System Designing!

System design is the process of designing the elements of a system such as the architecture, modules & components, the different interfaces of those components & the data that goes through that system.

types of system design

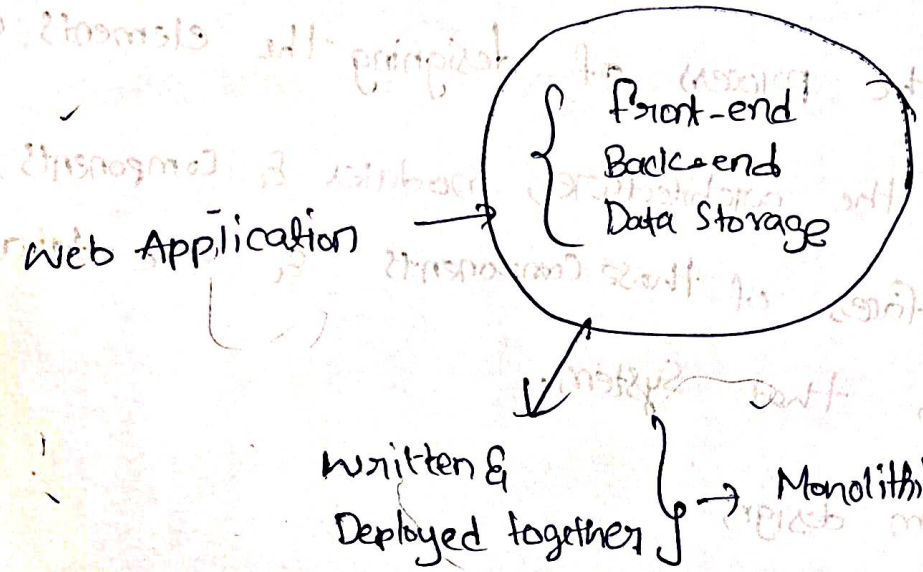
2 types

1. LLD (Low level Design)
2. HLD (High level Design)

HLD	LLD
Describes the main components that would be developed for the resulting product	Describe the design of each element mentioned in the High level Design of the System,
The System architecture details, database design, services & processes, the relationship b/w various modules & features	classes, interfaces, relationships b/w different classes, & actual logic of the various components.

Monolithic Architecture

Architecture (Internal design details for building the applications)



Monolithic Arch. has less complexity => Easier to Understand => Higher Productivity.

Monolithic system is also known as Centralised System

Adv

1. When we are just Starting
2. In Monolithic archi., all the modules are present in the Single System, So they require fewer network calls as compared to other architectures.
3. It is comparatively easier to secure Monolithic System.
4. Integration testing is easier.
5. Less Confusion.

Disadvantages

1. In Monolithic architecture, every module is combined in a single system, so if there is an error or bug in a single module, it can destroy the complete system.

2. In monolithic arch., whenever a single module is upd the whole system need to be updated to reflect the changes to the users. All modules are present in a single system & are connected to one another, so the whole system need to be updated.

3. In monolithic arch., if there is any change in a single module's programming lang. or framework, it will affec the entire system. The entire system needs to be changed because every module is interlinked & ~~this~~ tightly coupled.

Distributed Systems?

→ A distribution system is a collection of multiple individual systems. Connected through a network, tend share resources, communicate & coordinate, to achieve common goals.

Adv

→ Scalable.

→ no single point of failure.

→ Low latency.

Dis

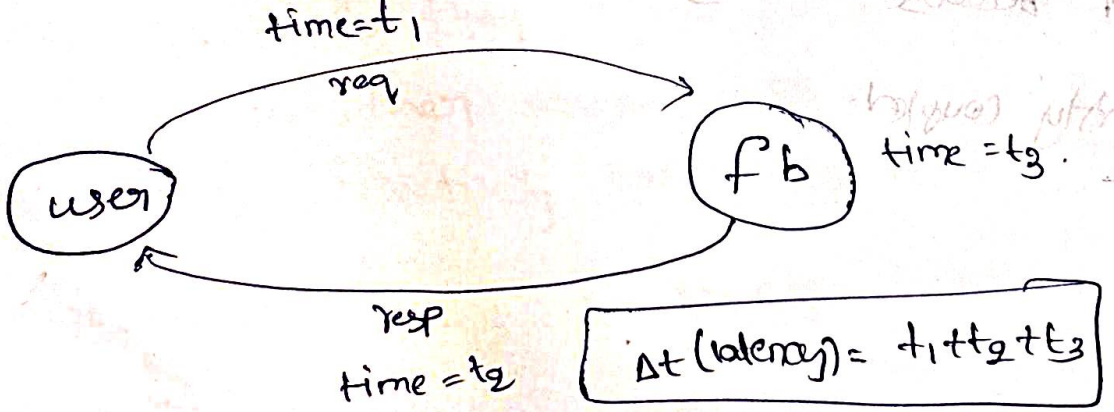
→ complex.

→ Mangement req

→ Difficult to Secure.

→ message may be lost in b/w nodes.

what is latency?



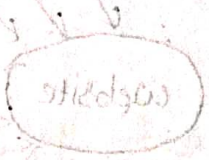
Δt (latency) = $t_1 + t_2 + t_3$

Latency = Network delay + computational delay

Reducing Latency

1. Caching
2. CDN (Content Delivery Network)
3. Upgrading

Mono → Lat(Mono) < Lat(Distr)



Throughput → the volume of work or information flowing through a system.

→ Throughput is the amount of data transmitted per unit of time. It is the process flow rate.

→ Throughput is measured in bits per second i.e bps

throug(Mono) < throug(Distri)

Causes of Low Throughput.

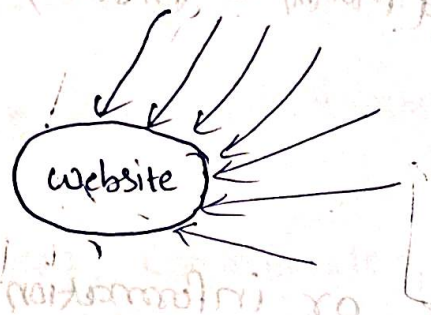
1. Latency
2. protocol overhead.
3. Congestion

Improve

→ CDN
→ Caching
→ D.S
→ Load Balancer.
→ Improve resources

Availability?

