



VOU - Marketing with Real-time Games	Phiên bản: 1.0
Thiết kế kiến trúc	Ngày: 08/09/2024

Kiến trúc phần mềm

VOU - Marketing with Real-time Games

Tự đánh giá

Nội dung	Điểm
Phân báo cáo kiến trúc(thang 10):	10.0
Phản ứng dụng(thang 10):	9.5
Phản nâng cao(cộng điểm thêm từ 0->4 điểm):	2

Sinh viên thực hiện:

Phần tự đánh giá % từng thành viên:

MSSV	Họ Và Tên	% đánh giá
21127113	Đinh Dương Hải Nam	100%
21127122	Hồ Thanh Nhân	100%
21127232	Nguyễn Thanh Bình	100%
21127677	Võ Phạm Thanh Phương	100%
21127740	Đoàn Nam Thắng	100%



VOU - Marketing with Real-time Games	Phiên bản: 1.0
Thiết kế kiến trúc	Ngày: 08/09/2024

Mục lục

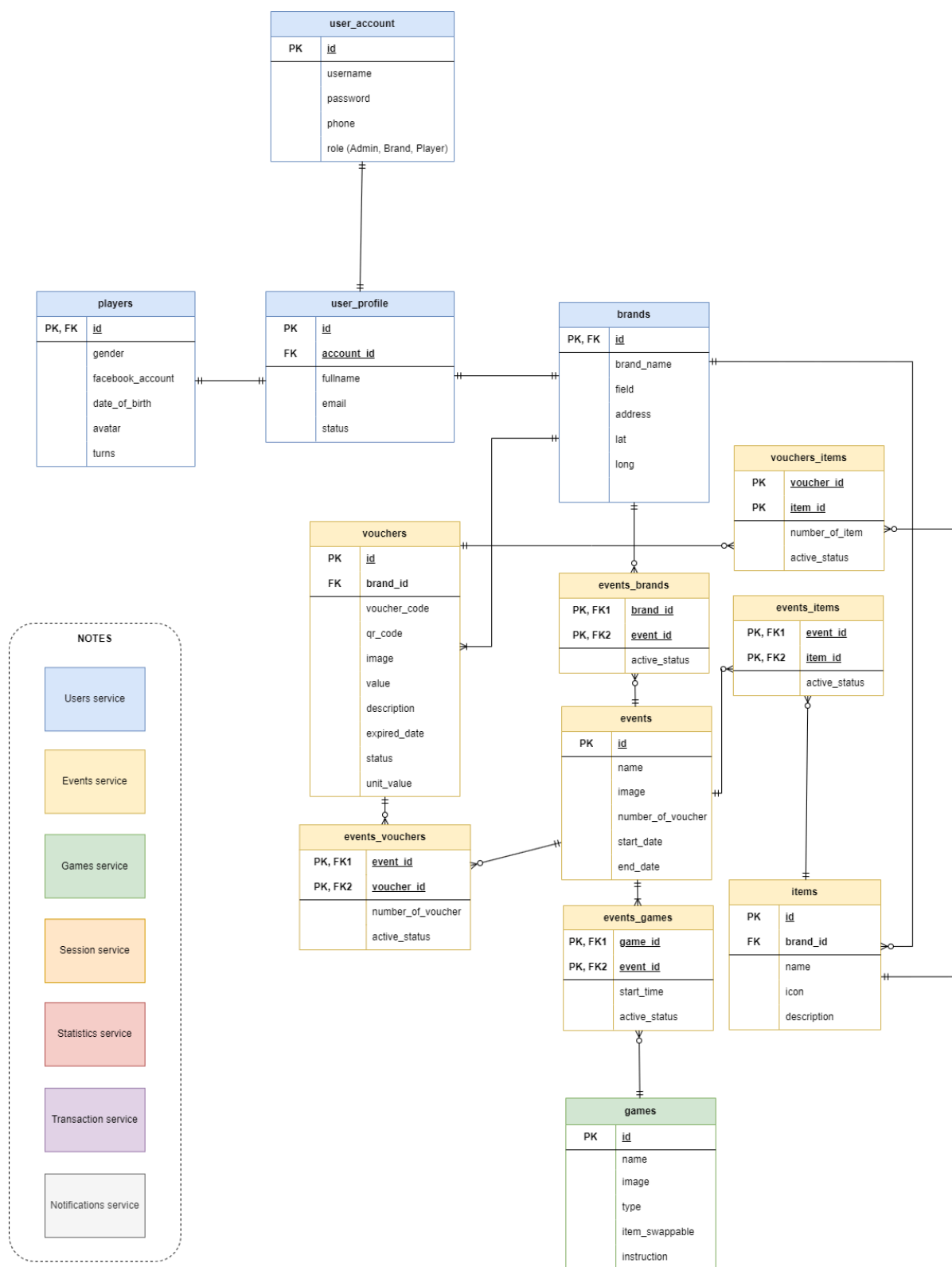
1. Cơ sở dữ liệu.....	3
2. Kiến trúc hệ thống.....	9
3. Các Thay Đổi Kiến Trúc Để Đáp Ứng Nhu Cầu Tăng Trưởng.....	16
4. Áp Dụng Design Patterns.....	23
5. Phần ứng dụng.....	32
6. Các phần nâng cao khác.....	48
7. Kết Luận và Đánh Giá.....	48
8. Tài Liệu Tham Khảo.....	50



VOU - Marketing with Real-time Games	Phiên bản: 1.0
Thiết kế kiến trúc	Ngày: 08/09/2024

1. Cơ sở dữ liệu

1. Users, Events, Games Services: thiết kế dữ liệu trên MySQL



1.1. Mô tả

a. Users service

- user_account: Quản lý thông tin tài khoản người dùng với các vai trò khác nhau như Admin, Brand, và Player.
- user_profile: Lưu trữ thông tin chi tiết của người dùng.
- players: Thông tin khác về người chơi, bao gồm giới tính, tài khoản Facebook, và số lượt chơi.
- brands: Lưu trữ thông tin khác của các thương hiệu.

b. Events Service

- vouchers: Quản lý thông tin voucher, bao gồm mã, hình ảnh, giá trị, và ngày hết hạn.
- vouchers_items: Liên kết voucher với các mục, quản lý số lượng mục liên quan.
- events: Quản lý thông tin sự kiện như tên, hình ảnh, số lượng voucher, ngày bắt đầu và kết thúc.
- events_brands: Liên kết sự kiện với các thương hiệu.
- events_vouchers: Liên kết sự kiện với voucher, quản lý số lượng và trạng thái hoạt động.
- events_games: Liên kết sự kiện với trò chơi, quản lý thời gian bắt đầu và trạng thái hoạt động.
- events_items: Liên kết sự kiện với các vật phẩm, quản lý trạng thái hoạt động.
- items: Quản lý thông tin về các vật phẩm có thể được trao đổi trong trò chơi.

c. Games service

- games: Quản lý thông tin về các trò chơi, bao gồm tên, hình ảnh, loại, và hướng dẫn.

1.2. Chiến lược mở rộng

- Database được thiết kế cho phép mở rộng số lượng profile, số lượng role của một user. Việc tách account riêng so với profile đảm bảo có thể bảo mật tốt với nhiều người cùng truy cập.
- Cho phép có thêm nhiều game, event khác được thêm vào trong hệ thống.
- Đảm bảo có thể cho phép nhiều người cùng chơi game với trải nghiệm tốt.

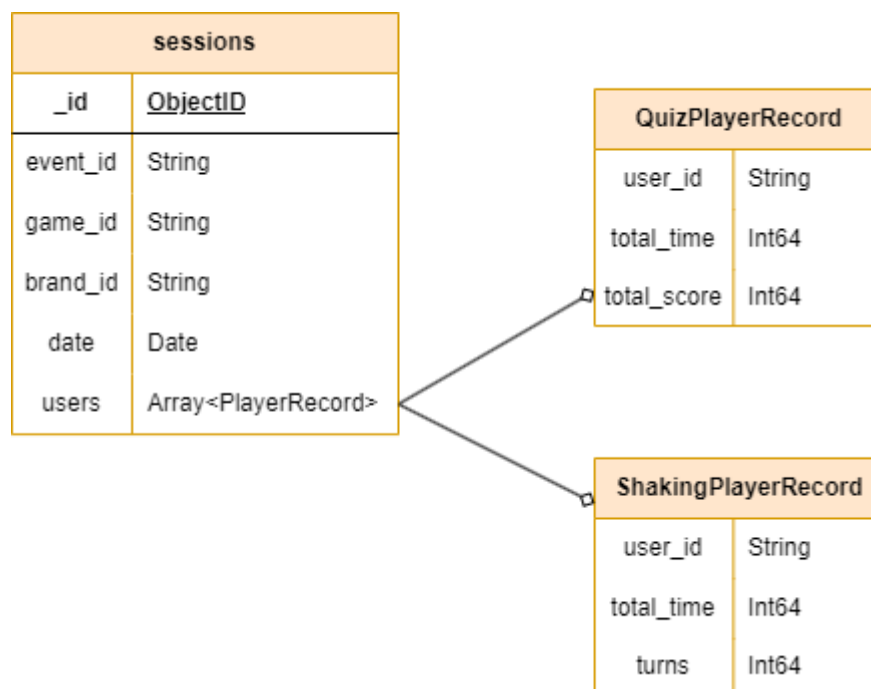
2. Session service: thiết kế dữ liệu trên MongoDB và Redis

2.1. Mô tả

a. Dữ liệu trên MongoDB

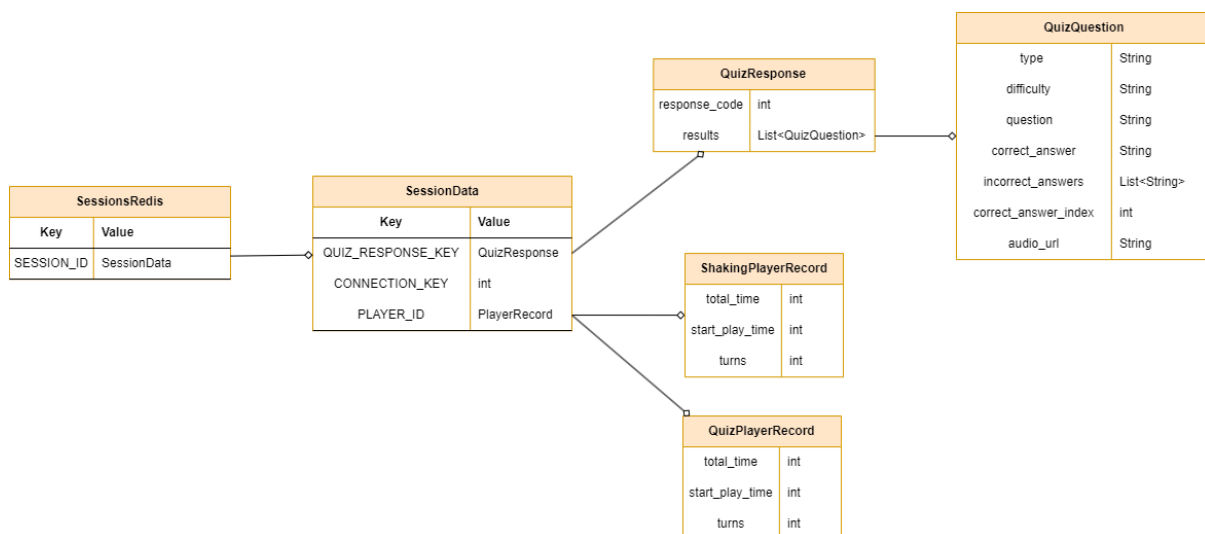
- Chứa thông tin về các phiên chơi, bao gồm event_id, game_id, brand_id, date, và danh sách users dưới dạng Array<PlayerRecord>.

- QuizPlayerRecord và ShakingPlayerRecord: Lưu trữ thông tin chi tiết của người chơi như user_id, total_time, total_score, và turns.



b. Dữ liệu trên redis

- SessionsRedis: Lưu trữ dữ liệu phiên dưới dạng cặp Key-Value, với SESSION_ID là khóa và SessionData là giá trị.
- SessionData: Bao gồm các khóa như QUIZ_RESPONSE_KEY, CONNECTION_KEY, và PLAYER_ID.
- QuizResponse: Chứa mã phản hồi và danh sách câu hỏi.
- QuizQuestion: Bao gồm thông tin chi tiết về câu hỏi như type, difficulty, question, correct_answer, và incorrect_answers.



- Sử dụng Redis làm bộ nhớ đệm để lưu trữ các phiên hoạt động gần đây, giảm tải cho MongoDB và cải thiện thời gian phản hồi.

3. Statistics service: thiết kế dữ liệu trên MongoDB

3.1. Mô tả



- Cho biết số lượng voucher, item mà người chơi sở hữu; số lượng event, game mà người đó tham gia.
- Thiết kế này cho phép theo dõi và quản lý thông tin liên quan đến người chơi, voucher, mục, trò chơi và sự kiện, hỗ trợ việc thống kê và phân tích dữ liệu trong hệ thống VOU.

4. Transactions service: thiết kế dữ liệu trên MongoDB

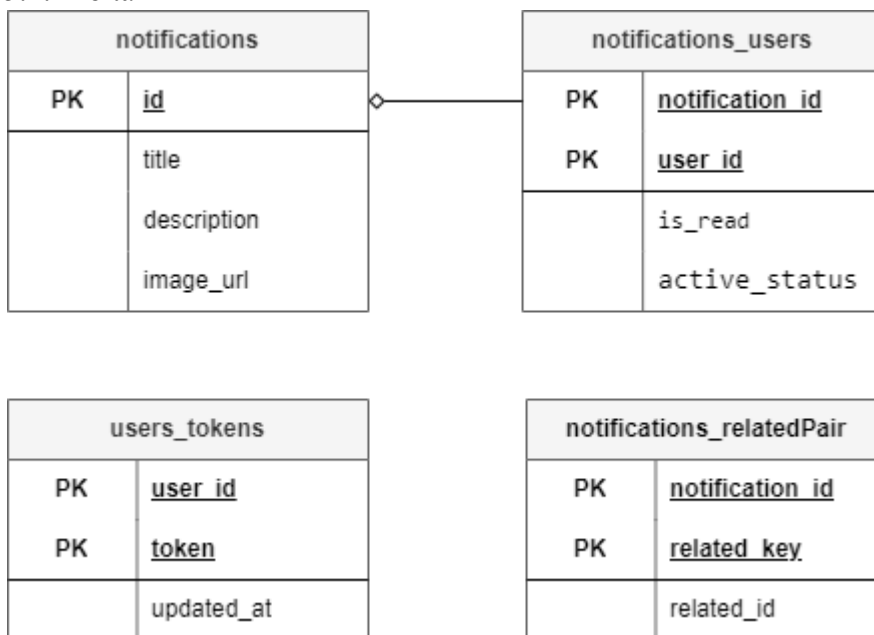
4.1. Mô tả

item_shared_transactions		item_received_transactions		voucher_conversion_transactions		voucher_used_transactions	
_id	ObjectID	_id	ObjectID	_id	ObjectID	_id	ObjectID
player_id	String	player_id	String	player_id	String	player_id	String
recepient_id	String	recepient_id	String	recepient_id	String	recepient_id	String
item_id	String	item_id	String	event_id	String	event_id	String
game_id	String	game_id	String	Array<Item>	String	Array<Item>	String
transaction_date	Date	transaction_date	Date	transaction_date	Date	transaction_date	Date
quantity	Int	quantity	Int	quantity	Int	quantity	Int
transaction_type	String	transaction_type	String	transaction_type	String	transaction_type	String

- Thiết kế này cho phép theo dõi và quản lý các giao dịch liên quan đến việc chia sẻ, nhận các voucher, chuyển đổi và sử dụng các items trong hệ thống VOU.

5. Notifications service: thiết kế dữ liệu trên Firestore

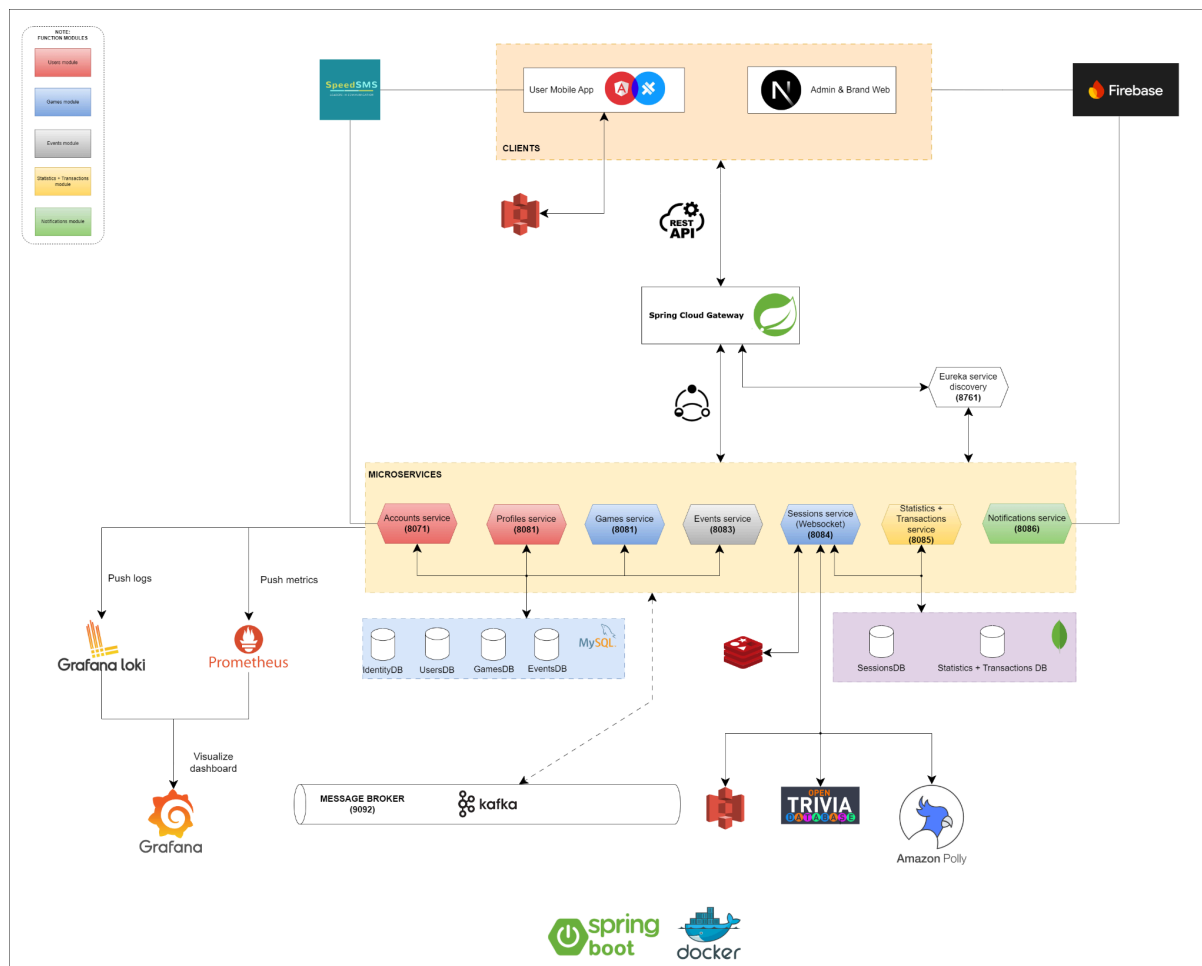
5.1. Mô tả



- Thiết kế này cho phép quản lý thông báo, theo dõi trạng thái đọc thông báo của người dùng và quản lý mã thông báo để gửi thông báo hiệu quả.

2. Kiến trúc hệ thống

1. Sơ đồ minh họa



2. Mô tả

- Hệ thống được thiết kế theo kiến trúc microservices, đảm bảo khả năng mở rộng, chịu tải cao và xử lý real-time. Khi người dùng gửi yêu cầu API từ ứng dụng di động hoặc web, yêu cầu này sẽ đi qua Spring Cloud Gateway để thực hiện xác thực và định tuyến. Eureka Service Discovery giúp Gateway tự động phát hiện và định tuyến các yêu cầu đến các dịch vụ hoạt động, đảm bảo tính linh hoạt khi thêm hoặc xóa dịch vụ.
- Các dịch vụ microservices được chia thành nhiều cụm chức năng khác nhau. Cụm chức năng Users bao gồm các dịch vụ quản lý tài khoản và hồ sơ người

dùng. SpeedSMS được sử dụng để gửi OTP xác thực số điện thoại. Cụm chức năng Events sử dụng Kafka để xử lý luồng dữ liệu thời gian thực, với Zookeeper quản lý metadata của cụm Kafka.

- Cụm chức năng Games tận dụng Open Trivia Database để lấy câu hỏi trắc nghiệm và Amazon Polly để chuyển đổi văn bản thành giọng nói, tạo ra file audio cho MC ảo, tối ưu tốc độ bằng việc sử dụng Redis. Cụm chức năng Statistics + Transactions sử dụng MongoDB để lưu trữ dữ liệu thời gian thực, giúp cải thiện hiệu năng truy xuất.
- Cụm chức năng Notifications sử dụng Firebase để gửi thông báo đến các thiết bị di động và ứng dụng web. Firestore lưu trữ và đồng bộ hóa dữ liệu giữa các ứng dụng khách và máy chủ.
- Hệ thống lưu trữ dữ liệu trên AWS S3, MySQL, và MongoDB, đảm bảo việc quản lý dữ liệu có cấu trúc và phi cấu trúc hiệu quả. Prometheus và Grafana được sử dụng để giám sát và trực quan hóa dữ liệu hệ thống, giúp theo dõi hiệu suất và phát hiện sự cố kịp thời.
- Cuối cùng, Docker đảm bảo việc triển khai và quản lý các container, giúp hệ thống dễ dàng mở rộng và duy trì tính nhất quán. Hệ thống này được thiết kế để đáp ứng nhu cầu xử lý hàng ngàn người dùng đồng thời, đảm bảo tính bảo mật và độ tin cậy cao.

3. Các thành phần chính

3.1. Spring Cloud Gateway

- Tất cả các API được gửi đến từ front-end cần phải được đi qua Gateway để thực hiện quá trình Authentication.
- Circuit breaker: Circuit breaker giúp ngăn chặn tình trạng quá tải hệ thống bằng cách dừng các yêu cầu đến một dịch vụ đang gặp sự cố hoặc chậm. Nó cải thiện khả năng chịu lỗi bằng cách phản hồi nhanh với một thông báo dự phòng hoặc lỗi thay vì chờ đợi dịch vụ gặp sự cố.
- Eureka service discovery: Eureka service discovery trong hệ thống

microservices giúp API gateway tự động phát hiện và định tuyến yêu cầu đến các dịch vụ đang hoạt động mà không cần cấu hình thủ công. Điều này giúp hệ thống linh hoạt hơn trong việc thêm, xóa, hoặc di chuyển các dịch vụ.

3.2. Cụm chức năng Users

- Kiến trúc phần Users được chia thành hai phần chính: Account service và Profile service. Account service chịu trách nhiệm về việc xác thực (authentication), trong khi Profile service lưu trữ thông tin người dùng.
- Account service tách riêng thông tin người dùng như username, password và roles để thực hiện nhiệm vụ xác thực và mã hóa JWT. Điều này giúp hệ thống dễ dàng mở rộng và bảo mật hơn trong môi trường microservices lớn.
- Profile service sử dụng chiến lược kế thừa dạng Joined để đảm bảo tính toàn vẹn dữ liệu và khả năng mở rộng. Các vai trò như Player, Admin, và Brand được kế thừa từ User, giúp dễ dàng mở rộng và quản lý.
- SpeedSMS: SpeedSMS được sử dụng để gửi OTP đến số điện thoại của người chơi nhằm xác thực số điện thoại khi đăng ký tài khoản, đảm bảo tính bảo mật.

3.3. Cụm chức năng Events

- Quản lý các sự kiện, bao gồm tạo, cập nhật, và xóa sự kiện. Khi tạo sự kiện, nhân hàng tạo sự kiện sẽ lựa chọn nhân hàng tham gia bằng cách chọn email tương ứng của nhân hàng đó, đồng thời có thể chọn cùng với các loại voucher, vật phẩm cần thiết để đổi voucher và game cho sự kiện. Khi tạo thành công một sự kiện thì sẽ tự động gửi các thông tin game cần thiết cho Sessions service, lúc này sessions service chạy tự động cài đặt game vào thời điểm nó cần diễn ra. Việc gửi thông tin từ Events Service sang Sessions Service sẽ thông qua Kafka.
- Kafka: Đảm bảo giao tiếp giữa các microservices thông qua việc truyền tải thông điệp. Ở đây sử dụng cho việc giao tiếp giữa Events Service và Sessions Service như đề cập ở trên.
- Zookeeper: Quản lý và duy trì trạng thái của các broker Kafka.

3.4. Cụm chức năng Games

- Phần cụm chức năng games gồm hai service là games và sessions. Trong đó service games giúp admin có thể điều chỉnh cấu hình, thông tin của 1 game còn sessions sẽ đảm nhận việc thiết lập và triển khai các phiên chơi game, cho phép người dùng tham gia chơi và nhận vật phẩm.
- Redis: là một hệ quản trị cơ sở dữ liệu mã nguồn mở, lưu trữ dữ liệu dạng key-value (cặp khóa-giá trị) trong bộ nhớ (in-memory). Redis thường được sử dụng như một bộ nhớ đệm (cache), một hàng đợi tin nhắn (message broker), hoặc một cơ sở dữ liệu NoSQL với hiệu năng cao.
- Kafka: Đảm bảo giao tiếp giữa các microservices thông qua việc truyền tải thông điệp. Ở đây sử dụng cho việc giao tiếp giữa Events Service và Sessions Service như đề cập ở trên.

3.5. Cụm chức năng Statistics - Transactions

- Phần Statistics dùng để thống kê những thông tin cần thiết (transaction logs,...) và Transactions sẽ dùng để thực hiện các transaction và lưu vào database.
- Ở phần Transactions ở đây sử dụng kiến trúc chính là Factory Method and Strategy Pattern.
- Kafka: Đảm bảo giao tiếp giữa các microservices thông qua việc truyền tải thông điệp. Ở đây Kafka được sử dụng ở cả 2 vai trò là Consumer và Producer.
- Với vai trò Consumer, Kafka được sử dụng để nhận thông điệp từ Sessions Service, các thông điệp đó là thông tin về giao dịch nhận quà mà người chơi được nhận sau khi chơi game xong và thông tin thứ 2 là về id của một sự kiện khi sự kiện đó gần diễn ra.
- Với vai trò Producer, Kafka được sử dụng để gửi thông điệp qua cho Notifications Service để sau đó Notifications Service sẽ lập tức gửi thông báo đến cho những người dùng phù hợp.

3.6. Cụm chức năng Notifications

- Firebase Cloud Messaging (FCM): là dịch vụ thông báo của Firebase, cho phép bạn gửi thông báo và tin nhắn đến các thiết bị di động và ứng dụng web. FCM hỗ trợ gửi thông báo đa nền tảng, bao gồm Android, iOS, và web.
- Firestore: là một cơ sở dữ liệu NoSQL trên nền tảng đám mây do Google cung cấp thông qua Firebase. Firestore được thiết kế để lưu trữ và đồng bộ hóa dữ liệu giữa các ứng dụng khách và máy chủ, và nó hỗ trợ cả ứng dụng web và di động.
- Kafka: Ở đây Kafka được sử dụng với vai trò là Consumer, Kafka được sử dụng để nhận thông điệp từ mọi Services khác, thông điệp đó là thông tin về những thuộc tính cần thiết của thông báo và danh sách tài khoản sẽ cần nhận thông báo đó.

3.7. Các thành phần chung trong hệ thống

3.7.1. Thành phần lưu trữ dữ liệu hệ thống:

- AWS S3: Lưu trữ và quản lý các tệp tin, hình ảnh, và các tài nguyên tĩnh.
- MySQL: Lưu trữ dữ liệu có cấu trúc, ít truy xuất hơn, gồm các thông tin liên quan đến thông tin Accounts, Profiles của các Users, Events và Games.
- MongoDB: Lưu trữ dữ liệu phi cấu trúc hoặc bán cấu trúc, có tần suất truy xuất cao như Game sessions, Statistics và Notifications.

3.7.2. Thành phần quản lý hệ thống:

- Prometheus: là một hệ thống giám sát và cảnh báo mã nguồn mở, được phát triển ban đầu tại SoundCloud và hiện nay là một dự án thuộc tổ chức Cloud Native Computing Foundation (CNCF). Prometheus được thiết kế đặc biệt để giám sát các hệ thống phân tán, với khả năng thu thập và lưu trữ số liệu thời gian thực (time-series data).
- Grafana: là một công cụ mã nguồn mở để trực quan hóa và giám sát dữ liệu thu thập từ nhiều nguồn khác nhau. Grafana thường được sử dụng cùng với

Prometheus để tạo ra các bảng điều khiển (dashboard) và biểu đồ thời gian thực, giúp các đội ngũ kỹ thuật giám sát sức khỏe và hiệu suất của hệ thống.

- Grafana loki: là một hệ thống ghi nhận log có thể mở rộng, được thiết kế để tích hợp chặt chẽ với Grafana. Loki tập trung vào việc quản lý và truy vấn logs, với cách tiếp cận tương tự như Prometheus, nhưng dành riêng cho logs thay vì số liệu.

3.7.3. Thành phần vận hành hệ thống:

- Docker: Đóng gói và chạy các dịch vụ của hệ thống dưới dạng các container độc lập, giúp đảm bảo tính nhất quán và dễ triển khai.

4. Các tiêu chí hiệu năng, khả năng mở rộng, độ tin cậy, và bảo mật

4.1. Hiệu năng

- Hệ thống sử dụng Redis caching để tăng tốc độ truy xuất dữ liệu, đặc biệt là khi truy cập vào cơ sở dữ liệu Open Trivia. Redis giúp giảm tải cho các dịch vụ bằng cách lưu trữ tạm thời các dữ liệu thường xuyên được truy vấn.
- Ngoài ra, Kafka được sử dụng như một message broker, hỗ trợ kiểu queue để tối ưu hóa giao tiếp giữa các microservices, giúp cải thiện hiệu năng tổng thể của hệ thống.

4.2. Khả năng mở rộng

- Kiến trúc microservices cho phép hệ thống dễ dàng mở rộng bằng cách thêm các dịch vụ mới hoặc mở rộng các dịch vụ hiện có mà không ảnh hưởng đến toàn bộ hệ thống.
- Load balancing và service registry (Eureka) cũng được áp dụng để hỗ trợ khả năng mở rộng, đảm bảo rằng các dịch vụ có thể xử lý lưu lượng lớn và tăng trưởng người dùng.

4.3. Độ tin cậy

- Hệ thống sử dụng circuit breaker pattern để đảm bảo độ tin cậy khi giao tiếp giữa các dịch vụ thông qua Spring Cloud Gateway.

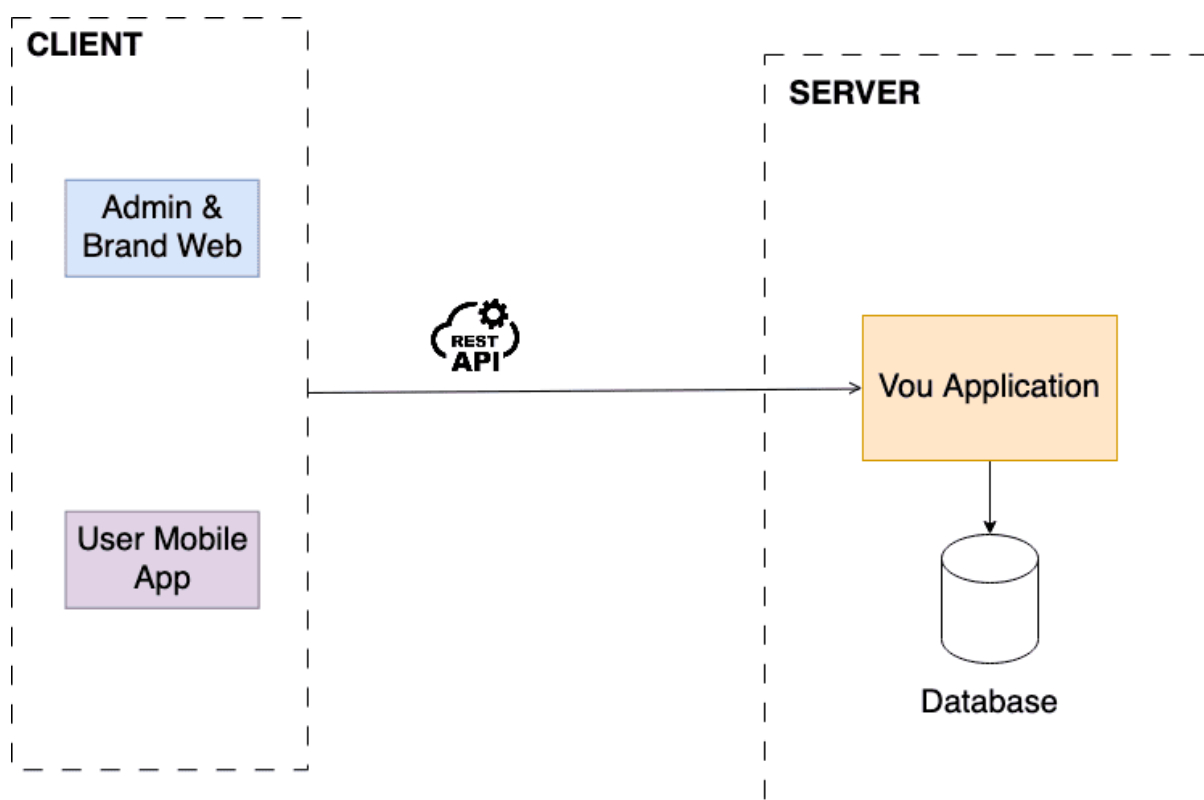
- Circuit breaker giúp ngăn chặn các lỗi lan rộng trong hệ thống bằng cách phát hiện và cô lập các dịch vụ gặp sự cố. Điều này đảm bảo rằng hệ thống vẫn hoạt động ổn định ngay cả khi một số dịch vụ bị lỗi.

4.4. Bảo mật

- Bảo mật được tăng cường thông qua việc sử dụng JSON Web Tokens (JWT) tại gateway để xác thực người dùng.
- Authentication được tách riêng cho identity service và gateway, trong khi các microservices như users và events sẽ decode JWT để thực hiện authorization. Điều này đảm bảo rằng chỉ những người dùng được phép mới có thể truy cập vào các dịch vụ cụ thể.

3. Các Thay Đổi Kiến Trúc Để Đáp Ứng Nhu Cầu Tăng Trưởng

3.1. Phiên bản 1



3.1.1. Mục tiêu

VOU - Marketing with Real-time Games	Phiên bản: 1.0
Thiết kế kiến trúc	Ngày: 08/09/2024

- Thiết kế hệ thống ở mức high level từ đó có cái nhìn tổng quát về các thành phần chính trong hệ thống bao gồm client, server và database.
- Ở phiên bản này, người dùng gửi yêu cầu đọc hoặc ghi dữ liệu: Người dùng có thể yêu cầu lấy thông tin từ hệ thống hoặc thêm thông tin mới vào hệ thống.

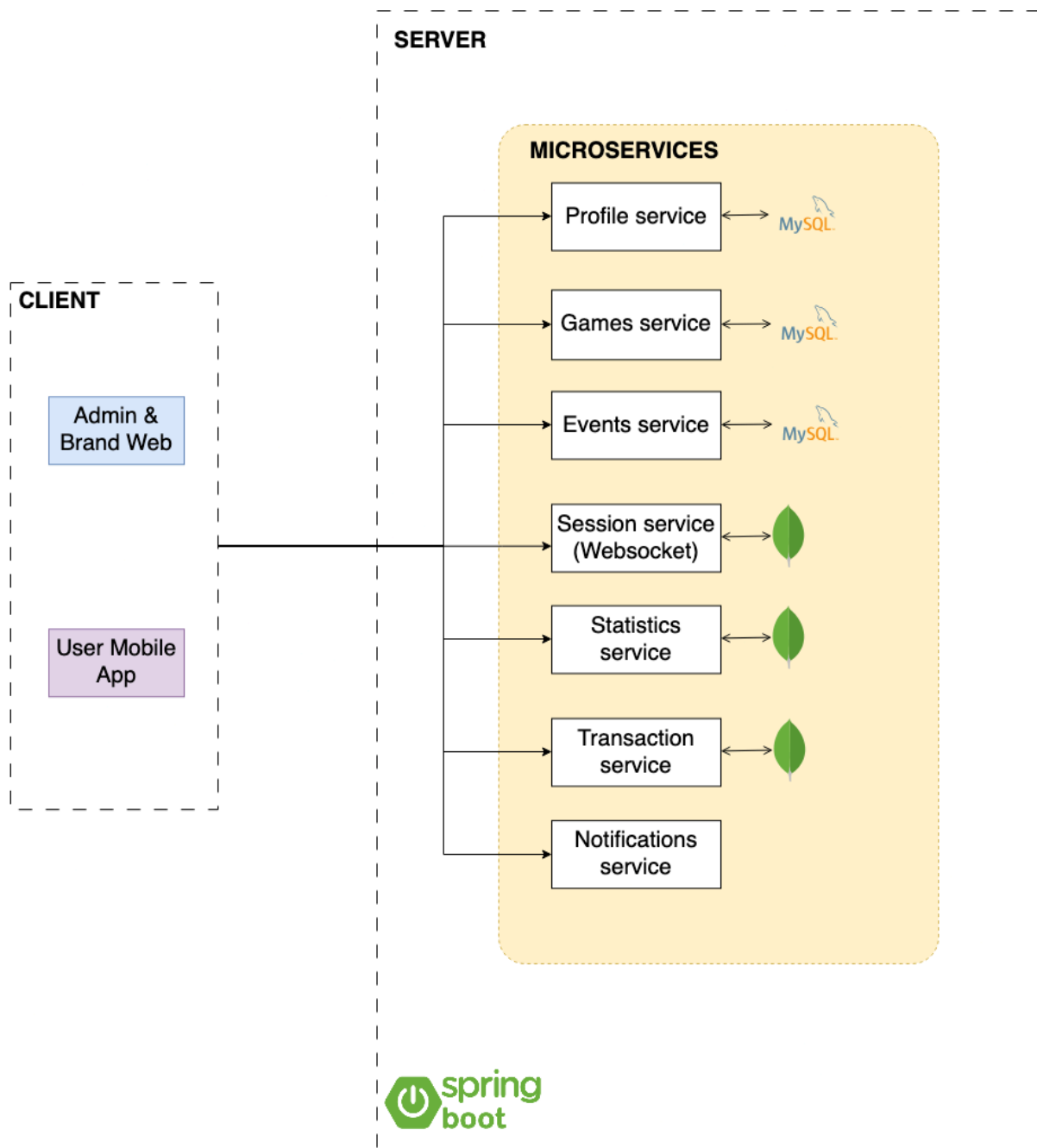
3.1.2. Các thành phần thay đổi:

- Client: là thành phần được sử dụng bởi người dùng gồm player, brand và admin. Trong đó brand và admin sẽ sử dụng web để truy cập và tương tác với hệ thống, còn player sẽ truy cập thông qua điện thoại di động.
- Server: là máy chủ của hệ thống, chịu trách nhiệm nhận yêu cầu từ phía client, xử lý logic nghiệp vụ và trả về phản hồi tương ứng.
- Database: là cơ sở dữ liệu của hệ thống, nơi sẽ lưu trữ toàn bộ dữ liệu cần thiết.

3.1.3. Giải thích lý do:

- Kiến trúc monolith vô cùng đơn giản, dễ thiết kế và cài đặt giúp việc phát triển hệ thống trở nên nhanh chóng và đáp ứng được các yêu cầu của người dùng.

3.2. Phiên bản 2



3.2.1. Mục tiêu:

- Áp dụng kiến trúc microservices vào hệ thống để có thể tận dụng được các ưu điểm của mô hình này và khắc phục những vấn đề của mô hình monolith dựa trên mô tả của đồ án, nhóm em chọn spring làm công cụ phát triển vì đây là

một framework java mạnh mẽ hỗ trợ phát triển các ứng dụng microservice một cách tiện lợi, nhanh chóng.

- Tách nhỏ logic nghiệp vụ một cách phù hợp thành các domain nhỏ tương ứng với các service để handle từng nhiệm vụ cụ thể thay vì gom nhóm toàn bộ.

3.2.2. Các thành phần thay đổi:

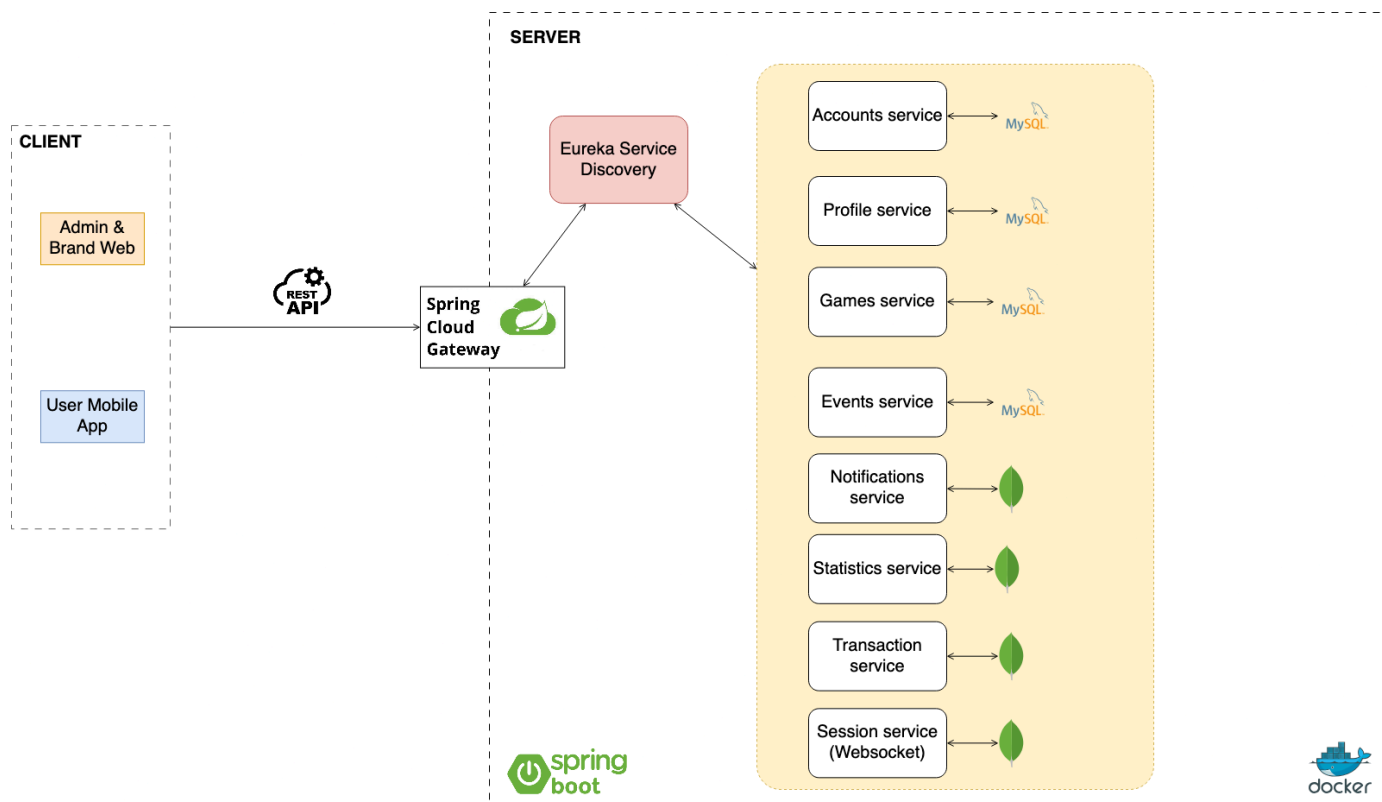
- Profile service: Xử lý logic nghiệp vụ liên quan đến việc quản lý tài khoản của người dùng
- Games service: Xử lý logic nghiệp vụ liên quan đến cấu hình game gồm 2 loại là quiz game và shaking game.
- Events service: Xử lý logic nghiệp vụ liên quan đến sự kiện khuyến mãi
- Sessions service: Xử lý logic nghiệp vụ liên quan đến tạo, thiết lập một phiên chơi game cho player, cho phép nhiều người cùng tham gia chơi game.
- Statistics service: Xử lý logic nghiệp vụ liên quan đến thống kê, quản lý dữ liệu người dùng, trò chơi, sự kiện, ...
- Transaction service: Xử lý logic nghiệp vụ liên quan đến các giao dịch, gửi, nhận quà, đổi hoặc sử dụng vouchers, ...
- Notification service: Xử lý logic nghiệp vụ liên quan đến thông báo

3.2.3. Giải thích lý do:

- Việc sử dụng microservices mang lại nhiều ưu điểm sau đây:
 - Mỗi microservice được phát triển, triển khai, và mở rộng độc lập. Điều này giúp các nhóm phát triển có thể làm việc song song mà không cần phải đợi lẫn nhau, tăng hiệu suất và tốc độ phát triển.
 - Mỗi microservice có thể được xây dựng bằng các ngôn ngữ lập trình, framework và công nghệ khác nhau phù hợp nhất với từng yêu cầu cụ thể. Điều này giúp tận dụng tốt hơn các công cụ và công nghệ tối ưu cho từng chức năng.
 - Khi một microservice gặp sự cố, nó không ảnh hưởng đến toàn bộ hệ thống. Các phần còn lại của hệ thống có thể tiếp tục hoạt động bình thường, giúp tăng khả năng chịu lỗi và duy trì dịch vụ liên tục.
 - Với microservices, ta có thể mở rộng riêng lẻ các thành phần của hệ

thông dựa trên nhu cầu mà không cần phải mở rộng toàn bộ từ đó tiết kiệm chi phí rất nhiều.

3.3. Phiên bản 3



3.3.1. Mục tiêu:

- Đơn giản hóa quá trình giao tiếp giữa client và các services nằm trong server thông qua API Gateway và Service Discovery.
- Sử dụng docker để xây dựng các container cho từng service một cách hiệu quả để đẩy nhanh quá trình phát triển.

3.3.2. Các thành phần thay đổi:

- API Gateway (Spring Cloud Gateway) là một phần của bộ công cụ Spring Cloud, cung cấp một cách tiếp cận hiện đại và dễ dàng để xây dựng và cấu hình API Gateway cho các ứng dụng microservices. Nó được thiết kế để thay thế hoặc bổ sung cho Zuul (một API Gateway khác trong hệ sinh thái Spring Cloud), và cung cấp nhiều tính năng mạnh mẽ hơn nhờ sử dụng các công nghệ hiện đại như WebFlux và Project Reactor của Spring.

- Eureka Service Discovery là một thành phần trong hệ sinh thái Spring Cloud, được phát triển bởi Netflix, dùng để quản lý việc đăng ký và tìm kiếm các microservices trong một hệ thống phân tán. Nó đóng vai trò như một "sổ địa chỉ" trung tâm, giúp các microservices tự động tìm thấy nhau và giao tiếp một cách linh hoạt, không cần phải cấu hình thủ công địa chỉ của từng dịch vụ.
- Docker là một nền tảng mã nguồn mở được sử dụng để tự động hóa quá trình triển khai, quản lý và chạy ứng dụng bên trong các "container". Docker giúp đóng gói các ứng dụng cùng với tất cả các phụ thuộc của chúng (như thư viện, công cụ, runtime, file cấu hình, v.v.) vào một đơn vị tiêu chuẩn (container), giúp việc triển khai và chạy ứng dụng trên các môi trường khác nhau trở nên nhất quán và dễ dàng hơn.

