 Mô tả collection Projects

Trường	**Kiểu**	**Mô tả**
id	ObjectId	Khóa chính
team_id	ObjectId	Tham chiếu đến nhóm sở hữu dự án
project_name	String	Tên dự án
description	String	Mô tả dự án
start_date	Date	Ngày bắt đầu
end_date	Date	Ngày kết thúc
progress	Number	% hoàn thành
create_by	ObjectId	Người tạo dự án
create_at	Date	Ngày tạo dự án
updated_at	Date	Ngày cập nhật lần cuối

Collection Tasks

Quản lý danh sách các việc cụ thể thuộc từng dự án.

Bảng 3.5 Mô tả collection Tasks

Trường	**Kiểu**	**Mô tả**
id	ObjectId	Khóa chính
project_id	ObjectId	Tham chiếu đến dự án chứa công việc
task_name	String	Tên công việc
description	String	Mô tả chi tiết
assigned_to	ObjectId	Người được giao

create_by	ObjectId	Người tạo task
start_date	Date	Ngày bắt đầu
due_date	Date	Hạn hoàn thành
status	String	Trạng thái (todo / in_progress / done)
priority	String	Độ ưu tiên (low / medium / high)
progress	Number	% hoàn thành

Collection Comments

Lưu trữ lịch sử trao đổi và thảo luận bên trong từng công việc.

Bảng 3.6 Mô tả collection Comments

Trường	**Kiểu**	**Mô tả**
id	ObjectId	Khóa chính
task_id	ObjectId	Tham chiếu đến công việc được bình luận
user_id	ObjectId	Người thực hiện bình luận
content	String	Nội dung comment

Collection Notifications

Lưu trữ các thông báo hệ thống gửi đến người dùng.

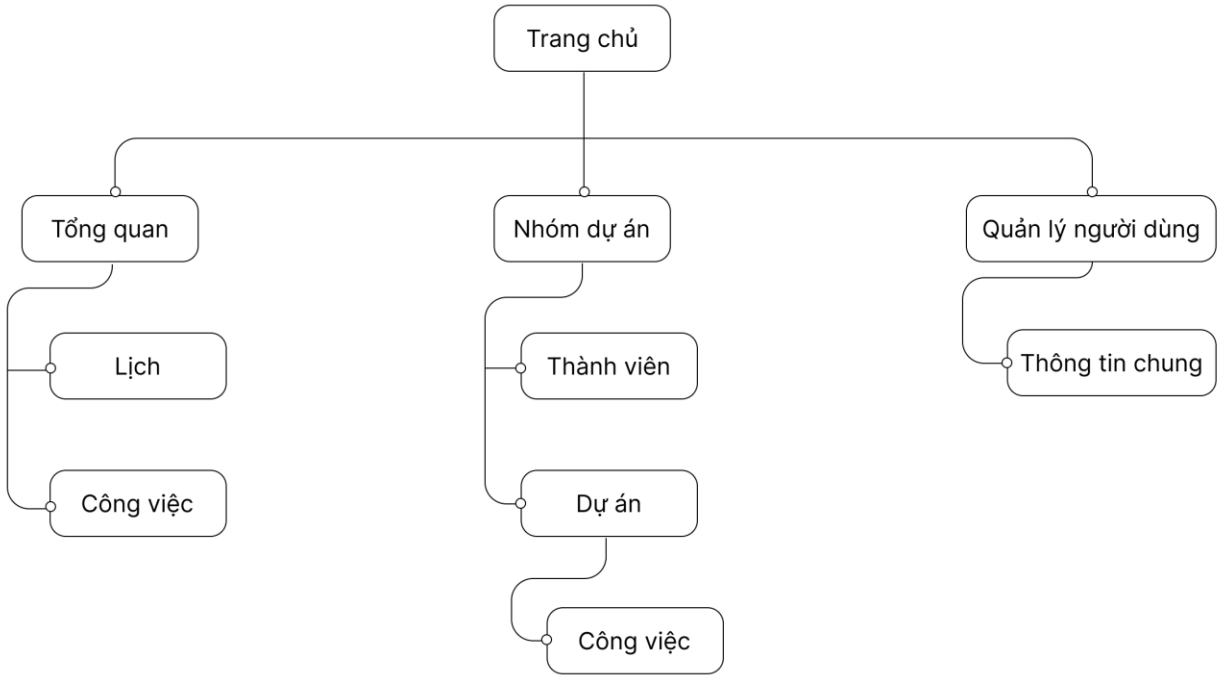
Bảng 3.7 Mô tả collection Notifications

Trường	**Kiểu**	**Mô tả**
id	ObjectId	Khóa chính
user_id	ObjectId	Người nhận thông báo
message	String	Nội dung thông báo
send_at	Date	Thời điểm gửi
is_read	Boolean	Đánh dấu đã đọc

3.5 THIẾT KẾ GIAO DIỆN

3.5.1 Sơ đồ tổ chức thông tin

Sơ đồ site map dưới đây mô tả cấu trúc phân cấp các trang chức năng trong hệ thống, giúp định hình luồng di chuyển của người dùng từ khi đã đăng nhập đến khi thực hiện các tác vụ quản lý dự án và công việc nhóm.



Hình 3.4 Sơ đồ tổ chức thông tin Sitemap

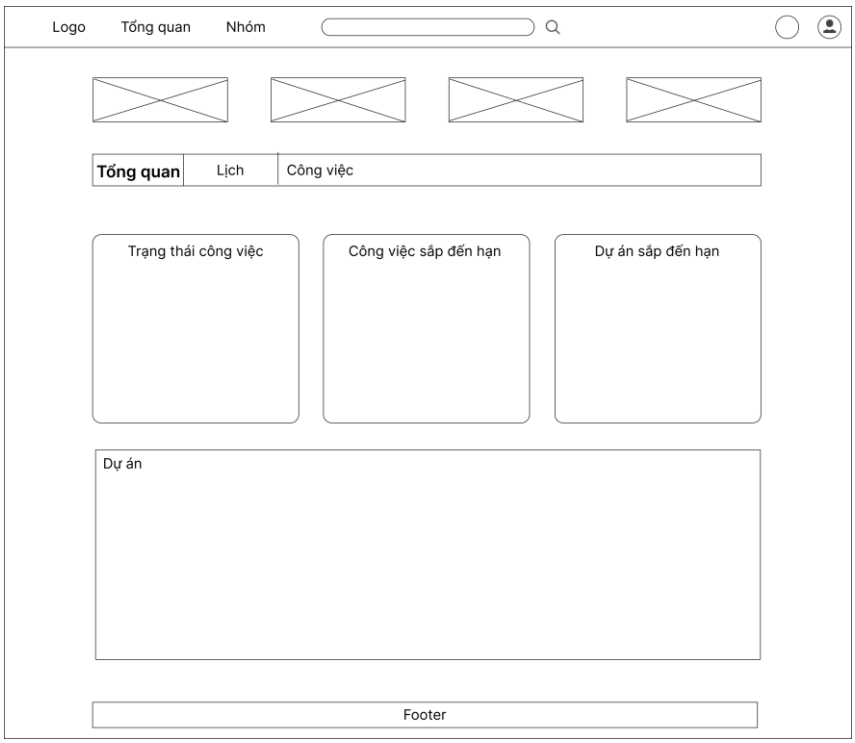
3.5.2 Thiết kế các giao diện chính

3.5.2.1 Thiết kế giao diện đăng nhập

Hình 3.5 Thiết kế giao diện đăng nhập

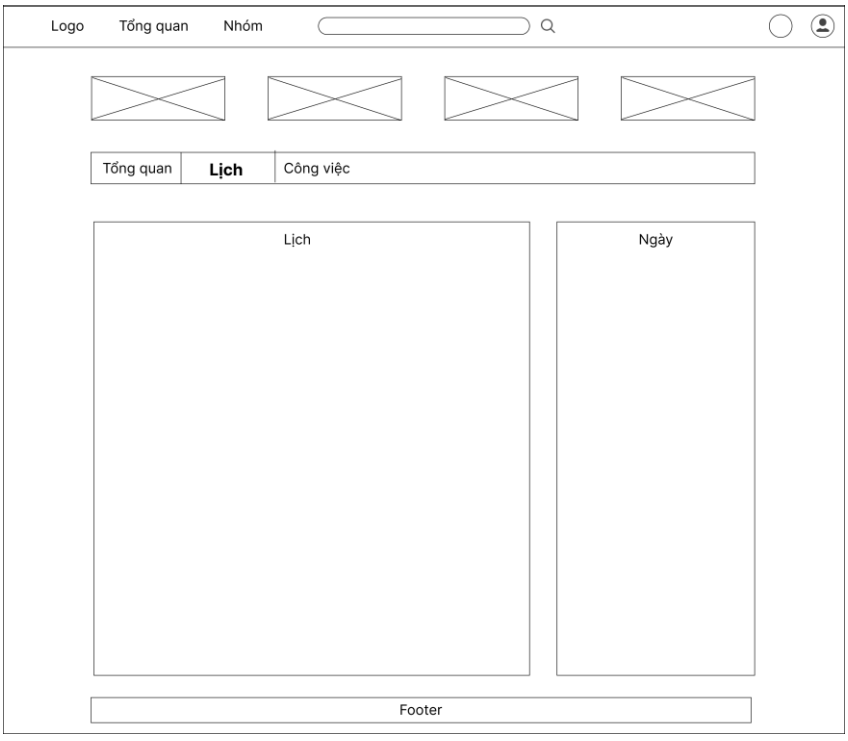
3.5.2.2 Thiết kế giao diện trang tổng quan

Giao diện trang tổng quan thể hiện tổng quát các thông tin như số lượng nhóm, dự án, công việc và công việc sắp đến hạn.



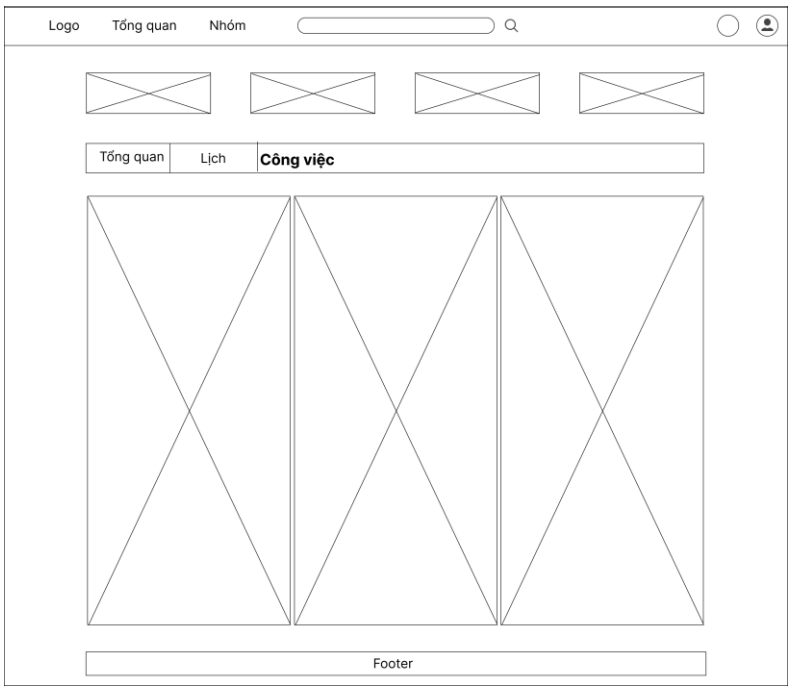
Hình 3.6 Thiết kế giao diện trang tổng quan

Trong giao diện trang lịch, người dùng có thể biết được trong ngày đó có bao nhiêu công việc và trạng thái khi chọn ngày.



Hình 3.7 Thiết kế giao diện trang lịch trong tổng quan

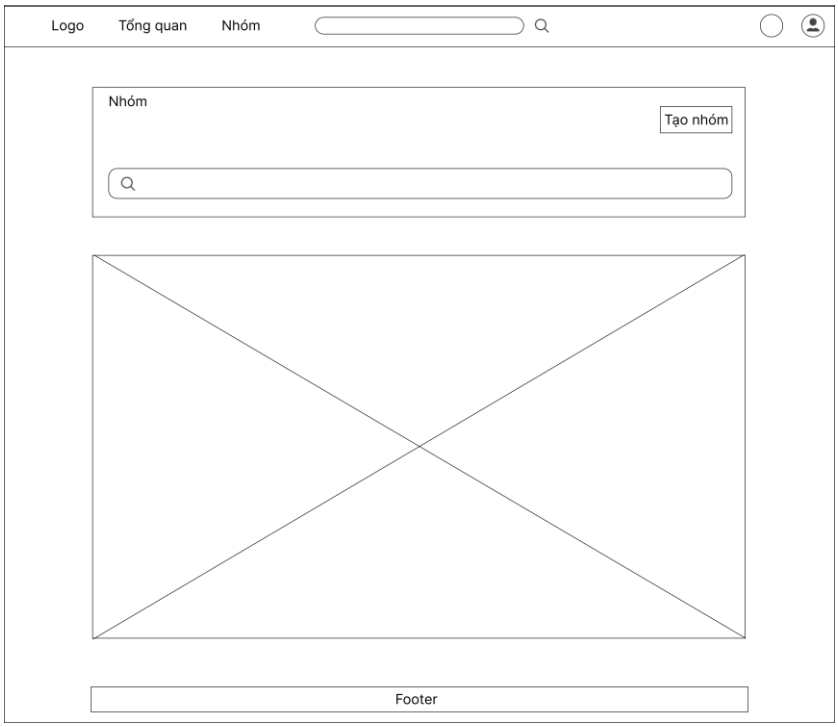
Giao diện công việc trong tổng quan, hiển thị tất cả công việc của người dùng. Thể hiện các thông tin như trang thái và số lượng công việc. Ngoài ra người dùng có thể trực tiếp cập nhật trạng thái bằng cách kéo thả.



