|  |  |
| --- | --- |
| **[1. Introduction System Design](#Introduction_System_Design)**  [**2. Components**](#Components)  [**3. Client Server Architecture**](#Client_Server_Architecture)  [**4. Proxies**](#Proxies)  [**5. Data & Data Flow**](#Data_Data_Flow)  [**6. Databases types: SQL, NoSQL, Column, Search, Key Value**](#Database_type)  [**7. Application & Services**](#Applications_Services)  [**8. API**](#API)  [**9. Caching**](#Caching)  [**10. REST API**](#Rest_API)  [**11. Message Queues**](#Message_Queues)  [**12. Pub Sub Messaging**](#Pub_Sub_Messaging)  [**13 Performance Metrics**](#Performance_Metrics)  [**14 Performance Metrics of components in a software**](#Performance_Metrics_components)  [**15 Fault & Failure in distributed systems**](#Fault_Failure_in_distributed_systems)  [**16 Horizontal vs vertical scaling**](#Horizontal_vertical_scaling)  [**17 Database Replication**](#Database_Replication)  **[18 CAP Theorem](#CAP_Theorem)**  [**19 What is Database Sharding, Logical and Physical Shards, Dynamic vs Algorithmic Sharding**](#Database_Sharding)  [**20 Hashing, Consistent Hashing**](#Hashing) | **https://www.youtube.com/channel/UCTCfIFumvw06VZmMSce\_pSw/videos**  [**https://softwarearchitect.ca/**](https://softwarearchitect.ca/)  [**https://www.fullstack.cafe/interview-questions/api-design**](https://www.fullstack.cafe/interview-questions/api-design)  [**https://github.com/growthbook/growthbook**](https://github.com/growthbook/growthbook)  [**https://www.youtube.com/c/sudoCODE/videos**](https://www.youtube.com/c/sudoCODE/videos)  [**https://www.youtube.com/watch?v=07BVxmVFDGY**](https://www.youtube.com/watch?v=07BVxmVFDGY)  **API Headers**  **5 mistakes I made in System Design Interviews that you should avoid**  **Foundation of System Design Interview starting with Functional vs Non Functional Requirements**  **How to solve capacity estimation problems faster? | Thumb rules and quick tips**  **System Design Books**  **High Level (HLD) Vs Low Level(LL) Design**  **CDN** |

**1.** **Introduction System Design**

**what is the system?**

Graphical user interface, application

Description automatically generated

a system is a loosely used term for an architecture or collection of software or technology that communicate with each other interact with each other in order to serve a certain set of users with a certain set of requirements

For example

* image sharing system (Instagram), texting (whatsapp), streaming systems (Netflix, hotstar)
* in real life also there are systems like buildings, hotels, hospitals, theaters.

Graphical user interface, application

Description automatically generated

* for example in buildings all the buildings have walls, floors, ceilings, electrical supply, water supply etc but different buildings serve different users with different set of requirements
* similarly in software systems even though basic building components are same but they serve different kind of users with different set of requirements.

**what is design**

Design is a process of understanding the user requirements and selecting the components modules and software technologies, how they are going to be communicating with each other to actually serve the need of the system.

Whole process of selecting those components, understanding the needs of the user and the requirements and also factoring in different constraints and concerns.

Even though the components of systems can be same the design for two different systems looks very different

let's say design of a duplex is going to look very different from design of a skyscraper

similarly design of a website that serves a static content or just like one or two videos is going to be very different from a streaming platform like netflix

**why system design**

In order to understand and develop this skill of designing, certain kind of systems which serve to larger scale and larger users the process of system design comes into the picture

why system design is needed as a skill or why system design is such a popular topic

The whole process is actually very complicated it needs a lot of experience expertise and knowledge about software technologies etc in order to build successful large scale systems in real world actually this is not done by one person but as an engineer one should know about the components, the trade-offs, the problems that have to be solved, the concerns of where systems can fail, how to handle those failures, what are the concerns and the constraints that have to be addressed and taken care

**2.** **Components**

A whiteboard with writing on it

Description automatically generated with medium confidence

Components which are the basic building blocks of system

Components could be divided into two parts 1) logical entities and 2) tangible entities

|  |  |  |  |
| --- | --- | --- | --- |
| Diagram  Description automatically generated | **Logical entities** are responsible for making up for the access system. All the systems are built up of data. Dbs are the core of any system, which allows you to store the data. so that it can be available to the users in a later run.  How do users interact( read, write) with databases? Communication through Application or Services (which is mobile, desktop or website) Layer. | Graphical user interface, application  Description automatically generated | Applications is running on a machine and dbs is running on another machine. Now these two machines have to interact with each other and that happens through communication protocols  **Communication protocols** is another logical entity which makes the communication between different machines, so that the components of the system can interact with each other  The way applications communicate over network via communication protocols like http, tcp etc  Similarly application or services communicate on a software level with each other through components like request in the form of apis, rpcs. |

all the systems may or may not have a **presentation layer**.

