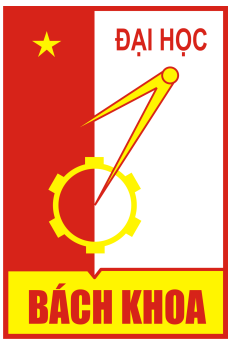
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN TRUYỀN THÔNG



BÁO CÁO BÀI TẬP LỚN

Đề tài: Ứng dụng hỗ trợ quản lý công việc thông qua mô hình bảng kanban

Giảng viên hướng dẫn	TS. Nguyễn Nhất Hải
Mã lớp học	121767
Thành viên	Nguyễn Bá Nam - 20176103 Phan Phú Thành - 20176113 Trần Đình Vũ - 20176126

Hà Nội, tháng 05 năm 2021

MỤC LỤC

I. Đặt vấn đề

1. Lý do chọn đề tài:

_ Các ứng dụng hỗ trợ quản lý đội nhóm và phân chia công việc đang ngày càng được sử dụng rộng rãi như Trello, Notion, Github Project,... Đa số các dự án này đều sử dụng mô hình bảng kanban để phân chia công việc.

_ Lựa chọn đề tài thiết kế và xây dựng lại một hệ thống mô phỏng các ứng dụng trên nhằm mục đích tìm hiểu về cách thức hoạt động của các ứng dụng đã có, luyện tập các kỹ năng đã học cũng như tạo cơ hội tiếp cận, luyện tập sử dụng các công nghệ, các thiết kế mới.

_ Hệ thống được xây dựng chỉ mang mục đích học tập và tìm hiểu nên còn nhiều điểm chưa được hợp lý và chưa hoàn thiện hoàn toàn.

2. Mục tiêu:

_ Thiết kế được một hệ thống phân tán sử dụng kiến trúc microservice để thực hiện các tính năng cơ bản của một ứng dụng hỗ trợ phân chia công việc:

+ Xác thực người dùng
+ Phân chia đội nhóm
+ Phân chia theo các dự án
+ Sử dụng được tính năng của bảng kanban
+ .....

_ Cùng với đó áp dụng các công cụ khác nhau để hỗ trợ phát triển cũng như tạo điều kiện để tìm hiểu cách áp dụng chúng trong môi trường phát triển phần mềm:

+ Docker, docker-compose: tự động hóa việc triển khai, tích hợp
+ Swagger, postman: Định nghĩa các Restful API
+ RabbitMQ, RPC, HTTP: các loại giao thức trao đổi tin nhắn

II. Công nghệ sử dụng

1. Ngôn ngữ lập trình:

1.1. Golang:

_ Golang(Go) là một mã nguồn mở được sáng tạo ra từ lập trình viên của Google từ năm 2009. Ngôn ngữ này giúp tạo được một ứng dụng đơn giản, nhanh chóng và đáng tin cậy.

_ Đi kèm với Go và một bộ các công cụ được nhóm sáng lập xây dựng kèm theo (Go toolchain) với mục đích tạo ra môi trường phát triển dễ sử dụng và cài đặt cho các lập trình viên như govet, golint, gopls,....

_ Go là một ngôn ngữ statically-typed nhưng không quá chặt chẽ như C. Bên cạnh đó Go tuy có hỗ trợ các phương thức hướng đối tượng nhưng không hoàn toàn là một ngôn ngữ hướng đối tượng. Cho nên số lượng mã nguồn của Go thường khá nhiều. Là một ngôn ngữ compiled, file thực thi được tạo ra có thể chạy được trên các môi trường mà không cần cài thêm bất cứ thứ gì, tuy nhiên do các thư viện được đóng gói vào trong file thực thi nên dung lượng của file này khá lớn (25 - 30MB).

1.2. HTML & CSS:

_ HTML là một phần không thể thiếu khi nói đến lập trình web. Mức độ phổ biến của HTML ngày càng tăng và nó được coi là tiêu chuẩn web chính thức.

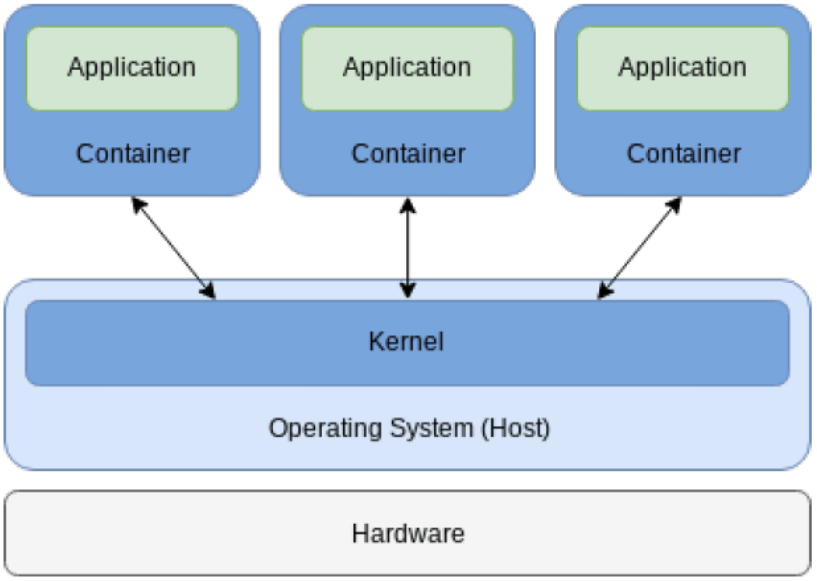
_ HTML là viết tắt của Hyper Text Markup Language (ngôn ngữ đánh dấu siêu văn bản). HTML cho phép người dùng tạo và cấu trúc hóa các thành phần trên một trang web như đoạn văn, tiêu đề, liên kết, trích dẫn, bảng biểu…

_ CSS là viết tắt của từ viết tắt của từ Cascading Style Sheets. Với CSS chúng ta có thể thỏa sức sáng tạo trong thiết kế website bằng cách tùy chỉnh vị trí các phần tử, màu sắc, màu nền, font chữ, thứ tự sắp xếp của các phần tử, hiệu ứng (đổ bóng, bo góc, xoay...) những điều mà HTML gần như không thể làm được.

2. Công cụ hỗ trợ tự động triển khai:

2.1. Docker:

_ Docker là một công cụ hỗ trợ xây dựng và triển khai các ứng dụng trên nền tảng các các containers.



