System Design / Architecture Questions for 5 Years Java Developer

This document lists practical, medium-scale system design questions commonly asked for ~5 years of Java experience. Use them to practice clarifying requirements, choosing storage, API design, caching, messaging, scalability, and fault tolerance.

# API / Service Design

* Design a Library Management System (add/remove books, borrow/return, track availability).
* Design a Ticket Booking System (movie/train) with seat selection, cancellations, and payment.
* Design a Food Delivery App (browse restaurants, place order, track delivery).
* Design an Online Banking System (transfers, balance, transaction history).
* Design a URL Shortener (short code, redirect, analytics).
* Design a Simple Blog Platform (posts, comments, likes, feeds).

# Messaging / Event-driven

* Design a Centralized Logging System (collect, store, query logs).
* Design a Metrics Monitoring System (collect metrics, dashboards, alerts).
* Design an Email/SMS/Push Notification Service with retries and DLQ.
* Design an Audit Trail Service (immutable event log for compliance).

# Caching / Performance

* Design a Caching Layer for an E-commerce Website (product/catalog cache).
* Design a Distributed Rate Limiter for an API Gateway.
* Design a Search Feature with Filters and Autocomplete (e.g., product search).
* