Complete Guide to System Design Interviews (and Top Questions)

Ready for a high-level look at the structure of system design interviews, the core concepts you'll be assessed on, and how to answer the most common questions?

This guide accompanies Exponent's complete system design interview course, trusted by 19,000+ software engineers, engineering managers, technical program managers, and other tech professionals to ace their interviews.

By the end, you'll understand which parts of system design interviews you feel confident in and which areas need improvement.

Who Wrote This Guide?

We wrote this system design interview guide with feedback from 50+ EM, SWE, and TPM technical interview coaches at startups and FAANG+ companies like Microsoft, Amazon, Meta, Google, Netflix, Dropbox, and Stripe.

It was reviewed and edited by these senior engineers and managers:

- Tom Lu (Google)
- Neeraj Gupta (eBay)
- Praveen Dubey (Microsoft)

It also includes contributions by Anthony Pellegrino.

Neamah (Meta PM, former ML engineer) gives a high-level overview of system design interviews and how to answer common questions.

What is the System Design Interview?

The system design interview evaluates your ability to solve a complex problem by designing a system or architecture in a semi-real-world setting.

Your solution doesn't need to be perfect.

Instead, you'll be evaluated on your ability to:

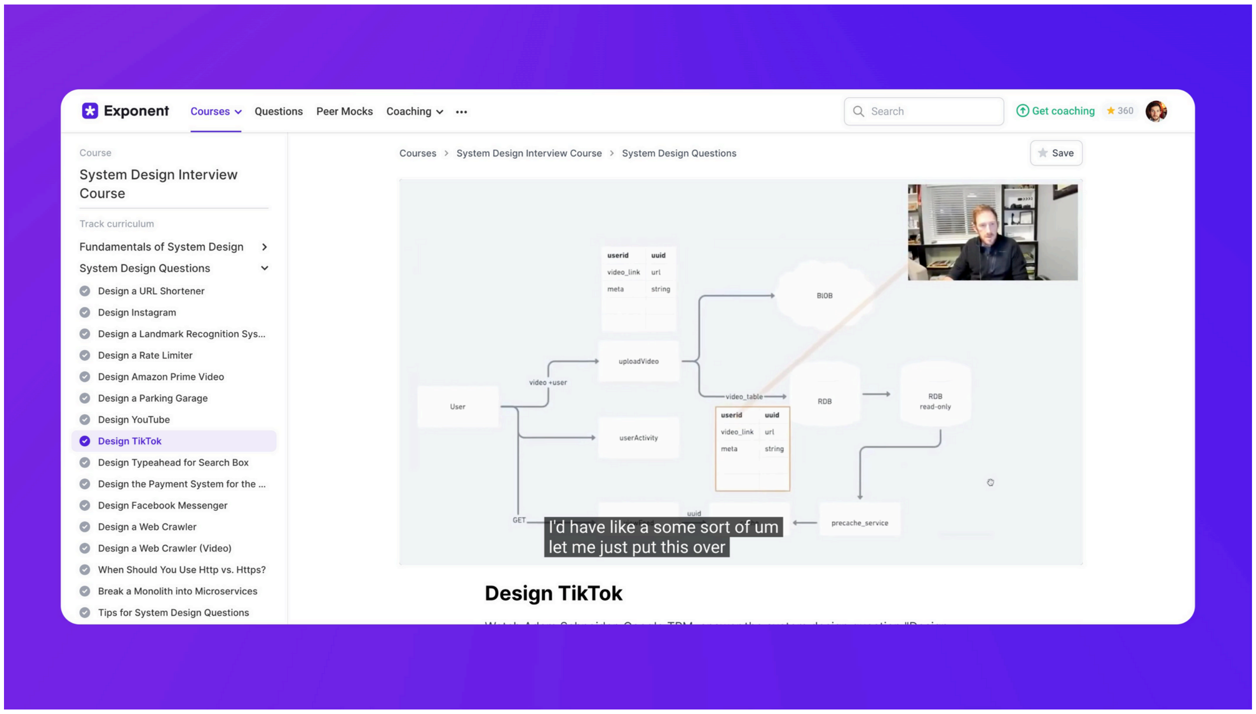
> analyze a problem,
> design a blueprint,
> discuss multiple requirements,
> and weigh the pros and cons to develop a workable solution.

This interview is often used to determine the level at which a candidate will be hired. It is usually accompanied by an engineering manager or behavioral interview.

A hiring manager will assess your ability to make decisions in the face of uncertainty, your confidence in taking risks, and your capacity to adapt to new information.

31 Top System Design Interview Questions

These are the most common questions you'll encounter during your technical interviews. They're from our database of the most frequently asked system design interview questions for EMs, TPMs, and SWEs.


Exponent
Courses
Questions
Peer Mocks
Coaching
Search
Get coaching
360
Course
System Design Interview Course
Track curriculum
Fundamentals of System Design
System Design Questions
Design a URL Shortener
Design Instagram
Design a Landmark Recognition Sys...
Design a Rate Limiter
Design Amazon Prime Video
Design a Parking Garage
Design YouTube
Design TikTok
Design Typeahead for Search Box
Design the Payment System for the ...
Design Facebook Messenger
Design a Web Crawler
Design a Web Crawler (Video)
When Should You Use Http vs. Https?
Break a Monolith into Microservices
Tips for System Design Questions
Courses > System Design Interview Course > System Design Questions
Save
userid uuid
video_link url
meta string
BLOB
uploadVideo
video +user
User
video_table
RDB
RDB read-only
userActivity
GET
uuid
precache_service
Design TikTok

I'd have like a some sort of um
let me just put this over

Watch detailed video breakdowns of system design questions with engineering managers and software engineers in our interview prep course.

Social Media

Social media platforms allow users to post photos and videos, follow and unfollow each other, like and comment on posts, search for content, and get personalized news feeds.

Design TikTok
Design Instagram
Design YouTube
Design Reddit
Design Twitter

Chat Applications

Chat applications enable users to send instant messages to each other.

Some popular applications like WeChat and WhatsApp have billions of users worldwide on hundreds of devices.

Chat apps require support for one-on-one communication, group threads, and sending text, images, videos, and files.

Design WhatsApp
Design WeChat
Design Telegram
Design Facebook Messenger

Media Storage and Streaming

Streaming platforms let users stream content on demand, get personalized recommendations, create multiple user profiles, and search an extensive content library.

Similarly, cloud storage applications efficiently handle storing and retrieving large amounts of data.

Design Amazon Prime Video
Design Netflix
Design Spotify
Design Dropbox

Design Google Drive

Collaborative Documents

Real-time collaboration with virtual tools has become commonplace as working remotely gains popularity.

To build these systems, consider real-time collaboration, version control, access control, and notifications.

Design Figma
Design Google Docs

Search and Location

Generating recommendations based on a user's current location can be essential for finding restaurants, hotels, and points of interest while on the go.

Location-based systems enable users to search for locations, obtain directions, estimate distances and travel times, and receive contextual search results.

Design Google Maps
Design Uber Eats
Design Airbnb
Design Yelp

Artificial Intelligence and Machine Learning

Machine learning systems rely on ML models to process data and function correctly.

You must know how to send data to a model, receive and process a response, and continuously improve the model based on feedback about the quality of its output.

Design a visual landmark recognition system.
Design YouTube's recommendation engine.
Design an AI chatbot.

Payments

Payment systems must keep track of inventory, handle transactions, issue receipts, and prevent orders if a product is out of stock or unavailable.

Design a payment system for a parking garage.
Design a vending machine.
Design a payment system for Kindle.

Video Games

When matching players in online multiplayer video games, consider factors such as latency, player skill levels, and match settings.