3.3.3. Giải thích lý do:

- Lý do sử dụng API Gateway: Nó cung cấp một điểm tiếp cận duy nhất cho tất cả các yêu cầu từ client đến các microservices. Điều này giúp quản lý lưu lượng và bảo mật một cách dễ dàng và hiệu quả hơn. Phía client sẽ không phải giao tiếp trực tiếp với các services mà sẽ đều thông qua gateway. Bên cạnh đó gateway còn hỗ trợ xác thực (authentication), ủy quyền (authorization), bảo vệ chống DDoS, giới hạn tốc độ (rate limiting), và quản lý session để bảo vệ hệ thống microservices.
- Lý do sử dụng Service Discovery:
 - Trong một hệ thống microservices, các dịch vụ thường thay đổi địa chỉ IP hoặc số cổng khi được triển khai lại hoặc mở rộng. Eureka giúp tự động đăng ký và khám phá các dịch vụ, giúp các microservices tìm thấy nhau mà không cần cấu hình thủ công.
 - Eureka có thể kết hợp với các cơ chế cân bằng tải như Ribbon để phân phối yêu cầu đồng đều giữa các instance của cùng một dịch vụ, giúp tăng cường độ tin cậy và khả năng chịu lỗi.
 - Khi một dịch vụ gặp sự cố hoặc không hoạt động, Eureka sẽ loại bỏ dịch vụ đó khỏi danh sách các dịch vụ khả dụng. Điều này giúp giảm thiểu lỗi khi giao tiếp giữa các microservices.
 - Eureka cung cấp một bảng điều khiển để theo dõi trạng thái của các dịch vụ trong hệ thống, giúp quản trị viên dễ dàng quản lý và phát hiện

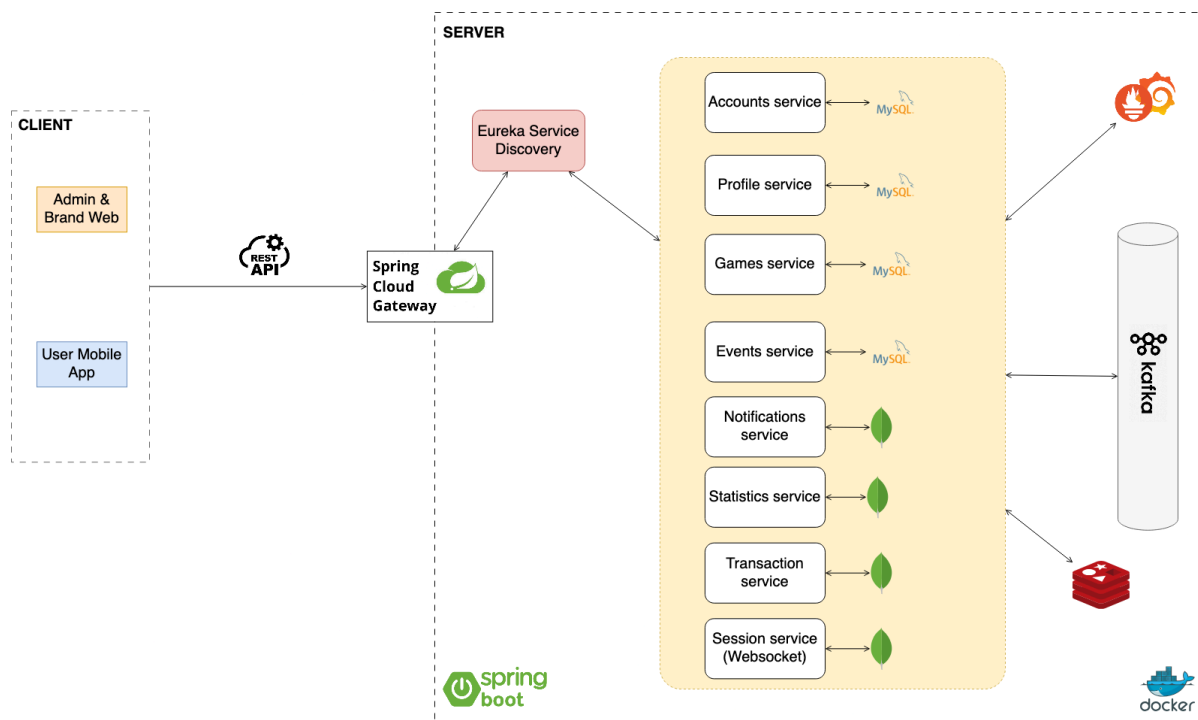


VOU - Marketing with Real-time Games	Phiên bản: 1.0
Thiết kế kiến trúc	Ngày: 08/09/2024

các vấn đề trong hệ thống.

- Lý do sử dụng Docker:
 - Docker đảm bảo rằng ứng dụng sẽ hoạt động giống nhau bất kể môi trường nào, từ máy tính cá nhân của lập trình viên đến môi trường staging hay production. Điều này giúp giảm thiểu các lỗi phát sinh do khác biệt giữa các môi trường.
 - Docker container có thể được khởi động và tắt rất nhanh, cho phép triển khai nhanh chóng các phiên bản mới của dịch vụ và giảm thời gian chết (downtime) khi cập nhật.
 - Docker cho phép dễ dàng nhân bản và mở rộng các microservices, đồng thời tối ưu hóa việc sử dụng tài nguyên phần cứng nhờ các container chia sẻ cùng một nhân hệ điều hành.
 - Một lý do khác nữa mà nhóm em sử dụng Docker đó là vì nhóm chia thành 2 team là Backend và Frontend do đó team Frontend sẽ sử dụng Docker để chạy container do phía Backend build ra thay vì phải tự chạy và host server Backend.

3.4. Phiên bản 4



3.4.1. Mục tiêu:

- Cải thiện quá trình giao tiếp giữa các service bằng việc sử dụng giao tiếp bất đồng bộ thông qua message broker
- Tối ưu tốc độ truy xuất dữ liệu và giảm tải cho cơ sở dữ liệu chính bằng Redis
- Giám sát và quản lý hệ thống bằng Grafana
- Tự động hóa quá trình triển khai, mở rộng và quản lý các container Docker

3.4.2. Các thành phần thay đổi:

- Kafka là một nền tảng streaming dữ liệu phân tán mã nguồn mở, được thiết kế để xử lý và truyền tải các luồng dữ liệu lớn giữa các hệ thống. Kafka hoạt động như một hệ thống message broker, cho phép các microservices giao tiếp với nhau một cách hiệu quả và đáng tin cậy.
- Redis là một cơ sở dữ liệu lưu trữ key-value trong bộ nhớ, mã nguồn mở, cung cấp khả năng lưu trữ dữ liệu nhanh chóng với độ trễ thấp. Redis thường được sử dụng để lưu trữ và truy xuất dữ liệu tạm thời, như bộ nhớ cache hoặc quản lý phiên người dùng.

3.4.3. Giải thích lý do:

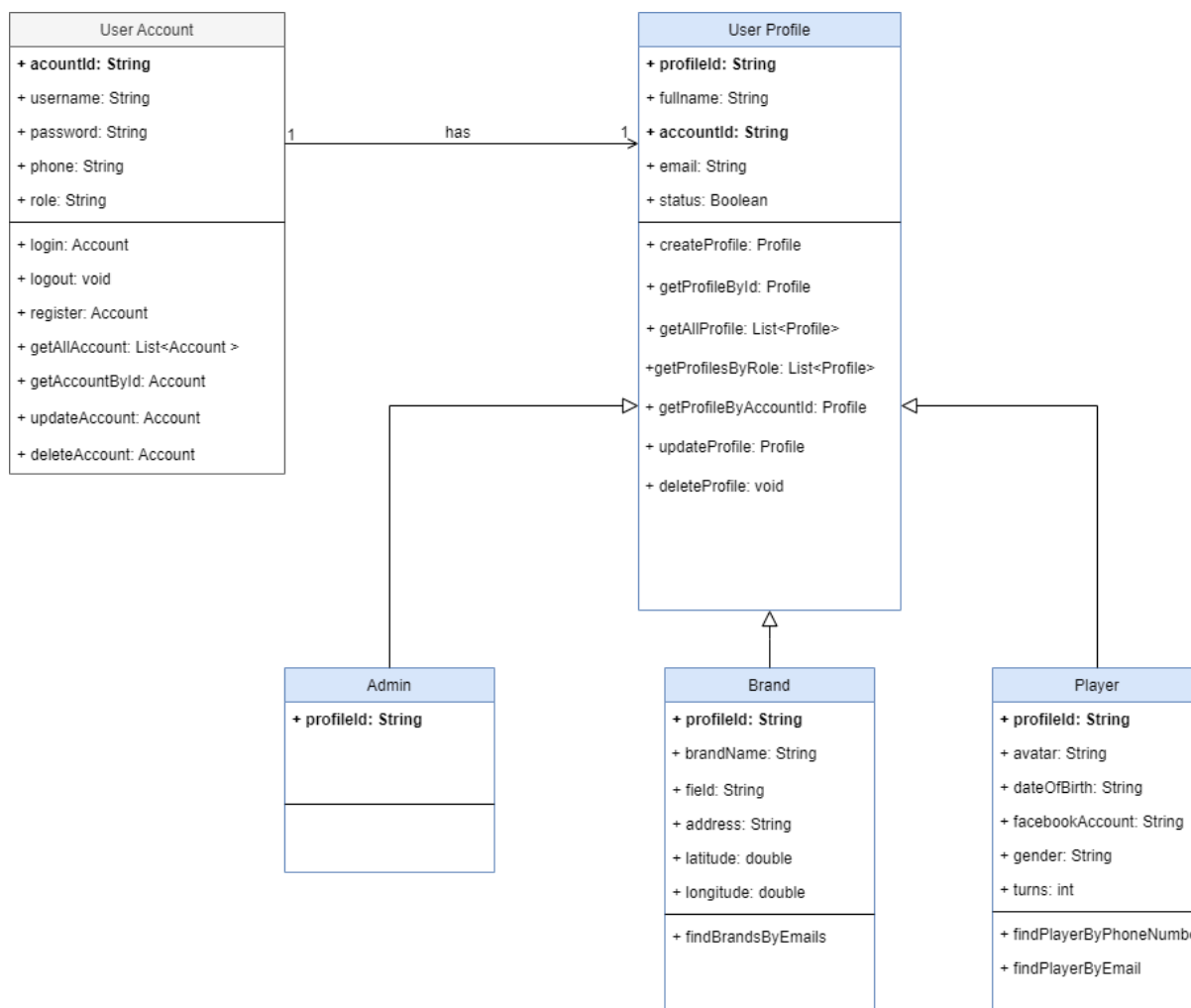
- Lý do sử dụng Kafka: Nhóm em chọn sử dụng Kafka thay vì các message

broker khác vì Kafka hỗ trợ phân tán và mở rộng dễ dàng, cho phép xử lý hàng triệu tin nhắn mỗi giây mà không làm giảm hiệu suất. Điều này làm cho Kafka phù hợp với các hệ thống yêu cầu xử lý dữ liệu quy mô lớn và lưu lượng cao, đặc biệt ở trong hệ thống VOU thì có rất nhiều nơi cần phải giao tiếp bất đồng bộ thay vì sử dụng đồng bộ như gửi thông báo, thiết lập game, cập nhật vật phẩm cho người dùng,

- Lý do sử dụng Redis: Nhóm em cần caching thông tin câu hỏi sau khi fetch từ Open Trivia Database và phục vụ cho việc cập nhật thông tin trạng thái game một cách nhanh chóng trong thời gian thực.

4. Áp Dụng Design Patterns

1. Users service: Mẫu composite
 - 1.1. Minh họa và mô tả:
 - 1.1.1. Minh họa



1.1.2. Mô tả

- Mẫu composite được áp dụng trong việc xây dựng User Profile với các role Admin, Brand, Player. Dựa vào giá trị trong field “role” được gửi đến từ các request đến User, có thể xác định được loại User tương ứng để từ đó đảm bảo được việc thực hiện các chức năng CRUD cho tất cả các loại User đều chỉ gọi đến cùng một api của User Profile. Điều này giúp giảm bảo được việc tái sử dụng code và tối ưu database.

1.2. Ý nghĩa và lợi ích:

1.2.1. Tránh trùng lặp mã nguồn (DRY - Don't Repeat Yourself):

- Mẫu Composite cho phép tổ chức các loại người dùng khác nhau (như Admin, Brand, Player) trong cùng một cấu trúc kế thừa. Điều này giúp tránh việc viết lại mã CRUD cho từng loại người dùng, vì các thao tác chung có thể được xử lý trong lớp cơ sở UserProfile.

1.2.2. Khả năng mở rộng:

- Khi cần thêm loại người dùng mới, chỉ cần tạo một lớp con mới kế thừa từ UserProfile. Điều này giúp hệ thống dễ dàng mở rộng mà không cần thay đổi cấu trúc hiện tại.

1.2.3. Bảo trì và vận hành hiệu quả:

- Với một cấu trúc rõ ràng và thống nhất, việc bảo trì mã nguồn trở nên dễ dàng hơn. Các thay đổi chỉ cần thực hiện ở một nơi và sẽ tự động áp dụng cho tất cả các loại người dùng.

1.2.4. Giảm số lượng cơ sở dữ liệu cần tạo:

- Mẫu Composite cho phép lưu trữ thông tin của các loại người dùng khác nhau trong cùng một bảng hoặc một nhóm bảng liên quan. Điều này giúp giảm số lượng cơ sở dữ liệu cần thiết, tối ưu hóa việc quản lý dữ liệu.

1.2.5. Tính linh hoạt:

- Mẫu Composite cho phép bạn dễ dàng thêm hoặc bớt các thuộc tính của người dùng mà không ảnh hưởng đến các phần khác của hệ thống. Điều này giúp hệ thống linh hoạt hơn trong việc đáp ứng các yêu cầu thay đổi từ phía người dùng hoặc doanh nghiệp.

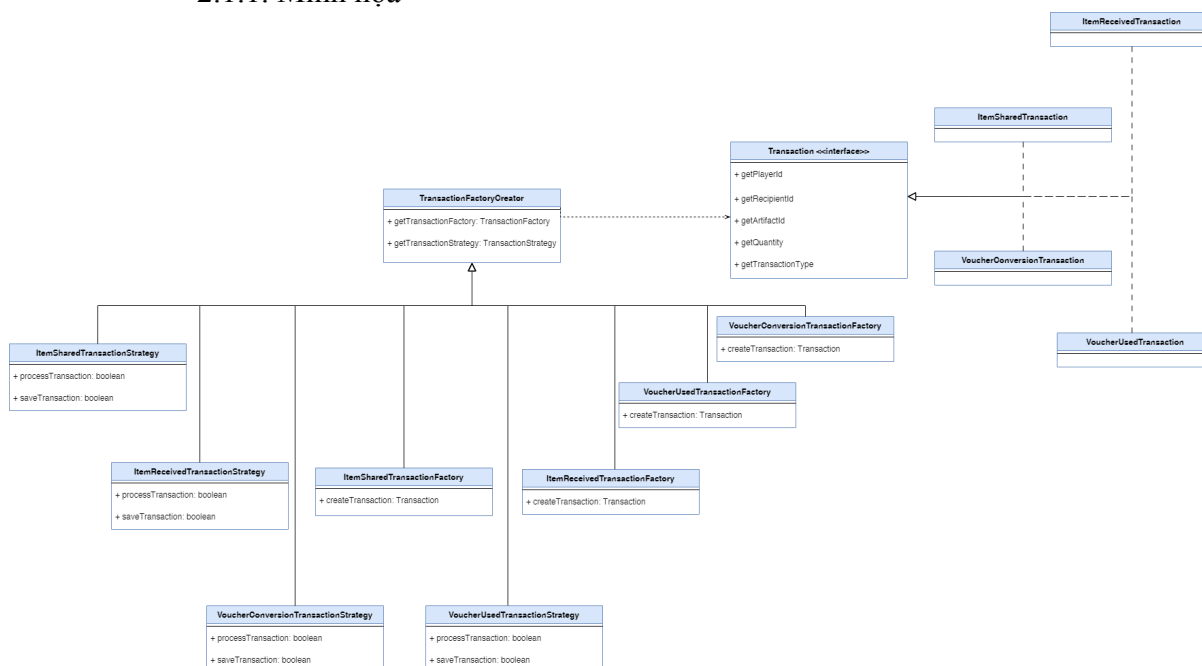
1.3. Mã nguồn

```
public User deserialize(JsonParser p, DeserializationContext ctxt) throws IOException {  
    ObjectMapper mapper = (ObjectMapper) p.getCodec();  
    JsonNode node = mapper.readTree(p);  
    String role = node.get("role").asText();  
  
    User user;  
    switch (UserRole.valueOf(role)) {  
        case player:  
            user = new Player();  
            break;  
        case brand:  
            user = new Brand();  
            break;  
        case admin:  
            user = new Admin();  
            break;  
        default:  
            throw new IllegalArgumentException("Unknown role: " + role);  
    }  
}
```

2. Statistics Service: Mẫu Factory Method

2.1. Minh họa và mô tả:

2.1.1. Minh họa



2.1.2 Mô tả

- Factory Method Pattern đang được áp dụng để quản lý việc tạo ra các đối tượng Transaction khác nhau như **ItemReceivedTransaction**, **ItemSharedTransaction**, **VoucherConversionTransaction**, và **VoucherUsedTransaction**. Các lớp factory cụ thể như **ItemReceivedTransactionFactory**, **ItemSharedTransactionFactory**, **VoucherConversionTransactionFactory**, và **VoucherUsedTransactionFactory** chịu trách nhiệm tạo các đối tượng tương ứng mà không yêu cầu lớp client phải biết chính xác lớp nào cần được khởi tạo.