SPOF - Single point of failure.



Monolithic < Distributed System

How to increase the availability

1. Replication → Includes redundancy, but involves the copying of data from one node to another or the synchronization of state b/w nodes.
2. Distributed System
3. Redundancy → Redundancy is the duplication of nodes, in case of some of them are failing.

Consistency

Factors Improving Consistency

→ Improving network Bandwidth
→ Stop the read.
→ Replication based on Distance aware strategy.

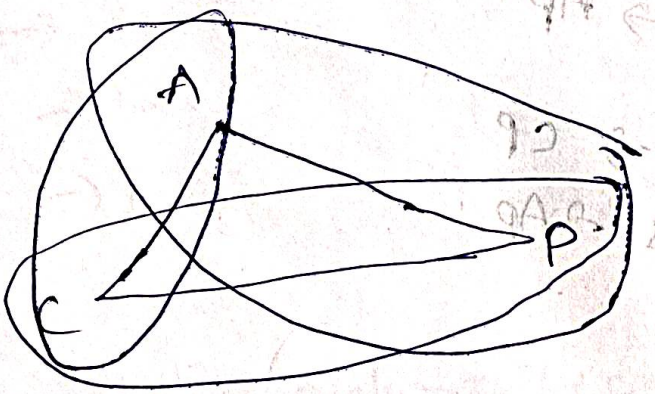
Types of Consistency

~~When the System doesn't allow read operation~~
Strong Consistency
Eventual consistency
Weak Consistency.

CAP Theorem (Consistency, Availability, Partition of Tolerance)

→ for a distributed system, the CAP Theorem states that it is possible to attain only two properties & the third would be always compromised.

→ The system requirements should define which two properties should be chosen over the rest.



✓ Consistency

✓ Availability

Compromised.

Partition of Tolerance.

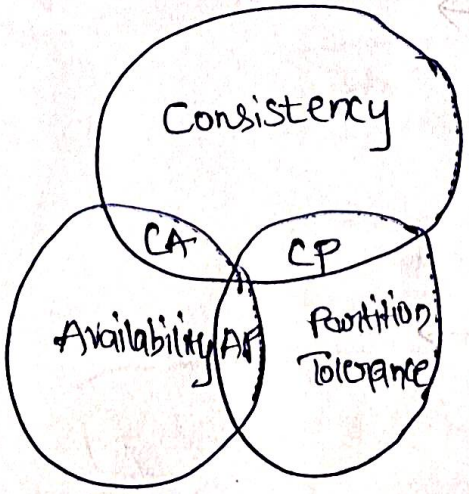
C

A

P

C

A



→ Partition Tolerance is important.

→ So, now the choice is b/w Consistency & Availability.

a. Blog website → AP

b. Multiplayer online games → ~~Consitency P~~ CP

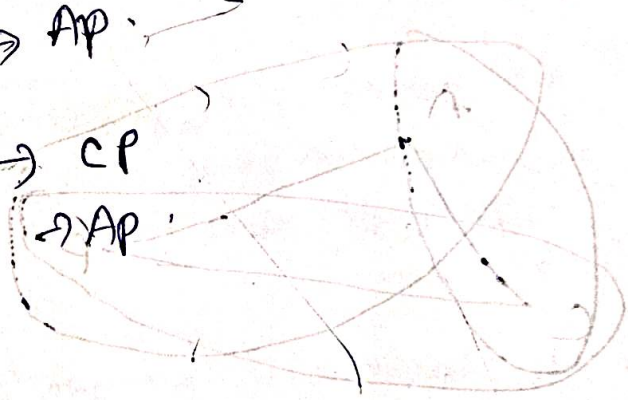
c. Stock trading platforms → CP

d. Video Streaming site → AP

e. Ticket booking System → CP

f. Vide chat applications → AP

g. Bank → CP

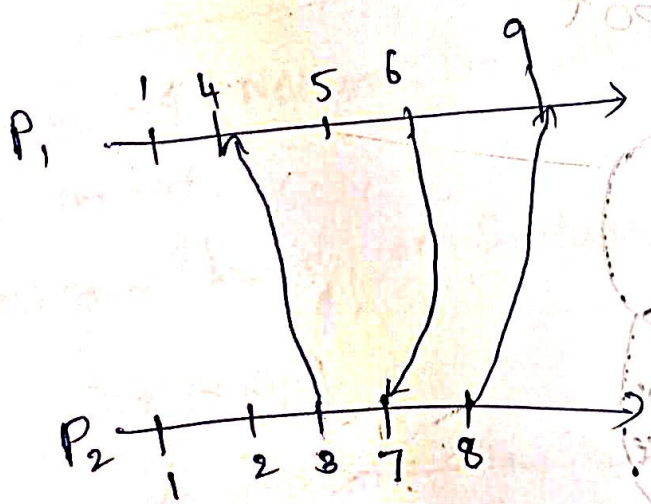


Lamport Logical Clock)

Time ⟶ Sequence;

event → c++

max(P_1 , P_2)



Difference b/w Horizontal & Vertical Scaling

Vertical

Pros → easy Impl
less power.
management is easy.

