Câu Hỏi Phỏng Vấn System Design

(Hay gặp nhất trong các buổi phỏng vấn)

KHÁI NIỆM NỀN TẢNG

1 System Design là gì?

- Là quá trình thiết kế các hệ thống quy mô lớn, đáng tin cậy, hiệu quả và dễ bảo trì nhằm đáp ứng các yêu cầu chức năng và phi chức năng.

> Mục tiêu là thiết kế hệ thống có khả năng mở rộng, đáng tin cậy, luôn sẵn sàng và tối ưu chi phí.

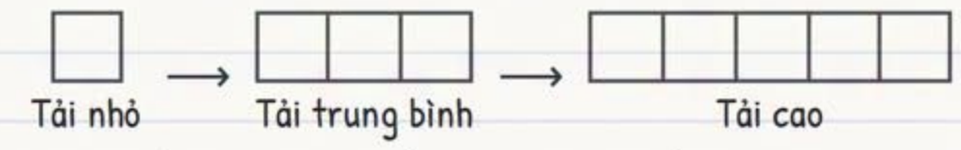
2 Yêu cầu chức năng vs phi chức năng?

- **Functional (chức năng)**: Mô tả hệ thống *làm gì*. (VD: đăng nhập, upload file)
- **Non-Functional (phi chức năng)**: Mô tả hệ thống *hoạt động tốt ra sao*. (VD: Scalability, Availability, Security)

Functional	Non-Functional
· Tính năng · Hành vi · Logic nghiệp vụ	· Hiệu năng · Scalability · Availability · Security

3 Scalability (khả năng mở rộng)?

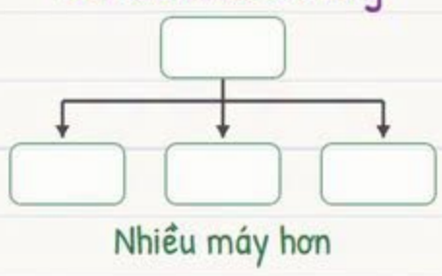
- Khả năng hệ thống xử lý được khối lượng công việc ngày càng tăng bằng cách bổ sung thêm tài nguyên.



Scale Up (theo chiều dọc) vs Scale Out (theo chiều ngang)

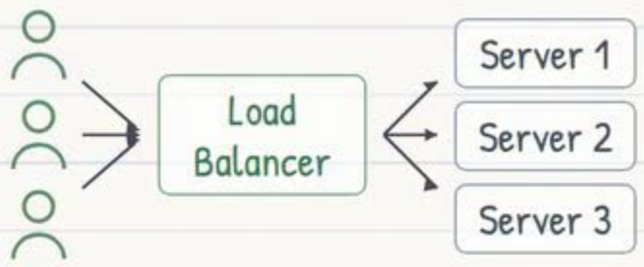
4 Horizontal vs Vertical Scaling?

- **Vertical Scaling (Scale Up)**: Tăng sức mạnh (CPU, RAM) cho chính máy đang có.
- **Horizontal Scaling (Scale Out)**: Thêm nhiều máy mới để chia tải.



5 Load Balancer?

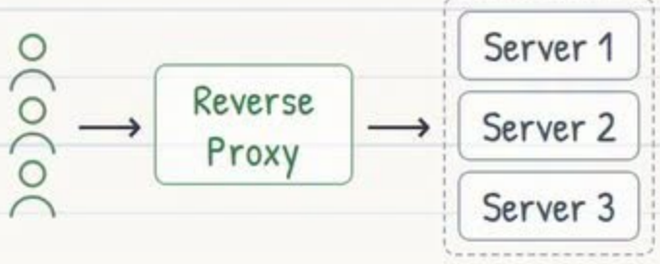
- Phân phối lưu lượng mạng đi vào cho nhiều server, đảm bảo không server nào bị quá tải.



Lợi ích
- High Availability
- Hiệu năng tốt hơn
- Fault Tolerance

6 Reverse Proxy?

- Server đứng trước các backend server, chuyển tiếp request của client tới chúng rồi trả response ngược về cho client.