Systems do have a presentation layer - the mobile, the desktop, the web apps

Systems do not have a presentation layer - if you are making a logger or a logging system which just collects the logs of your application (it may or may not need a presentation layer)

The way system is presented to the user so that the users can interact with it comes under one more logical entity of a system

All these components run on some computers or instances which are available by some cloud providers like azure, aws, google.

|  |  |
| --- | --- |
| **Tangible entities** or technologies that make up for these components  **F**or databases like mongodb, mysql, Cassandra, redis etc.  **F**or application and services or the code and their communication like apis, rpcs, etc  For presentation layer (front-end app), which are built using frameworks like amber, react, etc  For android /ios apps have their own native code bases in order to develop those applications  For the security of the systems, have certain security mechanisms and protocols to secure the data and avoid the attacks.  All these technologies are actually deployed or put on real hardware instances or computers which are provided by the cloud providers | Graphical user interface  Description automatically generated with medium confidence |

Now let's merge all , see a very superficial view of what our system looks like

There is a **presentation layer** where the system could be presented, where the user can interact with the system through a desktop, website or a mobile app which interacts in the backend with our system with the applications.

**Applications** interact with the databases for the exchange of data

All the machines (applications and databases) are actually deployed on instances physical **computers** which interact with each other over these blue lines which is **network**

these application code interact with each other through **apis**, messages

All these are housed inside a cloud provider like aws or gcp etc which is called **infrastructure**

Let's go back to our example of building, has walls floors, terraces, fire exit, electrical supply etc

similarly a system is composed of applications, databases, caches, load balancers, client interfaces, network request, security layer and infrastructure

A white board with writing on it

Description automatically generated with low confidence

**3.** **Client Server Architecture**

|  |  |  |  |
| --- | --- | --- | --- |
| **1st component** of system design, which is CSA. | Text, letter  Description automatically generated | **What is CSA?**  Basic 2 tier CSA - Client request for some particular data(image,text) from server. Server responds with the data.  Some logical manipulation (example, filtered house list) has to be done over the data. |  |

**Understanding Thick & Thin Client**

**Thick Client** - It’s not necessary logic sits in server. Logical manipulation or processing sits in client side. For example MS Outlook, video editing sw.

**Thin Client** – Logic or processing sits in server side, means client doesn’t have many responsibilities. Example streaming platform Netfilx, hotstar. Even though presentation layer, but majority processing happens in server/backend side.

**2 Tier, 3 Tier and N Tier Architecture**

**3 Tier** - Processing is huge, it has be done on server side Or Data is heavy , lot of amounts of data that has been to stored in process . Logic and Data layer broken (2 tier). Logic which is application and Data which is database.

**N Tier** – In some cases 3-tier is enough, system design is large or complex applications, where even processing or requirements can’t be full fill in efficient manner using 3 tier layer. Then they are multiple layer in between, for example **caching layer** between logic and data OR **Load balancer or proxy** in between client and logic

A white board with writing on it

Description automatically generated with low confidence

How do you decide if you have to select thin, thick, 2tier or 3 tier

Graphical user interface, application

Description automatically generated

**4.** **Proxies**

|  |  |
| --- | --- |
| Text, letter  Description automatically generated | **What is Proxy?** On behalf, for example when you can’t attend the class , you ask your friend attend the class behalf of you. Here friend is proxy.  **2 Types of** Proxy – Forward and Reverse |

A picture containing text, whiteboard

Description automatically generated

**Forward Proxy** – A Machine sits between client and server towards the client side, which talks to the server on behalf the client. Client sends request to the proxy, proxy sends to the server, server response back to cleint. Here client never talks to the server, the proxy behalf of the client talks to the server.

**Why It’s useful?** 1. For anonymity , where server doesn’t know IP of the client, server only knows IP of proxy.

2. In institution, where there can be multiple clients, all traffic is controls or monitored through forward proxy

A white board with writing on it

Description automatically generated with low confidence

3. Block access to certain sites (malicious)

4.Caching response at forward proxy site.

**Pros/Cons -** if country, organization, work place block access to some sites, in this case proxy can be used to by pass the rules & haves access to sites (useful or not useful)

A picture containing text, whiteboard

Description automatically generated

**Reverse Proxy –** Same proxy sits on server side acts as middle man for all server. Client talks to Proxy, Proxy server talks to all Server and gets response back to server.

In this case there is anonymity of servers, where client doesn’t know about IP of any servers, client only knows about the proxy.

**Why It’s useful?**  It can be useful for traffic controls, load balancing, caching response from server, Denial of services (DDos) attacks because servers are not exposed to the outside the world only proxy server expose to the outside the world, SSL/TLS encryption.