Cons → Spof
→ limit.
→ price

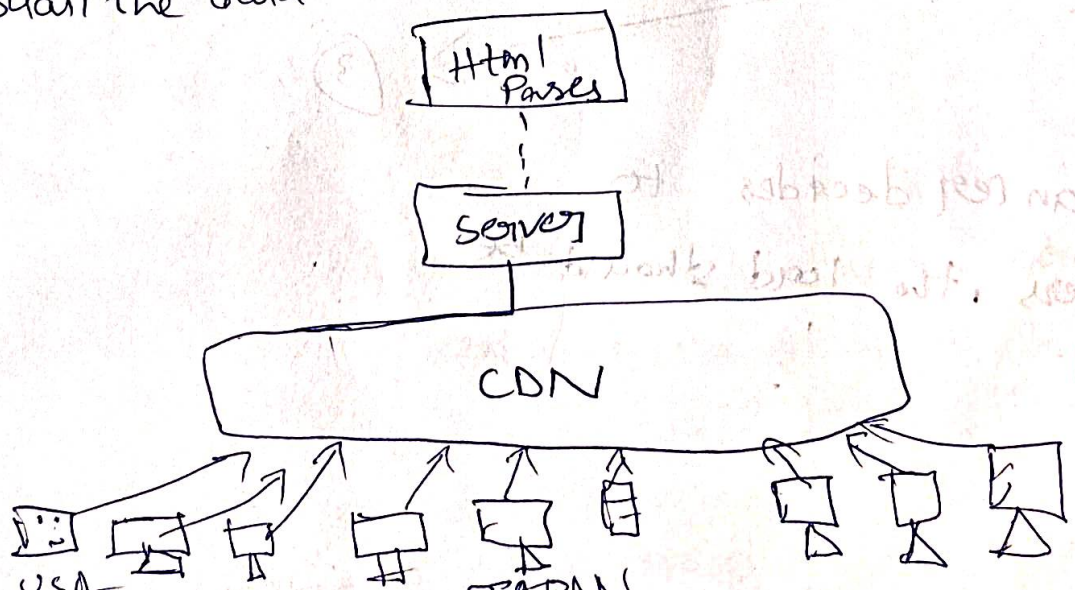
Horizontal

Pros Overcome Cons of Vertical scaling

Cons management is tough
security.

Content Delivery Network (CDN)

→ Local Regulations
→ High Latency issue
→ Eg:- Amazon Cloudfront.
→ usual the data is static data, Videos, Images, Audios, files.



Redundancy

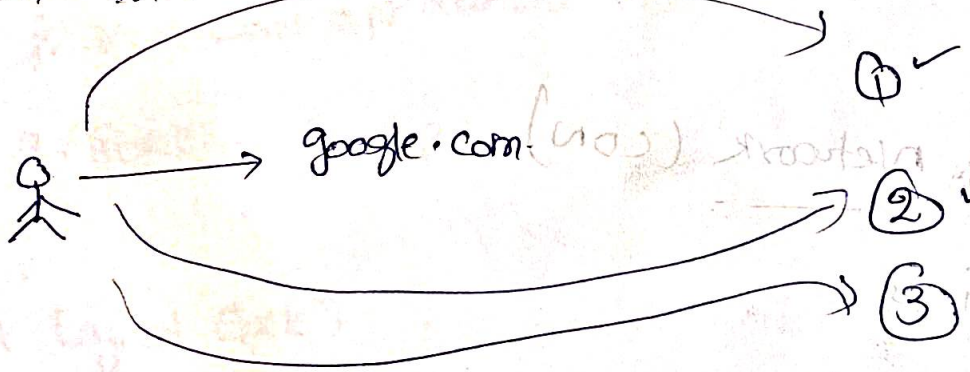
→ Duplication of nodes

Two types

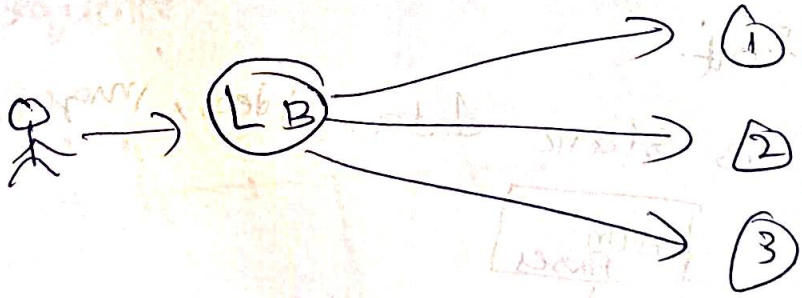
Active Redundancy

→ When each unit is operating (active & responding to the action.

→ Multiple nodes are connected to a load balancer, & each unit receives an equal load.



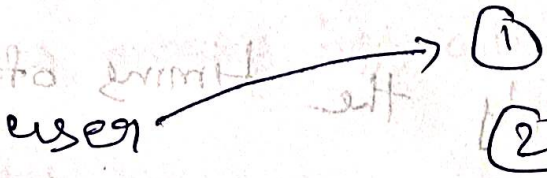
→ Here all three servers are active



Load Balancer decides to which servers, the load should be distributed.

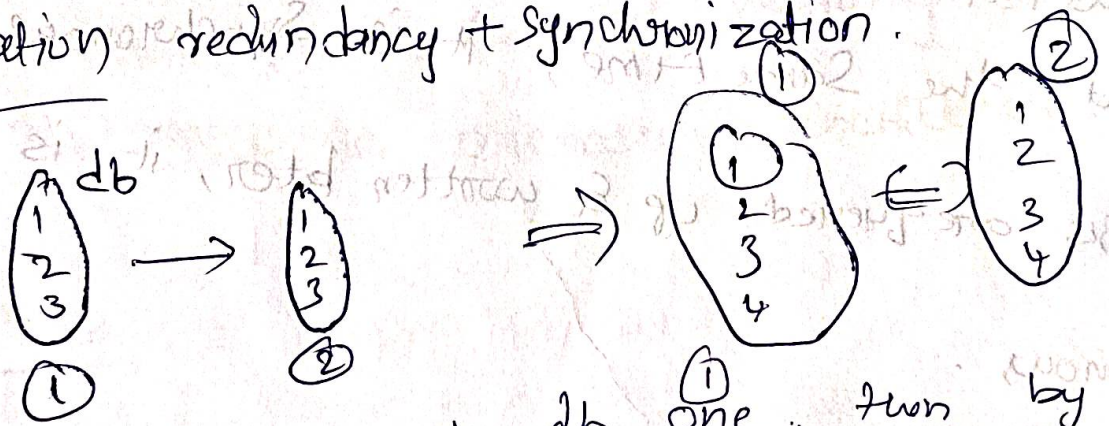
Passive Redundancy

→ Passive Redundancy is considered when one is active or operational & other is not operating. During the breakdown of the active node, the passive node maintains availability by becoming the active node.



→ Two nodes or servers. one is active, one is passive.

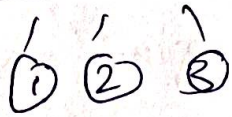
Replication redundancy + Synchronization.



Iam adding ④ to db one. then by synchronization it is added to db② as well.

Two types

Active



three servers or nodes are active state, & read & write

Passive

every read & write → master

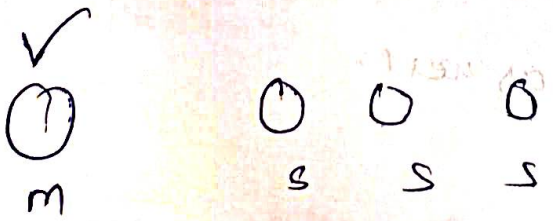
→ If master goes down one slave becomes master.

