**LỜI MỞ ĐẦU**

Trong thời đại công nghệ số phát triển mạnh mẽ như hiện nay, nhu cầu tìm kiếm thông tin nhanh chóng và tiện lợi trên các nền tảng trực tuyến ngày càng trở nên phổ biến. Đặc biệt, đối với sinh viên và người lao động xa nhà, việc tìm kiếm phòng trọ phù hợp là một nhu cầu thiết yếu nhưng cũng đầy khó khăn và mất nhiều thời gian. Chính vì vậy, việc xây dựng một trang web hỗ trợ tìm kiếm phòng trọ một cách thuận tiện, trực quan và hiệu quả là hết sức cần thiết.

Xuất phát từ thực tế đó, nhóm chúng tôi lựa chọn đề tài **“****Xây dựng trang web tìm kiếm phòng trọ áp dụng kiến trúc Client/Server và giao tiếp Frontend – Backend thông qua API”** với mục tiêu hướng đến một giải pháp công nghệ hiện đại, hiệu quả và có tính ứng dụng cao. Việc xây dựng hệ thống theo mô hình kiến trúc Client/Server giúp phân tách rõ ràng giữa phần giao diện người dùng (Client) và phần xử lý dữ liệu (Server), từ đó nâng cao khả năng quản lý, bảo trì và mở rộng hệ thống. Đặc biệt, việc sử dụng giao tiếp thông qua API giữa Frontend và Backend không chỉ đảm bảo tính linh hoạt trong phát triển, mà còn giúp dễ dàng tích hợp với các nền tảng khác như ứng dụng di động hoặc dịch vụ bên thứ ba trong tương lai.

Hệ thống được thiết kế nhằm hỗ trợ người dùng tra cứu thông tin phòng trọ theo các tiêu chí cụ thể như vị trí, giá cả, tiện nghi,… Đồng thời, chủ trọ cũng có thể dễ dàng đăng tải, quản lý thông tin về phòng trọ của mình. Bên cạnh việc áp dụng các nguyên lý về kiến trúc phần mềm hiện đại, nhóm còn chú trọng đến yếu tố trải nghiệm người dùng (UX/UI), bảo mật thông tin, và tính khả dụng của hệ thống.

Thông qua việc thực hiện đề tài này, nhóm mong muốn ứng dụng những kiến thức đã học về phát triển web, lập trình phía client và server, thiết kế API, quản lý cơ sở dữ liệu, cũng như các phương pháp triển khai và kiểm thử phần mềm vào một dự án thực tiễn. Đồng thời, đây cũng là cơ hội để nhóm rèn luyện kỹ năng làm việc nhóm, giải quyết vấn đề và triển khai quy trình phát triển phần mềm một cách bài bản.

**LỜI CẢM ƠN**

Đầu tiên, chúng em xin gửi lời cảm ơn đặt biệt đến giảng viên hướng dẫn của chúng em là thầy Nguyễn Bảo Ân, thầy đã luôn giảng dạy, cung cấp các kiến thức chuyên môn và chia sẻ kinh nghiệm của thầy để giúp chúng em thực hiện đề tài. Với sự chỉ dạy tận tình và kiến thức sâu rộng của thầy đã hỗ trợ rất nhiều trong việc làm báo cáo môn học của chúng em. Những buổi thảo luận và góp ý của thầy đã giúp chúng em nhìn nhận vấn đề một cách rõ ràng hơn, giải quyết những sai sót kịp thời giúp bài báo cáo được hoàn chỉnh hơn.

Chúng em cũng xin gửi lời cảm ơn đến Khoa Công nghệ Thông tin thuộc Trường Kỹ thuật và Công nghệ của Trường Đại học Trà Vinh đã cung cấp điều kiện và tài liệu giúp chúng em thu thập thông tin và tài liệu một cách thuận lợi.

Đồng thời, chúng em xin cảm ơn các bạn bè cũng như gia đình đã luôn đồng hành chia sẽ, hỗ trợ, tạo điều kiện trong suốt quá trình thực hiện đề tài báo cáo môn học. Mỗi sự hỗ trợ, dù là nhỏ nhất, đều góp phần không nhỏ giúp chúng em hoàn thành báo cáo môn học một cách hiệu quả và đạt được những mục tiêu đề ra.

Chúng em xin chân thành cảm ơn !

**MỤC LỤC**

[CHƯƠNG 1. MỞ ĐẦU 7](#_Toc204332238)

[**1.1** **Lý do chọn đề tài** 7](#_Toc204332239)

[**1.2** **Giới thiệu đề tài** 7](#_Toc204332240)

[**1.3** **Mục tiêu của ứng dụng** 8](#_Toc204332241)

[CHƯƠNG 2. CƠ SỞ LÝ THUYẾT 9](#_Toc204332242)

[**2.1** **Jira – Quản lý dự án và lập kế hoạch Sprint** 9](#_Toc204332243)

[**2.2** **Figma – Thiết kế giao diện người dùng (UI/UX)** 10](#_Toc204332244)

[***2.2.1*** ***Giới thiệu về Figma*** 10](#_Toc204332245)

[***2.2.2*** ***Các tính năng của Figma*** 10](#_Toc204332246)

[**2.3** **GitHub – Quản lý mã nguồn và kiểm soát phiên bản** 11](#_Toc204332247)

[***2.3.1*** ***Giới thiệu về GitHub*** 11](#_Toc204332248)

[***2.3.2*** ***Các chức năng chính của GitHub*** 11](#_Toc204332249)

[**2.4** **GitHub Actions – Tự động hóa kiểm thử và triển khai (CI/CD)** 12](#_Toc204332250)

[***2.4.1*** ***Giới thiệu về GitHub Actions*** 12](#_Toc204332251)

[***2.4.2*** ***Các thành phần chính của GitHub Actions*** 12](#_Toc204332252)

[**2.5** **Postman – Kiểm thử API** 13](#_Toc204332253)

[***2.5.1*** ***Giới thiệu về Postman*** 13](#_Toc204332254)

[***2.5.2*** ***Các thành phần chính của Postman*** 13](#_Toc204332255)

[**2.6** **Docker – Đóng gói và triển khai ứng dụng** 16](#_Toc204332256)

[***2.6.1*** ***Giới thiệu về Docker*** 16](#_Toc204332257)

[***2.6.2*** ***Các khái niệm chính trong Docker*** 16](#_Toc204332258)

[***2.6.3*** ***Vai trò của Docker*** 17](#_Toc204332259)

[**2.7** **RESTful API – Thiết kế API hiệu quả** 18](#_Toc204332260)

[***2.7.1*** ***Giới thiệu về RESTful API*** 18](#_Toc204332261)

[***2.7.2*** ***Đặc điểm chính của RESTful API*** 18](#_Toc204332262)

[**2.8** **Node.js – Nền tảng chạy JavaScript phía máy chủ (Backend)** 19](#_Toc204332263)

[***2.8.1*** ***Khái niệm*** 19](#_Toc204332264)

[***2.8.2*** ***Đặc điểm*** 19](#_Toc204332265)

[**2.9** **React.js – Framework phát triển ứng dụng web React hiện đại (Frontend)** 19](#_Toc204332266)

[***2.9.1*** ***Khái niệm*** 19](#_Toc204332267)

[***2.9.2*** ***Tính năng nổi bật*** 19](#_Toc204332268)

[**2.10** **MongoDB – Hệ quản trị cơ sở dữ liệu NoSQL** 20](#_Toc204332269)

[***2.10.1*** ***Khái niệm*** 20](#_Toc204332270)

[***2.10.2*** ***Đặc điểm nổi bật của MongoDB*** 20](#_Toc204332271)

[***2.10.3*** ***Các khái niệm cơ bản trong MongoDB*** 21](#_Toc204332272)

[CHƯƠNG 3. PHÂN TÍCH YÊU CẦU 23](#_Toc204332273)

[**3.1** **Các chức năng chính của hệ thống (Functional Requirements).** 23](#_Toc204332274)

[***3.1.1*** ***Quản lý người dùng*** 23](#_Toc204332275)

[***3.1.2*** ***Quản lý phòng trọ (Room Management)*** 23](#_Toc204332276)

[***3.1.3*** ***Quản lý bài đăng cho thuê (Dành cho chủ trọ)*** 24](#_Toc204332277)

[***3.1.4*** ***Tương tác người dùng*** 24](#_Toc204332278)

[***3.1.5*** ***Quản lý hệ thống (Admin)*** 24](#_Toc204332279)

[***3.1.6*** ***Bảo mật và xác thực*** 25](#_Toc204332280)

[***3.1.7*** ***RESTful API (Backend giao tiếp với Frontend)*** 25](#_Toc204332281)

[**3.2** **Các yêu cầu phi chức năng (Non-functional Requirements).** 25](#_Toc204332282)

[CHƯƠNG 4. THIẾT KẾ HỆ THỐNG 28](#_Toc204332283)

[**4.1** **Kiến trúc tổng thể: Sơ đồ kiến trúc hệ thống (client/server, API, cơ sở dữ liệu).** 28](#_Toc204332284)

[***4.1.1*** ***Cấu trúc thư mục hệ thống*** 28](#_Toc204332285)

[***4.1.2*** ***Sơ đồ Usse case cho hệ thống*** 31](#_Toc204332286)

[***4.1.3*** ***Sơ đồ luồng xử lý dữ liệu của hệ thống web sử dụng mô hình microservices.*** 32](#_Toc204332287)

[**4.2** **Thiết kế cơ sở dữ liệu:** 34](#_Toc204332288)

[***4.2.1*** ***Loại cơ sở dữ liệu sử dụng*** 34](#_Toc204332289)

[***4.2.2*** ***Mô hình ERD*** 36](#_Toc204332290)

[***4.2.3*** ***Mô hình vật lý.*** 39](#_Toc204332291)

[**4.3** **Thiết kế giao diện (UI/UX): Ảnh chụp các màn hình chính và liên kết với bản thiết kế Figma.** 42](#_Toc204332292)

[CHƯƠNG 5. TRIỂN KHAI VÀ CÔNG NGHỆ SỬ DỤNG 46](#_Toc204332293)

[**5.1** **Danh sách các công nghệ đã sử dụng (ngôn ngữ lập trình, framework, thư viện).** 46](#_Toc204332294)

[**5.2** **Quy trình CI/CD với GitHub Actions.** 46](#_Toc204332295)

[***5.2.1*** ***Giới thiệu về CI/CD*** 46](#_Toc204332296)

[***5.2.2*** ***GitHub Actions – Công cụ CI/CD tích hợp*** 47](#_Toc204332297)

[***5.2.3*** ***Quy trình CI/CD trong hệ thống tìm kiếm phòng trọ*** 47](#_Toc204332298)

[**5.3** **Cấu hình Docker và quy trình triển khai ứng dụng.** 48](#_Toc204332299)

[***5.3.1*** ***Cấu hình Docker*** 48](#_Toc204332300)

[***5.3.2*** ***Quy trình triển khai ứng dụng*** 49](#_Toc204332301)

[***5.3.3*** ***Quy trình sử dụng Jira trong dự án*** 50](#_Toc204332302)

[***5.3.4*** ***Lợi ích đạt được khi sử dụng Jira*** 51](#_Toc204332303)

[**5.4** **Phân công nhiệm vụ của từng thành viên trong nhóm.** 52](#_Toc204332304)

[CHƯƠNG 6. KIỂM THỬ 53](#_Toc204332305)

[**6.1** **Chiến lược kiểm thử và công cụ sử dụng (Postman).** 53](#_Toc204332306)

[***6.1.1*** ***Mục tiêu kiểm thử*** 53](#_Toc204332307)

[***6.1.2*** ***Các chức năng kiểm thử chính*** 53](#_Toc204332308)

[***6.1.3*** ***Kỹ thuật kiểm thử áp dụng*** 54](#_Toc204332309)

[***6.1.4*** ***Công cụ sử dụng: Postman*** 55](#_Toc204332310)

[**6.2** **Kết quả kiểm thử API (có thể đính kèm ảnh chụp màn hình từ Postman).** 55](#_Toc204332311)

[CHƯƠNG 7. ĐÁNH GIÁ VÀ KẾT LUẬN 57](#_Toc204332312)

[**7.1** **Những khó khăn gặp phải trong quá trình thực hiện.** 57](#_Toc204332313)

[**7.2** **Bài học rút ra và đề xuất cải thiện trong tương lai.** 57](#_Toc204332314)

[CHƯƠNG 8. PHỤ LỤC 59](#_Toc204332315)

[**8.1** **Hướng dẫn cài đặt và chạy ứng dụng.** 59](#_Toc204332316)

[***8.1.1*** ***Yêu cầu hệ thống*** 59](#_Toc204332317)

[***8.1.2*** ***Các bước cài đặt và chạy hệ thống*** 59](#_Toc204332318)

[***8.1.3*** ***Kiểm thử API (tùy chọn)*** 60](#_Toc204332319)

[***8.1.4*** ***Cách dừng và khởi động lại*** 60](#_Toc204332320)

[***8.1.5*** ***Một số lỗi thường gặp*** 60](#_Toc204332321)

[**8.2** **Liên kết GitHub repository và link demo (nếu có).** 60](#_Toc204332322)

**DANH MỤC HÌNH ẢNH, BẢNG BIỂU**

[Bảng 2. Các khái niệm cơ bản trong MongoDB 21](#_Toc204328774)

[Bảng 3. Các tác nhân của hệ thống 31](#_Toc204328775)

[Bảng 4. Các mối quan hệ của các thực thể trong mô hình ERD 38](#_Toc204328776)

[Bảng 5. Dữ liệu của user 39](#_Toc204328777)

[Bảng 6. Dữ liệu của Post 40](#_Toc204328778)

[Bảng 7. Dữ liệu của Notifition 40](#_Toc204328779)

[Bảng 8. Dữ liệu của Image 40](#_Toc204328780)

[Bảng 9. Các khóa ngoại và quan hệ 41](#_Toc204328781)

[Bảng 10. Ngôn ngữ lập trình 46](#_Toc204328782)

[Bảng 11. Các thư viện 46](#_Toc204328783)

[Bảng 12. Mô tả quy trình CI/CD 47](#_Toc204328784)

[Bảng 13. Thành phần và đường dẫn truy cập trong quy trình triển khai ứng dụng 50](#_Toc204328785)

[Bảng 14. Lợi ích của Jira 52](#_Toc204328786)

[Bảng 15. Các công cụ cần có để chạy ứng dụng 59](#_Toc204328787)

[Bảng 16. Kiểm thử API 60](#_Toc204328788)

[Bảng 17. Một số lỗi thường gặp khi chạy ứng dụng 60](#_Toc204328789)

[Hình 2.1. Thành phần Collections trong Postman 14](#_Toc204337437)

[Hình 2.2. Chức năng trong API content 15](#_Toc204337438)

[Hình 2.3. Docker Architecture 16](#_Toc204337439)

[Hình 4.1. Sơ đồ Usse case 31](#_Toc204337440)

[Hình 4.2. Sơ đồ luồng xử lý dữ liệu 33](#_Toc204337441)

[Hình 4.3. Mô hình ERD 36](#_Toc204337442)

[Hình 4.4. Mô hình vật lý 39](#_Toc204337443)

[Hình 4.5 Thiết kế figma 42](#_Toc204337444)

[Hình 4.6 Giao diện trang đăng nhập 42](#_Toc204337445)

[Hình 4.7 Giao diện trang đăng kí 43](#_Toc204337446)

[Hình 4.8 Giao diện trang tìm kiếm 43](#_Toc204337447)

[Hình 4.9 Giao diện trang cá nhân 44](#_Toc204337448)

[Hình 4.10 Giao diện tạo bài đăng 44](#_Toc204337449)

[Hình 4.11 Giao diện hồ sơ cá nhân 45](#_Toc204337450)

[Hình 4.12 Giao diện trang admin duyệt bài đăng 45](#_Toc204337451)

[Hình 5.1. CI/CD sau khi hoàn thành dự án 47](#_Toc204337452)

[Hình 6.1. Postman -- đăng kí tài khoản 55](#_Toc204337453)

[Hình 6.2. Đăng nhập trả về token 56](#_Toc204337454)

[Hình 6.3. Dùng token để trả về thông tin tài khoản 56](#_Toc204337455)

# MỞ ĐẦU

## Lý do chọn đề tài

Hiện nay, nhu cầu thuê nhà trọ tại các khu vực thành thị, đặc biệt là tại các thành phố lớn, ngày càng gia tăng mạnh mẽ, chủ yếu đến từ sinh viên, người lao động và các hộ gia đình có thu nhập trung bình. Tuy nhiên, việc tìm kiếm phòng trọ phù hợp thường gặp nhiều khó khăn do thông tin phân tán, thiếu minh bạch, cập nhật chậm và không có hệ thống quản lý tập trung. Nhiều người vẫn phải tìm phòng thông qua các hình thức truyền thống như đi khảo sát trực tiếp, hỏi bạn bè hoặc xem các thông tin rải rác trên mạng xã hội, vốn tiềm ẩn nhiều rủi ro và tốn kém thời gian.

Trước thực trạng đó, việc xây dựng một **trang web tìm kiếm phòng trọ chuyên nghiệp, đáng tin cậy và dễ sử dụng** là cần thiết nhằm hỗ trợ người thuê nhà tiếp cận thông tin nhanh chóng, chính xác; đồng thời giúp chủ trọ dễ dàng đăng tải và quản lý thông tin phòng trọ của mình. Để đảm bảo hệ thống có khả năng hoạt động ổn định, dễ dàng nâng cấp và bảo trì, nhóm quyết định lựa chọn kiến trúc **Client/Server** làm nền tảng phát triển, trong đó phần giao diện người dùng (Frontend) và phần xử lý nghiệp vụ (Backend) được tách biệt rõ ràng và giao tiếp với nhau thông qua **giao diện lập trình ứng dụng (API)**.

