BỘ THÔNG TIN VÀ TRUYỀN THÔNG

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

BÁO CÁO MÔN CÁC HỆ THỐNG PHÂN TÁN

HỆ THỐNG CHIA SẺ BẢNG TRẮNG KANVA

Người hướng dẫn:	**TS. Kim Ngọc Bách**
Học viên:	**Phạm Minh Dương – B24CHHT064**
	Ma Công Thành – NCS2024.20
	Vũ Quỳnh Anh – B24CHHT056
Nhóm:	**8**
Lớp:	**M24CQHT02-B**

Hà Nội, tháng 8/2025

Thành viên	Công việc
Ma Công Thành	Back-end
Phạm Minh Dương	Front-end
Vũ Quỳnh Anh	Tester, thiết kế DB, viết báo cáo, làm slide

MỤC LỤC

DANH MỤC HÌNH ẢNH

TÓM TẮT (ABSTRACT)

Trong thời đại làm việc và học tập từ xa ngày càng phổ biến, nhu cầu về các công cụ cộng tác trực quan và hiệu quả ngày càng tăng cao. Đề tài này trình bày quá trình thiết kế và phát triển Kanva, một hệ thống whiteboard cộng tác thời gian thực dựa trên kiến trúc microservices, lấy cảm hứng từ nền tảng Canva. Hệ thống cho phép nhiều người dùng cùng vẽ, viết, thêm hình ảnh, và tương tác trực tiếp trên một bảng trắng chia sẻ, với các thay đổi được đồng bộ gần như tức thời giữa các thiết bị. Kiến trúc microservice giúp tăng khả năng mở rộng, phân tách trách nhiệm rõ ràng giữa các thành phần như xử lý người dùng, lưu trữ phiên làm việc, truyền thông real-time và quản lý tài nguyên. Giao tiếp giữa các dịch vụ sử dụng giao thức nhẹ như REST và WebSocket, trong khi cơ sở dữ liệu NoSQL hỗ trợ lưu trữ các thay đổi dưới dạng sự kiện để dễ dàng phát lại hoặc khôi phục trạng thái. Hệ thống hướng đến tính mở rộng, độ trễ thấp và khả năng phục hồi cao, phù hợp để tích hợp trong các nền tảng giáo dục, doanh nghiệp hoặc làm việc nhóm sáng tạo.

I. Mở đầu

Trong kỷ nguyên số hiện nay, sự cộng tác trực tuyến đã trở thành một phần không thể thiếu trong môi trường học tập, làm việc và sáng tạo. Các công cụ như Google Docs, Miro hay Canva đã chứng minh tầm quan trọng của việc cho phép nhiều người dùng cùng thao tác trên một nền tảng chia sẻ theo thời gian thực. Đặc biệt, mô hình whiteboard số – bảng trắng ảo – nổi lên như một giải pháp linh hoạt cho việc brainstorming, giảng dạy, vẽ sơ đồ, lên kế hoạch hoặc làm việc nhóm trong các bối cảnh từ giáo dục đến doanh nghiệp.

Tuy nhiên, để xây dựng một hệ thống whiteboard cộng tác hiệu quả không đơn giản. Nó đòi hỏi khả năng xử lý đồng bộ nhiều người dùng cùng tương tác, độ trễ cực thấp, lưu trữ trạng thái liên tục và đảm bảo ổn định khi mở rộng quy mô. Để đáp ứng các yêu cầu này, nhóm chúng tôi lựa chọn tiếp cận bài toán bằng kiến trúc microservices – một mô hình giúp phân chia hệ thống thành các thành phần độc lập, có thể triển khai và mở rộng riêng biệt.

Hệ thống được thiết kế để cho phép nhiều người dùng thao tác trên cùng một bảng trắng, với các hành động như vẽ, xóa, di chuyển, chèn hình ảnh hoặc văn bản đều được cập nhật tức thời trên giao diện của tất cả các người dùng đang tham gia phiên làm việc. Các thành phần chính của hệ thống bao gồm: dịch vụ quản lý người dùng, dịch vụ xử lý thiết kế, và truyền thông real-time sử dụng WebSocket. Dữ liệu được lưu trữ trong cơ sở dữ liệu MongoDB, với hai bảng chính: Users và Designs, trong đó canvas được lưu dưới dạng JSON để dễ dàng biểu diễn các thao tác vẽ.

Với định hướng xây dựng một hệ thống linh hoạt, có khả năng mở rộng và thích ứng với nhiều loại hình ứng dụng khác nhau, đề tài này không chỉ dừng lại ở việc tạo ra một công cụ cộng tác, mà còn là bài học thực tiễn trong việc ứng dụng kiến trúc microservices vào các bài toán tương tác thời gian thực.

II. Cơ sở lý thuyết và công nghệ áp dụng

Hệ thống được xây dựng dựa trên các lý thuyết và công nghệ sau:

- **Kiến trúc Microservices**: Hệ thống được chia thành các dịch vụ riêng biệt như dịch vụ người dùng và dịch vụ thiết kế. Mỗi service đảm nhiệm một chức năng rõ ràng và có thể phát triển, mở rộng độc lập.

- **Docker:** Các microservice được đóng gói bằng Docker, giúp việc triển khai (deployment) và quản lý trở nên nhất quán và dễ dàng trên mọi môi trường. Docker đảm bảo mỗi dịch vụ hoạt động trong một môi trường độc lập, cô lập, tăng tính ổn định của hệ thống.
- **NestJS:** Hệ thống sử dụng NestJS để xây dựng các microservice ở backend. NestJS với kiến trúc dựa trên module và hỗ trợ mạnh mẽ cho TypeScript, là lựa chọn lý tưởng để tạo ra các RESTful API và WebSocket gateways một cách có cấu trúc, dễ bảo trì và mở rộng.
- **MongoDB**: Là hệ quản trị cơ sở dữ liệu NoSQL, rất phù hợp để lưu trữ dữ liệu phi cấu trúc như nội dung của canvas hay hình ảnh đại diện. Cấu trúc JSON giúp lưu dữ liệu canvas một cách linh hoạt.
- **Giao tiếp thời gian thực (WebSocket):** Giao tiếp giữa client và server sử dụng WebSocket, được tích hợp và quản lý hiệu quả bởi NestJS. Kênh giao tiếp hai chiều này giúp truyền tải hành động người dùng ngay lập tức giữa các client trong cùng một phiên whiteboard.
- **Frontend**: Giao diện bảng vẽ được xây dựng bằng Vuejs, sử dụng kết hợp với Fabricjs và HTML5. Vue 3 cung cấp một library mạnh mẽ, hiệu suất cao với khả năng xử lý trạng thái hiệu quả, giúp dễ dàng quản lý các hành động của người dùng và hiển thị bản vẽ một cách đồng bộ trên các client.

III. Thiết kế hệ thống

1. Lý do lựa chọn kiến trúc Microservices

Sau quá trình tìm hiểu lý thuyết và thực hành trong quá trình xây dựng hệ thống whiteboard cộng tác thời gian thực, nhóm chúng tôi quyết định lựa chọn kiến trúc Microservices để triển khai hệ thống phân tán này. Việc áp dụng microservices mang lại nhiều lợi ích như khả năng mở rộng tốt, dễ bảo trì, linh hoạt trong phát triển và triển khai.