→ master-slave replication can be either "synchronous or asynchronous".

→ The difference is simply the timing of propagation of changes. ~~If the changes are made to~~

→ Synchronous: If the changes are made to the master & slave at the same time, it is Synchronous.

→ If changes are queued up & written later, it is Asynchronous.



Load Balancers

Roles of Load Balancers

1. The load distribution is equal over every node.
2. Health check (if the node is not operational, the request is passed to another)
3. Load balancers ensures high scalability, high throughput & high availability.

When to use it & when not to?

In monolithic or in case of vertically scaled system we don't need it. But in microservice architecture we need it.

Challenges

SPOF - During a load balancer malfunctioning, the communication b/w clients & servers would be broken.

To solve this issue, we can use redundancy. The system can have an active load balancer & one passive load balancer.

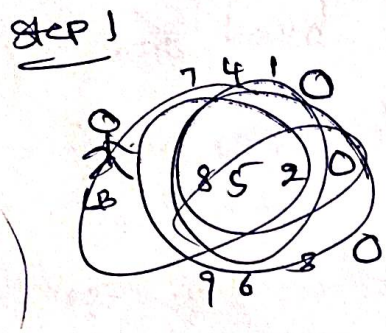
Adv

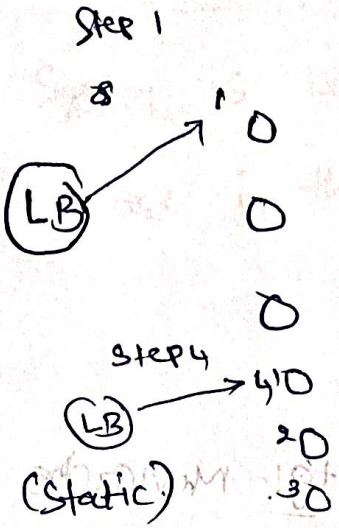
1. optimization
2. Better user experience.
3. Prevents Downtime.
4. Flexibility
5. scalability
6. Redundancy.

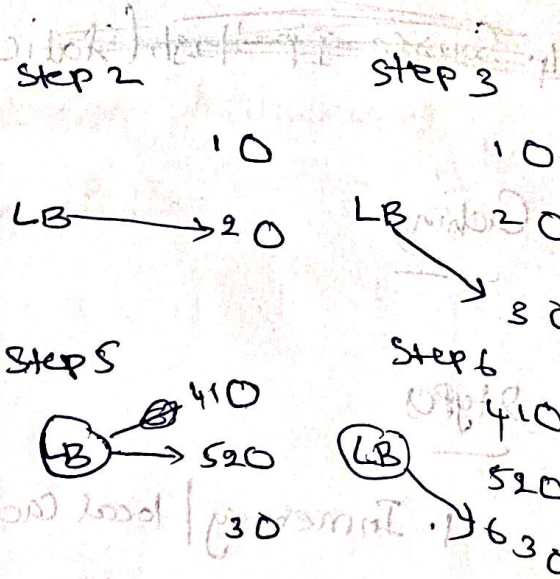
Load Balancing Algorithms

1. Round Robin (Static) - rotation fashion.
2. weighted Round Robin (Static) - It is similar to Round Robin when the servers are of different capacities. (some can have better resources, others might not have)
3. Ip Hash algorithm (Static) - The servers have almost equal capacity, & the hash function (Input in sourceIP) is used for random or unbiased distribution of requests to the nodes.
4. Source IP Hash (Static); Combines the server & client's source & destination IP addresses to request distribution.
5. Least Connection Algorithm (Dynamic) - client requests are distributed to the application server with the least number of active connections at the time the client request is received.
6. Least Response Time (Dynamic) - The request is distributed based on the server which has the least Response time.

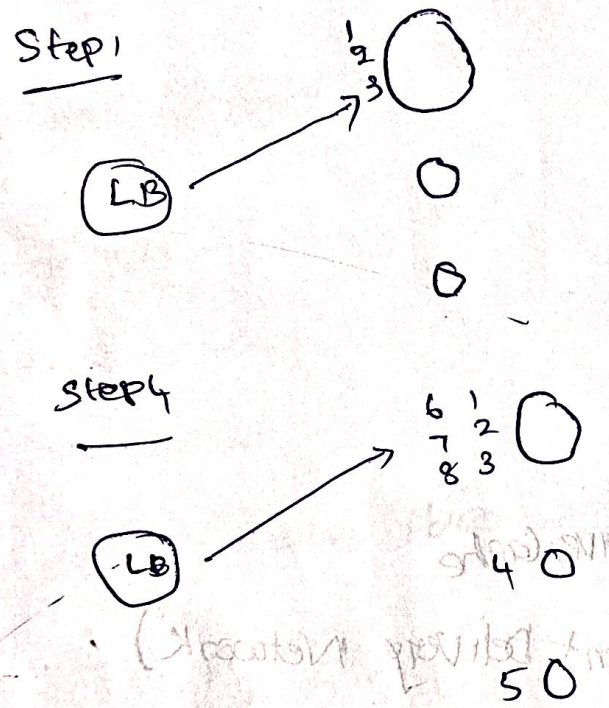
1. Round Robin (Static)