Bên cạnh yếu tố thực tiễn và nhu cầu xã hội, đề tài này còn phù hợp với định hướng học tập của nhóm, đặc biệt trong việc vận dụng các kiến thức về lập trình web, quản lý cơ sở dữ liệu, thiết kế API, mô hình kiến trúc phần mềm hiện đại cũng như các kỹ năng triển khai và kiểm thử hệ thống. Qua đó, đề tài không chỉ giúp nhóm củng cố và mở rộng kiến thức đã học, mà còn là cơ hội để tiếp cận quy trình phát triển phần mềm chuyên nghiệp, từ việc phân tích nhu cầu đến triển khai sản phẩm hoàn chỉnh.

## Giới thiệu đề tài

Đề tài “Xây dựng trang web tìm kiếm phòng trọ áp dụng kiến trúc Client/Server và giao tiếp Frontend – Backend thông qua API” được triển khai nhằm giải quyết bài toán trên bằng cách xây dựng một hệ thống web có khả năng hỗ trợ người dùng tìm kiếm thông tin phòng trọ theo các tiêu chí cụ thể như vị trí, giá thuê, loại phòng, tiện nghi,... Bên cạnh đó, hệ thống còn cho phép chủ trọ dễ dàng đăng tin, chỉnh sửa và quản lý thông tin phòng trọ của mình.

Trang web được phát triển theo mô hình kiến trúc Client/Server, trong đó phần Frontend (giao diện người dùng) và Backend (xử lý dữ liệu và nghiệp vụ) được tách biệt rõ ràng. Việc giao tiếp giữa hai phần này được thực hiện thông qua **API (Application Programming Interface)**, giúp hệ thống có khả năng mở rộng, bảo trì linh hoạt, dễ dàng tích hợp với các nền tảng khác như ứng dụng di động hoặc dịch vụ bản đồ. Đồng thời, cách tiếp cận này còn giúp nâng cao hiệu suất và khả năng tái sử dụng của mã nguồn.

Việc thực hiện đề tài không chỉ mang lại ý nghĩa thực tiễn cao, đáp ứng nhu cầu xã hội trong việc tối ưu hóa quy trình tìm và cho thuê phòng trọ, mà còn giúp nhóm thực hành, củng cố và phát triển các kỹ năng quan trọng trong quá trình học tập như: thiết kế giao diện người dùng, xây dựng API RESTful, quản lý cơ sở dữ liệu, và triển khai hệ thống web hoàn chỉnh theo quy trình phần mềm hiện đại.

## Mục tiêu của ứng dụng

Xây dựng một trang web hỗ trợ tìm kiếm và đăng bài cho thuê phòng trọ, dễ sử dụng, kết nối hiệu quả giữa người có nhu cầu thuê trọ và người cho thuê. Hệ thống được phát triển dựa trên kiến trúc Client/Server hiện đại, đảm bảo tính tách biệt giữa phần giao diện (Frontend) và phần xử lý nghiệp vụ (Backend), giúp nâng cao khả năng mở rộng, bảo trì và tích hợp với các hệ thống khác trong tương lai.

* Đối với người dùng tìm phòng trọ:

Cung cấp giao diện thân thiện, tìm kiếm thông tin phòng trọ theo nhiều tiêu chí: từ khóa, vị trí, giá thuê,...

Hiển thị thông tin phòng trọ chi tiết bao gồm: tiêu đề, mô tả, giá, địa chỉ, hình ảnh, số điện thoại liên hệ,...

* Đối với người cho thuê phòng trọ (chủ trọ):

Cung cấp chức năng đăng ký và đăng nhập để quản lý thông tin phòng trọ.

Cho phép tạo, xóa bài đăng phòng trọ.

Hiển thị các bài đăng đã tạo, tình trạng đăng bài (đã duyệt, chờ duyệt).

# CƠ SỞ LÝ THUYẾT

## Jira – Quản lý dự án và lập kế hoạch Sprint

**Jira** là một công cụ quản lý dự án nổi bật do Atlassian phát triển, được sử dụng phổ biến trong các nhóm phát triển phần mềm theo phương pháp Agile, đặc biệt là Scrum và Kanban. Jira không chỉ hỗ trợ theo dõi tiến độ công việc mà còn cho phép quản lý toàn bộ vòng đời phát triển phần mềm (Software Development Life Cycle – SDLC) một cách trực quan và có hệ thống.

Chức năng chính của Jira:

*Tạo và quản lý dự án*: Jira cho phép khởi tạo nhiều loại dự án (project) phù hợp với các quy trình khác nhau như Scrum, Kanban, hoặc dạng tùy chỉnh. Mỗi dự án có thể được cấu hình riêng về quy trình làm việc (workflow), phân quyền người dùng, và các loại công việc (issue type).

*Tạo nhiệm vụ (Issue)*:

* Trong Jira, mỗi đơn vị công việc được gọi là một "issue". Issue có thể là các loại khác nhau như:
* Story: mô tả một chức năng cụ thể mà người dùng cần.
* Task: nhiệm vụ cần thực hiện.
* Bug: lỗi cần được xử lý.
* Epic: nhóm các user stories lớn hơn, thường kéo dài nhiều sprint.
* Mỗi issue có thể chứa thông tin chi tiết như người phụ trách (assignee), thời gian dự kiến, mức độ ưu tiên, mô tả, trạng thái thực hiện,...

***Lập kế hoạch Sprint*:** Với mô hình Scrum, Jira cho phép chia dự án thành các **Sprint** – chu kỳ làm việc ngắn hạn (thường 1-4 tuần). Nhóm phát triển sẽ chọn các nhiệm vụ từ Product Backlog để đưa vào Sprint, đồng thời xác định mục tiêu cụ thể cần đạt được trong Sprint đó. Jira hỗ trợ kéo – thả các task giữa các Sprint, theo dõi khối lượng công việc (story point) và tính toán tự động tốc độ hoàn thành (velocity).

***Bảng Scrum (Scrum board)*:** Cung cấp giao diện dạng bảng (board) để hiển thị các nhiệm vụ được phân chia theo các trạng thái như “To Do”, “In Progress”, “Done”. Từ đó giúp nhóm phát triển dễ dàng theo dõi tiến độ và cập nhật trạng thái công việc theo thời gian thực.

## Figma – Thiết kế giao diện người dùng (UI/UX)

### *Giới thiệu về Figma*

Với thành công của phiên bản dùng thử vào cuối năm 2015, **Figma** đã chính thức ra mắt vào tháng 09 năm 2016 và tạo được vị trí riêng của mình trong cộng đồng thiết kế. Đến tháng 10 năm 2019, **Figma Community** ra đời. Từ đây, các nhà thiết kế có thể xuất bản tác phẩm của mình để người khác xem và phối lại.

Tháng 04 năm 2021, **Figma** phát triển **FigJam**, một công cụ cho phép người dùng sử dụng các công cụ ghi chú, vẽ hay thêm các biểu tượng cảm xúc vào các dự án.

**Figma** là một công cụ thiết kế giao diện người dùng dựa trên nền tảng web (cloud-based), cho phép làm việc nhóm theo thời gian thực (real-time collaboration). Không giống như các phần mềm thiết kế truyền thống như Adobe XD hay Sketch, Figma không yêu cầu cài đặt và có thể hoạt động trực tiếp trên trình duyệt, đồng thời hỗ trợ cả bản cài đặt desktop.

Figma phù hợp cho cả thiết kế giao diện, tạo nguyên mẫu tương tác (prototype), vẽ sơ đồ hệ thống và chia sẻ tài liệu thiết kế với nhóm phát triển.

### *Các tính năng của Figma*

*Thiết kế giao diện người dùng (UI Design)*: Figma cung cấp các công cụ thiết kế như frame, auto-layout, component, color style, font style,... giúp người thiết kế dễ dàng tạo ra các màn hình giao diện đẹp mắt, đồng nhất và dễ bảo trì.

*Thiết kế trải nghiệm người dùng (UX Design)*: Figma cho phép mô phỏng luồng di chuyển của người dùng thông qua chức năng Prototype, giúp kiểm tra tính hợp lý của điều hướng (navigation), bố cục nội dung và logic của các thao tác tương tác trước khi bắt đầu lập trình.

*Hỗ trợ làm việc nhóm*: Cho phép nhiều người cùng truy cập, chỉnh sửa và bình luận trực tiếp trên bản thiết kế theo thời gian thực. Điều này giúp nhóm thiết kế – lập trình – kiểm thử phối hợp hiệu quả, giảm sai sót và tiết kiệm thời gian.

*Tạo hệ thống thiết kế (Design System)*: Các thành phần giao diện có thể được tạo thành Component, giúp tái sử dụng, thống nhất phong cách và dễ cập nhật trong toàn bộ dự án. Việc này đặc biệt hữu ích khi thiết kế ứng dụng lớn hoặc làm việc theo nhóm.

*Xuất mã CSS/XAML*: Figma hỗ trợ xem trước mã CSS của các phần tử trong giao diện, hỗ trợ quá trình hiện thực giao diện trong mã nguồn thực tế, giúp giảm sự không đồng nhất giữa thiết kế và lập trình.

## GitHub – Quản lý mã nguồn và kiểm soát phiên bản

### *Giới thiệu về GitHub*

**Git** là một hệ thống quản lý phiên bản phân tán (Distributed Version Control System – DVCS), cho phép nhiều lập trình viên cùng làm việc trên cùng một mã nguồn mà không gây xung đột, đồng thời ghi nhận đầy đủ lịch sử thay đổi theo thời gian.

**GitHub** là một dịch vụ lưu trữ và chia sẻ mã nguồn trực tuyến, được xây dựng trên nền tảng Git. GitHub không chỉ cho phép lưu trữ và theo dõi mã nguồn, mà còn cung cấp giao diện web thân thiện, hỗ trợ làm việc nhóm, báo lỗi (issue tracking), phân nhánh (branching), hợp nhất mã (pull request), và tích hợp với các công cụ CI/CD, kiểm thử tự động,…

Sự phát triển của nền tảng GitHub bắt đầu vào ngày 19 tháng 10 năm 2007. Trang web được đưa ra vào tháng 4 năm 2008 do Tom Preston-Werner, Chris Wanstrath, và PJ Hyett thực hiện sau khi nó đã được hoàn thành một vài tháng trước đó, xem như giai đoạn beta.

GitHub là một dịch vụ cung cấp [kho lưu trữ mã nguồn](https://vi.wikipedia.org/w/index.php?title=Kho_l%C6%B0u_tr%E1%BB%AF_m%C3%A3_ngu%E1%BB%93n&action=edit&redlink=1) [Git](https://vi.wikipedia.org/wiki/Git_(ph%E1%BA%A7n_m%E1%BB%81m)) dựa trên nền web cho các dự án phát triển phần mềm. GitHub cung cấp cả phiên bản trả tiền lẫn miễn phí cho các tài khoản. Các dự án [mã nguồn mở](https://vi.wikipedia.org/wiki/Ph%E1%BA%A7n_m%E1%BB%81m_ngu%E1%BB%93n_m%E1%BB%9F) sẽ được cung cấp kho lưu trữ miễn phí. Tính đến tháng 4 năm 2016, GitHub có hơn 14 triệu người sử dụng với hơn 35 triệu kho mã nguồn, làm cho nó trở thành máy chủ chứa mã nguồn lớn trên thế giới.

### *Các chức năng chính của GitHub*

*Lưu trữ mã nguồn trực tuyến*: GitHub cung cấp kho lưu trữ (repository) cho phép lập trình viên tải lên, truy cập và chỉnh sửa mã nguồn ở mọi nơi có kết nối Internet. Mỗi lần thay đổi mã được ghi lại với nội dung cập nhật, thời gian, và người thực hiện.

*Phân nhánh (Branch)*: GitHub hỗ trợ tính năng tạo nhánh làm việc (branch), cho phép lập trình viên phát triển tính năng mới, sửa lỗi hoặc thử nghiệm mà không ảnh hưởng đến nhánh chính (main hoặc master).

*Pull Request (PR)*: Pull Request là quy trình gửi yêu cầu hợp nhất mã từ một nhánh phụ vào nhánh chính. Trong quá trình này, các thành viên khác có thể xem xét (review), bình luận, yêu cầu thay đổi hoặc chấp thuận hợp nhất. Đây là bước kiểm tra chất lượng và đồng thuận trước khi đưa mã vào sản phẩm chính thức.

*Quản lý lịch sử phiên bản*: GitHub lưu lại toàn bộ lịch sử thay đổi mã nguồn, cho phép xem lại phiên bản cũ, so sánh sự khác biệt giữa các lần commit, hoặc khôi phục mã khi gặp sự cố. Điều này giúp đảm bảo an toàn dữ liệu và hỗ trợ xử lý sự cố nhanh chóng.

*Hỗ trợ làm việc nhóm*: GitHub cho phép phân quyền truy cập, quản lý nhóm người dùng, tạo các nhãn (label), milestone, project board để quản lý công việc, theo dõi tiến độ và phân công nhiệm vụ rõ ràng.

## GitHub Actions – Tự động hóa kiểm thử và triển khai (CI/CD)

### *Giới thiệu về GitHub Actions*

**GitHub Actions** là một nền tảng tích hợp sẵn trong GitHub, cho phép thiết lập và thực thi các quy trình tự động hóa (workflows) khi có sự kiện xảy ra trong repository, chẳng hạn như khi có **commit mới**, **pull request**, hoặc **push lên nhánh chính**. Các quy trình này thường bao gồm kiểm thử mã nguồn, kiểm tra định dạng, build ứng dụng, đóng gói Docker, triển khai lên server, hoặc các hành động tùy chỉnh khác.

GitHub Actions là 1 nền tảng miễn phí do GitHub cung cấp để giúp chúng ta tự động hoá quá trình CI/CD, cho phép người dùng định nghĩa các workflow tự động hoá các hoạt động trong phát triển phần mềm.

Mỗi workflow trong GitHub Actions là một tập hợp các hành động (actions) được định nghĩa trong file YAML. Các actions này có thể là các lệnh cụ thể như: build ứng dụng, test, triển khai ứng dụng.

### *Các thành phần chính của GitHub Actions*

***Workflow*:** Là một quy trình tự động bao gồm một hoặc nhiều tác vụ. Được định nghĩa bằng file YAML đặt trong thư mục .github/workflows/.

***Job*:** Một workflow gồm nhiều job. Mỗi job có thể chạy song song hoặc tuần tự tùy cấu hình.

***Step*:** Mỗi job gồm nhiều bước (step), mỗi bước là một hành động cụ thể như chạy script, kiểm thử, hoặc triển khai.

***Action*:** Là các bước hành động cụ thể, có thể viết riêng hoặc dùng từ thư viện có sẵn của cộng đồng GitHub.

***Trigger (sự kiện kích hoạt)*:** GitHub Actions có thể được kích hoạt khi xảy ra một số sự kiện như push, pull\_request, release, workflow\_dispatch,...

## Postman – Kiểm thử API

### *Giới thiệu về Postman*

**Postman** là một công cụ hỗ trợ kiểm thử API giao tiếp qua giao thức HTTP/HTTPS. Nó cung cấp một giao diện trực quan cho phép lập trình viên và kiểm thử viên gửi các yêu cầu (requests) đến server và xem kết quả phản hồi (responses) trả về. Postman hỗ trợ tất cả các phương thức HTTP phổ biến như GET, POST, PUT, DELETE, giúp mô phỏng và xác thực hành vi của các endpoint trong hệ thống backend.

Postman là một ứng dụng mã nguồn mở dùng để phát triển và kiểm thử các API (Application Programming Interface). Công nghệ cung cấp môi trường cho các nhà phát triển thực hiện hoạt động tạo, chia sẻ, kiểm thử và quản lý các API của họ.

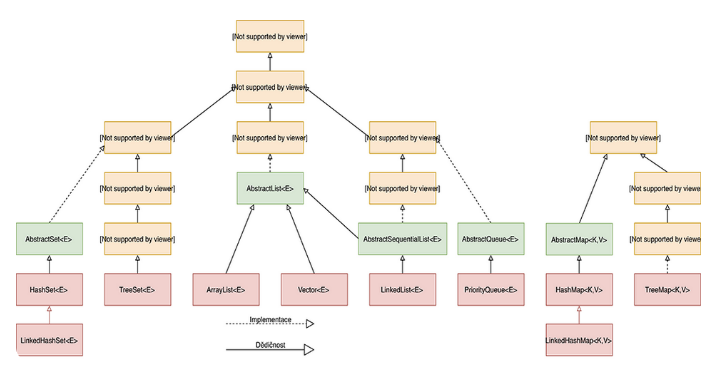
Postman hiện nay có thể được cài đặt dưới dạng ứng dụng desktop hoặc sử dụng trực tiếp trên nền tảng web (Postman Cloud), hỗ trợ đồng bộ dữ liệu qua tài khoản.

### *Các thành phần chính của Postman*

*Settings*: cung cấp các tùy chọn cài đặt chung để tinh chỉnh trải nghiệm sử dụng và hiệu suất của ứng dụng. Một số thông tin cụ thể về danh mục "cài đặt chung" bao gồm:

* Cài đặt tài khoản: Cho phép người dùng quản lý thông tin tài khoản, bao gồm tên người dùng, email và cách thức đăng nhập.
* Giao diện người dùng: Tùy chọn cho phép tùy chỉnh giao diện người dùng, bao gồm cài đặt ngôn ngữ, giao diện sáng/tối và các tùy chọn thể hiện khác.
* Cài đặt bảo mật: Cho phép người dùng quản lý các cài đặt liên quan đến bảo mật và quyền riêng tư, bao gồm quản lý cookie, các thiết lập HTTPS và các cải thiện bảo mật khác.
* Cài đặt proxy: Cho phép cấu hình cài đặt proxy để điều hướng yêu cầu qua máy chủ proxy trong môi trường mạng cụ thể.
* Cài đặt kết nối mạng: Các tùy chọn liên quan đến cài đặt kết nối mạng, bao gồm cài đặt proxy, quản lý kết nối và thông số kỹ thuật liên quan đến mạng.
* Cập nhật ứng dụng: Cho phép cài đặt cách thức và tần suất cập nhật ứng dụng Postman.

