



BY LEARNSTACK •

2026 Edition

System Design Handbook

Complete Architecture & System Design
Reference for Placements & Interviews

Scalability

APIs

Caching

Databases

CAP

Microservices

Load Balancing

Learn. Build. Grow.



linktr.ee/LearnStack

About This Handbook

This handbook is a premium, beginner-friendly System Design reference for CSE students, junior developers, self-learners, and SDE interview candidates.

What you will learn

- How real systems are built and why design choices matter.
- How to handle millions of users with caching, queues, databases, and load balancing.
- How top engineering teams reason about trade-offs and failure.
- How to answer system design interviews step by step.

Who it is for

- CSE students preparing for placements and interviews.
- Junior developers leveling up to backend/system design.
- Self-learners entering the tech industry.
- Anyone preparing for SDE system design rounds.

Prerequisites

- Basic programming knowledge.
- Basic understanding of websites and the internet.
- No prior system design experience needed.

TIP - How to use this handbook

Read linearly for complete understanding, use diagrams as quick reference, and practice designing a system after every chapter.

© 2026
LearnStack

All rights reserved.
Educational use only.
Resale without permission is
strictly prohibited.

linktr.ee/LearnStack

Learn. Build. Grow.

Table of Contents

Two-page chapter and subsection roadmap with page numbers.

About & Copyright	2	4.1 What is a Database?	29
01. What is System Design?	5	4.2 SQL vs NoSQL - The Big Decision	30
1.1 Introduction to System Design	6	4.3 SQL Databases - Deep Dive	31
1.2 Why System Design Matters	7	4.4 NoSQL Databases - Complete Guide	32
1.3 What Interviewers Expect	8	4.5 Database Scaling	33
1.4 High Level vs Low Level Design	9	4.6 Database Replication	34
1.5 How to Approach Any System Design Problem	10	4.7 Database Sharding	35
02. Fundamentals of Computer Networks	11	4.8 CAP Theorem	36
2.1 How the Internet Works	12	4.9 Database Indexing - Deep Dive	37
2.2 IP Addresses	13	4.10 SQL vs NoSQL Decision Guide	38
2.3 DNS - Domain Name System	14	05. Caching	39
2.4 HTTP vs HTTPS	15	5.1 What is Caching?	40
2.5 HTTP Methods - Complete Reference	16	5.2 Where to Cache	41
2.6 HTTP Status Codes	17	5.3 Cache Eviction Policies	42
2.7 TCP vs UDP	18	5.4 Cache Strategies	43
2.8 WebSockets vs HTTP	19	5.5 Redis - Complete Guide	44
03. APIs - Application Programming Interfaces	20	5.6 Cache Problems & Solutions	45
3.1 What is an API?	21	5.7 CDN - Content Delivery Network	46
3.2 REST API - Complete Guide	22	06. Load Balancing	47
3.3 REST vs GraphQL vs gRPC	23	6.1 What is Load Balancing?	48
3.4 API Authentication Methods	24	6.2 Load Balancing Algorithms	49
3.5 API Rate Limiting	25	6.3 Layer 4 vs Layer 7 Load Balancing	50
3.6 API Versioning	26	6.4 Health Checks	51
3.7 API Gateway	27	6.5 Popular Load Balancers	52
04. Databases	28		

TIP - Reading path

New learners: Chapters 1-4 first. Then Caching, Load Balancing, Queues, Microservices. Finally, practice the real systems and interview guide.

Table of Contents

Two-page chapter and subsection roadmap with page numbers.