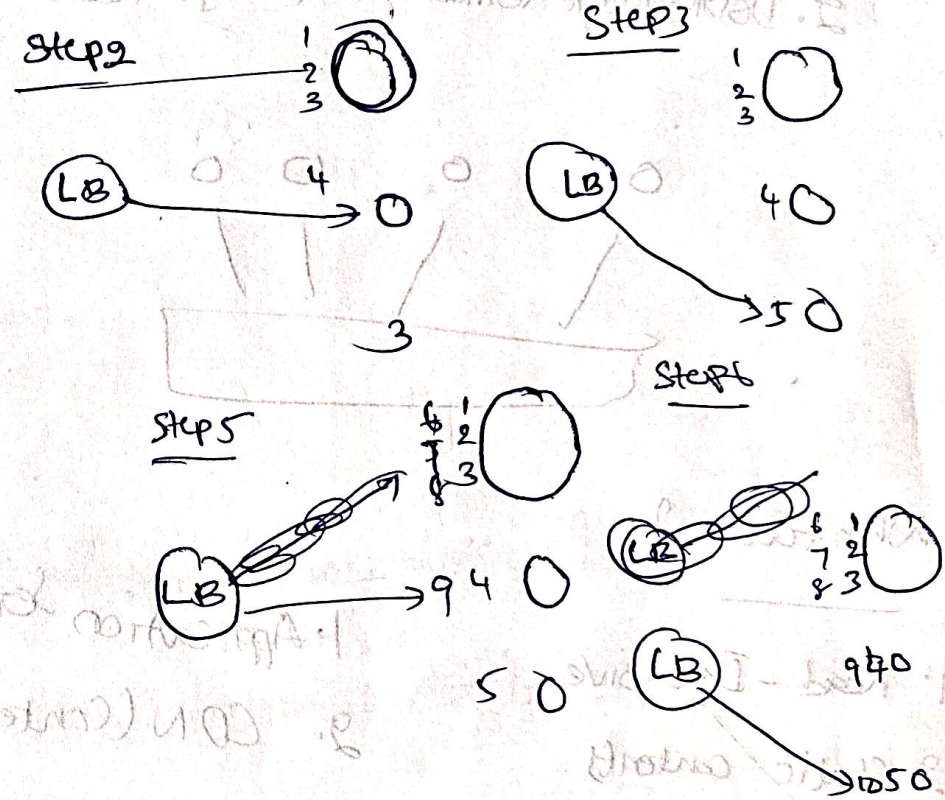




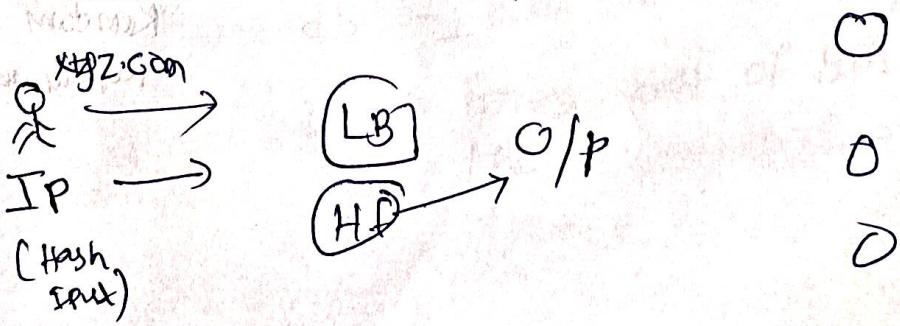


2. Weight Round Robin (Static)





3. IP Hash Algorithm (Static)

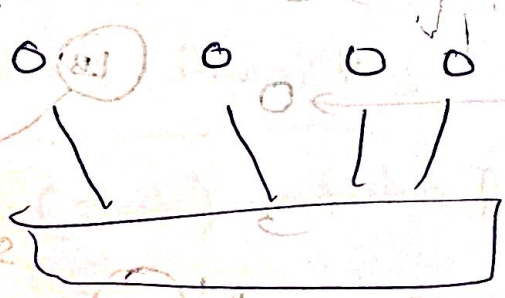


~~1. Source IP Hash (Static.~~

Caching

2 Types

1. In memory / local cache Eg:- Memcache
2. Distributed / external cache Eg:- Redis



When to use ?

1. Read - Intensive
2. Static contents

1. Application Server Cache
2. CDN (Content Delivery Network).

Cache Eviction Strategies (LRU, LFU, MRU, LIFO, FIFO & RR)

RR → Random Replacement

File Based Storage System/ File based DBMS.

→ A file based Storage System is a database management System where data is stored in the form of files.

Challenges

1. Data Redundancy.
2. Poor Security
3. Slow

~~RDM~~ · RDBMS

Disadva:

1. Rigid schema
2. High cost
3. Scalability issues (Horizontal scaling/sharding is very difficult).

NoSql (non SQL or non-relational database).