Lợi ích
- Bảo mật
- Caching
- Nén dữ liệu
- SSL Termination

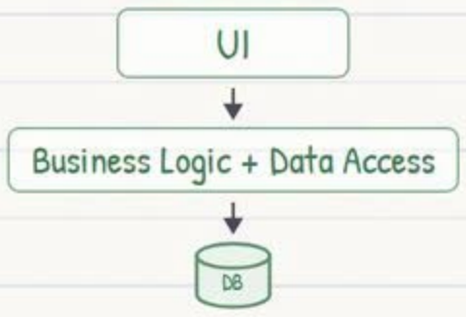
7 API Gateway?

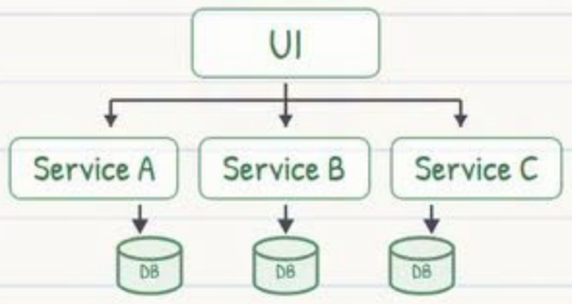
- Điểm vào duy nhất cho mọi request từ client, chịu trách nhiệm định tuyến chúng tới đúng service.

Lợi ích
- Routing
- Xác thực
- Rate Limiting
- Giám sát

8 Monolith vs Microservices?

- **Monolith**: Một codebase duy nhất, triển khai như một khối duy nhất.
- **Microservices**: Ứng dụng được tách thành nhiều service độc lập.





9 High Availability?

- Hệ thống duy trì hoạt động liên tục trong thời gian rất dài mà gần như không có downtime.

Đạt được nhờ Redundancy, Failover, Load Balancing, Replica, triển khai trên nhiều Data Center...

10 Fault Tolerance?

- Hệ thống vẫn tiếp tục hoạt động đúng ngay cả khi một vài thành phần bên trong gặp sự cố.

Đạt được nhờ Replication, Retry Mechanism, Circuit Breaker, thiết kế graceful degradation...

Câu Hỏi Phỏng Vấn System Design

(Hay gặp nhất trong các buổi phỏng vấn)

DATABASE & STORAGE

11 SQL vs NoSQL?

SQL	NoSQL
· Quan hệ (dạng bảng)	· Phi quan hệ (Key-Value, Document, Column)
· Schema cố định	· Schema linh hoạt
· Tuân thủ ACID	· BASE (Eventually Consistent)
· Vertical scaling	· Horizontal scaling
· VD: MySQL, PostgreSQL	· VD: MongoDB, Cassandra, DynamoDB, Redis

Vì sao quan trọng?

Giúp chọn đúng loại database dựa trên use case, nhu cầu mở rộng và mức độ nhất quán dữ liệu.

12 Database Indexing?

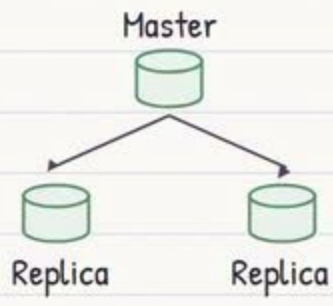
- Index là cấu trúc dữ liệu giúp tăng tốc độ truy xuất dữ liệu.
- Các loại: Primary Index, Secondary Index, Clustered Index, Non-Clustered Index.

Vì sao quan trọng?

Tăng tốc truy vấn tìm kiếm nhưng làm chậm thao tác ghi. Đánh index hợp lý là chìa khoá cho hiệu năng.

13 Database Replication?

- Sao chép dữ liệu từ database này sang database khác để cải thiện tính sẵn sàng và hiệu năng đọc.
- Các kiểu: Master-Slave, Master-Master.

