**Thiết kế Push Notification System**

# **Mục Tiêu (Objectives):**

Mục tiêu là thiết kế hệ thống gửi thông báo đến các device của người dùng. Bộ phận Sales sử dụng hệ thống này để gửi thông báo đến người dùng giúp tăng khả năng tương tác. Điều này dẫn đến người dùng sẽ mở app, trải nghiệm và xem phim trên ứng dụng lâu hơn. Hệ thống có thể gửi thông báo đến 1 lượng người dùng lớn (trên 5 triệu) mà vẫn đảm bảo được hoạt động ổn định và ít sai sót.

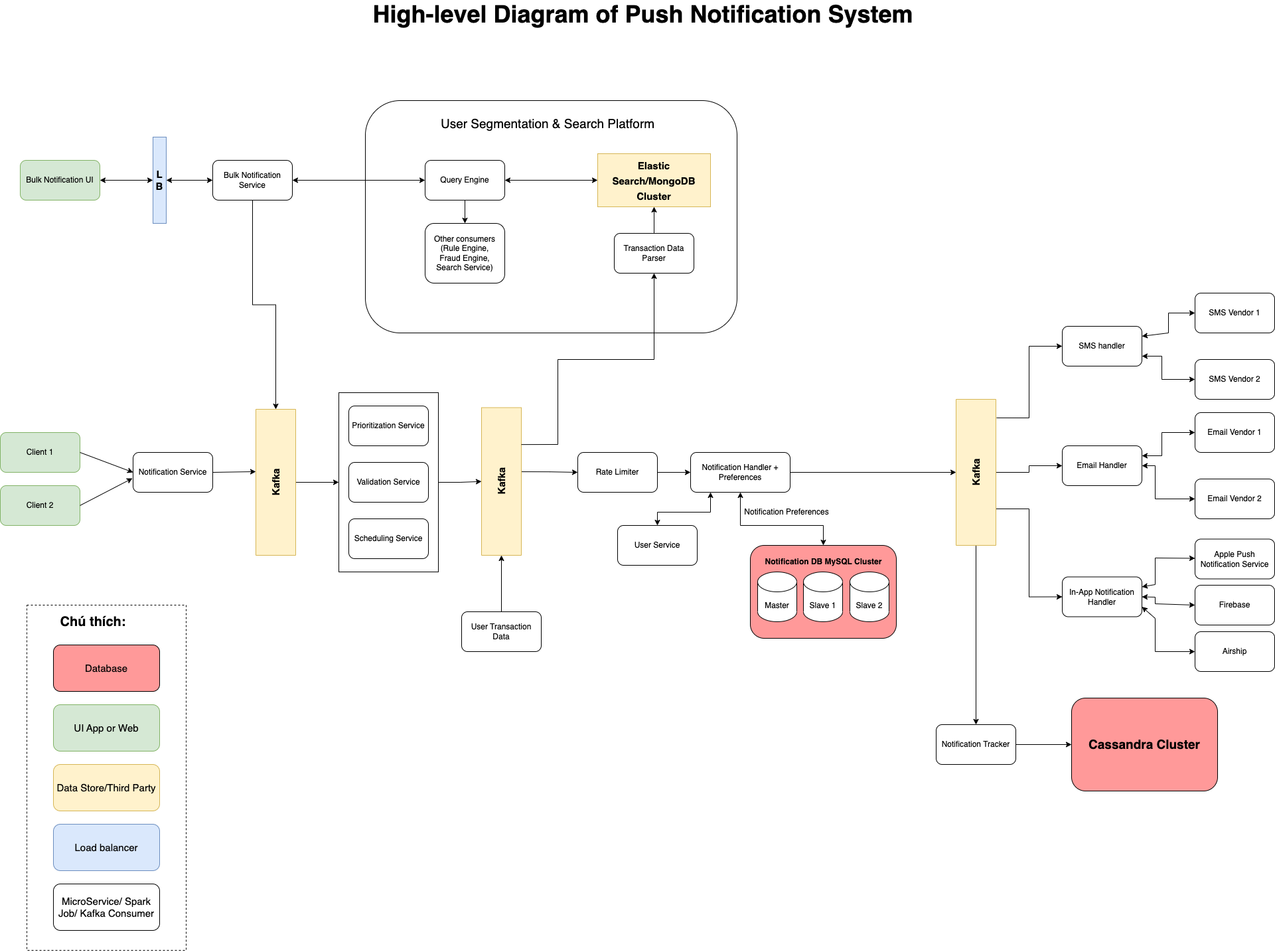
# **Functional Requirement:**

* Gửi được thông báo cho 1 người hoặc nhóm người.
* Hệ thống lưu trữ thông báo.
* Gửi thông báo ưu tiên (Prioritization).
* Ít lỗi (gửi đúng lịch, đúng người, đúng nội dung và không duplicate).

