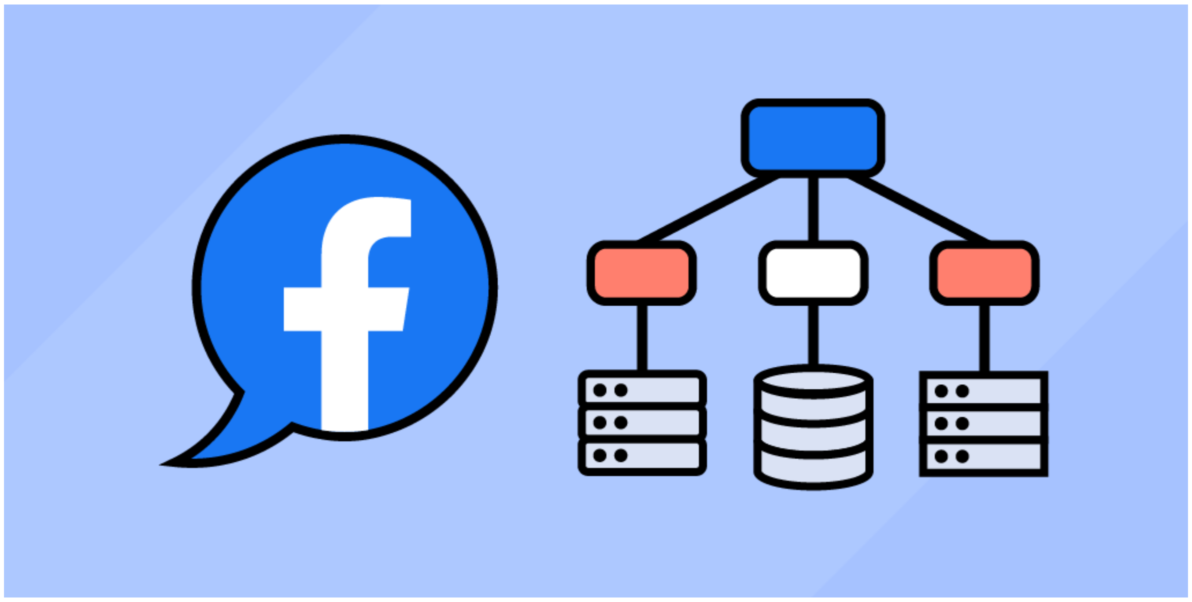
grokkingtechinterview.com

Top 10 Facebook system design interview questions - Grokking the Tech Interview

The Educative Team

13-17 minutes



The system design interview is a necessary hurdle as you advance in your career as a software engineer. System design interviews assess your design skills by asking you to build a scalable system. Even seasoned engineers can struggle with this interview round, but you're already ahead of the game if you're here for interview preparation.

Today we'll discuss the top system design interview questions asked at Facebook. These questions are also common fare for system design interviews at all top tech companies, including FAANG (Facebook, Apple, Amazon, Netflix, and Google).

We'll cover:

- Facebook's interview process
- How to answer system design interview questions
- Top 10 Facebook system design interview questions
- Wrapping up and next steps

Facebook's interview process

Facebook's interview rounds begin remotely and progress to onsite interviews. The remote interviews consist of a phone screen with a recruiter and a technical phone screen with a software engineer. The onsite interviews are 45 minutes each and consist of the coding interview, system design interview, and behavioral interview. Facebook might allow you to choose between a system design or product design interview. While similar, product design may be relevant for roles handling a user-facing product, and system design for roles handling large-scale distributed systems.

> *Facebook's system design interview is also referred to as the Pirate Interview Round.*

How to answer system design interview questions

Your interviewers will give you an open-ended question like, "How would you design Messenger?" You'll spend most of your time asking questions, presenting your thought process, and diagramming at the whiteboard. There is no right or wrong answer to these questions. Your goal is to show your design skills and understanding of the many components that constitute a scalable system.

We recommend approaching your answer like this:

- **State what you know**: State what's known about the design problem you've been given.
- **Clarify and ask questions**: Clarify the requirements of your system.
- **Identify and narrate tradeoffs at decision points**: Narrate tradeoffs between system design components

at all decision points. For example, reference CAP Theorem when you have tradeoffs between consistency and availability.