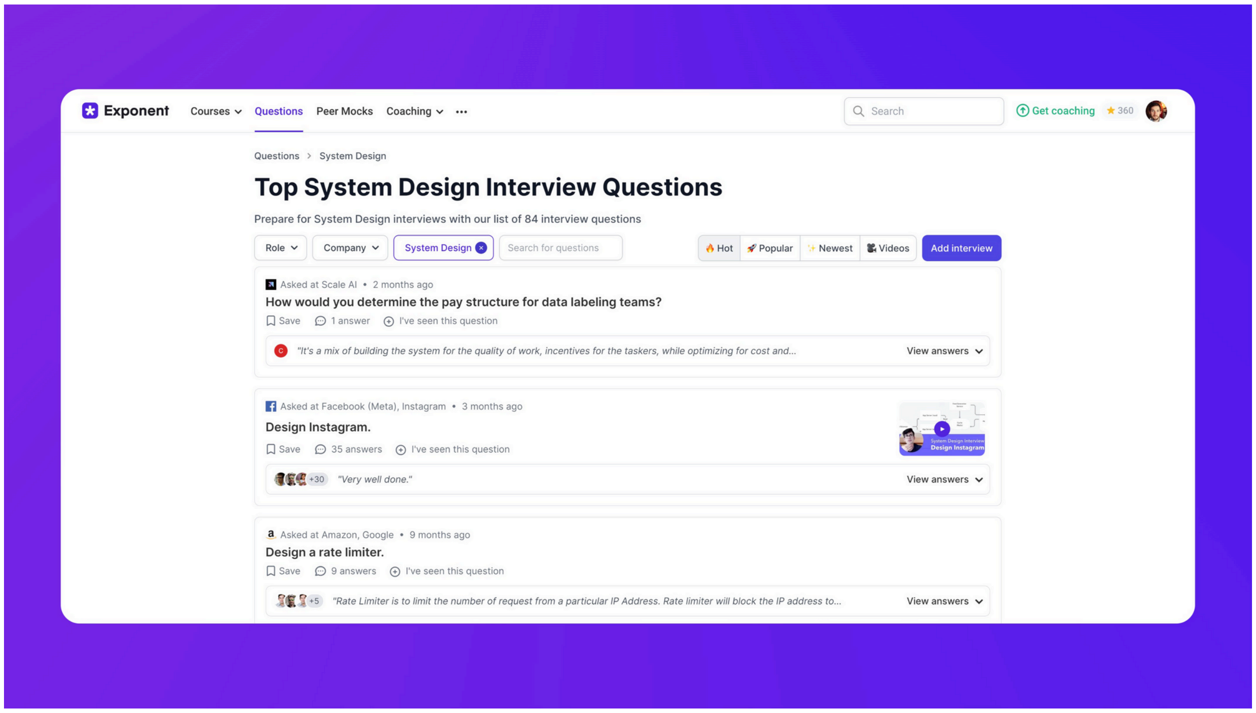
Design a bot-detection system for Minecraft.
Design a video game leaderboard.
Design a matchmaking system for similarly-skilled players.

Analytics

Design a metrics and logging service.

Job Processing

Design a web crawler.



Questions > System Design

Top System Design Interview Questions

Prepare for System Design interviews with our list of 84 interview questions

Asked at Scale AI • 2 months ago

How would you determine the pay structure for data labeling teams?

Save 1 answer I've seen this question

"It's a mix of building the system for the quality of work, incentives for the taskers, while optimizing for cost and... View answers

Asked at Facebook (Meta), Instagram • 3 months ago

Design Instagram.

Save 35 answers I've seen this question

+30 "Very well done." View answers

Asked at Amazon, Google • 9 months ago

Design a rate limiter.

Save 9 answers I've seen this question

+5 "Rate Limiter is to limit the number of request from a particular IP Address. Rate limiter will block the IP address to... View answers

View more expert and community answers to recently asked questions.

System Design Interview Framework

A system design interview is usually composed of 5 steps:

Step 1: Understand the problem. Get to know the problem and define the scope of the design.
Step 2: Design the system. Outline the most essential parts of the system and show how they work together to achieve the desired function.
Step 3: Explore the design. You or your interviewer will choose an interesting component and discuss its details.
Step 4: Improve the design. Consider the current design's issues and how to fix them and support more users.
Step 5: Wrap up. Check that the design meets all requirements and suggest ways to improve it.

💬
"Concentrate on the basic principles of system design instead of memorizing particular product setups.

Knowing the underlying principles will help you apply your knowledge to different situations and technologies, no matter the question." - Daisy M.

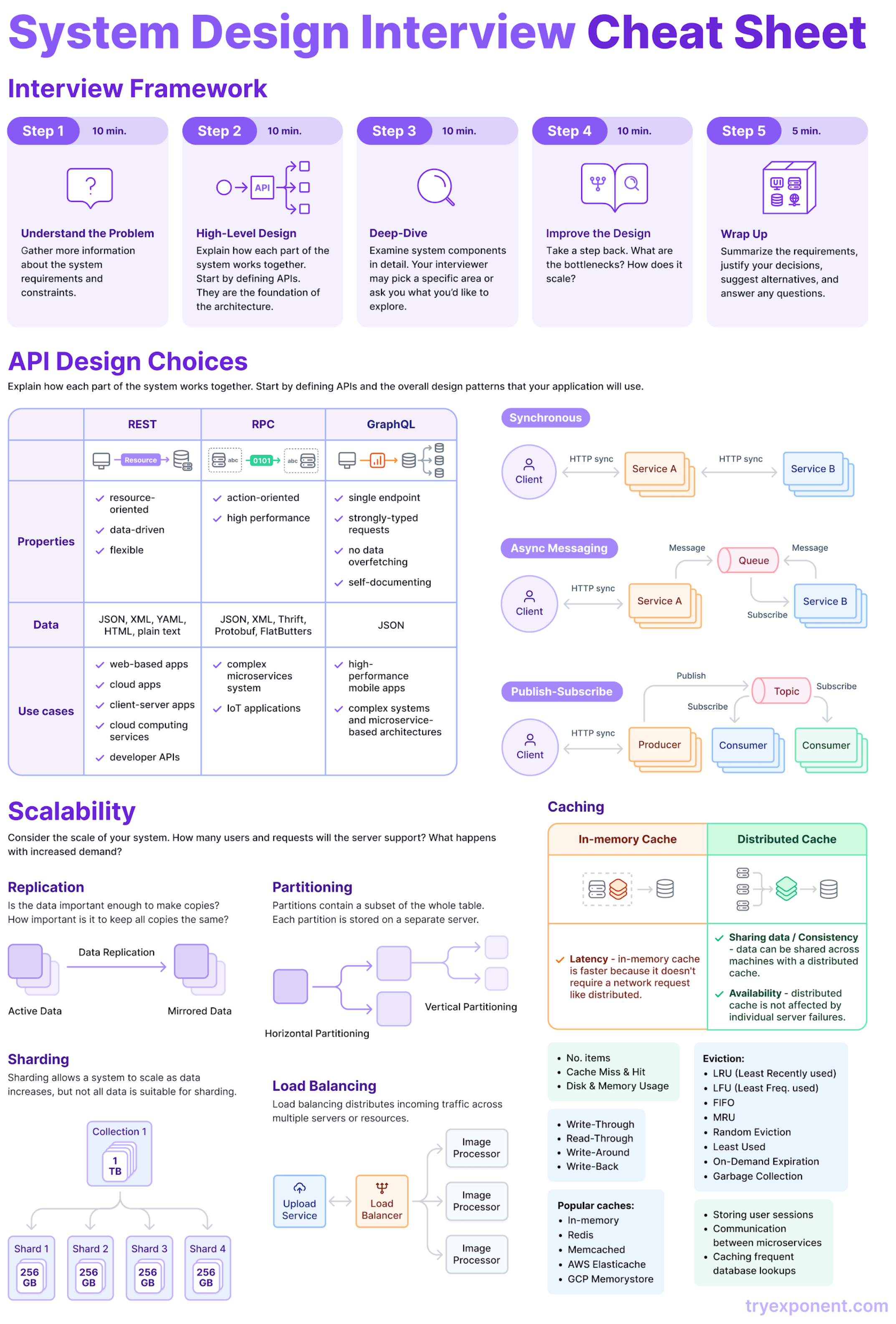
Daisy Modi is an Exponent technical interview coach and senior engineer with experience at Google, Uber, Adobe, and Twitter. Get coached by Daisy M.

A typical system design interview lasts between 45 to 60 minutes.

Good interviewers will leave a few minutes in the beginning for introductions and a couple at the end for questions.

This is just an estimate. Modify it based on your own personal interviewing style.

📁
Download: Exponent's free system design interview cheat sheet PDF.



System Design Interview Cheat Sheet

Interview Framework

Step 1 10 min.

Understand the Problem
Gather more information about the system requirements and constraints.

Step 2 10 min.

High-Level Design
Explain how each part of the system works together. Start by defining APIs. They are the foundation of the architecture.

Step 3 10 min.

Deep-Dive
Examine system components in detail. Your interviewer may pick a specific area or ask you what you'd like to explore.

Step 4 10 min.

Improve the Design
Take a step back. What are the bottlenecks? How does it scale?

Step 5 5 min.

Wrap Up
Summarize the requirements, justify your decisions, suggest alternatives, and answer any questions.

API Design Choices

Explain how each part of the system works together. Start by defining APIs and the overall design patterns that your application will use.

	REST	RPC	GraphQL
	Resource	abc 0101 abc	
Properties	✓ resource-oriented ✓ data-driven ✓ flexible	✓ action-oriented ✓ high performance	✓ single endpoint ✓ strongly-typed requests ✓ no data overfetching ✓ self-documenting
Data	JSON, XML, YAML, HTML, plain text	JSON, XML, Thrift, Protobuf, FlatButters	JSON
Use cases	✓ web-based apps ✓ cloud apps ✓ client-server apps ✓ cloud computing services ✓ developer APIs	✓ complex microservices system ✓ IoT applications	✓ high-performance mobile apps ✓ complex systems and microservice-based architectures

Scalability

Consider the scale of your system. How many users and requests will the server support? What happens with increased demand?

Replication

Is the data important enough to make copies? How important is it to keep all copies the same?

Partitioning

Partitions contain a subset of the whole table. Each partition is stored on a separate server.

Sharding

Sharding allows a system to scale as data increases, but not all data is suitable for sharding.

Load Balancing

Load balancing distributes incoming traffic across multiple servers or resources.

Caching

In-memory Cache	Distributed Cache
✓ **Latency** - in-memory cache is faster because it doesn't require a network request like distributed.	✓ **Sharing data / Consistency** - data can be shared across machines with a distributed cache. ✓ **Availability** - distributed cache is not affected by individual server failures.

- No. items
- Cache Miss & Hit
- Disk & Memory Usage

Eviction:
- LRU (Least Recently used)
- LFU (Least Freq. used)
- FIFO
- MRU
- Random Eviction
- Least Used
- On-Demand Expiration
- Garbage Collection

- Write-Through
- Read-Through
- Write-Around
- Write-Back

Popular caches:
- In-memory
- Redis
- Memcached
- AWS Elasticache
- GCP Memorystore

- Storing user sessions
- Communication between microservices
- Caching frequent database lookups

tryexponent.com

Step 1: Understand the Problem

Time estimate: 8 minutes

Start by gathering more information from your interviewer about the system's constraints.

Use a combination of context clues and direct questions to answer:

- What are the functional and non-functional requirements for the system's design?
- What is included and excluded from this design?
- Who are the clients/consumers?
- Do you need to talk to pieces of an existing system?

💡
Interview Tip: Avoid going into a design without first clarifying the problem.

Non-Functional Requirements

Once you've identified and confirmed the functional requirements with your interviewer, consider the non-functional requirements of the system design.

These may be related to business objectives or user experience.

Non-functional requirements include things like:

- availability,
- consistency,
- speed,
- security,
- reliability,
- maintainability,
- and cost.

Consider these questions to identify the non-functional requirements:

- What is the scale of the system?
- How many users should the application support?
- How many requests should the server handle?
- Are most use cases read-only?
- Do users typically read the data shortly after someone else overwrites it?
- Are most users on mobile devices?

If there are many design constraints and some are more important than others, focus on the most critical ones.

For example, if designing a Twitter timeline, focus on Tweet posting and timeline generation services instead of user registration or how to follow another user.

Requirement	**Question**
Performance	How fast is the system?
Scalability	How will the system respond to increased demand?
Reliability	What is the system’s uptime?
Resilience	How will the system recover if it fails?
Security	How are the system and data protected?
Usability	How do users interact with the system?
Maintainability	How will you troubleshoot the system?

Modifiability	Can users customize features? Can developers change the code?
Localization	Will the system handle currencies and languages?

Interview Tip: Discuss your non-functional requirements and your reasoning with the interviewer and check in with them. They may be interested in a particular aspect of your system, so listen to their hints if they nudge you in a specific direction.

Estimating Data

You can estimate the data volume roughly by performing some quick calculations.

For example, you can present queries per second (QPS), storage size, and bandwidth requirements to your interviewer.

This can help you pick components for your system. It will also give you an idea of scaling opportunities later.

As you estimate data, make some assumptions about user volume and typical user behavior.

💡
Interview Tip: Check with your interviewer if these assumptions match their expectations.

Step 2: High-Level System Design

Time estimate: 10 minutes

Next, explain how each part of the system will work together.

Start by designing APIs (Application Programming Interfaces).

APIs define how clients can access your system's resources or functionality via requests and responses.

Consider how clients interact with the system and the types of data they're passing through.

Clients may want to create/delete resources or read/update existing ones.

Each system requirement should translate to one or more APIs.

At this step, you should choose what type of APIs you want to use and why—such as:

> Representational State Transfer [REST],
> Simple Object Access Protocol [SOAP],
> Remote Procedure Call [RPC],
> or GraphQL.

Consider the request's parameters and the response type.

Once the APIs are established, they shouldn't be easily modified. They are the foundation of our system's architecture.

Server and Client Communication

Next, think about how the client and web server will communicate.

There are several popular options to choose from:

> Ajax Polling
> Long Polling
> WebSockets
> Server-Sent Events

Each has different communication directions and varying performance advantages and disadvantages.

	Pros	**Cons**

Ajax Polling	Easy to implement, works with all browsers	High server load, high latency
Long Polling	Low latency, less server load	High server load, not supported by all browsers
Websockets	Real-time communication	May require more complex server setup
Server-sent Events	Efficient, low latency	Unidirectional communication, not supported by all browsers

💡
Interview Tip: Discuss and explain your communication strategy with your interviewer.

To save time and keep your design in scope, avoid introducing APIs irrelevant to the functional requirements.

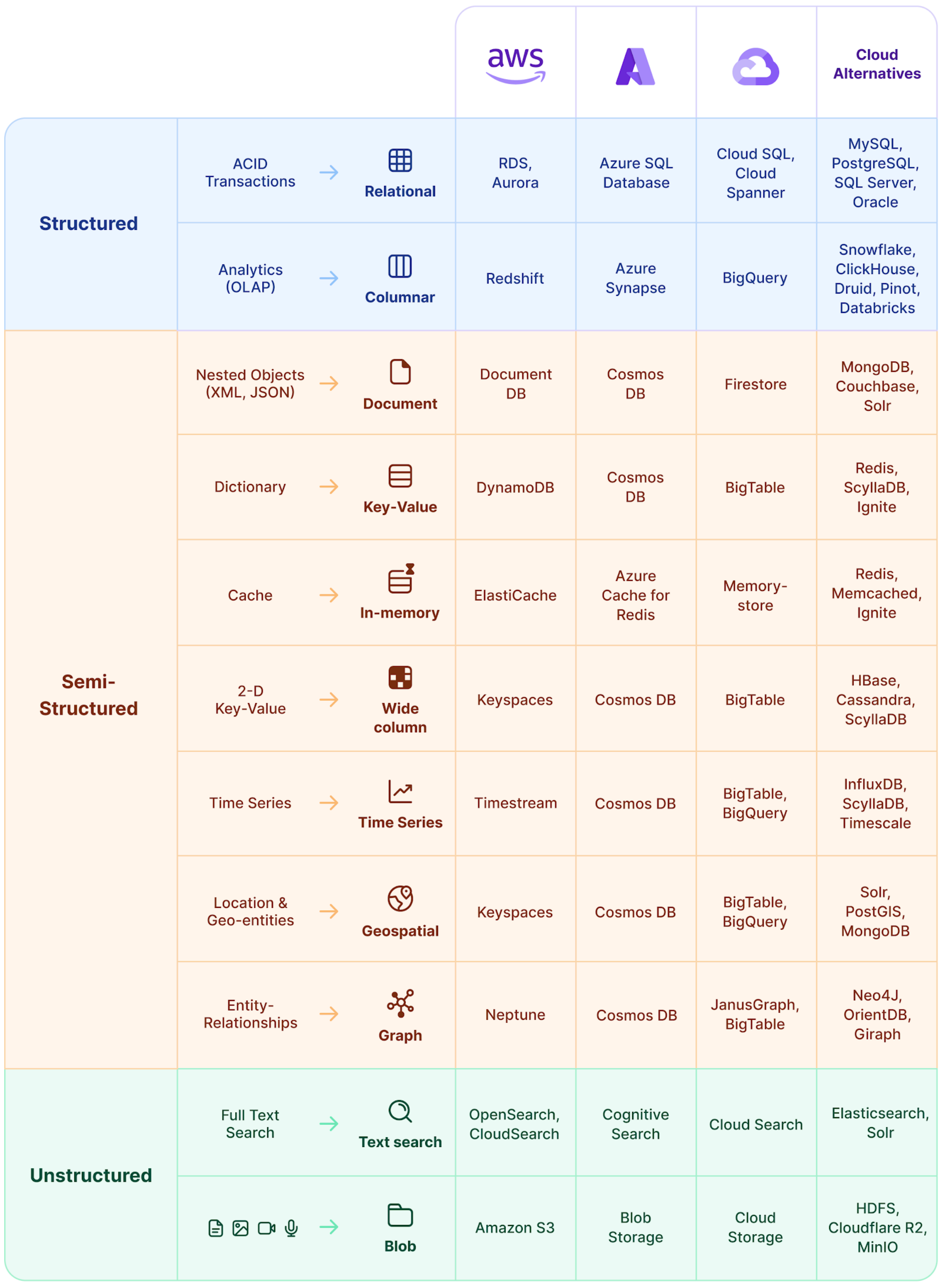
Data Modeling

Once you've designed the API and established a communication protocol, determine the core database data models.