2.2. Ý nghĩa và lợi ích:

2.2.1. Ý nghĩa:

- Factory Method Design Pattern hay gọi ngắn gọn là Factory Pattern là một trong những Pattern thuộc nhóm Creational Design Pattern được sử dụng để tách biệt việc khởi tạo đối tượng từ mã logic, đồng thời trao quyền tạo đối tượng cho các lớp con. Điều này có nghĩa là thay vì việc client trực tiếp khởi tạo đối tượng, quá trình này được ủy thác cho các Factory cụ thể.

2.2.2. Giảm sự phụ thuộc vào các lớp cụ thể:

- Factory Method giúp giảm sự phụ thuộc giữa các lớp khi client không cần biết chính xác đối tượng nào sẽ được tạo ra. Việc khởi tạo đối tượng được xử lý thông qua các factory cụ thể, giúp giảm thiểu rủi ro khi thay đổi code trong tương lai. Ví dụ: Khi cần thay đổi cách khởi tạo của một loại Transaction, chỉ cần chỉnh sửa trong lớp Factory tương ứng mà không cần thay đổi logic của các lớp khác.

2.2.3. Mở rộng dễ dàng:

- Khi cần thêm loại transaction mới (ví dụ, **VoucherRefundTransaction**), chỉ cần tạo thêm một factory mới mà không cần chỉnh sửa các lớp hiện tại. Điều này tuân theo nguyên tắc Open/Closed Principle (mở để mở rộng, đóng để chỉnh sửa) trong SOLID.

2.2.4. Tối ưu hóa cho sự phân chia nhiệm vụ:

- Mỗi Factory có nhiệm vụ cụ thể, giúp mã code dễ bảo trì và dễ đọc hơn. Mỗi lớp con của TransactionFactory chỉ tập trung vào việc tạo ra các transaction cụ thể. Điều này giúp tránh các đoạn mã phức tạp hoặc chồng chéo trong quá trình khởi tạo các đối tượng khác nhau.

2.2.5. Tái sử dụng code:

- Các lớp con có thể chia sẻ hoặc kế thừa logic từ lớp cha. Ví dụ: **TransactionFactoryCreator** có thể chứa các phương thức chung cho tất cả các Factory, như **getTransactionStrategy**, giúp đơn giản hóa quá trình xây dựng chiến lược xử lý Transaction mà không phải viết lại nhiều lần.

2.2.6. Dễ kiểm thử:

- Việc sử dụng các Factory giúp dễ dàng hơn trong quá trình viết Unit Test. Bạn có thể mock hoặc thay thế các Factory để kiểm thử từng thành phần riêng lẻ mà không cần tạo tất cả các loại đối tượng phức tạp.

2.3. Mã nguồn:

- Transaction Interface:

```
public interface Transaction {
    String getPlayerId();
    String getRecipientId();
    String getArtifactId();
    int getQuantity();
    String getTransactionType();
    Date getCreatedAt();
    void setCreatedAt(Date createdAt);
    Date getUpdatedAt();
    void setUpdatedAt(Date updatedAt);
    default String getEventId() { return null; }
    default void setRecipientId(String _recipientId) {}
    default void setArtifactId(String _artifactId) {}
    default void setEventId(String _eventId) {}
    default void onCreate() {
        Date now = new Date();
        setCreatedAt(now);
        setUpdatedAt(now);
    }
    default void onUpdate() {
        setUpdatedAt(new Date());
    }
}
```

- Creator:

```
public class TransactionFactoryCreator {
    public static TransactionFactory getTransactionFactory(String transactionType) {
        switch (transactionType.toLowerCase()) {
            case TRANSACTION_TYPE_ITEM_SHARED:
                return new ItemSharedTransactionFactory();
            case TRANSACTION_TYPE_ITEM_RECEIVED:
                return new ItemReceivedTransactionFactory();
            case TRANSACTION_TYPE_VOUCHER_CONVERSION:
                return new VoucherConversionTransactionFactory();
            case TRANSACTION_TYPE_VOUCHER_USED:
                return new VoucherUsedTransactionFactory();
            default:
                throw new IllegalArgumentException("Unknown transaction type: " + transactionType);
        }
    }

    public static TransactionStrategy getTransactionStrategy(String transactionType) {
        switch (transactionType.toLowerCase()) {
            case TRANSACTION_TYPE_ITEM_SHARED:
                return new ItemSharedTransactionStrategy();
            case TRANSACTION_TYPE_ITEM_RECEIVED:
                return new ItemReceivedTransactionStrategy();
            case TRANSACTION_TYPE_VOUCHER_CONVERSION:
                return new VoucherConversionTransactionStrategy();
            case TRANSACTION_TYPE_VOUCHER_USED:
                return new VoucherUsedTransactionStrategy();
            default:
                throw new IllegalArgumentException("Unknown transaction type: " + transactionType);
        }
    }
}
```

- A concrete transaction:

```
public class ItemSharedTransaction implements Transaction {
    @Id
    private ObjectId id;
    @Field(value = "player_id")
    private String playerId;
    @Field(value = "recipient_id")
    private String recipientId;
    @Field(value = "item_id")
    private String artifactId;
    @Field(value = "game_id")
    private Long gameId;
    @Field(value = "transaction_date")
    private LocalDateTime transactionDate;
    @Field(value = "quantity")
    private int quantity;
    @Field(value = "transaction_type")
    private String transactionType = TRANSACTION_TYPE_ITEM_SHARED;
    @Field(value = "created_at")
    private Date createdAt;
    @Field(value = "updated_at")
    private Date updatedAt;
    public String getPlayerId() {
        return playerId;
    }
    public String getRecipientId() {
        return recipientId;
    }
    public String getArtifactId() {
        return artifactId;
    }
    public int getQuantity() {
        return quantity;
    }
    public String getTransactionType() {
        return transactionType;
    }
    public Date getCreatedAt() {
        return createdAt;
    }
    public void setArtifactId(String _artifactId) {
        this.artifactId = _artifactId;
    }
    public void setCreatedAt(Date createdAt) {
        this.createdAt = createdAt;
    }
    public Date getUpdatedAt() {
        return updatedAt;
    }
    public void setUpdatedAt(Date updatedAt) {
        this.updatedAt = updatedAt;
    }
    @PrePersist
    public void onCreate() {
        Transaction.super.onCreate();
    }
    @PreUpdate
    public void onUpdate() {
        Transaction.super.onUpdate();
    }
}
```

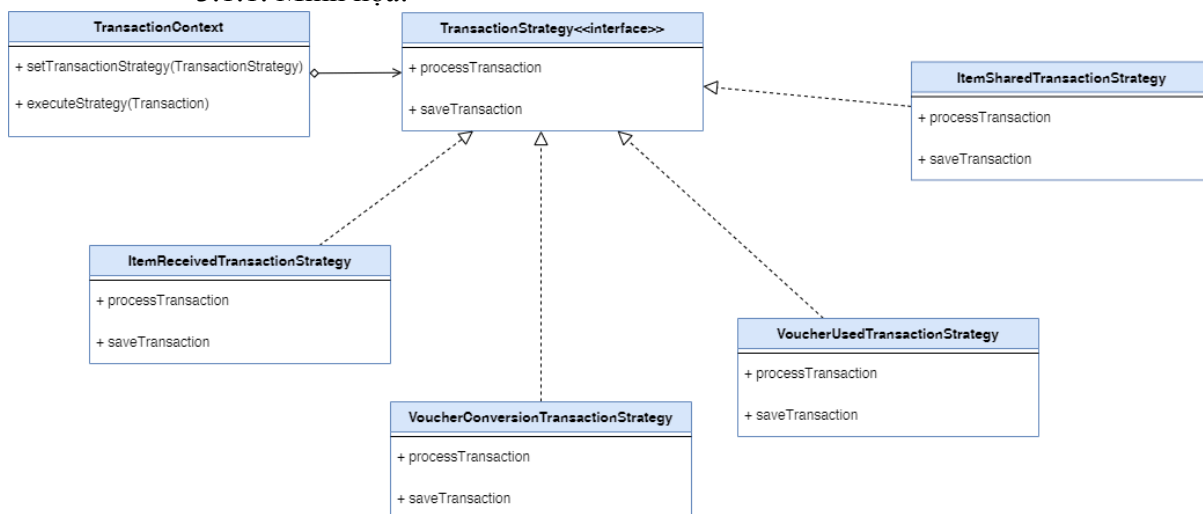
- A concrete creator:

```
public class ItemSharedTransactionFactory implements TransactionFactory {
    @Override
    public Transaction createTransaction(TransactionDto transactionDto ) {
        return TransactionMapper.toEntity(transactionDto);
    }
}
```

3. Statistics Service: Strategy Pattern

3.1. Minh họa và mô tả:

3.1.1. Minh họa:



3.1.2. Mô tả

- Các strategy xử lý transaction như **ItemReceivedTransactionStrategy**, **ItemSharedTransactionStrategy**, **VoucherConversionTransactionStrategy**, và **VoucherUsedTransactionStrategy** đều kế thừa từ interface **TransactionStrategy**. Lớp **TransactionContext** sử dụng các strategies này để thực hiện các hoạt động xử lý giao dịch và lưu giao dịch. Việc tách rời phần xử lý thuật toán ra khỏi lớp context giúp code dễ mở rộng và bảo trì hơn.

3.2. Ý nghĩa và lợi ích:

3.2.1. Ý nghĩa:

- Strategy Design Pattern là một mẫu thiết kế hành vi (Behavioral Design Pattern) giúp tách biệt và lựa chọn các thuật toán khác nhau cho một tác vụ cụ thể mà không cần thay đổi lớp sử dụng thuật toán đó. Bằng cách đóng gói các thuật toán thành những lớp riêng biệt, có thể dễ dàng thay đổi hoặc mở rộng chúng mà không ảnh hưởng đến

phần còn lại của hệ thống.

3.2.2. Thay đổi thuật toán một cách linh hoạt:

- Strategy Pattern cho phép dễ dàng thay đổi hoặc thêm mới các chiến lược xử lý mà không cần chỉnh sửa mã của lớp **TransactionContext**. Bạn chỉ cần tạo một lớp chiến lược mới (như **NewTransactionStrategy**) và cài đặt các phương thức **processTransaction** và **saveTransaction**, sau đó cài đặt chiến lược mới này vào TransactionContext.

3.2.3. Tuân thủ nguyên tắc Open/Closed:

- Mẫu thiết kế này giúp lớp TransactionContext tuân theo nguyên tắc Open/Closed Principle trong SOLID (Mở để mở rộng, đóng để chỉnh sửa). Khi có nhu cầu thêm các loại transaction khác nhau, bạn chỉ cần mở rộng hệ thống bằng cách thêm chiến lược mới mà không cần thay đổi logic của lớp hiện tại.

3.2.4. Tối ưu hóa khả năng tái sử dụng:

- Các lớp chiến lược cụ thể (**ItemReceivedTransactionStrategy**, **VoucherUsedTransactionStrategy**,...) đều có thể được tái sử dụng ở những nơi khác mà không bị phụ thuộc vào **TransactionContext**. Điều này giúp giảm thiểu sự lặp lại trong code và tăng khả năng tái sử dụng.

3.2.5. Tăng tính bảo trì:

- Strategy Pattern giúp cho mã nguồn trở nên gọn gàng và dễ bảo trì. Bằng cách tách các thuật toán khác nhau thành các lớp riêng, việc bảo trì hoặc thay đổi từng chiến lược trở nên dễ dàng hơn. Nếu cần sửa đổi một chiến lược cụ thể, chỉ cần chỉnh sửa lớp chiến lược tương ứng mà không ảnh hưởng đến các phần khác của hệ thống.

3.2.6. Giảm sự phức tạp của lớp Context:

- Lớp **TransactionContext** chỉ cần xử lý logic chung như thiết lập và thực thi chiến lược, mà không cần biết chi tiết cụ thể của từng loại transaction. Điều này giúp lớp context không trở nên quá phức tạp và dễ quản lý hơn.

3.2.7. Khả năng mở rộng không giới hạn:

- Khi hệ thống cần thêm các loại giao dịch mới trong tương lai (ví dụ, thêm loại **TransactionStrategy** cho một giao dịch mới), chỉ cần tạo thêm các lớp chiến lược mà không làm phức tạp hóa hệ thống hiện có. Điều này giúp hệ thống có thể mở rộng không giới hạn mà vẫn giữ được tính dễ bảo trì.

3.3. Mã nguồn:

- Context:

```
public class TransactionContext {
    private TransactionStrategy transactionStrategy;

    public void setTransactionStrategy(TransactionStrategy transactionStrategy) {
        this.transactionStrategy = transactionStrategy;
    }

    public boolean executeStrategy(Transaction transaction, PlayerVoucherService play
        try {
            return transactionStrategy.processTransaction(transaction, playerVouchers
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }
}
```

- Strategy interface:

```
public interface TransactionStrategy {
    boolean processTransaction(Transaction transaction, PlayerVoucherService playerVoucherService, PlayerItemService pl
    boolean saveTransaction(Transaction transaction, TransactionRepository<Transaction> transactionRepository);
}
```

- Concrete strategies:

```
public class ItemSharedTransactionStrategy implements TransactionStrategy {

    private PlayerItemService playerItemService;
    private ItemSharedTransactionRepository itemSharedTransactionRepository;

    public ItemSharedTransactionStrategy(PlayerItemService playerItemService,
        ItemSharedTransactionRepository itemSharedTransactionRepository) {
        this.playerItemService = playerItemService;
        this.itemSharedTransactionRepository = itemSharedTransactionRepository;
    }

    @Override
    public boolean processTransaction(Transaction transaction, PlayerVoucherService playerVoucherService, PlayerItemService player
        if (!transaction.getTransactionType().equalsIgnoreCase(TRANSACTION_TYPE_ITEM_SHARED)) {
            throw new IllegalArgumentException(s:"Invalid transaction type for ItemSharedTransactionStrategy");
        }

        try {
            ItemSharedTransaction itemSharedTransaction = (ItemSharedTransaction) transaction;
        }
    }
}
```

5. Phần ứng dụng

5.1. Phân hệ dành cho Admin (web):

5.1.1. Quản lý tài khoản người dùng (Thêm mới/Cập nhật/Xóa) (Người dùng ở đây có thể là các thương hiệu, người tham gia trò chơi, admin,...)

Admin có khả năng quản lý tất cả tài khoản người dùng trong hệ thống, bao gồm các thương hiệu, người tham gia trò chơi và chính admin. Chức năng này cho phép admin

thêm mới, cập nhật thông tin hoặc xóa tài khoản khi cần thiết. Thông tin người dùng bao gồm họ tên, tên đăng nhập, mật khẩu, email, số điện thoại và quyền hạn.

5.1.2. Kích hoạt/khóa tài khoản người dùng Admin (phải sử dụng chức năng cập nhật tài khoản.)

Admin có thể kích hoạt hoặc khóa tài khoản của người dùng thông qua chức năng cập nhật tài khoản. Điều này giúp đảm bảo rằng chỉ những người dùng hợp lệ mới có quyền truy cập vào hệ thống.

5.1.3. Quản lý thông tin trò chơi (Cập nhật thông tin/trạng thái)

Admin có thể cập nhật thông tin và trạng thái của các trò chơi trong hệ thống. Mỗi trò chơi sẽ bao gồm các thông tin như tên, hình ảnh, loại trò chơi và hướng dẫn chơi, giúp người dùng dễ dàng tham gia.

5.1.4. Chức năng báo cáo thống kê (Thống kê dữ liệu đối tác, người chơi, trò chơi)

Admin có thể truy cập vào các báo cáo thống kê về dữ liệu đối tác, người chơi và trò chơi. Chức năng này giúp admin theo dõi hiệu suất và tình hình hoạt động của hệ thống.

5.1.5. Thống kê:

a. Thống kê dữ liệu đối tác.

Cung cấp thông tin về số lượng và hiệu quả của các đối tác tham gia.

b. Thống kê dữ liệu người chơi (Tổng thời gian chơi của mỗi người).

Theo dõi tổng thời gian chơi của mỗi người dùng, giúp admin đánh giá mức độ tham gia của người chơi.

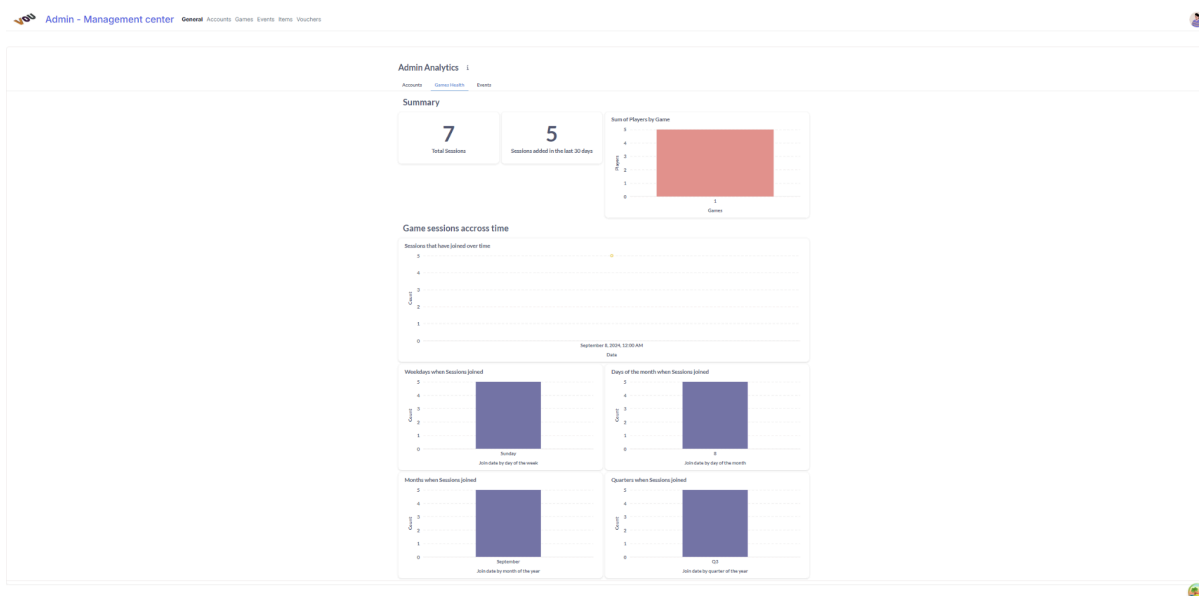
c. Thống kê dữ liệu trò chơi. (Tổng số người chơi)

Cung cấp thông tin về tổng số người chơi tham gia vào từng trò chơi, từ đó giúp admin phân tích và cải thiện trò chơi.

5.2. Phân hệ dành cho Brand (web):

5.2.1. Các thương hiệu đăng ký với các thông tin

Thương hiệu có thể đăng ký tài khoản với các thông tin như tên, lĩnh vực, địa chỉ và trạng thái. Điều này giúp hệ thống quản lý và phân loại các thương hiệu một cách hiệu quả.



5.2.2. Đăng ký các sự kiện khuyến mãi (gồm thêm mới, tra cứu và cập nhật thông tin)

Thương hiệu có thể tạo và quản lý các sự kiện khuyến mãi, bao gồm thêm mới và cập nhật thông tin sự kiện. Mỗi sự kiện sẽ có các thông tin như tên, hình ảnh và thời gian diễn ra.

Create event

Create Event Information

Event name

New Event 09:26

Ngày bắt đầu sự kiện

September 10th, 2024

Ngày kết thúc sự kiện

September 12th, 2024

Event Image

Choose file fab-news.1.jpg



Game(s) use in this Event

At least 1


☒ Quiz
☒ Shake


Start time every day

09 : 26
09 : 28







Add (existing) Vouchers to this Event

Add

#	Voucher	Quantity	Delete
1.	kitto_katsu (offline)	50	


2.	kitkat25 (online)	50	
----	-------------------	----	---

Voucher - Item Exchange

	Kitkat Brown Bag	02	
	Kitkat Brown Bag	04	
	Kitkat Iron Boot	03	
	Select item	02	

Invite Cooperations

Invite more

coffehouse@gmail.com 

Submit

You've been invited to New Event 09:27 event!

Kitkat invited you to join!

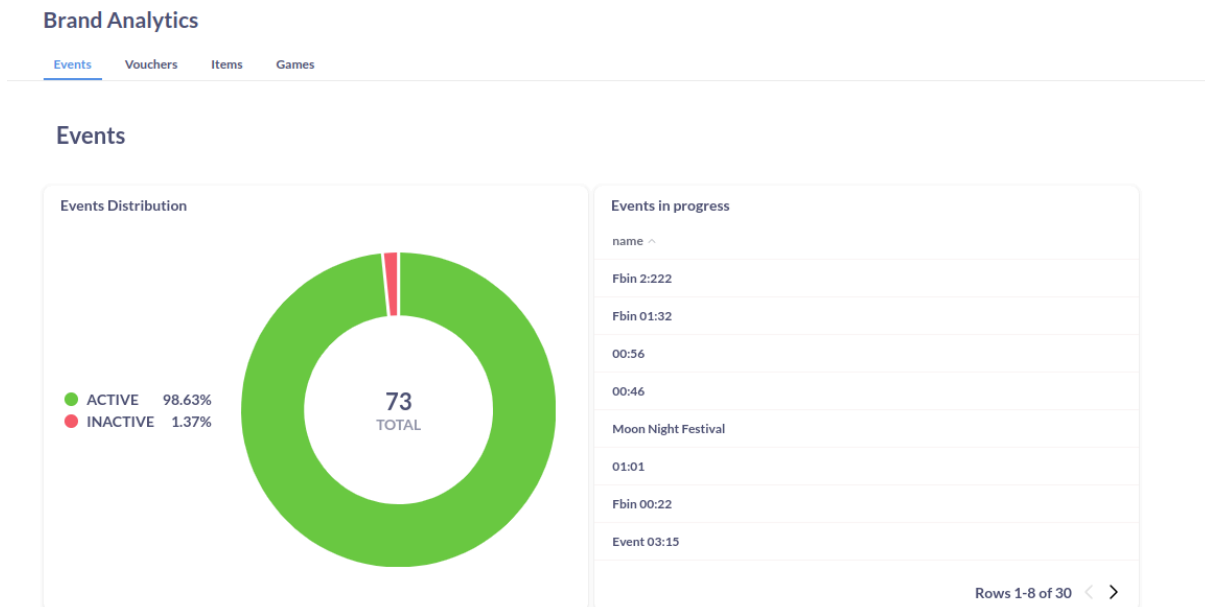


5.2.3. Khi cập nhật, các thương hiệu có thể sử dụng chức năng tra cứu sự kiện

Sau khi thương hiệu cập nhật sự kiện thì có thể tra cứu thông tin về sự kiện đó.

5.2.4. Chức năng báo cáo: Thống kê ngân sách, Thống kê tình trạng khuyến mãi

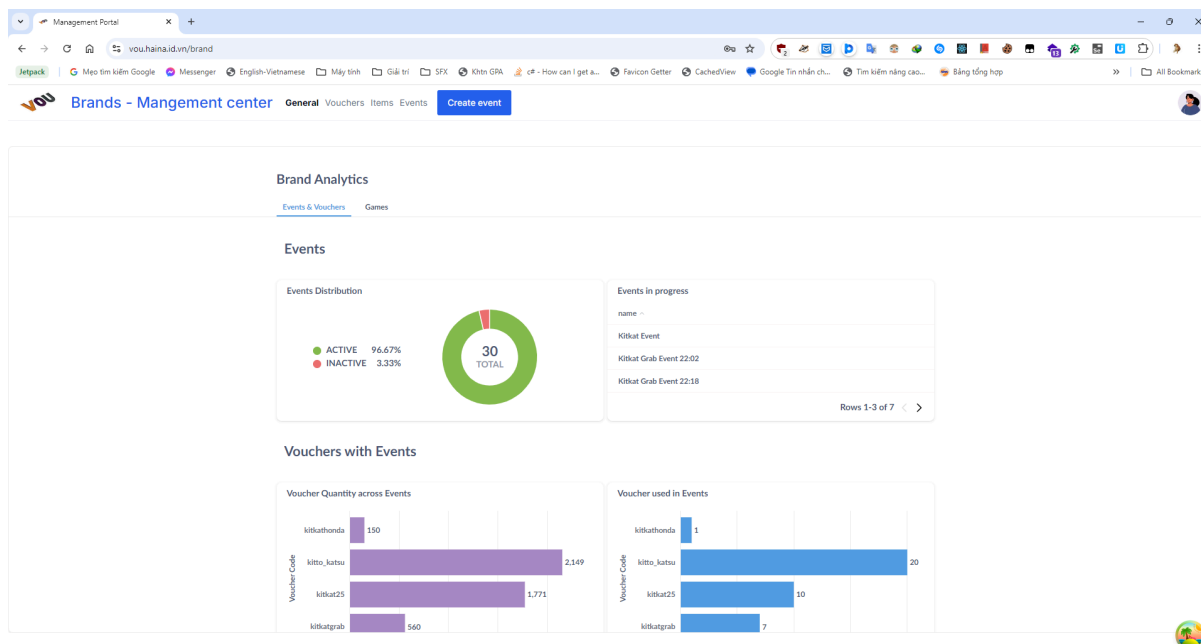
Thương hiệu có thể theo dõi ngân sách và tình trạng của các chương trình khuyến mãi thông qua chức năng báo cáo, giúp họ điều chỉnh chiến lược marketing hiệu quả hơn.



5.2.6. Voucher Online / Offline

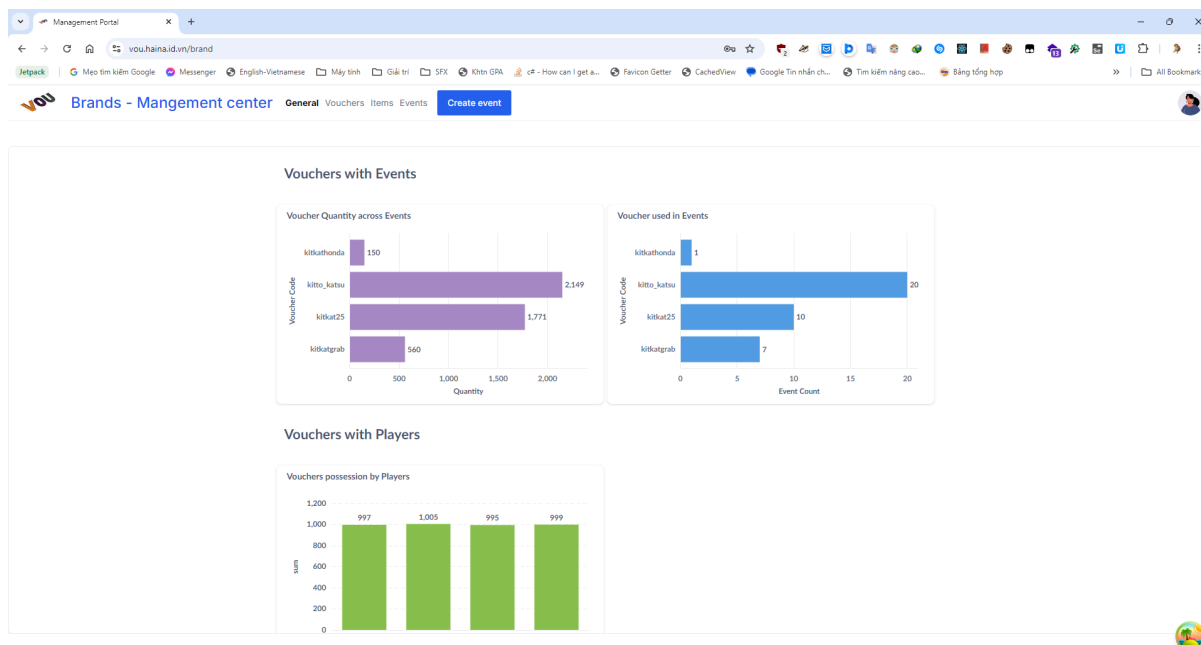
Hệ thống hỗ trợ quản lý các voucher dùng để khuyến mãi, bao gồm cả voucher sử dụng trực tuyến và tại cửa hàng, giúp thương hiệu dễ dàng theo dõi và quản lý.

5.2.7. Thống kê:

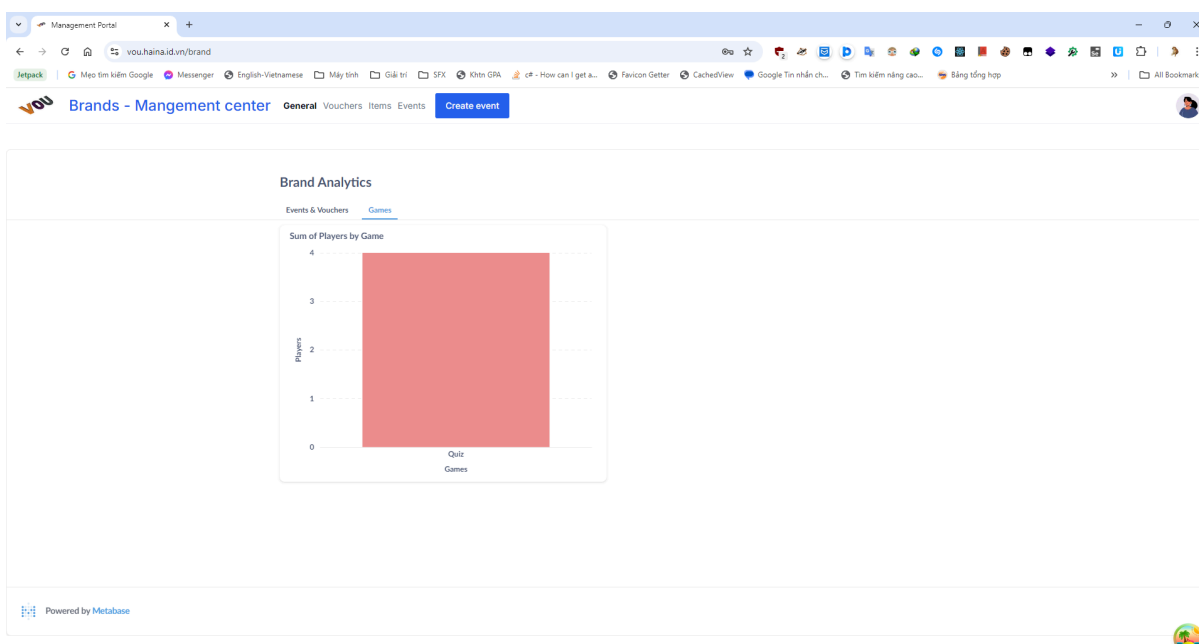


Thương hiệu có thể thống kê ngân sách, tình trạng khuyến mãi, số lượng người chơi tham gia sự kiện, thời gian tham gia của mỗi người chơi và số lượng voucher đã được sử dụng.

a. Thống kê tình trạng khuyến mãi.