- **Discuss recent developments in technology**: Show that you're a next-gen engineer by discussing how you'd implement technologies such as machine learning into your system's design.

Top 10 Facebook system design interview questions

1. Design Facebook Newsfeed

You may be asked to design Facebook Newsfeed, which displays a scrollable feed of status updates. These updates can contain photos, videos, or text. Users can post status updates or engage with status updates from accounts and pages they follow.

> *Similar services: Twitter Newsfeed, Instagram Newsfeed, Quora Newsfeed*
>
> *Difficulty: Hard*

Functional requirements include:

- Users should see posts from people, pages, and groups that they follow on the Newsfeed.

- The service should append new posts for all active users as they arrive to the feed.
- The service should support live commenting.

Non-functional requirements include:

- The system should be able to generate any user's newsfeed in real-time.
- Minimum latency is desirable so users don't experience a lag.

At a high level, this problem can be split into two parts: feed generation and feed publishing. For feed generation, you'll want to determine **how many feed items to store in memory** for a user's feed as well as **whether the newsfeed should be generated or kept in memory** for all users. For feed publishing, you'll want to consider whether to use a "Push" or "Pull" model to publish feed data to users.

2. Design Messenger

You may be asked to design Messenger, an instant messaging service through which users send messages to each other. Messages can contain text or other media attachments.

Similar services: WhatsApp

Difficulty: Medium

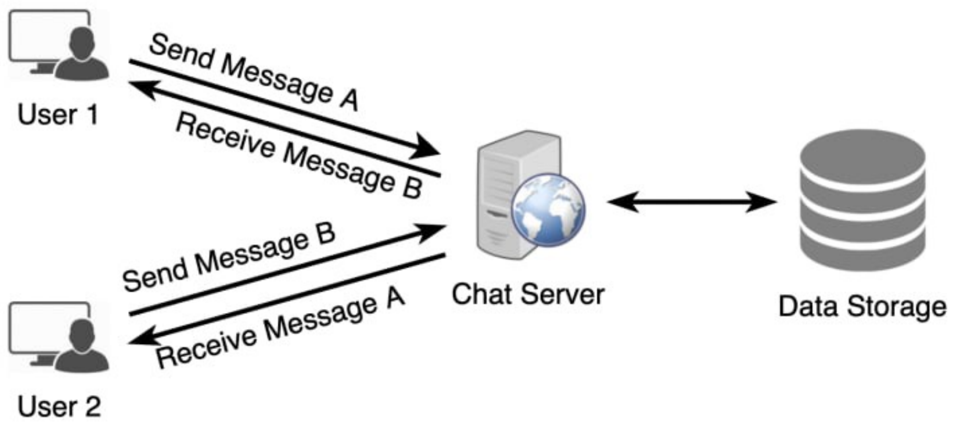
Functional requirements include:

- The service should support one-on-one conversations as well as group chats between users.
- The service should support persistent storage of chat history.

Non-functional requirements include:

- Minimum latency is needed to support real-time chat.
- High consistency is needed so that users' devices display the same chat history.
- Availability should be high, but can be compromised as a tradeoff for high consistency.

At a high level, you'll need a chat server to manage communication between users. Each message a user sends will go to the server before going to the recipient. The server will also store the message in its database.



Some questions you'll need to address for this problem are:

- How will clients maintain an open connection with the server?
- How can the server track all opened connections to efficiently redirect messages to users?
- How will the server handle messages sent to offline users?

3. Design Instagram

You may be asked to design Instagram or a similar photo-sharing service. Instagram allows users to upload photos and share them with other users.

> *Similar services: Flickr, Picasa*
>
> *Difficulty: Medium*

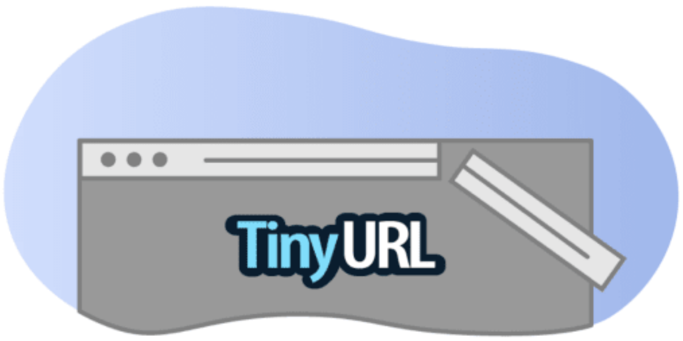
Functional requirements include:

- Users can upload, download, and view photos.
- Users can follow other users. Non-functional requirements include:
- High availability can take priority over consistency.
- High reliability is needed so that no uploaded media is lost.

Redundancy and reliability are crucial to this system. You must ensure that no uploaded files are lost. You need to create redundancy to ensure that the system still runs in the event that one server fails. You'll need to store multiple copies of each file across several storage servers. You'll also have several replicas of services running in the system to protect your system from having a single point of failure.

4. Design TinyURL

You may be asked to design TinyURL or a URL shortening service. These services create shorter aliases for long URLs. The short aliases are known as "short links" and redirect users to the original URL.



> *Similar services: bitly*
>
> *Difficulty: Easy*

Functional requirements include:

- The service should generate a unique short link for any URL it is given.
- The service should redirect users from short link to original link.
- Users can customize their short link.

Non-functional requirements include:

- The system should be highly available so redirections don't fail due to service interruption.

- Minimal latency is needed for real-time URL redirection.

You could use two solutions to ensure a short and unique key for a given URL: encoding actual URLs or generating keys offline. To scale your database, you'll need to partition it to store information about billions of URLs. You could use either range-based partitioning or hash-based partitioning. You can also cache frequently accessed URLs so that application servers don't always need to access backend storage.

5. Design Uber

You may be asked to design Uber or another rideshare service. Uber allows customers seeking rides to connect with available drivers in their proximity. The application has two types of users: drivers and customers.

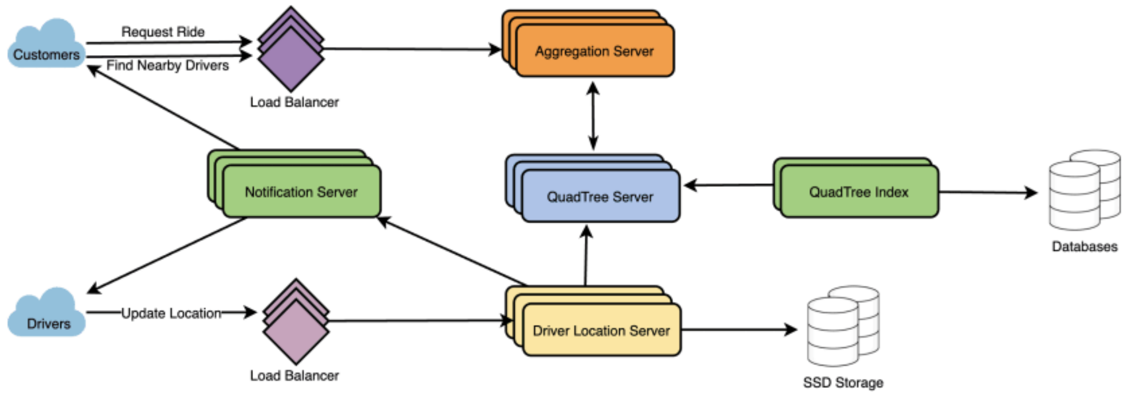
> *Similar services: Lyft, Didi, Via, Sidecar*
>
> *Difficulty: Hard*

The requirements for this system include:

- Drivers should be able to regularly notify service of their current location and availability.

- Customers should be able to see nearby available drivers.
- Customers should be able to request rides and drivers should be notified.

This system's design can include load balancers, notification servers, aggregation servers, driver location servers, QuadTree servers, and a QuadTree index. You use server replicas in case the driver location or notification servers die. You can also store data in persistent storage like solid state drives (SSDs) to provide fast input and output. You can recover data from this storage in the event that both primary and secondary servers die.



6. Design Twitter

You may be asked to design Twitter or a similar social network. Twitter allows users to publish short-form posts called "tweets." Twitter users can post tweets, favorite and retweet tweets, and follow other users.

Difficulty: Medium

Functional requirements include:

- Users can post tweets containing text, photos, or videos.
- Users can follow other users.
- Users can favorite and retweet tweets.

Non-functional requirements include:

- The system should be highly available.
- The system's latency should be no greater than 200ms for timeline generation.
- Consistency can take a hit as a tradeoff for availability.

This will be a read-heavy system. At a high level, you'll need multiple application servers with load balancers to serve these requests. You'll need an efficient database that can store new tweets and support a large number of reads. You'll also need file storage to store photos and videos.

7. Design Ticketmaster

You may be asked to design Ticketmaster or another online ticketing system. Ticketmaster is a booking system that allows customers to purchase theater seats for a specific movie screening. Customers can search for movie screenings in any location.

Difficulty: Hard

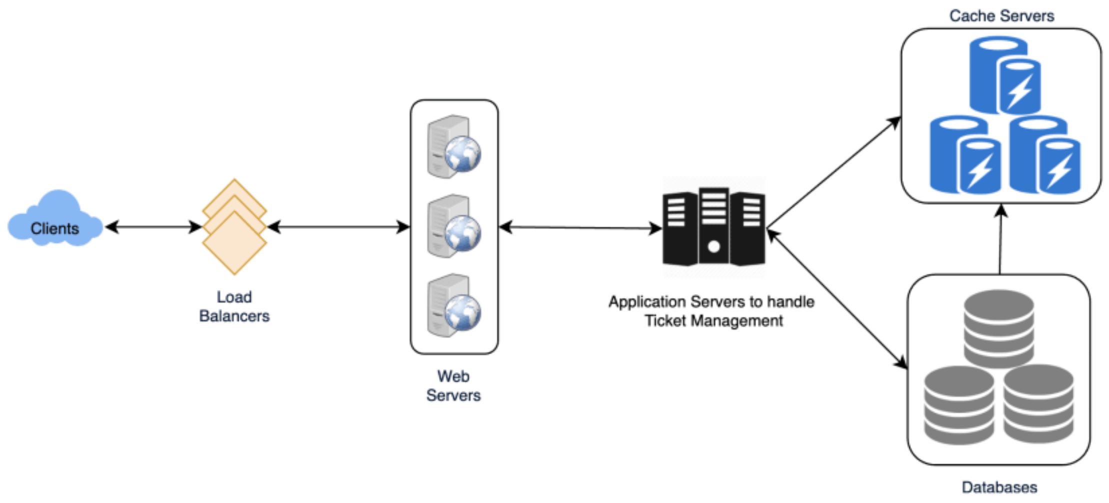
Functional requirements include:

- The service should list all locations of affiliate cinemas.
- Users should be able to book tickets for specific shows at cinemas.
- Users should be able to put a hold on the seats for five minutes before making a payment.

Non-functional requirements include:

- The system should be highly concurrent to handle multiple booking requests for the same seat at any point in time.
- Since the service manages payments, it should be secure and the database should be ACID compliant.

At a high-level, you can use web servers with load balancers to handle user sessions. Application servers can handle ticket management, storing data in the databases, and working with cache servers to process reservations.



8. Design an API rate limiter

An API rate limiter controls how many requests a sender can issue to an API in a specified time period. The limiter blocks requests once the defined cap is reached. Throttling is the process of controlling the usage of the APIs by users during a given period. Rate limiters protect against attacks that are typically generated by bots.

Difficulty: Medium

Functional requirements include:

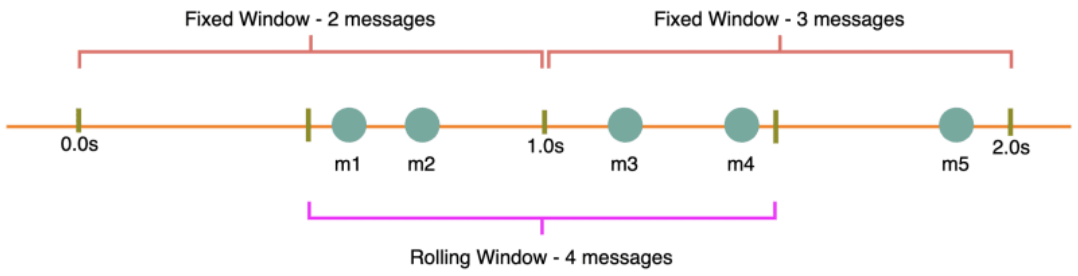
- The service should limit the number of requests an entity can send to an API in a given time window.

