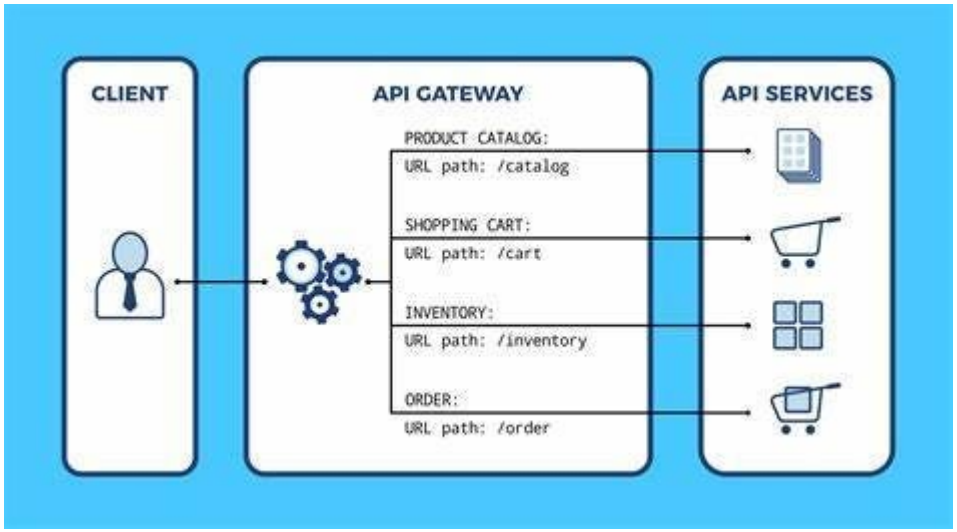
5 Important Microservices Design Patterns

Kishore Karnatakapu · Follow
Published in Javarevisited · 6 min read · May 12

Microservices design patterns have become increasingly popular due to their ability to improve software agility, scalability, resilience, and maintainability. Microservices design patterns are a set of principles and best practices used to develop and maintain software systems that are composed of small, independently deployable services.In this article, we will discuss the important microservice design patterns that can be used to create robust and efficient microservices architectures.


Microservices
Design Patterns

API Gateway:


CLIENT
API GATEWAY
API SERVICES
PRODUCT CATALOG:
URL path: /catalog
SHOPPING CART:
URL path: /cart
INVENTORY:
URL path: /inventory
ORDER:
URL path: /order