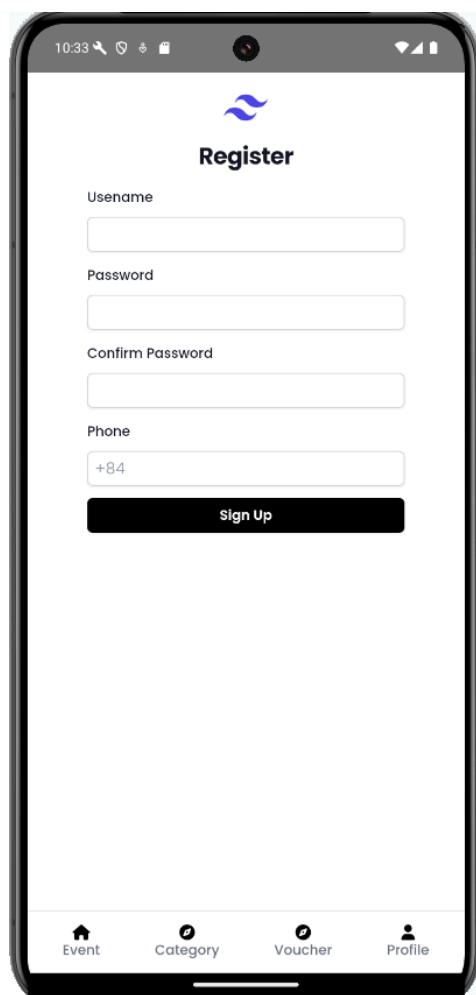
b. Sự kiện đó có bao nhiêu người chơi.



5.3 Phân hệ dành cho người chơi (mobile):

5.3.1. Người chơi đăng ký tài khoản.

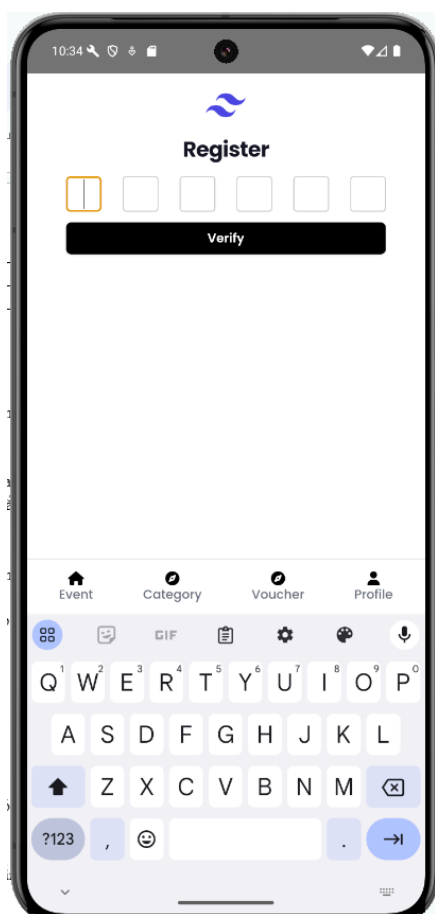
Người chơi có thể dễ dàng đăng ký tài khoản trong hệ thống VOU với các thông tin cá nhân cần thiết.



The image shows a mobile application interface for registration. At the top, there is a status bar with the time 10:33 and various icons. Below the status bar is a blue header with a logo and the word "Register". The main content area contains four input fields: "Username", "Password", "Confirm Password", and "Phone". The "Phone" field has a dropdown menu showing "+84". Below the input fields is a black button with the text "Sign Up". At the bottom of the screen is a navigation bar with four icons: "Event", "Category", "Voucher", and "Profile".

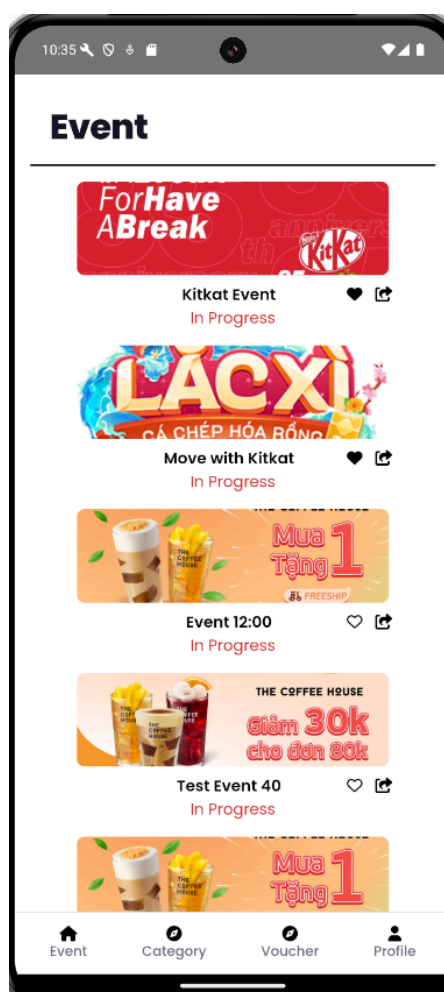
5.3.2. Đăng ký tài khoản bằng cách nhập số điện thoại và xác thực OTP để định danh khi nhận thưởng.

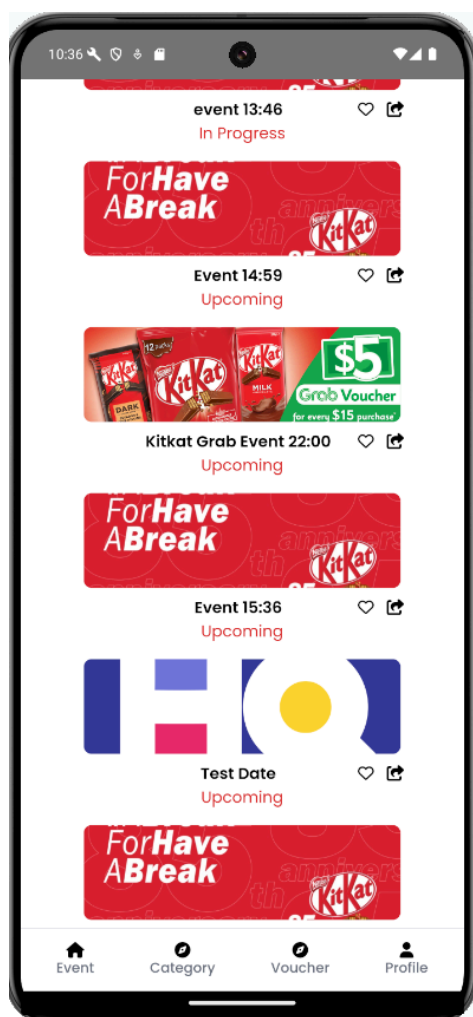
Hệ thống yêu cầu người chơi nhập số điện thoại và xác thực qua OTP để đảm bảo tính chính xác và bảo mật cho tài khoản.



5.3.3. Hệ thống VOU hiển thị tất cả các sự kiện khuyến mãi của các thương hiệu đang và sắp diễn ra.

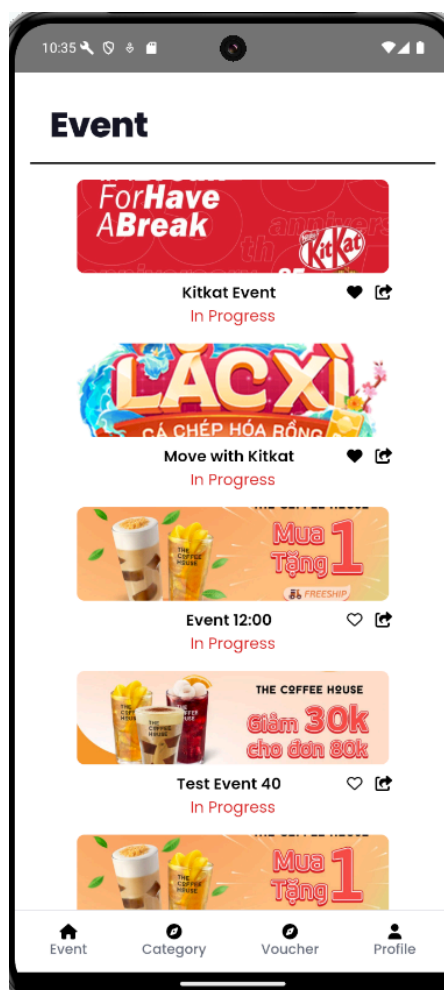
Người chơi có thể xem tất cả các sự kiện khuyến mãi của các thương hiệu đang diễn ra và sắp diễn ra, giúp họ không bỏ lỡ cơ hội tham gia.





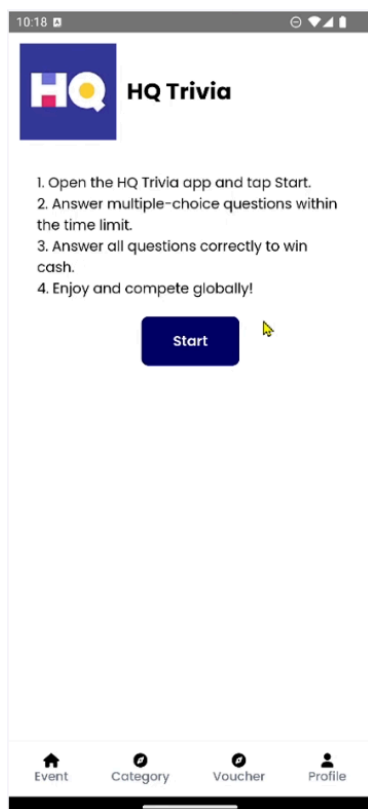
5.3.4. Trong quá trình xem khách hàng có thể sử dụng chức năng Sự Kiện Yêu Thích để lưu lại danh sách để notify khi sự kiện sắp diễn ra.

Người chơi có thể lưu lại danh sách các sự kiện yêu thích để nhận thông báo khi sự kiện sắp diễn ra, tạo sự tiện lợi và thu hút hơn.



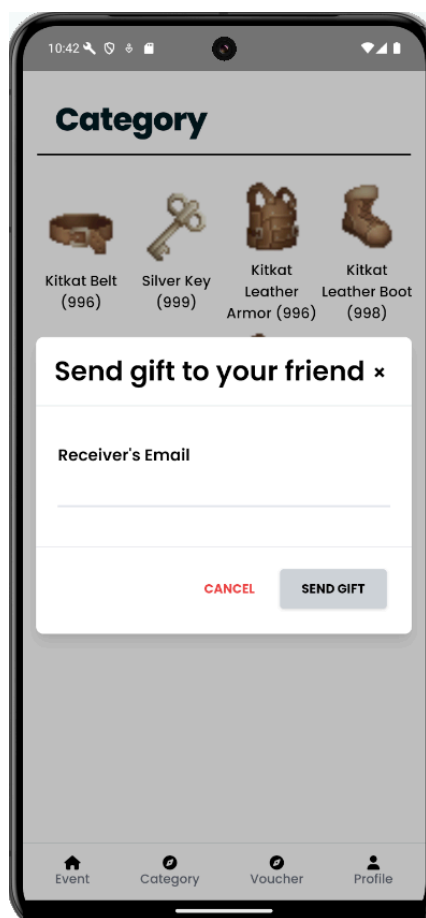
5.3.5. Người chơi chọn sự kiện và xem hướng dẫn và tham gia trò chơi của thương hiệu đó và cách nhận phần thưởng nếu chiến thắng.

Người chơi có thể chọn sự kiện, xem hướng dẫn và tham gia trò chơi của thương hiệu, từ đó biết cách nhận phần thưởng nếu chiến thắng.



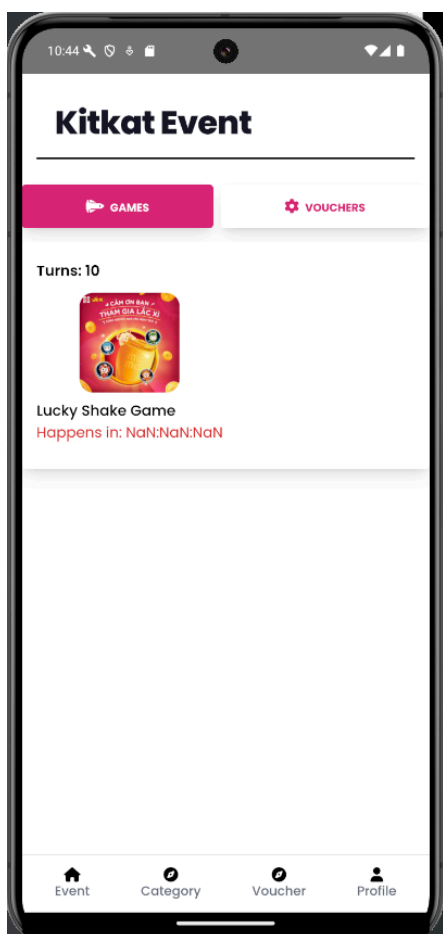
5.3.6. Đối với các trò chơi sưu tầm các vật phẩm để đổi voucher, khách hàng có thể tặng các vật phẩm cho bạn bè bằng cách nhập số điện thoại hoặc email hoặc định danh của bạn bè.

Người chơi có thể tặng các vật phẩm cho bạn bè bằng cách nhập số điện thoại hoặc email, tạo sự gắn kết trong cộng đồng người chơi.



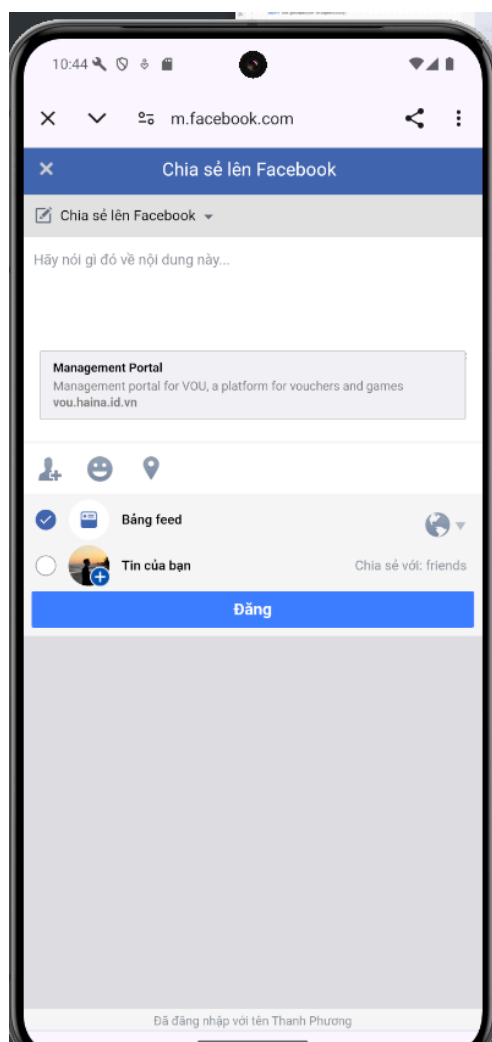
5.3.7. Mỗi khách hàng ban đầu được tặng 10 (Config) lượt chơi.