07. Message Queues	53	10.1 Design URL Shortener	77
7.1 What is a Message Queue?	54	10.2 Design Twitter/X	79
7.2 Why Use Message Queues?	55	10.3 Design WhatsApp	81
7.3 Kafka - Complete Guide	56	10.4 Design YouTube/Netflix	83
7.4 RabbitMQ - Guide	57	10.5 Design Uber/Ola	85
7.5 Message Queue Patterns	58	10.6 Design Instagram	87
7.6 When to Use Which	59	11. System Design Interview Guide	89
08. Microservices Architecture	60	11.1 Step-by-Step Interview Framework	90
8.1 Monolith vs Microservices	61	11.2 Numbers Every Engineer Must Know	91
8.2 Monolith vs Microservices - Master Comparison	62	11.3 Top 30 System Design Q&A	92
8.3 Microservices Communication	63	11.4 Red Flags in System Design Interviews	95
8.4 Service Discovery	64	11.5 System Design Template	96
8.5 Circuit Breaker Pattern	65	12. Quick Reference Cheat Sheet	97
8.6 API Gateway in Microservices	66	12.1 Networking Quick Reference	98
8.7 Microservices Design Patterns	67	12.2 Database Selection Guide	99
09. System Design Components	68	12.3 Cache Strategy Selection	100
9.1 Proxy vs Reverse Proxy	69	12.4 Load Balancing Algorithms	101
9.2 Consistent Hashing	70	12.5 Message Queue Use Cases	102
9.3 Bloom Filter	71	12.6 Microservices Patterns	103
9.4 Rate Limiting	72	12.7 Scaling Decision Guide	104
9.5 Long Polling vs WebSockets vs SSE	73	12.8 Numbers Cheat Sheet	105
9.6 Distributed Systems Concepts	74	12.9 Technology Selection Guide	106
9.7 Monitoring & Observability	75	12.10 Interview Checklist	107
10. Designing Real Systems	76	Resources & Links	108

TIP - Reading path

New learners: Chapters 1-4 first. Then Caching, Load Balancing, Queues, Microservices. Finally, practice the real systems and interview guide.

CHAPTER 01

What is System Design?

Complete beginner-friendly notes, diagrams, examples, and interview guidance.

TIP - Chapter goal

By the end of this chapter, you should be able to explain what is system design? clearly, draw the basic architecture, identify trade-offs, and answer interview follow-up questions.

1.1 Introduction to System Design

INFO - Definition

System design defines the architecture, components, interfaces, and data flow required to satisfy a product goal at a given scale.

Introduction to System Design should be understood in terms of four questions: what problem it solves, why it is needed, how it works internally, and when it should be used. In interviews, explain the concept in simple words first, then add trade-offs and production concerns.

REAL WORLD - Real-world analogy

Think of introduction to system design like a well-run city service: it has a clear responsibility, rules for using it, capacity limits, failure procedures, and coordination with other services.

ARCHITECTURE - Introduction to System Design - diagram

```
Client -> Gateway -> Introduction to System Design
      -> Cache/Queue/Database
      -> Monitoring and Alerts
```

This diagram shows the normal flow. In a real production system, every arrow can fail, slow down, or return partial data. Good design handles those cases.

Aspect	Explanation
What it is	System design defines the architecture, components, interfaces, and data flow required to satisfy a product goal at a given scale.
Why needed	It improves scalability, reliability, performance, security, or maintainability depending on the context.
How it works	A request enters the component, the component applies rules or data access logic, then passes the result to the next layer.
When to use	Use it when the problem it solves appears clearly in your requirements or bottleneck analysis.
Trade-off	It adds complexity, cost, operations, and failure modes, so justify it instead of adding it blindly.

TIP - Best practices

Start simple, measure bottlenecks, add this component only when it solves a clear problem, monitor it, and document its failure behavior. Use capacity numbers to defend your choice.

CAUTION - Common mistakes

Jumping to technology names without explaining the problem. Forgetting failure modes. Ignoring latency, cost, security, and operational complexity. Not discussing trade-offs.

INTERVIEW - Interview note

For Introduction to System Design, say what you would choose first, what bottleneck would force you to change, and what trade-off your decision creates.

1.2 Why System Design Matters

INFO - Definition

System design matters because real products must handle growth, traffic spikes, failures, latency, security, and long-term maintenance.

Why System Design Matters should be understood in terms of four questions: what problem it solves, why it is needed, how it works internally, and when it should be used. In interviews, explain the concept in simple words first, then add trade-offs and production concerns.

REAL WORLD - Real-world analogy

Think of why system design matters like a well-run city service: it has a clear responsibility, rules for using it, capacity limits, failure procedures, and coordination with other services.

ARCHITECTURE - Why System Design Matters - diagram

```
Client -> Gateway -> Why System Design Matters
          -> Cache/Queue/Database
          -> Monitoring and Alerts
```

This diagram shows the normal flow. In a real production system, every arrow can fail, slow down, or return partial data. Good design handles those cases.