_ Container được coi như một tiến trình trong Linux được cấu hình (CPU, RAM, threads, network,...), đây cũng là một môi trường dùng để đóng gói ứng dụng và các thành phần liên quan giúp người phát triển có thể dễ dàng triển khai thông qua các image.

_ Container image là các file nén (tar) được sắp xếp theo mô hình layer kèm với các metadata. Kiến trúc mỗi một layer là một file nén hỗ trợ cho việc tái sử dụng các layer này để xây dựng image một cách nhanh chóng và tốn ít tài nguyên hơn.

2.2. Docker-compose:

_ Docker-compose là một công cụ dùng để định nghĩa và khởi chạy các docker containers thông qua việc cấu hình file YAML.

_ Việc sử dụng docker-compose giúp cho triển khai cũng như quản lý các docker containers được hiệu quả và nhanh chóng hơn. Ví dụ một số tính năng như:

+ Tự động tạo network layer ảo cho các containers
+ Tự động mount và tạo các volume
+ Kéo và triển khai các container từ docker hub

2.3. Docker-compose-wait:

_ Do docker-compose thiếu tính năng đợi các container được triển khai hoàn toàn rồi mới triển khai công cụ khác, cho nên docker-compose-wait ra đời nhằm mục đích khắc phục điểm yếu đó.

_ Với công cụ này, người phát triển có thể đính kèm nó vào các container image và điều chỉnh thông số qua các biến môi trường như WAIT_HOSTS, WAIT_AFTER,...

_ Để đọc thêm về cách thức triển khai và tích hợp, vui lòng xem thêm tại: https://github.com/ufoscout/docker-compose-wait

3. Công cụ hỗ trợ tích hợp hệ thống:

3.1. Hamachi:

_ Trong không gian mạng Internet, việc truy cập thông qua public IP của nhau là rất khó khăn do IP thay đổi liên tục cũng như cần phải thông qua được các lớp NAT, firewall,...

_ Cho nên Hamachi ra đời nhằm mục đích tạo mạng LAN ảo với static public IP cho người dùng. Việc cài đặt hamachi rất dễ dàng, chỉ cần đăng ký tài khoản, cài và khởi chạy ứng dụng là có thể tự tạo được một mạng riêng ảo cho bản thân và mọi người.

_ Hamachi cũng hỗ trợ miễn phí tới tối đa 5 người dùng trong một mạng MESH, khá phù hợp với các nhóm phát triển nhỏ lẻ, cần tích hợp và sửa lỗi nhanh chóng. Tuy nhiên việc kết nối thông qua mạng LAN ảo này khá chậm và không ổn định, cho nên không khuyến khích sử dụng công cụ này cho môi trường production.

_ Để tìm hiểu thêm về hamachi, vui lòng truy cập: https://www.vpn.net

3.2. Postman:

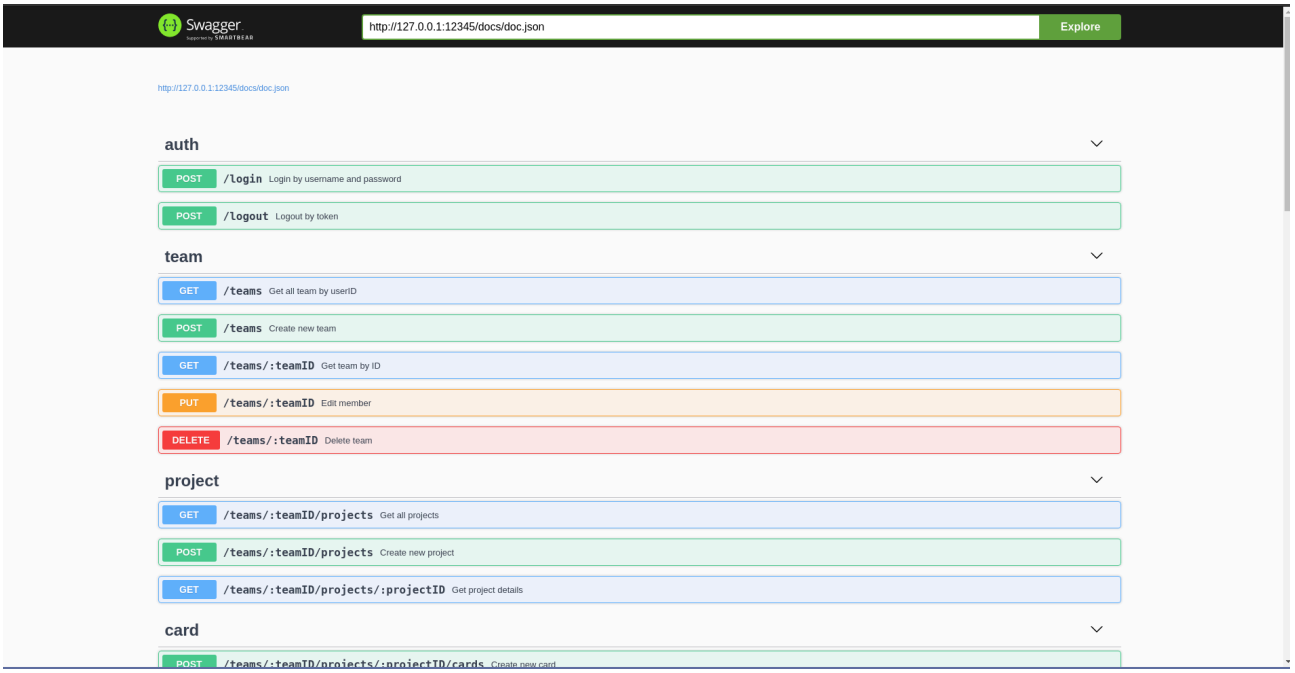
_ Postman là một công cụ cho phép chúng ta thao tác với API, đặc biệt là RESTful API, đây cũng là một công cụ phổ biến nhất giúp việc chia sẻ và kiểm thử API được tiện lợi hơn.

_ Postman hỗ trợ việc tạo các Collections(nhóm các APIs) để thuận tiện trong việc quản lý. Cùng với đó, công cụ này cũng cung cấp một môi trường chia sẻ đồng bộ giữa nhiều người trong cùng một nhóm.

_ Nhược điểm của việc sử dụng Postman là khi trong môi trường nhóm, các request và API bị giới hạn, khi vượt quá giới hạn này thì các Collections sẽ tự động bị archived và không thể sử dụng trên Postman nữa.

