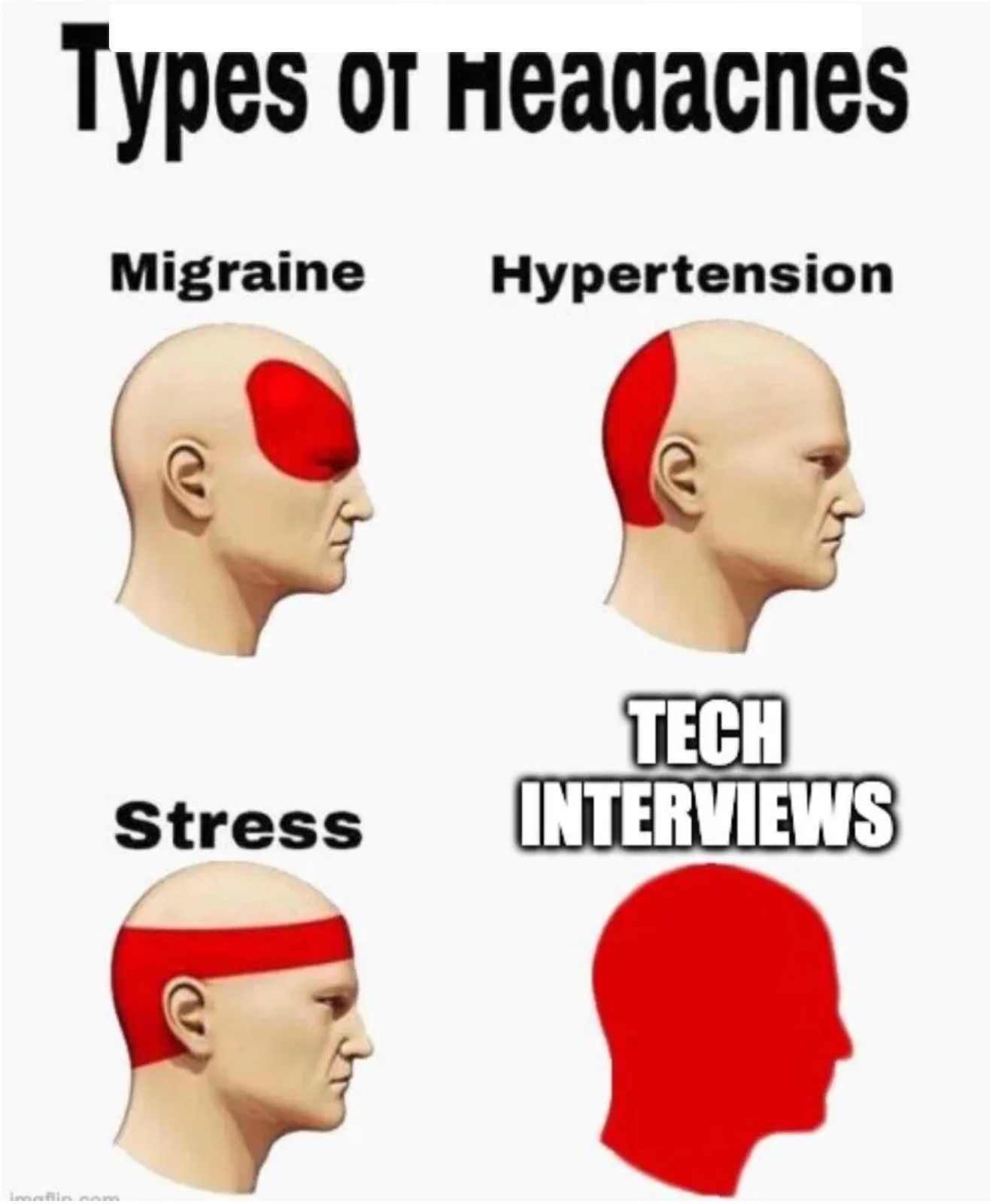
System Design Interview Study Notes I

Yusuf Ali Koyuncu · Follow
10 min read · Dec 12, 2023

Hello, the requirements of the applications we have developed in line with the increasing number of users in recent years and different use cases that we need to consider in the face of this heavy traffic are emerging. In order to handle these situations in the most accurate way, we need to develop our mentality and architectural perspective. A systemic approach is required for a consistent and well-functioning system. System Design is the process of defining elements of a system such as modules, architecture, components and their interfaces and data for a system based on specified requirements. It is the process of identifying, developing and designing systems that meet the specific needs and requirements of a business or organization. In this article, I will share with you the notes I took while working on System Design. So let's start;

How to approach a System Design problem/question?



The questions are usually open-ended and there is more than one solution. At this point, the expectation from you is that you can manage the process.

- Use case, outline constraints and assumptions. Gather the requirements and determine the scope of the problem. Look for answers to the following questions to clarify use cases and constraints.

- “Who will use it?” “How will they use it?” “How many users are there?”, “What does th ’, “How much data do we expect to process?”, “How many requests per second are we waiting for?”, “What is the expected literacy rate?”
- Create a general design. All major components of an overall design should be outlined and drawn. Reflect your ideas on the design here.
- Design core components. Dive into the details for each key component.
- Scale the design. Identify and address bottlenecks given the constraints.

Core Concepts to Scalability

The scalability of an application is measured by the number of successful requests it receives. CPU, RAM, Network, Storage are common targets. There are 2 scaling concepts;

Vertical Scaling (Scaling Up)

It is to add new resources to the system within the necessary needs. An Example, is to increase RAM from 8GB to 16GB. Basically, vertical growth gives you the ability to

expand your existing hardware and software capabilities. It provides this operation on a single

Horizontal Scaling (Scaling Out)

The aim is to share the load on more than one machine. It is more complex than vertical scaling. It is more successful in responding to instant changes. Master-Slave structures are used, and additions and subtractions can be made as needed.