Hình 3.8 Thiết kế giao diện trang công việc trong tổng quan

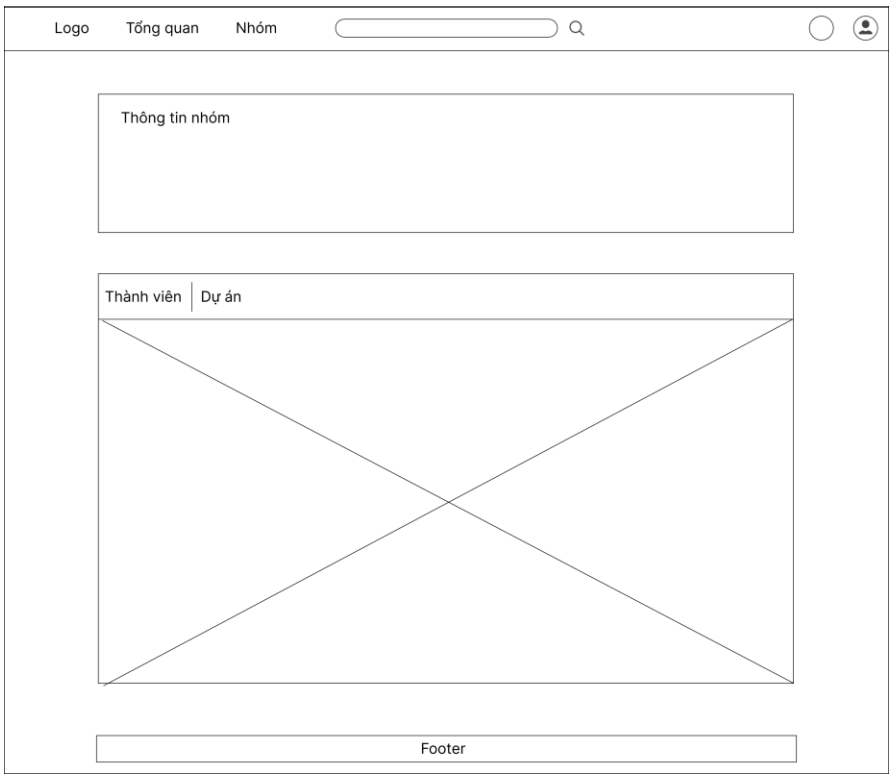
3.5.2.3 Thiết kế giao diện trang nhóm dự án

Trong giao diện trang nhóm dự án, người dùng có thể tạo nhóm và xem tất cả các nhóm hoặc các nhóm do mình tạo.



Hình 3.9 Thiết kế giao diện trang nhóm dự án

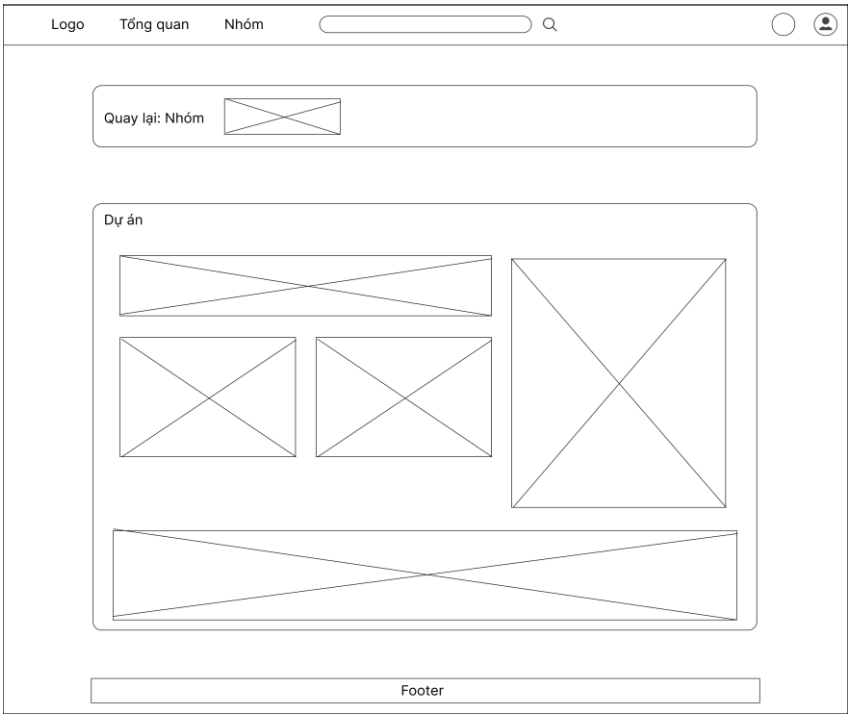
Trong trang chi tiết nhóm dự án, thể hiện danh sách các thành viên và dự án. Người dùng có thể thêm thành viên vào nhóm hoặc tạo dự án tại đây.



Hình 3.10 Thiết kế giao diện trang chi tiết nhóm dự án

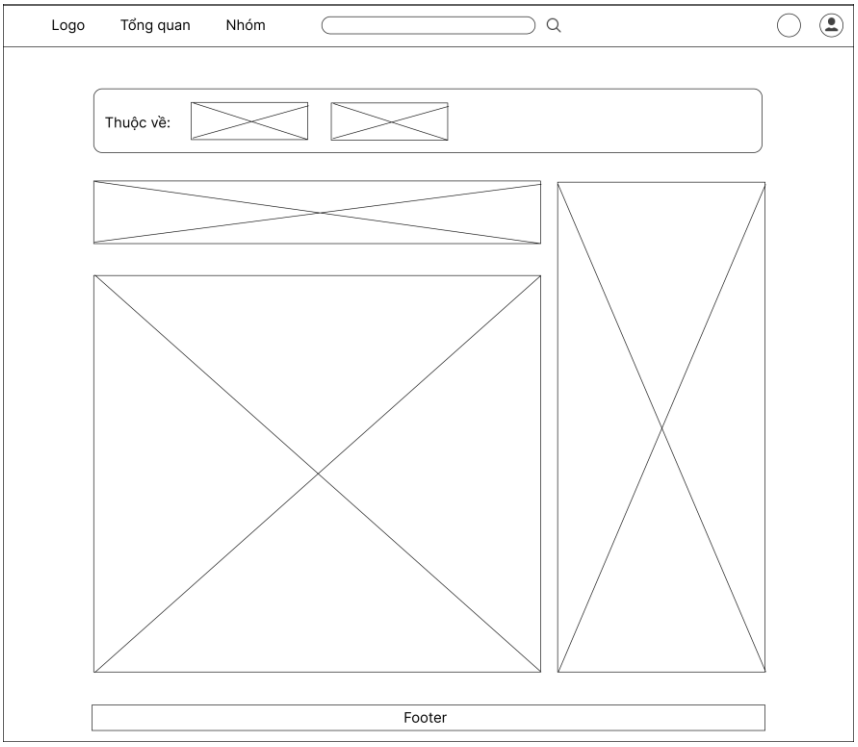
Trong trang chi tiết dự án, người dùng có thể xem các thông tin của dự án như tên, mô tả, thời gian thực hiện và tiến độ dự án. Ngoài ra còn có biểu đồ tròn thể hiện các trạng thái công việc thuộc dự án.

Phía trên footer là danh sách công việc, người dùng có thể xem hoặc lọc theo trạng thái và mức độ ưu tiên



Hình 3.11 Thiết kế trang chi tiết dự án

Trang chi tiết công việc thể hiện chi tiết các thông tin của công việc, người tạo và người được giao. Người dùng có thể cập nhật công việc tại đây. Ngoài ra còn có thể bình luận phía dưới các công việc.



Hình 3.12 Thiết kế trang chi tiết công việc

3.5.2.4 Thiết kế giao diện trang quản lý người dùng

Trang thông tin chung thể hiện các thông tin cá nhân như họ tên, email và thông tin tài khoản.

Hình 3.13 Thiết kế trang thông tin chung trong quản lý người dùng

3.6 CÀI ĐẶT VÀ HIỆN THỰC HÓA

Quá trình hiện thực hóa hệ thống chuyển đổi các thiết kế kiến trúc và cơ sở dữ liệu thành phần mềm hoạt động thực tế. Hệ thống được xây dựng theo mô hình kiến trúc vi dịch vụ, tách biệt hoàn toàn giữa frontend và các dịch vụ backend, được đóng gói và vận hành đồng bộ thông qua nền tảng Docker.

3.6.1 Cấu hình môi trường và cấu trúc thư mục

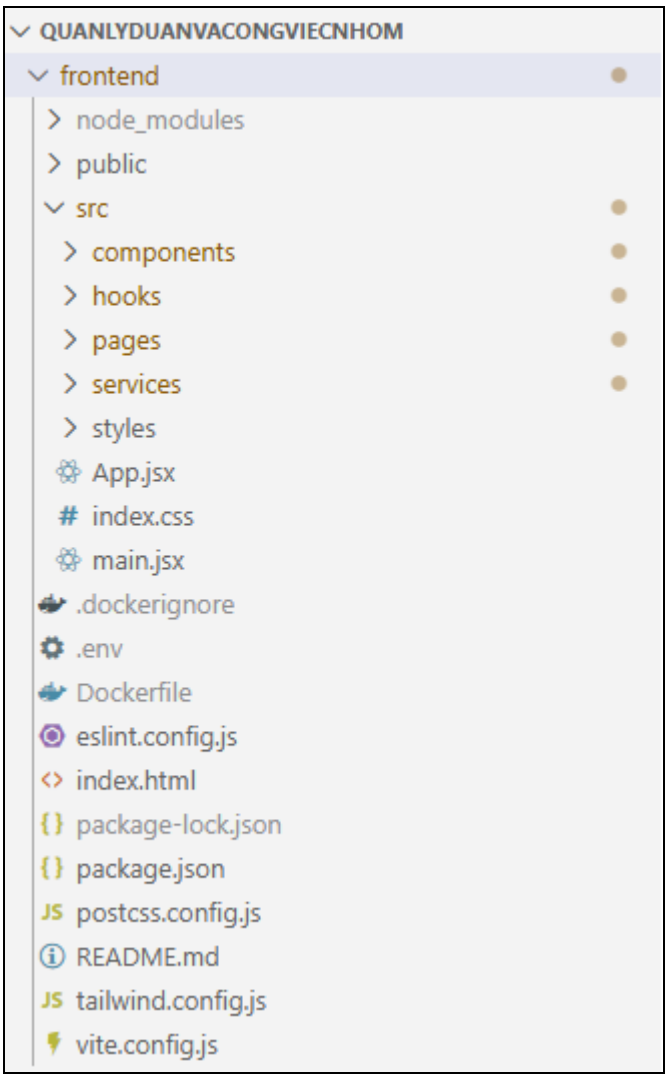
3.6.1.1 Frontend

Được xây dựng dựa trên thư viện ReactJS kết hợp với công cụ Vite nhằm tối ưu hóa hiệu năng và tốc độ phát triển. Cấu trúc mã nguồn tuân thủ các quy chuẩn thiết kế hiện đại, đảm bảo tính module hóa và khả năng bảo trì cao.

Thư mục mã nguồn chính (src): Đóng vai trò hạt nhân của ứng dụng, được tổ chức thành các phân hệ chức năng:

- components: Chứa các thành phần giao diện có khả năng tái sử dụng.
- pages: Quản lý các trang hiển thị và điều hướng.
- hooks: Lưu trữ các logic nghiệp vụ tùy biến (custom hooks).
- services: Xử lý các tác vụ gọi API và tích hợp dữ liệu từ Backend.
- App.jsx: Điểm khởi chạy chính của ứng dụng client.

Tài nguyên và cấu hình: Thư mục public lưu trữ các tài nguyên tĩnh. Các tệp cấu hình hệ thống như .env (biến môi trường), Dockerfile (đóng gói container) và tailwind.config.js (cấu hình giao diện) được thiết lập tại thư mục gốc để quản lý đồng bộ môi trường thực thi và hiển thị.