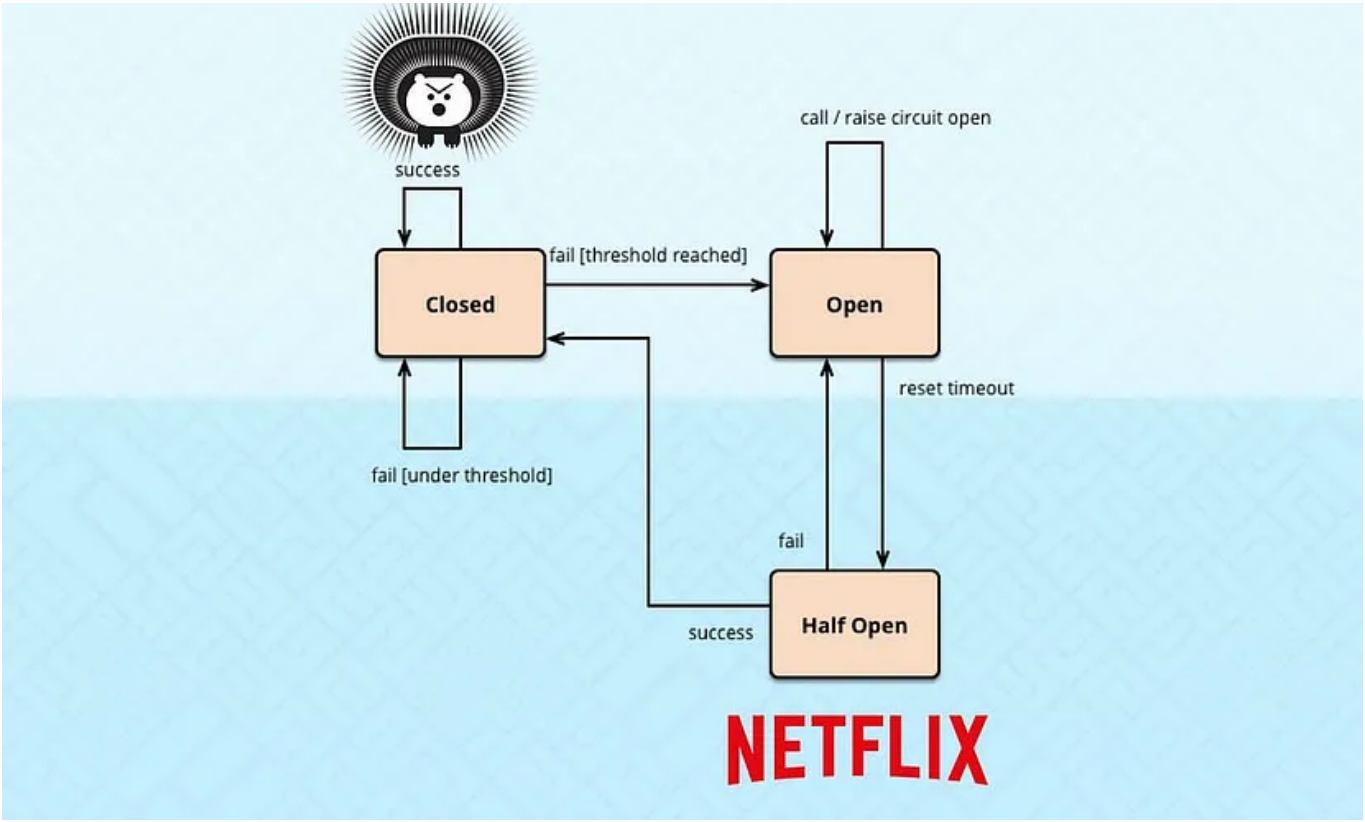
An API Gateway is a service that acts as a front-end for microservices. It receives requests from clients and routes them to the appropriate service.

How API Gateway Pattern works:

The API Gateway pattern works by intercepting requests from clients and routing them to the appropriate service. When a client makes a request to the system, the request is first sent to the API Gateway. The API Gateway then checks if the request is authorized and if so, it routes the request to the appropriate service.

The API Gateway can also perform other functions such as rate limiting, caching, and authentication. For example, it can limit the number of requests that a client can make to a service in a given time frame. It can also cache responses from services to reduce the load on the underlying services.

Circuit Breaker:



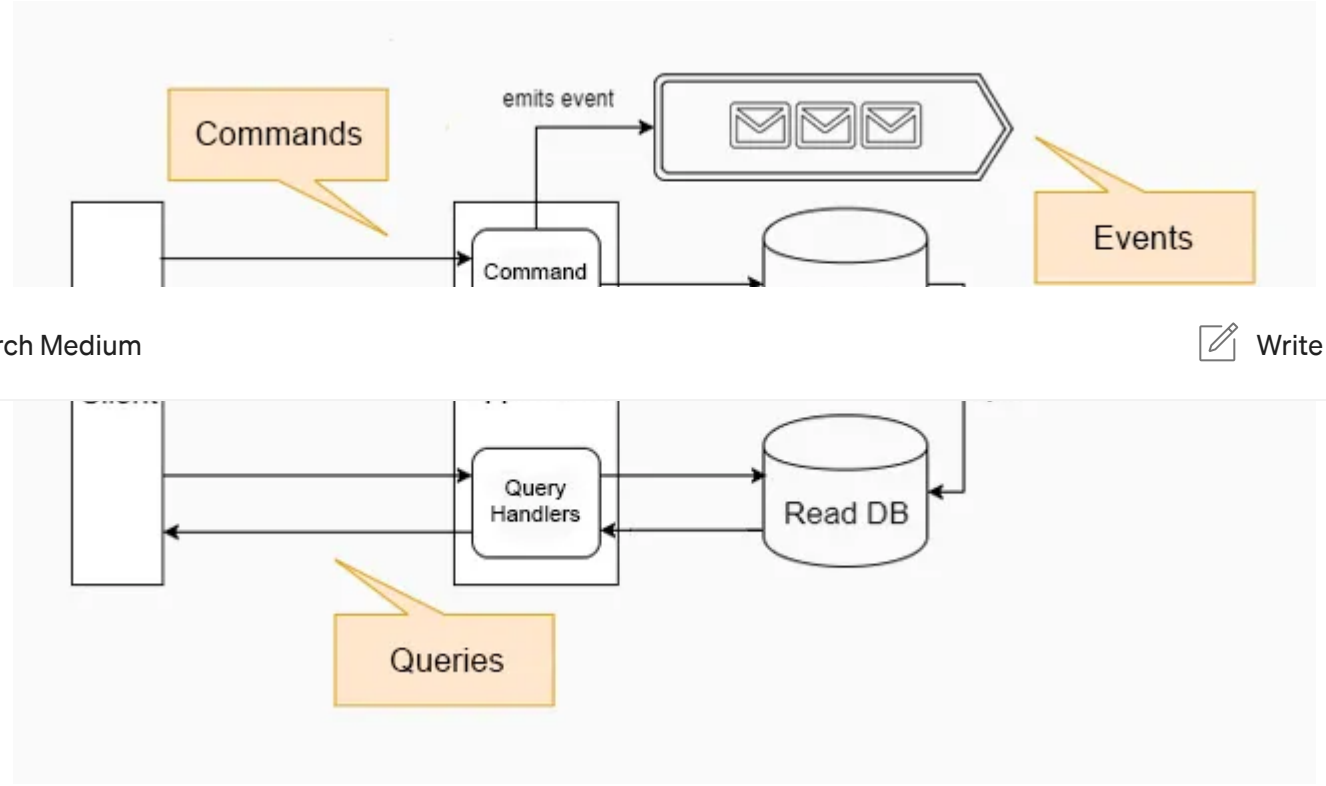
The **Circuit Breaker pattern** is used to detect failures and prevent cascading failures in a distributed system. It operates like an electrical circuit breaker, which trips and breaks the circuit when there is an overload, preventing damage to the electrical equipment. Similarly, the Circuit Breaker pattern can trip and stop the flow of traffic to a failing system.

How Circuit Breaker pattern works:

The Circuit Breaker pattern works by monitoring the health of a system component and taking action when it detects a failure. The Circuit Breaker has three states: closed, open, and half-open.

1. Closed state: In the closed state, the Circuit Breaker allows traffic to flow normally to the system component. The Circuit Breaker monitors the response times and error rates of the system component.
2. Open state: If the Circuit Breaker detects that the system component is failing, it trips and enters the open state. In the open state, the Circuit Breaker stops all traffic to the system component and redirects traffic to a backup component or returns an error message to the client.
3. Half-open state: After a certain amount of time has passed, the Circuit Breaker enters the half-open state. In the half-open state, the Circuit Breaker allows a limited amount of traffic to flow to the system component. If the system component responds successfully to the traffic, the Circuit Breaker returns to the closed state. If the system component fails to respond successfully, the Circuit Breaker returns to the open state.

CQRS (Command Query Responsibility Segregation):



In a traditional system, there is a single model that handles both read and write operations. This model is responsible for maintaining the state of the system, processing commands, and returning data to the client. However, as the system grows in complexity and size, it becomes difficult to scale and maintain.

In order to overcome the above issue, CQRS pattern separates the read and write operations into two different models: the Command model and the Query model. The Command model is responsible for handling write