Kiến trúc microservices trong một hệ thống phân tán là phương pháp phân chia ứng dụng lớn thành các dịch vụ nhỏ, độc lập và giao tiếp với nhau thông qua mạng (thường là HTTP hoặc WebSocket). Mỗi dịch vụ đảm nhiệm một chức năng riêng biệt, giúp cho việc phát triển, triển khai và vận hành hệ thống trở nên hiệu quả và dễ kiểm soát hơn. Cụ thể, những lý do chính khiến nhóm chọn kiến trúc này bao gồm:

a. Tính linh hoạt và dễ mở rộng: Microservices cho phép tách hệ thống thành các dịch vụ nhỏ độc lập như: quản lý người dùng, quản lý thiết kế, và xử lý real-time. Nhờ đó, các dịch vụ có thể được triển khai và mở rộng riêng biệt mà không ảnh hưởng đến toàn bộ hệ thống.

b. Phát triển và triển khai độc lập: Mỗi dịch vụ có thể được phát triển và kiểm thử độc lập bởi từng thành viên trong nhóm, không bị phụ thuộc lẫn nhau. Điều này giúp rút ngắn thời gian phát triển, dễ dàng kiểm soát lỗi và tối ưu quá trình triển khai.

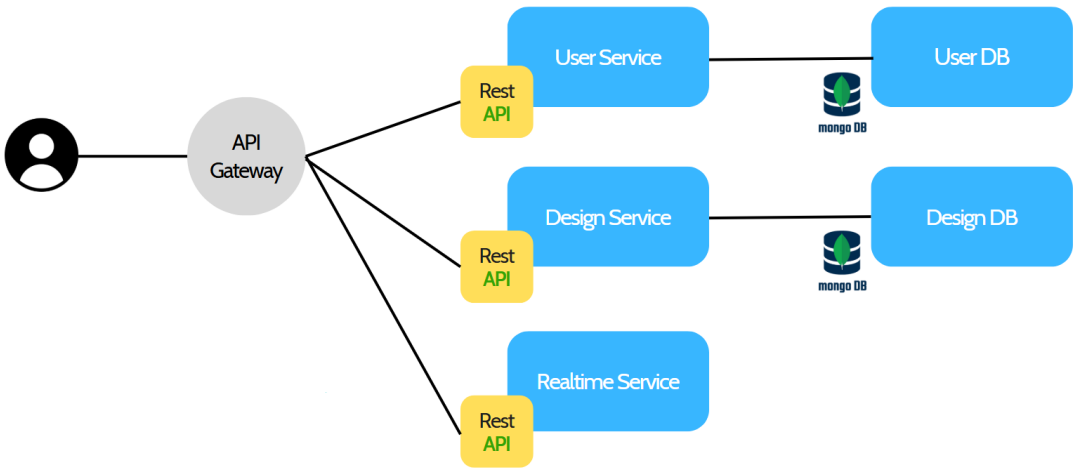
c. Khả năng chịu lỗi cao: Do các dịch vụ hoạt động độc lập, nên nếu một dịch vụ gặp sự cố (ví dụ như dịch vụ lưu thiết kế tạm thời), các dịch vụ còn lại vẫn có thể tiếp tục hoạt động bình thường. Điều này giúp giảm thiểu rủi ro gián đoạn toàn hệ thống.

2. Ứng dụng kiến trúc Microservices trong hệ thống Kanva

2.1. Kiến trúc tổng thể

Trong dự án này, nhóm áp dụng kiến trúc microservices bằng cách phân chia hệ thống thành các dịch vụ chính, mỗi dịch vụ đảm nhiệm một chức năng riêng biệt. Cụ thể, hệ thống bao gồm ba dịch vụ chính: User Service, Design Service, và Realtime Service.

- User Service: Quản lý thông tin người dùng, đăng ký, đăng nhập.
- Design Service: Lưu trữ và xử lý dữ liệu thiết kế (canvas).
- Realtime Service: Đồng bộ thao tác vẽ giữa nhiều người dùng trong thời gian thực, sử dụng WebSocket hoặc Socket.IO.

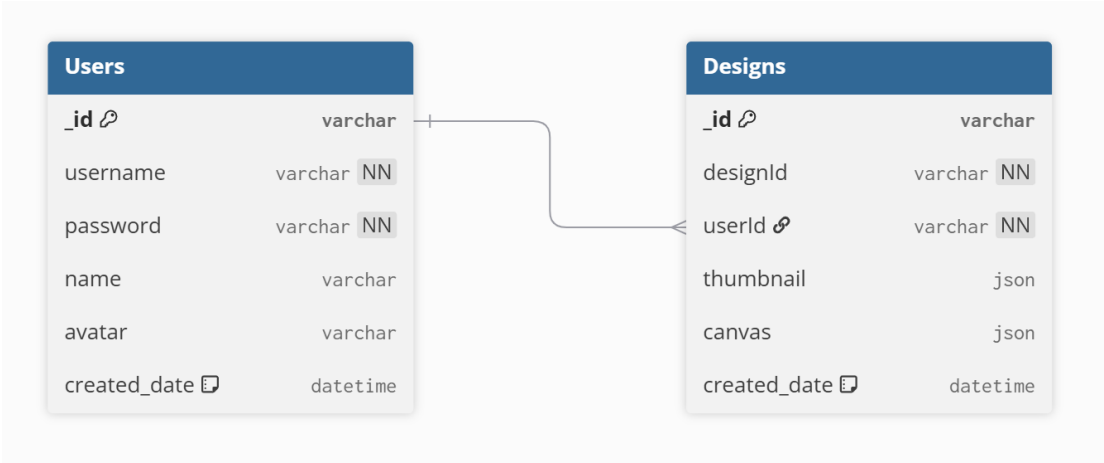


Hình 1: Services trong Microservices

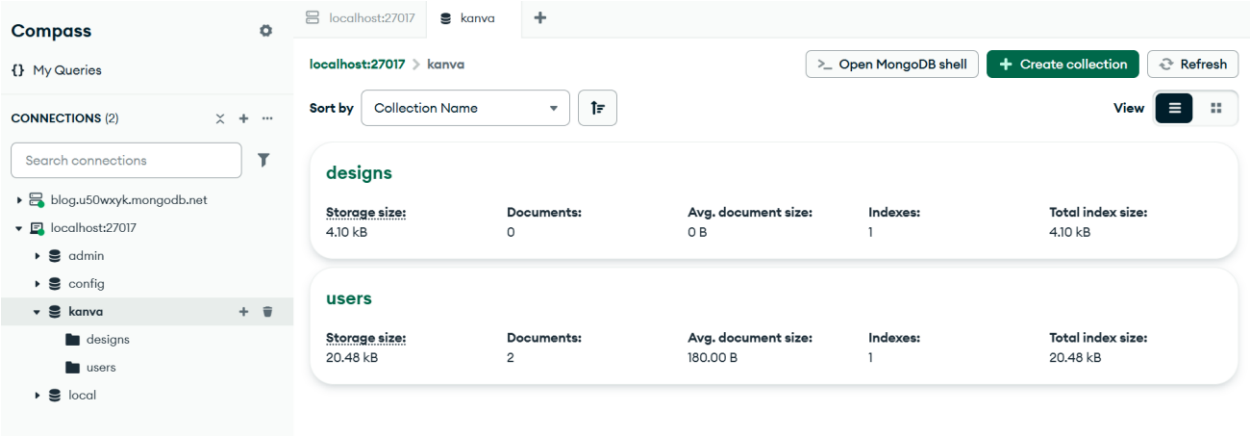
2.2 Thiết kế cơ sở dữ liệu

Ngoài ra, microservices còn được áp dụng ở tầng dữ liệu. Các dịch vụ sử dụng MongoDB – một cơ sở dữ liệu NoSQL phù hợp với việc lưu trữ các cấu trúc dữ liệu linh hoạt như canvas hoặc thumbnail. Hệ thống được tổ chức với hai bảng chính là Users và Designs, trong đó bảng Users lưu trữ thông tin người dùng, còn bảng Designs lưu dữ liệu thiết kế, canvas và thông tin liên kết với người tạo.

