System design

https://www.educative.io/blog/top-10-system-design-interview-questions

Distributed system fundamentals

Data durability and consistency
- differences and impacts of failure rates of storage solutions and corruption rates in read-write processes

Replication
- Backing up data and repeating processes at scale

Consensus
- ensuring all nodes are in agreement, which prevents faulty processes from running and ensures consistency and replication of data and processes

Partitioning
- divide data across different nodes within systems, reducing reliance on pure replication

Distributed transactions
- once consensus is reached, transactions from applications need to be committed across databases, with fault checks by every resource involved

REST
- a set of design principles that interact with HTTP to enable system efficiency and scalability

Caching
- tradeoffs to determine
  - what should be in a cache
  - how to direct traffic to a cache
  - how to ensure you have the right data in a cache

Stream processing

N-tier applications?

Steps to answer a question
0. Ask clarifying questions at every step!!
1. Clarify the goals

2. Determine the scope
    - describe the feature set you will be discussion, define every feature and its importance to the end goal
3. Design for the right scale
    - Can you use a single machine for your problem? Or do you need to scale?
4. Start simple, then iterate
    - Describe the high level process end to end based on your feature set, and discuss bottlenecks
5. Consider relevant data structures and algorithms
6. Describe trade offs

Start each problem by stating what you know: List all required features of the system, common problems you expect to encounter with this sort of system, and the traffic you expect the system to handle. The listing process lets the interviewer see your planning skills and correct any possible misunderstandings before you begin the solution.

Narrate any trade-offs: Every system design choice matters. At each decision point, list at least one positive and negative effect of that choice.

Ask your interviewer to clarify: Most system design questions are purposefully vague. Ask clarifying questions to show the interviewer how you're viewing the question and your knowledge of the system's needs.

Encryption?

Sample questions
1. Design a messaging/chat system
2.

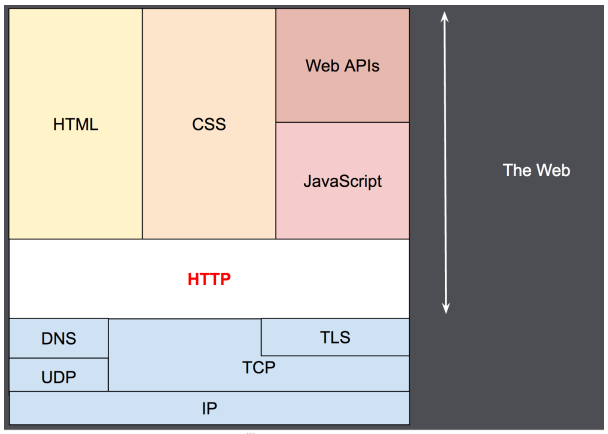
https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

An overview of HTTP

HTTP is a protocol for fetching resources such as HTML documents. It is the foundation of any data exchange on the Web and it is a client-server protocol, which means requests are initiated by the recipient, usually the Web browser. A complete document is reconstructed from the different sub-documents fetched, for instance, text, layout description, images, videos, scripts, and more.

Clients and servers communicate by exchanging individual messages (as opposed to a stream of data). The messages sent by the client, usually a Web browser, are called requests

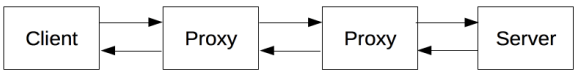
and the messages sent by the server as an answer are called responses.



Designed in the early 1990s, HTTP is an extensible protocol which has evolved over time. It is an application layer protocol that is sent over TCP, or over a TLS-encrypted TCP connection, though any reliable transport protocol could theoretically be used. Due to its extensibility, it is used to not only fetch hypertext documents, but also images and videos or to post content to servers, like with HTML form results. HTTP can also be used to fetch parts of documents to update Web pages on demand.

HTTP is a client-server protocol: requests are sent by one entity, the user-agent (or a proxy on behalf of it). Most of the time the user-agent is a Web browser, but it can be anything, for example, a robot that crawls the Web to populate and maintain a search engine index.

Each individual request is sent to a server, which handles it and provides an answer called the response. Between the client and the server there are numerous entities, collectively called proxies, which perform different operations and act as gateways or caches, for example.



In reality, there are more computers between a browser and the server handling the request: there are routers, modems, and more. Thanks to the layered design of the Web, these are hidden in the network and transport layers. HTTP is on top, at the application layer. Although important for diagnosing network problems, the underlying layers are mostly irrelevant to the

description of HTTP.