Hình 3.14 Cấu trúc thư mục frontend

Đối với phía frontend, tệp .env được sử dụng để thiết lập các biến môi trường công khai. Cụ thể, biến VITE_API_URL được khai báo để định nghĩa địa chỉ gốc của API Gateway. Việc tách biệt cấu hình này giúp ứng dụng linh hoạt chuyển đổi kết nối giữa môi trường phát triển và môi trường triển khai thực tế mà không cần sửa đổi mã nguồn cốt lõi.

```
# Địa chỉ API Gateway
VITE_API_URL=http://localhost:3000/api
```

Tệp tin Dockerfile được sử dụng để định nghĩa quy trình đóng gói ứng dụng Frontend vào môi trường Container. Cấu hình này thiết lập môi trường thực thi dựa trên Node.js phiên bản 20, tự động hóa việc sao chép mã nguồn, cài đặt các thư viện phụ thuộc và mở cổng kết nối 5173. Mục đích chính là tạo ra một môi trường phát triển đồng nhất, độc lập với hệ điều hành của máy chủ chứa.

```
FROM node:20
WORKDIR /app
# Sao chép file định nghĩa dependencies trước để tận dụng Docker
Cache
COPY package*.json ./
# Cài đặt thư viện (sử dụng legacy-peer-deps để tránh xung đột
phiên bản)
RUN npm install --legacy-peer-deps
COPY . .
# Mở cổng mặc định của Vite
EXPOSE 5173
# Khởi chạy server ở chế độ Development
CMD ["npm", "run", "dev"]
```

Tệp tin package.json đóng vai trò là bản kê khai trung tâm của dự án frontend. Nó định nghĩa các thông tin metadata, cấu hình script tự động hóa và quản lý toàn bộ các gói thư viện. Trong dự án này, hệ thống sử dụng Vite làm công cụ build tool để tối ưu hiệu suất, kết hợp với các thư viện hiện đại như TanStack Query để quản lý trạng thái server, Recharts để vẽ biểu đồ thống kê và Tailwind CSS cho việc thiết kế giao diện. Cấu hình script dev cũng đã được tùy chỉnh để hỗ trợ triển khai trên môi trường Docker.

```
{
  "name": "frontend",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite --host 0.0.0.0",
    "build": "vite build",
    "lint": "eslint .",
    "preview": "vite preview"
  },
  "dependencies": {
    "@tanstack/react-query": "^5.90.12",
    "framer-motion": "^12.23.24",
    "jwt-decode": "^3.1.2",
```

```
    "lucide-react": "^0.367.0",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-hot-toast": "^2.6.0",
    "react-icons": "^4.12.0",
    "react-router-dom": "^6.22.3",
    "react-toastify": "^9.1.3",
    "recharts": "^2.12.7"
  },
  "devDependencies": {
    "@eslint/js": "^9.33.0",
    "@tanstack/react-query-devtools": "^5.91.1",
    "@types/react": "^19.1.10",
    "@types/react-dom": "^19.1.7",
    "@vitejs/plugin-react": "^5.0.0",
    "autoprefixer": "^10.4.21",
    "eslint": "^9.33.0",
    "eslint-plugin-react-hooks": "^5.2.0",
    "eslint-plugin-react-refresh": "^0.4.20",
    "globals": "^16.3.0",
    "postcss": "^8.5.6",
    "tailwindcss": "^3.4.17",
    "vite": "^7.1.2"
  }}
```

Tệp tin tailwind.config.js đóng vai trò là trung tâm điều khiển cho hệ thống giao diện của dự án. Cấu hình này định nghĩa phạm vi hoạt động của Tailwind CSS thông qua thuộc tính content, chỉ định cho bộ biên dịch quét các tệp tin trong thư mục src. Cơ chế này giúp tối ưu hóa dung lượng file CSS đầu ra bằng cách chỉ sinh ra các lớp thực sự được sử dụng trong mã nguồn, đồng thời cho phép mở rộng hệ thống giao diện mặc định khi cần thiết.

```
/** @type {import('tailwindcss').Config} */
export default {
  content: [
    "./index.html",
```

```
    "./src/**/*.{js,ts,jsx,tsx}"
  ],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

Tệp tin vite.config.js đóng vai trò cấu hình server trong giai đoạn phát triển. Để đảm bảo hệ thống hoạt động ổn định trong môi trường Docker, cơ chế theo dõi thay đổi mã nguồn đã được điều chỉnh theo hướng chủ động kiểm tra. Nhờ đó, giao diện ứng dụng luôn được cập nhật kịp thời khi có thay đổi, góp phần nâng cao hiệu quả trong quá trình phát triển phần mềm.

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'
export default defineConfig({
  plugins: [react()],
  server: {
    host: '0.0.0.0',    // Cho phép truy cập từ máy host
    port: 5173,         // Đảm bảo port khớp với docker-compose
    watch: {
      usePolling: true, // Quan trọng: giúp hot reload trong
Docker
      interval: 100,    // Tần suất kiểm tra thay đổi file},},})
```

3.6.1.2 Backend

A. Services

Bao gồm các thư mục con đại diện cho từng vi dịch vụ độc lập: auth-service, team-service, project-service, task-service, notification-service. Mỗi dịch vụ sở hữu cấu trúc riêng biệt gồm: models, controllers, routes, middleware, config và untils.

Models: Định nghĩa cấu trúc dữ liệu và các ràng buộc cho từng thực thể, đóng vai trò là lớp tương tác trực tiếp với cơ sở dữ liệu để thực hiện các truy vấn và đảm bảo tính toàn vẹn của dữ liệu.

Controllers: Chứa toàn bộ logic nghiệp vụ của dịch vụ, chịu trách nhiệm tiếp nhận yêu cầu từ Routes, xử lý tính toán, gọi dữ liệu từ Models và trả về phản hồi cho Client.

Routes: Hệ thống định tuyến quản lý các điểm cuối và phương thức HTTP (GET, POST, PUT, DELETE), có nhiệm vụ điều hướng chính xác các yêu cầu từ người dùng đến hàm xử lý tương ứng trong Controller.

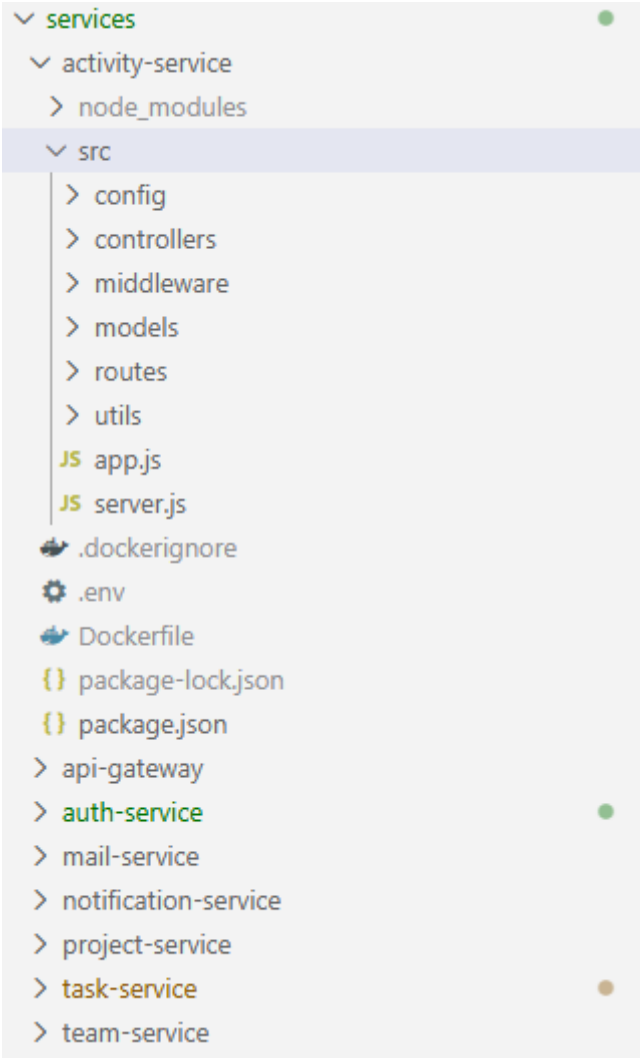
Middleware: Các hàm trung gian được thực thi trước khi request đến được Controller, đảm nhiệm các tác vụ quan trọng như xác thực danh tính, phân quyền, kiểm tra dữ liệu đầu vào và xử lý lỗi tập trung.

Config: Quản lý tập trung các tham số cấu hình hệ thống (như chuỗi kết nối cơ sở dữ liệu), giúp tách biệt mã nguồn khỏi các thiết lập môi trường.

App: Chịu trách nhiệm khởi tạo ứng dụng Express, tích hợp các middleware toàn cục và đăng ký các Routes đã định nghĩa.

Server: Đóng vai trò là điểm khởi chạy của dịch vụ, thực hiện kết nối đến cơ sở dữ liệu và lắng nghe các yêu cầu trên cổng mạng đã định.

Utils (Utilities): Chứa các hàm tiện ích dùng chung để tái sử dụng mã nguồn. Thành phần quan trọng nhất tại đây là httpClient, đóng vai trò là lớp giao tiếp nội bộ, giúp thực hiện các lệnh gọi HTTP giữa các microservices một cách đồng bộ và an toàn.



Hình 3.15 Cấu trúc thư mục của services

B. Môi trường thực thi

Sử dụng Node.js 20 phiên bản Alpine Linux (node:20-alpine). Việc sử dụng bản phân phối Alpine giúp giảm thiểu đáng kể kích thước của Docker Image (từ ~1GB xuống còn dưới 200MB), tăng tốc độ tải và khởi động container, đồng thời giảm bề mặt tấn công bảo mật.

```
# Sử dụng base image Alpine để tối ưu dung lượng (nhẹ hơn bản
chuẩn ~80%)
FROM node:20-alpine
# Thiết lập thư mục làm việc trong container
WORKDIR /usr/src/app
# Copy file dependency trước để tận dụng Docker Layer Caching
COPY package*.json ./

# Cài đặt thư viện
RUN npm install
# Copy toàn bộ mã nguồn
COPY . .
# Mở port (Thay đổi số port này tùy theo từng Service cụ thể)
EXPOSE 5001
# Lưu ý:
# Auth: 5001, Team: 5002, Project: 5003,
# Task: 5004, Notification: 5005, Activity: 5007
# Lệnh khởi chạy mặc định
CMD ["npm", "run", "dev"]
```

C. Cấu hình môi trường .env

Mỗi dịch vụ hoạt động trên một cổng riêng biệt để tránh xung đột mạng. Các kết nối cơ sở dữ liệu sử dụng tên miền nội bộ Docker (mongo thay vì localhost) để đảm bảo các container nhìn thấy nhau trong cùng một mạng ảo.

```
PORT=500x
NODE_ENV=development
MONGO_URI=mongodb://mongo:27017/ten_database_rieng_biet
JWT_SECRET=quocdamchuyennganh2025
```

PORT: Định nghĩa cổng mạng (Network Port) mà dịch vụ sẽ lắng nghe. Cần thay thế giá trị 500x bằng số cổng cụ thể được quy hoạch cho từng dịch vụ (Ví dụ: Auth Service là 5001, Team Service là 5002...).

NODE_ENV: Xác định môi trường thực thi của ứng. Hiện tại đang thiết lập ở chế độ development để hỗ trợ quá trình gỡ lỗi và ghi log chi tiết.

MONGO_URI: Chuỗi kết nối đến cơ sở dữ liệu MongoDB.

- Host: Sử dụng tên mongo thay vì localhost để trỏ đến container database trong mạng Docker nội bộ.

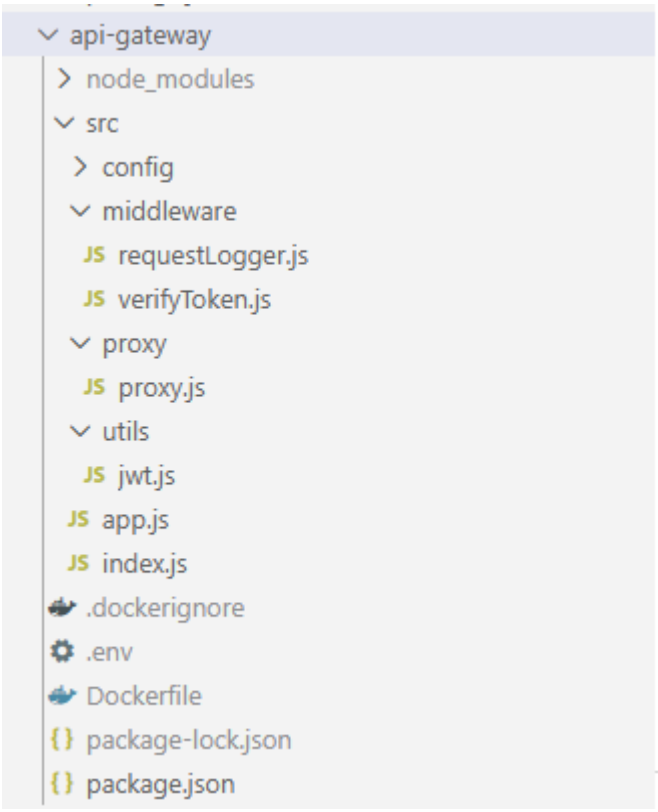
- Database Name: Cần thay thế ten_database_rieng_biet bằng tên cơ sở dữ liệu tương ứng của từng dịch vụ (Ví dụ: auth_service_db, task_service_db) để đảm bảo nguyên tắc tách biệt dữ liệu.

JWT_SECRET: Khóa bí mật dùng để ký và xác thực JSON Web Token. Giá trị này bắt buộc phải đồng nhất giữa tất cả các vi dịch vụ. Điều này cho phép các dịch vụ con (Team, Task, Project...) có thể giải mã và xác thực được token do Auth Service cấp phát.

3.6.1.3 API Gateway

API Gateway đóng vai trò là điểm truy cập duy nhất cho toàn bộ hệ thống. Nó chịu trách nhiệm tiếp nhận tất cả các yêu cầu từ phía Client, thực hiện xác thực, định tuyến đến các vi dịch vụ tương ứng và trả về kết quả thống nhất. Cấu trúc mã nguồn của Gateway được tổ chức thành các module chuyên biệt:

- Config: Quản lý các tham số cấu hình và biến môi trường.
- Middleware: Xử lý các tác vụ trung gian như xác thực JWT, ghi log.
- Proxy: Cấu hình chuyển tiếp yêu cầu đến các service đích.
- Utils: Các hàm tiện ích dùng chung.
- File App và Index: Khởi tạo ứng dụng và khởi chạy server.