Caching

- It is a high-speed data storage layer that stores a subset of data that is typically temporary. In later requests, instead of going to the 1st warehouse source, the demand is met through the cache and this accelerates the process.
- Usually data is stored on RAM, sometimes a software component is used.
- The main purpose is to reduce access to the 1st repository resource and increase performance.
- It keeps data in a subset and temporary, unlike databases that are complete and durable.
- ROM & In-Memory Engines; improves performance and reduces cost. Doing the same build on traditional databases and disk builds requires additional resources and increases the cost. It doesn't give the same performance.
- Application; Can be applied to various layers. For workload, many platforms use it to avoid the cost of latency. The information received can usually be database queries, intensive computational information, API requests/responses, HTML, JS, image files. Recommendation engines and high performance computing platforms can use In-Memory layer and provide decentralization of this system. But this sometimes causes bottleneck.
- Design Patterns; In a distributed system, the cache layer allows the system and application to operate independently in its lifecycle without the risk of affecting the cache. This acts as a central system where different systems can be accessed. This means that it can be dynamically scaled. (Master-Slave)
- Best practices; TTL can be checked for control. In-Memory solutions such as Redis can be used. If it is to be used as an independent data layer, Redis RTO and RPO must be defined.

- Cache is always in-memory cache like Memcached or Redis. File-based cache should

- The application must first read data from the cache. If not, it should ask the data source. This build adds speed.

- There are 2 patterns of caching data.

1. Data Cached Database Query: Each query to the database is stored in the cache of the data source. The hashes are kept. When the query comes, it first checks the cache. It is difficult to react to changing data in this model. It will be necessary to clear the cache each time.

2. Cached Object: Sees data as objects. Allows the class to create a dataset from the database and then caches the full instance or aggregated dataset of the class. It also enables asynchronous operations.

- It can be useful to use Memcached only if there is data to be cached, because it can scale very well. If you need a main switching, you can use Redis.