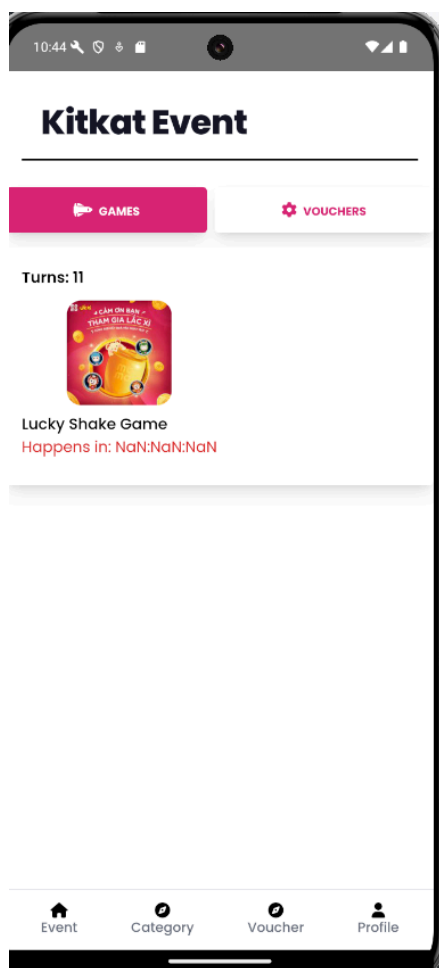
Mỗi khách hàng mới sẽ được tặng 10 lượt chơi, khuyến khích họ tham gia vào các trò chơi.



5.3.8. Sau khi sử dụng hết các lượt chơi của mình, người chơi có thể tìm kiếm thêm lượt chơi bằng cách: (Xin lượt chơi từ bạn bè, Chia sẻ Facebook để thêm lượt chơi).

Sau khi sử dụng hết lượt chơi, người chơi có thể xin lượt chơi từ bạn bè hoặc chia sẻ trên Facebook để nhận thêm lượt chơi.





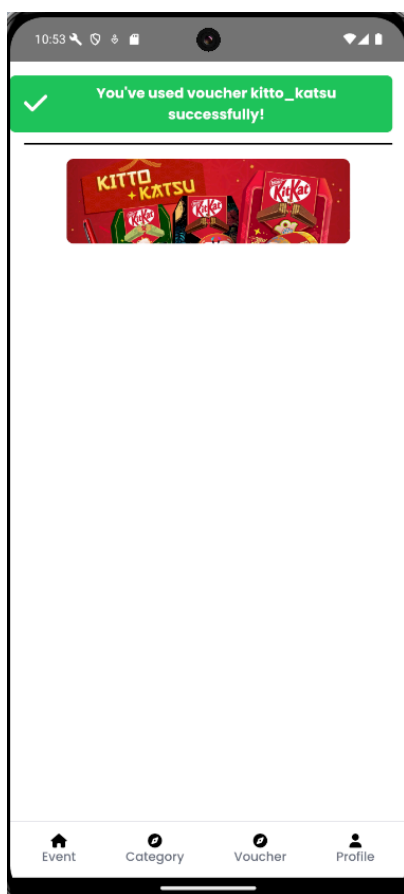
5.3.9. Voucher sử dụng Online, trả về thông tin quà.

Trừ voucher đó ra khỏi túi của người dùng sau đó người dùng sẽ nhận được một voucher khác với trạng thái là offline



5.3.10. Voucher sử dụng Offline, hệ thống VOU sẽ có thông báo cho khách hàng biết nếu mã voucher sử dụng thành công.

Hệ thống hỗ trợ người dùng nhận thông báo khi mã voucher sử dụng thành công tại cửa hàng.



5.3.11. Save Transaction (addition, optional).

Hệ thống có thể lưu lại thông tin giao dịch để người chơi dễ dàng theo dõi và quản lý các hoạt động của mình.

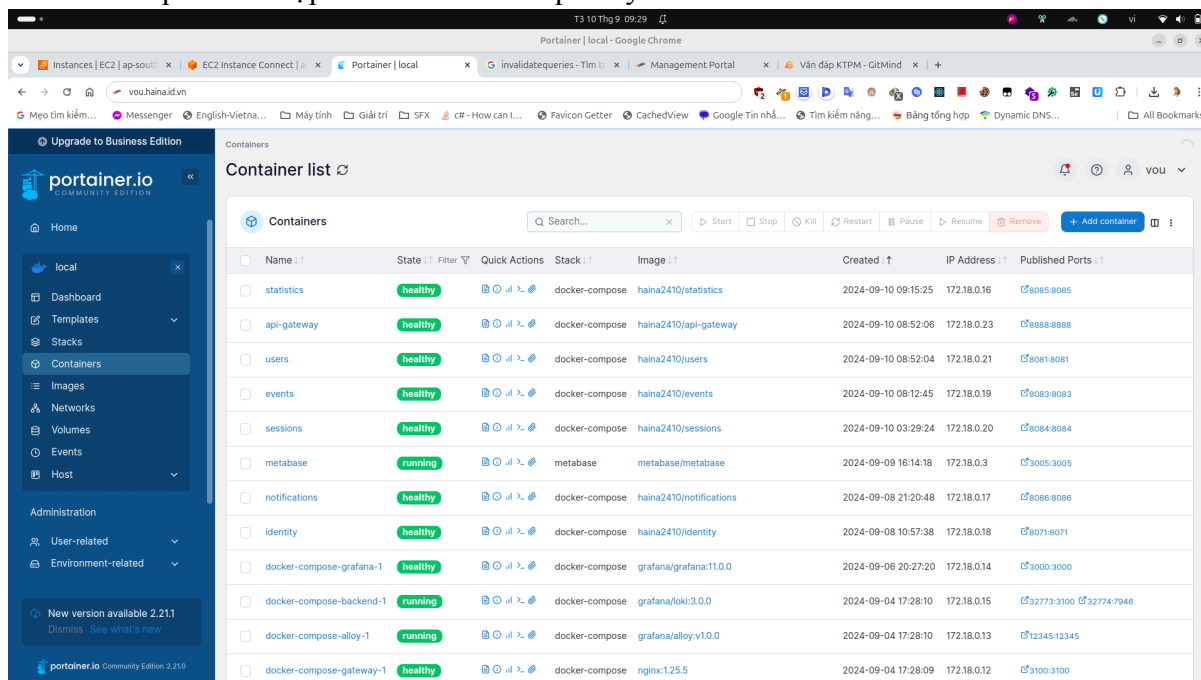
6. Các phần nâng cao khác

6.1. Chức năng đặc biệt và có ý nghĩa:

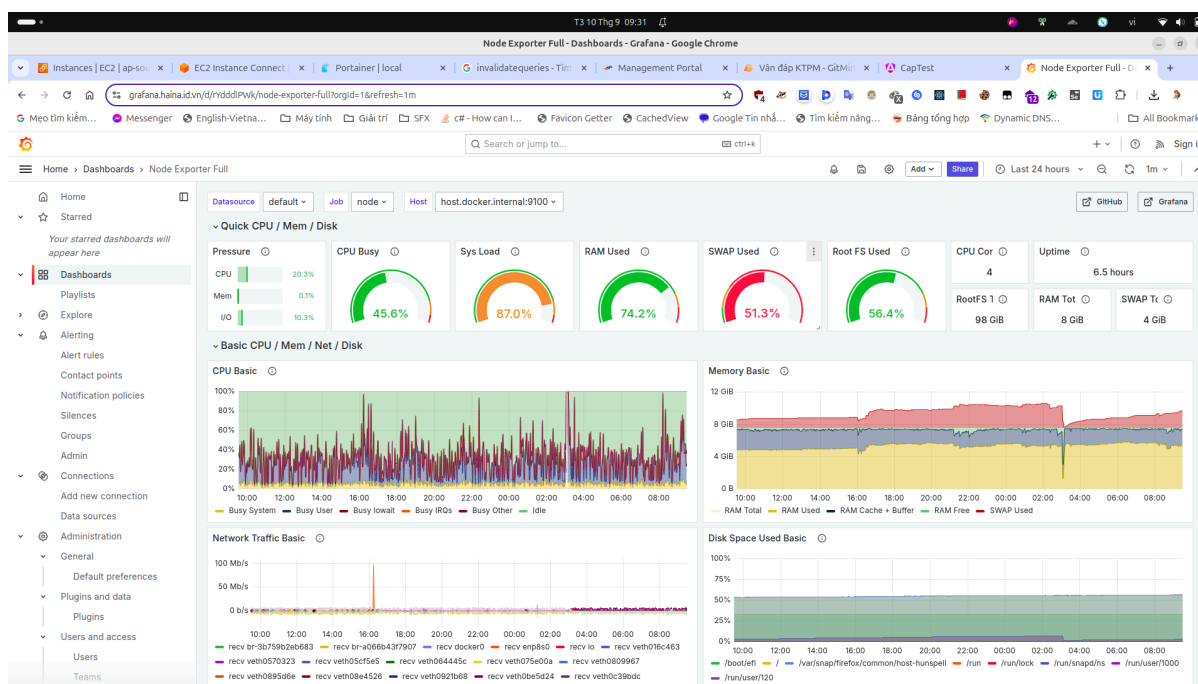
- Có lưu lại transactions log cho tất cả loại transaction diễn ra của hệ thống trên MongoDB → Đây là chức năng không được yêu cầu trong đề bài, hơn nữa ý nghĩa đặc biệt là có thể dùng để thống kê hoặc theo dõi lịch sử các loại giao dịch khác nhau của người dùng trong hệ thống khi cần.
- Thiết lập các thành phần liên quan đến Observability và Monitoring giúp giám sát hiệu suất, metrics và logs sinh ra từ hệ thống, từ đó giúp phát hiện sớm các lỗi trong quá trình runtime và khắc phục kịp thời. Công cụ tụi em sử dụng là Grafana stack gồm Prometheus để thu thập và quản lý metrics, Loki để thu thập và quản lý logs, Grafana để hiển thị dữ liệu dưới dạng biểu đồ trực quan.
- Phần MC ảo có sử dụng Amazon Polly để tạo file audio có định dạng .mp3 từ nội

dung câu hỏi và câu trả lời được lấy từ Open Trivia Database (Third-party API). File audio có phát âm chuẩn tiếng anh, ngắt nghỉ giữa các câu hỏi và các câu trả lời tạo cảm giác giống người thật đọc. Bên cạnh đó, video hiển thị hình ảnh MC ảo cũng được phát một cách phù hợp để nhép miệng theo âm thanh được phát ra từ audio mang đến cảm giác chân thật hơn.

- On Premise Hosting: Toàn bộ hệ thống VOU được host tại máy nhà, sử dụng Docker Compose kết hợp với Portainer để quản lý container:



- Server Web Admin / Brand được host tại địa chỉ <https://vou.haina.id.vn>
- Server Mobile được host tại địa chỉ <https://mobile.haina.id.vn/>
- Server Grafana được host tại địa chỉ <https://grafana.haina.id.vn/>



7. Kết Luận và Đánh Giá

7.1. Đánh giá hiệu quả của kiến trúc hiện tại và các thay đổi đã thực hiện.

Kiến trúc hiện tại của hệ thống đã đáp ứng đầy đủ các yêu cầu của đồ án, đảm bảo hiệu năng, khả năng mở rộng, độ tin cậy và bảo mật ở mức cao. Việc sử dụng Spring Cloud Gateway giúp điều phối các yêu cầu từ phía client một cách hiệu quả, trong khi Eureka Service Discovery đảm bảo các dịch vụ có thể tìm thấy nhau dễ dàng, tăng cường khả năng mở rộng và linh hoạt của hệ thống. Các thay đổi được thực hiện đã tối ưu hóa và nâng cao mức độ hoàn thiện của hệ thống, đặc biệt là việc tích hợp các công cụ giám sát như Grafana và Prometheus để theo dõi hiệu suất và sức khỏe của hệ thống. Tuy nhiên, vẫn cần phải cải thiện thêm về bảo mật nâng cao để đối phó với các mối đe dọa tiềm ẩn, chẳng hạn như việc triển khai các biện pháp bảo mật mạnh mẽ hơn cho các dịch vụ nhạy cảm.

7.2. Nêu ra những bài học kinh nghiệm trong quá trình thiết kế và triển khai hệ thống.

Trong quá trình phát triển hệ thống microservice cho đồ án VOU - Marketing with Real-time Games, nhóm đã nhận ra rằng thiếu kinh nghiệm dẫn đến việc không có lộ trình làm việc tối ưu. Việc thiết lập phân authentication và authorization trễ đã khiến hệ thống chưa đảm bảo tốt về bảo mật. Ngoài ra, cần tuân thủ các quy chuẩn tốt hơn khi xây dựng API cho CRUD trên back-end để đảm bảo tính nhất quán và dễ bảo trì. Việc sử dụng Docker đã giúp nhóm hiểu rõ hơn về cách quản lý và triển khai các container, nhưng cũng đòi hỏi nhiều thời gian để làm quen và tối ưu hóa. Nhóm cũng nhận ra tầm quan trọng của việc lập kế hoạch chi tiết và phân chia công việc rõ ràng để tránh tình trạng quá tải và đảm bảo tiến độ dự án.

7.3. Đề xuất các cải tiến và phát triển hệ thống trong tương lai dựa trên kinh nghiệm thu được.

Dựa trên kinh nghiệm thu được, hệ thống cần được cải tiến về bảo mật và khả năng mở rộng. Cần tránh việc chỉ đáp ứng các yêu cầu chức năng hiện tại mà cần xây dựng một nền tảng linh hoạt cho các nhu cầu tương lai. Việc áp dụng các công nghệ mới và cải tiến quy trình phát triển sẽ giúp hệ thống hoạt động hiệu quả hơn. Chúng tôi đề xuất tích hợp thêm các công cụ bảo mật như OAuth2 và JWT để nâng cao khả năng bảo vệ dữ liệu người dùng. Ngoài ra, việc sử dụng các dịch vụ đám mây như AWS hoặc Azure có thể giúp tăng cường khả năng mở rộng và giảm chi phí vận hành. Đối với việc phát triển tính năng mới, cần áp dụng các phương pháp phát triển linh hoạt (Agile) để nhanh chóng thích ứng với thay đổi và cải tiến liên tục.

7.4. Các thuận lợi và khó khăn khi làm đồ án.

Thuận lợi: Nhóm có đủ tài nguyên và nhân lực cần thiết, cùng với kinh nghiệm tốt trong việc phát triển front-end, giúp cho việc xây dựng giao diện người dùng diễn ra suôn sẻ. Việc sử dụng các công cụ hiện đại như Firebase và SpeedSMS đã giúp tăng cường khả năng tương tác và thông báo cho người dùng, đồng thời cải thiện trải nghiệm người dùng tổng thể.

Khó khăn: Thiếu kinh nghiệm trong phát triển cơ sở dữ liệu và hệ thống microservice đã gây ra nhiều thách thức. Việc này đòi hỏi nhóm phải đầu tư nhiều thời gian hơn để nghiên cứu và thử nghiệm các giải pháp phù hợp. Việc quản lý và đồng bộ hóa dữ liệu giữa các dịch vụ cũng là một thách thức lớn, đòi hỏi sự cẩn trọng và kỹ năng xử lý dữ liệu tốt. Ngoài ra, việc đảm bảo tính nhất quán và độ tin cậy của hệ thống trong môi trường phân tán cũng là một vấn đề cần được giải quyết.

8. Tài Liệu Tham Khảo

1. Title: Inheritance Strategies with JPA and Hibernate – The Complete Guide. Accessed: (August 23, 2024).

<https://thorben-janssen.com/complete-guide-inheritance-strategies-jpa-hibernate/>

2. Title: API Gateway Authentication and Authorization in Spring Boot. Accessed: (August 23, 2024).

<https://www.geeksforgeeks.org/api-gateway-authentication-and-authorization-in-spring-boot/>



3. Title: Open Trivia Database. Accessed: (August 23, 2024).

<https://opentdb.com/>

4. Title: Spring Boot. Accessed: (August 23, 2024).

<https://spring.io/projects/spring-boot>

5. Title: Apache Kafka từ zero đến one. Accessed: (August 10, 2024).

<https://viblo.asia/s/apache-kafka-tu-zero-den-one-aGK7jPbA5j2>

6. Title: Firebase Cloud Messaging. Accessed: (August 20, 2024).

<https://firebase.google.com/docs/cloud-messaging>