Hình 3.16 Cấu trúc thư mục của API Gateway

Tệp tin .env của API Gateway đóng vai trò như một bản đồ định tuyến. Ngoài việc cấu hình cổng hoạt động và khóa bảo mật JWT, tệp tin này định nghĩa toàn bộ địa chỉ của các vi dịch vụ nội bộ. Các địa chỉ này sử dụng Service Name (ví dụ: http://auth-service:5001) thay vì địa chỉ IP cụ thể, tận dụng cơ chế phân giải tên miền của mạng Docker nội bộ để kết nối các container với nhau.

```
# Cấu hình cơ bản
PORT=3000
NODE_ENV=development

# JWT cấu hình chung (Đồng bộ với các Service con)
JWT_SECRET=quocdamchuyennganh2025
JWT_EXPIRES_IN=1d

# Đường dẫn các service nội bộ (Docker Network Map)
AUTH_SERVICE_URL=http://auth-service:5001/api/auth
USER_SERVICE_URL=http://auth-service:5001/api/user
TEAM_SERVICE_URL=http://team-service:5002/api/teams
PROJECT_SERVICE_URL=http://project-service:5003/api/projects
```

```
# Task Service (Gộp 3 modules vào cùng 1 host port 5004)
TASK_SERVICE_URL=http://task-service:5004/api/tasks
TASK_COMMENT_SERVICE_URL=http://task-service:5004/api/task-
comments
TASK_ATTACHMENT_SERVICE_URL=http://task-service:5004/api/task-
attachments

# Các Service bổ trợ
NOTIFICATION_SERVICE_URL=http://notification-
service:5005/api/notifications
MAIL_SERVICE_URL=http://mail-service:5006/api/mail
ACTIVITY_SERVICE_URL=http://activity-service:5007/api/activity-
logs
```

API Gateway được đóng gói bằng Dockerfile tối ưu hóa cho môi trường Production. Sử dụng nền tảng node:20-alpine giúp giảm thiểu kích thước image. Container được cấu hình mở cổng 3000 để giao tiếp với thế giới bên ngoài (Frontend/Client) và sử dụng lệnh node src/index.js để khởi chạy trực tiếp ứng dụng, đảm bảo hiệu suất ổn định nhất.

```
FROM node:20-alpine
WORKDIR /usr/src/app
COPY package*.json ./
RUN npm install
COPY . .
# Gateway là cổng giao tiếp chính với Client nên dùng port 3000
EXPOSE 3000
# Chạy trực tiếp file index để khởi động server
CMD ["node", "src/index.js"]
```

3.6.1.4 Cấu hình môi trường container

A. Tổng quan về cấu hình container

Tệp tin docker-compose.yml đóng vai trò là kịch bản điều phối trung tâm cho toàn bộ hệ thống. Thay vì khởi chạy thủ công từng thành phần, Docker Compose cho phép định nghĩa và vận hành đồng thời 11 dịch vụ (bao gồm Database, Backend Services, API

Gateway và Frontend) trong một mạng ảo thống nhất. Cấu hình này mô phỏng chính xác môi trường sản phẩm thực tế ngay trên máy cục bộ, đảm bảo tính nhất quán giữa quá trình phát triển và triển khai.

B. Phân tích chi tiết các nhóm dịch vụ

Lớp hạ tầng dữ liệu

mongo: Container cơ sở dữ liệu MongoDB phiên bản 7.0. Dữ liệu được lưu trữ bền vững thông qua volume mongo_data, đảm bảo không bị mất mát khi khởi động lại container.

Lớp vi dịch vụ nghiệp vụ

Bao gồm 5 dịch vụ độc lập: Auth, Team, Project, Task, Notification.

Cơ chế depends_on: - mongo đảm bảo các dịch vụ này chỉ khởi chạy sau khi cơ sở dữ liệu đã sẵn sàng.

Lớp định tuyến

api-gateway (Cổng 3000): Là điểm truy cập duy nhất, kết nối Frontend với các vi dịch vụ phía sau. Nó phụ thuộc vào sự sẵn sàng của các dịch vụ lõi như auth-service và team-service.

Lớp giao diện người dùng: frontend (Cổng 5173): Chạy môi trường React/Vite.

Điểm nổi bật về kỹ thuật: Cấu hình này cho phép cập nhật code theo thời gian thực. Mọi chỉnh sửa trên máy tính sẽ có hiệu lực ngay lập tức trong ứng dụng mà không cần chờ đợi quá trình đóng gói lại, giúp tối ưu hóa hiệu suất làm việc.

```
version: '3.9'
services:
  # 1. DATABASE (Hạ tầng dùng chung)
  mongo:
    image: mongo:7
    ports: ["27017:27017"]
    volumes: ["mongo_data:/data/db"]
  # 2. BACKEND SERVICES (Mẫu đại diện cho các vi dịch vụ)
  # (Auth, Team, Project, Task... đều tuân theo cấu trúc này)
  auth-service:
```

```
    build:
      context: ./services/auth-service
    ports: ["5001:5001"]
    env_file: ["./services/auth-service/.env"]
    depends_on: ["mongo"]
    command: npm run dev
  # ... (Các service Team, Project, Task, Notification cấu hình
tương tự) ...
  # 3. API GATEWAY (Cổng kết nối trung tâm)
  api-gateway:
    build:
      context: ./services/api-gateway
    ports: ["3000:3000"]
    env_file: ["./services/api-gateway/.env"]
    depends_on: ["auth-service", "team-service"]
  # 4. FRONTEND (Giao diện người dùng)
  frontend:
    build:
      context: ./frontend
    ports: ["5173:5173"]
    volumes: ["./frontend:/app"]

volumes:
  mongo_data:
```

3.6.2 Triển khai API

3.6.2.1 Kiến trúc triển khai

Hệ thống được xây dựng dựa trên mô hình Microservices, trong đó các chức năng nghiệp vụ được tách rời thành các dịch vụ độc lập để dễ dàng bảo trì và mở rộng. Cấu trúc cụ thể bao gồm:

API Gateway (Port 3000): Đóng vai trò là điểm truy cập trung gian duy nhất. Gateway chịu trách nhiệm tiếp nhận yêu cầu từ Client, thực hiện xác thực và điều hướng đến các dịch vụ đích.

Các Service thành phần:

- Auth Service: Quản lý đăng ký, đăng nhập và cấp phát JWT.
- Team Service: Quản lý thông tin nhóm làm việc và thành viên.
- Project Service: Xử lý logic tạo, sửa, xóa và theo dõi tiến độ dự án.
- Task Service: Quản lý công việc, bình luận.
- Notification/Mail Service: Xử lý thông báo và gửi email.

Toàn bộ các API được thiết kế theo chuẩn RESTful, sử dụng các phương thức HTTP tiêu chuẩn (GET, POST, PUT, DELETE) và định dạng dữ liệu trao đổi là JSON.

3.6.2.2 Cài đặt các API chính

Dưới đây là mô tả các nhóm API cốt lõi đã được triển khai và đưa vào hoạt động:

Nhóm API Xác thực và người dùng (Auth Service):

- POST /api/auth/register: Đăng ký tài khoản mới.
- POST /api/auth/login: Đăng nhập, trả về Access Token và Refresh Token.
- GET /api/user/profile: Lấy thông tin cá nhân (yêu cầu Token).

Nhóm API quản lý dự án (Project Service):

- POST /api/projects: Tạo dự án mới (có logic kiểm tra quyền trong Team).
- GET /api/projects/:id: Lấy chi tiết dự án.
- PUT /api/projects/:id: Cập nhật tiến độ hoặc thông tin dự án.

Nhóm API quản lý công việc (Task Service):

- POST /api/tasks: Tạo công việc mới trong dự án.
- PATCH /api/tasks/:id/status: Cập nhật trạng thái công.

Nhóm API quản lý nhóm và thành viên (Team Service):

- POST /api/teams: Tạo nhóm làm việc mới và thiết lập quyền Leader.
- GET /api/teams: Lấy danh sách tất cả các nhóm mà người dùng đang tham gia.
- POST /api/teams/:id/members/batch: Thêm hàng loạt thành viên vào nhóm.

– DELETE /api/teams/:id/members/:uid: Xóa thành viên ra khỏi nhóm.

– POST /api/teams/:id/leave: Cho phép thành viên tự rời khỏi nhóm.

Nhóm API thông báo (Notification Service):

– GET /api/notifications/my: Lấy danh sách thông báo của người dùng hiện tại (sắp xếp theo thời gian).

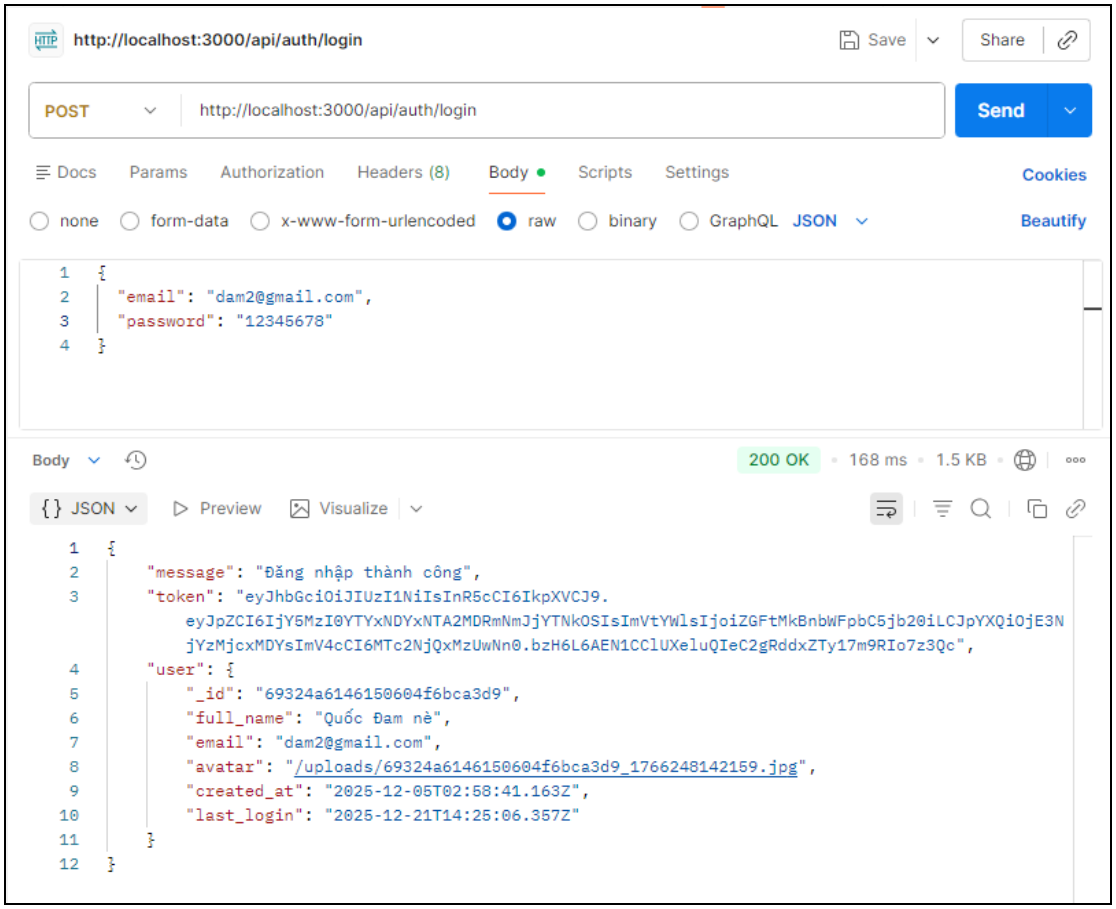
– PUT /api/notifications/:id/read: Đánh dấu một thông báo là đã đọc.

– POST /api/notifications/send: API nội bộ gửi email thông báo đến người dùng.

3.6.2.3 Kiểm thử API

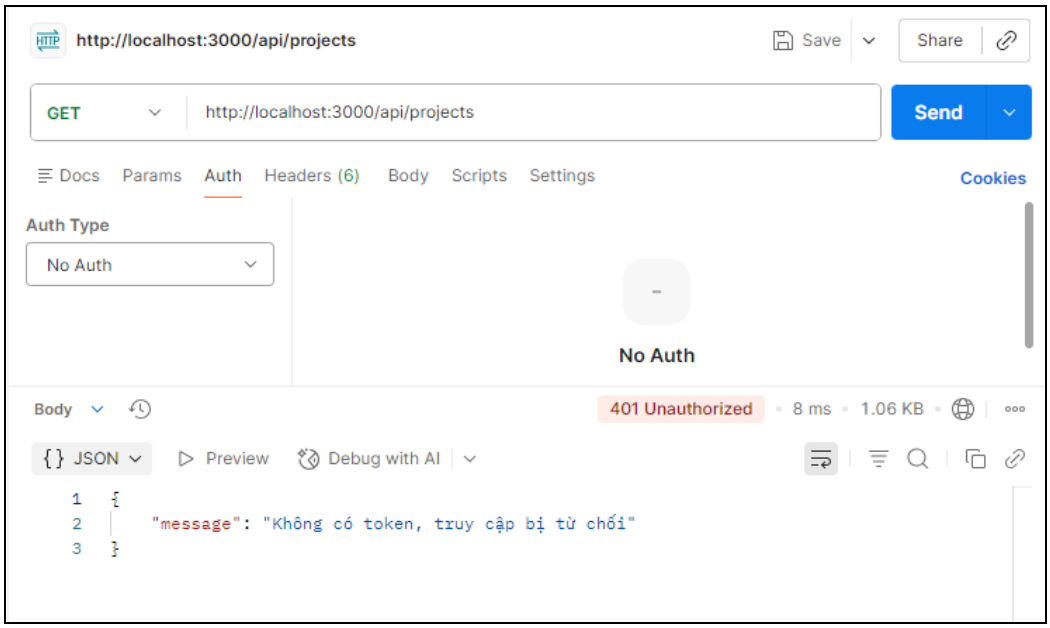
Quá trình kiểm thử được thực hiện bằng công cụ Postman để đảm bảo độ chính xác và tính bảo mật của dữ liệu:

Kiểm tra luồng dữ liệu hợp lệ: Đảm bảo các request gửi lên đúng định dạng JSON trả về mã 200 OK hoặc 201 Created.