Load Balancing

- It is the structure responsible for the distribution of traffic on more than one server.

- It can be applied to systems such as website, application, database and it provides to increase performance and security.

- After meeting the incoming request, it does not matter which server you send it to. Because server/service is duplicate. It does the same.

- It can balance HTTP/HTTPS, TCP, UDP structures.

- It decides in 2 steps. First, the availability of the server is checked. Then it selects the server for distribution according to the rule set.

- Algorithm; Round Robin, Weighted Round Robin, The Least connections, weighted least connections, Source IP hash can be used as needed.

- **Network Load Balancing:** Network load balancing, as the name suggests, leverages network layer information to decide where to send network traffic. This is accomplished through layer 4 load balancing, which is designed to handle all forms of TCP/UDP traffic. Network load balancing is considered the

fastest of all load balancing solutions, but it often falls short when it comes to balanci

— L4: works with special processors and is fast

— L4: increased security

— L4: one time purchase

— L4: professional team (maintenance/repair)

— L4: no scale

— L4: maintenance and repair costs are high

- **HTTP/HTTPS Load Balancing:** HTTP load balancing is one of the oldest forms of load balancing. This form of load balancing is based on layer 7 which means it works at the application layer. HTTP load balancing is often referred to as the most flexible type of load balancing, as it allows you to make distribution decisions based on any information that comes with an HTTP address.

— L7: responsive to change, providing flexibility

— L7: low cost

— L7: enables hybrid systems

— L7: it may take time to scale

— L7: cost of upgrade

- **Internal Load Balancing:** Internal load balancing is almost the same as network load balancing, but can be used to balance internal infrastructure.
- **Hardware Load Balancer:** As the name suggests, a hardware load balancer relies on physical, on-premises hardware to distribute application and network traffic. These devices can handle a large volume of traffic, but often carry a hefty price tag and are quite limited in flexibility.
- **Software Load Balancer:** A software load balancer comes in two forms, commercial or open source, and must be installed before use. Like cloud-based

stabilizers, these tend to be more affordable than hardware solutions.

- **Virtual Load Balancer:** A virtual load balancer is different from software load balancers because it distributes the software of a hardware load balancer to a virtual machine.

Database

- There are two ways;

1. The database must be connected and up and running. A master-slave structure is set up here (master=write/slave=read). We can move forward by adding RAM to the main server. Here are the following concepts: Sharding, denormalization and SQL tuning. This is time and cost.

2. The first is to “denormalize” and switch to the appropriate NoSQL database. After a point, a cache database or cache will be needed.

Database Replication

- **Data Replication:** It is the process of copying the same data from one database to another database.
- **Mirroring:** Creating the primary database on a backup server as a security measure for the server. Mirroring copies the entire database. Not applicable in distributed systems. The main purpose is to backup and it is an expensive operation.
- **Replication:** It is preferred in distributed structures. There is a Master-Slave relationship. It is used to avoid network delays and response times. It is implemented on database objects. It is successfully implemented in distributed systems. Data distribution and efficiency is the goal. It is an inexpensive method.

— Easy access(available)

— Multiple users and high performance

— Balances data between master-slave

— The number of transactions increases

- There are three types;

— **Full Replication:** It is the copying of everything from source to target. More processing [illegible] increases according to the number of rows copied.

— **Key-Based Incremental Replication:** Replication is a method in which it identifies new and updated data using the data column called key. Can understand the deletions in the source.

— **Log-Based Replication:** This is where the changes are saved to make the necessary changes. CRUD operations are performed by looking at this system. It can only be done in supported databases.

- **Replication cons;**

— Reliability Availability

— Disaster Recovery

— Server Performance

— Better network performance

— Enchanted Test System Performance

- **Replication pros**

— High const

— Time constraints

— Network Bandwidth

— Inconsistent Data

Database Partition

- It is the division of the database into independent sections.
- Improves performance in transaction transactions and improves availability, security.
- Partition Criteria;

— Range Partitioning: divides a given range by addressing it

— List Partitioning: splits by grouping

— Composite Partitioning: combines multiple partitions

— Round-Robin Partitioning: provides sequential access to parallel processes

— Hash Partitioning: splits by hashes.