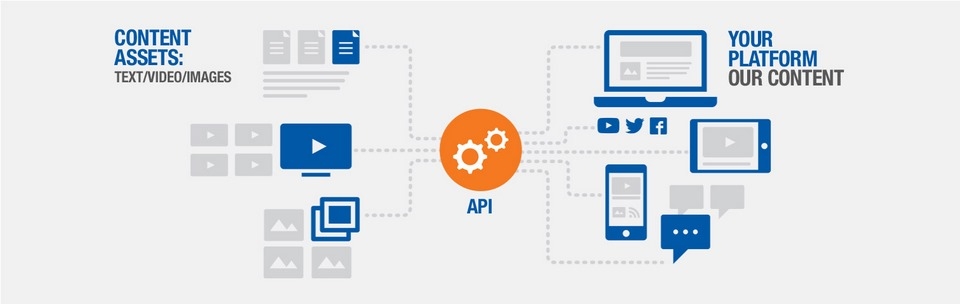
*Collections*: là một phần quan trọng để tổ chức và quản lý các yêu cầu API. Thành phần cung cấp cách thức lưu trữ, sắp xếp và chia sẻ các yêu cầu API vô cùng dễ dàng.



Hình 2.1. Thành phần Collections trong Postman

* Lưu trữ yêu cầu: Collections cho phép bạn tạo và lưu trữ các yêu cầu HTTP, bao gồm các phương thức như GET, POST, PUT, DELETE và các yêu cầu khác.
* Sắp xếp và quản lý: Bằng cách sử dụng collections, bạn có thể tổ chức các yêu cầu vào các nhóm cụ thể, giúp dễ dàng quản lý, tìm kiếm và thực hiện các thao tác nhóm trên chúng.
* Biến môi trường: Collections cũng hỗ trợ việc sử dụng các biến môi trường, cho phép bạn tương tác và chia sẻ thông tin giữa các yêu cầu trong collection một cách hiệu quả.
* Chia sẻ và sử dụng chung: Bạn có thể chia sẻ collection với đồng nghiệp hoặc cộng đồng thông qua các liên kết chia sẻ, hoặc sử dụng các collection có sẵn từ cộng đồng Postman.
* Kiểm thử tự động: Collections cũng hỗ trợ việc tạo và quản lý các bộ kiểm thử tự động, cho phép kiểm tra tính đúng đắn và hiệu suất của API một cách tự động.

*API content*: đề cập đến dữ liệu cụ thể mà bạn gửi hoặc nhận từ các APIs đang tương tác. Người dùng cần lưu ý các chức năng trong thành phần này như sau:



Hình 2.2. Chức năng trong API content

* Headers (Đầu mục): Thông tin tiêu đề gửi đi hoặc nhận về, bao gồm thông tin như kiểu ngôn ngữ (content-type), định dạng (JSON, XML) và các thông tin xác thực.
* Body (Nội dung): Dữ liệu cụ thể được gửi đi hoặc nhận về từ API, có thể chứa thông tin tương tác như thông tin đăng ký, thông tin sản phẩm hoặc bất kỳ dữ liệu nào khác mà API yêu cầu hoặc gửi lại.
* Response (Phản hồi): Thông tin phản hồi từ API sau khi bạn gửi yêu cầu, bao gồm trạng thái HTTP (200, 404, v.v.), dữ liệu phản hồi và thông tin khác như thời gian phản hồi.

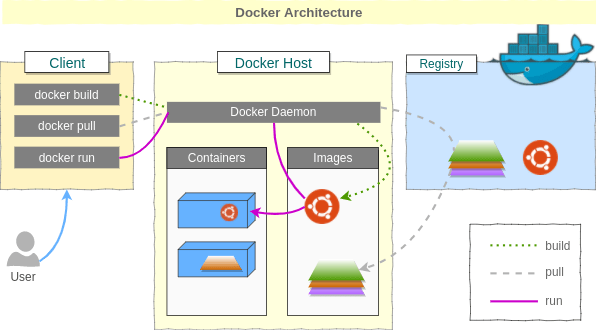
## Docker – Đóng gói và triển khai ứng dụng

### *Giới thiệu về Docker*

**Docker** là một nền tảng mã nguồn mở cho phép phát triển, vận hành và triển khai ứng dụng trong môi trường **container** – một môi trường nhẹ, biệt lập, chứa đầy đủ mã nguồn, thư viện, cấu hình và các phần mềm cần thiết để ứng dụng hoạt động.

Khác với máy ảo (virtual machine) truyền thống, container sử dụng chung nhân hệ điều hành (OS kernel) và do đó nhẹ hơn, khởi động nhanh hơn và tiêu tốn ít tài nguyên hơn.

### ***Các khái niệm chính trong Docker***



Hình 2.3. Docker Architecture

***Image***: Là một bản đóng gói (snapshot) chứa toàn bộ môi trường, mã nguồn và các thư viện cần thiết. Đây là "bản mẫu" để tạo ra các container.

***Container***: Là một instance đang chạy của một image. Mỗi container hoạt động độc lập và tách biệt với hệ thống chủ cũng như các container khác.

***Dockerfile***: Là tập tin văn bản chứa các lệnh cấu hình dùng để xây dựng image. Ví dụ, cài Node.js, copy mã nguồn, cài đặt thư viện,...

***Docker Hub***: Là kho lưu trữ image trực tuyến, nơi người dùng có thể tải hoặc chia sẻ image.

***Docker Compose***: Là công cụ cho phép định nghĩa và chạy nhiều container cùng lúc (ví dụ: ứng dụng + cơ sở dữ liệu) thông qua file cấu hình docker-compose.yml.

*Docker Client*: là cách mà bạn tương tác với docker thông qua command trong terminal. Docker Client sẽ sử dụng API gửi lệnh tới Docker Daemon.

*Docker Daemon*: là server Docker cho yêu cầu từ Docker API. Nó quản lý images, containers, networks và volume.

*Docker Volumes*: là cách tốt nhất để lưu trữ dữ liệu liên tục cho việc sử dụng và tạo apps.

*Docker Registry*: là nơi lưu trữ riêng của Docker Images. Images được push vào registry và client sẽ pull images từ registry. Có thể sử dụng registry của riêng bạn hoặc registry của nhà cung cấp như : AWS, Google Cloud, Microsoft Azure.

*Docker Repository*: là tập hợp các Docker Images cùng tên nhưng khác tags. VD: golang:1.11-alpine.

*Docker Networking*: cho phép kết nối các container lại với nhau. Kết nối này có thể trên 1 host hoặc nhiều host.

*Docker Swarm*: để phối hợp triển khai container.

*Docker Services*: là các containers trong production. 1 service chỉ run 1 image nhưng nó mã hoá cách thức để run image — sử dụng port nào, bao nhiêu bản sao container run để service có hiệu năng cần thiết và ngay lập tức.

### *Vai trò của Docker*

***Đóng gói toàn bộ ứng dụng và môi trường***: Docker giúp đảm bảo ứng dụng có thể chạy ổn định ở bất kỳ máy nào có Docker, tránh các lỗi thường gặp như “chạy được ở máy em nhưng không chạy ở máy anh”.

***Triển khai nhanh chóng và dễ dàng***: Việc triển khai ứng dụng chỉ cần chạy container từ image đã build sẵn, không cần cài đặt lại môi trường.

***Tách biệt và cô lập ứng dụng***: Mỗi container là một môi trường độc lập, giảm rủi ro xung đột giữa các dịch vụ hoặc phần mềm.

***Quản lý phiên bản dễ dàng***: Có thể tạo các phiên bản image khác nhau cho các bản phát hành của ứng dụng.

***Dễ dàng mở rộng hệ thống***: Có thể nhân bản nhiều container để xử lý song song hoặc phục vụ cân bằng tải.

## RESTful API – Thiết kế API hiệu quả

### *Giới thiệu về RESTful API*

REST (Representational State Transfer) là một phong cách kiến trúc được giới thiệu bởi Roy Fielding trong luận án tiến sĩ năm 2000. API được thiết kế tuân theo các nguyên tắc REST được gọi là RESTful API.

RESTful API sử dụng các phương thức HTTP tiêu chuẩn như:

* GET: Lấy dữ liệu
* POST: Tạo dữ liệu mới
* PUT / PATCH: Cập nhật dữ liệu
* DELETE: Xoá dữ liệu

### *Đặc điểm chính của RESTful API*

***Stateless (Không trạng thái)*:** Mỗi yêu cầu từ client đến server phải độc lập, không lưu trữ trạng thái người dùng giữa các request. Thông tin xác thực (như token) phải được gửi cùng mỗi request.

***Client – Server*:** REST tách biệt giao diện người dùng (client) với dữ liệu và logic xử lý (server). Điều này giúp frontend và backend có thể phát triển độc lập.

***Resource-based (Tài nguyên định danh)*:** Tài nguyên được biểu diễn dưới dạng URI, ví dụ /api/users, /api/rooms.

***Uniform Interface (Giao diện thống nhất)*:** Giao tiếp giữa client và server phải được thống nhất theo chuẩn HTTP và JSON.

***Representation*:** Tài nguyên có thể được đại diện bằng nhiều định dạng khác nhau, phổ biến nhất là **JSON**.

***Caching (Bộ nhớ đệm)*:** REST hỗ trợ cache các response để tăng hiệu suất hệ thống.

## Node.js – Nền tảng chạy JavaScript phía máy chủ (Backend)

### *Khái niệm*

**Node.js** là một nền tảng mã nguồn mở được xây dựng trên **V8 JavaScript Engine** (trình thông dịch JavaScript của Google Chrome), cho phép thực thi mã JavaScript ở phía máy chủ (server-side). Khác với trước đây khi JavaScript chỉ được sử dụng trong trình duyệt để xử lý logic phía client, Node.js đã mở rộng phạm vi sử dụng của ngôn ngữ này sang lĩnh vực phát triển backend, từ đó tạo nên một **hệ sinh thái JavaScript toàn diện** cho cả client và server.

### *Đặc điểm*

***Kiến trúc bất đồng bộ (asynchronous) và hướng sự kiện (event-driven)*:** Node.js sử dụng mô hình non-blocking I/O, giúp xử lý đồng thời nhiều kết nối một cách hiệu quả, rất phù hợp với các ứng dụng real-time (chat, streaming, API service).

***Hiệu suất cao*:** Nhờ sử dụng V8 engine của Google và mô hình xử lý sự kiện, Node.js có tốc độ thực thi nhanh và sử dụng bộ nhớ hiệu quả.

***Hệ sinh thái npm (Node Package Manager)*:** Node.js có một thư viện mã nguồn mở phong phú với hàng trăm nghìn gói (packages), giúp phát triển ứng dụng nhanh chóng (ví dụ: Express, Mongoose, Socket.io,...).

***Khả năng mở rộng*:** Dễ dàng xây dựng hệ thống microservices, hoặc tích hợp với Docker và các công cụ CI/CD.

## React.js – Framework phát triển ứng dụng web React hiện đại (Frontend)

### *Khái niệm*

React.js là một thư viện JavaScript mã nguồn mở được phát triển bởi Facebook dùng để xây dựng giao diện người dùng (UI). Tập trung vào phần “View” trong mô hình “MVC” cho phép xây dựng các component (thành phần) có thể tái sử dụng và dễ bảo trì.

### *Tính năng nổi bật*

Kiến trúc dựa trên thành phần (Component-Based): Cho phép chia nhỏ giao diện thành các phần nhỏ, dễ quản lý và tái sử dụng.

Virtual DOM: Tăng hiệu suất hiển thị bằng cách cập nhật phần giao diện bị thay đổi thay vì toàn bộ trang.

One-way Data Binding: Dữ liệu chỉ chảy theo một chiều, giúp kiểm soát trạng thái dễ hơn và giảm lỗi.

JSX (JavaScript XML): Cho phép viết mã HTML trong JavaScript, giúp code trực quan và dễ hiểu hơn.

Hỗ trợ xây dựng SPA (Single Page Application): Tạo trải nghiệm người dùng mượt mà, không cần tải lại trang.

Tính mở rộng cao: Dễ tích hợp với các thư viện khác như Redux, React Router,… để mở rộng chức năng.

Thân thiện với kiểm thử: Các component độc lập giúp dễ dàng viết và chạy các bài test.

Cộng đồng lớn và tài liệu phong phú: Hệ sinh thái mạnh mẽ với nhiều tài nguyên hỗ trợ từ cộng đồng và Facebook.

## MongoDB – Hệ quản trị cơ sở dữ liệu NoSQL

### *Khái niệm*

**MongoDB** là một hệ quản trị cơ sở dữ liệu NoSQL mã nguồn mở, được thiết kế để lưu trữ và truy xuất dữ liệu theo mô hình **document-oriented** (hướng tài liệu), trong đó dữ liệu được biểu diễn dưới dạng tài liệu **BSON** (Binary JSON). Khác với cơ sở dữ liệu quan hệ (RDBMS) như MySQL hay PostgreSQL, MongoDB không sử dụng bảng, dòng và cột, mà thay vào đó tổ chức dữ liệu theo các **collection** và **document** linh hoạt, không ràng buộc lược đồ cứng (schema-less).

### *Đặc điểm nổi bật của MongoDB*

***Không có lược đồ cố định (Schema-less)*:** Các document trong cùng một collection có thể có cấu trúc khác nhau, cho phép lưu trữ dữ liệu linh hoạt và mở rộng theo nhu cầu thực tế.

***Lưu trữ dưới dạng JSON/BSON*:** Dữ liệu được biểu diễn ở định dạng dễ hiểu (JSON-like), thuận tiện cho việc xử lý và truyền dữ liệu giữa client – server, đặc biệt với các ứng dụng JavaScript/Node.js.

***Khả năng mở rộng ngang (horizontal scaling)*:** MongoDB hỗ trợ **sharding** – phân mảnh dữ liệu qua nhiều máy chủ, giúp tăng khả năng xử lý khi hệ thống có khối lượng lớn dữ liệu.

***Hiệu suất cao*:** MongoDB sử dụng cơ chế ghi không đồng bộ và bộ nhớ đệm (caching) để tối ưu hiệu năng truy xuất dữ liệu.

***Tích hợp tốt với các ngôn ngữ hiện đại*:** MongoDB cung cấp thư viện cho hầu hết các ngôn ngữ lập trình như JavaScript (Mongoose, MongoDB Driver), Python, Java, Go, C#,...

### *Các khái niệm cơ bản trong MongoDB*

| **Khái niệm** | **Mô tả** |
| --- | --- |
| **Database** | Một tập hợp các collection |
| **Collection** | Tương tự bảng trong RDBMS, chứa nhiều document |
| **Document** | Một bản ghi dữ liệu, thường ở dạng JSON hoặc BSON |
| **\_id** | Khóa chính mặc định, tự động sinh cho mỗi document |
| **Query** | Câu truy vấn dữ liệu theo điều kiện linh hoạt (tương tự WHERE trong SQL) |

Bảng 2. Các khái niệm cơ bản trong MongoDB

# PHÂN TÍCH YÊU CẦU

## Các chức năng chính của hệ thống (Functional Requirements).

### *Quản lý người dùng*

Đăng ký tài khoản:

* Cho phép người dùng tạo tài khoản bằng email, mật khẩu và thông tin cơ bản.
* Hệ thống kiểm tra trùng lặp email và xác nhận mật khẩu hợp lệ.

Đăng nhập/đăng xuất:

* Cho phép người dùng đăng nhập bằng email và mật khẩu.
* Sử dụng mã thông báo (JWT) để xác thực và duy trì phiên làm việc.

Phân quyền người dùng:

* Phân biệt giữa người dùng, chủ trọ và admin duyệt bài đăng.
* Các chức năng hiển thị tương ứng với vai trò đã đăng ký.

Cập nhật thông tin cá nhân:

* Cho phép người dùng chỉnh sửa tên, số điện thoại, mật khẩu,…

### *Quản lý phòng trọ (Room Management)*

Tìm kiếm phòng trọ:

* Cho phép người dùng tìm kiếm phòng theo từ khóa, vị trí, khoảng giá.
* Kết hợp lọc (filter) theo từ khóa như: có máy lạnh, wifi, toilet riêng,...

Xem danh sách phòng trọ:

* Hiển thị danh sách phòng trọ dưới dạng lưới hoặc danh sách, có phân trang hoặc cuộn vô hạn (infinite scroll).
* Mỗi mục hiển thị tiêu đề, giá, vị trí, hình ảnh.

Xem chi tiết phòng trọ:

* Hiển thị đầy đủ thông tin như mô tả, địa chỉ cụ thể, hình ảnh, số điện thoại liên hệ,...

### *Quản lý bài đăng cho thuê (Dành cho chủ trọ)*

Tạo bài đăng mới:

* Chủ trọ có thể tạo bài đăng phòng trọ với đầy đủ thông tin như tiêu đề, mô tả, giá thuê, vị trí, hình ảnh,...
* Hệ thống kiểm tra dữ liệu trước khi lưu.

Xóa bài đăng:

* Cho phép chủ trọ xóa bài đăng nếu phòng đã cho thuê hoặc không còn nhu cầu.

