SYSTEM DESIGN:

System Design is the process of designing the architecture, components, and interfaces for a system so that it meets the end-user requirements.

Approaching Design Problems.

Breaking Down the Problem:

Communicating your Ideas

Assumptions that make sense

System should be :

If your system can serve all those features without wearing out then your System can be considered to be **Reliable.**

A **Fault Tolerant** system can be one that can continue to be functioning reliably even in the presence of faults

Availability:(depending on the services)-No single point of failure

Scalability refers to the ability of the System to cope up with the increasing load.

Client Server Model:

DNS request: Request for the ip address for the url..

Client(Browser) ----------Dns request------> dns server
<-------Respond ip address----

client(Browser)------- Http/https request(encrypted)----> Web Server
< -----------------Http response with data----

Need to specify a port while communicating with the server.

centralized system but prone to attack of data (modification , spoofing)

Network Protocol:

Please Do Not Touch Sachin's Patni Anjali. :)
Physical data link, network, tcp, session, presentation, APplication

Protocol: rules for interaction between two parties.

IP:
IP packet have two main section : Header(version, header length, type of service, total

length identification, flags, fragment offset, ttl, protocol header checksum,Source ip, destination ip,) data, option.
Have limited data length so multiple packets are send

TCP :build on top ip protocol
Send ip packet in ordered way(in reliable way)
error control.
Add tcp header img
have ip header , tcp header, data
use 3 way handshaking for communication

HTTP:
build on top of tcp, higher level of abstraction, request response paradigm,

Availability

Mesure in Nines
99% : 2 nines availability (3.65 days per yr downtime)
99.9% : 3 nines availability(8.77 hours per yr downtime)
5nines : 5.26 minutes : standaard good system
SLA: service Level Agreement : Agreement between service provider and customer(end user)
SLO Service Level Objective; Component of SLA : percentage of error, service time etc…
For ensuring availability .

There should not be single point of failure
Including redundancy by adding machines, server.
Passive Redundancy: application of redundant elements only when an active element fails
Active Redundancy : is the application of redundant elements at all times.
 it distributes loads across all elements such that the load on the each element and the overall system is reduced.

Caching:

Caching is a process that stores multiple copies of data or files in a temporary storage location(cache)so they can be accessed faster.

Latency and throughput

Latency: How long it takes for the data to traverse from one point to another. eg time taken to traverse one request from client to server and response back to client .

like(Roughly)
1 MB SSD take 1000us
1MB HDD take 2000us

Throughput: work done by machine in unit of time
eg how many requests can be served in a unit sec.

Pay increase throughput (Cloud provider)
Latency and throughput are not necessary correlated(cannot make assumption of one by other)

Load Balancer:

when we have large request from client to server then to handle those request we do scaling (Horizontal and vertical scaling)
Horizontal Scaling - add more machines, server
Vertical Scaling- adding more resource to existing server
Load Balancer - Balancer is a server which balance workload(traffic rerouting) to the different server.
Help in preventing overloading of particular server or route

client server
client -------> Load Balancer ------> server
client server
client

Load Balancer resides between
server and db
dns queries
client and server
servers and the cache servers

We have software as well hardware load balancer

Actual Working of Load Balancer

Use **Round Robin algo** to select particular server(to server request)

Weighted round robin strategy server (weighting server according to power of server) and sending request accordingly

Performance based strategy : Balanced load and redirect traffic based on performance and response time of the server.

Ip Based : Hashed ip address of request and balance accordingly.
Used when we have caching.

Path Based : Redirecting according to request path same type request are routed to a same server.

We can have multiple load balancer(which communicate with each other)

STORAGE:

Database is just a server
When database server goes down then data stored in memory is lost but disk stored it permanently.

Proxies
Server sits in bw set of clients and a set of servers.

Forward Proxies : Acts on behalf of the client side .

Client-<----Forward Proxy------->Server(get request from forward proxy)
client asks forward proxy to communicate on its behalf.

To hide the identity of the client.
generally used to pass requests from an isolated, private network to the Internet through a firewall.

Eg VPN

Reverse Proxy: Act on behalf of server

client-<-----Reverse proxy--------> Server
client Request to server but that request goes to reverse proxy but client does not know about this(He think it is interacting with server

use: To filter out request, caching at reverse proxy, use reverse proxy as load balancer

Hashing
Action to perform to transform arbitrary piece of data into fixed size of data. eg hashing ip address username

URL Shortening:

URL Shortenings per month

million

Total years

URL object size

Bytes

Total Files

billion

Total Storage

TB

Types of URLs	Time estimates
New URLs	200/s
URL redirections	20K/s
Incoming data	100KB/s
Outgoing data	10 MB/s
Storage for 5 years	15 TB

Memory for cache	170 GB

Key Take Away:
1. 100:1 read write request
2. 80:20 cached url(frequently visited).

```
createURL(api_dev_key, original_url, custom_alias=None,
user_name=None, expire_date=None)
```

api_dev_key (string): The API developer key of a registered account.

```
deleteURL(api_dev_key, url_key)
```

3. Each api_dev_key(user) can be limited to a certain number of URL creations and redirections per some time period.

4. we don't need to use relationships between objects – a NoSQL store like DynamoDB, Cassandra or Riak is a better choice. A NoSQL choice would also be easier to scale

5. **Key Generation Service (KGS)** that generates random six-letter strings beforehand and stores them in a database (let's call it key-DB).