# **Non-functional requirements:**

* Hoạt động ổn định (High-available)
* Chịu được lượng tải lớn (High performance)
* Có khả năng tích hợp cao và Scale-up về mặt hệ thống

# **Về hệ thống**



**Hình 1:** High-level Diagram of Push Notification System (có thể xem chi tiết hình ở thư mục /Bài test/**Diagram/High-level diagram.png**).

## **Notification service:**

Khi 1 client hoặc nhiều client gửi Noti đến Notification Service (là 1 interface để cho mọi người trong công ty có thể nói chuyện với nhau, hoặc là công ty khác hoặc là bất kỳ). Để gửi noti đến device của user thì client cần gửi user id và nội dung thông báo cho Notification Service. Ở trong Notification Service sẽ có thêm Template Service chứa các mẫu thông báo có sẵn cho client (Ví dụ: OTP, SMS, Email, thông báo nhanh, Deep link, thông báo chi tiết có thumbnail, content và title) ~> sau khi xử lý sẽ được đẩy vào Kafka.

## **Validation Service,Prioritization Service & Scheduling Service**

Validation Service chịu toàn bộ trách nhiệm cho việc xác thực thông báo theo đúng các quy tắc kinh doanh và định dạng dự kiến.Nó cũng kết hợp với Prioritization Service để quyết định quyền ưu tiên của các Noti (Ví dụ: OTP thường là thông báo được ưu tiên cao nhất ,hoặc là khi user mua phim, tiến hành thanh toán thì thông báo gửi kết quả thanh toán cho khách hàng cũng được ưu tiên cao nhất). Còn Scheduling Service là hỗ trợ cho xử lý các thông báo được lên lịch, có thời gian cụ thể (có thể là hàng ngày, hàng tháng, hàng năm hoặc tùy chỉnh) ~> sau khi xử lý sẽ được đẩy vào Kafka theo thứ tự như sau: thông báo high priority, đến medium priority và cuối cùng là low priority. Điều này có thể đảm bảo các thông báo ưu tiên cao (high priority) luôn được gửi tới khách hàng đầu tiên.

## **Rate Limiter**

Rate Limiter giúp cho việc giới hạn lượng gửi thông báo đến user trong thời gian nhất định. Ví dụ: một user mua gói xem phim VIP yêu cầu chi được gửi 3 thông báo/1 ngày, thì Rate Limiter sẽ tránh các trải nghiệm không tốt (bad experience) cho người dùng ~> nếu thông báo vượt quá yêu cầu thì nó sẽ bị loại bỏ.

Một chức năng có thể tồn tại trong Rate Limiter là Request Counter. Nó sẽ đếm lượng request mà client (bộ phận Sale) gửi tới hệ thống để tiến hành tính KPI trong ngày cho client đó ~> dựa vào đó để tính lương hàng tháng cho họ.

## **Notification Handler & User Preferences**

Notification Handler sẽ gửi request tới User Service để lấy thông tin người dùng như email, số điện thoại,... để gửi thông báo email hoặc SMS. User Preferences giúp người dùng xử lý việc tùy chọn gửi thông báo. Ví dụ: người dùng mua gói xem phim VIP có thể yêu cầu hủy đăng ký nhận thông báo khuyến mại hoặc các loại thông báo khác. Tất cả thông tin về preference của user được lưu trữ trong Notification Database mà được kết nối với User Preferences.

~> Sau khi xử lý sẽ được đẩy vào Kafka.

## **Push Service**

Push Service chứa In-App Notification Handler để xử lý việc gửi thông báo đến các device (iOS, Android, TV và Web). Khi Scale-up hệ thống, có thể gắn thêm các handler khác như SMS Handler, Email Handler để có thể gửi thêm các loại thông báo như SMS hoặc Email. Các Handler này sẽ gửi thông báo tới các nhà cung cấp tương ứng (Vendor) để gửi thông báo đến user.

## **Notification Tracker**

Tất cả thông báo đều sẽ được gửi qua Notification Tracker để theo dõi cho mục đích kiểm tra và tiến hành lưu trữ ở Cassandra. Sau này, chúng ta có thể dựa vào dữ liệu ở đây để làm report, dashboard hoặc tiến hành query để show các thông báo cho Anonymous user và logged user phía giao diện trên app, web,...