This includes:

creating a simplified schema that lists only the most important fields,
discussing data access patterns and the read-to-write ratio,
considering indexing options,
and at a high level, identifying which database to use.

Database Cheatsheet for AWS, Azure, and Google Cloud



			aws			Cloud Alternatives
Structured	ACID Transactions →	Relational	RDS, Aurora	Azure SQL Database	Cloud SQL, Cloud Spanner	MySQL, PostgreSQL, SQL Server, Oracle
	Analytics (OLAP) →	Columnar	Redshift	Azure Synapse	BigQuery	Snowflake, ClickHouse, Druid, Pinot, Databricks
Semi-Structured	Nested Objects (XML, JSON) →	Document	Document DB	Cosmos DB	Firestore	MongoDB, Couchbase, Solr
	Dictionary →	Key-Value	DynamoDB	Cosmos DB	BigTable	Redis, ScyllaDB, Ignite
	Cache →	In-memory	ElastiCache	Azure Cache for Redis	Memory-store	Redis, Memcached, Ignite
	2-D Key-Value →	Wide column	Keyspaces	Cosmos DB	BigTable	HBase, Cassandra, ScyllaDB
	Time Series →	Time Series	Timestream	Cosmos DB	BigTable, BigQuery	InfluxDB, ScyllaDB, Timescale
	Location & Geo-entities →	Geospatial	Keyspaces	Cosmos DB	BigTable, BigQuery	Solr, PostGIS, MongoDB
	Entity-Relationships →	Graph	Neptune	Cosmos DB	JanusGraph, BigTable	Neo4J, OrientDB, Giraph
Unstructured	Full Text Search →	Text search	OpenSearch, CloudSearch	Cognitive Search	Cloud Search	Elasticsearch, Solr
	📄 🖼 📹 🎙 →	Blob	Amazon S3	Blob Storage	Cloud Storage	HDFS, Cloudflare R2, MinIO

Exponent

This system design database cheatsheet can help you decide between SQL and NoSQL database options for your design.

High-Level Design Diagram

After designing the API, establishing a communication protocol, and building a rough data model, the next step is to create a high-level design diagram.

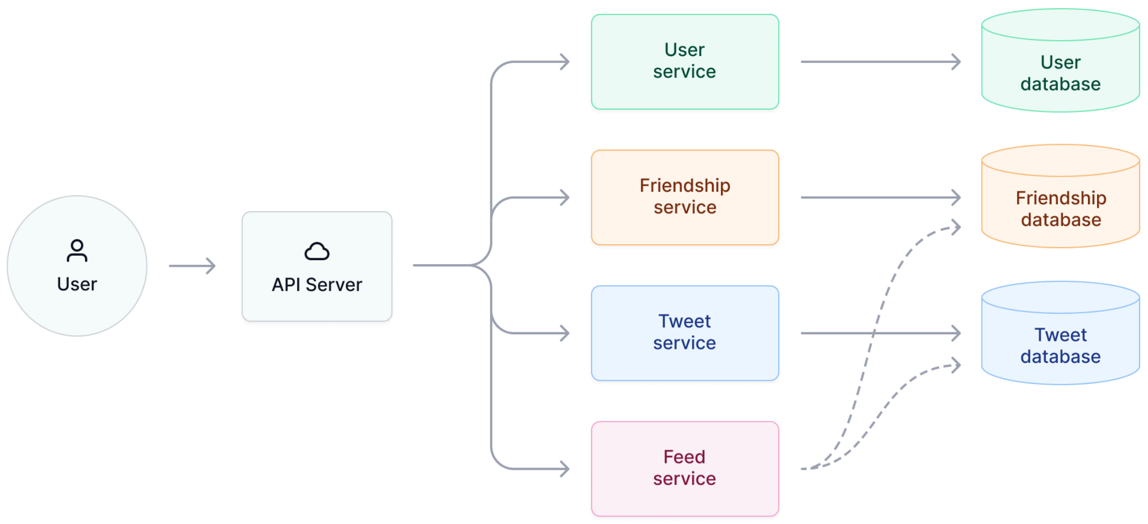
The diagram should serve as a blueprint for your design.

It highlights the most essential pieces to fulfill the functional requirements.

Don't need to go into too much detail about each service yet.

Your goal at this step is to confirm that your design meets all functional requirements.

Demonstrate the data and control flow for each requirement to your interviewer.



In this diagram, Twitter is abstracted into an API server, several services, and core databases. These servers and services are behind a load balancer. This helps route and balance traffic among different servers.

In the Twitter design example above, you could explain to your interviewer how the following features work:

How a Twitter user creates or logs in to their account
How a Twitter user can follow or unfollow another user
How a Twitter user can post a tweet
How a Twitter user can see their news feed

Step 3: Explore the Design: Deep-Dive

Time estimate: 10 minutes

Next, examine system components and relationships in more detail.

The interviewer may prompt you to focus on a particular area but don't rely on them to drive the conversation.

💡
Interview Tip: Check in regularly with your interviewer to see if they have questions or concerns in a specific area.

Considerations for Non-Functional Requirements

Consider how non-functional requirements impact design choices.

> Transactions: If the system requires transactions, consider a database that offers the ACID (Atomicity, Consistency, Isolation, and Durability) property.
> Data freshness: If an online system requires fresh data, think about how to speed up the data ingestion, processing, and query process.
> Data size: If the data size is small enough to fit into memory (up to hundreds of GBs), you can place it in memory. However, RAM is prone to data loss, so if you can't afford to lose data, you must find a way to make it persistent.
> Partitioning: If the volume of data we need to store is large, you may want to partition the database to balance storage and query traffic.
> Offline processing: If some processing can be done offline or delayed, you may want to rely on message queues and consumers.
> Access patterns: Revisit the data access pattern, QPS number, and read/write ratio, and consider how they impact your choices for databases, database schemas, and indexing options.

System design questions have no "correct" answer. Every question can be answered in multiple ways.

The most important skill of a system design interview is your ability to weigh trade-offs as you consider functional and non-functional requirements.

Step 4: Improve the Design: Bottlenecks and Scale

Time estimate: 10 minutes

After thoroughly examining the system components, take a step back.

Are there any bottlenecks in this system? How well does it scale?

Evaluate if the system can operate effectively under different conditions and has the flexibility to support future growth.

Consider these points:

- Single points of failure: Is there a single point that could cause the entire system to fail? How could the system be more robust and maintain uptime?
- Data replication: Is the data important enough to make copies? How important is it to keep all copies the same?
- CDNs: Does it provide a service for people all over the world? Would data centers in different parts of the world make it faster?
- High traffic: Are there any special situations, like when many people use the system simultaneously, that could make it slow or even break it?
- Scalability: How can the system work for 10 times more people?

Message Queues and Publish/Subscribe

Decoupling backend services is crucial for achieving scalability and reliability in system design.

By breaking down processes and implementing queuing mechanisms to manage traffic, systems can be optimized for high performance at scale.

An example of event-driven architecture is Pramp, a peer-to-peer mock-interview tool for software engineers.