- Horizontal Partition: divides by line
- Splits by Vertical Partition column

Clones

- Every server must be the same and the user must not keep the data locally
- Different tools should be used to deploy the change on the code (Async-Distribute deployment).
- An image of the same server is created and horizontal scaling is applied if needed.

Asynchronism

- There is a paradigm that generally has two ways;

1. Preparing time-consuming work and delivering with low response time. It is used to convert dynamic content to static content.

2. The user wants an intensive and long job. The system takes the job and sends it to queue and tells the user via message that it will notify when it is finished and continue. Queue listening structures receive and finish the transaction. Then it notifies the user that the process is complete.

- Time-consuming work should be done async.

Distributed Architecture Components

- **Distributed Caching:** Provides high data throughput and enables scaling. Generally used to keep application data and web sessions. Increasing throughput is the main goal.
- **Distributed Locking:** It is necessary because we want efficiency or accuracy. Prevents doing the same job more than once. Prevents race condition in simultaneous operations.
- **Distributed Tracking, Tracing, and Measuring:** A method for tracking application requests as they flow from front-end devices to back-end services and databases. Developers can use distributed tracing to troubleshoot requests that show high latency or errors. Some important metrics; User Satisfaction / Apdex Scores, Average Response Time, Error Rates, Count of Application Instances, Request Rate, Application & Server CPU, Application Availability, Garbage Collection, Memory Usage and Throughput.
- **Distributed Scheduling:** It allows us to run different jobs on different machines with a coordinated workflow.

- **Distributed Monitoring and Alerting:** data in distributed systems is collected central [illegible] has multiple definitions;

— **Monitoring:** The collection, processing, aggregation and display of real-time quantitative data about a system, such as query counts and types, error counts and types, transaction times and server lifetimes.

— **White-box Monitoring:** Monitoring based on metrics exposed by internal components of the system, including logs, interfaces such as the Java Virtual Machine Profiling Interface, or an HTTP handler that publishes internal statistics.

— **Black-box Monitoring:** Testing behavior that is visible from the outside in a way that is visible to the user.

— **Dashboard**: An application (usually web-based) that provides a summary view of a service's key metrics. A dashboard can have filters, selectors, etc., but is pre-built to show the metrics that matter most to users. The dashboard can also display team information such as ticket queue length, list of high priority errors, current call engineer for a specific area of responsibility, or recent pushes.

— **Alert**: A notification intended to be read by a human and sent to a system such as a fault or ticket queue, email alias or pager. These alerts are categorized as support notifications, email alerts and pages respectively.

- **Distributed Security:** Whenever someone discusses distributed computing, one of the first questions that comes up is: "Can it be secure?". Authentication and Access Control are important constructs to ensure this security.
- **Distributed Messaging, Queuing, and Event Streaming:** Provides persistence, reliability and scalability.
- **Distributed Logging:** Gathering all the logs in distributed systems in one place to search and extract meaningful results from them.
- **Distributed Searching:** It is the ability to search on multiple machines. It sends the operations to the lower machines and returns the answer from the top machine.

- **Distributed Storage:** It is formed by dividing and keeping data on multiple machir

> It is impossible to design a system so perfect that no one needs to be good. — T. S. Eliot

Reference

- https://github.com/madd86/awesome-system-design
- https://economictimes.indiatimes.com/definition/systems-design
- https://github.com/binhnguyennus/awesome-scalability
- https://github.com/donnemartin/system-design-primer

System Design Interview | Software Engineering | Software Development | Distributed Systems | Systems Thinking

Follow

Written by Yusuf Ali Koyuncu

137 Followers

Back-end Software Engineer - Part-Time Open-Sourcerer. Focuses on Go.