Một điểm đặc biệt trong hệ thống là: dữ liệu canvas chỉ được ghi vào cơ sở dữ liệu khi tất cả người dùng trong phiên whiteboard cùng thoát ra. Cách tiếp cận này giúp giảm thiểu số lần ghi dữ liệu không cần thiết trong quá trình tương tác liên tục, đồng thời tối ưu hiệu năng hệ thống. Việc phân tách chức năng và dữ liệu như vậy không chỉ đảm bảo khả năng mở rộng và bảo trì, mà còn tạo tiền đề để hệ thống phát triển thêm nhiều tính năng trong tương lai như undo/redo thao tác, phân quyền truy cập thiết kế.



Hình 2: ER diagram



Hình 3: MongoDB database

Quá trình lưu trữ dữ liệu diễn ra như sau:

- **Lưu trữ dữ liệu tạm thời (In-memory):** Trong suốt quá trình tương tác thời gian thực, dữ liệu bảng trắng được giữ trong bộ nhớ (in-memory) của Realtime Service. Mỗi phiên làm việc được gán một trạng thái tạm thời, bao gồm dữ liệu canvas hiện tại và danh sách các người dùng đang tham gia. Phương pháp này giúp giảm thiểu đáng kể số lượng thao tác ghi vào cơ sở dữ liệu, tránh tình trạng quá tải khi nhiều người dùng cùng thao tác liên tục.
- **Lưu trữ dữ liệu vĩnh viễn (Persistence):** Dữ liệu canvas chỉ được ghi vào Design Service khi tất cả người dùng trong một phiên làm việc đã thoát ra. Khi người dùng cuối cùng rời khỏi phiên,
- Realtime Service sẽ kích hoạt một sự kiện để gửi dữ liệu canvas cuối cùng đến Design Service.
- Design Service sẽ nhận dữ liệu này và cập nhật vào trường canvas của bảng Designs trong MongoDB. Cách tiếp cận này giúp tối ưu hiệu năng hệ thống và đảm bảo tính nhất quán của dữ liệu đã lưu.

2.3. Cơ chế giao tiếp và luồng dữ liệu giữa các dịch vụ

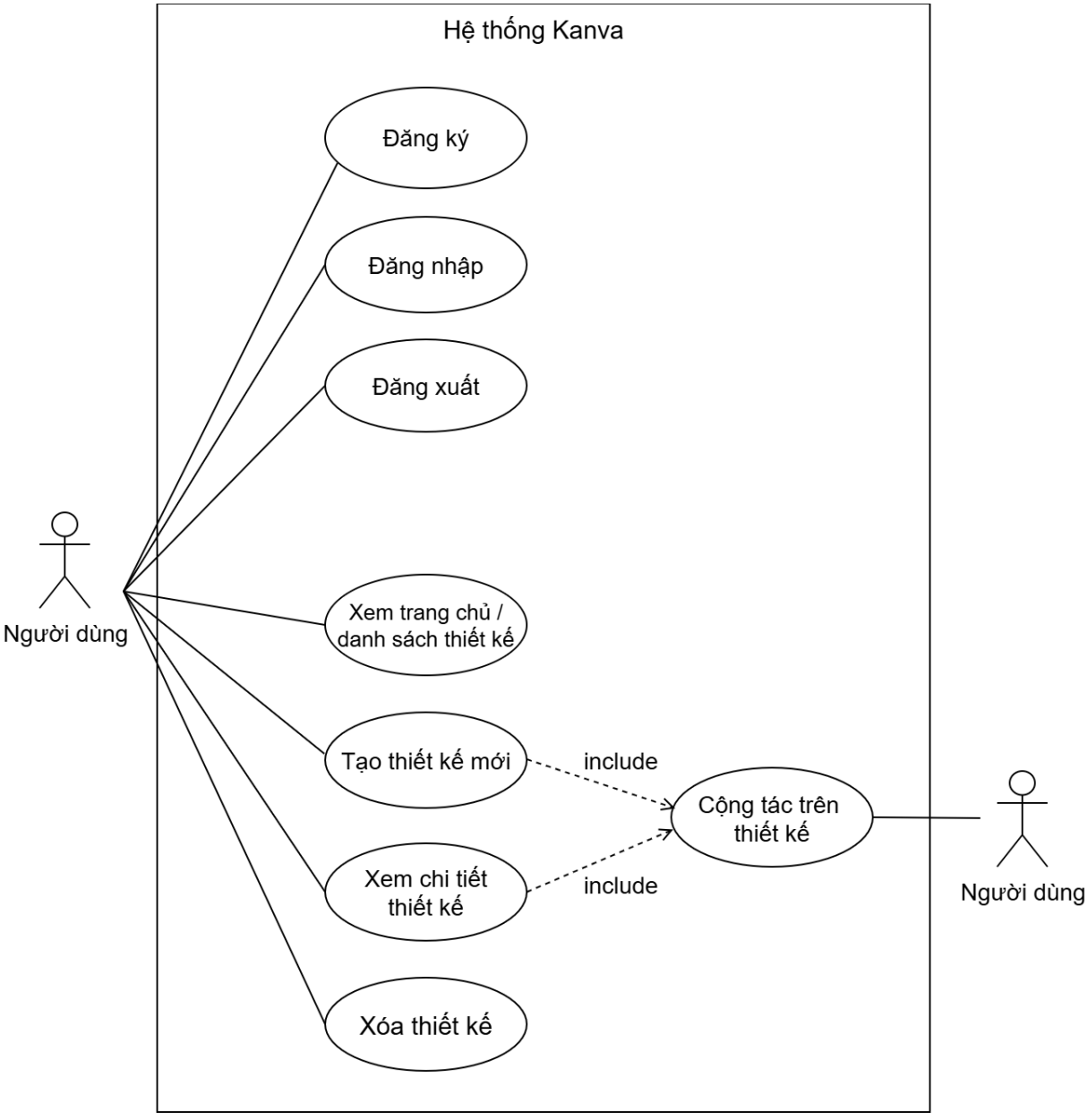
Trong kiến trúc microservices của Kanva, các dịch vụ không hoạt động độc lập mà giao tiếp với nhau để hoàn thành các tác vụ phức tạp. Luồng dữ liệu chính của hệ thống được tổ chức như sau:

- **RESTful API cho các tác vụ quản lý:** Các dịch vụ User Service và Design Service sử dụng **RESTful API** để xử lý các yêu cầu đồng bộ (synchronous) từ phía client. Ví dụ, khi người dùng đăng ký hoặc đăng nhập, frontend sẽ gửi yêu cầu HTTP POST đến User Service. Tương tự, các yêu cầu tạo, mở hoặc xóa thiết kế cũng được gửi qua API này đến Design Service.

- **WebSocket cho giao tiếp thời gian thực:** Trọng tâm của hệ thống Kanva là khả năng cộng tác theo thời gian thực, được thực hiện thông qua **Realtime Service** sử dụng giao thức **WebSocket**. Khi một người dùng tham gia vào một phiên làm việc, client sẽ thiết lập một kết nối WebSocket duy nhất đến Realtime Service. Mọi thao tác của người dùng trên bảng trắng (vẽ, thêm đối tượng) sẽ được gửi qua kênh WebSocket này.

- Realtime Service sau đó sẽ broadcast (phát sóng) các thay đổi này đến tất cả các client khác đang kết nối đến cùng một phiên. Cơ chế này đảm bảo độ trễ thấp và đồng bộ hóa tức thời giữa các người dùng.