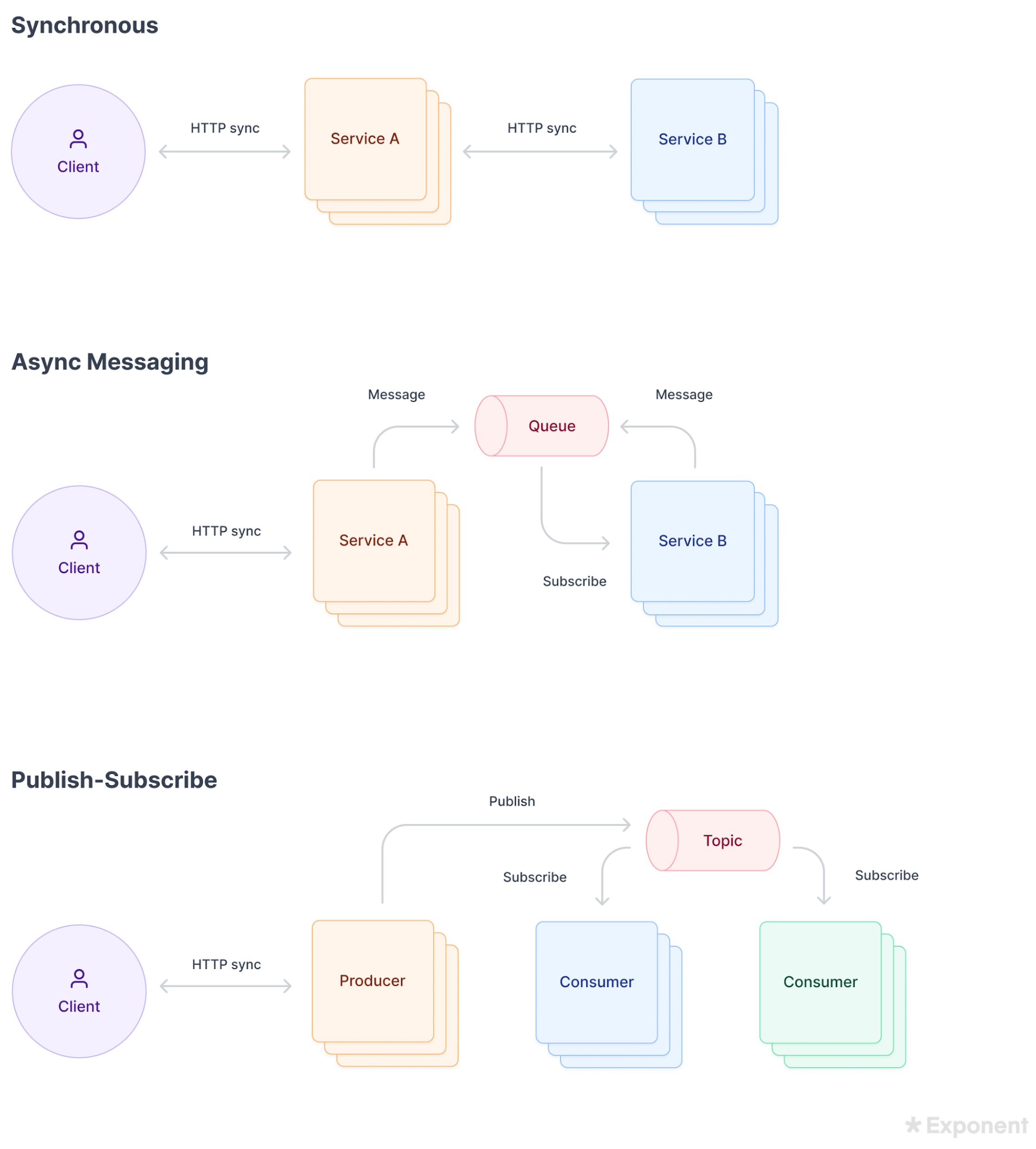
On Pramp, registering a user is handled as an asynchronous event, involving multiple services working in tandem.

Message Queues (MQs) play a pivotal role in enabling orderly and efficient message transmission to a single receiver.

On the other hand, Publish-Subscribe (Pub/Sub) systems excel at broadcasting information to multiple subscribers simultaneously.

- Message Queues (MQs): MQs are ideal for scenarios where processing jobs in a specific order is essential. They ensure that tasks are executed sequentially, maintaining the integrity of the workflow.
- Publish-Subscribe (Pub/Sub) systems: Pub/Sub systems shine when it comes to disseminating events or notifications to a large number of recipients concurrently.

Here are examples of synchronous, asynchronous, and pub/sub-messaging queues:



Decoupling backend services using synchronous, asynchronous, and pub/sub message queues can improve scalability and reliability.

Discuss Bottlenecks

To talk about bottlenecks, follow this structure.

Focus on the 2 or 3 most important limitations to keep your answer concise.

First, identify a bottleneck in the system.

Next, propose a single alternative to it.
Discuss the trade-offs of this alternative.
Decide and recommend an option between the alternative and your original solution.
Repeat for each bottleneck.

💬
"I appreciate it when candidates show that they have considered multiple options to solve a problem. This broad understanding of different technologies shows me that they are not simply memorizing answers or the use cases of a single tech stack." - Suman B

Suman B is an Exponent software engineering interview coach and an engineering manager at Amazon. Work with Suman.

Step 5: Wrap Up

Time Estimate: 4 minutes

This is the end of the interview. You can summarize the requirements, justify decisions, suggest alternatives, and answer any questions.

Walk through your decisions, providing justification for each and discussing any space, time, and complexity tradeoffs.

Throughout the discussion, refer back to the requirements periodically.

Leadership and Senior Candidates

💬
The following advice on leadership in SD interviews comes from Geoff Mendal. Geoff is an Exponent leadership coach and former software engineer with over 30 years of engineering experience at Google, Microsoft, and Pandora.

System design interviews help determine the level at which a candidate will be hired.

For junior engineers and new graduates, the focus on system design interviews is lesser. Junior candidates are expected to know the basics but not every detailed concept.

For instance, junior candidates don't need to know when to use NGINX or AWS' native load balancer. They only need to know that a load balancer is necessary.

However, for senior, staff, and lead candidates, having an in-depth understanding of system design and various trade-offs becomes vital.

Having more than one system design interview for higher-level roles is common.

During a system design interview, candidates often overlook or are not prepared for the evaluation of their leadership behaviors and skills.

In addition to assessing technical skills for designing at scale, the interviewer also tries to answer, "What is it like to work with you, and would they want you on their team?"

You can demonstrate leadership skills in an interview by:

- asking powerful open-ended questions at the outset, such as "What are the goals of this system?" and "What does success look like?"
- actively listening (sometimes referred to as level 5 listening),
- and collaborating with the interviewer rather than treating them as a stenographer. This is particularly important for more senior roles, where you will be expected to use leadership behaviors and skills heavily.

Demonstrating these skills during the interview is critical to receiving a positive evaluation.

Q1: Design ChatGPT

ℹ️
Watch our complete answer to how to "Design ChatGPT" here.

Some key considerations when designing ChatGPT include:

- Choosing a Database: Select a NoSQL database capable of handling large amounts of data.
- Reliable and Resilient Infrastructure: ChatGPT requires designing for scalability, security, and performance.
- Rate Limiting: Implement rate-limiting procedures to prevent DDoS attempts aimed at the system and ensure its safety.

Watch Neeraj Gupta, an eBay engineering manager, design a chatbot like ChatGPT.

When designing ChatGPT, several functional requirements, such as creating, updating, viewing, and deleting conversations, need to be considered.

Additionally, rating a response by giving thumbs up or down can help train the model.

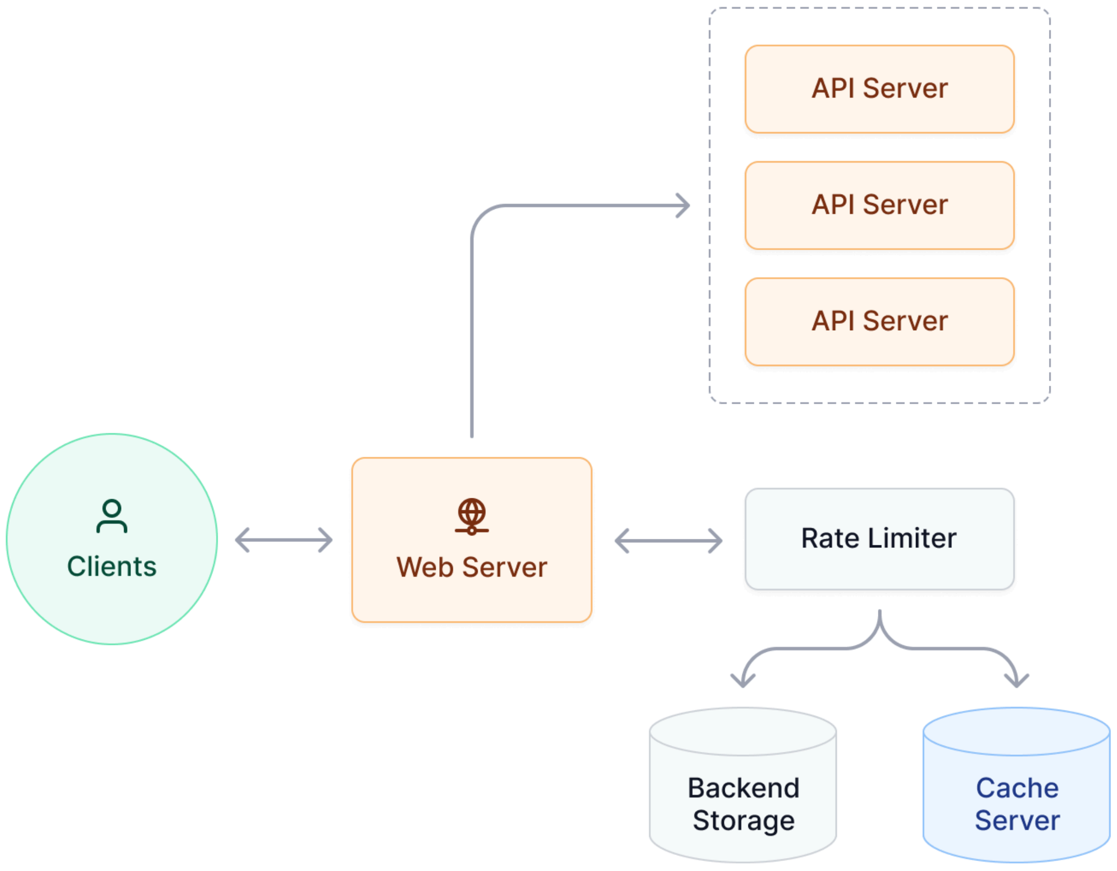
Text-based inputs in English are assumed, and inputs and outputs go through a sanitization phase to remove profanity and detect insults.