More from Yusuf Ali Koyuncu

Yusuf Ali Koyuncu

Software Architecture Study Notes III

Hello there, in this article I am here with the 3rd part of this series. This is the continuation of Distributed Systems Patterns. You can...

14 min read · Jan 23, 2024

1

Yusuf Ali Koyuncu

Software Architecture Study Notes II

Yusuf Ali Koyuncu

System Design Interview Study Notes II

Hello there, in this article I am here with the 2nd part of this series. You can access the previous part of the series from the link...

10 min read · Dec 21, 2023

Yusuf Ali Koyuncu

Software Architecture Study Notes I

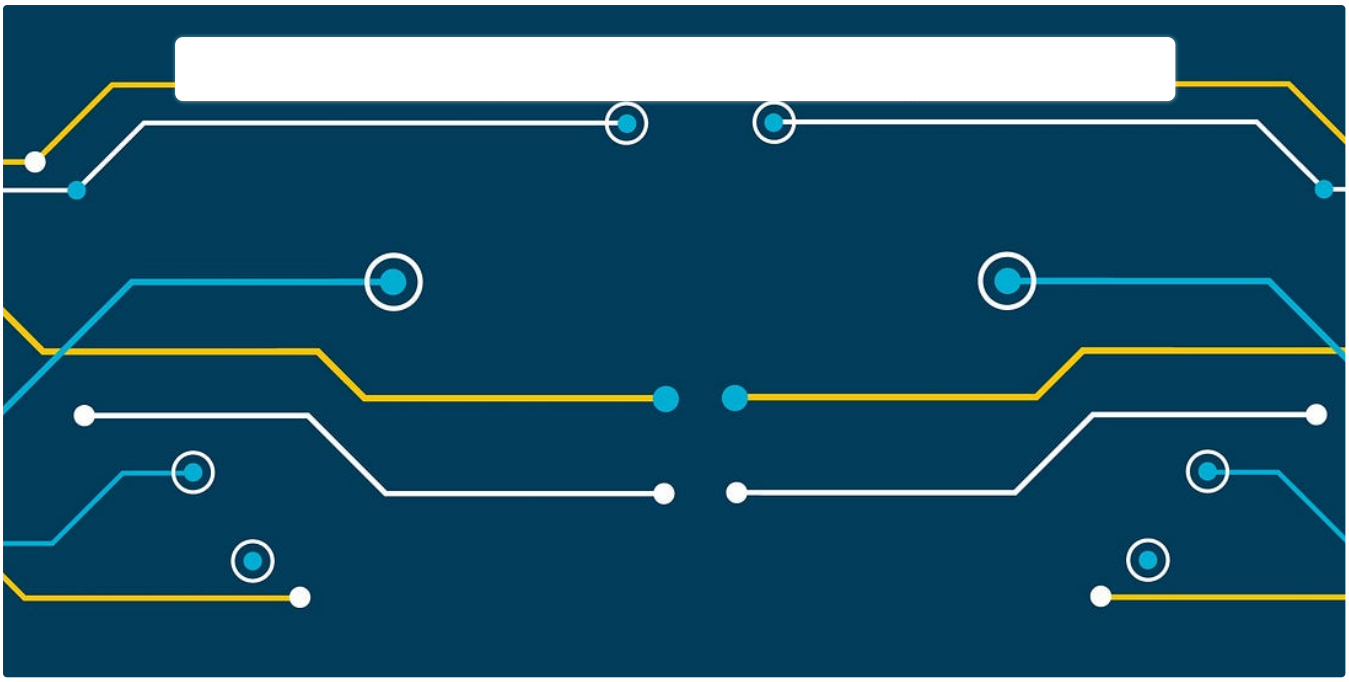
Hello there, in recent years, new software architectures have been developed due to increasing internet usage and new needs. For this...