3.3. Swagger (swaggo):

_ Swagger cũng là một công cụ hỗ trợ phát triển và kiểm thử API. Bằng cách sử dụng cách định nghĩa OpenAPI, các file này có thể được chia sẻ rộng rãi giữa các lập trình viên. Cùng với đó là OpenAPI cũng hỗ trợ việc tái sử dụng component để định nghĩa các API nhằm mục đích tối thiểu công sức mà người phát triển cần bỏ ra. Bên cạnh đó là WebUI để thực hiện gọi các API thông qua trình duyệt



_ Swaggo là công cụ hỗ trợ việc sinh file swagger và tích hợp luôn Web UI vào trong ứng dụng, đọc thêm tại: https://github.com/swaggo

4. Các dịch vụ lưu trữ:

4.1. Github:

_ Github là kho lưu trữ mã nguồn có thể nói là phổ biến nhất trên thế giới hiện nay. Github hỗ trợ việc tạo các repositories, branches, projects,... giúp tối ưu việc quản lý mã nguồn của các nhà phát triển.

4.2. Github packages:

_ Github packages có thể nói là một trong các tính năng ưu việt của Github hỗ trợ việc lưu các docker images, npm, Maven,...

_ Việc lưu trữ các docker images trên Github packages của repo tạo điều kiện thuận lợi để chia sẻ và tích hợp giữa các lập trình viên mà không cần sử dụng thêm một nền tảng nào khác.

5. Các công cụ giao tiếp:

5.1. RabbitMQ:

_ RabbitMQ được giới thiệu là open source message broker được sử dụng rộng rãi nhất trên thế giới. Việc sử dụng ít tài nguyên cùng với độ dễ dàng khi triển khai có thể là một trong những ưu điểm khiến cho công cụ này đạt được thành tựu trên.

_ RabbitMQ sử dụng chủ yếu giao thức AMQP (Advanced Message Queuing Protocol - Giao thức hàng đợi thư nâng cao) là một giao thức lớp ứng dụng tiêu chuẩn mở. Đó là một giao thức nhị phân được thiết kế để hỗ trợ một loạt các ứng dụng nhắn tin và các mẫu liên lạc.

5.2. gRPC:

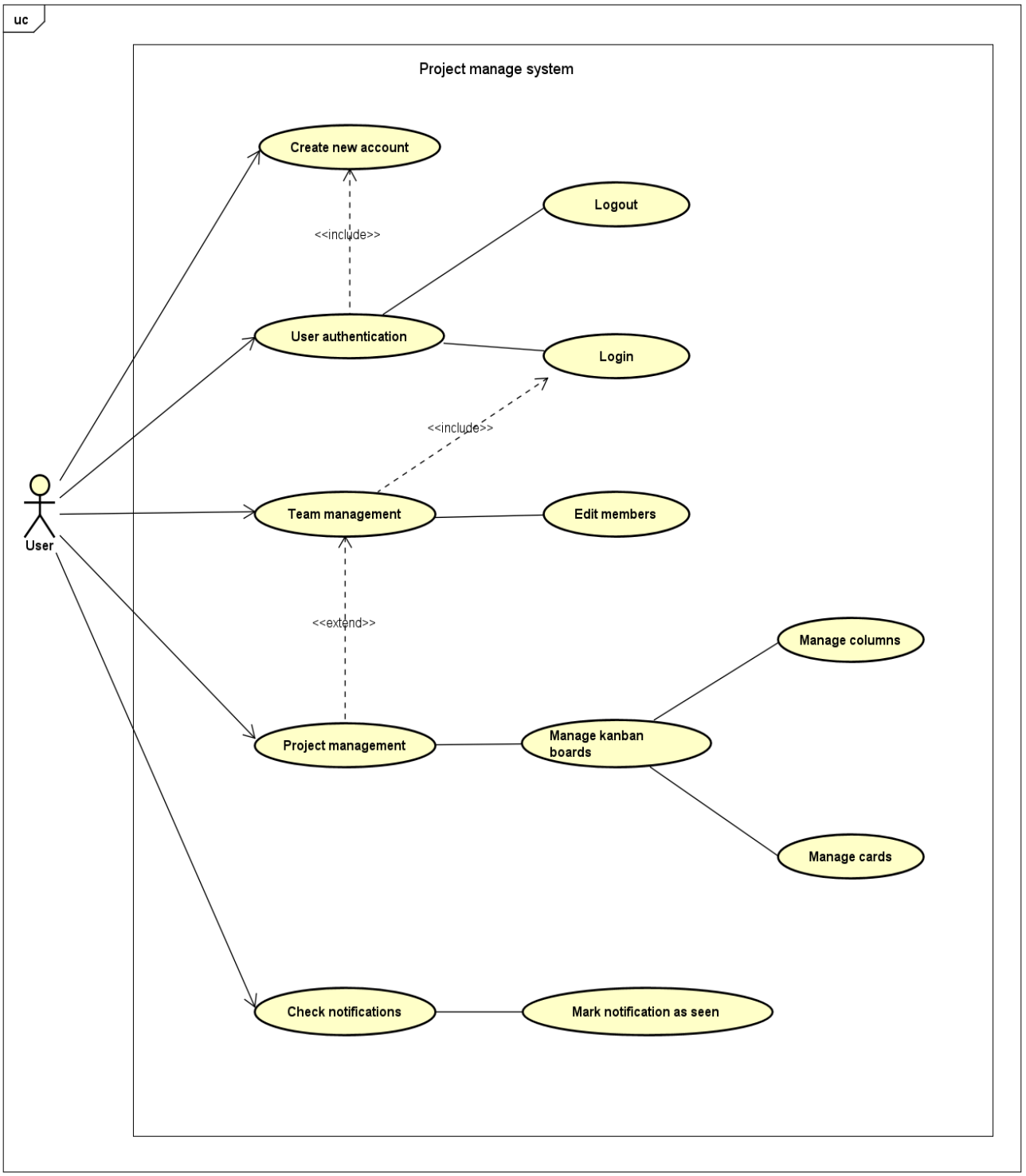
_ gRPC là một open source framework hỗ trợ giao thức RPC thông qua việc trao đổi các tin nhắn dưới dạng protobuf được phát triển bởi Google. gRPC cho thấy tốc độ nhanh hơn REST từ 7 - 10 lần.