Quản lý danh sách bài đăng:

* Hiển thị danh sách các bài đăng của chủ trọ, bao gồm trạng thái hiển thị, ngày đăng và lượt đánh giá.

### *Tương tác người dùng*

Đánh dấu phòng yêu thích:

* Người thuê có thể đánh giá số sao với phòng trọ yêu thích.

Liên hệ chủ trọ:

* Hiển thị số điện thoại hoặc email của chủ trọ.
* (Mở rộng): Có thể gửi tin nhắn cho chủ trọ qua biểu mẫu liên hệ.

Gửi phản hồi hoặc báo cáo bài đăng:

* (Mở rộng): Cho phép người dùng báo cáo bài viết sai lệch, lừa đảo,...

### *Quản lý hệ thống (Admin)*

Phê duyệt bài đăng:

* Quản trị viên có thể duyệt thủ công hoặc tự động bài viết trước khi hiển thị (tùy chính sách).

Xóa bài đăng vi phạm:

* Gỡ bài không đúng nội dung, spam hoặc bị báo cáo.

### *Bảo mật và xác thực*

Mã hóa mật khẩu:

* Mật khẩu người dùng được mã hóa trước khi lưu vào cơ sở dữ liệu (bcrypt).

Xác thực bằng JWT:

* Sử dụng JSON Web Token để xác thực người dùng khi truy cập API.

Kiểm tra quyền truy cập:

* Phân quyền chức năng dựa trên vai trò người dùng.

### *RESTful API (Backend giao tiếp với Frontend)*

Các chức năng frontend đều giao tiếp với backend thông qua các RESTful API, ví dụ:

* POST /api/login: Đăng nhập
* GET /api/rooms: Lấy danh sách phòng
* POST /api/rooms: Tạo phòng mới
* PUT /api/rooms/:id: Cập nhật phòng
* DELETE /api/rooms/:id: Xoá phòng
* GET /api/users/me: Lấy thông tin người dùng hiện tại

## Các yêu cầu phi chức năng (Non-functional Requirements).

Tính hiệu năng (Performance Requirements)

* Hệ thống phải có khả năng xử lý **ít nhất 100 yêu cầu truy cập/phút** mà không xảy ra tình trạng treo hoặc chậm phản hồi.
* Tốc độ phản hồi của các API RESTful không vượt quá **1 giây** trong điều kiện tải bình thường.
* Giao diện người dùng phải được hiển thị đầy đủ trong vòng **< 2 giây** kể từ khi người dùng truy cập trang.

Tính sẵn sàng (Availability)

* Hệ thống cần hoạt động ổn định với thời gian sẵn sàng (uptime) tối thiểu 99%.
* Trong trường hợp xảy ra lỗi máy chủ backend hoặc database, hệ thống cần có cơ chế hiển thị thông báo lỗi rõ ràng và ghi log để khắc phục nhanh chóng.

Khả năng mở rộng (Scalability)

* Hệ thống cần được thiết kế để dễ dàng mở rộng, bao gồm:
* Thêm các chức năng mới (ví dụ: thanh toán, chatbot, đề xuất thông minh).
* Hỗ trợ tăng số lượng người dùng đồng thời mà không ảnh hưởng đến hiệu suất.
* Cơ sở dữ liệu NoSQL (MongoDB) hỗ trợ mở rộng ngang (horizontal scaling) thông qua sharding hoặc replica set.

Bảo mật (Security Requirements)

* Dữ liệu nhạy cảm như mật khẩu phải được mã hóa (sử dụng thuật toán bcrypt hoặc tương đương).
* Hệ thống sử dụng JWT (JSON Web Token) để xác thực và phân quyền truy cập các tài nguyên.
* Các API phải được kiểm tra quyền truy cập, không cho phép người dùng không hợp lệ truy cập trái phép tài nguyên.

Tính khả dụng (Usability)

* Giao diện được thiết kế theo nguyên tắc **UI/UX thân thiện, dễ sử dụng**, phù hợp với cả người dùng phổ thông.
* Hệ thống hỗ trợ **responsive design**: tương thích với các thiết bị khác nhau (máy tính, máy tính bảng, điện thoại).
* Các thao tác quan trọng (đăng nhập, tìm kiếm, đăng bài) phải được thực hiện với **số bước tối thiểu**.

Dễ bảo trì và phát triển (Maintainability & Extensibility)

* Mã nguồn được tổ chức theo module rõ ràng, sử dụng các framework phổ biến và có tài liệu đầy đủ như: Node.js, Express, Next.js, MongoDB, Docker.
* Dễ dàng cập nhật, nâng cấp các thành phần frontend/backend độc lập.
* Hệ thống API được mô tả rõ ràng bằng tài liệu Swagger, giúp các lập trình viên dễ hiểu và tái sử dụng.

Khả năng kiểm thử (Testability)

* Mỗi thành phần trong hệ thống có thể được kiểm thử độc lập:
* API backend kiểm thử bằng Postman và script tự động.
* Frontend kiểm thử thủ công qua trình duyệt.
* Có thể tích hợp kiểm thử tự động (CI/CD) bằng GitHub Actions.

Khả năng triển khai và vận hành (Deployability)

* Hệ thống có thể được **triển khai dễ dàng bằng Docker**, hỗ trợ môi trường container hoá.
* Có thể chạy trên các nền tảng cloud như **Heroku, Vercel, hoặc DigitalOcean** với cấu hình đơn giản.
* Hệ thống cần có cơ chế ghi log (logging) để giám sát hoạt động và phát hiện lỗi.

# THIẾT KẾ HỆ THỐNG

## Kiến trúc tổng thể: Sơ đồ kiến trúc hệ thống (client/server, API, cơ sở dữ liệu).

### *Cấu trúc thư mục hệ thống*

CNPM\_timkiem\_phongtro/

├── auth-service/ # Service xử lý xác thực (đăng nhập, đăng ký, cấp token)

│ ├── Dockerfile # Cấu hình Docker để đóng gói service

│ ├── package.json # Quản lý dependencies Node.js

│ ├── node\_modules/ # Thư viện đã cài đặt

│ └── src/ # Mã nguồn chính

│ ├── controllers/ # Xử lý request từ client (auth.controller.ts)

│ ├── index.ts # Điểm khởi động service

│ ├── models/ # Định nghĩa dữ liệu người dùng (user.model.ts)

│ ├── routes/ # Khai báo các endpoint xác thực

│ └── services/ # Logic nghiệp vụ xác thực (auth.service.ts)

├── docker/ # Thư mục chứa cấu hình Docker và dữ liệu mẫu MongoDB

│ ├── docker-compose.yml # Cấu hình để chạy toàn bộ hệ thống (multi-container)

│ └── dump/ # Dữ liệu mẫu để khôi phục MongoDB

│ └── findhouse/

│ ├── posts.bson # Dữ liệu bài đăng

│ ├── posts.metadata.json # Metadata bài đăng

│ ├── prelude.json # File thiết lập chung

│ ├── users.bson # Dữ liệu người dùng

│ └── users.metadata.json # Metadata người dùng

├── frontend/ # Ứng dụng giao diện người dùng (React)

│ ├── Dockerfile # Đóng gói giao diện thành Docker container

│ ├── package.json # Quản lý dependencies React

│ ├── node\_modules/ # Thư viện frontend

│ └── src/ # Mã nguồn React

│ ├── api/ # Gọi API backend (auth, user, post)

│ ├── App.tsx # Component chính của ứng dụng

│ ├── components/ # Thành phần giao diện dùng lại (Header)

│ ├── pages/ # Các trang chức năng (Login, Register, Search, ...)

│ └── index.tsx # Điểm khởi động React app

├── gateway/ # API Gateway – trung gian nhận và định tuyến request

│ ├── Dockerfile # Docker cấu hình cho gateway

│ ├── package.json # Quản lý dependencies

│ ├── node\_modules/ # Thư viện đã cài

│ └── src/

│ └── index.ts # Xử lý định tuyến request đến các service

├── post-service/ # Service quản lý bài đăng trọ

│ ├── Dockerfile # Cấu hình Docker

│ ├── package.json # Thư viện backend

│ ├── node\_modules/ # Dependencies

│ └── src/

│ ├── controllers/ # Xử lý các request bài đăng (CRUD)

│ ├── index.ts # Điểm khởi động service

│ ├── middleware.ts # Middleware xử lý request

│ ├── models/ # Định nghĩa dữ liệu bài đăng (post.model.ts)

│ ├── routes/ # Khai báo endpoint bài đăng

│ └── services/ # Logic nghiệp vụ liên quan bài đăng

│ └── uploads/

│ └── images/ # Thư mục lưu ảnh upload

├── shared/ # Cấu hình TypeScript dùng chung giữa các service

│ └── tsconfig.base.json # Cấu hình TS cơ bản dùng lại trong toàn bộ hệ thống

├── user-service/ # Service quản lý thông tin người dùng (profile, cập nhật, ...)

│ ├── Dockerfile # Cấu hình đóng gói Docker

│ ├── env.example # File cấu hình biến môi trường mẫu

│ ├── package.json # Dependencies Node.js

│ ├── node\_modules/ # Thư viện backend

│ └── src/

│ ├── controllers/ # Xử lý request user (user.controller.ts)

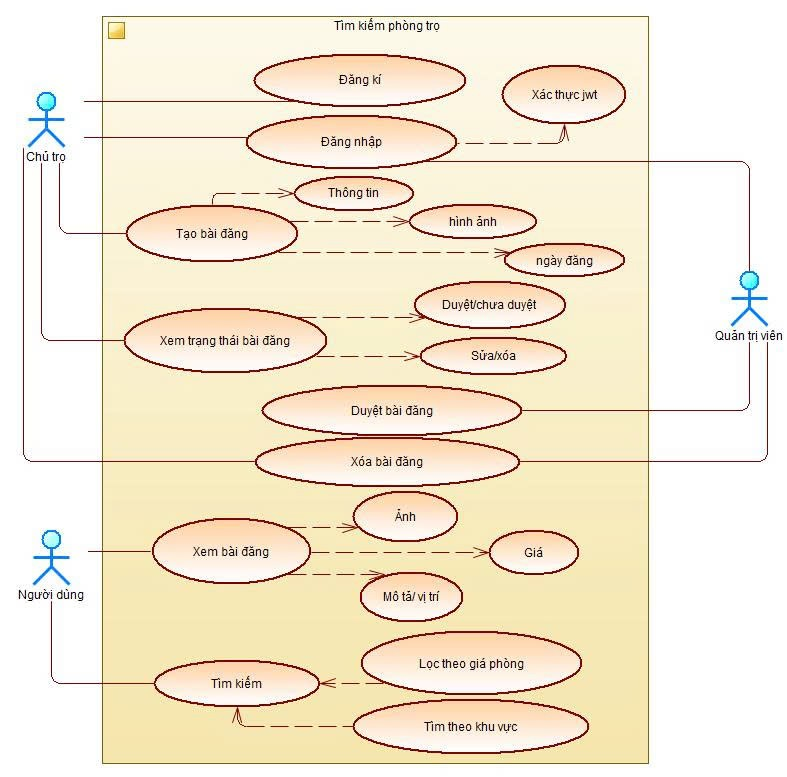
│ ├── index.ts # Khởi động service

│ ├── middleware/ # Middleware xác thực JWT

│ ├── routes/ # Khai báo API cho người dùng

│ └── services/ # Logic nghiệp vụ người dùng

### *Sơ đồ Usse case cho hệ thống*



Hình 4.1. Sơ đồ Usse case

1. *Các tác nhân (Actors)*

|  |  |
| --- | --- |
| **Tác nhân** | **Vai trò** |
| **Chủ trọ** | Đăng ký tài khoản, tạo và quản lý bài đăng phòng trọ. |
| **Người dùng** | Tìm kiếm, xem bài đăng, lọc phòng trọ theo tiêu chí. |
| **Quản trị viên** | Duyệt, sửa, xoá bài đăng để đảm bảo tính hợp lệ của nội dung. |

Bảng 3. Các tác nhân của hệ thống

1. *Các chức năng chính (Use Cases)*

Đăng ký / Đăng nhập / Xác thực

* **Đăng ký**: Chủ trọ đăng ký tài khoản.
* **Đăng nhập**: Tất cả các tác nhân đăng nhập để truy cập hệ thống.
* **Xác thực JWT**: Bảo vệ phiên đăng nhập bằng token xác thực.

Quản lý bài đăng (Chủ trọ)

* **Tạo bài đăng**: Bao gồm nhập **thông tin, hình ảnh, ngày đăng**.
* **Xem trạng thái bài đăng**: Chủ trọ kiểm tra bài đã duyệt/chưa duyệt.
* **Sửa/xoá bài đăng**: Chủ trọ chỉnh sửa hoặc xóa các bài đã đăng.

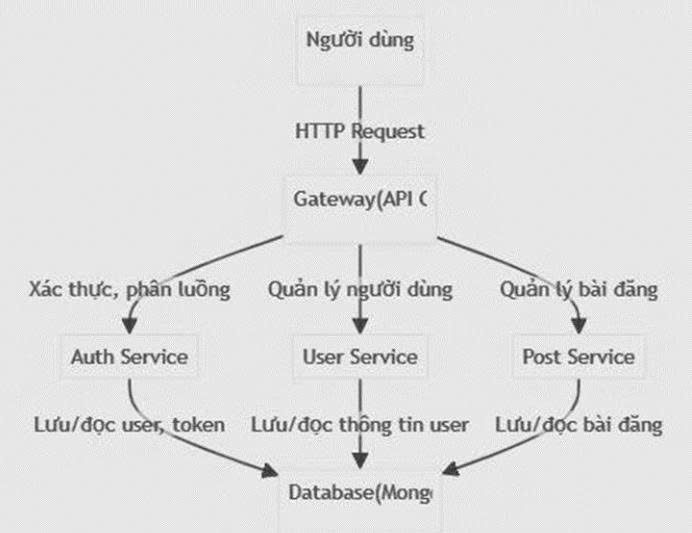
Duyệt bài đăng (Quản trị viên)

* **Duyệt bài đăng**: Xét duyệt bài đăng của chủ trọ.
* **Xóa bài đăng**: Có thể xóa bài vi phạm hoặc không hợp lệ.

**Tìm kiếm phòng trọ (Người dùng)**

* **Xem bài đăng**: Xem chi tiết bài đăng gồm **ảnh, giá, mô tả/vị trí**.
* **Tìm kiếm**: Tìm kiếm phòng theo từ khoá.
* **Lọc theo giá phòng**: Lọc kết quả theo khoảng giá.
* **Tìm theo khu vực**: Lọc theo vị trí địa lý hoặc địa phương cụ thể.

### *Sơ đồ luồng xử lý dữ liệu của hệ thống web sử dụng mô hình microservices.*



Hình 4.2. Sơ đồ luồng xử lý dữ liệu

*1. Người dùng (User)*

Vai trò: Là người sử dụng hệ thống, thao tác qua giao diện web hoặc app.

Tương tác: Gửi các yêu cầu (request) như đăng nhập, đăng ký, xem bài đăng, tạo bài đăng... đến hệ thống.

*2. Gateway (API Gateway)*

Vai trò: Là cổng vào duy nhất của hệ thống, nhận tất cả các request từ người dùng.

Chức năng:

Định tuyến (route) các request đến đúng service phía sau.

Có thể kiểm tra xác thực (authentication), phân quyền (authorization), log, kiểm soát tần suất truy cập (rate limiting), v.v.

*3. Auth Service*

Vai trò: Xử lý các chức năng liên quan đến xác thực (authentication) như đăng nhập, đăng ký, cấp phát token.

Tương tác: Khi người dùng đăng nhập/đăng ký, Gateway sẽ chuyển tiếp request đến Auth Service.

Kết nối Database: Lưu trữ và truy xuất thông tin tài khoản, token xác thực.

*4. User Service*

Vai trò: Quản lý thông tin người dùng (profile, cập nhật thông tin cá nhân, v.v.).

Tương tác: Gateway chuyển các request liên quan đến thông tin người dùng cho User Service xử lý.

Kết nối Database: Lưu trữ và truy xuất thông tin chi tiết của người dùng.

*5. Post Service*

Vai trò: Quản lý các bài đăng (tạo, xóa, tìm kiếm bài đăng phòng trọ...).

Tương tác: Gateway chuyển các request liên quan đến bài đăng cho Post Service xử lý.

Kết nối Database: Lưu trữ và truy xuất dữ liệu các bài đăng.

*6. Database*

Vai trò: Lưu trữ dữ liệu cho toàn bộ hệ thống (người dùng, bài đăng, token...).

Kết nối: Các service (Auth, User, Post) đều kết nối trực tiếp đến Database để lưu và lấy dữ liệu cần thiết.

Tóm lại: Người dùng chỉ giao tiếp với hệ thống qua Gateway. Gateway sẽ định tuyến các request đến các service chuyên biệt (Auth, User, Post). Mỗi service sẽ xử lý nghiệp vụ của mình và thao tác với Database để lưu trữ hoặc truy xuất dữ liệu.

## Thiết kế cơ sở dữ liệu:

### *Loại cơ sở dữ liệu sử dụng*

Hệ thống sử dụng **MongoDB** – một hệ quản trị cơ sở dữ liệu NoSQL dạng document. Dữ liệu được lưu trữ dưới dạng các **document** (tài liệu JSON) trong các **collection** (tương đương bảng trong SQL). Quan hệ giữa các thực thể được quản lý tại tầng **backend service**, không ràng buộc cứng như trong SQL.

1. *Các collection chính trong MongoDB*

Collection users

* Lưu trữ thông tin người dùng hệ thống.
* Trường điển hình:
* \_id: Mã định danh tự động của MongoDB.
* name: Tên người dùng.
* email: Địa chỉ email (dùng để đăng nhập).
* password: Mật khẩu đã mã hóa.
* role: Vai trò người dùng (user, admin, v.v).
* avatar (tuỳ chọn): Ảnh đại diện.
* phone (tuỳ chọn): Số điện thoại.