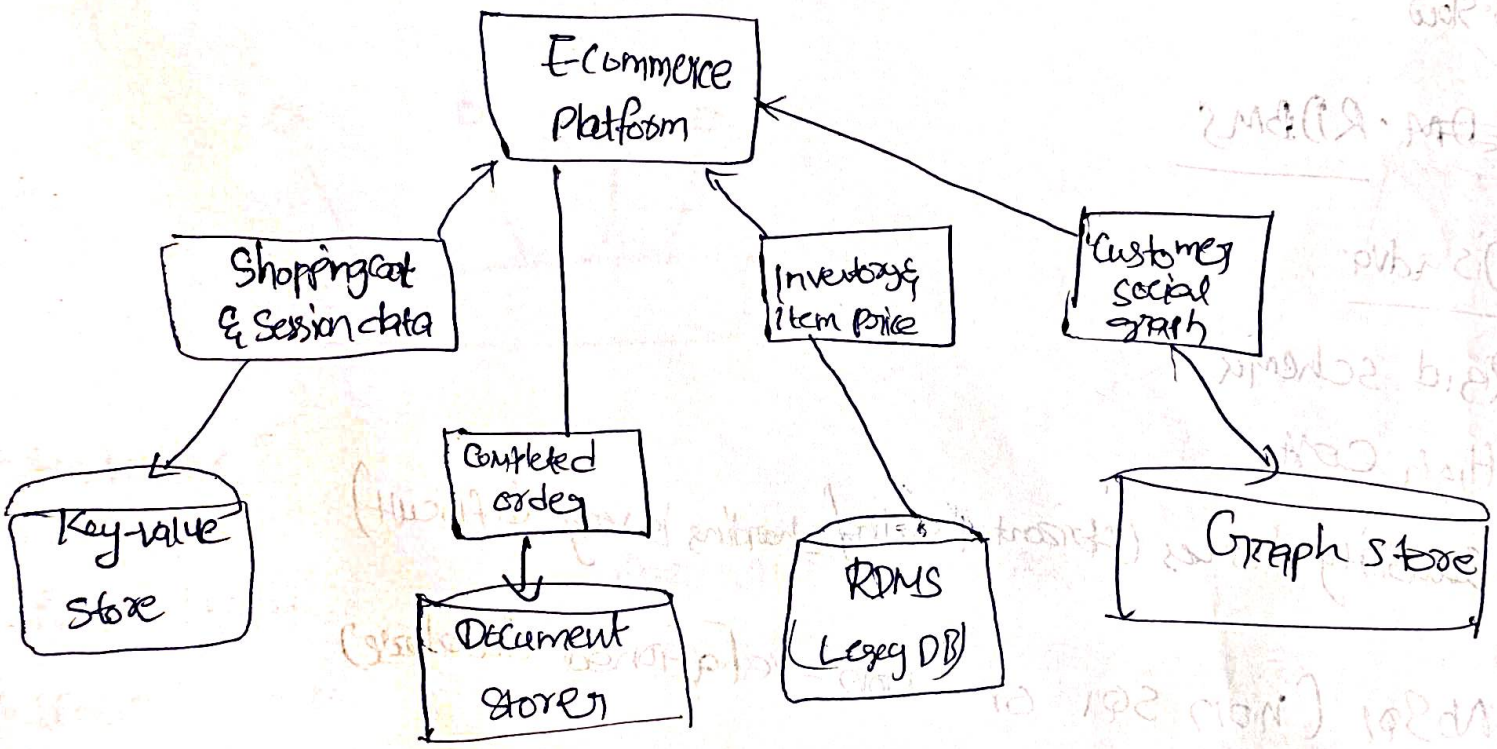
4 types

1. Key Value db → Key Value db generally used for caching (Redis)
2. Document db → Bring best of both RDMS & NOSql. It combines the relationship concept from RDBMS & dynamic schema & horizontal scaling from NoSql databases
3. Columnar db (Cassandra)
4. Graph db → when the ~~eg~~ points are connected then we use

for Payments → we don't use NoSQL, we only use RDMS, why?

Polyglot persistance.

→ If your application is using different databases for different



Normalization Indexing.

B-tree data Structures is used to store indexing as it is a multilevel format of tree based indexed which has balanced binary search trees.

→ Read-only, then only use Indexing.

Synchronous Communication (Blocking call)

→ To acheive Consistency.

→ Transaction.

Asynchronous Communication (Non Blocking call)

Message Based Communication

Producer
Consumer
Agent

} Kafka
RabbitMQ.

Web Server.

→ Tools or programs that help keep the web application always up & running.

Communication protocols

Styles

1. Push
2. Pull/Polling
3. Long Polling
4. Socket
5. Server Sent events.

Rest - Representational State Transfer.

SOA - Service oriented Architecture.

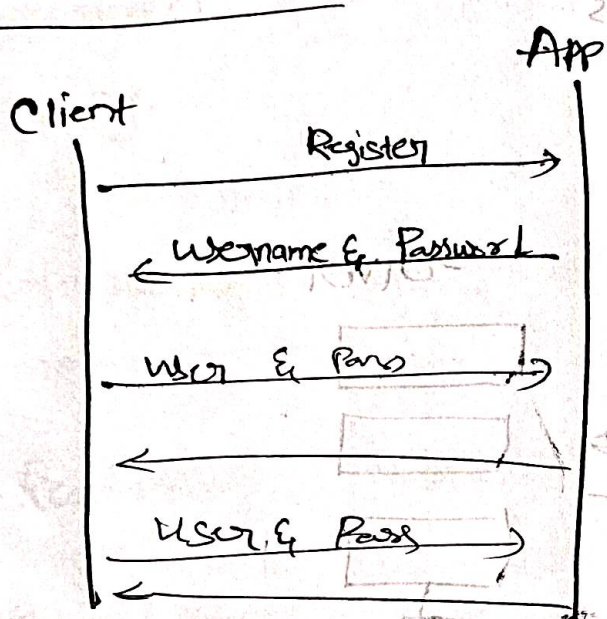
is a style of arch. that promotes loose coupling and granular applications to make the components of the software reusable.

Authentication → Who are you?

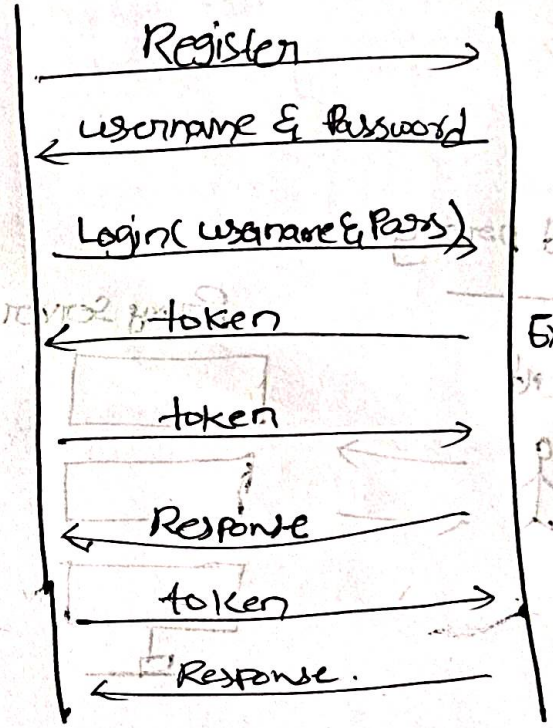
Vs

Authorization → What can you do?

Basic Authentication

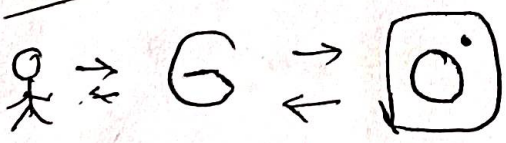


Token based Auth



OAuth - open Authorization.