_ Việc sử dụng gRPC và protobuf có thể cải thiện hiệu năng của hệ thống, cũng như các tin nhắn ở dạng nhị phân nên thời gian chuyển tiếp nhanh hơn và khả năng bị đọc trộm khó hơn. gRPC cũng hỗ trợ việc streaming giúp việc trả về dữ liệu được nhanh chóng và ổn định hơn

III. Phân tích và thiết kế hệ thống

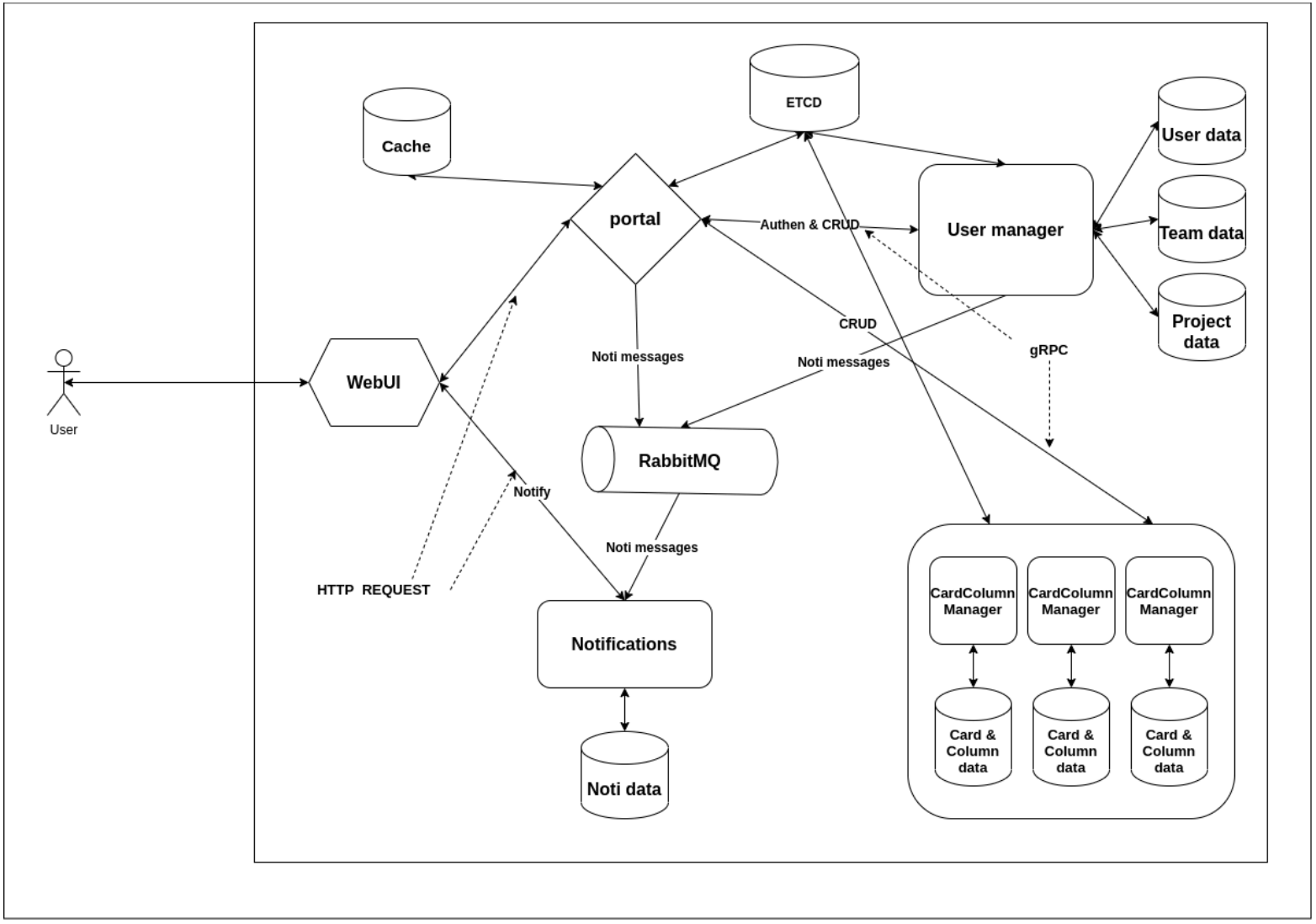
1. Sơ đồ usecase:

_ Sơ đồ usecase về các nghiệp vụ chính của actor user.



2. Thiết kế hệ thống:

2.1. Sơ đồ tổng quan:



2.2. Giải thích các thành phần:

2.2.1. WebUI:

_ Giao diện hiển thị cho người dùng truy cập. Có thể hiển thị được trên các trình duyệt, người dùng sẽ chỉ thao tác thông qua thành phần này của hệ thống.

2.2.2. Portal & cache:

_ Portal có trách nhiệm như một cổng ra vào đối với hệ thống, toàn bộ các request của người dùng sẽ phải đi qua đây để thực hiện xác thực.
_ Portal cũng có trách nhiệm chuyển đổi các HTTP request sang các định dạng khác phù hợp để thực hiện tác vụ của người dùng như gRPC, hay message queue.
_ Cache được sử dụng để lưu lại các thông tin đăng nhập của người dùng, phục vụ cho việc xác thực.

2.2.3. Usermanager & usermanagerDB:

_ 2 thành phần này dùng để lưu trữ và thực hiện các yêu cầu liên quan đến người dùng, nhóm và dự án.
_ Đây cũng là nơi lưu trữ thông tin về các shardID của project để hỗ trợ cho việc query bảng kanban của chính project đó

2.2.4. CardColumnManager & CardColumnManagerDB:

_ 2 thành phần này dùng để thực hiện các thao tác liên quan tới dữ liệu của bảng kanban. Mỗi một CardColumnManager sẽ có 1 shardID và 1 DB đi kèm để hỗ trợ cho việc sharding, nhằm chia tải của hệ thống.