Collection posts

* Lưu thông tin bài đăng phòng trọ.
* Trường điển hình:
* \_id: Mã định danh bài viết.
* title: Tiêu đề bài đăng.
* description: Mô tả chi tiết.
* note: Ghi chú thêm (tuỳ chọn).
* status: Trạng thái (ví dụ: đang hiển thị, đã ẩn).
* price: Giá thuê.
* address: Địa chỉ phòng trọ.
* userId: Tham chiếu đến \_id của users.
* imageIds: Danh sách ID ảnh (liên kết với collection images).
* createdAt, updatedAt: Thời gian tạo và cập nhật.

Collection images

* Lưu trữ thông tin ảnh đính kèm bài đăng.
* Trường điển hình:
* \_id: Mã định danh ảnh.
* url: Đường dẫn tới ảnh.
* postId: Tham chiếu đến \_id trong collection posts.

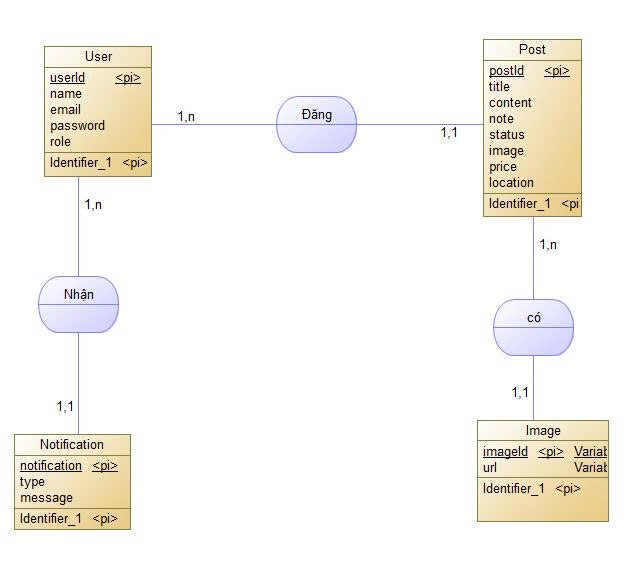
Collection notifications

* Lưu thông báo gửi đến người dùng.
* Trường điển hình:
* \_id: Mã định danh thông báo.
* type: Loại thông báo (ví dụ: tin mới, hệ thống, cảnh báo...).
* message: Nội dung thông báo.
* userId: Tham chiếu đến người dùng nhận thông báo (liên kết với \_id trong users).
* createdAt: Thời gian tạo thông báo.

1. *Các service sử dụng CSDL*

* auth-service
* Xử lý đăng nhập, đăng ký, xác thực người dùng.
* Truy vấn collection users.
* user-service
* Quản lý thông tin cá nhân, cập nhật profile, đổi mật khẩu, v.v.
* Truy vấn collection users.
* post-service
* Tạo, sửa, xoá, tìm kiếm bài đăng.
* Truy vấn posts, images.
* notification-service
* Tạo và gửi thông báo đến người dùng.
* Truy vấn collection notifications.

### *Mô hình ERD*



Hình 4.3. Mô hình ERD

Các thực thể

*User:*

* Đại diện cho người dùng hệ thống, có thể là người thuê phòng hoặc chủ trọ.
* userId (Primary Key - PK): Mã định danh người dùng.
* name: Tên người dùng.
* email: Địa chỉ email.
* password: Mật khẩu đăng nhập (nên được mã hóa).
* role: Vai trò (user/admin/landlord).
* Quan hệ:
* Mỗi người dùng có thể đăng nhiều bài viết (Post).
* Mỗi người dùng có thể nhận nhiều thông báo (Notification).

*Post:*

* Đại diện cho bài đăng cho thuê phòng trọ.
* postId (PK): Mã bài đăng.
* title: Tiêu đề bài viết.
* content: Nội dung chi tiết.
* note: Ghi chú thêm.
* status: Trạng thái bài (đang hiển thị, ẩn, đã cho thuê...).
* image: Tham chiếu tới hình ảnh (nhiều ảnh).
* price: Giá thuê.
* location: Địa chỉ hoặc khu vực.
* Quan hệ:
* Mỗi bài đăng do một người dùng tạo (1-1).
* Mỗi bài đăng có thể có nhiều hình ảnh.

*Image:*

* Đại diện cho hình ảnh minh họa cho bài đăng.
* imageId (PK): Mã ảnh.
* url: Đường dẫn ảnh.
* Các thuộc tính phụ: variant1, variant2 (có thể là các kích thước hoặc phiên bản khác của ảnh).
* Quan hệ:
* Một bài đăng có thể chứa nhiều hình ảnh (1-N).
* Mỗi hình ảnh thuộc về chính xác một bài đăng (1-1).

*Notification:*

* Đại diện cho thông báo hệ thống gửi đến người dùng.
* notificationId (PK): Mã thông báo.
* type: Loại thông báo (ví dụ: hệ thống, cảnh báo, tin nhắn mới,...).
* message: Nội dung thông báo.
* Quan hệ:
* Mỗi thông báo gửi đến một người dùng cụ thể.
* Một người dùng có thể nhận nhiều thông báo.

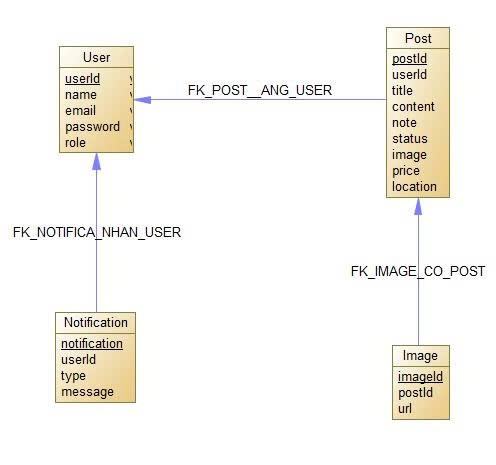
Các mối quan hệ (Relationships)

|  |  |  |  |
| --- | --- | --- | --- |
| **Mối quan hệ** | **Giữa các thực thể** | **Bản chất** | **Diễn giải** |
| Đăng | User – Post | 1:N | Một người dùng **có thể đăng nhiều bài**, mỗi bài đăng chỉ thuộc **một người dùng** |
| Có | Post – Image | 1:N | Một bài đăng **có nhiều ảnh**, mỗi ảnh thuộc **duy nhất một bài** |
| Nhận | User – Notification | 1:N | Một người dùng **có thể nhận nhiều thông báo**, mỗi thông báo chỉ gửi đến **một người** |

Bảng 4. Các mối quan hệ của các thực thể trong mô hình ERD

* Trong mô hình:
* Ký hiệu 1,1 nghĩa là bắt buộc có 1 (ràng buộc toàn phần).
* Ký hiệu 1,n nghĩa là có từ 1 đến nhiều.
* Các vòng tròn thể hiện mối quan hệ, đặt tên như "Đăng", "Có", "Nhận".

### *Mô hình vật lý.*



Hình 4.4. Mô hình vật lý

1. *Các bảng dữ liệu*

User

* Đại diện cho người dùng của hệ thống.

|  |  |
| --- | --- |
| **Cột** | **Ý nghĩa** |
| userId | **Khóa chính (PK)** – mã định danh người dùng |
| name | Tên người dùng |
| email | Email đăng ký |
| password | Mật khẩu đăng nhập |
| role | Vai trò (ví dụ: user, admin, landlord) |

Bảng 5. Dữ liệu của user

Post

* Đại diện cho bài đăng cho thuê phòng trọ.

| **Cột** | **Ý nghĩa** |
| --- | --- |
| postId | **Khóa chính (PK)** – mã bài đăng |
| userId | **Khóa ngoại (FK)** – tham chiếu đến người đăng (User) |
| title | Tiêu đề bài đăng |
| content | Nội dung mô tả |
| note | Ghi chú thêm |
| status | Trạng thái bài đăng |
| image | Trường liên quan ảnh (có thể không cần nếu có bảng Image) |
| price | Giá thuê |
| location | Địa điểm |

Bảng 6. Dữ liệu của Post

Notification

* Đại diện cho thông báo hệ thống gửi đến người dùng.

|  |  |
| --- | --- |
| **Cột** | **Ý nghĩa** |
| notification | **Khóa chính (PK)** – mã thông báo |
| userId | **Khóa ngoại (FK)** – người nhận thông báo |
| type | Loại thông báo (hệ thống, cảnh báo, v.v.) |
| message | Nội dung thông báo |

Bảng 7. Dữ liệu của Notifition

Image

* Đại diện cho hình ảnh đính kèm trong bài đăng.

|  |  |
| --- | --- |
| **Cột** | **Ý nghĩa** |
| imageId | **Khóa chính (PK)** – mã ảnh |
| postId | **Khóa ngoại (FK)** – bài đăng chứa ảnh |
| url | Đường dẫn ảnh |

Bảng 8. Dữ liệu của Image

1. *Các khóa ngoại và quan hệ*

| **Khóa ngoại (FK)** | **Mối quan hệ** |
| --- | --- |
| FK\_POST\_DANG\_USER | Mỗi bài đăng (Post) do một người dùng (User) tạo ra. |
| FK\_NOTIFICA\_NHAN\_USER | Mỗi thông báo (Notification) thuộc về một người dùng. |
| FK\_IMAGE\_CO\_POST | Mỗi hình ảnh (Image) thuộc về một bài đăng (Post). |

Bảng 9. Các khóa ngoại và quan hệ

Người dùng (User) có thể:

* Đăng nhiều bài viết (Post)
* Nhận nhiều thông báo (Notification)

Mỗi bài viết (Post) có thể:

* Đính kèm nhiều hình ảnh (Image)

Hệ thống lưu trữ thông báo (Notification) để thông tin đến từng người dùng, ví dụ: "Phòng của bạn đã được duyệt", "Bài đăng bị báo cáo", v.v.

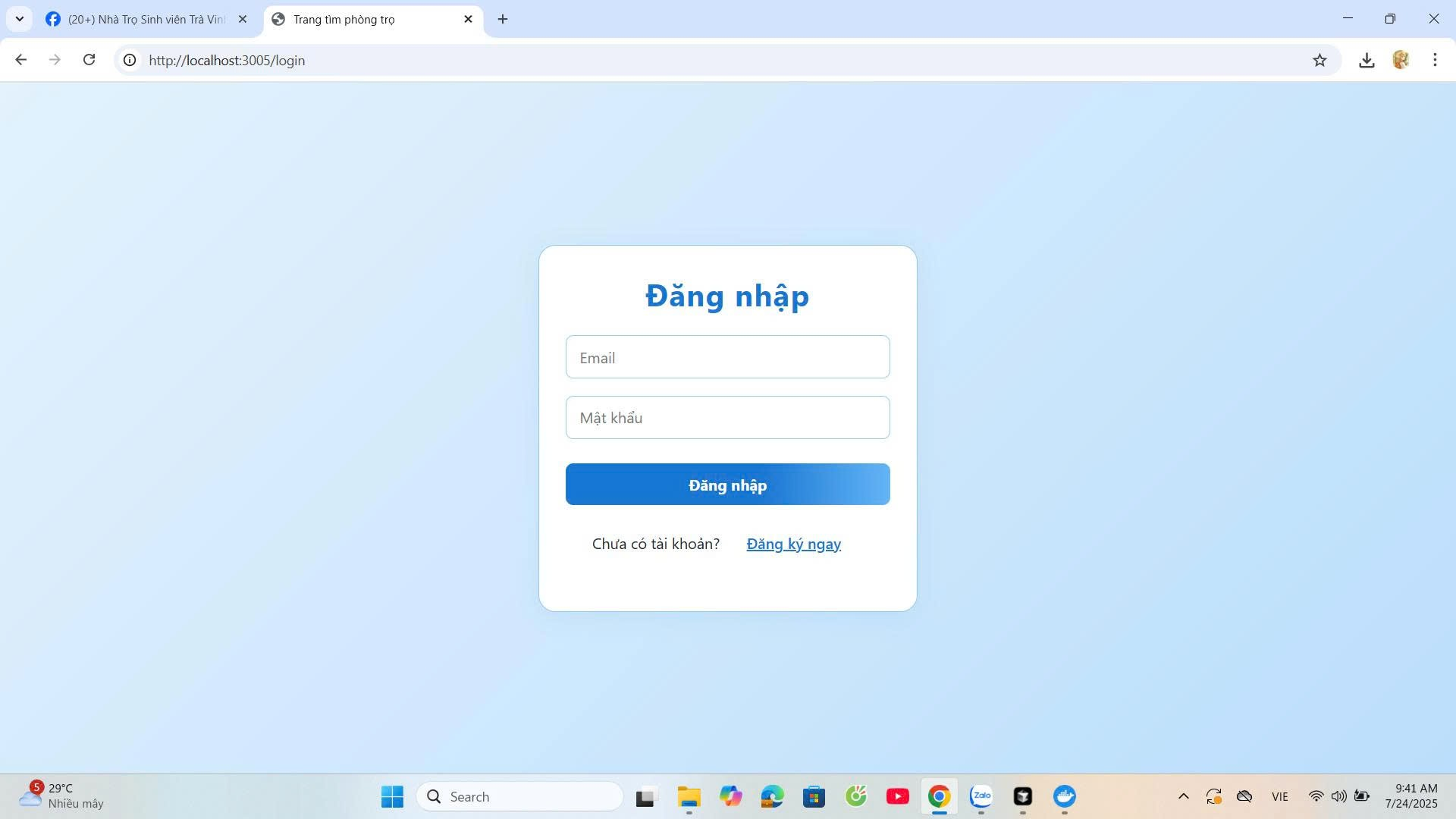
## Thiết kế giao diện (UI/UX): Ảnh chụp các màn hình chính và liên kết với bản thiết kế Figma.



