educative

System Design Cheat Sheet Bundle

educative

Solve Any System Design Interview Question

The 8-part **RESHADED** method:

1. Requirements
2. Estimation
3. Storage schema (optional)
4. High-level design
5. APIs
6. Detailed design
7. Evaluation
8. Distinctive component/feature

Building Blocks Glossary:

Domain Name System: *Maps domain names to IP addresses.*

Load Balancers: *Distributes client requests among servers.*

Databases: *Stores, retrieves, modifies, & deletes data.*

Key-Value Store: *Stores data as key-value pairs.*

Content Delivery Network: *Distributes in-demand content to end users.*

Sequencer: *Generates unique IDs for events & database entries.*

Service Monitoring: *Analyzes system for failures & sends alerts.*

Distributed Caching: *Stores frequently accessed data.*

Distributed Messaging Queue: *Decouples messaging producers from consumers.*

Publish-Subscribe System: *Supports asynchronous service-to-service communication.*

Rate Limiter: *Throttles incoming requests for services.*

Blob Store: *Stores unstructured data.*

Distributed Search: *Returns relevant content for user queries.*

Distributed Logging: *Enables services to log events.*

Distributed Task Scheduling: *Allocates resources to tasks.*

Sharded Counters: *Counts concurrent read/write requests.*

Step 1: Requirements

Gather functional & non-functional requirements

Consider:

- *System goals*
- *Key features*
- *System constraints*
- *User expectations*

Step 2: Estimation

Estimate hardware & infrastructure needed to implement at scale

Consider requirements for:

- *Number of servers*
- *Daily storage*
- *Network*

Step 3: Storage schema (optional)*

Articulate data model

Define:

- *Structure of data*
- *Tables to use*
- *Type of fields in tables*
- *Relationship between tables (optional)*

Relevant when you:

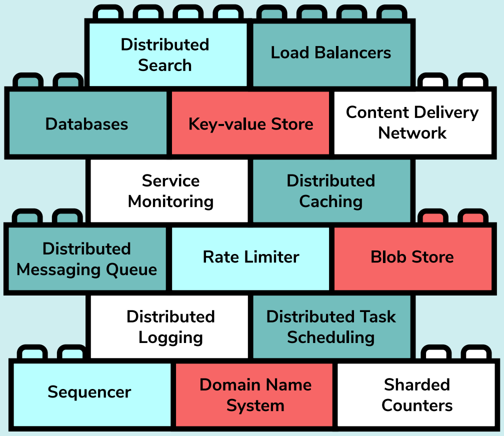
- *Expect highly normalized data*
- *Will store different parts of data in various formats*
- *Face performance & efficiency concerns around storage*

Step 4: High-level design

- Build high-level design
- Choose building blocks to meet functional requirements

For each, identify:

- ***How*** *they work*
- ***Why*** *they're needed*
- ***How*** *they integrate*



This layered visual shows dependencies between building blocks. ***Blocks in lower layers support those above.***

Step 5: APIs

Translate functional requirements into API calls

E.g.:

- ***Requirement:*** *Users should be able to access all items*
- ***API call:*** *GET / items*

Step 6: Detailed design

- *Improve high-level design*
- *Consider all non-functional requirements & complete design*

Step 7: Evaluation

- *Evaluate design against requirements*
- *Explain trade offs & pros/cons of different solutions*
- *Address overlooked design problems*

(8*) Distinctive component/feature

Discuss a distinctive feature that meets requirements

- *E.g. Concurrency control in high-traffic apps*

** Timing varies. Best done after completing design (E.g. Step 6 & 7)*

System Design Interview Cheat Sheet

educative

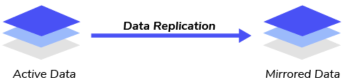
Distributed system fundamentals

Data durability and consistency

The differences and impacts of failure rates of storage solutions and corruption rates in read-write processes

Replication

Backing up data and repeating processes at scale

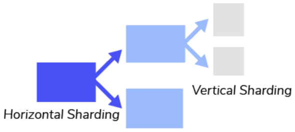


Consensus

Ensuring all nodes are in agreement, which prevents fault processes from running and ensures consistency and replication of data and processes

Partitioning

Dividing data across different nodes within systems, which reduces reliance on pure replication



Distributed transactions

Once consensus is reached, transactions from applications need to be committed across databases with fault checks by each resource involved

Architecture of scalable web applications

HTTP

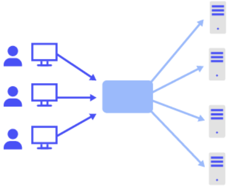
The API on which the entire internet runs

REST

The set of design principles that directly interact with HTTP to enable system efficiency and scalability

DNS and load balancing

Routing client requests to the right servers and the right tiers when processing happens to ensure system stability

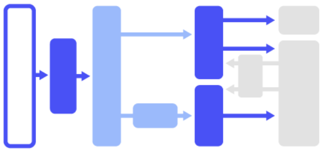


Caching

Making tradeoffs and caching decisions to determine what should be stored in a cache, how to direct traffic to a cache, and how to ensure we have the appropriate data in the cache

N-tier applications

Understanding how processing tiers interact with each other and the specific process they control



Stream processing

Applying uniform processes to data streams to allow for efficient use of local resources

How to design large-scale systems

Step 1: Clarify the goals
Make sure you understand the basic requirements and ask any clarifying questions.