Client: the user-agent
The user-agent is any tool that acts on behalf of the user. This role is primarily performed by the Web browser, but it may also be performed by programs used by engineers and Web developers to debug their applications.

The browser is always the entity initiating the request. It is never the server (though some mechanisms have been added over the years to simulate server-initiated messages).

To display a Web page, the browser sends an original request to fetch the HTML document that represents the page. It then parses this file, making additional requests corresponding to execution scripts, layout information (CSS) to display, and sub-resources contained within the page (usually images and videos). The Web browser then combines these resources to present the complete document, the Web page. Scripts executed by the browser can fetch more resources in later phases and the browser updates the Web page accordingly.

A Web page is a hypertext document. This means some parts of the displayed content are links, which can be activated (usually by a click of the mouse) to fetch a new Web page, allowing the user to direct their user-agent and navigate through the Web. The browser translates these directions into HTTP requests, and further interprets the HTTP responses to present the user with a clear response.

The Web server
On the opposite side of the communication channel is the server, which serves the document as requested by the client. A server appears as only a single machine virtually; but it may actually be a collection of servers sharing the load (load balancing), or a complex piece of software interrogating other computers (like cache, a DB server, or e-commerce servers), totally or partially generating the document on demand.

A server is not necessarily a single machine, but several server software instances can be hosted on the same machine. With HTTP/1.1 and the Host header, they may even share the same IP address.

Proxies
Between the Web browser and the server, numerous computers and machines relay the HTTP messages. Due to the layered structure of the Web stack, most of these operate at the transport, network or physical levels, becoming transparent at the HTTP layer and potentially having a significant impact on performance. Those operating at the application layers are generally called proxies. These can be transparent, forwarding on the requests they receive without altering them in any way, or non-transparent, in which case they will change the

request in some way before passing it along to the server. Proxies may perform numerous functions:

caching (the cache can be public or private, like the browser cache)
filtering (like an antivirus scan or parental controls)
load balancing (to allow multiple servers to serve different requests)
authentication (to control access to different resources)
logging (allowing the storage of historical information)

HTTP flow
When a client wants to communicate with a server, either the final server or an intermediate proxy, it performs the following steps:

Open a TCP connection: The TCP connection is used to send a request, or several, and receive an answer. The client may open a new connection, reuse an existing connection, or open several TCP connections to the servers.

Send an HTTP message: HTTP messages (before HTTP/2) are human-readable. With HTTP/2, these simple messages are encapsulated in frames, making them impossible to read directly, but the principle remains the same. For example:
GET / HTTP/1.1
Host: developer.mozilla.org
Accept-Language: fr

Read the response sent by the server, such as:
HTTP/1.1 200 OK
Date: Sat, 09 Oct 2010 14:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html

<!DOCTYPE html... (here come the 29769 bytes of the requested web page)

Close or reuse the connection for further requests.

If HTTP pipelining is activated, several requests can be sent without waiting for the first response to be fully received. HTTP pipelining has proven difficult to implement in existing

networks, where old pieces of software coexist with modern versions. HTTP pipelining has been superseded in HTTP/2 with more robust multiplexing requests within a frame.

https://developer.mozilla.org/en-US/docs/Web/HTTP/Proxy_servers_and_tunneling

When navigating through different networks of the Internet, proxy servers and HTTP tunnels are facilitating access to content on the World Wide Web. A proxy can be on the user's local computer, or anywhere between the user's computer and a destination server on the Internet. This page outlines some basics about proxies and introduces a few configuration options.

There are two types of proxies: forward proxies (or tunnel, or gateway) and reverse proxies (used to control and protect access to a server for load-balancing, authentication, decryption or caching).

Forward proxies
A forward proxy, or gateway, or just "proxy" provides proxy services to a client or a group of clients. There are likely hundreds of thousands of open forward proxies on the Internet. They store and forward Internet services (like the DNS, or web pages) to reduce and control the bandwidth used by the group.

Forward proxies can also be anonymous proxies and allow users to hide their IP address while browsing the Web or using other Internet services. TOR (The Onion Router), routes internet traffic through multiple proxies for anonymity.

Reverse proxies
As the name implies, a reverse proxy does the opposite of what a forward proxy does: A forward proxy acts on behalf of clients (or requesting hosts). Forward proxies can hide the identities of clients whereas reverse proxies can hide the identities of servers. Reverse proxies have several use cases, a few are:

Load balancing: distribute the load to several web servers,
Cache static content: offload the web servers by caching static content like pictures,
Compression: compress and optimize content to speed up load time.