**YouTube:**

1. **Usecase**

Follow

🔥 SYSTEM DESIGN TEMPLATE 🔥  
  
I came across this System Design Interview template on Leetcode a few years ago. Say, the interviewer has asked you to System Design TinyUrl, Instagram, etc. So you break down your answer in the following sections and also manage the time accordingly.  
  
So you will start with,  
  
FEATURE EXPECTATIONS [5 min]  
(1) Use cases  
(2) Scenarios that will not be covered  
(3) Who will use  
(4) How many will use  
(5) Usage /Access patterns  
  
  
ESTIMATIONS [5 min]  
(1) Throughput (QPS for read and write queries)  
(2) Latency expected from the system (for read and write queries)  
(3) Read/Write ratio   
(4) Traffic estimates  
- Write (QPS, Volume of data)  
- Read (QPS, Volume of data)  
(5) Storage estimates  
(6) Memory estimates  
- If we are using a cache,  
 what is the kind of data we want to store in cache  
- How much RAM and how many machines do we need for us to achieve this ?  
- Amount of data you want to store in HDD/SSD  
  
DESIGN GOALS [5 min]  
(1) Latency and Throughput requirements  
(2) Consistency vs Availability [Weak/strong/eventual => consistency | Failover/replication => availability]  
  
HIGH LEVEL DESIGN [5-10 min]  
(1) APIs for Read/Write scenarios for crucial components  
(2) Database schema  
(3) Basic algorithms  
(4) High-level design for Read/Write scenario  
  
  
DEEP DIVE [15-20 min]  
(1) Scaling the algorithm  
(2) Scaling individual components:  
- Availability, Consistency, and Scale story for each component  
- Consistency and availability patterns  
(3) Think about the following components, how they can help  
a) DNS  
b) CDN [Push vs Pull]  
c) Load Balancers [Active-Passive, Active-Active, Layer 4, Layer 7]  
d) Reverse Proxy  
e) Application layer scaling [Microservices, Service Discovery]  
f) DB [RDBMS, NoSQL]  
-> RDBMS  
  - Talk about Replication, Sharding,     
   Denormalization, Partitioning, SQL Tuning, etc  
-> NoSQL  
  - Talk about Key-Value, Wide-Column, Graph, Document Db, etc  
  
g) Caches  
- Client caching, CDN caching, Application server caching, Database caching, Application caching, Cache @Query level, Cache @Object level  
- Caching Patterns:  
  e.g. Cache aside, Write through, Write behind  
- Cache Eviction Policies  
  e.g. LRU, LFU, LIFO, FIFO, RR, etc  
  
h) Asynchronism  
- Message queues  
- Task queues  
- Back pressure  
  
i) Communication Protocols  
- TCP, UDP, REST, RPC  
  
JUSTIFY [5 min]  
(1) Throughput of each layer  
(2) Latency caused between each layer  
(3) Overall latency justification  
(4) Also how you can handle the scaling of each layer in the fut

**GENERIC:**

**System Design:**

* Load balancer:
  + Reverse Proxy
  + Response Aggregation
  + SSL Termination
  + L4 vs L7 level
  + Active-Active, Active-Passive

Cache:

Client, CDN,Webserver Cache,DB,Application Cache

At DB query Level, At Object level

Asynchronous:

Message Queue, Task Queue, Back Pressure

Communications:

TCP, UDP,RPC,REST

API Gateway

CDN:

Push vs Pull

Application Layer:

Microservices

Service Discovery

**Microservices Patterns:**

[**https://docs.microsoft.com/en-us/azure/architecture/microservices/design/interservice-communication**](https://docs.microsoft.com/en-us/azure/architecture/microservices/design/interservice-communication)

A service mesh is a software layer that handles service-to-service communication.

**Service Mesh??**

* Load balancing at the session level, based on observed latencies or number of outstanding requests. This can improve performance over the layer-4 load balancing that is provided by Kubernetes.
* For Distributed Transactions; using the Compensating Transactions:<https://docs.microsoft.com/en-us/azure/architecture/patterns/scheduler-agent-supervisor>

[**https://docs.microsoft.com/en-us/azure/architecture/microservices/design/api-design**](https://docs.microsoft.com/en-us/azure/architecture/microservices/design/api-design)

* Public APIs that client applications call.
* Backend APIs that are used for interservice communication.

<https://docs.microsoft.com/en-us/azure/architecture/patterns/backends-for-frontends>

(For the different client application as browser or mobile)