Hình 4.5 Thiết kế figma

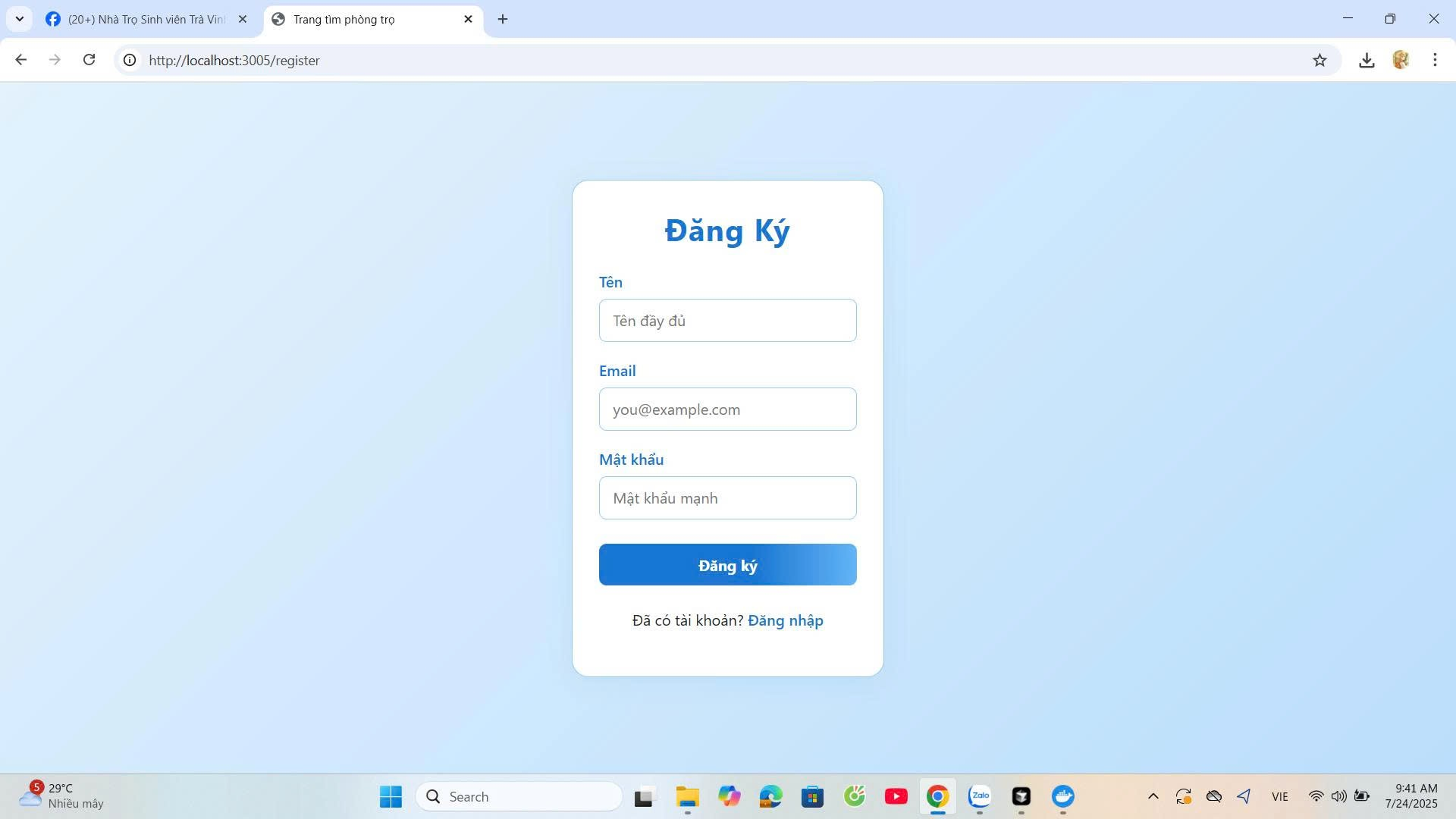
## Các giao diện website

### Giao diện trang đăng nhập



Hình 4.6 Giao diện trang đăng nhập

### Giao diện đăng kí

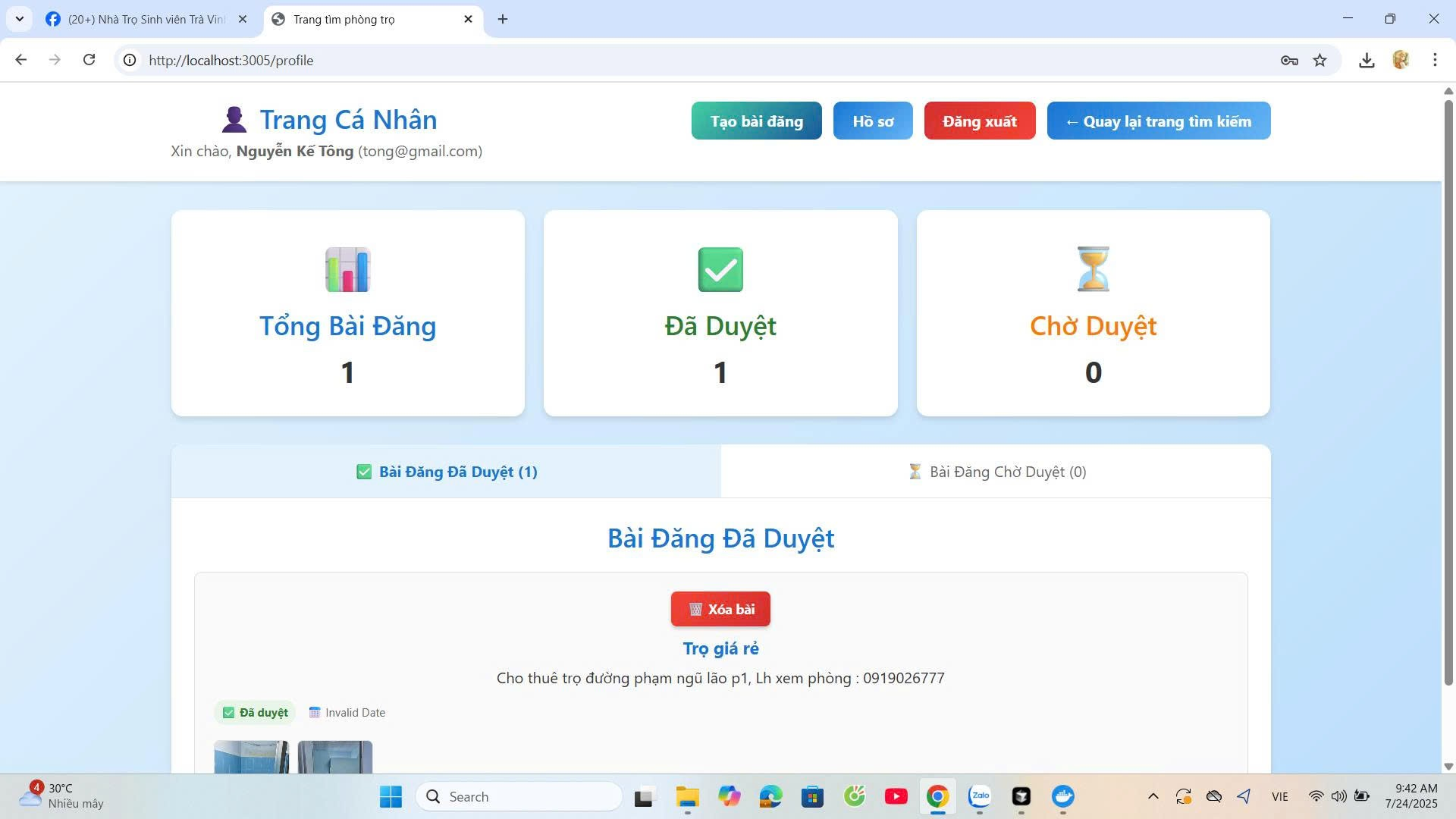


Hình 4.7 Giao diện trang đăng kí

### Giao diện trang tìm kiếm

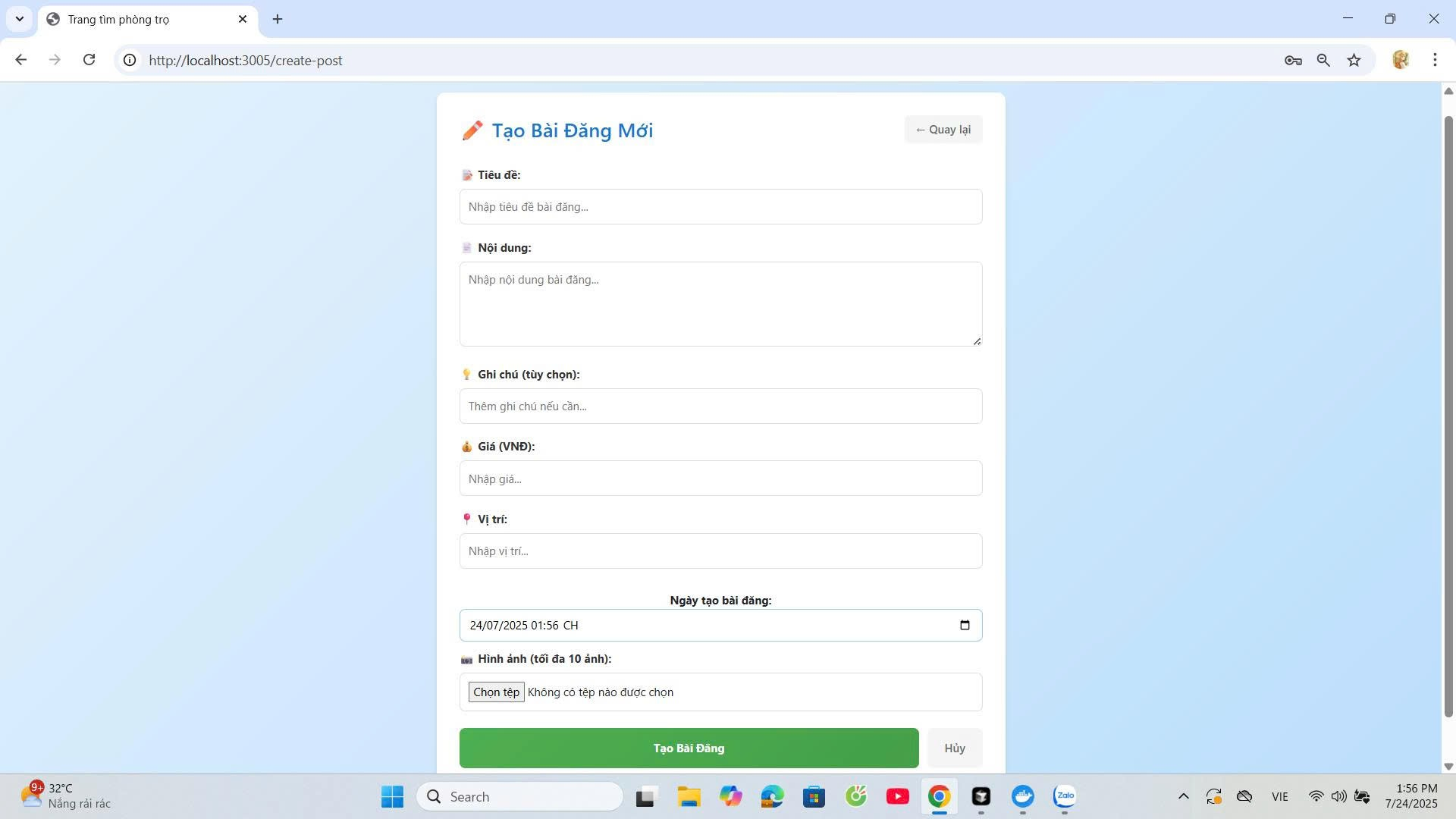
Hình 4.8 Giao diện trang tìm kiếm

### Giao diện trang cá nhân



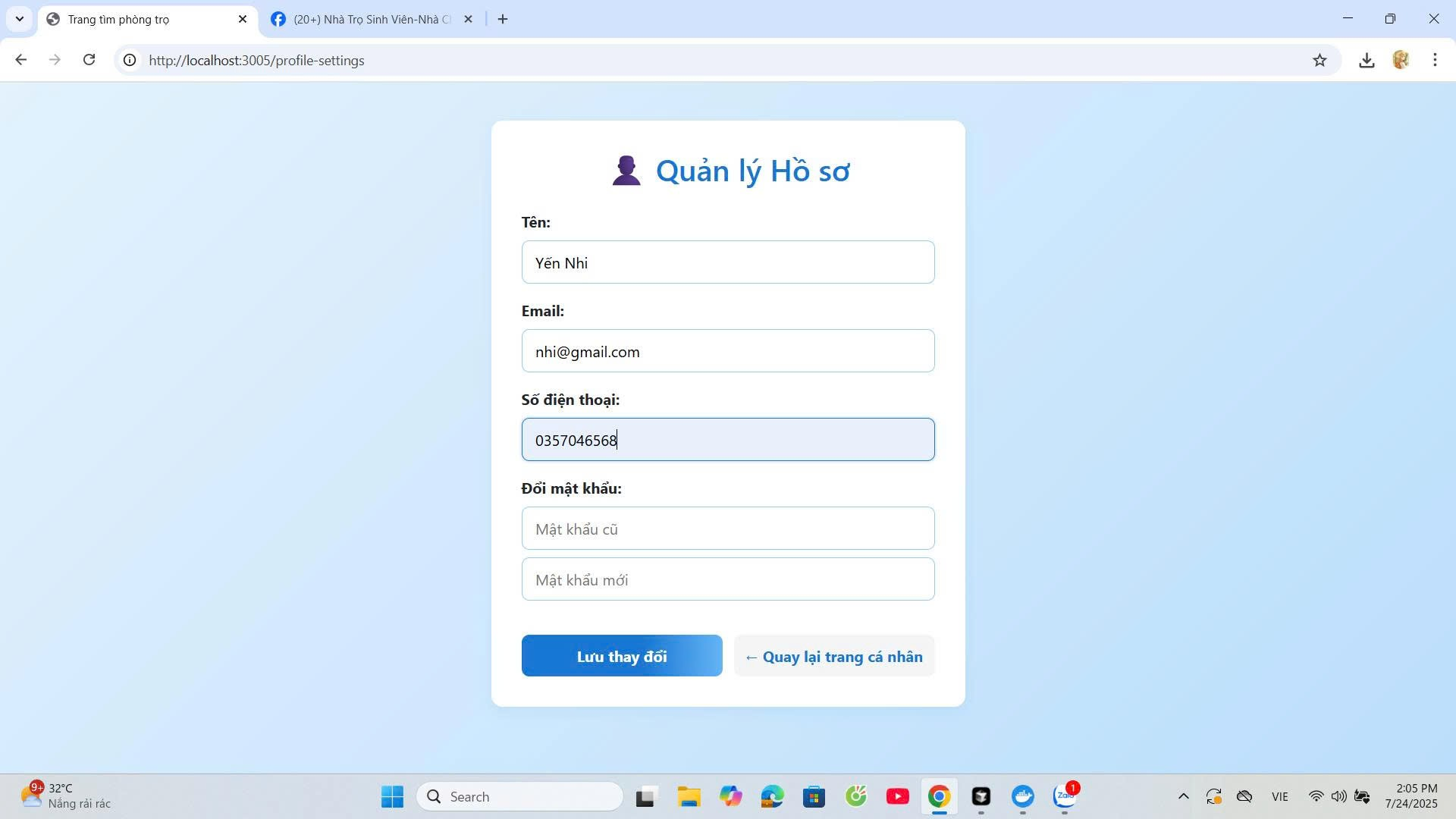
Hình 4.9 Giao diện trang cá nhân

### Giao diện tạo bài đăng mới



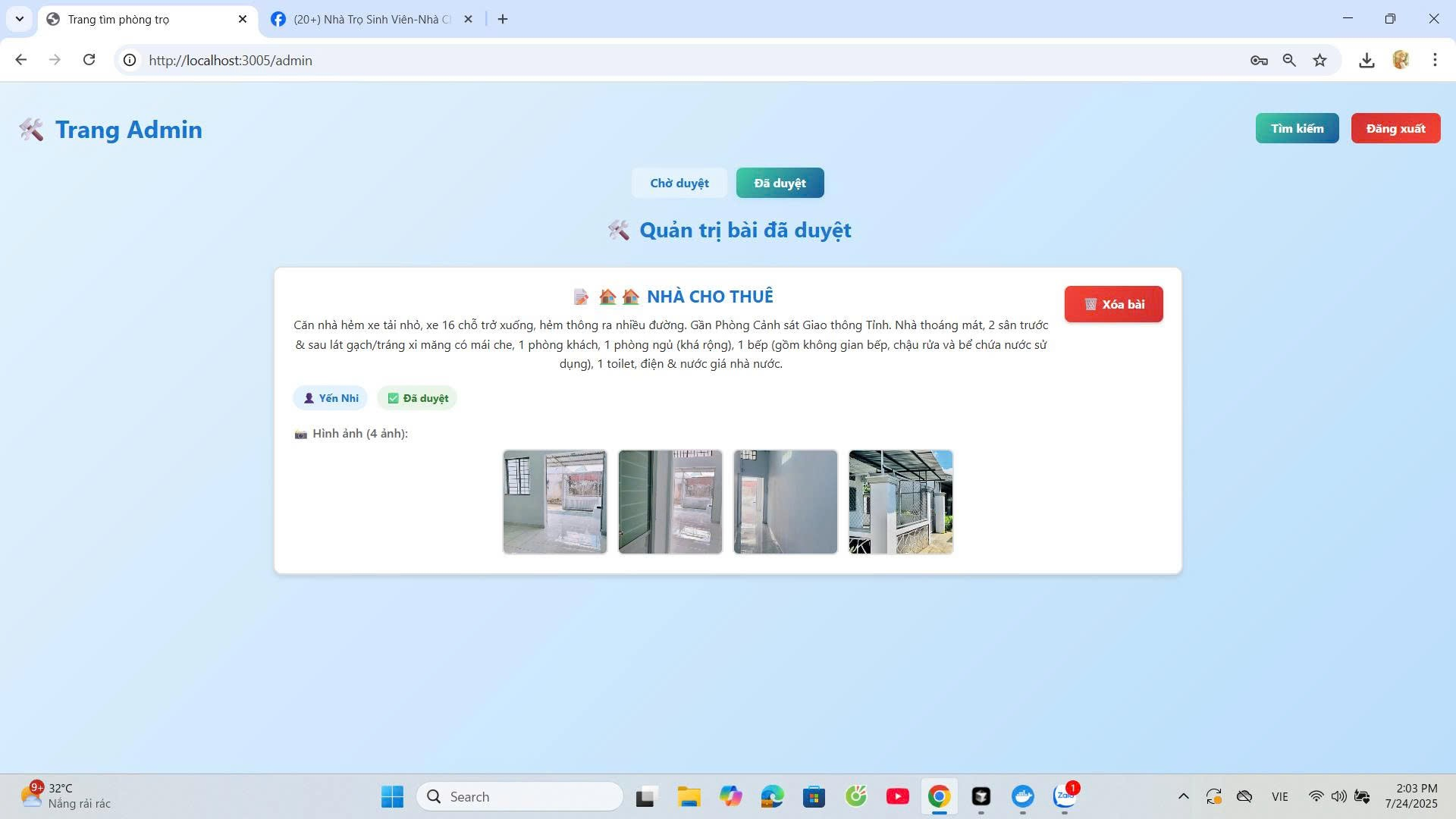
Hình 4.10 Giao diện tạo bài đăng

### Giao diện quản lý hồ sơ cá nhân



Hình 4.11 Giao diện hồ sơ cá nhân

### Giao diện trang admin



Hình 4.12 Giao diện trang admin duyệt bài đăng

# TRIỂN KHAI VÀ CÔNG NGHỆ SỬ DỤNG

## Danh sách các công nghệ đã sử dụng (ngôn ngữ lập trình, framework, thư viện).

Ngôn ngữ lập trình:

|  |  |
| --- | --- |
| **Ngôn ngữ** | **Mô tả** |
| **TypeScript** | Được sử dụng cho toàn bộ backend và frontend để tăng khả năng kiểm soát kiểu dữ liệu, phát hiện lỗi sớm và dễ bảo trì mã nguồn. |
| **JavaScript** | Sử dụng gián tiếp qua React/Next.js (ở frontend) sau khi biên dịch từ TypeScript. |

Bảng 10. Ngôn ngữ lập trình

Frameword: *React.js* và *Express.js* - xây dựng RESTful API cho các service như auth, user, post, gateway

Thư viện:

|  |  |
| --- | --- |
| **Thư viện** | **Mô tả & Chức năng chính** |
| bcryptjs | Mã hóa mật khẩu người dùng trước khi lưu vào DB |
| jsonwebtoken | Xác thực người dùng bằng JSON Web Token (JWT) |
| cors | Cho phép truy cập tài nguyên từ frontend (Cross-Origin) |
| dotenv | Quản lý biến môi trường từ file .env |
| multer | Xử lý upload file ảnh trong post-service |

Bảng 11. Các thư viện

## Quy trình CI/CD với GitHub Actions.

### *Giới thiệu về CI/CD*

CI/CD (Continuous Integration / Continuous Deployment) là quy trình tự động hóa trong phát triển phần mềm nhằm:

* CI (Tích hợp liên tục): Tự động kiểm tra, xây dựng (build) và kiểm thử (test) mỗi khi có thay đổi mã nguồn.
* CD (Triển khai liên tục): Tự động triển khai ứng dụng lên môi trường thực tế sau khi build/test thành công.