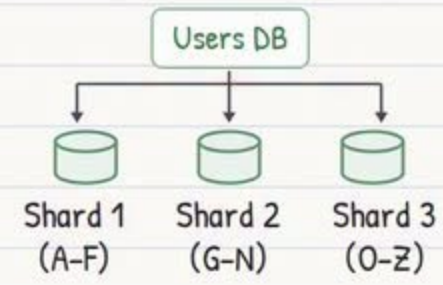


Vì sao quan trọng?

Mang lại tính sẵn sàng cao và khả năng chịu lỗi. Lưu lượng đọc có thể được san sẻ sang các replica.

14 Database Sharding?

- Chia một database lớn thành nhiều phần nhỏ, độc lập nhau gọi là shard.
- Mỗi shard được lưu trên một server riêng biệt.



Vì sao quan trọng?

Hỗ trợ mở rộng theo chiều ngang bằng cách phân tán dữ liệu trên nhiều server.

15 Partitioning vs Sharding?

Partitioning

- Chia một bảng thành nhiều phần nhỏ hơn **trong cùng một database / server.**

Sharding

- Chia dữ liệu ra **nhiều database / server khác nhau.**

Vì sao quan trọng?

Partitioning để mở rộng bên trong một DB. Sharding để mở rộng ra nhiều DB.

16 Read Replicas?

- Các bản sao của database chính, dùng riêng cho thao tác đọc.
- Hữu ích khi mở rộng ứng dụng có lượng đọc lớn.

Vì sao quan trọng?

Cải thiện hiệu năng đọc và giảm tải cho database chính.

17 Định lý CAP?

- Một hệ thống phân tán chỉ có thể đạt được tối đa **2 trong 3** yếu tố sau:

C – Consistency (nhất quán)

A – Availability (sẵn sàng)

P – Partition Tolerance (chịu phân mảnh mạng)

Vì sao quan trọng?

Giúp hiểu rõ các đánh đổi khi thiết kế hệ thống phân tán.

18 ACID vs BASE?

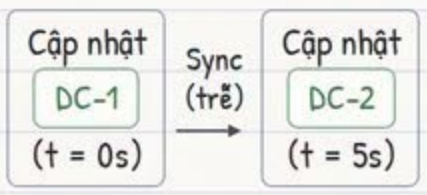
ACID (SQL)	BASE (NoSQL)
· Atomicity – nguyên tử	· Basically Available
· Consistency – nhất quán	· Soft State
· Isolation – cô lập	· Eventually Consistent
· Durability – bền vững	

Vì sao quan trọng?

ACID đảm bảo độ tin cậy cho DB quan hệ. BASE được dùng trong NoSQL để đạt tính sẵn sàng và khả năng mở rộng cao.

19 Eventual Consistency?

- Hệ thống sẽ trở nên nhất quán sau một khoảng thời gian, chứ không nhất quán ngay lập tức.
- Rất phổ biến trong các hệ thống phân tán.



Vì sao quan trọng?

Mang lại tính sẵn sàng cao bằng cách chấp nhận dữ liệu lệch nhau tạm thời.

20 Normalization vs Denormalization?

- **Normalization:** Tách dữ liệu thành nhiều bảng để loại bỏ trùng lặp – ghi nhanh, đọc phải JOIN nhiều.
- **Denormalization:** Cố ý lặp lại dữ liệu để giảm JOIN – đọc nhanh, tốn dung lượng và khó đồng bộ hơn.

Vì sao quan trọng?

Thể hiện khả năng cân bằng giữa tính toàn vẹn dữ liệu và hiệu năng đọc khi thiết kế schema.