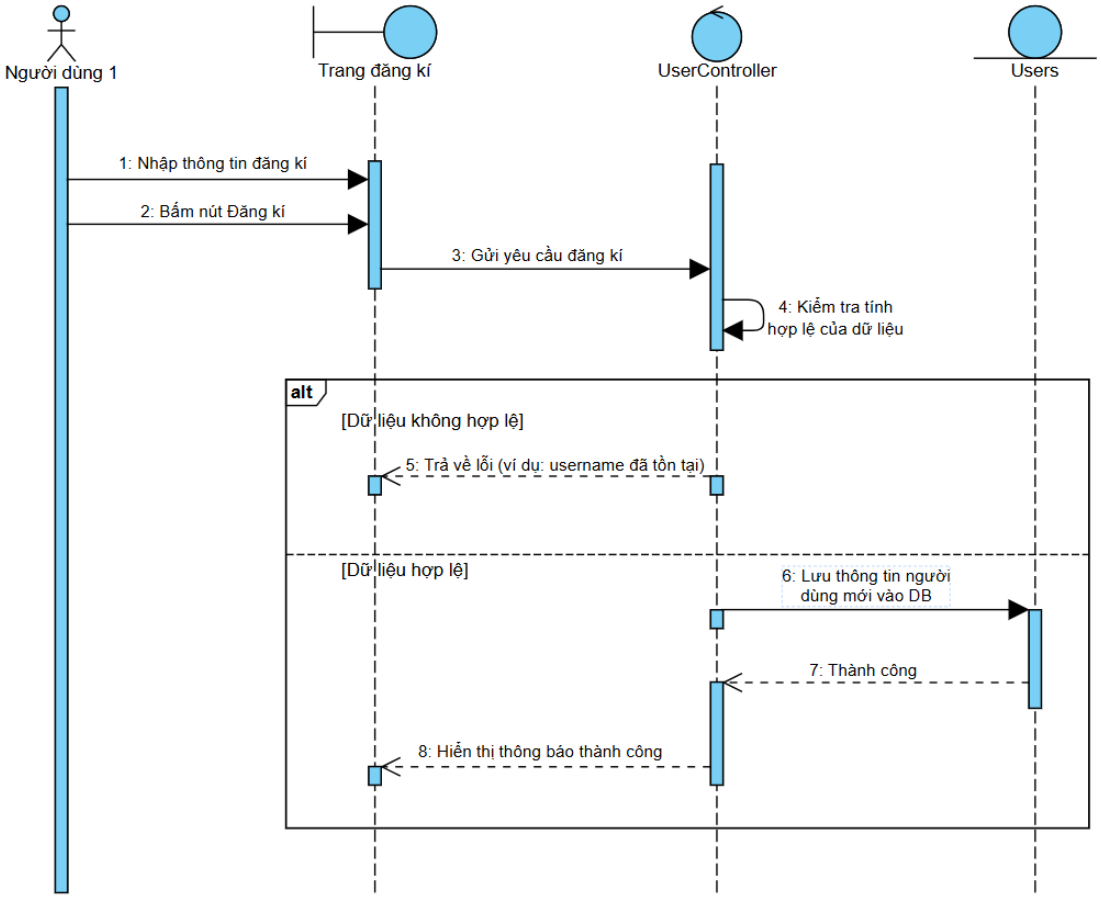
IV. Sơ đồ Use Case



Hình 4: Sơ đồ Use Case của Kanva

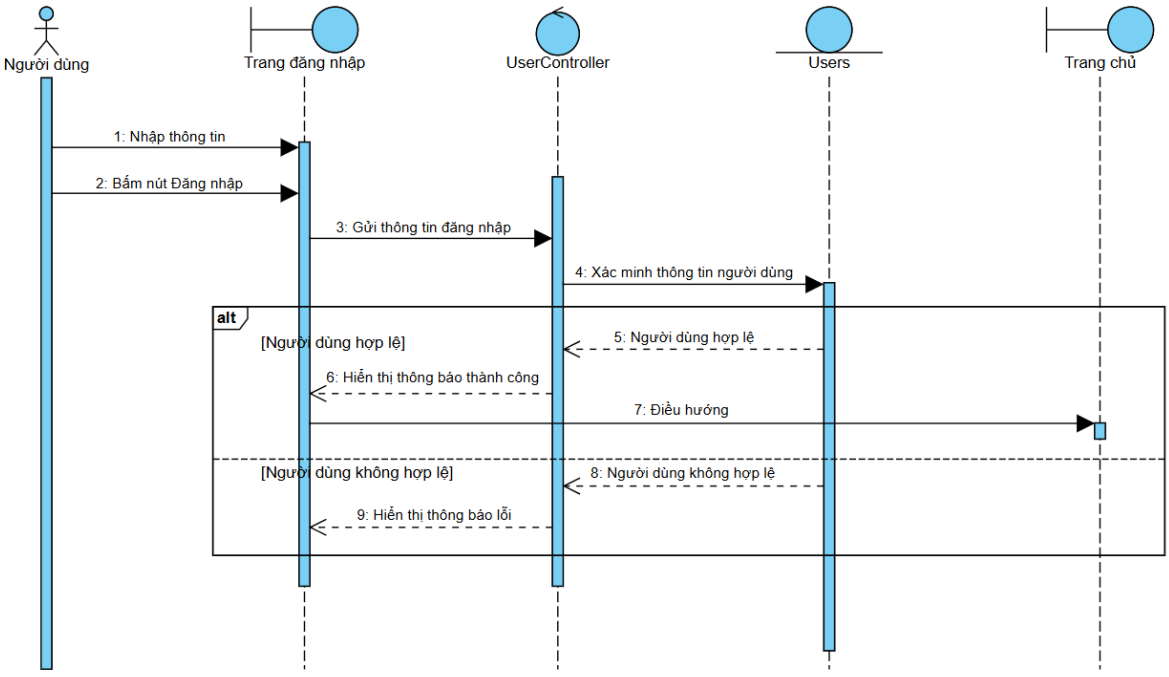
V. Sơ đồ tuần tự

1. Đăng kí



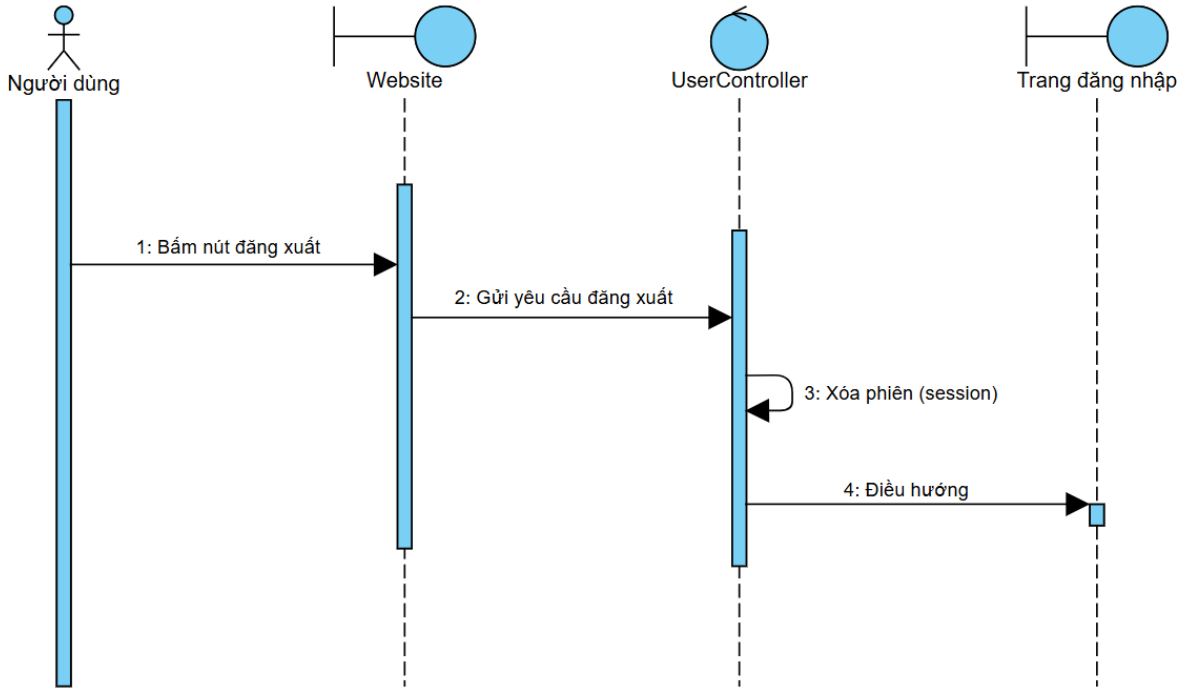
Hình 5: Sơ đồ tuần tự của Đăng kí

2. Đăng nhập