Step 2: Determine the scope
Describe the feature set you'll be discussing in the given solution, and define all of the features and their importance to the end goal.

Step 3: Design for the right scale
Determine the scale so you know whether the data can be supported by a single machine or if you need to scale.

Step 4: Start simple, then iterate
Describe the high-level process end-to-end based on your feature set and overall goals. This is a good time to discuss potential bottlenecks.

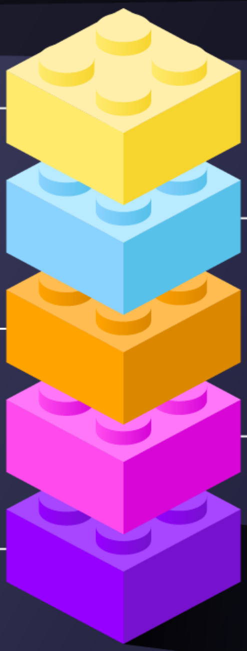
Step 5: Consider relevant DSA
Determine which fundamental data structures and algorithms will help your system perform efficiently and appropriately.

Step 6: Describe trade-offs
Describe trade-offs while explaining your solution to show you understand large-scale systems and their complexities.

*Ask clarifying questions at each step of the process!

5 SYSTEM DESIGN FUNDAMENTALS FOR TECHNICAL PRODUCT MANAGERS

Learn 5 of the most common fundamentals of System Design that you must know to succeed in your role in technical product management.



1. LOAD BALANCER:

Helps TPMs enhance server efficiency and cut down costs

2. KEY-VALUE STORE:

Helps TPMs boost processing power and increase system fault tolerance

3. RATE LIMITER:

Helps TPMs ensure that servers are running optimally and efficiently

4. CDNS:

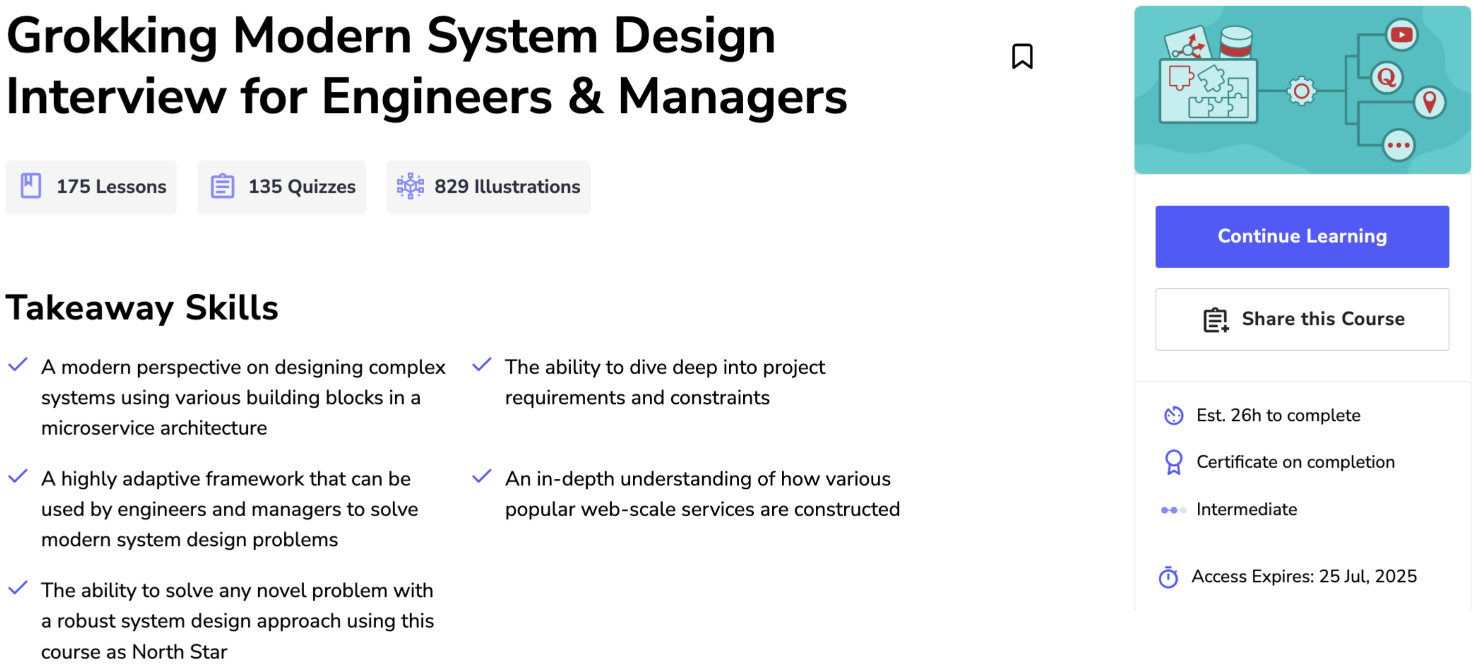
Helps TPMs minimize data loadtimes, reduce redundancy and bandwidth costs