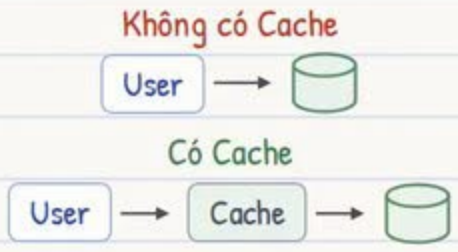
Câu Hỏi Phỏng Vấn System Design

(Hay gặp nhất trong các buổi phỏng vấn)

HIỆU NĂNG & CACHING

21 Caching là gì?

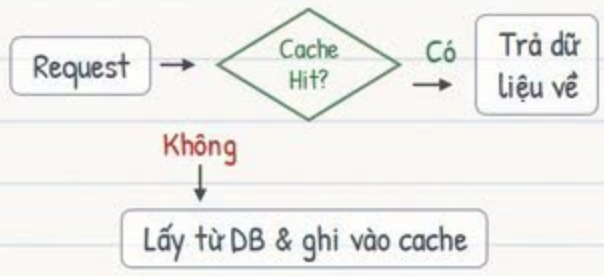
- Lưu dữ liệu hay được truy cập vào bộ nhớ để những request sau được phục vụ nhanh hơn.
- Giảm độ trễ và giảm tải cho database.



Vì sao quan trọng?
Cải thiện thời gian phản hồi, giảm tải database và tăng khả năng mở rộng.

22 Cache Aside Pattern?

- Ứng dụng tìm dữ liệu trong cache trước.
- Nếu miss thì lấy từ DB, ghi vào cache rồi trả về cho user.
- Là chiến lược caching phổ biến nhất.



Vì sao quan trọng?
Đảm bảo dữ liệu luôn mới và giảm bớt các truy vấn database không cần thiết.

23 Write Through vs Write Back?

Write Through	Write Back
• Ghi dữ liệu vào cache và DB **cùng lúc.**	• Ghi vào cache trước, DB được cập nhật sau.
• An toàn hơn, nhưng chậm hơn.	• Nhanh hơn, nhưng có rủi ro mất dữ liệu.

Vì sao quan trọng?
Giúp chọn đúng chiến lược caching dựa trên đánh đổi giữa tính nhất quán và hiệu năng.

24 Redis?

- Kho dữ liệu in-memory, dùng làm database, cache và message broker.
- Hỗ trợ strings, hashes, lists, sets, sorted sets...

Ứng dụng thực tế

- Caching
- Lưu session
- Bảng xếp hạng
- Rate Limiting
- Analytics thời gian thực

Vì sao quan trọng?
Redis cực kỳ nhanh và là mảnh ghép quan trọng khi xây dựng hệ thống hiệu năng cao.

25 CDN?

- Content Delivery Network – mạng phân phối nội dung.
- Phân tán nội dung tới nhiều server đặt ở nhiều vị trí địa lý.
- Đưa nội dung tới người dùng nhanh hơn.

Vì sao quan trọng?
Giảm độ trễ, tăng tốc độ tải trang và chịu được lượng truy cập lớn.

26 Rate Limiting?

- Giới hạn số lượng request mà một user được phép gửi trong một khoảng thời gian.
- Ngăn lạm dụng và bảo vệ hệ thống khỏi quá tải.

Vì sao quan trọng?
Bảo vệ API khỏi bị lạm dụng và đảm bảo chia sẻ tài nguyên công bằng.

27 Message Queue?

- Hàng đợi giữ các message giữa các service, cho phép giao tiếp bất đồng bộ.
- Giúp giảm phụ thuộc lẫn nhau và làm phẳng các đợt tải đột biến.

Vì sao quan trọng?
Cải thiện hiệu năng, độ tin cậy và giảm sự phụ thuộc chặt giữa các thành phần.

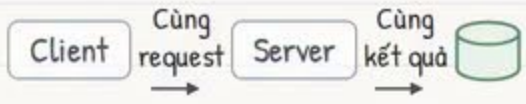
28 Kafka vs RabbitMQ?

Tiêu chí	Kafka	RabbitMQ
Bản chất	Distributed Commit Log	Message Broker
Mô hình	Publish / Subscribe	Queue (Point to Point)
Lưu trữ	Trên đĩa, throughput cao	In-memory, linh hoạt
Hiệu năng	Rất cao	Trung bình
Use case	Event Streaming, Analytics	Task Queue, Workflow

Vì sao quan trọng?
Giúp chọn đúng hệ thống messaging dựa trên use case và yêu cầu cụ thể.