Aspect	Explanation
What it is	System design matters because real products must handle growth, traffic spikes, failures, latency, security, and long-term maintenance.
Why needed	It improves scalability, reliability, performance, security, or maintainability depending on the context.
How it works	A request enters the component, the component applies rules or data access logic, then passes the result to the next layer.
When to use	Use it when the problem it solves appears clearly in your requirements or bottleneck analysis.
Trade-off	It adds complexity, cost, operations, and failure modes, so justify it instead of adding it blindly.

TIP - Best practices

Start simple, measure bottlenecks, add this component only when it solves a clear problem, monitor it, and document its failure behavior. Use capacity numbers to defend your choice.

CAUTION - Common mistakes

Jumping to technology names without explaining the problem. Forgetting failure modes. Ignoring latency, cost, security, and operational complexity. Not discussing trade-offs.

INTERVIEW - Interview note

For Why System Design Matters, say what you would choose first, what bottleneck would force you to change, and what trade-off your decision creates.

1.3 What Interviewers Expect

INFO - Definition

Interviewers expect you to clarify scope, estimate scale, draw a simple architecture, discuss trade-offs, and improve the design based on bottlenecks.

What Interviewers Expect should be understood in terms of four questions: what problem it solves, why it is needed, how it works internally, and when it should be used. In interviews, explain the concept in simple words first, then add trade-offs and production concerns.

REAL WORLD - Real-world analogy

Think of what interviewers expect like a well-run city service: it has a clear responsibility, rules for using it, capacity limits, failure procedures, and coordination with other services.

ARCHITECTURE - What Interviewers Expect - diagram

```
Client -> Gateway -> What Interviewers Expect
          -> Cache/Queue/Database
          -> Monitoring and Alerts
```

This diagram shows the normal flow. In a real production system, every arrow can fail, slow down, or return partial data. Good design handles those cases.

Aspect	Explanation
What it is	Interviewers expect you to clarify scope, estimate scale, draw a simple architecture, discuss trade-offs, and improve the design based on bottlenecks.
Why needed	It improves scalability, reliability, performance, security, or maintainability depending on the context.
How it works	A request enters the component, the component applies rules or data access logic, then passes the result to the next layer.
When to use	Use it when the problem it solves appears clearly in your requirements or bottleneck analysis.
Trade-off	It adds complexity, cost, operations, and failure modes, so justify it instead of adding it blindly.

TIP - Best practices

Start simple, measure bottlenecks, add this component only when it solves a clear problem, monitor it, and document its failure behavior. Use capacity numbers to defend your choice.

CAUTION - Common mistakes

Jumping to technology names without explaining the problem. Forgetting failure modes. Ignoring latency, cost, security, and operational complexity. Not discussing trade-offs.

INTERVIEW - Interview note

For What Interviewers Expect, say what you would choose first, what bottleneck would force you to change, and what trade-off your decision creates.

1.4 High Level vs Low Level Design

INFO - Definition

High Level Design focuses on architecture and components; Low Level Design focuses on classes, methods, and implementation details.

High Level vs Low Level Design should be understood in terms of four questions: what problem it solves, why it is needed, how it works internally, and when it should be used. In interviews, explain the concept in simple words first, then add trade-offs and production concerns.

REAL WORLD - Real-world analogy

Think of high level vs low level design like a well-run city service: it has a clear responsibility, rules for using it, capacity limits, failure procedures, and coordination with other services.

ARCHITECTURE - High Level vs Low Level Design - diagram

```
Client -> Gateway -> High Level vs Low Level Design
      -> Cache/Queue/Database
      -> Monitoring and Alerts
```

This diagram shows the normal flow. In a real production system, every arrow can fail, slow down, or return partial data. Good design handles those cases.

Feature	High Level Design	Low Level Design
Focus	Architecture	Implementation
Components	Servers, DBs, APIs	Classes, methods
Diagram	Block diagram	Class diagram
Example	Design Twitter	Design Tweet class

TIP - Best practices

Start simple, measure bottlenecks, add this component only when it solves a clear problem, monitor it, and document its failure behavior. Use capacity numbers to defend your choice.

CAUTION - Common mistakes

Jumping to technology names without explaining the problem. Forgetting failure modes. Ignoring latency, cost, security, and operational complexity. Not discussing trade-offs.