## **Bulk Notification Service**

Bulk Notification Service hỗ trợ cho việc gửi thông báo hàng loạt dùng cho việc gửi thông báo cho 1 nhóm người cụ thể. Ví dụ: bộ phận sales muốn gửi thông báo phim hot, phim hay cho nhóm người mua VIP xem phim trong vòng 24h. User Transaction Data là hệ thống lưu trữ các giao dịch của người dùng với phía app/web. Những giao dịch này sẽ được đưa vào Kafka để gửi tới Transaction Data Parser. Hệ thống này hỗ trợ parse các dự liệu theo format chuẩn hoặc convert nó theo format chuẩn để đẩy dữ liệu vào data store như ElasticSearch/MongoDB. Query Engine sẽ tiến hành query dữ liệu từ ElasticSearch/MongoDB để trả lại dữ liệu nhóm user cho Bulk Notification Service ~> gửi thông báo tới đúng nhóm user. Query Engine có thể thực hiện tính năng như Rule Engine (dựa vào Rule của công ty để config), Fraud Engine (để phát hiện Spam, gian lận) hoặc sử dụng dịch vụ Search khác.

# **Về bài test**

Theo thiết kế **(hình 1)** trên, hệ thống có thể đáp ứng như sau:

* Về mặt người dùng (sale/staff), hệ thống hỗ trợ:
  + Có thể cấu hình, chọn khoảng thời gian lên lịch gửi push notification đến user devices: **yêu cầu này** được xử lý ở **Scheduling Service** trong thiết kế trên.
  + Chọn đối tượng gửi:
    - Theo từng user (gửi thông báo chúc mừng sinh nhật, gợi ý phim phù hợp …): **ngày sinh và độ tuổi** của user sẽ được lấy từ **User Service** ~> gửi đến **Notification Handler** ~> giải quyết được yêu cầu này.
    - Theo nhóm user (nhóm độ tuổi, giới tính …): yêu cầu này sẽ thuộc về **Bulk Notification** (thông báo hàng loạt) ~> giải quyết được yêu cầu này.
    - Theo thể loại yêu thích (phim hành động, chính kịch, hài kịch …): yêu cầu này sẽ thuộc về **Bulk Notification** (thông báo hàng loạt) ~> giải quyết được yêu cầu này.
  + Chọn platform gửi:
    - IOS, Android, TV, Web: **In-App Notification Handler** hỗ trợ cho vấn đề này
  + Chọn hình thức hiển thị thông báo trên ứng dụng:
    - Hiển thị thông báo nhanh
    - Deep link
    - Hiển thị thông báo chi tiết (thumbnail, content, title)

~> 3 loại hình thức hiển thị trên được lưu cấu hình ở **Notification DB**

* + Chọn cách thức gửi thông báo:
    - Normal
    - Silent push

~> Vấn đề này sẽ được xử lý ở **In-App Notification Handler**

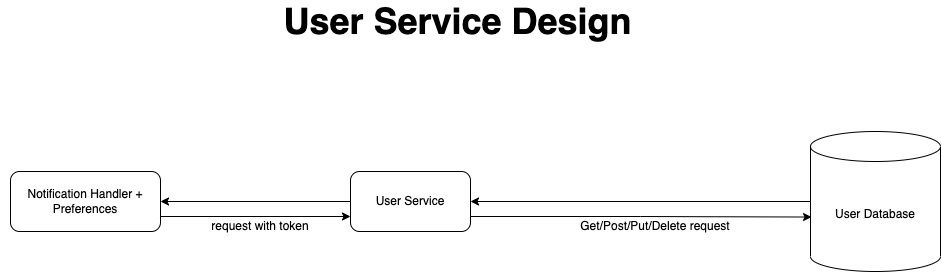
* Về mặt tính năng, hoạt động:
  + Chịu được lượng tải lớn, hoạt động ổn định: Khi có nhiều Noti được gửi đến, thì **In-App Service (hoặc các service khác)** không thể xử lý hết nếu lượng **consumer** tăng lên nhiều, hoặc là lượng **clients** tăng lên đột ngột thì chúng ta có thể đưa nó vào **Kafka** để hỗ trợ **In-App Service (hoặc các service khác)** gửi Noti đến đúng chỗ.
  + Có khả năng mở rộng và tích hợp cao (có thể linh hoạt tích hợp sử dụng nhiều worker service như firebase, airship, apple push notification …): hệ thống có khả năng mở rộng và tích hợp thêm handler khác cho việc gửi noti. Ví dụ: gửi noti trên **WhatsApp** thì tích hợp thêm **WhatsApp Handler**
  + Quản lý được lượng device token của anonymous user và logged user. **(1)**
  + Chạy đúng lịch được lên, cũng như phải gửi tới đúng đối tượng như đã được cấu hình. **(2)**
  + Không được gửi sai đối tượng hoặc gửi duplicate nội dung. **(3)**