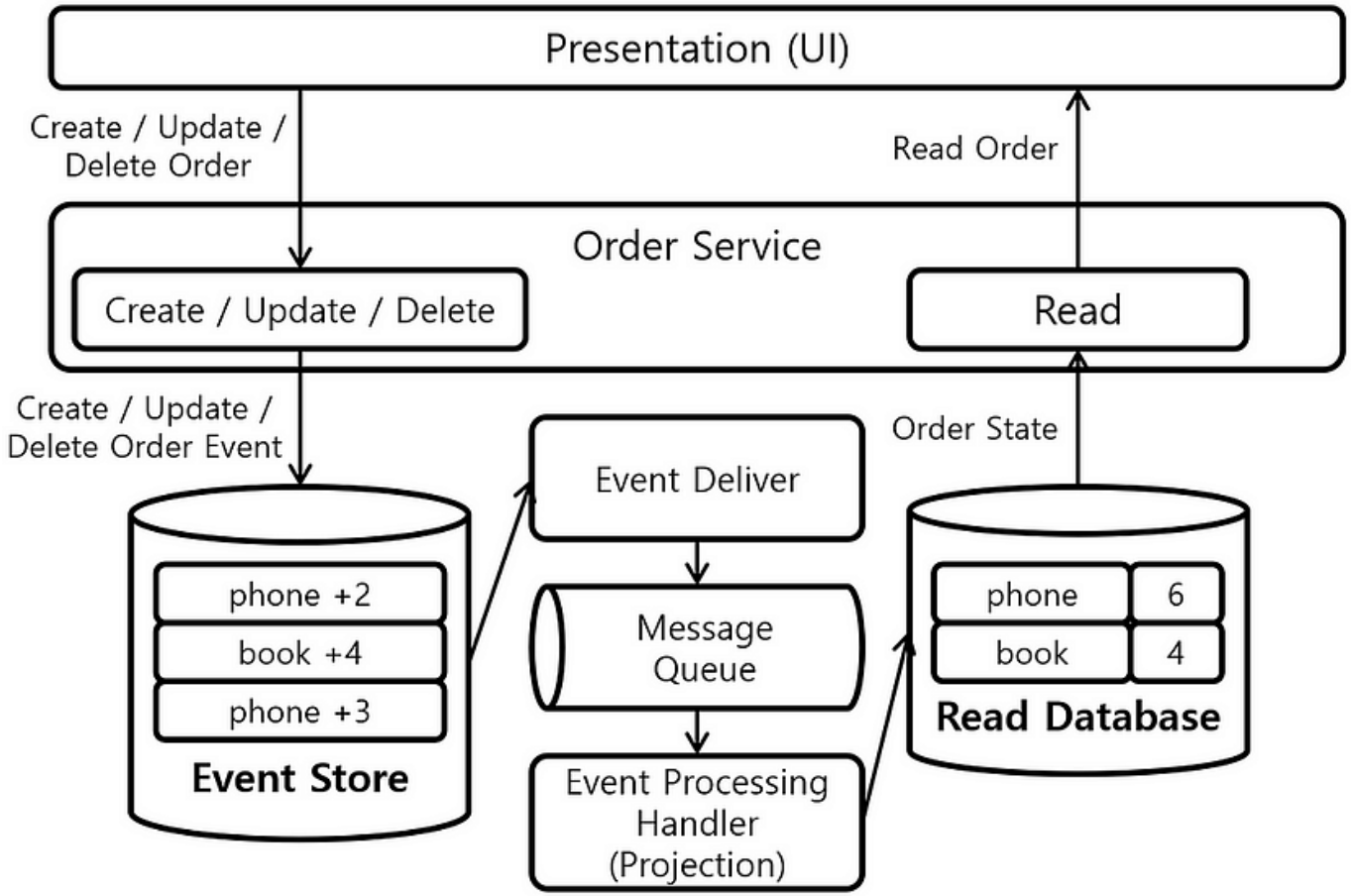
operations and updating the state of the system. The Query model is responsible for handling read operations and returning data to the client.

How CQRS pattern works:

When a client sends a command to the system, it is handled by the Command model. The Command model processes the command and updates the state of the system. If the command is successful, it returns a success message to the client.

When a client sends a query to the system, it is handled by the Query model. The Query model retrieves the data from the system and returns it to the client. The Query model can be optimized for read operations, which can improve the performance of the system.

Event Sourcing



Event sourcing is a pattern used in software design that involves modeling the state of an application as a sequence of events. Instead of storing the current state of the application, the system stores a sequence of events.