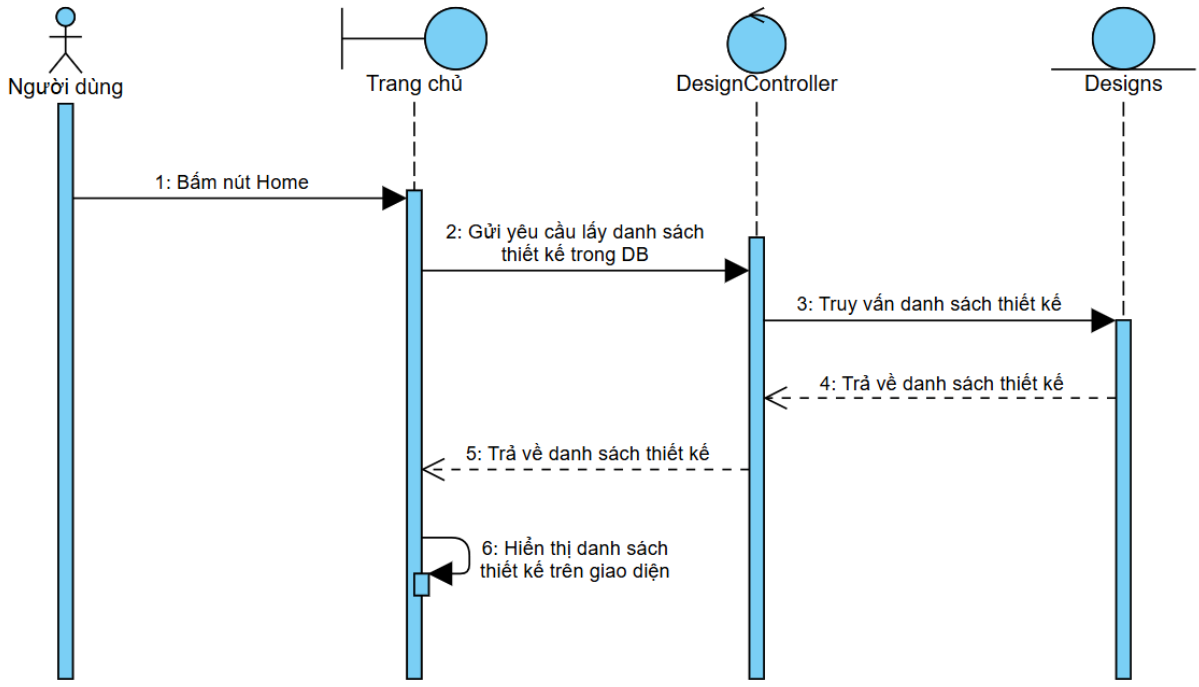
Hình 6: Sơ đồ tuần tự của Đăng nhập

3. Đăng xuất



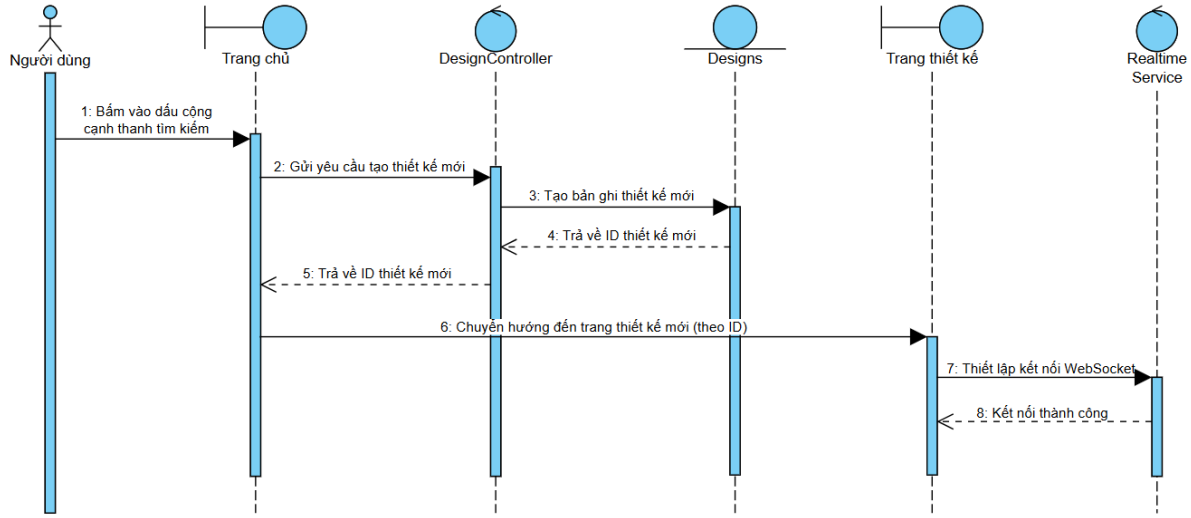
Hình 7: Sơ đồ tuần tự của Đăng xuất

4. Xem danh sách thiết kế



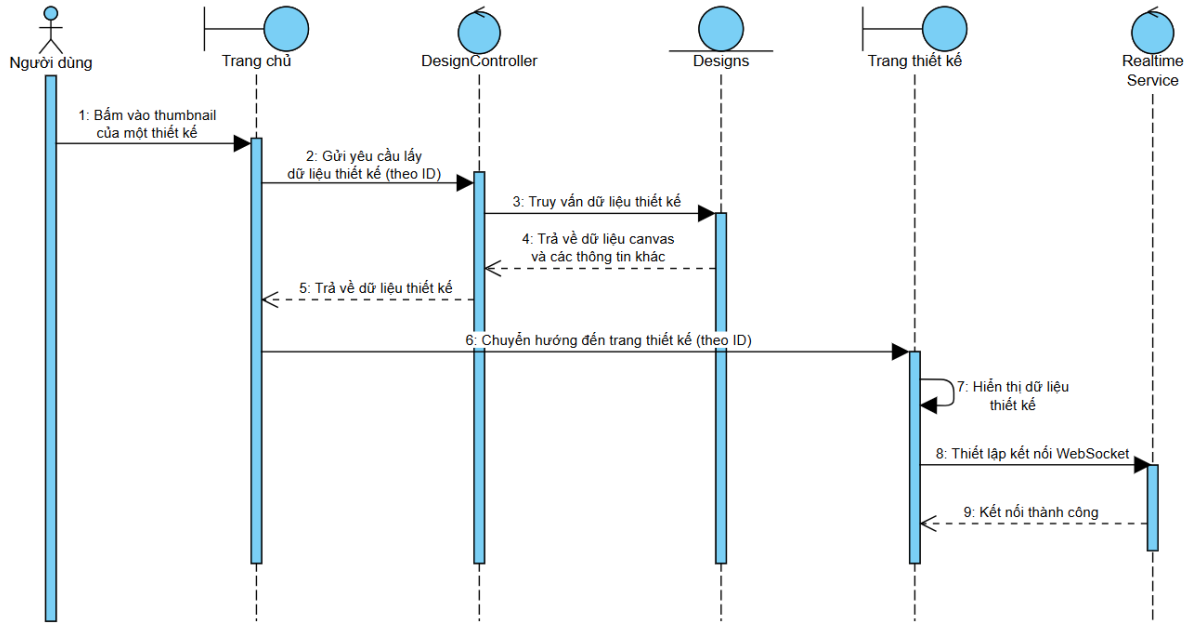
Hình 8: Sơ đồ tuần tự của danh sách thiết kế