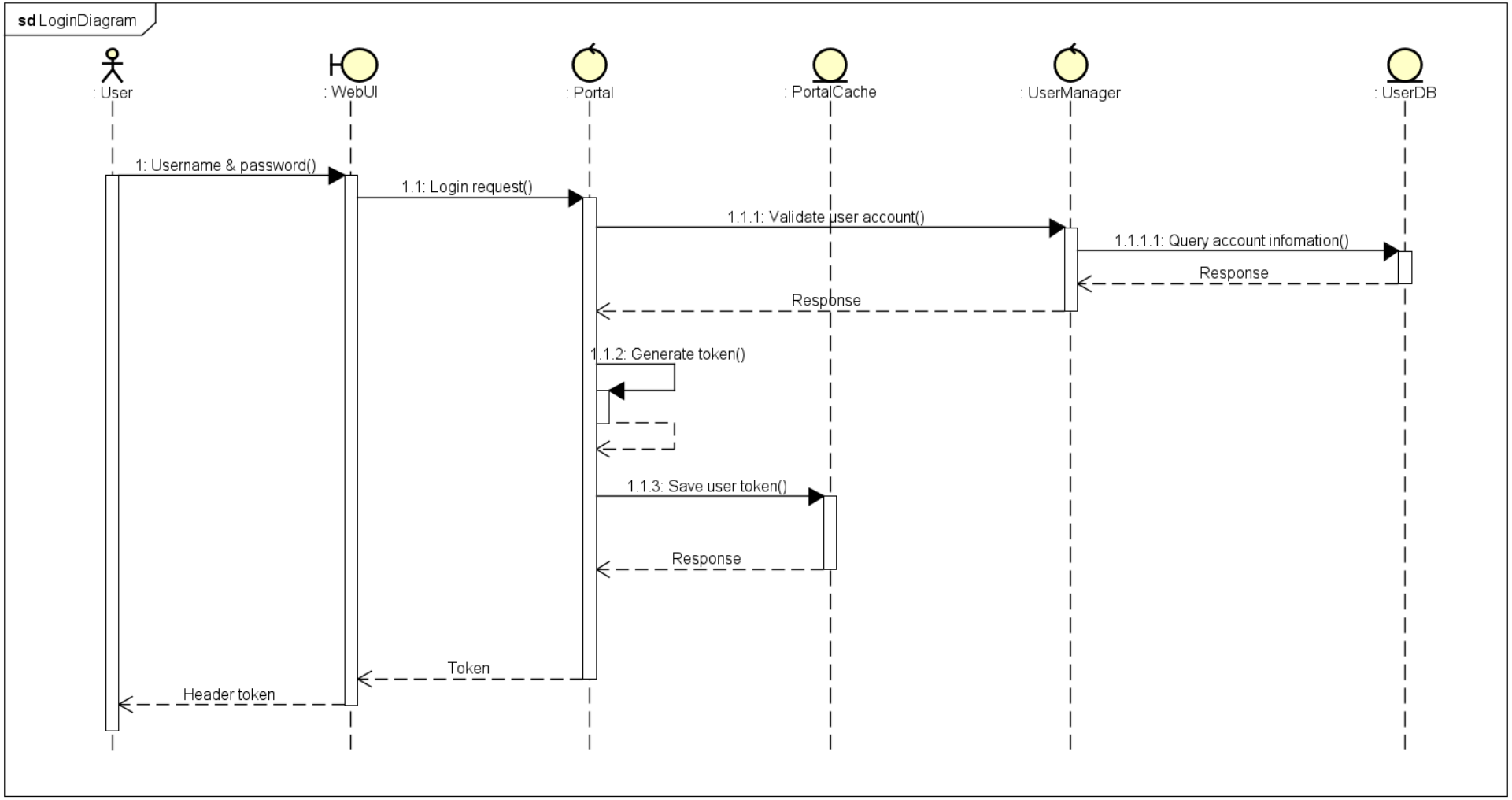
2.2.5. Notifications & RabbitMQ:

_ Notifications được dùng để lưu trữ các thông tin liên quan tới thông báo tới người dùng, tuy nhiên hiện tại đang dùng HTTP Request nên có vấn đề về bảo mật
_ RabbitMQ là 1 message queue dùng để chuyển các tin nhắn thông báo từ các thành phần khác tới notifications.

3. Sơ đồ sequence:

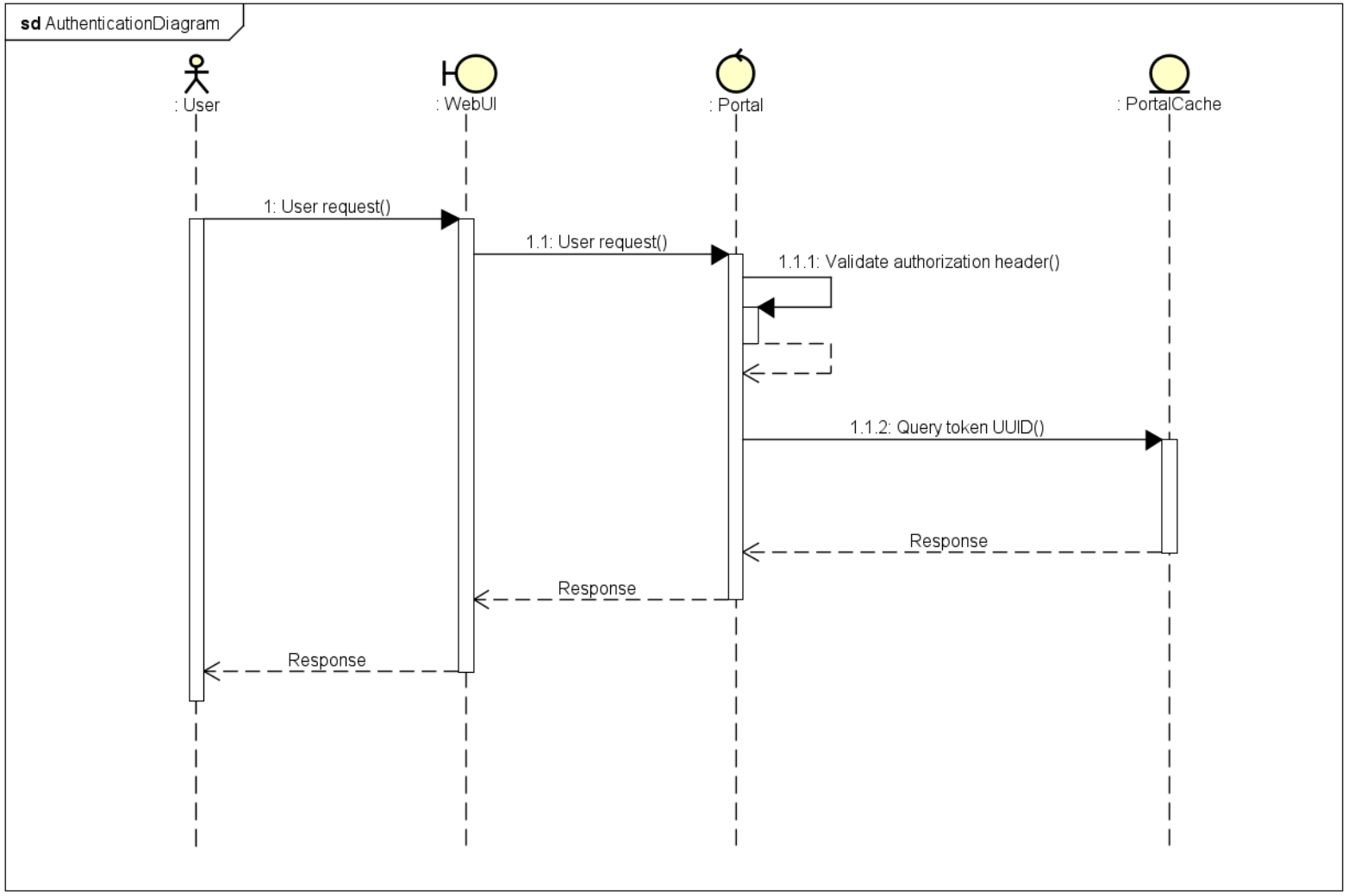
3.1. Login:

_ Sequence diagram về hoạt động đăng nhập của user



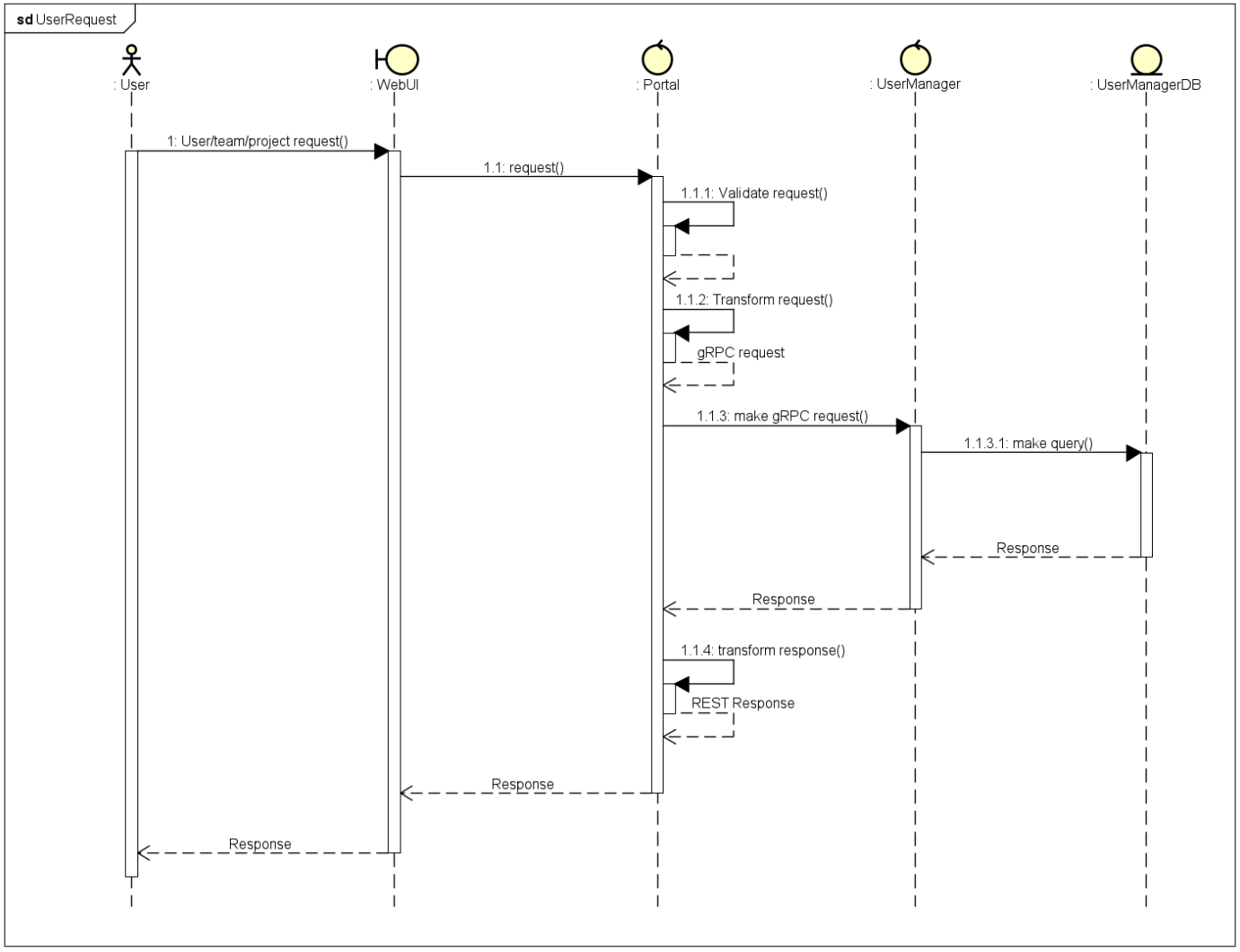
3.2. Authentication:

_ Sequence diagram về xác thực các request trong hệ thống



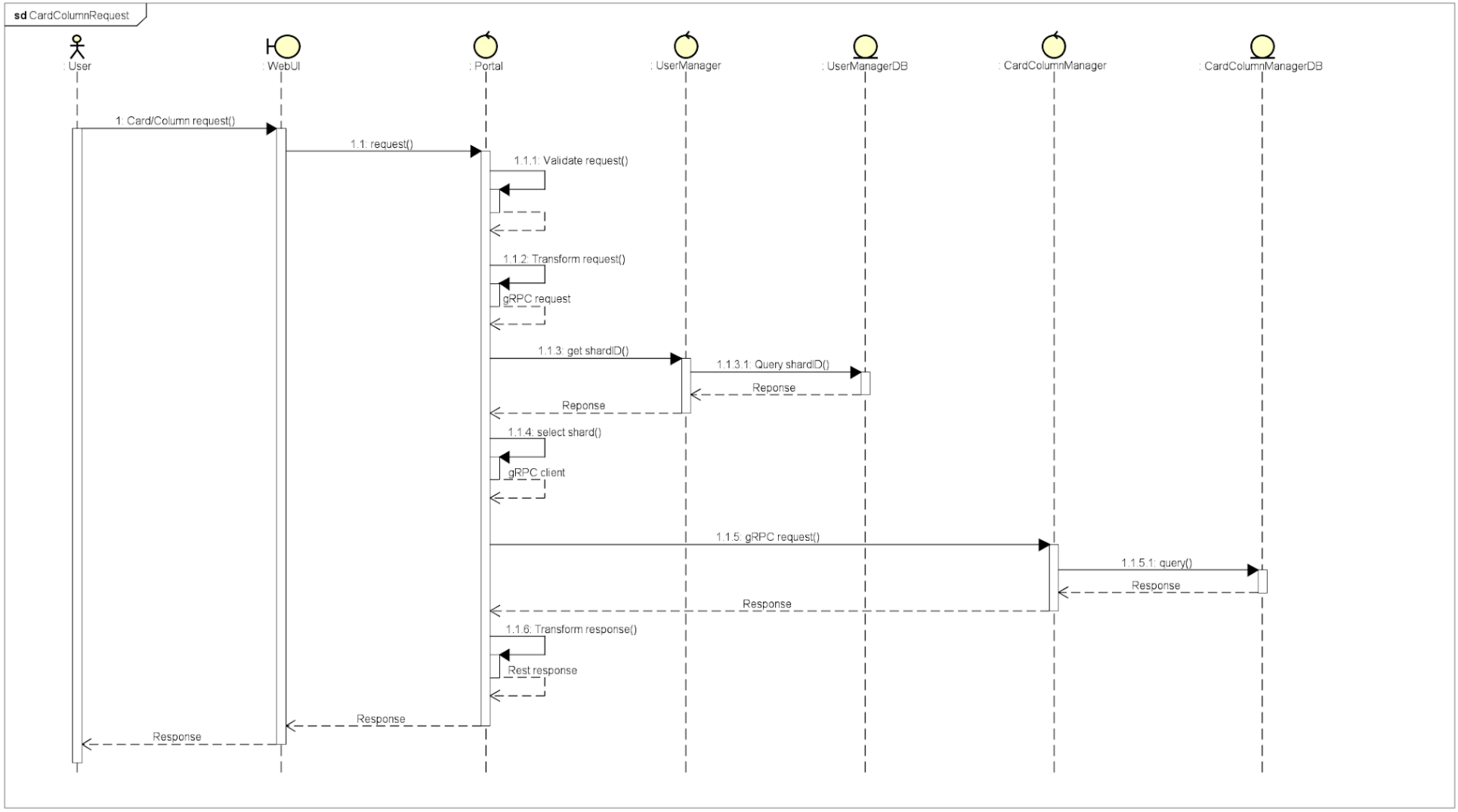
3.3. Usermanager request:

_ Sequence diagram về thực hiện các request liên quan đến user, team và project.



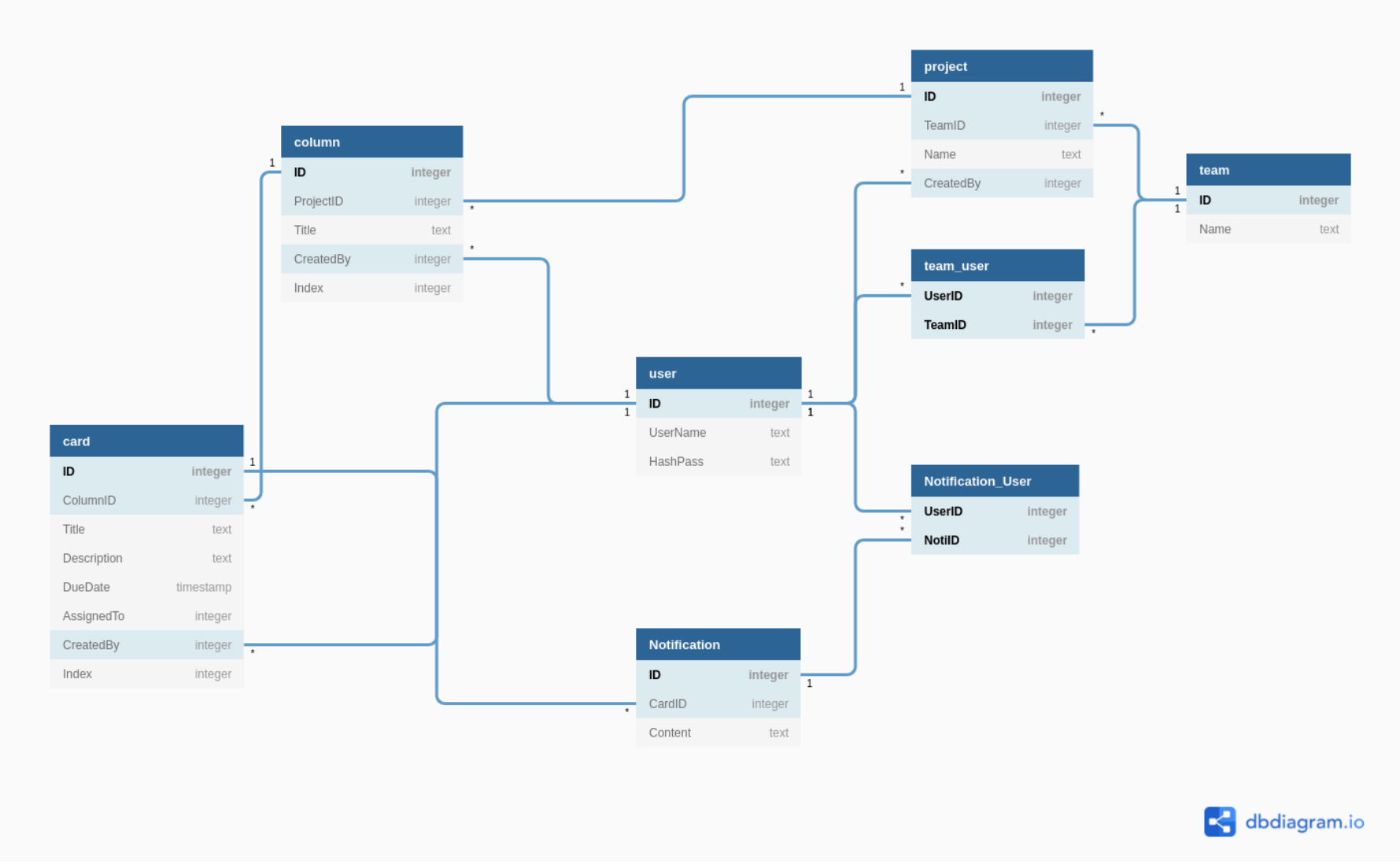
3.4. Card & column request:

_ Sequence diagram về việc thực hiện các request liên quan đến bảng kanban (gồm card và column)