INTERVIEW - Interview note

For High Level vs Low Level Design, say what you would choose first, what bottleneck would force you to change, and what trade-off your decision creates.

1.5 How to Approach Any System Design Problem

INFO - Definition

A reliable interview approach is: clarify requirements, estimate capacity, draw high-level design, deep dive, scale, and review trade-offs.

How to Approach Any System Design Problem should be understood in terms of four questions: what problem it solves, why it is needed, how it works internally, and when it should be used. In interviews, explain the concept in simple words first, then add trade-offs and production concerns.

REAL WORLD - Real-world analogy

Think of how to approach any system design problem like a well-run city service: it has a clear responsibility, rules for using it, capacity limits, failure procedures, and coordination with other services.

ARCHITECTURE - How to Approach Any System Design Problem - diagram

```
Client -> Gateway -> How to Approach Any System Design Problem
                -> Cache/Queue/Database
                -> Monitoring and Alerts
```

This diagram shows the normal flow. In a real production system, every arrow can fail, slow down, or return partial data. Good design handles those cases.

Aspect	Explanation
What it is	A reliable interview approach is: clarify requirements, estimate capacity, draw high-level design, deep dive, scale, and review trade-offs.
Why needed	It improves scalability, reliability, performance, security, or maintainability depending on the context.
How it works	A request enters the component, the component applies rules or data access logic, then passes the result to the next layer.
When to use	Use it when the problem it solves appears clearly in your requirements or bottleneck analysis.
Trade-off	It adds complexity, cost, operations, and failure modes, so justify it instead of adding it blindly.

TIP - Best practices

Start simple, measure bottlenecks, add this component only when it solves a clear problem, monitor it, and document its failure behavior. Use capacity numbers to defend your choice.

CAUTION - Common mistakes

Jumping to technology names without explaining the problem. Forgetting failure modes. Ignoring latency, cost, security, and operational complexity. Not discussing trade-offs.

INTERVIEW - Interview note

For How to Approach Any System Design Problem, say what you would choose first, what bottleneck would force you to change, and what trade-off your decision creates.

CHAPTER 02

Fundamentals of Computer Networks

Complete beginner-friendly notes, diagrams, examples, and interview guidance.

TIP - Chapter goal

By the end of this chapter, you should be able to explain fundamentals of computer networks clearly, draw the basic architecture, identify trade-offs, and answer interview follow-up questions.

2.1 How the Internet Works

INFO - Definition

The internet is a network of networks where devices exchange packets using protocols and routers forward packets toward destination servers.

How the Internet Works should be understood in terms of four questions: what problem it solves, why it is needed, how it works internally, and when it should be used. In interviews, explain the concept in simple words first, then add trade-offs and production concerns.

REAL WORLD - Real-world analogy

Think of how the internet works like a well-run city service: it has a clear responsibility, rules for using it, capacity limits, failure procedures, and coordination with other services.

ARCHITECTURE - How the Internet Works - diagram

```
Your Phone -> WiFi Router -> ISP -> Internet Backbone
-> DNS Resolver -> Web Server -> Response Back
```

This diagram shows the normal flow. In a real production system, every arrow can fail, slow down, or return partial data. Good design handles those cases.

Aspect	Explanation
What it is	The internet is a network of networks where devices exchange packets using protocols and routers forward packets toward destination servers.
Why needed	It improves scalability, reliability, performance, security, or maintainability depending on the context.
How it works	A request enters the component, the component applies rules or data access logic, then passes the result to the next layer.
When to use	Use it when the problem it solves appears clearly in your requirements or bottleneck analysis.
Trade-off	It adds complexity, cost, operations, and failure modes, so justify it instead of adding it blindly.

TIP - Best practices

Start simple, measure bottlenecks, add this component only when it solves a clear problem, monitor it, and document its failure behavior. Use capacity numbers to defend your choice.

CAUTION - Common mistakes

Jumping to technology names without explaining the problem. Forgetting failure modes. Ignoring latency, cost, security, and operational complexity. Not discussing trade-offs.

INTERVIEW - Interview note

For How the Internet Works, say what you would choose first, what bottleneck would force you to change, and what trade-off your decision creates.

LearnStack Free Preview

This was a free preview. Get the full book on LearnStack.

Visit: <https://www.learnstack.co.in>

Digital PDF delivery is handled through Gumroad email after purchase.