29 Idempotency?

- Đảm bảo nhiều request giống hệt nhau tạo ra kết quả y như khi chỉ gửi một request duy nhất.



Vì sao quan trọng?
Ngăn các thao tác bị lặp lại, ví dụ trừ tiền hai lần khi thanh toán.

30 Bloom Filter?

- Cấu trúc dữ liệu xác suất, trả lời nhanh câu hỏi "phần tử này **chắc chắn không** tồn tại hay **có thể** tồn tại?".
- Rất tiết kiệm bộ nhớ; có false positive nhưng không bao giờ có false negative.

Vì sao quan trọng?
Giúp tránh những lần truy vấn DB vô ích khi dữ liệu chắc chắn không tồn tại.

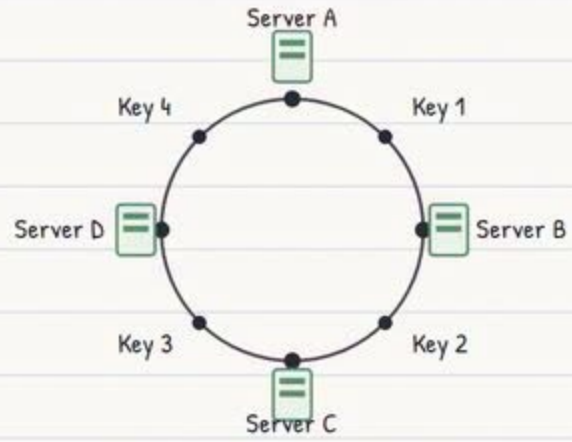
Câu Hỏi Phỏng Vấn System Design

(Hay gặp nhất trong các buổi phỏng vấn)

HỆ THỐNG PHÂN TÁN

31 Consistent Hashing?

- Kỹ thuật băm dùng trong hệ phân tán để trải dữ liệu ra nhiều server.
- Giảm thiểu lượng dữ liệu phải di chuyển khi thêm hoặc bớt node.



Vì sao quan trọng?
Đảm bảo dữ liệu phân bố đều và giảm việc phải cân bằng lại khi số lượng server thay đổi.

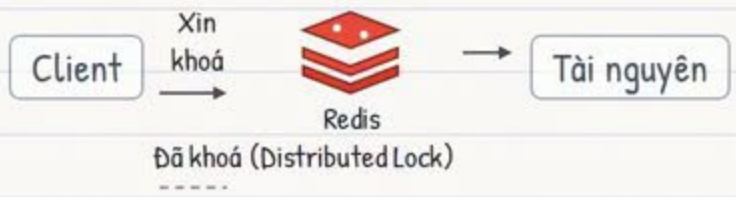
32 Leader Election?

- Quá trình chọn ra một node duy nhất làm leader trong số nhiều node của hệ phân tán.
- Đảm bảo sự phối hợp và tính nhất quán.

Vì sao quan trọng?
Cần thiết cho việc phối hợp, xử lý failover và tránh xung đột giữa các node.

33 Distributed Lock?

- Khoá hoạt động xuyên suốt nhiều tiến trình / nhiều server.
- Đảm bảo tại một thời điểm chỉ một tiến trình được truy cập tài nguyên dùng chung.



Vì sao quan trọng?
Ngăn race condition trong môi trường phân tán, ví dụ khi xử lý thanh toán.

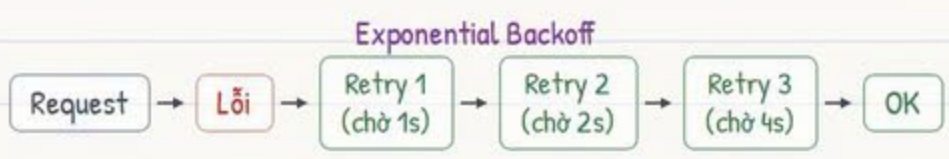
34 Circuit Breaker?