5. Tạo thiết kế



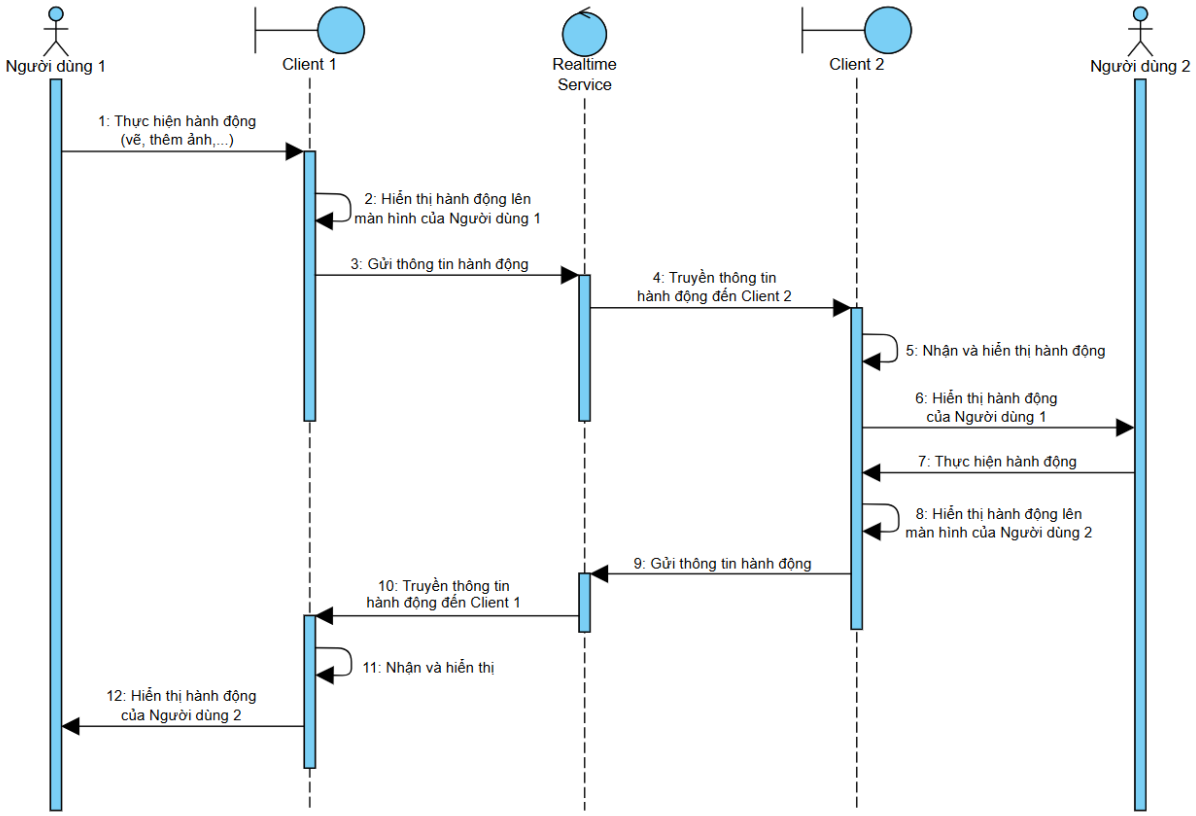
Hình 9: Sơ đồ tuần tự của Tạo thiết kế

6. Xem chi tiết thiết kế



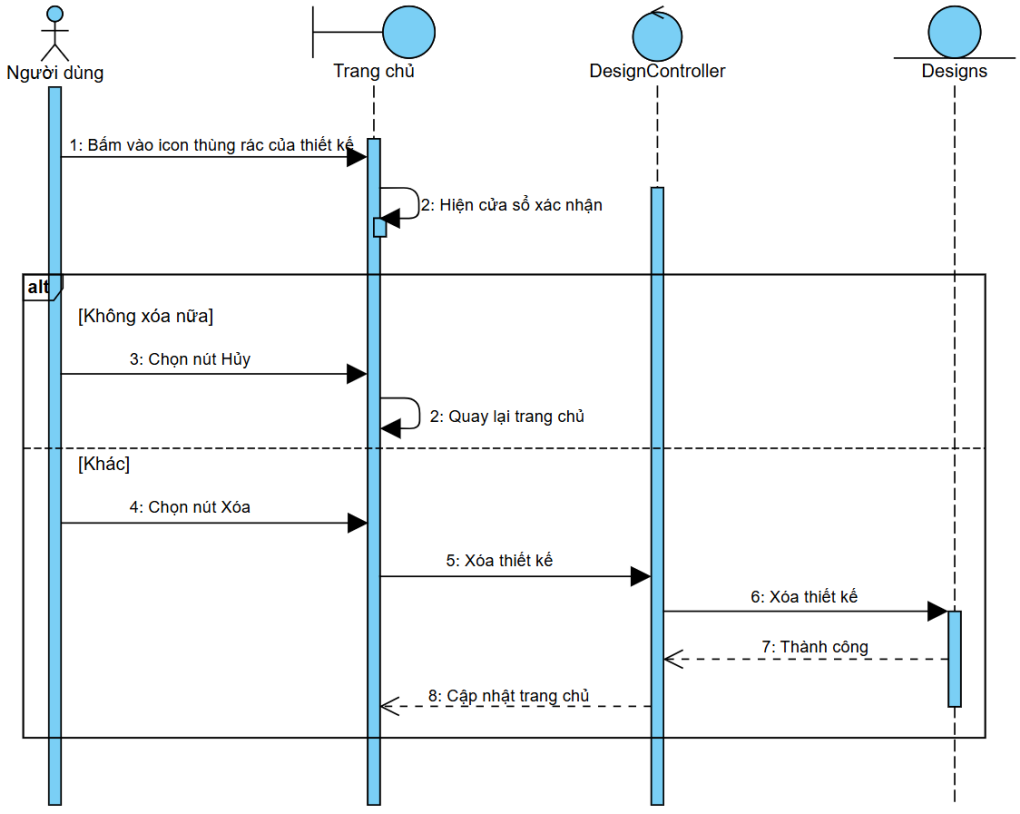
Hình 10: Sơ đồ tuần tự của xem chi tiết thiết kế