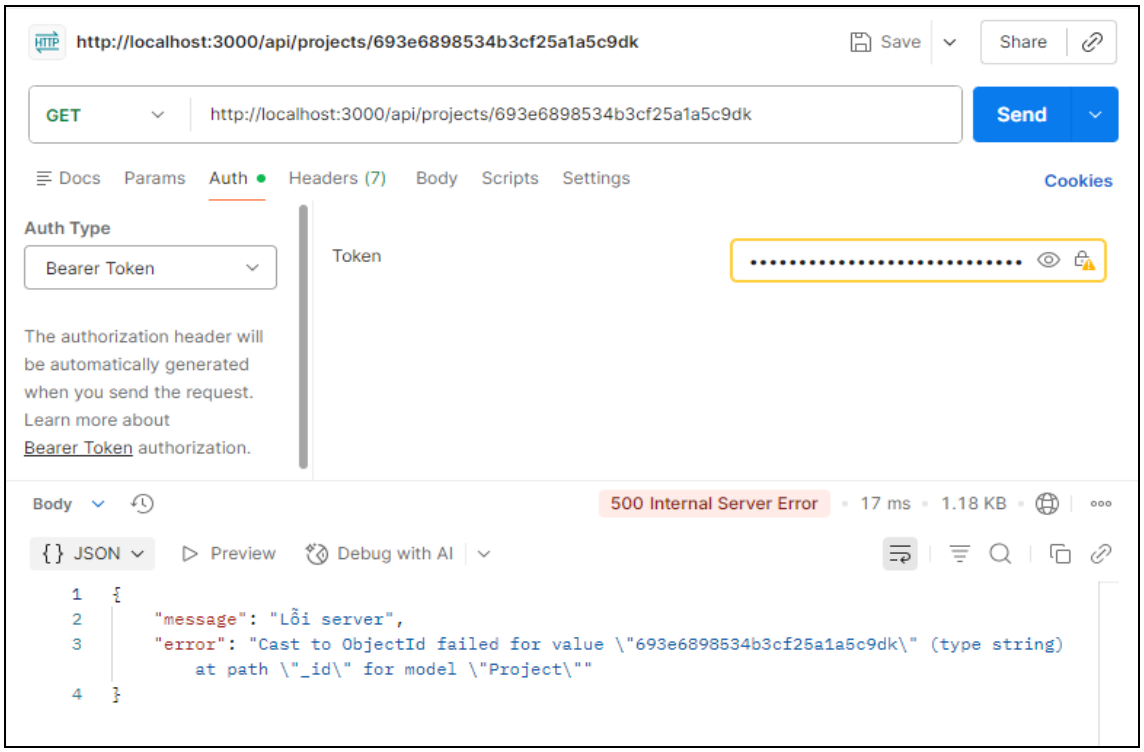
Hình 3.17 Đăng nhập thành công trả về mã 200

Kiểm tra cơ chế bảo mật: Các API quan trọng yêu cầu Header Authorization: Bearer <token>. Nếu thiếu hoặc token hết hạn, hệ thống trả về lỗi 401 Unauthorized.



Hình 3.18 Truy cập thiếu Token bị từ chối với mã 401

Kiểm tra xử lý lỗi: Hệ thống trả về thông báo lỗi rõ ràng (mã 400 Bad Request hoặc 500 Internal Server Error) khi dữ liệu đầu vào sai hoặc có lỗi hệ thống, thay vì làm sập server.



Hình 3.19 Xử lý ngoại lệ ID sai định dạng trả về mã 500

3.6.3 Xác thực và đăng nhập bằng Google OAuth 2.0

3.6.3.1 Kiến trúc tổng thể của cơ chế xác thực

Hệ thống xác thực được triển khai theo mô hình Client – Server, trong đó:

Frontend chịu trách nhiệm khởi tạo yêu cầu đăng nhập Google và tiếp nhận kết quả xác thực.

Backend (Node.js) đóng vai trò trung gian, xác thực người dùng với Google, quản lý thông tin người dùng và phát hành JWT.

Cơ sở dữ liệu MongoDB lưu trữ thông tin người dùng và trạng thái xác thực.

3.6.3.2 Cấu hình môi trường và thông số OAuth

Các tham số cấu hình được quản lý thông qua biến môi trường nhằm đảm bảo tính bảo mật và linh hoạt khi triển khai hệ thống:

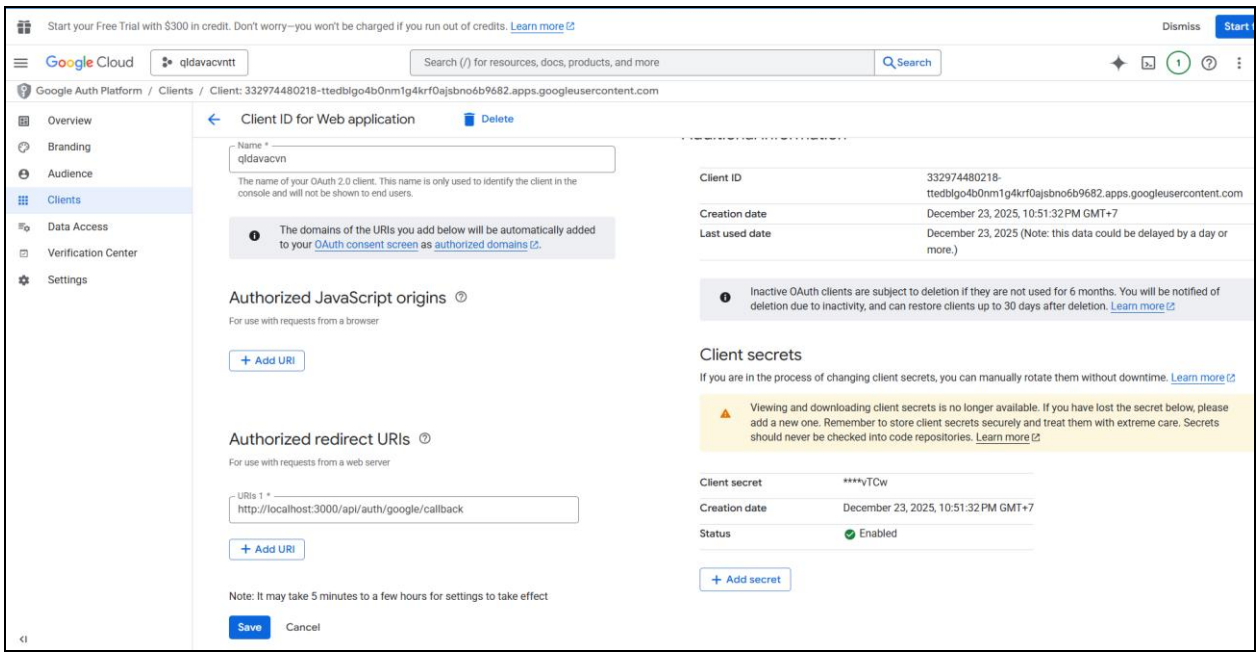
– GOOGLE_CLIENT_ID: Định danh ứng dụng web với Google OAuth.

– GOOGLE_CLIENT_SECRET: Khóa bí mật dùng để xác thực phía máy chủ với Google.

– GOOGLE_CALLBACK_URL: Địa chỉ callback để Google chuyển hướng sau khi xác thực thành công.

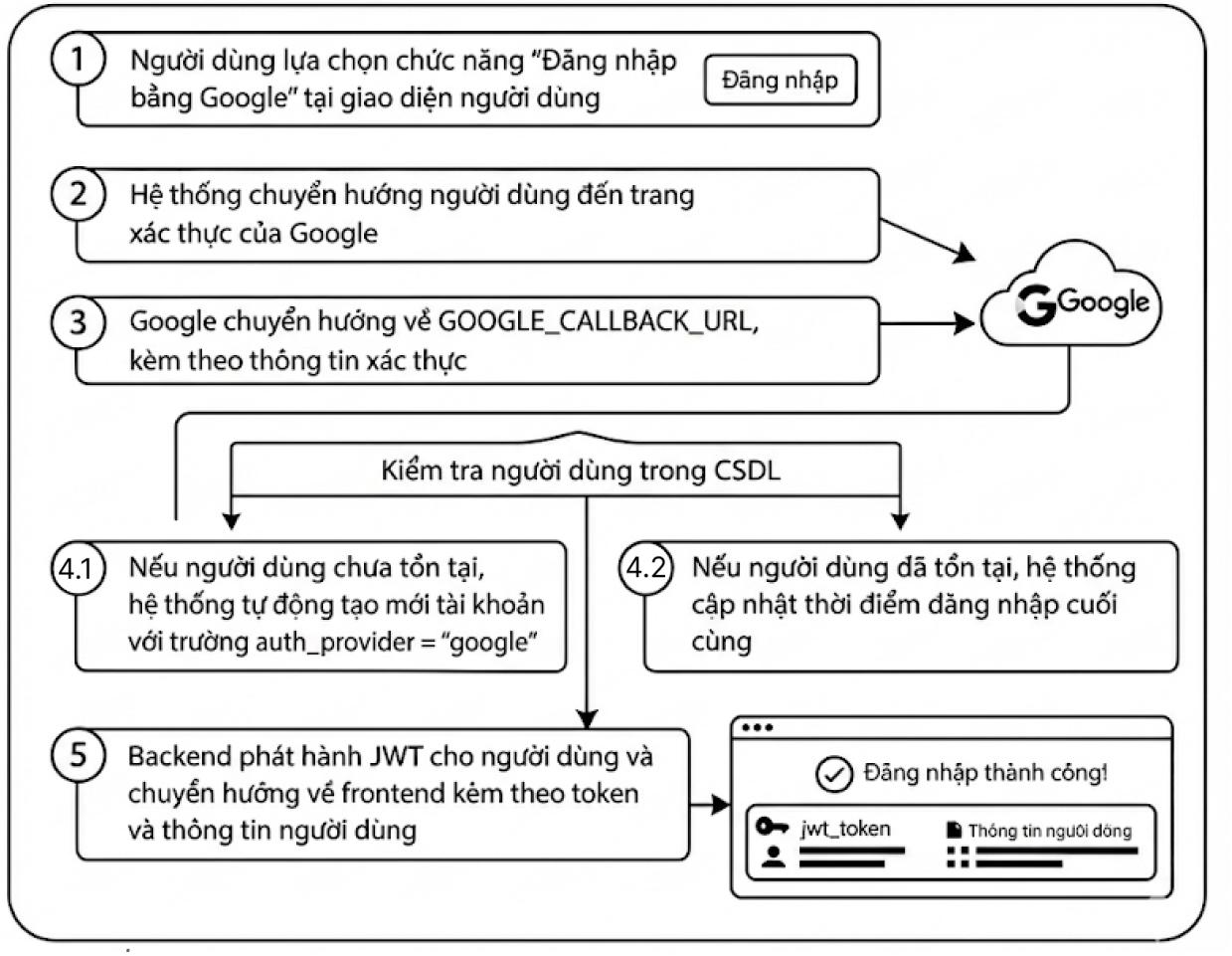
– FRONTEND_URL: Địa chỉ giao diện người dùng tiếp nhận kết quả đăng nhập.

```
# Google OAuth
GOOGLE_CLIENT_ID=332974480218-
ttedblgo4b0nm1g4krf0ajsbno6b9682.apps.googleusercontent.com
GOOGLE_CLIENT_SECRET=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
GOOGLE_CALLBACK_URL=http://localhost:3000/api/auth/google/callba
ck
FRONTEND_URL=http://localhost:5173
```



Hình 3.20 Cấu hình đăng nhập bằng Google cho backend

3.6.3.3 Quy trình đăng nhập bằng Google



Hình 3.21 Quy trình đăng nhập bằng Google

3.6.3.4 Hiện thực xử lý Google OAuth Callback

Sau khi xác thực thành công với Google, hàm xử lý callback tại backend thực hiện các nhiệm vụ chính:

Tiếp nhận thông tin người dùng đã được Google xác thực.

Sinh mã thông báo JWT dựa trên thông tin người dùng.

Chuẩn hóa dữ liệu phản hồi (ID, họ tên, email, ảnh đại diện, thời điểm tạo tài khoản).

Chuyển hướng người dùng về frontend cùng với token và thông tin người dùng.

CHƯƠNG 4 KẾT QUẢ NGHIÊN CỨU

4.1 KẾT QUẢ TRIỂN KHAI HỆ THỐNG

4.1.1 Môi trường triển khai Docker

Hệ thống đã được đóng gói và triển khai thành công trên môi trường Docker. Quá trình khởi chạy được thực hiện thông qua công cụ Docker Compose, giúp điều phối đồng thời toàn bộ các dịch vụ thành phần trong một mạng ảo thống nhất.