**Pros/Cons** - if proxy fails, it becomes reason for single point of failure or bottleneck.

**Proxy are important elements for** security, privacy, handling traffic.

**5.** **Data & Data Flow**

|  |  |
| --- | --- |
| Text, letter  Description automatically generated | For example of buildings, people are core entity, so people get into/out/stayin the building  Similarly, in computing systems data gets entered/manipulated or computed in the system and then served again back to the user  People can actually create the data and then access it, can actually just access the data and not create it , they can do both (youtube)  So data stays at the core of system design |

Let's understand how data is represented at different layers in different formats

A picture containing text, whiteboard, outdoor, document

Description automatically generated

Timeline

Description automatically generated

**1.Business layer or as a user** you interact with data in the format of text, videos, images, notes, etc

when the data travels to the 2.**application layer,** it is transformed in a different format like json xml etc,

when the data gets stored in **3.data stores or databases,** it gets stored in different data structures, which store the data effectively, ensure that the retrieval and the storage is efficient

Then at the **4.network** layer data gets transferred from one machine to another over the network in the form of packets

Diagram

Description automatically generated

Finally at the hardware level, it's just zeros and ones

A white board with writing

Description automatically generated with medium confidence

Now let's understand what are the **different data stores and the flows** through which this data flows inside the system

**why queues, caches and indexes are listed** Q has data in some format. Cache also has data which is easily retrievable rather than making a query to db, similarly for indexes data is stored in such a manner that it could be searched quickly

**Now how data flows among these data stores**. The data travel from app to db, from db to cache, from cache to app, from app to queue and then to db.

So the methods (**apis, messages, events, etc**) through which the data will flow ,which allow this data in different stores and the app to flow and get exchange between these components

|  |  |  |  |
| --- | --- | --- | --- |
| Graphical user interface, application  Description automatically generated | **Why this is important** because you need to understand when you have to store certain data in dbs, when you need queues/caches, which method you will use, so that the data flow becomes effective | Text, letter  Description automatically generated | **Now let's understand where does data come from or how the data is generated**  **Users** create the data or put input the data inside the system. Users will interact with the system and will create more data.  For example a notes or calendar app, e-commerce website.  **Internal data** when you have to store data about data or the logs of the application, which are running actually to serve the system, which systems populate on their own  **Insights** when users actually interact with the system and upload some kind of data or retrieve some kind of data some insights are generated  For example if the user is buying something from an ecommerce website, invoice will be Generated for that user. similarly if the user is interacting with a streaming system like youtube or netflix etc, there will be a history of what content they have watched, profile history, their payment/subscription details |

|  |  |
| --- | --- |
| Text, letter  Description automatically generated | **Understand the factors**  **Type of data - D**ata could be text, images, videos. A system which deals with videos and computing on those videos will be different from a system that deals with only text type of data.  It’s important because it will lead to decisions like which db going to use for storing videos vs for store some text.  **Volume of data**, a system which supports storage of terabytes of data would be completely different from a system which supports a storage of some gbs of data  **Consumption and retrieval of data,** some systems which are populating(writing) a lot of data and also consuming the same amount of data.  Systems which are writing a lot of data like even streaming systems but the consumption is not that high  Systems which are not writing that much data the system internally is populating a lot of information but the retrieval is high.  There could be both  **Volume** and reads and writes(**Consumption and retrieval)** these factors decide which kind of data store you will choose, which kind of storage strategy you will choose and what will your system look like for handling with such amount of data  **Security** there could be systems which are like transactional systems like banking etc, where the security of data is of utmost important  If it’s okay if you are not able to serve a request, if a user cannot log in a few minutes, but it is not okay that the data for that user gets leaked or compromised  Depending on these factors you will be able to **make informed decision on what kind of technology** you have to use in order to build a system that supports a certain volume and certain requirement for data |

|  |  |
| --- | --- |
| Timeline  Description automatically generated | **Some examples of different systems which have different data and data flow requirements**  For example **an authorization system,** the volume of data might not be that high because it’s store some user and credential details, but the level of security is comparatively higher because make sure that the user credentials are safe whoever is requesting the access, we should not be given any wrong access to the resources  **Streaming systems,** the data volume is high and also the data retrieval speed, the number of requests that are coming to the system in order to retrieve the data (people are watching a lot of videos) has to be high. so here the data volume, the data retrieval is high. so streaming systems have to handle with both of those factors  **Transactional systems**, like banking systems or e-commerce websites, where you're placing in orders or interacting with a certain transaction, in those cases the journey of the transaction should not fail, it should not be done twice and the money in the bank should be intact, all those factors are really important.  for example if you have ordered for a headphones from amazon, if the data is compromised or something wrong with the processing of the data, you might end up getting a wrong order or not get your order  **heavy compute systems,**  so with the advance of ML and AI. for example systems which are dealing with the lot tons and tons of data, which is recorded by cameras & have to process over videos, have to use gpus in order to mine over the data and run sophisticated algorithms. the requirements of databases and the data storage are different because nobody is retrieving the data but a lot of data is being uploaded and being computed on these |