- Since APIs are accessible through a cluster, the rate limit should be considered across different servers.

Non-functional requirements include:

- High availability is needed to protect the service from external attacks.
- Low latency is needed so that the rate limiter doesn't negatively impact user experience.

Rate limiting can be done with either a fixed window algorithm or rolling window algorithm. The fixed window algorithm defines the time window from the start of the time-unit to the end of the time-unit. The rolling window algorithm defines the time window as the fraction of the time at which the request was made plus the time window length.


Fixed Window - 2 messages
Fixed Window - 3 messages
0.0s
m1
m2
1.0s
m3
m4
m5
2.0s
Rolling Window - 4 messages

9. Design a web crawler

A web crawler is a software program that visits sites across the web and reads them to gather information. Web crawlers are often used by search engines. Search engines download webpages to create an updated index of web content for faster searches.

Difficulty: Hard

A web crawling service needs to be designed with **scalability and extensibility** in mind. It’ll need to crawl the entire Web and fetch hundreds of millions of Web documents. Web crawling is difficult because of the sheer volume of pages that change frequently on the World Wide Web. The crawler can only download a fraction of web pages at a time, so it must be intelligent enough to prioritize download. You might also opt for a modular design with the expectation of adding new functionalities later. After all, with technology changing every day, there could be newer document types in the future that your crawler must download and process.

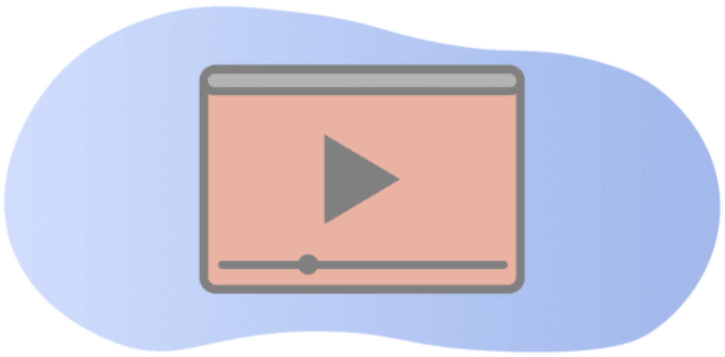
At minimum, a web crawler needs these components:

- **A URL frontier** to store the URLs to download and determine which URLs to crawl first
- **An HTML fetcher** to retrieve a web page from the server
- **An extractor** to extract links from HTML documents

- **A duplicate eliminator** to ensure the same content is not extracted twice
- **A datastore** to store retrieved pages, URLs, and other metadata

10. Design YouTube

You may be asked to design YouTube or another video streaming service. YouTube allows users to upload, view, share, rate, report, and comment on other users' videos. Users can also subscribe to other users' accounts.



> *Similar services: Netflix, Vimeo*
>
> *Difficulty: Medium*

Functional requirements include:

- Users can upload and view videos.
- Users can perform a search based on video titles.

- The system monitors and displays each video's statistics, such as likes and views.

Non-functional requirements include:

- A highly reliable system is needed so that uploaded videos aren't lost.
- The system should prioritize availability over consistency.
- The system should have minimum latency so that users don't experience delays.

You'll need to address important questions in this system such as where videos and thumbnails will be stored. You'll also need to consider how to manage traffic effectively. This system will have a very high read load. Since you'll have a large number of new videos each day, the data should be distributed across several machines to efficiently perform read/write operations.

Wrapping up and next steps

System design interviews have long been common practice for software engineering interviews, and they aren't going away anytime soon!

To help you level up in your career, we created the

Grokking Modern System Design for Software Engineers & Managers course. You'll learn about system design principles and examine complexities and solutions of several common system design questions, including the ones in this article.

Happy learning!

Continue learning about system design on Educative

- Top 5 hardest coding questions from recent FAANG interviews
- System design fundamentals: What is the CAP theorem?
- System design interview guide: Tips from an industry expert
- Grokking Modern System Design for Software Engineers & Managers

Start a discussion

How are you preparing for the Facebook system design interview? Was this article helpful? Let us know in the comments below!