Kết quả thực nghiệm cho thấy toàn bộ container dịch vụ đều hoạt động ổn định ở trạng thái running (đang chạy), không phát sinh lỗi xung đột tài nguyên hay cổng kết nối. Cụ thể danh sách các dịch vụ bao gồm:

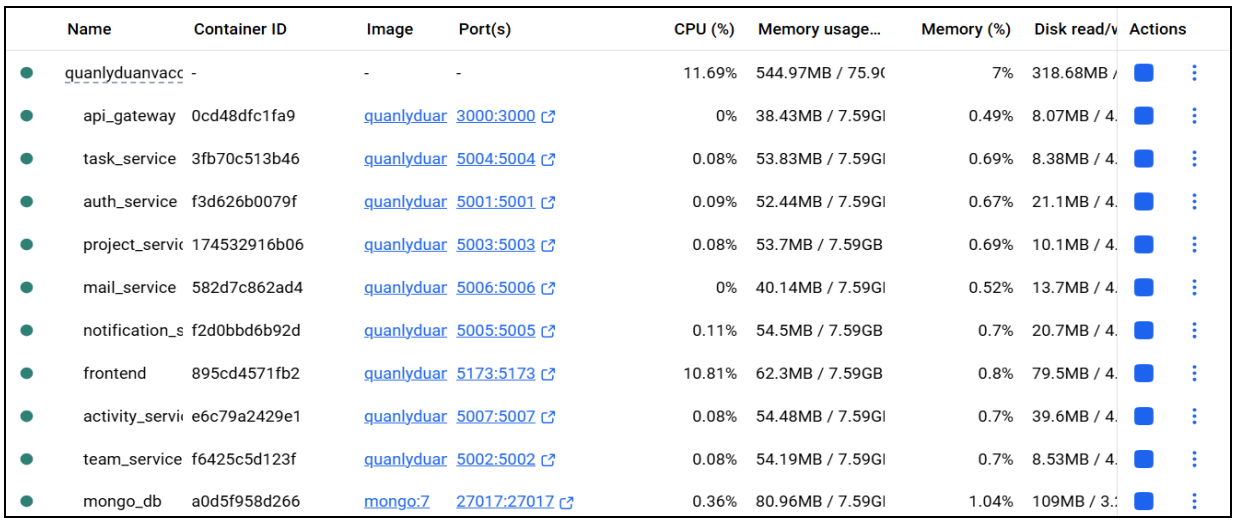
Lớp giao diện (Frontend): Container frontend hoạt động trên cổng 5173, đóng vai trò là điểm tương tác với người dùng.

Cổng kết nối (API Gateway): Container api_gateway hoạt động trên cổng 3000, tiếp nhận yêu cầu và điều hướng đến các vi dịch vụ.

Các vi dịch vụ nghiệp vụ (Microservices): Các dịch vụ backend hoạt động độc lập trên các cổng riêng biệt đúng theo thiết kế đã đề ra:

- auth_service (5001): Xử lý xác thực.
- team_service (5002): Quản lý nhóm.
- project_service (5003): Quản lý dự án.
- task_service (5004): Quản lý công việc.
- notification_service (5005) và mail_service (5006): Hệ thống thông báo.
- activity_service (5007): Ghi nhật ký hoạt động.

Cơ sở dữ liệu: Container mongo_db hoạt động trên cổng tiêu chuẩn 27017, đảm bảo khả năng lưu trữ dữ liệu bền vững cho toàn bộ hệ thống.



	Name	Container ID	Image	Port(s)	CPU (%)	Memory usage...	Memory (%)	Disk read/v	Actions
●	quanlyduanvacc	-	-	-	11.69%	544.97MB / 75.9(	7%	318.68MB /	
●	api_gateway	0cd48dfc1fa9	quanlyduar	3000:3000	0%	38.43MB / 7.59GI	0.49%	8.07MB / 4.	
●	task_service	3fb70c513b46	quanlyduar	5004:5004	0.08%	53.83MB / 7.59GI	0.69%	8.38MB / 4.	
●	auth_service	f3d626b0079f	quanlyduar	5001:5001	0.09%	52.44MB / 7.59GI	0.67%	21.1MB / 4.	
●	project_servic	174532916b06	quanlyduar	5003:5003	0.08%	53.7MB / 7.59GB	0.69%	10.1MB / 4.	
●	mail_service	582d7c862ad4	quanlyduar	5006:5006	0%	40.14MB / 7.59GI	0.52%	13.7MB / 4.	
●	notification_s	f2d0bbd6b92d	quanlyduar	5005:5005	0.11%	54.5MB / 7.59GB	0.7%	20.7MB / 4.	
●	frontend	895cd4571fb2	quanlyduar	5173:5173	10.81%	62.3MB / 7.59GB	0.8%	79.5MB / 4.	
●	activity_servi	e6c79a2429e1	quanlyduar	5007:5007	0.08%	54.48MB / 7.59GI	0.7%	39.6MB / 4.	
●	team_service	f6425c5d123f	quanlyduar	5002:5002	0.08%	54.19MB / 7.59GI	0.7%	8.53MB / 4.	
●	mongo_db	a0d5f958d266	mongo:7	27017:27017	0.36%	80.96MB / 7.59GI	1.04%	109MB / 3.:	

Hình 4.1 Danh sách các container dịch vụ đang hoạt động trên Docker Desktop

4.1.2 Đánh giá hiệu quả sử dụng tài nguyên

Các dịch vụ backend (như auth_service, task_service) tiêu tốn lượng bộ nhớ RAM rất thấp (trung bình từ 40MB - 60MB mỗi dịch vụ). Điều này đạt được nhờ việc sử dụng nền tảng Alpine Linux (node:20-alpine) trong quá trình đóng gói, giúp hệ thống nhẹ và khởi động nhanh.

Dịch vụ frontend chiếm tài nguyên cao hơn một chút do đang chạy ở chế độ Development server (Vite) để hỗ trợ tính năng cập nhật tức thời, tuy nhiên vẫn nằm trong ngưỡng cho phép của máy tính cá nhân thông thường.

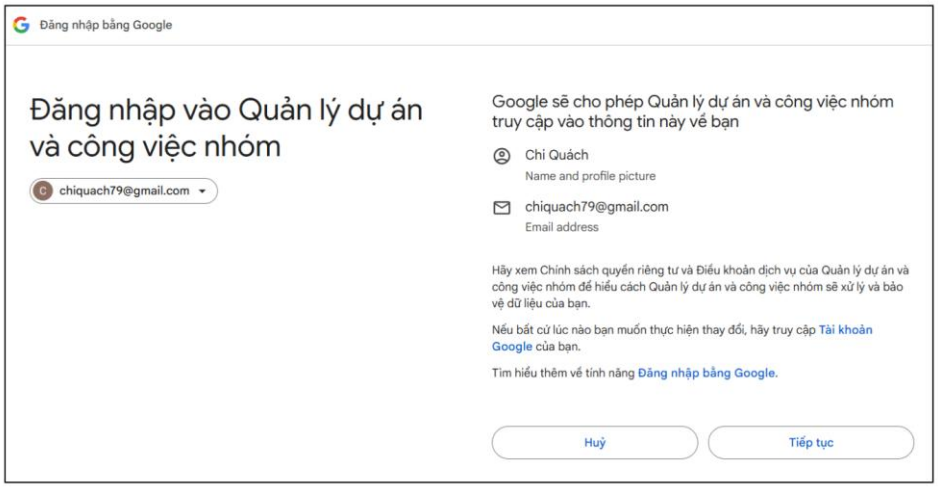
Kết quả này khẳng định tính khả thi của việc triển khai kiến trúc Microservices phức tạp ngay trên môi trường máy tính cá nhân mà không yêu cầu phần cứng máy chủ quá mạnh mẽ.

4.2 KẾT QUẢ GIAO DIỆN VÀ CHỨC NĂNG NGƯỜI DÙNG

4.2.1 Phân hệ xác thực và người dùng

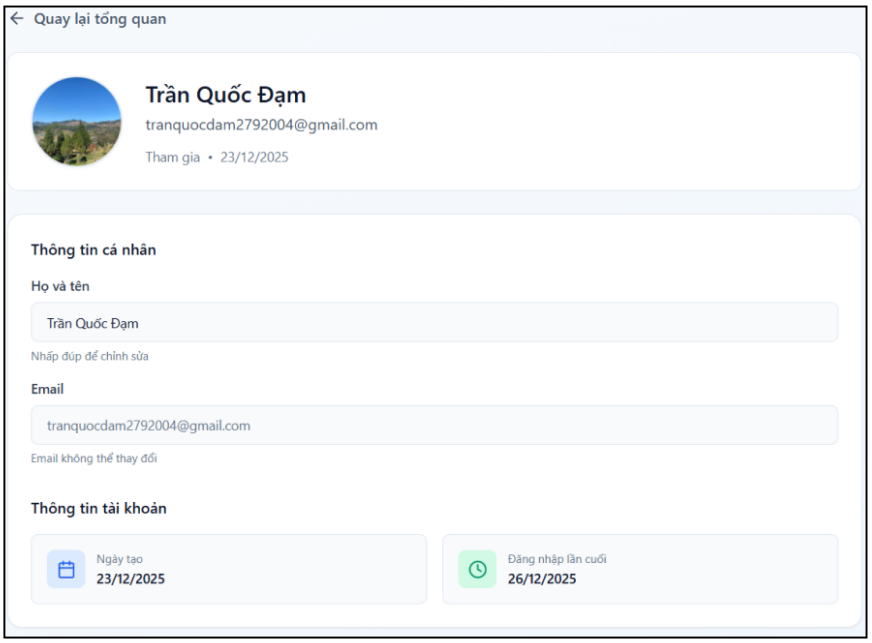
Hệ thống đã hoàn thiện quy trình xác thực bảo mật sử dụng JWT và tích hợp thành công phương thức đăng nhập thông qua Google OAuth 2.0. Người dùng có thể đăng ký tài khoản mới hoặc truy cập nhanh bằng tài khoản Google, giúp giảm thiểu thao tác nhập liệu và tăng tính tiện lợi.

Hình 4.2 Đăng nhập và chọn tài khoản Google



Hình 4.3 Xác nhận đăng ký tài khoản vào hệ thống

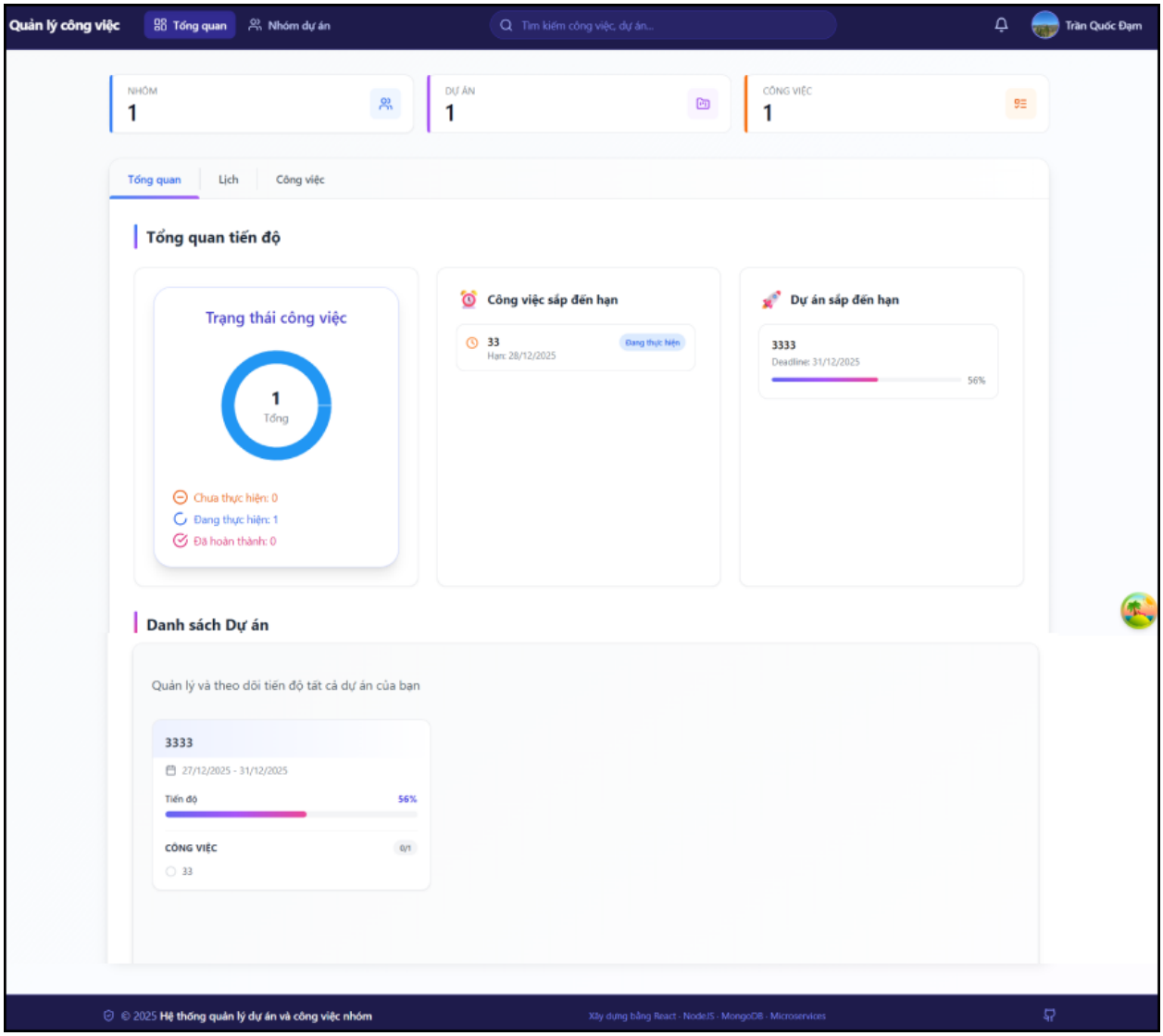
Sau khi đăng nhập, người dùng có thể quản lý và cập nhật thông tin cá nhân. Giao diện trang hồ sơ (Profile) hiển thị chi tiết các thông tin định danh và cho phép chỉnh sửa dữ liệu, đảm bảo tính nhất quán của tài khoản trong toàn hệ thống.



