System Design Interviews: How Senior Engineers Think and Solve Architecture Problems

Subtitle: Learn how experienced engineers approach large-scale architecture problems, communicate technical decisions, and think during real system design interviews.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app

Introduction

Most developers fear system design interviews because they misunderstand what companies are actually testing.

Companies are NOT checking:
• memorized diagrams
• random buzzwords
• perfect architecture answers

They are evaluating:
• engineering thinking
• scalability understanding
• tradeoff analysis
• communication clarity
• decision-making ability

This is why many developers fail system design interviews even after watching dozens of tutorials.

System design interviews are less about:
• knowing everything

and more about:
• thinking like an engineer under uncertainty.

This PDF explains:
• how system design interviews work
• how senior engineers approach problems

• how to structure architecture discussions
• how to avoid beginner mistakes

Chapter 1: What Companies Actually Evaluate

Most candidates think:

> “I need the perfect architecture.”

Wrong.

Interviewers care more about:
• thought process
• reasoning
• tradeoff awareness

They evaluate:

1. Problem understanding

Can you clarify requirements correctly?

---

2. Scalability thinking

Can system handle growth?

---

3. Communication

Can you explain technical ideas clearly?

---

4. Tradeoffs

Do you understand compromises?

---

5. Prioritization

Can you identify important components first?

Chapter 2: Biggest Beginner Mistake

Beginners immediately start drawing architecture.

Senior engineers first:
• clarify requirements
• define constraints
• estimate scale

Real Engineering Thinking Starts With Questions

Example questions:

• How many users?
• Real-time or not?
• Read-heavy or write-heavy?
• Availability requirements?
• Expected traffic?
• Data size?

Why This Matters

Architecture changes completely depending on constraints.

Example

1000-user system ≠ 100 million-user system

Chapter 3: Functional vs Non-Functional Requirements

Functional Requirements

What system should do.

Example:
• users can upload videos

Non-Functional Requirements

How system should behave.

Example:
• low latency
• scalability
• reliability

Most beginners ignore non-functional requirements.

That is a major mistake.

Chapter 4: Estimating Scale

Senior engineers estimate before designing.

Example Calculations

Users:
• 10 million daily users

Requests:
• 500 requests per second

Storage:
• 5 TB per month

Why Estimation Matters

It determines:
• database choices
• caching strategy
• infrastructure needs

Chapter 5: High-Level Design First

Good interviews start broad.

Basic Structure

Client
↓
Load Balancer
↓
Application Servers
↓
Cache
↓
Database
↓
Storage/CDN

Important Rule

Do NOT start with deep technical details immediately.

Chapter 6: Breaking System Into Components

Large systems are divided into services.

Example: Video Platform

Separate services:

• upload service
• transcoding service
• recommendation system
• notification system
• analytics service

Why Separation Matters

Improves:
• scalability
• maintainability
• reliability

Chapter 7: Database Design Thinking

Interviewers expect database reasoning.

Questions to Consider

• SQL or NoSQL?
• Read-heavy or write-heavy?
• Strong consistency needed?

Example

Banking systems:
• strong consistency required

Social media feed:
• eventual consistency acceptable

Chapter 8: Caching Strategy

Caching is expected in scalable systems.

Common Cache Targets

• popular content
• user sessions
• timelines
• frequently accessed data

Important Insight

Without caching:
• databases become bottlenecks quickly

Chapter 9: Load Balancing

Large systems require traffic distribution.

Why?

Single server:
• cannot handle massive scale

Load balancer helps:

• distribute requests
• improve reliability
• prevent overload

Chapter 10: CDN Thinking

Systems serving media use CDNs.

Example

YouTube stores videos globally.

Users access nearest server:
• lower latency
• faster delivery

Interview Insight

Mentioning CDN appropriately shows:
• infrastructure awareness

Chapter 11: Asynchronous Systems

Real systems avoid synchronous bottlenecks.

Example

Uploading video triggers:

• encoding
• thumbnail generation
• notifications

These tasks happen asynchronously.

Technologies Used

• queues
• Kafka
• workers

Chapter 12: Reliability Engineering

Senior engineers design assuming failure.

Example Failures

• server crashes
• database outage
• traffic spike

Systems require:

• redundancy
• retries
• failover mechanisms

Chapter 13: Tradeoffs Matter More Than Perfection

Every design decision has tradeoffs.

Example

SQL:
• strong consistency
• harder horizontal scaling

NoSQL:
• scalable
• weaker consistency sometimes

Important Truth

No perfect architecture exists.

Chapter 14: Real-Time Systems Thinking

Real-time systems require different architecture.

Example

Chat application needs:

• WebSockets
• persistent connections
• low latency messaging

Compared To

Blog website:
• simple request-response model enough

Chapter 15: API Gateway Concept

Large systems often use API gateways.

Responsibilities

• authentication
• routing
• rate limiting
• monitoring

Benefit

Centralized request management.

Chapter 16: Data Partitioning and Sharding

Massive datasets require splitting.

Example

User data distributed across servers.

Benefit

• scalability
• reduced database pressure

Challenge

Adds operational complexity.

Chapter 17: Monitoring and Observability

Production systems need visibility.

Engineers monitor:

• error rates
• traffic patterns
• latency
• server health

Important Insight

You cannot manage systems you cannot observe.

Chapter 18: Security Awareness

Good system design includes security thinking.

Examples

• authentication
• authorization
• encryption
• rate limiting

Security is NOT optional in production systems.

Chapter 19: Common Interview Mistakes

Mistake 1

Jumping into deep details too early

Mistake 2

Ignoring scale assumptions

Mistake 3

Using buzzwords without understanding

Mistake 4

Not discussing tradeoffs

Mistake 5

Overengineering small systems

Chapter 20: What Senior Engineers Do Differently

Senior engineers:

• simplify complexity
• prioritize correctly
• communicate clearly
• think in systems
• identify bottlenecks early

Important Observation

Senior-level thinking is:
• structured
not:
• chaotic knowledge dumping

Chapter 21: How to Practice System Design Properly

Most people practice incorrectly.

Bad Practice

Memorizing architecture diagrams.

Good Practice

Choose one system:
• Instagram
• WhatsApp
• YouTube

Then analyze:

• traffic flow
• scaling challenges
• bottlenecks
• tradeoffs

Chapter 22: Engineering Communication Matters

System design interviews are collaborative discussions.

Interviewers expect:

• clarity
• structure
• reasoning

Poor communication destroys strong technical knowledge.

Chapter 23: The Most Important Mindset Shift

Beginner mindset:
• “What technology should I use?”

Engineer mindset:

• “What problem am I solving?”

Technologies are tools.

Architecture decisions depend on:
• constraints
• scale
• reliability needs

Chapter 24: Real Engineering Principle

System design is:

> controlled compromise under constraints

Every decision balances:

• speed
• complexity
• cost
• scalability
• reliability

Chapter 25: Final Interview Truth

Companies know:
• nobody builds perfect systems alone

They want engineers who can:
• reason clearly
• collaborate effectively
• adapt under uncertainty

Key Takeaways

• System design interviews evaluate engineering thinking
• Clarifying requirements is critical
• Tradeoff analysis matters more than memorization
• Scalable systems require caching, load balancing, and distribution
• Reliability engineering is part of architecture design
• Communication skills strongly impact interview performance
• Senior engineers think structurally under constraints
• Real system design is about solving problems efficiently

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app