Mục tiêu chính của CI/CD là rút ngắn thời gian phát triển, giảm lỗi, tăng tính nhất quán, và tối ưu hóa quy trình DevOps.

### *GitHub Actions – Công cụ CI/CD tích hợp*

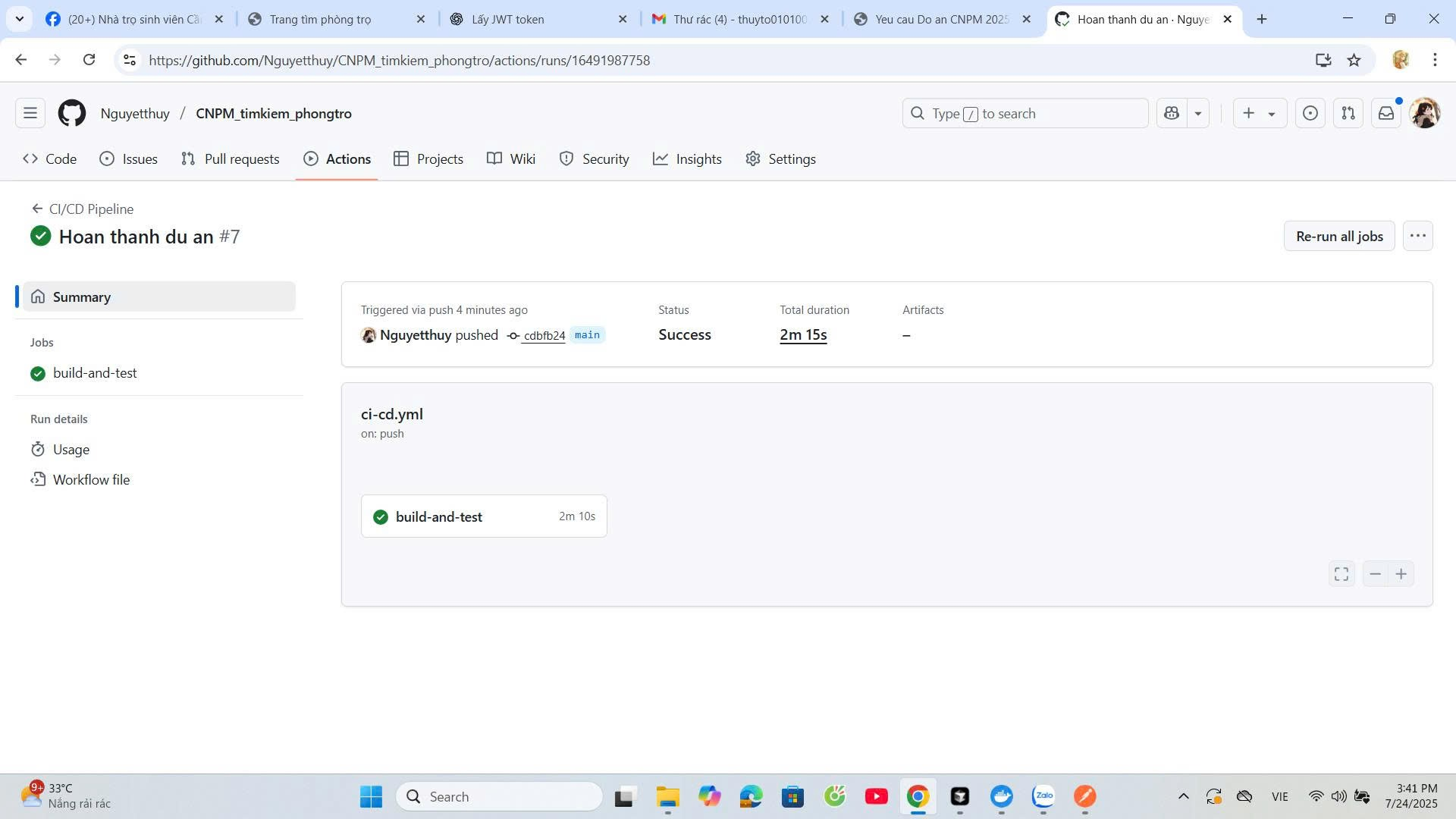
**GitHub Actions** là công cụ tích hợp sẵn trong GitHub, cho phép thiết lập các **workflow tự động** (các tác vụ CI/CD) khi có sự kiện xảy ra như push, pull request, hay merge. GitHub Actions sử dụng file cấu hình .yml để mô tả các bước thực thi.

### *Quy trình CI/CD trong hệ thống tìm kiếm phòng trọ*

Hệ thống sử dụng GitHub Actions để tự động hóa các bước sau:

|  |  |
| --- | --- |
| **Bước** | **Mô tả** |
| 1️ | Phát hiện thay đổi (trigger khi push hoặc pull request vào nhánh main hoặc dev) |
| 2️ | Cài đặt môi trường (Node.js, MongoDB container nếu cần) |
| 3️ | Cài đặt dependencies (npm install) cho frontend và backend |
| 4️ | Kiểm thử (unit test/API test) bằng các công cụ như Jest, Postman, hoặc Supertest |
| 5️ | Build ứng dụng (Next.js frontend, Node.js backend) |
| 6️ | (Tùy chọn) Triển khai tự động lên nền tảng như **Vercel**, **Heroku**, **Render**, hoặc **Docker Hub** |

Bảng 12. Mô tả quy trình CI/CD



Hình 5.1. CI/CD sau khi hoàn thành dự án

## Cấu hình Docker và quy trình triển khai ứng dụng.

### *Cấu hình Docker*

project-root/

│

├── docker-compose.yml

├── .env # (tuỳ chọn - chứa biến môi trường dùng chung)

│

├── auth-service/

│ ├── Dockerfile

│ ├── package.json

│ └── src/

│

├── user-service/

│ ├── Dockerfile

│ ├── package.json

│ └── src/

│

├── post-service/

│ ├── Dockerfile

│ ├── package.json

│ ├── uploads/ # Lưu ảnh bài viết

│ └── src/

│

├── notification-service/

│ ├── Dockerfile

│ ├── package.json

│ └── src/

│

├── gateway/

│ ├── Dockerfile

│ ├── package.json

│ └── src/

│

└── frontend/

├── Dockerfile

├── package.json

└── src/

### *Quy trình triển khai ứng dụng*

Bước 1: Chuẩn bị thư mục

* Mã nguồn đầy đủ cho từng service (auth, user, post, v.v.)
* File Dockerfile đúng cho mỗi thư mục
* File docker-compose.yml đặt ở thư mục Docker của dự án.

Bước 2: Tạo file .env (nếu dùng)

* Tạo file .env tại từng service với nội dung tương ứng,VD:

*MONGODB\_URI=mongodb://mongo:27017/findhouse*

*JWT\_SECRET=your-super-secret-jwt-key-here*

* Trong docker-compose.yml, thay dòng environment bằng:

*env\_file:*

*- .env*

Bước 3: Build và khởi động hệ thống

*# Di chuyển vào Docker*

*cd clean\_arch\_microservices/docker*

*# Build tất cả container*

*docker-compose build*

*# Khởi động dịch vụ (nền)*

*docker-compose up -d*

*# Theo dõi logs toàn bộ*

*docker-compose logs –f*

Bước 4: Truy cập hệ thống

|  |  |  |
| --- | --- | --- |
| **Thành phần** | **Đường dẫn truy cập** | **Ghi chú** |
| Frontend | [http://localhost:3005](http://localhost:3005/) | Giao diện người dùng |
| API Gateway | [http://localhost:3000](http://localhost:3000/) | Trung gian định tuyến |
| Auth Service | [http://localhost:3001](http://localhost:3001/) | Đăng ký / đăng nhập |
| Post Service | [http://localhost:3002](http://localhost:3002/) | Quản lý bài đăng |
| User Service | [http://localhost:3004](http://localhost:3004/) | Hồ sơ người dùng |
| MongoDB | localhost:27017 | DB nội bộ (NoSQL) |

Bảng 13. Thành phần và đường dẫn truy cập trong quy trình triển khai ứng dụng

Bước 5: Dừng và dọn hệ thống

*# Dừng các container*

*docker-compose down*

*# Dừng + xóa cả volume (data MongoDB)*

*docker-compose down -v*

### *Quy trình sử dụng Jira trong dự án*

1. *Tạo dự án và cấu hình bảng làm việc (Scrum Board)*

Nhóm khởi tạo một dự án mới trên Jira với loại dự án là **Scrum software project**.

Hệ thống tự động tạo một bảng làm việc với các cột mặc định: To Do – In Progress – Done, giúp nhóm theo dõi trạng thái các công việc.

1. *Xây dựng Product Backlog*

Các yêu cầu chức năng và nhiệm vụ được phân tích từ đề tài và chuyển hóa thành các issue (công việc), dạng:

* Task (nhiệm vụ cụ thể)
* Story (chức năng người dùng)
* Bug (lỗi)

Mỗi issue được gán:

* Người thực hiện (Assignee)
* Mức độ ưu tiên (Priority)
* Ước lượng thời gian (Story Points hoặc giờ)

1. *Lập kế hoạch Sprint (Sprint Planning)*

* Các thành viên lựa chọn các nhiệm vụ phù hợp đưa vào **Sprint** (ví dụ: 1 Sprint kéo dài 7 ngày).
* Tên Sprint, thời gian bắt đầu và kết thúc được thiết lập rõ ràng.
* Jira sẽ thống kê tổng số story points để đo lường khối lượng công việc cho Sprint.

1. *Theo dõi tiến độ thực hiện*

* Trong quá trình làm việc, các thành viên cập nhật trạng thái của từng task trên bảng:
* Từ To Do → In Progress → Done
* Các biểu đồ hỗ trợ theo dõi:
* Burndown Chart: Thể hiện lượng công việc còn lại theo thời gian.
* Sprint Report: Tổng hợp kết quả của từng Sprint.
* Velocity Chart: Thống kê năng suất của nhóm trong các Sprint liên tiếp.

1. *Đánh giá và cải tiến sau Sprint*

* Cuối mỗi Sprint, nhóm thực hiện:
* Sprint Review: Xem xét công việc đã hoàn thành.
* Sprint Retrospective: Phân tích điểm mạnh/yếu trong quy trình, từ đó cải tiến cho Sprint kế tiếp.

### *Lợi ích đạt được khi sử dụng Jira*

|  |  |
| --- | --- |
| **Tiêu chí** | **Lợi ích mang lại** |
| Quản lý công việc | Giao việc rõ ràng, dễ theo dõi tiến độ từng task |
| Minh bạch dự án | Tất cả thành viên nắm được trạng thái chung |
| Cải thiện hiệu suất | Dựa vào các biểu đồ Jira để tối ưu quy trình |
| Dễ đánh giá kết quả | Mỗi Sprint có báo cáo định lượng cụ thể |

Bảng 14. Lợi ích của Jira

## Phân công nhiệm vụ của từng thành viên trong nhóm.

| **STT** | **Mã công việc** | **Tên công việc** | **Sprint** | **Người phụ trách** |
| --- | --- | --- | --- | --- |
| 1 | FH-1 | Đăng ký | Sprint 1 (8-11 Jul) | Tô Đoàn nguyệt Thủy |
| 2 | FH-2 | Đăng nhập | Sprint 1 | Tô Đoàn nguyệt Thủy |
| 3 | FH-3 | Xác thực người dùng bằng JWT | Sprint 1 | Tô Đoàn nguyệt Thủy |
| 4 | FH-4 | Đăng bài | Sprint 1 | Phạm Thành Tín |
| 5 | FH-5 | Upload ảnh | Sprint 1 | Phạm Thành Tín |
| 6 | FH-6 | Lưu bài đăng | Sprint 1 | Phạm Thành Tín |
| 7 | FH-7 | Xem bài đăng | Sprint 1 | Phạm Thành Tín |
| 8 | FH-8 | Giao diện tìm kiếm | Sprint 2 (12-15 Jul) | Đinh Thị Yến Nhi |
| 9 | FH-9 | Tìm kiếm theo tiêu chí | Sprint 2 | Đinh Thị Yến Nhi |
| 10 | FH-10 | Hiển thị tìm kiếm | Sprint 2 | Đinh Thị Yến Nhi |
| 11 | FH-11 | Form liên hệ | Sprint 2 | Đinh Thị Yến Nhi |
| 12 | FH-12 | Xóa bài đăng | Sprint 3 (16-19 Jul) | Tô Đoàn nguyệt Thủy |
| 13 | FH-13 | Đánh giá sao | Sprint 3 | Tô Đoàn nguyệt Thủy |
| 14 | FH-14 | Xem đánh giá | Sprint 3 | Tô Đoàn nguyệt Thủy |
| 15 | FH-15 | Nhận thông báo bài đã được duyệt | Sprint 4 (20-23 Jul) | Phạm Thành Tín |
| 16 | FH-16 | Nhận thông báo liên hệ | Sprint 4 | Phạm Thành Tín |
| 17 | FH-17 | Hiển thị thông báo real-time | Sprint 4 | Phạm Thành Tín |

# KIỂM THỬ

## Chiến lược kiểm thử và công cụ sử dụng (Postman).

### ***Mục tiêu kiểm thử***

Đảm bảo các API hoạt động đúng theo thiết kế.

Đảm bảo việc xử lý thông tin người dùng (đăng ký, đăng nhập, truy vấn) chính xác và an toàn.

Kiểm tra xác thực bằng **JWT Token** có hoạt động tốt không.

### *Các chức năng kiểm thử chính*

1. *Đăng ký người dùng*

**Phương thức:** *POST*

**Endpoint:** */api/auth/register*

**Body:**

*{*

*"name": "Thành Tín",*

*"email": "pttin@gmail.com",*

*"password": "2222"*

*}*

Kiểm thử:

* Kiểm tra đăng ký thành công.
* Kiểm tra email đã tồn tại (kỳ vọng trả về lỗi).
* Kiểm tra định dạng email không hợp lệ.
* Kiểm tra mật khẩu trống hoặc quá ngắn.

1. *Đăng nhập người dùng*

**Phương thức:** *POST*

**Endpoint:** */api/auth/login*

Body:

*{*

*"email": "pttin@gmail.com",*

*"password": "2222"*

*}*

Kiểm thử:

* Đăng nhập thành công và trả về token, role.
* Đăng nhập sai mật khẩu hoặc email.
* Kiểm tra phản hồi khi gửi thiếu trường email hoặc password.

1. *Lấy thông tin người dùng*

**Phương thức:** *GET*

**Endpoint:** */api/auth/user/:id*

**Yêu cầu:** Bearer Token (từ đăng nhập)

Kiểm thử:

* Token hợp lệ -> trả về thông tin người dùng.
* Token sai hoặc thiếu -> trả về lỗi 401 Unauthorized.
* ID không tồn tại -> trả về lỗi 404 Not Found (nếu có xử lý).

### *Kỹ thuật kiểm thử áp dụng*

***Kiểm thử chức năng (Functional Testing)***: Đảm bảo các endpoint trả về đúng dữ liệu mong muốn.

***Kiểm thử bảo mật (Security Testing)***: Đảm bảo thông tin bảo mật như mật khẩu không bị trả về, token phải được xác thực.

***Kiểm thử biên (Boundary Testing)***: Kiểm tra độ dài tối đa/tối thiểu của trường đầu vào (name, password...).

***Kiểm thử API bằng dữ liệu sai (Negative Testing)***: Kiểm tra phản hồi khi dữ liệu đầu vào không hợp lệ.

### *Công cụ sử dụng: Postman*

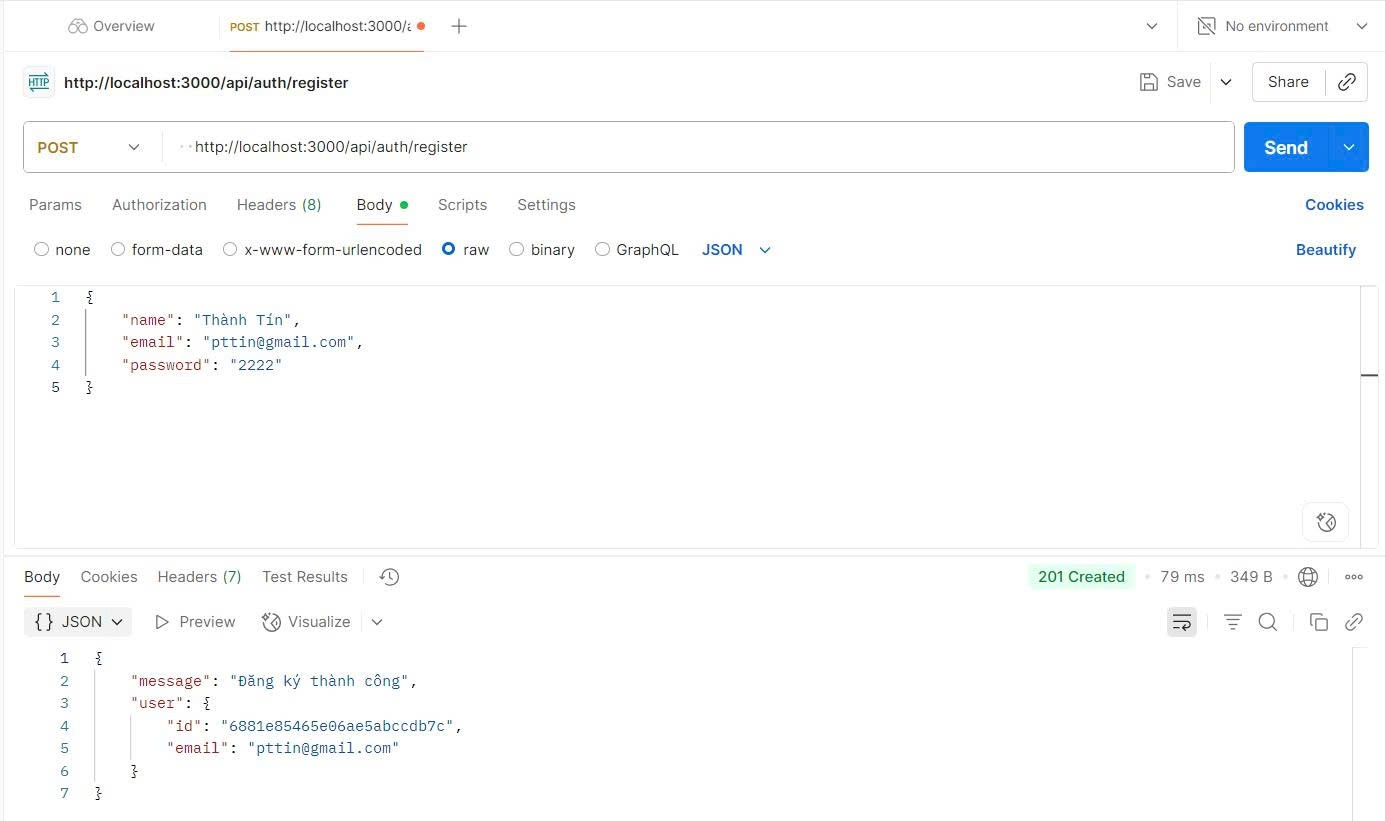
Vai trò:

* Gửi request đến API (GET, POST...).
* Kiểm tra phản hồi trả về (status code, response body, headers...).
* Gửi Token trong phần **Authorization**.
* Lưu trữ các collection kiểm thử để tái sử dụng.