Sign up

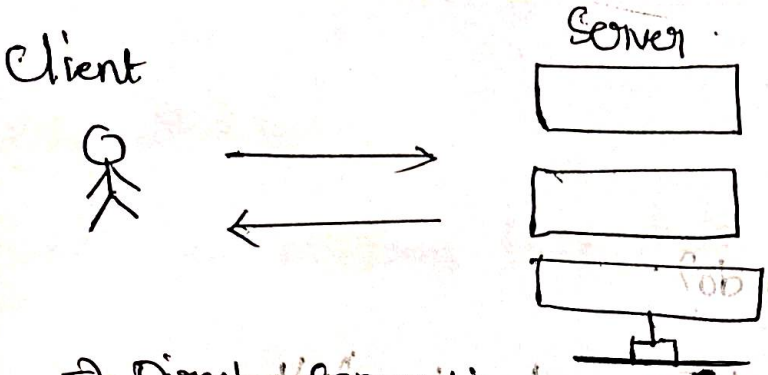


Proxies

→ A proxy server is a hardware or a piece of software that is placed b/w a client & an application to provide intermediary services in the communication

2 types

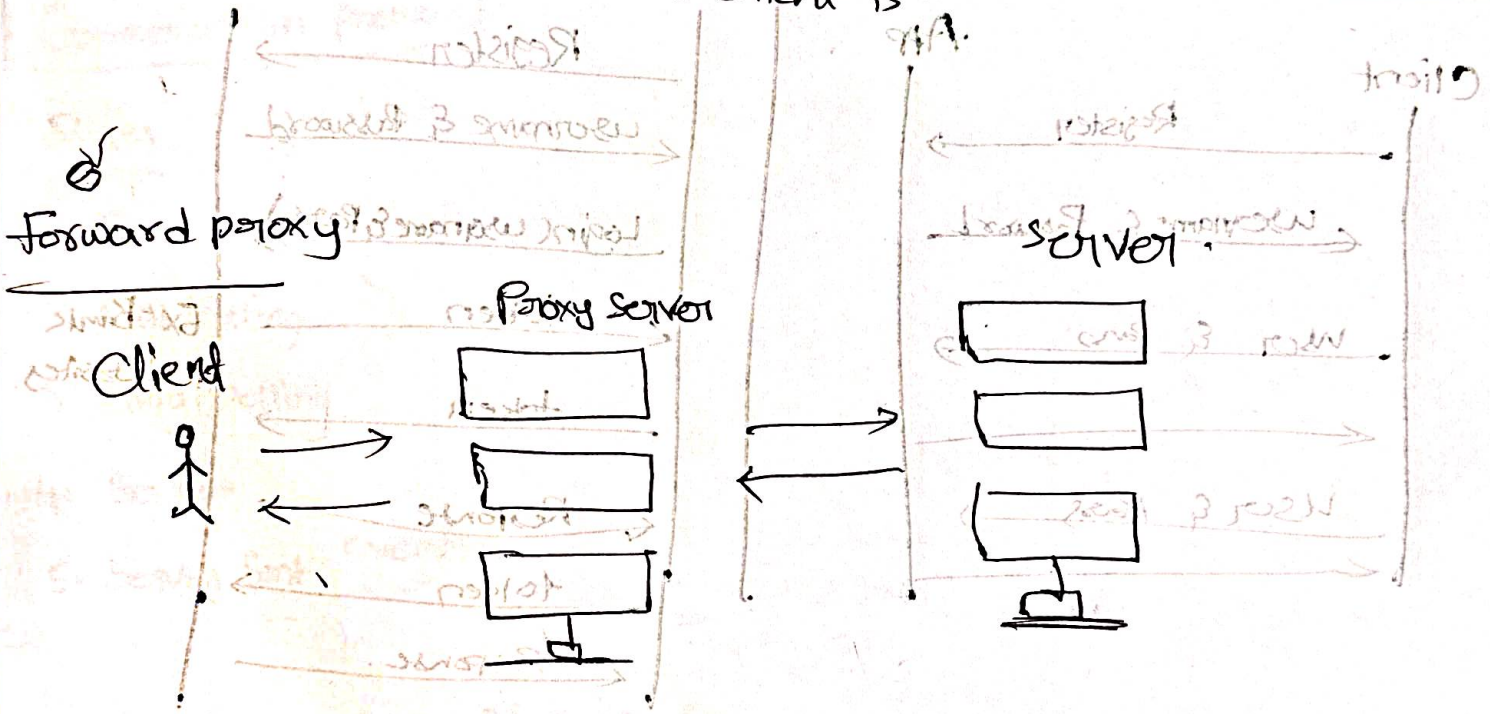
1. Forward proxy
2. Reverse proxy.



→ Direct connection b/w client & server.

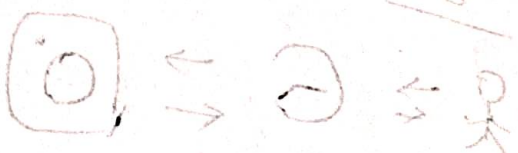
→ Server knows who the client is.

Forward proxy



→ Server doesn't know the client.

Reverse Proxy



Forward proxy hides the client

Reverse Proxy hides the server

URL Shortener

Functional

→ Shorten
→ Redirection
→ Expiry

Non-functional

→ High available
→ Low Latency
→ URL's should not be Predictable.

Estimating the System Capacity

Storage RAM

Read heavy or write heavy

Read calls. Write calls.

100 : 1

Every Month

1 million new URLS

Every month

100 million redirections.

Queries Per Sec (QPS)

$$\frac{100,000,000}{30*24*3600} \approx 50 \text{ QPS}$$

Every month ↓

1 million new URLs

Expiry ↓ 10 years

no. of URLs ↓ × no. of months ↓ × no. of years ↓ × Bytes ↓

1,000,000 × 12 × 10 × 500 = 60GB

RAM:

1 day TTL

~~50 × 60~~

50 × 24 × 3600 ≈ 5m QPS

95% of 5 million queries

↓

1.25 million

↓

1.25 million × 500 bytes

1 GB

→ Read Intensive

→ Highly available

→ Low latency

→ Security (not every can shorten Url)

HLD

- APIs
- Database
- Algorithm
- Design.