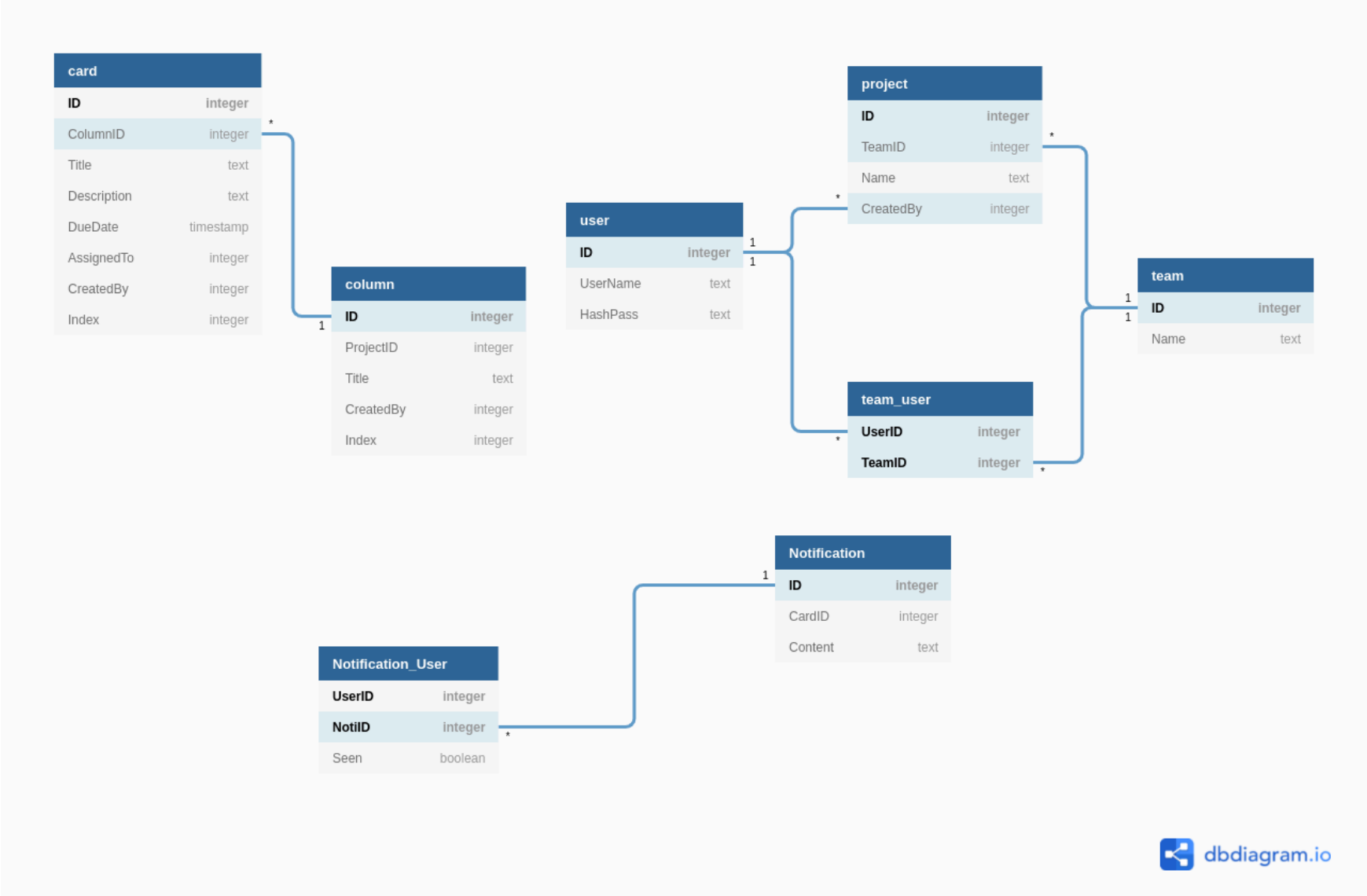


4. Cơ sở dữ liệu:

4.1. Cơ sở dữ liệu tổng quát



4.2. Cơ sở dữ liệu phân tán



IV. Đánh giá và kết luận

1. Các mục tiêu đã hoàn thành:

_ Đã hoàn thiện được các chức năng chính của các thành phần trong hệ thống.
_ Xây dựng được WebUI
_ Sử dụng docker để tự động hóa việc triển khai trên các máy khác
_ Hoàn thiện các tài liệu liên quan

2. Các nhược điểm còn tồn tại:

_ Hệ thống có chỗ còn chưa đảm bảo xác thực của người dùng (notifications không làm nhiệm vụ xác thực JWT qua token tại header của request)
_ Chưa đảm bảo được phân quyền trong hệ thống (người dùng có thể truy cập tới các project, column khác thông qua việc thay đổi đường dẫn URL)
_ Các dữ liệu trong hệ thống chưa được ràng buộc chặt chẽ, dẫn tới việc có thể xảy ra sai sót trong khi chỉnh sửa dữ liệu

3. Phương hướng phát triển:

_ Sử dụng Firebase hoặc các ứng dụng tương tự để thực hiện việc đẩy thông báo cho người dùng một cách phù hợp nhất.
_ Xây dựng ứng dụng phân quyền cho người dùng và các tài nguyên tránh cho việc truy cập tới các tài nguyên không được phép.
_ Áp dụng thêm các ràng buộc về dữ liệu, cùng các kỹ thuật như 2-phase-commit để đảm bảo toàn vẹn dữ liệu.

V. Phân công công việc

STT	Họ và tên	Công việc	Mức độ hoàn thành
1	Nguyễn Bá Nam	_ Tổng hợp và hoàn thiện báo cáo _ Hỗ trợ thiết kế và xây dựng front-end	100%
2	Phan Phú Thành	_ Thiết kế hệ thống back-end _ Xây dựng các thành phần back-end _ Viết docker, docker-compose để hỗ trợ việc tự động triển khai _ Vẽ các sơ đồ miêu tả hệ thống	90%
3	Trần Đình Vũ	_ Thiết kế giao diện back-end _ Góp ý xây dựng các chức năng của ứng dụng _ Cài đặt hamachi để kết hợp kiểm thử	100%

VI. Tài liệu tham khảo

1.https://www.researchgate.net/figure/Docker-container-architecture_fig1_333235708
2.https://golang.org
3.https://docs.docker.com
4.https://github.com/ufoscout/docker-compose-wait
5.https://www.vpn.net
6.https://www.postman.com/
7.https://swagger.io/
8.https://github.com/
9.https://github.com/features/packages
10.https://grpc.io/doc
11.https://dev.to/koddr/working-with-rabbitmq-in-golang-by-examples-2dcn
12.https://etcd.io/docs/v3.4/
13.https://redis.io/documentation
14.https://www.postgresql.org/docs/