~> **Vấn đề (1) (2) (3)** sẽ được xử lý trong bộ 3 **Prioritization Service, Validation Service và Scheduling Service**

* + Đối với các thông báo bằng silent push:
    - Anonymous user:
      * có thể xem được danh sách thông báo hiện có trên hệ thống (vd như phim mới, phim hot …): toàn bộ dữ liệu thông báo của hệ thống sẽ được gửi qua **Notification Tracker** rồi sau đó tiến hành lưu trữ ở **Cassandra**
    - Logged user:
      * Hệ thống hỗ trợ lưu trữ những thông báo mà user đã nhận được.**(4)**
      * User có thể thấy được danh sách các thông báo thông qua giao diện trên app, web … (hỗ trợ api để app, web tương tác).**(5)**
      * User có thể đánh dấu đã đọc, delete tuỳ ý. **(6)**

~> do toàn bộ noti được đưa qua **Notification Tracker** để xử lý lưu trữ tại **Cassandra** nên có thể show được các thông báo này cho người dùng qua app, web bằng REST API ~> **Vấn đề (4) (5) (6)** có thể được xử lý

# **Đặc tả thiết kế chi tiết User Service và công nghệ áp dụng**



**Hình 2:** Đặc tả thiết kế User Service (có thể xem chi tiết hình ở thư mục /Bài test/**Diagram/Đặc tả thiết kế chi tiết User Service.png**).

* User Service sẽ nhận request từ Notification Handler với token đã được validate từ Validation Service.
* Sau đó, User Service sẽ gửi request đến User Database để tiến hành lấy dữ liệu của user trước khi gửi notification.
* Ngoài ra, User Service cũng cấp 1 vài tính năng để quản lý người dùng (Post/Put/Delete features).
* User Database là database lưu trữ dữ liệu của user bao gồm: profile của user như là id, số điện thoại, email,...

**Công nghệ áp dụng cho User Service:**

* MERN Stack:
  + MongoDB là database system
  + Express là back-end framework
  + ReactJS là front-end framework
  + NodeJS là back-end runtime environment. NodeJS là một cross-platform, đóng vai trò như là OS, viết code server-side và có thể hoạt động như một web server.

Lý do lựa chọn Stack này là vì toàn bộ stack giúp xây dựng hệ thống chỉ bằng một ngôn ngữ duy nhất, chạy ở cả front-end và back-end, tiết kiệm thời gian và chi phí.

# **Mô hình triển khai (deployment) đáp ứng operation, scale-up**

* Quy trình deployment cần áp dụng CI/CD cho auto-deployment. Do đó, chúng ta có thể dùng Gitlab CI/CD hoặc AWS CodeCommit/CodeDeploy/CodePipeline cho vấn đề thiết kế CI/CD như sau:

1. Đầu tiên là khởi tạo repository với 2 branch là master và dev.
2. Sau khi dev push code mới nhất lên branch dev, hệ thống sẽ tự động test source code (code quality, performance testing, JUnit test, Container Scanning, Dependency Scanning). Nếu pass thì hệ thống sẽ tự động deploy lên staging environment. Còn fail thì sẽ gửi feedback về cho dev để dev tiến hành sửa.
3. Tester/QA sẽ vào staging environment để làm UAT. Nếu fail thì gửi feedback về cho dev để dev tiến hành sửa source code. Còn pass thì người maintainer sẽ tiến hành merge code từ branch dev sang branch master bằng tay (manually) vì quy trình này sẽ không tự động để tránh lỗi.
4. Hệ thống sẽ tự động kiểm tra source code 1 lần nữa, nếu pass thì đủ yêu cầu để deploy phía production.
5. Khi đủ điều kiện, người maintainer sẽ deploy lên production environment.
6. Nếu có vấn đề sau khi deploy lên production environment, người maintainer có thể tiến hành rollback về commit trước đó.

* Khi cần scale-up về hệ thống, theo hình 1, chúng ta có thể gom Email handler, SMS handler và In-app handler vào Push service và tiến hành deployment. Tiếp theo là gom Rate Limiter, Notification handler và User Preference và hệ thống DB liên quan vào 1 hệ thống service và tiến hành deployment. Sau đó là gom Validation Service, Prioritization Service và Scheduling Service vào 1 hệ thống service và tiến hành deployment. Cuối cùng là gom Notification Service và Bulk Notification Service vào chung 1 hệ thống service và tiến hành deployment.