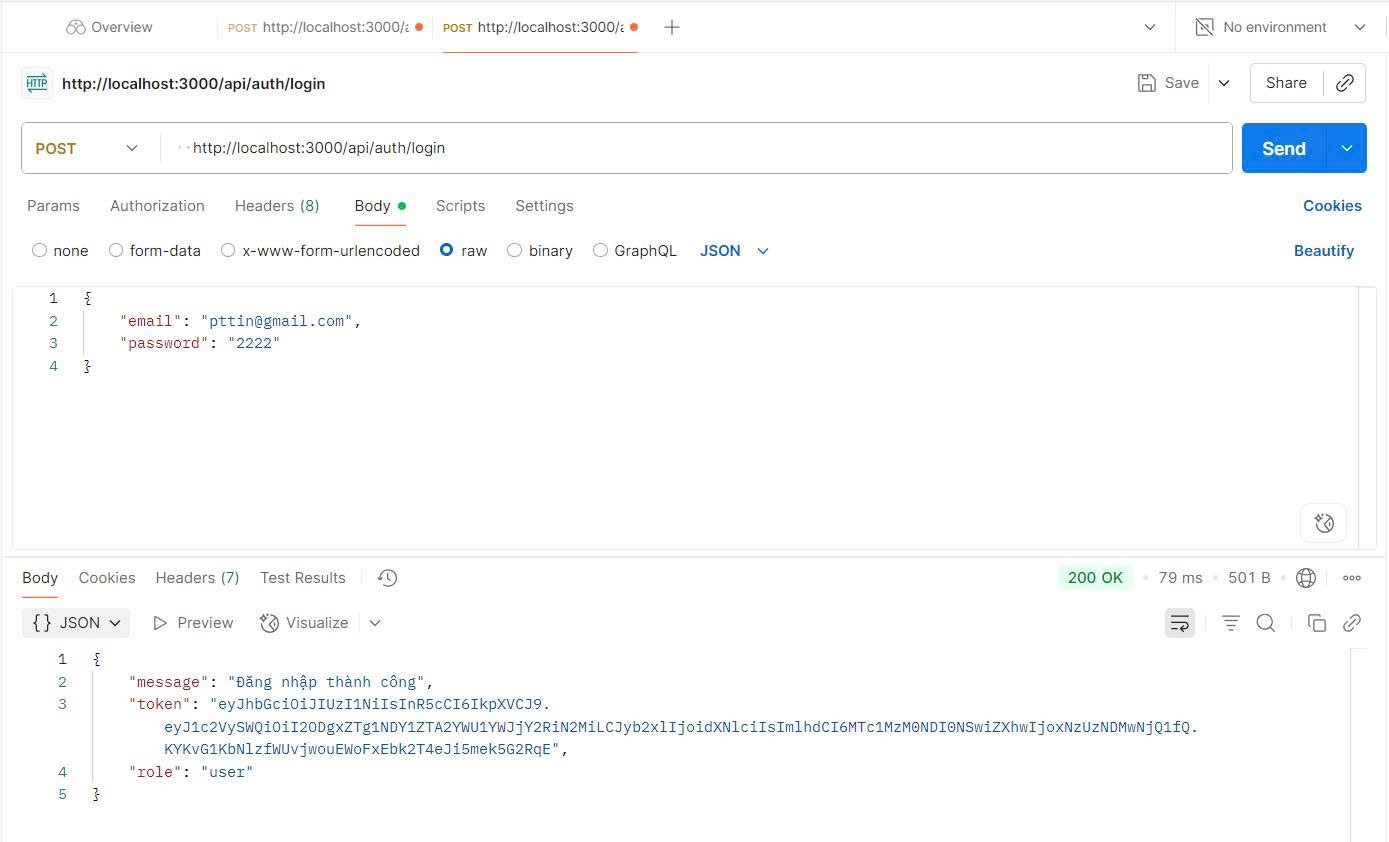
Các tiện ích hỗ trợ:

* **Tests tab**: Viết script kiểm tra tự động (kiểm tra code 200, 401...).
* **Pre-request Script**: Chuẩn bị token hoặc dữ liệu trước khi gửi.
* **Environment Variables**: Quản lý token, ID người dùng, baseURL...

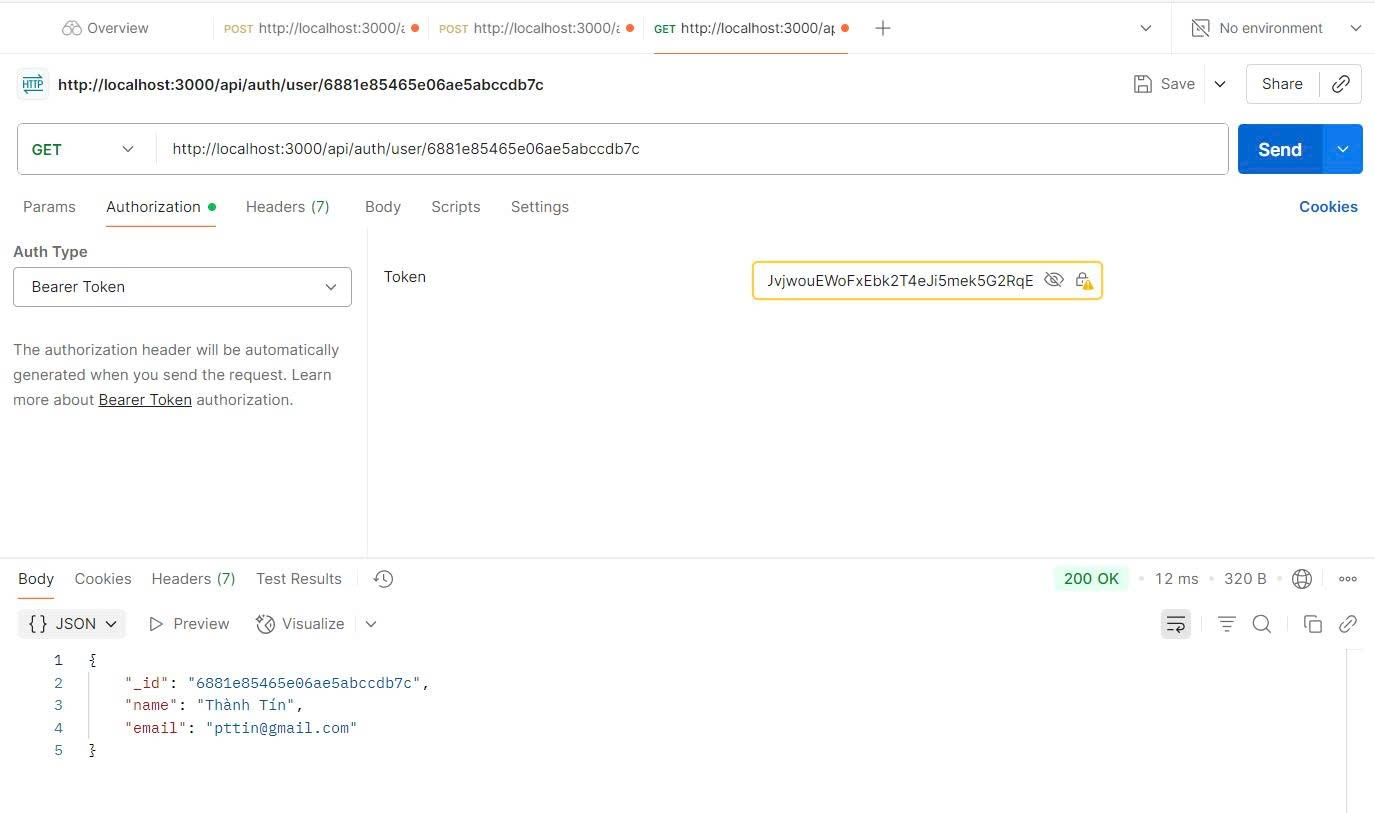
## Kết quả kiểm thử API (có thể đính kèm ảnh chụp màn hình từ Postman).



Hình 6.1. Postman -- đăng kí tài khoản



Hình 6.2. Đăng nhập trả về token



Hình 6.3. Dùng token để trả về thông tin tài khoản

# ĐÁNH GIÁ VÀ KẾT LUẬN

## Những khó khăn gặp phải trong quá trình thực hiện.

*Thời gian hạn chế*: Trong quá trình phát triển dự án, nhóm gặp khó khăn về mặt thời gian do phải phân bổ cho nhiều môn học và hoạt động khác nhau. Điều này ảnh hưởng đến tiến độ, khiến một số tính năng chưa thể hoàn thiện như mong muốn. Việc quản lý thời gian và phân chia công việc là thách thức lớn, đặc biệt khi làm việc nhóm và phối hợp nhiều thành phần hệ thống.

*Kiến thức về các công nghệ còn hạn chế, chưa khai thác triệt để*: Dự án áp dụng nhiều công nghệ mới như Docker, Microservices, MongoDB, và các công cụ CI/CD. Tuy nhiên, nhóm vẫn còn hạn chế về kinh nghiệm thực tiễn, đặc biệt trong việc triển khai đồng bộ hệ thống, xử lý lỗi, tối ưu bảo mật và hiệu suất. Việc tìm hiểu và áp dụng các công nghệ này tiêu tốn nhiều thời gian, dẫn đến quá tải trong một số giai đoạn.

*Trang web còn đơn giản, chức năng chưa đa dạng*: Do giới hạn về thời gian và nguồn lực, giao diện người dùng và trải nghiệm người dùng (UI/UX) chưa được đầu tư đúng mức. Các tính năng cơ bản đã được triển khai, nhưng hệ thống vẫn còn thiếu một số chức năng nâng cao như bộ lọc thông minh, đánh giá người dùng, gợi ý phòng trọ, hoặc hệ thống quản lý báo cáo vi phạm. Điều này khiến sản phẩm chưa thực sự hoàn chỉnh so với kỳ vọng ban đầu.

## Bài học rút ra và đề xuất cải thiện trong tương lai.

*Sắp xếp thời gian hợp lý*: Qua quá trình thực hiện dự án, nhóm nhận thấy tầm quan trọng của việc lập kế hoạch và quản lý thời gian hiệu quả. Việc không phân bổ thời gian hợp lý dẫn đến tình trạng dồn việc vào giai đoạn cuối, gây áp lực lớn và ảnh hưởng đến chất lượng sản phẩm. Trong các dự án sau, nhóm sẽ áp dụng các công cụ quản lý tiến độ như Trello hoặc GitHub Projects để theo dõi công việc, đồng thời chia nhỏ đầu việc theo từng tuần nhằm đảm bảo tiến độ và tránh quá tải.

*Phân công nhiệm vụ cụ thể hơn*: Một số giai đoạn trong dự án bị trùng lặp công việc và thiếu người phụ trách, gây mất thời gian điều phối và xử lý lỗi. Nhóm nhận ra rằng việc phân công cụ thể từng phần việc cho từng thành viên từ đầu, kèm theo mô tả rõ ràng về trách nhiệm, là yếu tố then chốt giúp dự án vận hành trôi chảy. Việc này không chỉ tăng hiệu suất làm việc mà còn giúp thành viên chủ động hơn trong quá trình thực hiện.

*Tìm hiểu kỹ hơn về các công nghệ để tránh lỗi*: Trong quá trình triển khai, nhóm gặp một số lỗi kỹ thuật do chưa nắm vững công nghệ sử dụng như Docker, MongoDB, hay kiến trúc Microservices. Việc xử lý sự cố tốn khá nhiều thời gian, ảnh hưởng đến tiến độ chung. Bài học rút ra là cần dành thời gian tìm hiểu kỹ trước khi bắt tay vào lập trình, đồng thời ưu tiên thực hành nhiều hơn để hiểu sâu bản chất. Trong các dự án tương lai, nhóm sẽ chú trọng hơn vào bước nghiên cứu công nghệ và xây dựng tài liệu nội bộ nhằm chuẩn hóa quy trình phát triển.

# PHỤ LỤC

## Hướng dẫn cài đặt và chạy ứng dụng.

### *Yêu cầu hệ thống*

Trước khi tiến hành cài đặt, đảm bảo máy tính có các công cụ sau:

|  |  |
| --- | --- |
| **Thành phần** | **Phiên bản khuyến nghị** |
| Git | >= 2.30.0 |
| Docker | >= 20.10 |
| Docker Compose | >= 1.29 |
| Node.js (tùy chọn) | >= 18.x |
| Trình duyệt | Chrome, Edge, Firefox |

Bảng 15. Các công cụ cần có để chạy ứng dụng

### *Các bước cài đặt và chạy hệ thống*

Bước 1: Clone mã nguồn từ GitHub

*git clone https://github.com/your-username/CNPM\_timkiem\_phongtro.git*

*cd CNPM\_timkiem\_phongtro*

Bước 2: Kiểm tra file cấu hình .env

* Tạo các file .env dựa trên env.example (nếu có), ví dụ cho user-service:

*cp user-service/env.example user-service/.env*

* Điền các biến môi trường như PORT, MONGODB\_URI, JWT\_SECRET,...

Bước 3: Khởi động hệ thống bằng Docker Compose

*docker-compose up –build*

* Build và chạy toàn bộ các service: auth-service, user-service, post-service, gateway, frontend, mongodb.
* Tự động khôi phục dữ liệu mẫu từ thư mục docker/dump/findhouse/ nếu cấu hình khớp.
* Sau khi chạy thành công, frontend có thể truy cập tại <http://localhost:3000>

### *Kiểm thử API (tùy chọn)*

Sử dụng công cụ như **Postman** để kiểm tra các API chính:

|  |  |  |
| --- | --- | --- |
| **API** | **Phương thức** | **URL ví dụ** |
| Đăng ký người dùng | POST | /api/auth/register |
| Đăng nhập | POST | /api/auth/login |
| Tạo bài đăng | POST | /api/posts/create |
| Tìm kiếm phòng trọ | GET | /api/posts/search?q=... |
| Cập nhật thông tin | PUT | /api/users/update |

Bảng 16. Kiểm thử API

### *Cách dừng và khởi động lại*

Dừng toàn bộ container:

*docker-compose down*

Chạy lại sau khi sửa mã nguồn:

*docker-compose up --build*

### *Một số lỗi thường gặp*

|  |  |
| --- | --- |
| **Vấn đề** | **Cách xử lý** |
| Cổng 3000 hoặc 5000 bị chiếm dụng | Dừng tiến trình khác hoặc sửa file .env |
| MongoDB không chạy | Kiểm tra docker-compose.yml và logs container |
| Không khôi phục được dữ liệu | Kiểm tra thư mục docker/dump/findhouse/ |

Bảng 17. Một số lỗi thường gặp khi chạy ứng dụng

## Liên kết GitHub repository và link demo (nếu có).

Liên kết GitHub: https://github.com/Nguyetthuy/CNPM\_timkiem\_phongtro.git

**DANH MỤC TÀI LIỆU THAM KHẢO**

[1] R. T. Fielding, Architectural Styles and the Design of Network-based Software Architectures, Ph.D. dissertation, University of California, Irvine, 2000.

[2] Postman, Inc., Postman API Platform Documentation, 2023. [Online]. Available: https://learning.postman.com/

[3] Docker, Inc., Docker Documentation, 2023. [Online]. Available: https://docs.docker.com/

[4] GitHub, Inc., GitHub Actions Documentation, 2023. [Online]. Available: https://docs.github.com/en/actions

[5] MongoDB, Inc., MongoDB Manual, 2023. [Online]. Available: https://www.mongodb.com/docs/manual/

[6] Atlassian, Jira Software Documentation, 2023. [Online]. Available: https://support.atlassian.com/jira-software-cloud/

[7] Node.js Foundation, Node.js Documentation, 2023. [Online]. Available: https://nodejs.org/en/docs/

[8] S. Newman, Building Microservices: Designing Fine-Grained Systems, 2nd ed., O'Reilly Media, 2019.