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.

Table of Contents

Two-page chapter and subsection roadmap with page numbers.

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.

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.