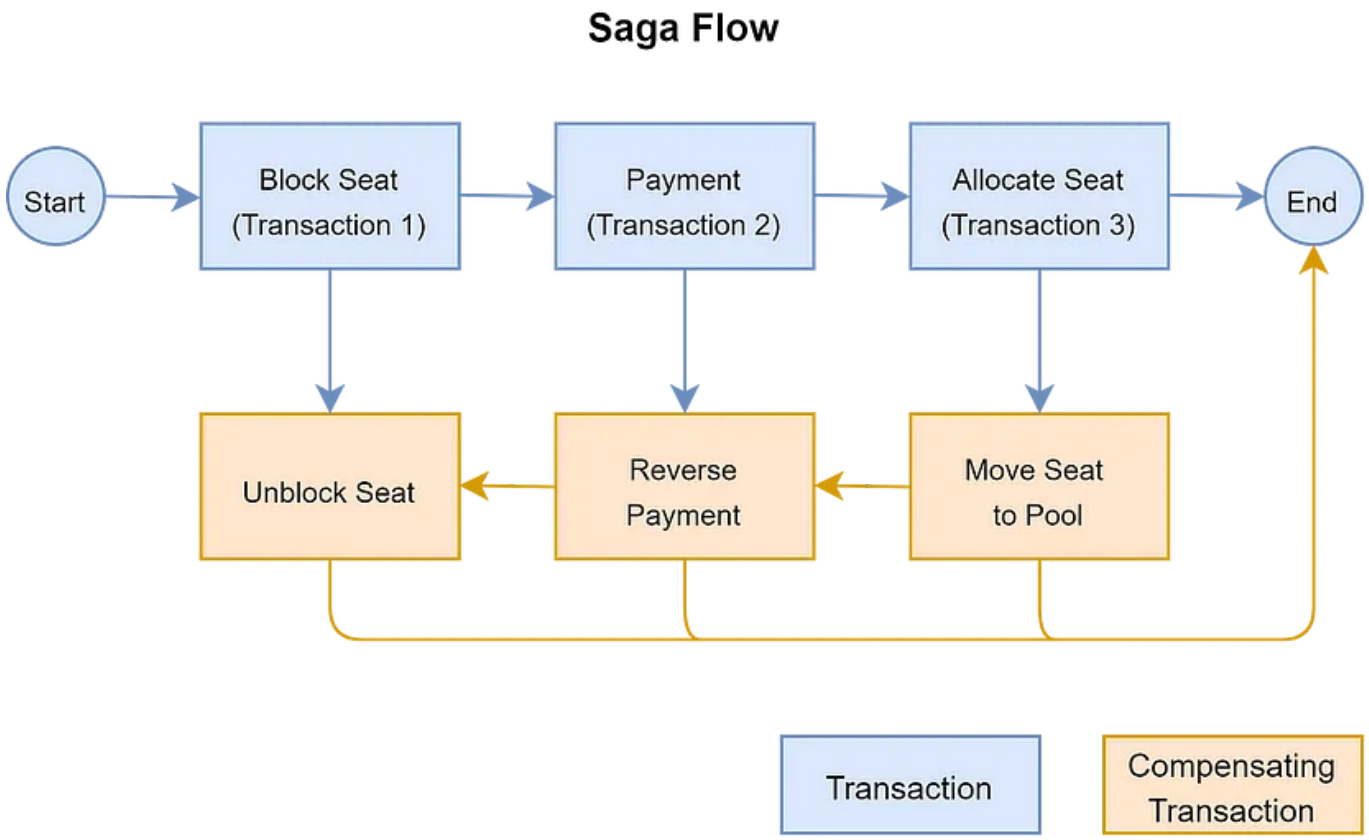
How event sourcing works:

The event sourcing pattern works by storing a sequence of events that describe changes to the state of an application. Each event is stored as a separate record in an event store, which is a database that is optimized for storing large numbers of events.

When a user interacts with the system, the system generates an event that describes the user's action. For example, if a user places an order, the system would generate an event that describes the order details. The event would be stored in the event store, and the system would update the state of the application based on the event.

When the state of the application needs to be queried, the system reads all of the events in the event store and applies them in sequence to reconstruct the current state of the application. This process is known as event replay.

Saga Pattern



Saga design pattern is a technique used in distributed systems to maintain consistency across multiple transactions involving multiple services. It is used to manage distributed transactions across multiple services in a way that ensures data consistency.

In a distributed system, it's common for a single transaction to involve multiple services. For example, in an e-commerce system, a single order may require updates to the inventory service, payment service, and shipping

service. If any of these services fail, it can lead to inconsistencies in the data. The Saga pattern helps to ensure that the transaction is completed successfully or rolled back if any errors occur.

The Saga pattern works by breaking a single transaction into multiple smaller transactions, also known as “compensating transactions.” These compensating transactions are executed in a particular order and are designed to undo the effects of the previous transaction if any errors occur. The order of the compensating transactions is crucial to ensure that the system remains consistent.