7. Cộng tác trên thiết kế



Hình 11: Sơ đồ tuần tự của Cộng tác trên thiết kế

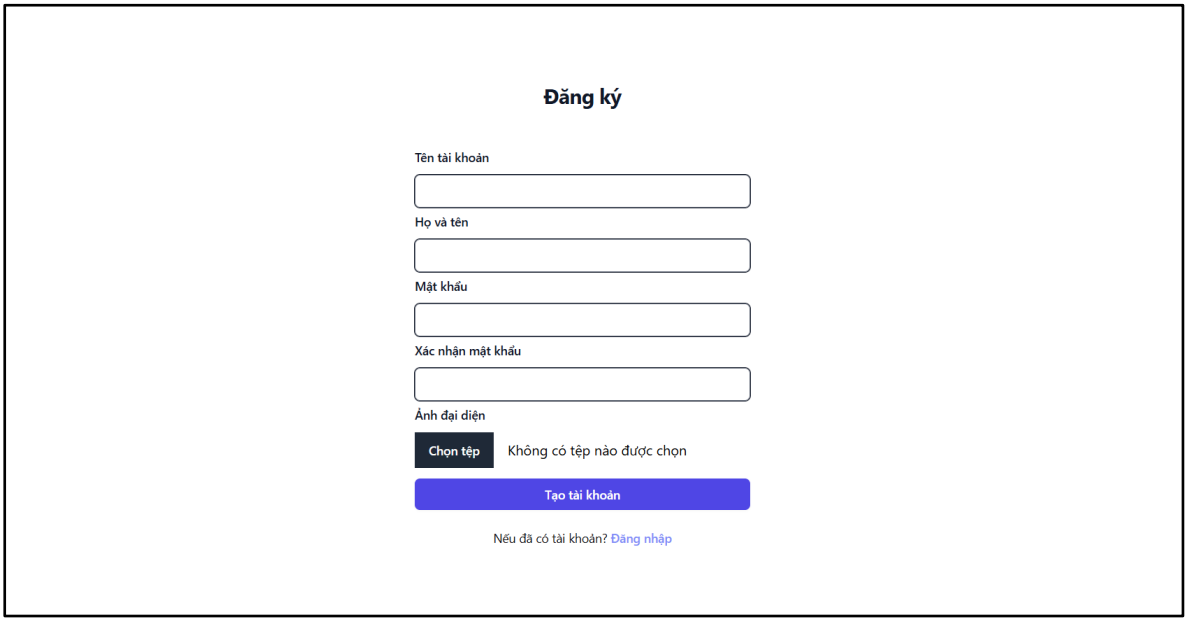
8. Xóa thiết kế



Hình 12: Sơ đồ tuần tự của Xóa thiết kế

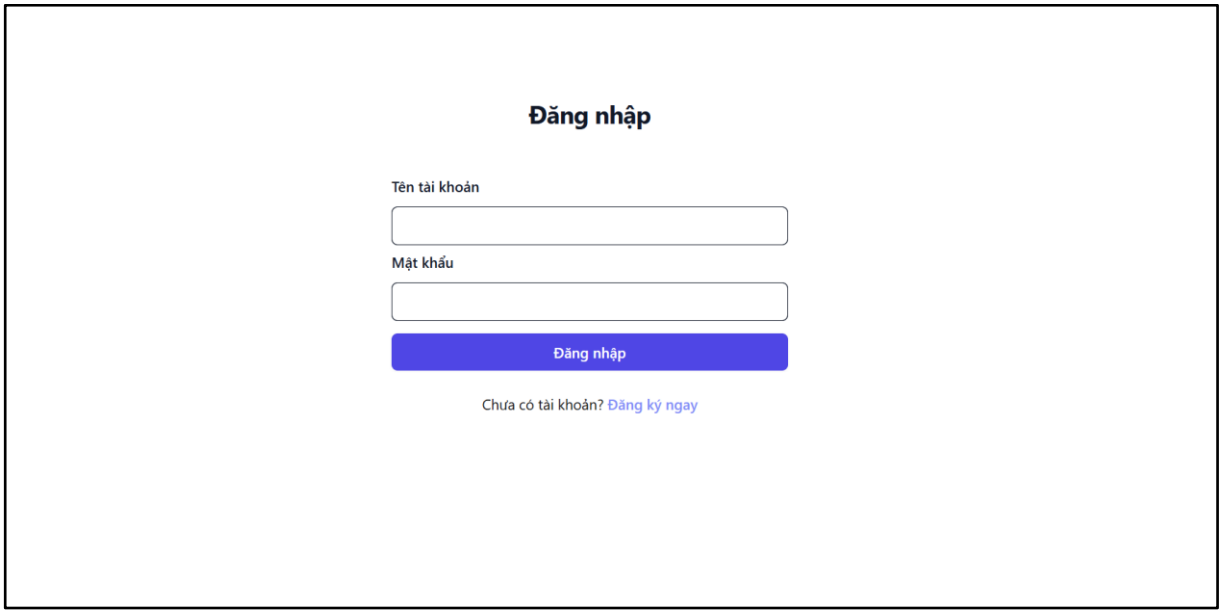
VI. Thiết kế giao diện

1. Màn hình đăng kí



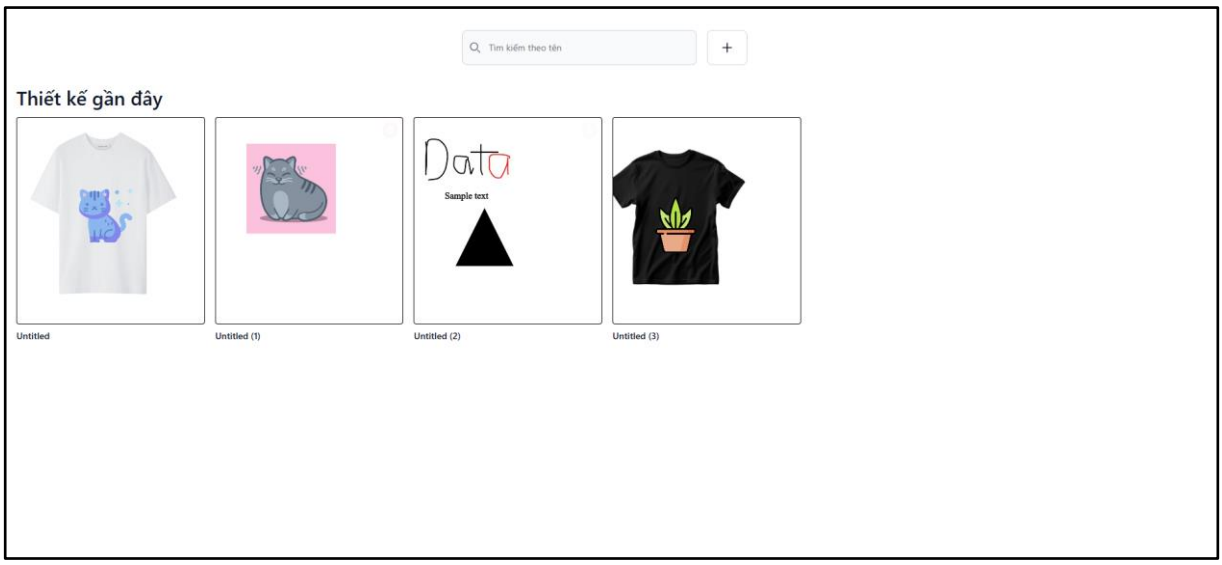
Hình 13: Giao diện màn hình đăng ký

2. Màn hình đăng nhập



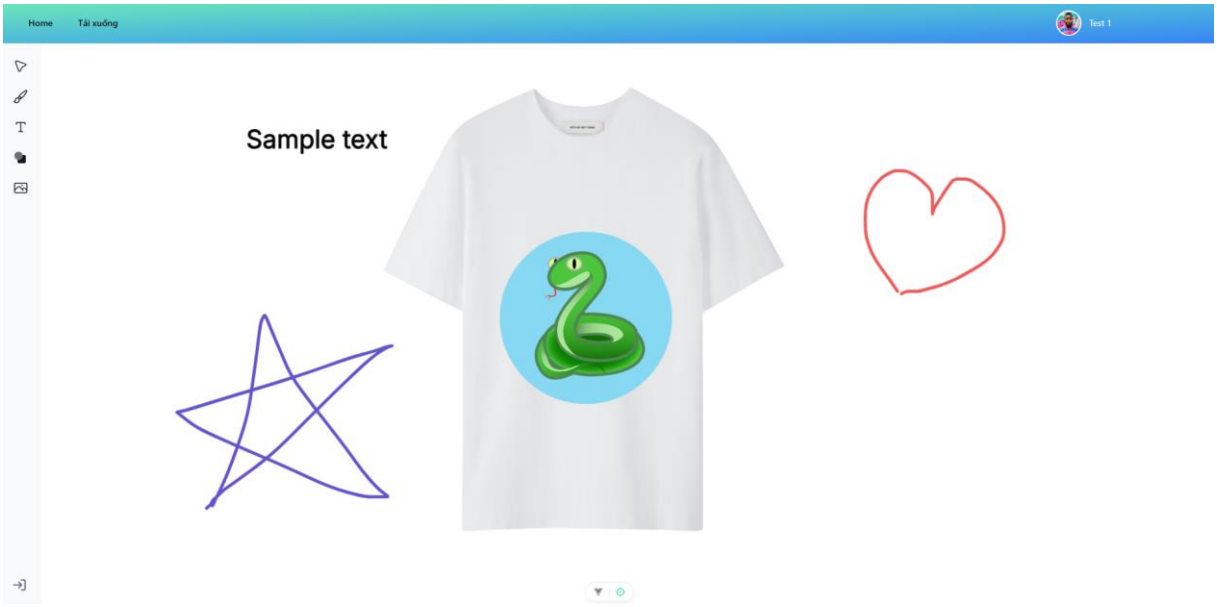
Hình 14: Giao diện màn hình đăng nhập

3. Trang chủ



Hình 15: Màn hình Trang chủ / Danh sách thiết kế

4. Giao diện thiết kế



Hình 16: Giao diện khi mở thiết kế

VII. Kết quả và đánh giá

Sau quá trình thiết kế, phát triển và triển khai, hệ thống Kanva đã đạt được những kết quả tích cực, chứng minh tính khả thi của kiến trúc microservices trong việc xây dựng một nền tảng cộng tác thời gian thực. Các mục tiêu đề ra ban đầu đã được hoàn thành, và hệ thống cho thấy tiềm năng lớn để phát triển trong tương lai.

1. Các tính năng đã triển khai thành công

Hệ thống đã triển khai thành công các chức năng cốt lõi, phản ánh đúng các yêu cầu được định nghĩa trong sơ đồ Use Case và sơ đồ tuần tự. Cụ thể:

- **Quản lý người dùng:** Chức năng đăng ký, đăng nhập và đăng xuất đã được xây dựng một cách an toàn và hiệu quả thông qua **User Service**. Người dùng có thể dễ dàng quản lý tài khoản của mình.
- **Quản lý thiết kế: Design Service** xử lý việc tạo, mở, và xóa các thiết kế. Dữ liệu của các thiết kế, bao gồm cả nội dung canvas, được lưu trữ linh hoạt dưới dạng JSON trong MongoDB, giúp hệ thống dễ dàng thích ứng với các loại đối tượng vẽ khác nhau.
- **Cộng tác thời gian thực:** Đây là tính năng trung tâm của hệ thống, được hiện thực hóa thông qua **Realtime Service** sử dụng giao thức **WebSocket**. Mọi hành động của người dùng (vẽ, thêm văn bản, di chuyển đối tượng, v.v.) đều được đồng bộ hóa tức thời trên màn hình của tất cả những người dùng khác đang tham gia cùng một phiên làm việc.
- **Lưu trữ dữ liệu có điều kiện:** Cơ chế lưu trữ dữ liệu chỉ khi tất cả người dùng thoát khỏi phiên đã được triển khai thành công, giúp giảm tải cho cơ sở dữ liệu và tối ưu hiệu suất trong các phiên làm việc kéo dài và có nhiều tương tác liên tục.

2. Đánh giá về hiệu suất và kiến trúc

- **Độ trễ thấp:** Nhờ vào giao thức WebSocket, hệ thống đạt được độ trễ thấp trong quá trình đồng bộ hóa. Các hành động của người dùng được cập nhật gần như ngay lập tức, mang lại trải nghiệm mượt mà và trực quan.