10 min read · Jan 8, 2024

3

See all from Yusuf Ali Koyuncu

Recommended from Medium



Capital One Tech in Capital One Tech

10 microservices design patterns for better architecture

Consider using these popular design patterns in your next microservices app and make organization more manageable.

11 min read · Jan 10, 2024

1K 4

Chaudhary Vivek Kadiyan

20 Tricky Java Architect Interview Question 2024

I am posting some of the tricky and important interview questions that have been asked to me , related to Ja

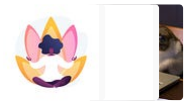
4 min read · Feb 4, 2024

187 5

Lists

General Coding Knowledge

20 stories · 917 saves



Stories to Help You Grow as a Software Developer

19 stories · 810 saves

Leadership

46 stories · 235 saves

Coding & Development

11 stories · 445 saves

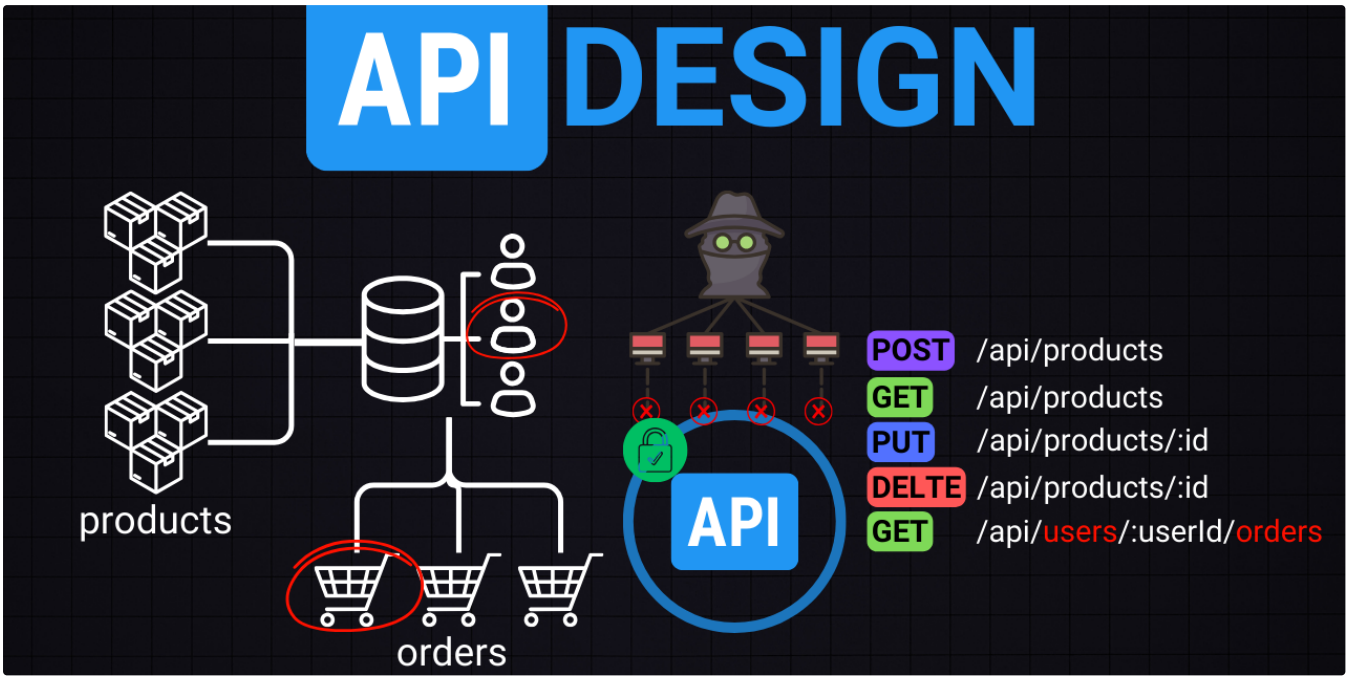
Talha Şahin

High-Level System Architecture of Booking.com

Take an in-depth look at the possible high-level architecture of Booking.com.