Conclusion

While microservices design patterns offer many benefits, they also introduce new complexities, such as service orchestration, service-to-service communication, and distributed systems management. Therefore, it’s important to carefully consider the design patterns that are appropriate for a given application, taking into account the business requirements, technical constraints, and development team’s expertise.

Overall with microservices design patterns, organizations can build software systems that are more flexible, scalable, and resilient.

> *As an Indian writer, montenization is not available for the content I publish, If you like the content and want to support me more, you can buy me a coffee here: https://bmc.link/kishorek2511 or https://www.patreon.com/Kishore834 or https://ko-fi.com/kishorek2511*

Refer links for other Spring Boot Concepts:

Spring Security roled based access with Spring Boot

Spring Security Authenctication & Authorization with JWT

Spring AOP tutorial

Spring Boot Logging Tutorial

Centralized Configuration in Spring Boot

Custom Validations in Spring Boot

javinpaul Domenico Nicoli Trey Huffine Mehmet Ozkaya
The Educative Team

Technology Software Development Programming Software Engineering
Coding

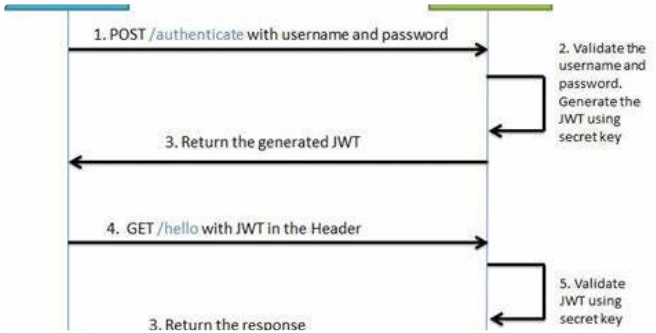
Written by Kishore Karnatakapu

Follow

224 Followers · Writer for Javarevisited

Software Engineer | Java | Spring Boot| Microservices

More from Kishore Karnatakapu and Javarevisited



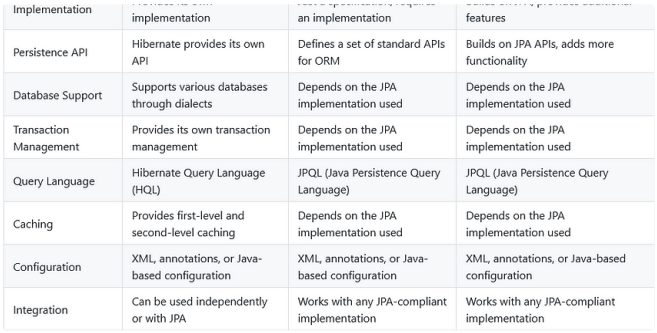
Kishore Karnatakapu in Javarevisited

How to secure Spring Boot with JWT Authentication and...

Hello learners, here we are going to learn about spring security implementation with...

4 min read · Apr 19

96 6



Implementation		an implementation	features
Persistence API	Hibernate provides its own API	Defines a set of standard APIs for ORM	Builds on JPA APIs, adds more functionality
Database Support	Supports various databases through dialects	Depends on the JPA implementation used	Depends on the JPA implementation used
Transaction Management	Provides its own transaction management	Depends on the JPA implementation used	Depends on the JPA implementation used
Query Language	Hibernate Query Language (HQL)	JPQL (Java Persistence Query Language)	JPQL (Java Persistence Query Language)
Caching	Provides first-level and second-level caching	Depends on the JPA implementation used	Depends on the JPA implementation used
Configuration	XML, annotations, or Java-based configuration	XML, annotations, or Java-based configuration	XML, annotations, or Java-based configuration
Integration	Can be used independently or with JPA	Works with any JPA-compliant implementation	Works with any JPA-compliant implementation

Soma in Javarevisited

Difference between Hibernate, JPA, and Spring Data JPA?

Hello folks, if you are preparing for Java Developer interviews then part from...

✨ · 10 min read · May 26

159 1

Messaging Model	Traditional	Publish/Subscribe	Traditional
Scalability	Clustering/Network of Brokers	Partitioning	Clustering/Network of Brokers
Performance	Moderate	High	High
Data Persistence	On Disk (default), In-memory	On Disk	On Disk (default), Database
Integration	Programming Languages, Databases, Web Servers	Data Processing Systems, Databases, Data Sources	JMS Clients, Apache Camel, Apache CXF
Suitable For	Strict Ordering, Reliable Delivery, Moderate-High	Streaming Data, High	High Data Durability,

Soma in Javarevisited

Difference between RabbitMQ, Apache Kafka, and ActiveMQ

Hello folks, if you are preparing for Java Developer interviews the along with Spring...