Hình 4.4 Giao diện thông tin người dùng

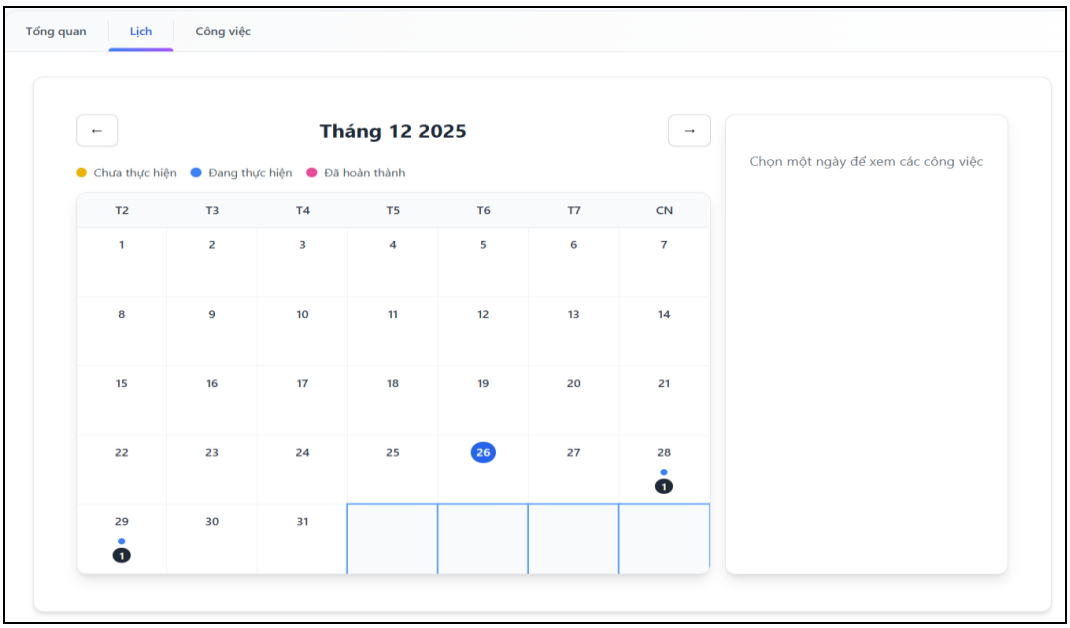
4.2.2 Giao diện trang tổng quan

Trang Dashboard đóng vai trò là trung tâm điều khiển, cung cấp cái nhìn tổng quan về tình hình làm việc của người dùng. Hệ thống hiển thị các số liệu thống kê quan trọng như tổng số dự án tham gia, số lượng công việc đang thực hiện và các công việc sắp đến hạn.



Hình 4.5 Giao diện trang tổng quan

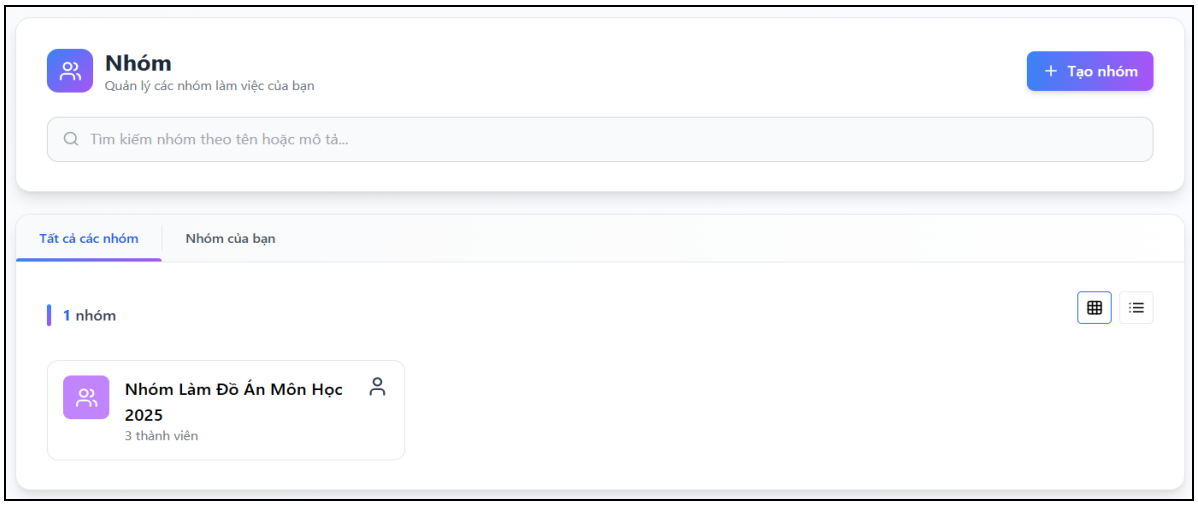
Ngoài ra, giao diện Lịch được tích hợp để trực quan hóa khối lượng công việc theo thời gian. Người dùng có thể chọn từng ngày cụ thể trên lịch để xem danh sách chi tiết các đầu việc cần hoàn thành trong ngày đó.



Hình 4.6 Giao diện lịch

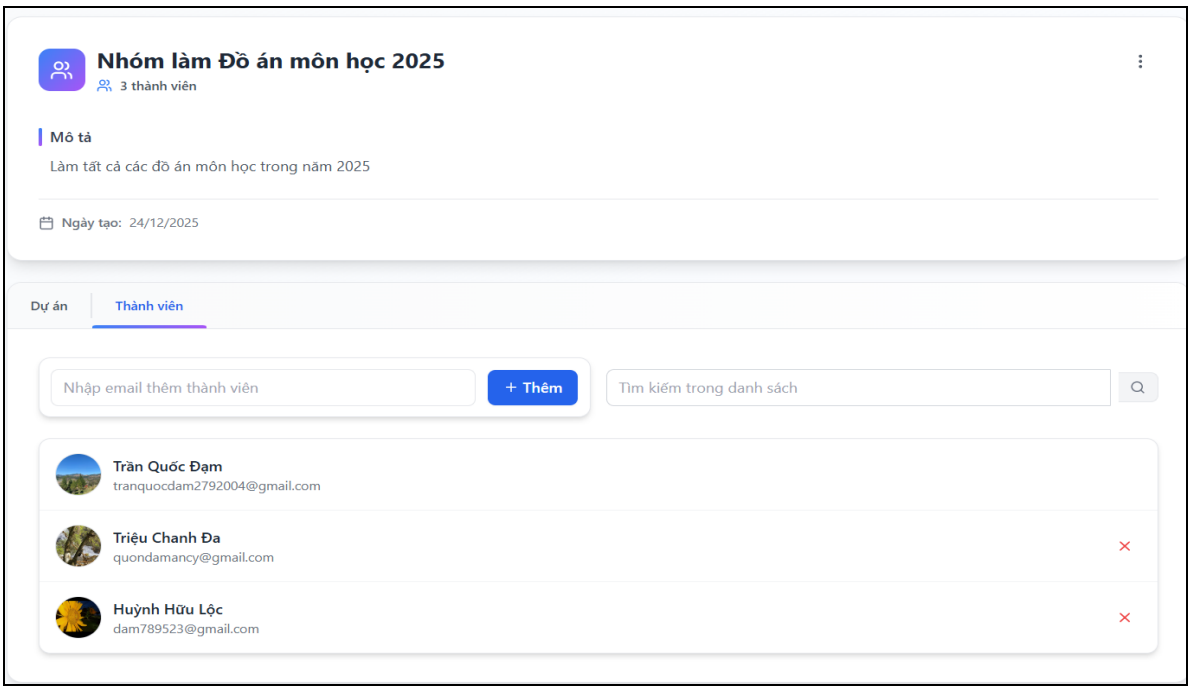
4.2.3 Giao diện chức năng quản lý nhóm và dự án

Hệ thống cho phép người dùng khởi tạo và quản lý nhiều nhóm làm việc khác nhau. Tại giao diện danh sách nhóm, người dùng có thể xem tất cả các nhóm mình đang tham gia hoặc thực hiện tạo nhóm mới để bắt đầu không gian cộng tác.

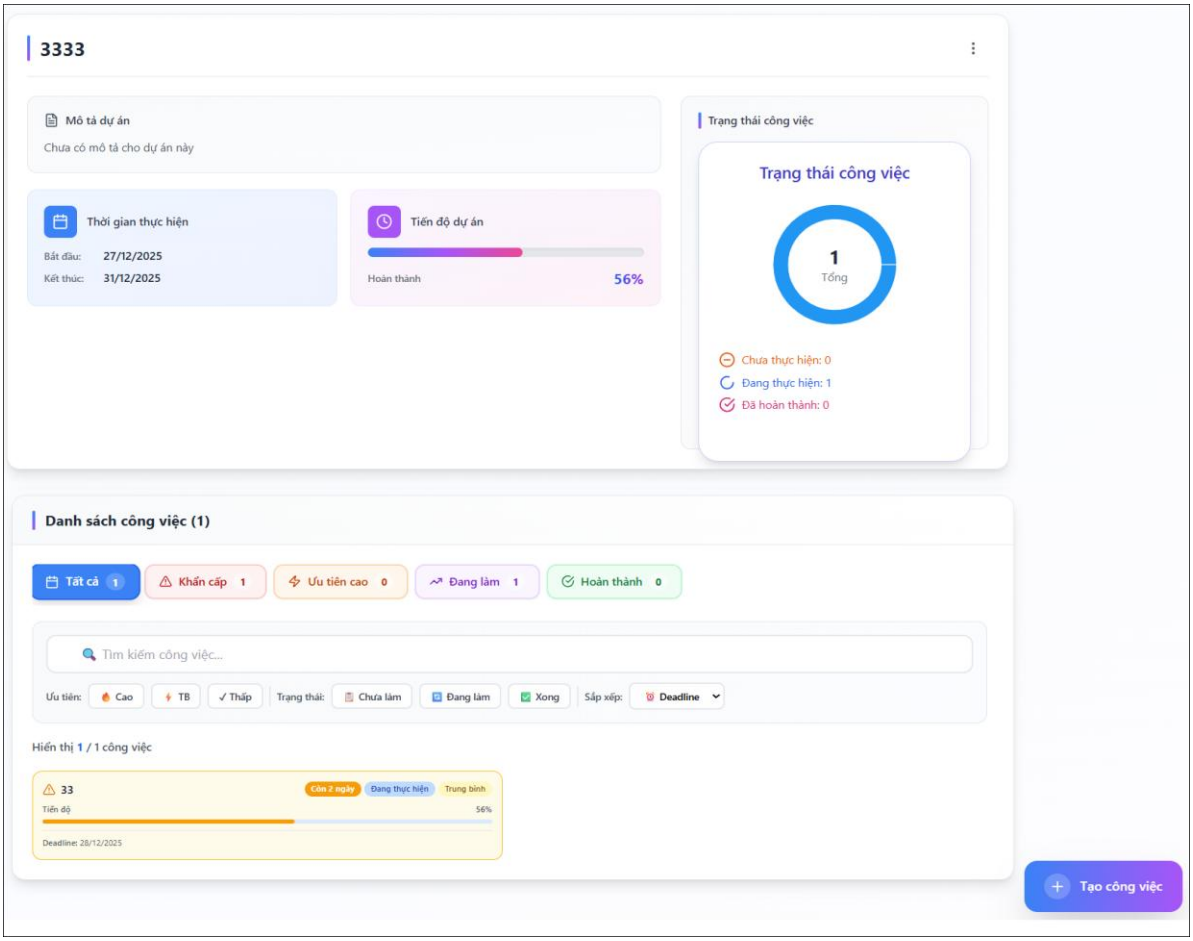


Hình 4.7 Giao diện quản lý danh sách nhóm

Trong mỗi nhóm, giao diện chi tiết dự án cung cấp thông tin toàn diện về tiến độ thực hiện. Biểu đồ trực quan giúp trưởng nhóm và thành viên dễ dàng nắm bắt tỷ lệ hoàn thành công việc. Đồng thời, danh sách thành viên cũng được hiển thị rõ ràng để thuận tiện cho việc điều phối nhân sự, các bộ lọc phù hợp cho công việc.