Non-functional Requirements

The latency for server responses can become lengthy due to extensive processing time on the back end.

Login flows and rate limitations can prevent DDoS attempts.

The rate limiter must also be scalable. This will help to host lots of users simultaneously without any issues.



Use rate limiting to protect your service from overuse - intentional or not.

A scalable database should be designed to handle this storage, with NoSQL being an ideal option.

The average message size is estimated at 100 bytes, meaning that 200 million messages per day translate to 20 GB per day and 7.3 terabytes annually, which amounts to 76 terabytes for ten years.

High-level Design

The system's high-level design should include a conversation service. This manages user inputs and the ChatGPT model, the system's core component.

Before a message is processed, it undergoes sanitization and analysis. This ensures it meets the standards required for processing.

The model's results are stored in a single conversation database.

Finally, a thumbs-down rating indicates that the model needs to be retrained, and a risk model is designed to detect the legitimacy of user ratings.

REST API

The conversation service is a REST API that includes creating, deleting, viewing, and sending a message. Each conversation has a unique ID assigned to each message. The user can rate each message with a thumbs up or thumbs down.

The data is stored in a NoSQL database, where the conversation table contains various conversation IDs, while each message contains an ID, text, author, and parent.

ChatGPT Model

ChatGPT uses a Transformer model, which predicts natural sequences of words.

The model is trained on internet data such as websites, books, and Wikipedia links to provide semantically meaningful and grammatically correct replies. The model can use multiple approaches such as top-K, greedy, or nucleus temperature to select the most accurate prediction.

A dataset of question-and-answer pairs is initially used to train the model.

However, since training the model on every possible question-and-answer combination is impossible, a reward model is designed to select the best responses. The reward model is also trained using reinforcement learning, where the chatbot is rewarded for giving appropriate responses and penalized for inappropriate ones.

The chatbot design is expected to support different input and output formats, including images, audio, and video.

Even if the model does not have a large dataset initially, it can still provide accurate responses in natural language. The reward model considers the emotion and tone of the question and answer and continuously trains itself to improve accuracy.

💬
"During interviews, tell an engaging story about your own work. Or weave a creative narrative about a fictional user using the application you’re talking about. How you communicate is just as important as your technical skills.

Your hiring manager or interviewer will likely interview multiple candidates in a single day. To stand out, try to make your interview memorable. " - Jess G

Jess Gardner is an Exponent communication and storytelling coach. Work with Jess G.

Q2: Design YouTube

ℹ️
Watch our complete answer to how to "Design YouTube" here.

A system like YouTube must cater to both content creators and consumers of the content. Content creators should be able to upload videos in different formats from any device, and the system should take care of post-processing.

The viewing experience should be device-agnostic, allowing for a seamless viewing experience across any screen size or device.

Non-Functional Requirements

High availability is crucial for non-functional requirements; eventual consistency can be targeted. The system needs to account for the ratio of reads to writes, which will heavily favor reads.

Content Creation

To facilitate content creation, the system should have an API for uploading metadata around the video and the video itself, with an open socket connection for large video transferring.

The system can use a queueing system for processing videos into different formats and resolutions, which will be stored in a relational database and blob storage like S3.

In addition, the system can shard the databases by users/creators for scalability.

Content Consumption

Viewers will use a streaming service with a Content Delivery Network (CDN) that checks the database to validate permissions and then retrieves the video from the blob storage, storing it in the CDN for future requests.

The system could partition data by geography or genre, but it makes sense to shard creator videos.

To optimize the user experience, an adaptive bitrate system can adjust streaming quality based on the user's connection speed. Additionally, an in-memory cache can be implemented to prevent overload during high-traffic periods, and an invalidation strategy like LRU can be used to efficiently remove outdated videos from the CDN.

For analytics purposes, a stream or analytics system can be established.

Improvements

One suggestion for improving the design is to add more fault tolerance measures, such as a master-safe slave configuration for the database.

Additionally, space and cost optimization should be considered, especially for a data-heavy system like a video streaming service.

Q3: Design a Parking Garage

This is a common Amazon system design interview question.

Consider dividing the system into public and internal endpoints and using a web or mobile app that interfaces with a back-end server.

Consistency and Scalability

To ensure high consistency and avoid double booking, use a strong consistency approach instead of an eventual consistency approach for a reservation system.

This approach would involve using read locks on replicas before writing to the Postgres database to keep the data as up-to-date as possible.

Additionally, sharding based on location can improve consistency, and read replicas and load balancing can help distribute load and maintain performance.

💬
"Good job on this breakdown! I do question the need for read-replicas, though. Another topic to consider is how to prevent two customers from paying for the same spot or how to lock a spot until payment is processed." - Venkat S. (Exponent member)

View more discussions about this question.

Data Schema

The data schema for the system should include a reservations table with foreign keys for garage ID, spot ID, start and end time, and payment status. The garage table should have an ID that is a primary key, a zip code, and rates based on vehicle size.

The spot table should include a serial primary key, a foreign key for garage ID, and a status that can be reserved, unavailable, or empty.

The optional users table may be added, including an ID that is the primary key, an email, first and last name, vehicle type, and user ID.

Trade-offs

Remember the potential trade-offs, such as flexibility versus enums in the vehicle type column and load balancing versus maintaining consistency.

Third-party Payment System

Consider using an existing third-party payment system instead of developing one in-house to save time and resources.

Q4: Design Twitter

ℹ️
View the full answer to how to "Design Twitter" here.
Hozefa, an engineering manager at Wealthfront, talks through how to design Twitter.

To design a minimum viable product (MVP) for a two-sided network on Twitter, focus on tweet creation, generating a timeline, and allowing users to follow others and interact with tweets.

These endpoints would be handled by multiple servers behind load balancers.

Timeline Generation

For timeline generation, active users' timelines are updated using a stack-like approach, adding daily tweets and considering user engagement for optimal timeline generation.

Influencers require a more dynamic approach, with data added to timelines on the fly when requested.

High Availability

High availability is maintained through multiple servers and databases set up in a master-slave configuration.

Twitter must optimize for reads and high availability as a read-heavy system while ensuring influencers cannot bring the system down.

Non-functional Requirements

Additional non-functional requirements include eventual low latency and high availability.

Interaction with multimedia and text content is similar, and the system would need algorithms that prioritize user engagement to limit the subset of tweets generating the most interaction.

Security Considerations

Consider potential security issues such as DDoS attacks. Login flows and rate limitation procedures can prevent DDoS attempts aimed at the system, ensuring its safety.

When making this application scalable, have an API server read from a separate cache for the newsfeed.

You should also use a feed service to refresh the feed cache regularly.

Q5: Design Netflix

Static Content

Use blob storage services like Amazon S3 or Google Cloud to handle static content. This will give you fast response times and high availability.

Metadata

Use a strong SQL database management system like PostgreSQL that works well with large-scale indexing.

Using a cache for frequently-accessed content will ensure that it's easy to get to and will give you faster response times. Using in-memory caching with Redis or Memcached for metadata could also improve performance.

Consistency, Availability, and Partition Tolerance

Balance the trade-offs between consistency, availability, and partition tolerance. The CAP theorem can help you decide which factors are most important in your application. A load balancer and sharding can help balance the load on the database server.

CDNs

Use a content delivery network (CDN) for low-latency delivery of globally-distributed media.

With an estimated 200 million users, optimizing for global impact is a key part of the design process.

Q6: Design TikTok

ℹ️
Watch our complete answer to how to "Design TikTok" here.

Consider the Scale of the System

To store compressed video files and user metadata, you need an always-available system that responds quickly and can grow to support more users. TikTok has one million users who use it every day, so the system must be able to handle that many people.

API and Database Schema Design

An API can be designed with endpoints such as "Upload Video" and "Get User Activity."

A relational database system like PostgreSQL can store and link structured user data objects.

To reduce the load on the database and improve latency, preloading a cache of the top ten videos for each user can be implemented.

Video UUID and ID Field

To associate user activity, such as liking and following videos, with specific videos, a video `UUID` and `ID` field is required.

User activity can be stored in a database, and an API endpoint can be developed for a `GET` request to return a list of the user's likes and followers.

Content Delivery Network and Load Balancer

To handle a 10x increase in traffic, a Content Delivery Network (CDN) can cache and route video content traffic to the closest node.

A load balancer can also be used for scaling deployments and performing zero downtime deployments.