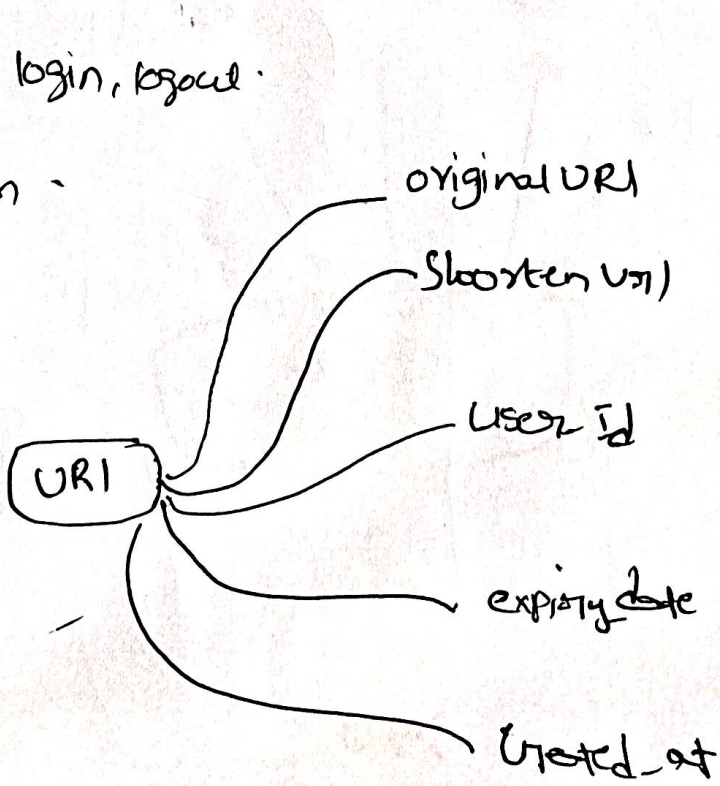
String shortenUrl(String originalUrl, long userId, String apikey, Date expiry);

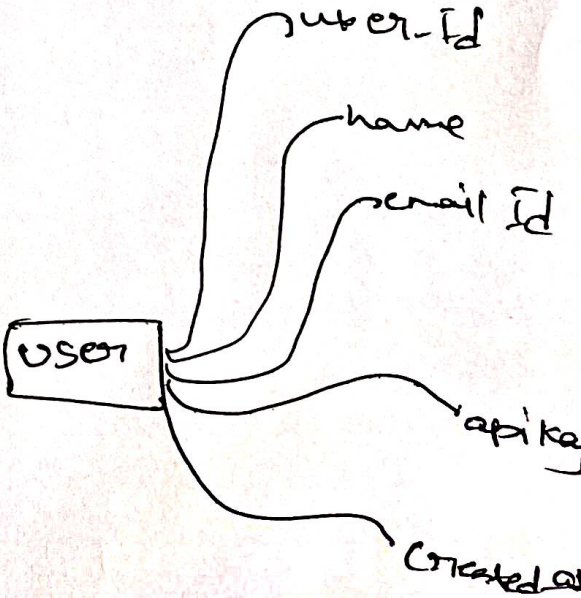
Boolean deleteUrl(String apikey, String originalUrl, String shortUrl);

// signup, login, logout.

// redirection.

MongoDB



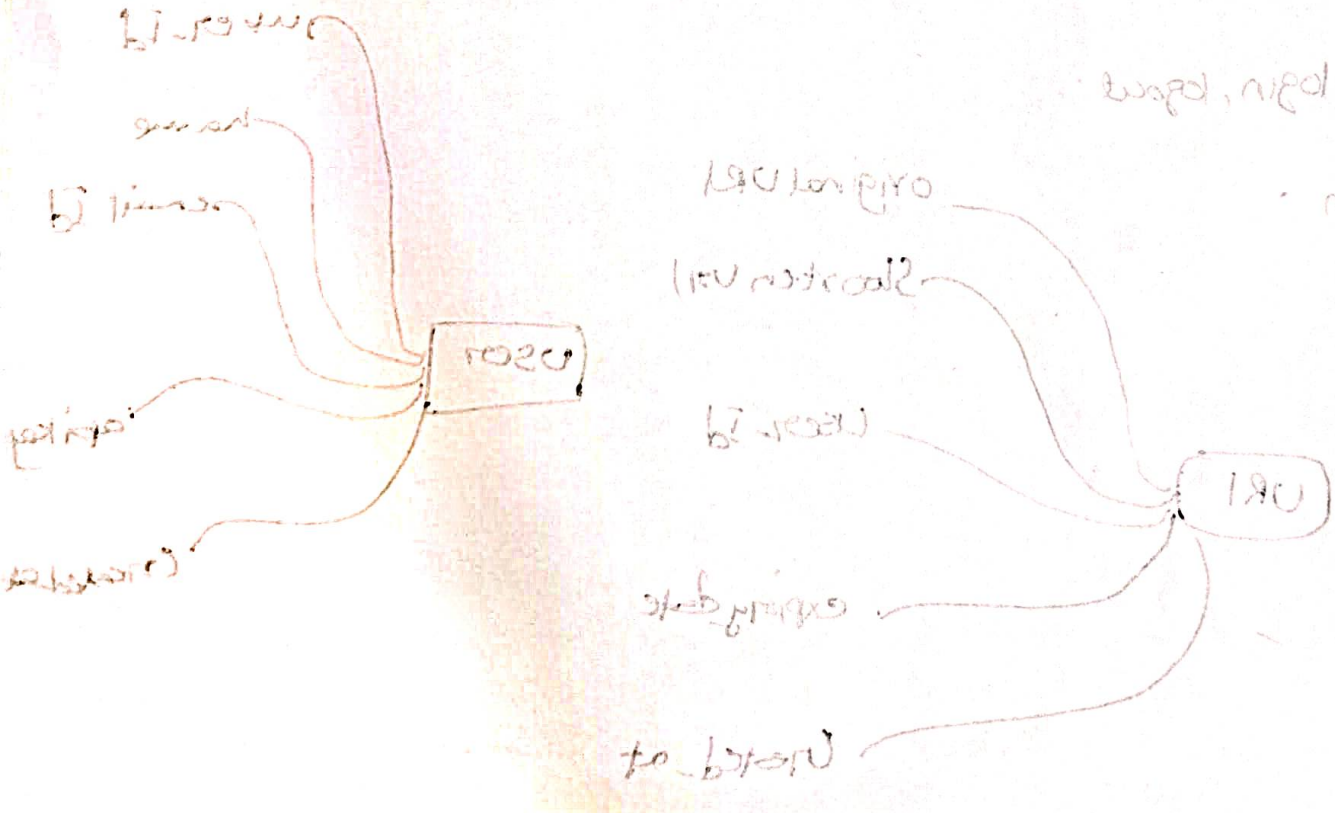


long → Short

avoid collisions

Character used.

a-z
A-Z
0-9
} 62 characters.



URL-Shortener Design