Hình 4.8 Giao diện chi tiết nhóm

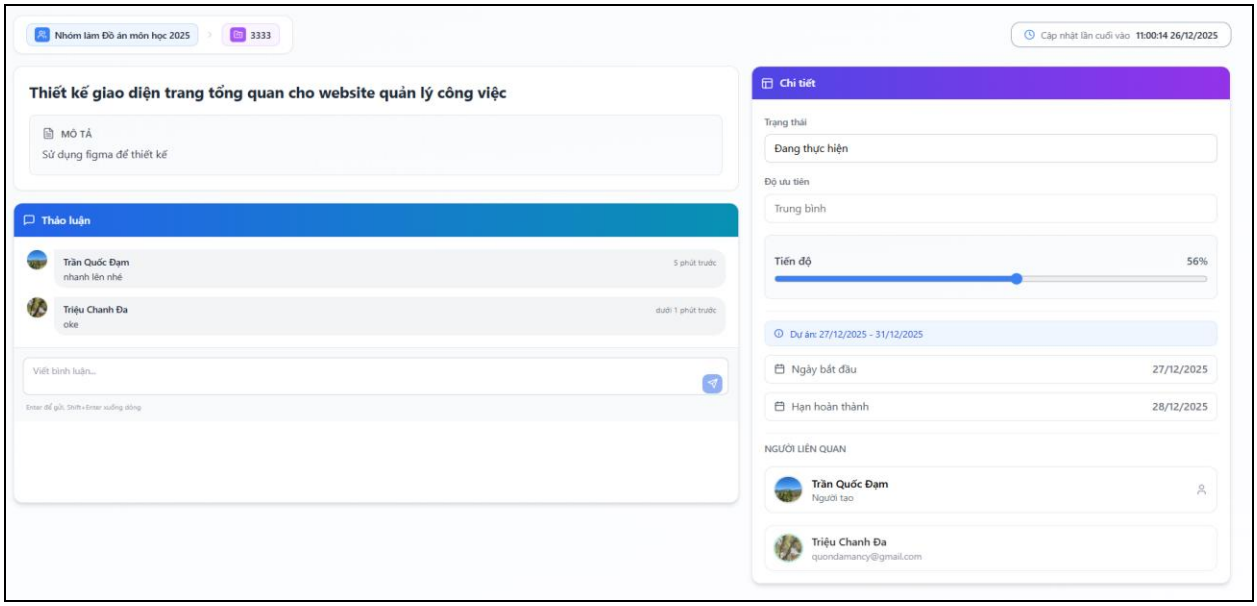


Hình 4.9 Giao diện chi tiết dự án

4.2.4 Giao diện chức năng quản lý công việc và cộng tác

Để tối ưu trải nghiệm người dùng, hệ thống hỗ trợ cập nhật trạng thái công việc thông qua thao tác kéo thả trực quan hoặc thay đổi trực tiếp trong trang chi tiết. Tại giao

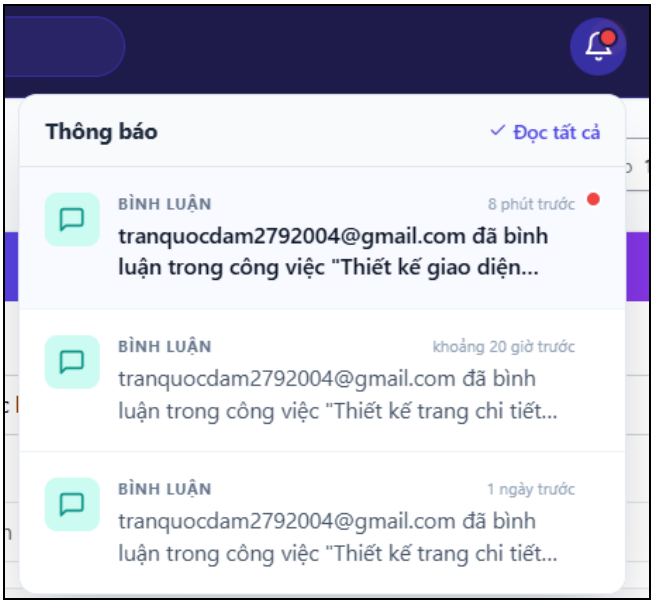
diện chi tiết công việc, người dùng có thể thực hiện gán người chịu trách nhiệm và thảo luận thông qua tính năng bình luận, giúp tăng cường sự tương tác giữa các thành viên.



Hình 4.10 Giao diện chức năng quản lý công việc và bình luận

4.2.5 Hệ thống thông báo

Hệ thống thông báo hoạt động theo thời gian thực, giúp người dùng không bỏ lỡ các thông tin quan trọng. Danh sách thông báo sẽ hiển thị ngay khi có công việc mới được phân công, có bình luận mới hoặc khi trạng thái dự án thay đổi.



Hình 4.11 Giao diện thông báo

4.3 ĐÁNH GIÁ HỆ THỐNG

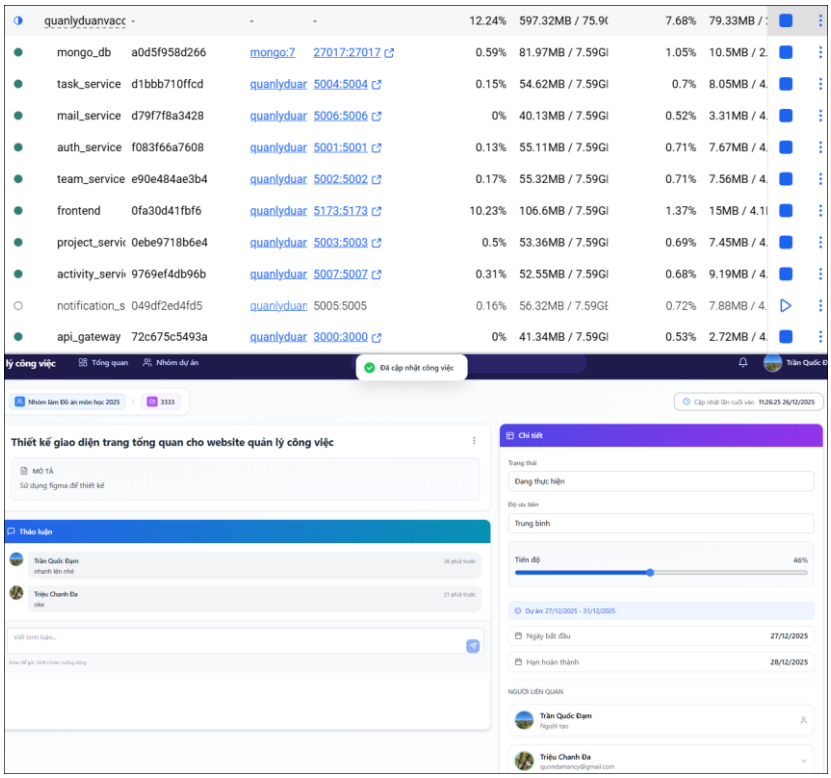
4.3.1 Đánh giá về mặt kỹ thuật

Hệ thống đã được xây dựng thành công dựa trên các tiêu chuẩn công nghệ hiện đại, đáp ứng tốt các yêu cầu về kiến trúc và quy trình triển khai:

Ưu điểm của kiến trúc Microservices:

Khả năng độc lập và chịu lỗi cao: việc tách hệ thống thành các dịch vụ riêng biệt (Auth, Team, Project, Task, Notification) giúp hạn chế rủi ro dây chuyền. Nếu một dịch vụ gặp sự cố (ví dụ: dịch vụ thông báo bị lỗi), các chức năng cốt lõi khác như quản lý dự án hay công việc vẫn hoạt động bình thường, khắc phục được nhược điểm "chết toàn bộ" của kiến trúc Monolithic truyền thống.

Khả năng mở rộng linh hoạt: Mỗi dịch vụ có thể được phát triển và nâng cấp độc lập mà không ảnh hưởng đến toàn bộ mã nguồn chung. Điều này tạo thuận lợi cho việc bảo trì và mở rộng thêm các tính năng mới trong tương lai.



Hình 4.12 Giao diện vẫn hoạt động khi tắt notification service

Hiệu quả của công nghệ Docker:

Đồng bộ môi trường: Docker giúp đóng gói toàn bộ mã nguồn và thư viện phụ thuộc vào các container, đảm bảo ứng dụng chạy nhất quán trên mọi môi trường.

Đơn giản hóa quy trình vận hành: Việc sử dụng docker-compose cho phép khởi chạy đồng thời 10 dịch vụ chỉ với một câu lệnh, giúp giảm thiểu đáng kể thời gian thiết lập và rủi ro xung đột cấu hình so với phương pháp cài đặt thủ công.

4.3.2 Tự đánh giá so với mục tiêu ban đầu

Nhóm chức năng quản lý người dùng: Đã hoàn thiện đăng ký, đăng nhập (bao gồm Google OAuth), bảo mật với JWT và quản lý hồ sơ cá nhân.

Nhóm chức năng quản lý dự án và nhóm: Đã cho phép tạo nhóm, mời thành viên, khởi tạo dự án và thiết lập kế hoạch.

Nhóm chức năng quản lý công việc: Đã đáp ứng tốt việc tạo tác vụ, phân công người thực hiện, cập nhật trạng thái và theo dõi tiến độ.

Nhóm chức năng tương tác: Đã tích hợp thành công tính năng bình luận và hệ thống thông báo thời gian thực, đảm bảo sự phối hợp nhịp nhàng giữa các thành viên.

CHƯƠNG 5 KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

5.1 KẾT LUẬN

Sau quá trình nghiên cứu lý thuyết và hiện thực hóa ứng dụng, đề tài "Phát triển hệ thống quản lý dự án và công việc nhóm trực tuyến" đã hoàn thành các mục tiêu đề ra ban đầu, đạt được những kết quả cụ thể sau:

Về mặt sản phẩm: Đề tài đã xây dựng thành công hệ thống website quản lý dự án và công việc nhóm với giao diện trực quan, thân thiện, thay thế hiệu quả cho các phương thức quản lý thủ công rời rạc hiện nay. Hệ thống đảm bảo đầy đủ các chức năng nghiệp vụ cốt lõi bao gồm: quản lý người dùng và xác thực bảo mật (JWT, Google OAuth), quản lý không gian làm việc, quản lý và phân công công việc, cũng như các tính năng tương tác thời gian thực (bình luận, thông báo).

Về mặt công nghệ và kiến trúc: Đề tài đã ứng dụng thành công kiến trúc vi dịch vụ kết hợp với công nghệ container hóa Docker. Việc tách biệt hệ thống thành các dịch vụ độc lập kết nối qua API Gateway không chỉ giúp hệ thống vận hành ổn định, giảm thiểu rủi ro lỗi dây chuyền mà còn chứng minh được tính khả thi khi triển khai trên môi trường máy tính cá nhân với tài nguyên hạn chế.

Về mặt thực tiễn: Hệ thống đã giải quyết được bài toán về sự thiếu hụt công cụ quản lý chuyên biệt nhưng đơn giản dành cho sinh viên và các nhóm nghiên cứu nhỏ. Sản phẩm giúp tăng cường tính minh bạch trong phân công nhiệm vụ, hỗ trợ theo dõi tiến độ trực quan và nâng cao hiệu quả cộng tác nhóm trong môi trường giáo dục.

5.2 HƯỚNG PHÁT TRIỂN

Mặc dù hệ thống đã đáp ứng được các yêu cầu cơ bản, để nâng cao tính ứng dụng và khả năng cạnh tranh trong bối cảnh công nghệ thay đổi nhanh chóng, đề tài đề xuất các hướng nghiên cứu và phát triển tiếp theo như sau:

Tích hợp trí tuệ nhân tạo và kỹ thuật RAG (Retrieval-Augmented Generation):Đây là hướng phát triển trọng tâm nhằm thông minh hóa hệ thống. Nghiên cứu sẽ tập trung vào việc tích hợp các mô hình ngôn ngữ lớn kết hợp với kỹ thuật RAG để truy xuất dữ liệu nội bộ của dự án. Tính năng này sẽ cho phép xây dựng trợ lý ảo thông minh, hỗ trợ người dùng:

– Tra cứu nhanh thông tin quy định, tài liệu đặc tả hoặc lịch sử thay đổi của dự án bằng ngôn ngữ tự nhiên.

– Tự động tóm tắt nội dung thảo luận trong các bình luận và đề xuất các hành động tiếp theo.

– Hỗ trợ phân tích rủi ro dựa trên dữ liệu lịch sử của các công việc quá hạn.

Nâng cao khả năng phân tích và báo cáo dữ liệu: Phát triển thêm các biểu đồ phân tích chuyên sâu như biểu đồ Burndown, biểu đồ Gantt và các chỉ số hiệu suất cá nhân. Điều này giúp trưởng nhóm và giảng viên hướng dẫn có cái nhìn đa chiều và chính xác hơn về năng suất làm việc của từng thành viên.

Tối ưu hóa quy trình vận hành: Hiện tại hệ thống đang chạy trên môi trường Docker cục bộ. Hướng phát triển tiếp theo là triển khai hệ thống lên các nền tảng điện toán đám mây kết hợp với quy trình CI/CD để tự động hóa việc kiểm thử và triển khai, đảm bảo tính sẵn sàng cao cho hệ thống.

DANH MỤC TÀI LIỆU THAM KHẢO

[1] "Website là gì? Trang web là gì? Cấu tạo website phổ biến". Truy cập: 26 Tháng Chạp 2025. [Online]. Available at: https://fptshop.com.vn/tin-tuc/danh-gia/website-la-gi-169249

[2] Hưng N., "[2025] Mô hình Client Server là gì? Tổng quan về Client Server". Truy cập: 26 Tháng Chạp 2025. [Online]. Available at: https://vietnix.vn/mo-hinh-client-server/

[3] Ngọc T. T. M., "Microservices là gì? Ứng dụng của kiến trúc này như thế nào?", Sunteco | Beyond The Clouds. Truy cập: 26 Tháng Chạp 2025. [Online]. Available at: https://sunteco.vn/microservices-la-gi-ung-dung-cua-kien-truc-nay-nhu-the-nao/

[4] "Node.js — Run JavaScript Everywhere". Truy cập: 27 Tháng Chạp 2025. [Online]. Available at: https://nodejs.org/en

[5] "ReactJS là gì: Tính năng nổi bật, cách hoạt động và Lifecycle – ITviec Blog". Truy cập: 26 Tháng Chạp 2025. [Online]. Available at: https://itviec.com/blog/reactjs-la-gi/

[6] V. IDC, "Khám phá chi tiết về cơ sở dữ liệu phi quan hệ", viettelidc.com.vn. Truy cập: 26 Tháng Chạp 2025. [Online]. Available at: https://viettelidc.com.vn/tin-tuc/kham-pha-chi-tiet-ve-co-so-du-lieu-phi-quan-he

[7] Tiến C. L. V., "JWT là gì? Tổng quan kiến thức về JSON Web Token 2025". Truy cập: 27 Tháng Chạp 2025. [Online]. Available at: https://vietnix.vn/jwt-la-gi/

[8] Bộ L., "Giáo trình phân tích thiết kế hệ thống".