Regional Database Sharding and Pre-Caching Service

A regional database sharding service can facilitate load distribution between databases and a read-only worker.

A pre-caching service can be implemented to manage `GET` requests and preloading video content.

Q7: Design Uber Eats

Functional Requirements

The design of Uber Eats requires attention to both functional and non-functional requirements.

First, define the needs of all stakeholders – restaurants, customers, and delivery people.

The top priority is to design a system that allows restaurants to add their information and customers to view and search for nearby restaurants based on delivery time and distance. This system requires eventual consistency to ensure the accuracy of restaurant information.

Non-Functional Requirements

High availability, security, scalability, and latency are all critical non-functional requirements. The expected numbers of daily views, customers, and restaurants must be taken into account.

Data Modeling

Successful design involves data modeling for restaurant and menu items, including geohashing for optimizing delivery routes.

To optimize proximity comparisons between different locations such as restaurants and customers, the system subdivides the world into smaller grids and maps them into binary values encoded using base 32.

This encoding system optimizes proximity matching between different locations.

Data modeling involves using relational and NoSQL databases for different data tables and optimizing the user experience for uploading new restaurant information.

Services and Components

Employing ElasticSearch service on Cassandra for sharding and scaling with separate databases for optimization purposes is crucial.

Multiple services such as a search service, viewing service, and restaurant service could be employed for efficiency, with the search service incorporating ElasticSearch and geohashing.

Isochrones and polygons help estimate delivery times and identify delivery areas.

The viewing service could include caching frequently viewed menus and leveraging isochrones to estimate customer delivery times.

The restaurant service could track all events using Kafka queues and implement an API for moderation, optimizing the system's efficiency.

Other Considerations

Scalability, optimization, and customer experience are critical factors in Uber Eats' design.

A seamless system must be designed to cater to all stakeholders' needs.

Considering all stakeholders' requirements, an efficient system can be developed to provide a seamless experience.

Google System Design Interviews

💬
Preparing for a Google engineering manager interview? This advice comes from an Exponent Slack community member, Yaro. Yaro is a Google engineering manager.

"I recently completed the Google engineering manager interview loop. These were regular system design interviews with different engineers and teams.

To prepare for the interviews, I watched a lot of mock interviews from Exponent. I also read some books and practiced answering system design questions in Google

Docs. I practiced writing solutions for 3-4 systems, including Google Drive, Instagram, a hotel booking system, Google Maps, an analytics system, and blob storage.

A coding interview round was also evaluated by an L6 engineering manager. They advised me to spend time understanding which database to choose.

I recommend checking out Alex Xu's system design database table and use cases on Twitter. Spend an evening learning about all the different use cases for these database types. Google likes to ask detailed questions about database selections.

Additionally, I reviewed all of the databases used by Google, including Bigtable, Spanner, Firestore, and BigQuery. This gave me a few more points with the interviewer since I approached the problems with their internal tech, not just AWS or Azure. This was probably overkill, but it helped me feel more prepared."

Amazon System Design Interviews

During an Amazon system design interview, a big focus will be on behavioral questions based on Amazon's Leadership Principles.

However, the interview will also evaluate your technical, functional job fit, specifically in system design.

Focus on the big picture rather than becoming an expert on the specific system they want you to create.

Whether you come from a FinTech or HealthTech background, Amazon will likely ask you to design an Amazon-type product. This could be Alexa or Amazon Prime.

Focus on the fundamentals that create a cohesive experience across different layers required for a complex environment to work.

During the interview, you may be asked to optimize your solution or test different parameters to see how you adjust the scope and handle unforeseen circumstances.

Fundamental Concepts

The last part of this guide is a breakdown of the fundamental principles and concepts of designing scalable systems.

Network and Web Protocols

Network and web protocols are the rules and standards that govern how information is transmitted over the internet.

Refresh your knowledge of these key web protocols before your interview:

- HTTP: the primary protocol used for transferring data between a web server and a client.
- HTTPS: a secure version of HTTP that encrypts data transmitted between a client and a server.
- TCP/IP: a suite of communication protocols that connect devices on the internet and transmit data between them. It is the primary protocol used on the internet and routes data between devices on different networks.
- OSI Model: a framework describing how different communication protocols and technologies work together to enable communication between computers and other devices on a network. It divides the communication process into seven distinct layers (instead of TCP/IP’s four), each of which serves a specific purpose.

Sharding

Typically, data is stored in tables with rows and columns (RDBMS) in a 1-to-1 relationship.

A normalization process stores data with 1-to-N or N-to-N relationships in separate tables joined by Foreign Keys.

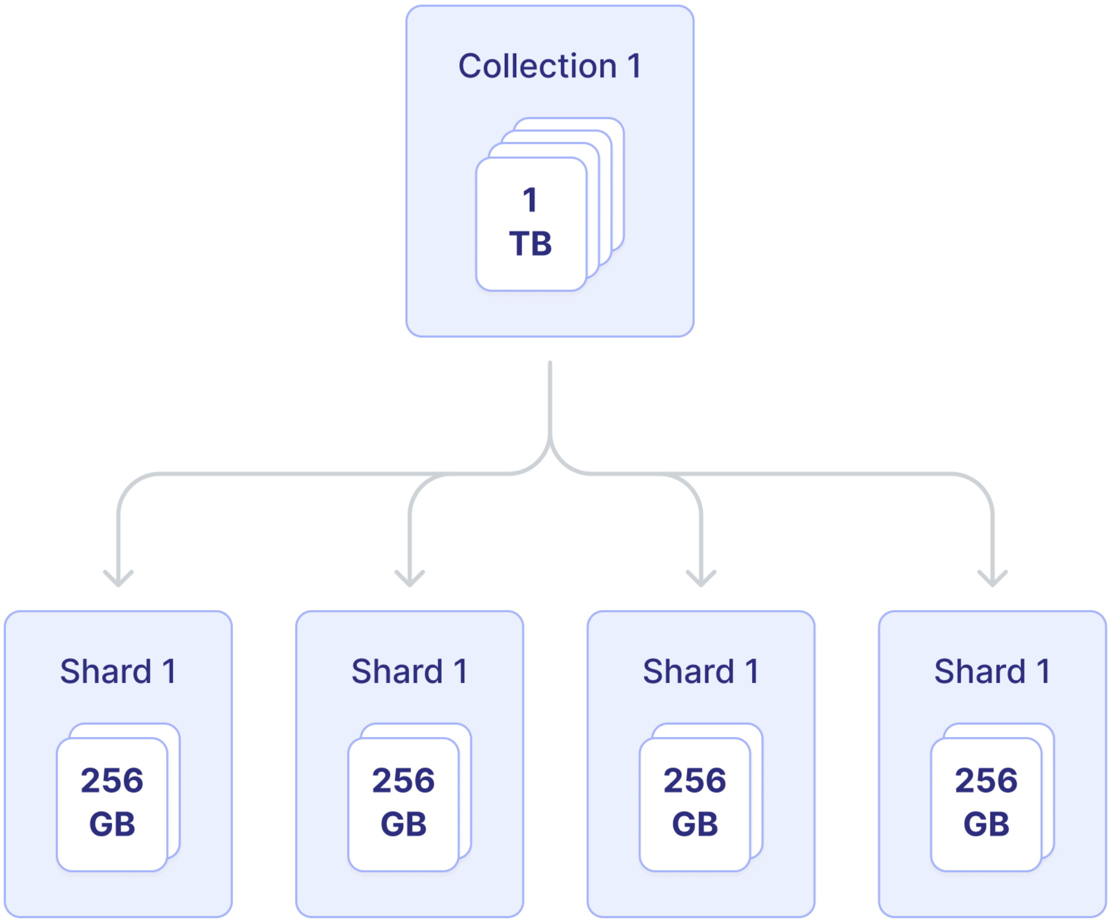
This ensures that the data in these tables are consistent and can be joined for a complete view of the data.

As data size increases, traditional database systems face CPU, memory, or disk usage bottlenecks that require high-end and expensive hardware.

However, even with top-quality hardware, most successful modern applications require more data than a traditional RDBMS can handle.

Sometimes, data is split into large tables with horizontal data partitions. Each partition contains a subset of the whole table. And each partition is stored on a separate database server.

This process is called sharding. Each partition is called a shard.



Sharding stores large amounts of data across multiple servers.

Sharding Techniques

The technique used to partition data often depends on the data's structure.

Some common sharding techniques include:

Geo-based Sharding

This technique partitions data based on the user's location, such as their continent of origin or a similarly large area (e.g., "East US," "West US").

This technique allows users to be routed to the node closest to their location, reducing latency.

Bottlenecks: There may not be an even distribution of users in the various geographical areas.

Range-based Sharding

Range-based sharding divides the data based on the ranges of the key value.

For example, selecting the first letter of the user's first name as the shard key divides the data into 26 buckets (assuming English names).