· 8 min read · Jan 10, 2024

2.5K



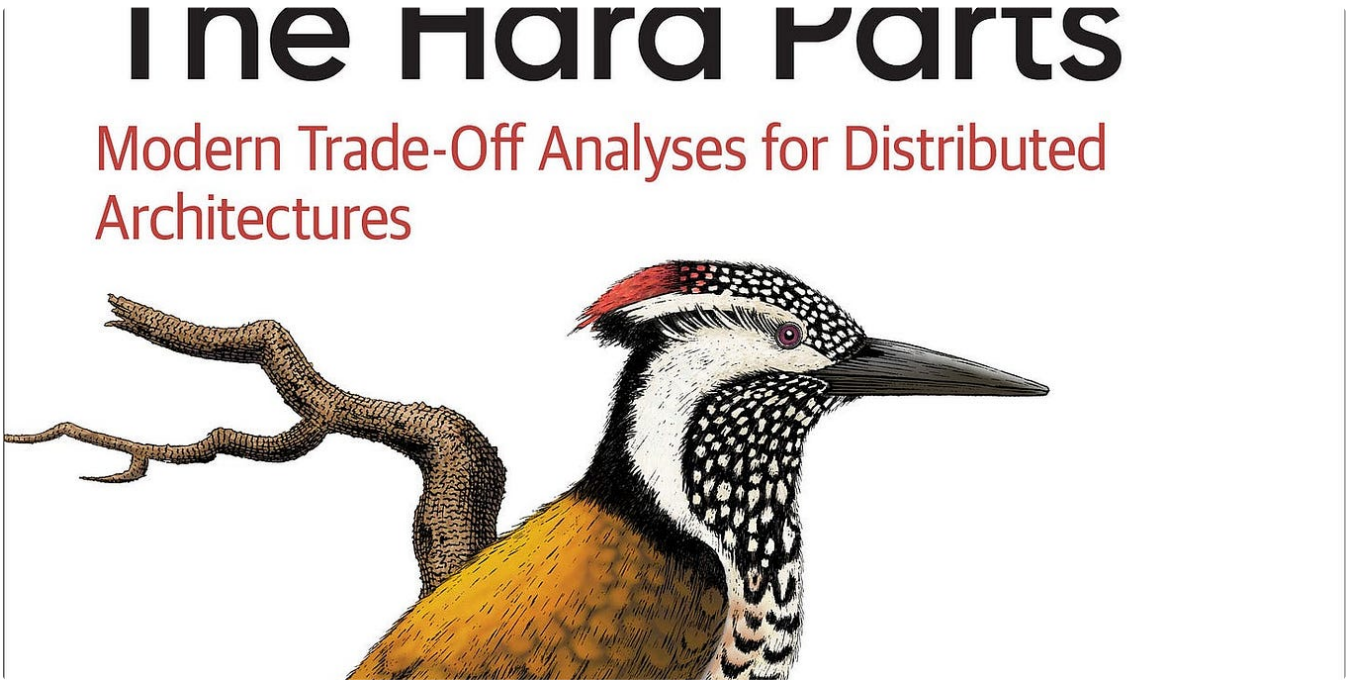
Hayk Simonyan in Level Up Coding

API Design 101: From Basics to Best Practices

API design, from the basics to best practices.

5 min read · Dec 28, 2023

1.4K 9



Dr Milan M

What I learned from the book Software Architecture: The Hard Parts

I recently read the book “ Software Architecture: The Hard Parts” by Neal Ford, Mark, Richards, Pramod Sadalage & Zhamak Dehghani, and this...

8 min read · Feb 8, 2024

544 7

Vishal Barvaliya

Golden Rules for System Design Interview.

System Design is explained in detail with examples.

· 17 min read · Jan 6, 2024

595 2

See more recommendations