· 8 min read · May 19

204 3

Kishore Karnatakapu in Level Up Coding

How to call Spring Boot Rest API's Concurrently

How to call Spring REST APIs concurrently using Java CompletableFuture.

5 min read · May 11

102

See all from Kishore Karnatakapu

See all from Javarevisited

Recommended from Medium

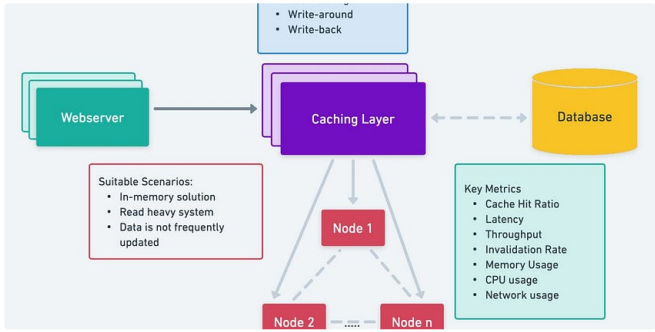
John Raines

Be an Engineer, not a Frameworker

Time to level up.

✦ · 10 min read · Mar 7, 2022

2.1K 31



Love Sharma in Dev Genius

A Comprehensive Guide to Distributed Caching

An essential website requires a web server to receive requests and a database to write or...

✦ · 13 min read · Feb 8

369 3

Lists

General Coding Knowledge

20 stories · 1 save

Stories to Help You Grow as a Software Developer

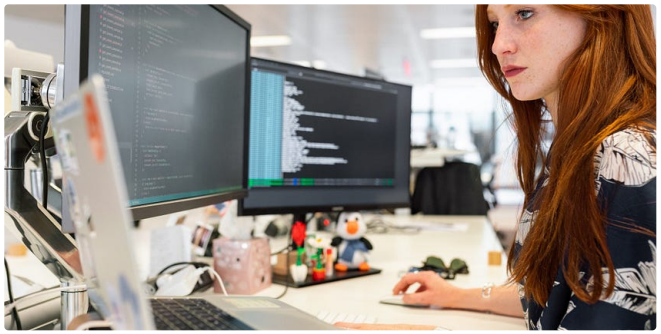
19 stories · 138 saves

It's never too late or early to start something

10 stories · 4 saves

Coding & Development

11 stories



The Coding Diaries in The Coding Diaries

Why Experienced Programmers Fail Coding Interviews

A friend of mine recently joined a FAANG company as an engineering manager, and...

· 5 min read · Nov 2, 2022

4K 86

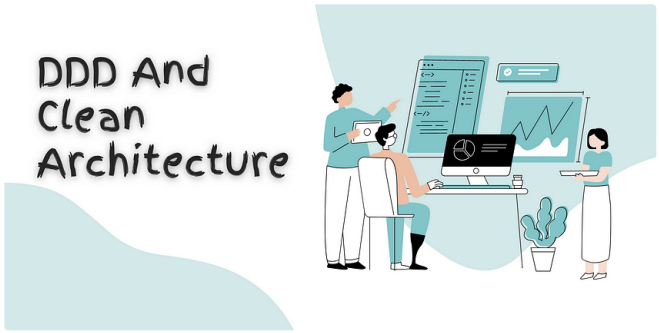
Edward Huang in Better Programming

How To Update Your Status During Standup Like a Senior Engineer

A status update is where you can showcase how well you manage ambiguity and is an...

· 9 min read · Oct 20, 2022

2.5K 14



Kunal Nalawade in Level Up Coding

5 Design Patterns in Java that Solve Major Problems!

In a previous post, I showed you how the OOP world looks like. OOP is a pretty convenient...

11 min read · Jan 19

293 6

Hassan Ibrahim

DDD and Clean Architecture—part 1

Clean Architecture (CA) is the system architecture guideline proposed by Robert C...

7 min read · Feb 7

463 6

See more recommendations