5. DATABASE:

Helps TPMs improve organizational workflow and efficiency

educative

Free Course Lessons from:



Grokking Modern System Design Interview for Engineers & Managers

175 Lessons | 135 Quizzes | 829 Illustrations

Est. 26h to complete

Certificate on completion

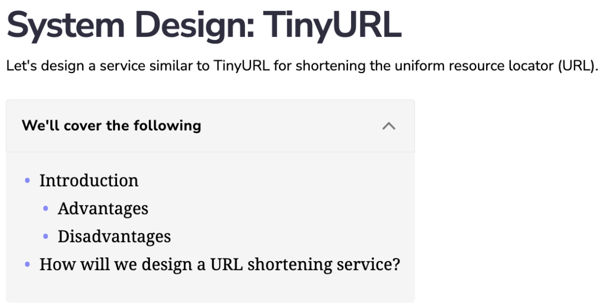
Intermediate

Access Expires: 25 Jul, 2025

Takeaway Skills

- ✓ A modern perspective on designing complex systems using various building blocks in a microservice architecture
- ✓ A highly adaptive framework that can be used by engineers and managers to solve modern system design problems
- ✓ The ability to solve any novel problem with a robust system design approach using this course as North Star
- ✓ The ability to dive deep into project requirements and constraints
- ✓ An in-depth understanding of how various popular web-scale services are constructed

1



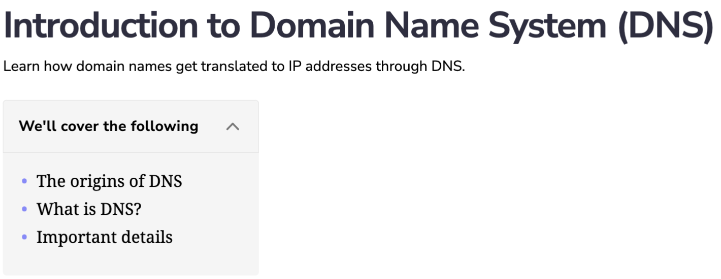
System Design: TinyURL

Let's design a service similar to TinyURL for shortening the uniform resource locator (URL).

We'll cover the following

- Introduction
  - Advantages
  - Disadvantages
- How will we design a URL shortening service?

2



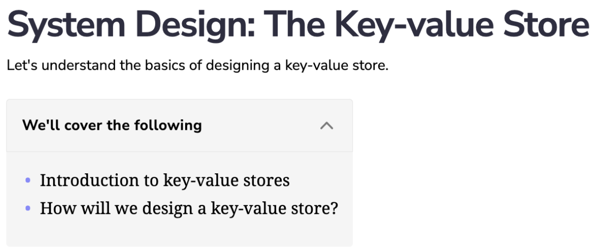
Introduction to Domain Name System (DNS)

Learn how domain names get translated to IP addresses through DNS.

We'll cover the following

- The origins of DNS
- What is DNS?
- Important details

3



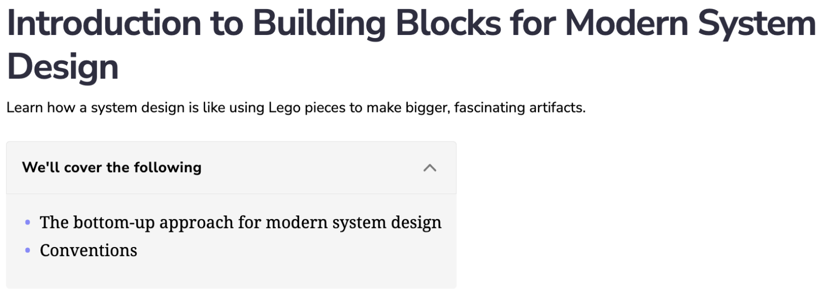
System Design: The Key-value Store

Let's understand the basics of designing a key-value store.

We'll cover the following

- Introduction to key-value stores
- How will we design a key-value store?

4



Introduction to Building Blocks for Modern System Design

Learn how a system design is like using Lego pieces to make bigger, fascinating artifacts.

We'll cover the following

- The bottom-up approach for modern system design
- Conventions