- **Tính mở rộng và khả năng chịu lỗi:** Việc áp dụng kiến trúc microservices đã chứng minh được tính hiệu quả. Mỗi dịch vụ hoạt động độc lập, do đó nếu một dịch vụ gặp sự cố, các dịch vụ còn lại vẫn có thể tiếp tục hoạt động. Điều này giúp tăng cường khả năng chịu lỗi của toàn hệ thống.
- **Ưu điểm của Docker và NestJS:** Sử dụng Docker giúp việc triển khai trở nên nhất quán và dễ dàng trên các môi trường khác nhau. NestJS đã cung cấp một framework mạnh mẽ và có cấu trúc để xây dựng các microservice một cách hiệu quả, đặc biệt trong việc tích hợp WebSocket một cách liền mạch.

3. Hạn chế

Mặc dù hệ thống đã đạt được các mục tiêu đề ra, vẫn còn tồn tại một số hạn chế nhất định cần được cải thiện trong các phiên bản tương lai:

- **Tính năng cơ bản:** Hệ thống hiện tại chỉ hỗ trợ các công cụ vẽ và đối tượng cơ bản.
- **Khả năng phục hồi trạng thái:** Hiện tại, hệ thống chưa có tính năng undo/redo hoặc lịch sử phiên bản của thiết kế, điều này có thể hạn chế trải nghiệm người dùng trong các trường hợp cần khôi phục lại thao tác.
- **Cơ chế phân quyền:** Cơ chế quản lý phiên bản thiết kế và phân quyền truy cập vẫn còn đơn giản, cần được mở rộng để hỗ trợ các kịch bản làm việc nhóm phức tạp hơn.

VII. Kết luận

Việc hoàn thành thành công dự án Kanva đánh dấu một cột mốc quan trọng, khẳng định năng lực và tinh thần làm việc chuyên nghiệp của đội ngũ phát triển. Kanva, được thiết kế dưới dạng một hệ thống phân tán, giải quyết những thách thức trong việc cộng tác sáng tạo trực tuyến. Các tính năng cốt lõi của hệ thống bao gồm tạo, chỉnh sửa và chia sẻ bảng trắng theo thời gian thực, cho phép nhiều người dùng cùng làm việc trên một thiết kế một cách hiệu quả.

Phần backend của Kanva được xây dựng dựa trên kiến trúc Microservices, một lựa chọn chiến lược nhằm đảm bảo hiệu suất cao và khả năng mở rộng linh hoạt. Mỗi dịch vụ và cơ sở dữ liệu đều được thiết kế độc lập, tối ưu hóa cho từng chức năng cụ thể, từ đó nâng cao tính ổn định và khả năng phục hồi của toàn hệ thống. Sự thành công của dự án là minh chứng cho việc ứng dụng một cách sâu sắc và bài bản các kiến thức về hệ thống phân tán vào toàn bộ quy trình từ lên ý tưởng, xây dựng kiến trúc cho đến triển khai chi tiết.

Chúng tôi tin rằng những kinh nghiệm và kỹ năng có được trong quá trình phát triển Kanva sẽ là nền tảng vững chắc cho các dự án công nghệ phức tạp trong tương lai, góp phần thúc đẩy sự đổi mới và sáng tạo.