Bottlenecks: This simplifies partition computation but can lead to uneven splits across data partitions. In this example, more users have names starting with the letter A than Z.

Hash-based

This technique uses a hashing algorithm to generate a hash from the key value. It then computes the partition using the hash value.

A good hash algorithm distributes data evenly across partitions, reducing the risk of hotspots.

Bottlenecks: It can assign related rows to different partitions, so the server cannot enhance performance by predicting and pre-loading future queries.

Advantages vs. Disadvantages of Sharding

Consider the pros and cons of sharding techniques before suggesting one in an interview.

Advantages

- Sharding allows a system to scale out as data size increases, enabling it to handle more data than a traditional RDBMS.
- Having a smaller set of data in each shard results in smaller indexes and faster query performance.
- In case of an unplanned outage that takes down a shard, most of the system remains accessible while that shard is restored, minimizing downtime.
- Smaller amounts of data in each shard mean that nodes can run on commodity hardware, avoiding the need for expensive high-end hardware to deliver acceptable performance.

Disadvantages

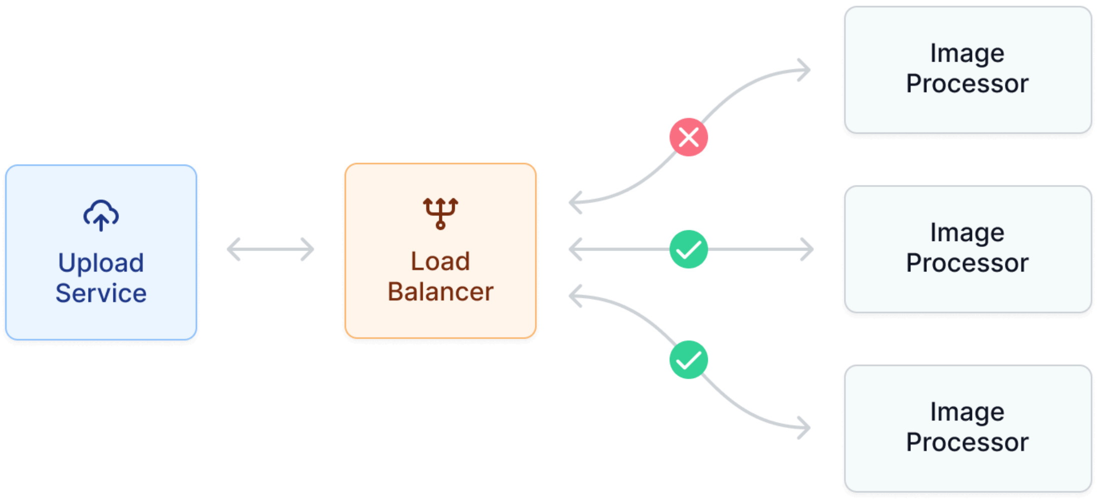
Sharding may not be suitable for all types of data.
Foreign key relationships can only be maintained within a single shard.
Manual sharding can be very complex and can lead to hotspots.
Because each shard runs on a separate database server, some types of cross-shard queries (such as table joins) are either very expensive or not possible.
Once sharding has been set up, it is very hard (if not impossible) on some systems to undo sharding or to change the shard key.
Each shard is a live production database server, so it needs to ensure high availability (via replication or other techniques), increasing operational costs compared to a single RDBMS.

Load Balancing

Load balancing is a technique used to distribute incoming traffic across multiple servers or resources to ensure that no single server becomes overloaded and unable to handle the traffic.

It allows a system to scale horizontally, meaning it can handle a larger workload by adding more servers or resources rather than relying on a single, powerful server.

Load balancers are essential to many modern technical systems and frequently come up in system design interviews.



A load *balancer* is a server that distributes incoming web traffic across multiple backend servers.

Load Balancing Algorithms

Load balancers can use two types of algorithms: static and dynamic.

- Static algorithms, like round-robin, distribute traffic evenly to all servers.
- Dynamic algorithms consider each server's current state and distribute traffic accordingly.

CDNs (Content Delivery/Distribution Networks)

Content Delivery Networks (CDNs) are a distributed network of servers that deliver content, such as web pages, web documents, images, and videos, to users based on their geographic location.

CDNs replicate content across a network of servers located in strategic locations around the world. When a user requests content, the CDN determines the user’s location and subsequently directs the request to the server that is closest to the user. In doing so, CDNs reduce latency and improve the overall user experience.

CAP Theorem

A distributed database system cannot guarantee all three of the following properties simultaneously:

- Consistency
- Availability
- Partition tolerance

Instead, a system must choose between consistency and availability in the face of network partitions.

ℹ️
A system can either prioritize consistency and sacrifice some availability or prioritize availability and sacrifice some consistency.

Databases

Databases are a critical component of technical systems and will inevitably be involved in your system design interviews.

A database is a structured collection of data that is stored and accessed electronically.

There are many kinds of databases you can choose from when designing systems:

Relational databases
NoSQL databases
Object-oriented databases
Graph databases

Understand the trade-offs between different database technologies and how to choose the best database for a particular application.

Caching

Caching stores frequently accessed data in a temporary storage location, typically in memory, to improve the performance of a system.

Caching is commonly used in system design because it can significantly improve the speed at which a system can retrieve data. There are several types of caching that are used often:

Client-side caching
Server-side caching
CDN caching

System Design Interviews vs. Coding Interviews

System design interview questions are similar to coding questions in that they are fundamentally technical.

However, they differ in a few key ways:

System design questions are vague on purpose: Whereas coding questions must be clear to the candidate to answer effectively, system design questions are just the opposite. Here, the interviewer is less concerned about getting a correct answer per se. Instead, they are interested in how you think about answering a question.
System design interview questions do not have straightforward answers: Interviewers won't be looking for the "right" answer. While both good and bad designs are possible, your interviewer is evaluating the design choices you make. They will assess how you choose between tradeoffs along with your thought process more generally.
System design interviews are two-way discussions: The most unique aspect of system design interviews is the two-way nature between candidate and interviewer. During your answer, you will ideally work with your hiring manager along the way. Ask plenty of clarifying questions throughout your interview. Even if your recruiter answered questions about the job, you could

ask the same questions to your hiring manager to show you're engaged with the position.

Interview Tips

You will need knowledge and comfort with diverse technologies to effectively answer these interview questions.

Engineers, for example, will need to elaborate deeply on the systems within their areas of expertise.

However, management roles, such as TPM, need a much broader knowledge of the systems and technologies they use.

Define success: Don't forget to clearly define the who and the what of your solution early on when clarifying requirements. Explain the nature of the problem and refer back to it frequently as you build your system.
Ask clarifying questions: You wouldn't design and implement a whole system without plenty of back-and-forth communication in the real world (we hope), so don't do it here.
Answer the "why": A successful answer is always preemptively answering the "whys?" that come alongside each design decision. Clearly explain why your design decisions are appropriate for the problem.
Be thorough: Carefully explain why you make the decisions you do. Don't skip something, even if it seems obvious! Your interviewer is highly invested in your thought and decision process. Explaining the obvious is a critical piece of that.
Architectures: Borrow from the design architectures that you are most comfortable or experienced with. So long as you can substantively explain why it is the best architecture for the required solution. This doesn't mean you should try to fit every potential system design into the same architectural pattern, though.

FAQs about SD Interviews

These are some of the most commonly asked questions around prepping for these tough interviews.

Does Amazon ask system design interview questions?

Yes and no. Amazon asks system design questions in their engineering interviews.

However, they don't ask these types of questions to freshers and recent graduates. System design questions are usually only asked in interviews for experienced positions (4-5 years of experience).

Does Google ask system design interview questions?

Yes, Google asks system design questions. They are asked during the technical phone interviews.

Your initial phone screens won't have any system design elements to them.

Instead, you'll be asked about algorithms and data structures. You'll encounter system design questions if you're advanced to the next interview round.

To pass the Google system design interview, focus on your whiteboarding skills.

Are system design interviews difficult?

System design interview questions are notoriously difficult to prepare for. Unlike algorithmic questions, they don't reduce down to a handful of prescribed patterns. Instead, they require years of technical knowledge and experience to answer well.

For junior engineers, this can be tricky. Even senior developers sometimes find themselves scratching their heads trying to understand a system.

The key to doing well in these types of interviews is to use your entire knowledge base to think about scalability and reliability in your answer.

What is the difference between high-level and low-level system design?

The high-level design focuses on the problem you're trying to solve. The low-level design breaks down how you'll actually achieve it. That includes breaking down systems into their individual components and explaining the logic behind each step. System design interviews focus on both high-level and low-level design elements so that your interviewer can understand your entire thought process.

Everything you need to know for your next interview

Create your free Exponent account and learn how to ace your interviews.

Get started free ->