- Ngăn hệ thống liên tục gọi lại những thao tác nhiều khả năng sẽ thất bại.
- Có 3 trạng thái: Closed, Open, Half-Open.

Vì sao quan trọng?
Tăng độ ổn định và ngăn hiệu ứng lỗi lan truyền dây chuyền (cascading failure).

35 Retry Mechanism?

- Tự động thử lại một thao tác đã thất bại sau một khoảng chờ nhất định.
- Nên dùng kèm exponential backoff.



Vì sao quan trọng?
Xử lý được các lỗi tạm thời và nâng cao độ tin cậy của hệ thống.

36 Saga Pattern?

- Dùng để quản lý transaction trải dài trên nhiều microservice.
- Gồm một chuỗi các transaction cục bộ đi kèm transaction bù trừ (compensating).

Vì sao quan trọng?
Giữ được tính nhất quán dữ liệu qua nhiều service mà không cần dùng 2PC.

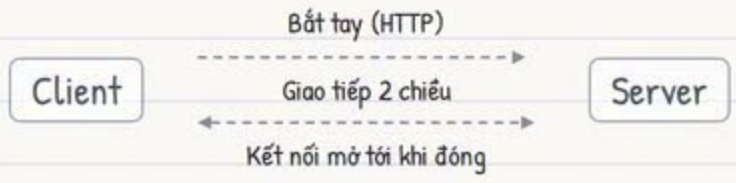
37 Event-Driven Architecture?

- Các thành phần giao tiếp với nhau thông qua sự kiện (event).
- Producer phát ra event, consumer phản ứng lại với chúng.

Vì sao quan trọng?
Mở rộng rất tốt, các thành phần ít ràng buộc nhau và xử lý được theo thời gian thực.

38 WebSockets?

- Giao thức cho phép giao tiếp hai chiều (full-duplex) trên một kết nối TCP duy nhất.
- Lý tưởng cho các ứng dụng thời gian thực.



Vì sao quan trọng?
Cho phép các tính năng real-time như chat, cập nhật trực tiếp và game online.

39 Long Polling vs WebSockets?

Tiêu chí	Long Polling	WebSockets
Kết nối	Lặp lại nhiều request HTTP	Kết nối bền, giữ mở liên tục
Giao tiếp	Server trả lời khi có dữ liệu	Hai chiều, thời gian thực
Độ trễ	Cao hơn	Thấp hơn

Vì sao quan trọng?
Giúp chọn đúng phương thức giao tiếp cho từng bài toán real-time.

40 Service Discovery?

- Cơ chế giúp các service tự tìm được địa chỉ của nhau mà không cần hard-code IP.
- Hai kiểu phổ biến: client-side (Eureka) và server-side (qua load balancer). Công cụ: Consul, etcd, Kubernetes DNS.

Vì sao quan trọng?
Cần thiết khi các instance được scale lên xuống liên tục và địa chỉ luôn thay đổi.

Câu Hỏi Phỏng Vấn System Design

(Hay gặp nhất trong các buổi phỏng vấn)

BÀI TOÁN THIẾT KẾ (HỆ THỐNG THỰC TẾ)

41 Thiết kế URL Shortener

- Rút gọn URL dài thành mã ngắn duy nhất.
- Chuyển hướng từ URL ngắn về URL gốc.
- Chịu lưu lượng đọc lớn, lưu hàng tỷ URL.

Vì sao quan trọng?
Kiểm tra hiểu biết về hashing, sinh ID, redirect và khả năng mở rộng.

42 Thiết kế WhatsApp (hệ thống chat)

- Nhắn tin 1-1 và nhóm theo thời gian thực.
- Gửi tin, trạng thái online và thông báo.
- Mở rộng tới hàng tỷ người dùng.

Vì sao quan trọng?
Bao quát hệ thống real-time, messaging, scaling và tính nhất quán dữ liệu.

43 Thiết kế Instagram Feed

- Hiển thị feed cá nhân hoá gồm ảnh, video, reels.
- Xử lý hàng triệu bài đăng và lưu lượng đọc rất lớn.

Vì sao quan trọng?
Kiểm tra xếp hạng feed, caching, phân phối nội dung và khả năng mở rộng.

44 Thiết kế YouTube

- Upload, lưu trữ, xử lý và stream video.
- Xử lý metadata, tìm kiếm, like, bình luận, gợi ý.
- Phân phối video chất lượng cao ở quy mô lớn.

Vì sao quan trọng?
Bao quát xử lý video, CDN, lưu trữ, tìm kiếm và bài toán quy mô lớn.

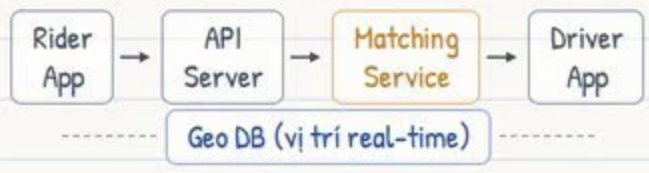
45 Thiết kế Netflix

- Stream phim và TV show tới người dùng.
- Gợi ý nội dung, nhiều thiết bị, tải offline.
- Quy mô toàn cầu với tính sẵn sàng cao.

Vì sao quan trọng?
Kiểm tra kiến trúc streaming, hệ gợi ý, caching và phân phối toàn cầu.

46 Thiết kế Uber

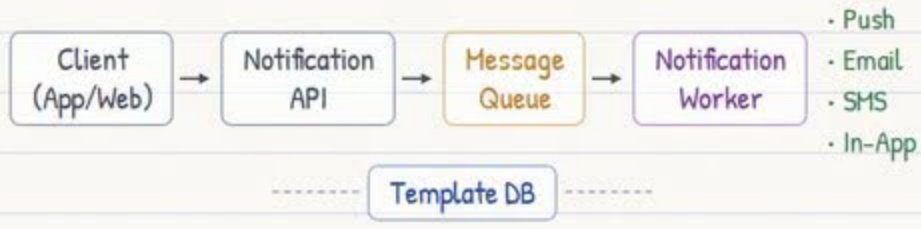
- Đặt xe, ghép tài xế, theo dõi hành trình real-time, thanh toán.
- Xử lý cập nhật vị trí liên tục và mức đồng thời cao.



Vì sao quan trọng?
Kiểm tra hệ thống phân tán theo địa lý, dữ liệu real-time, thuật toán ghép cặp và scaling.

47 Thiết kế hệ thống Notification

- Gửi thông báo qua push, email, SMS, in-app.
- Xử lý khối lượng lớn và cơ chế retry.
- Hỗ trợ lên lịch gửi và template.



Vì sao quan trọng?
Bao quát độ tin cậy, cơ chế retry, đa kênh gửi và khả năng mở rộng.

48 Thiết kế Payment Gateway

- Thanh toán an toàn, nhiều phương thức, hoàn tiền, lịch sử giao dịch.
- Đảm bảo độ tin cậy và tính nhất quán.

Vì sao quan trọng?
Kiểm tra bảo mật, độ tin cậy, xử lý giao dịch và tích hợp bên thứ ba.

49 Thiết kế ứng dụng Chat (real-time)

- Nhắn tin real-time, chat nhóm, hiện "đang gõ", trạng thái đã gửi.
- Lưu tin nhắn và đồng bộ lại khi user offline.

Vì sao quan trọng?
Kiểm tra giao tiếp real-time, WebSockets, presence và độ tin cậy khi gửi tin.

50 Thiết kế Rate Limiter phân tán

- Giới hạn số request theo user / API key trên nhiều server cùng lúc.
- Thuật toán phổ biến: Token Bucket, Leaky Bucket, Sliding Window Log / Counter. Thường dùng Redis làm bộ đếm dùng chung.

Vì sao quan trọng?
Kiểm tra khả năng quản lý state dùng chung, xử lý đồng thời và tối ưu độ trễ.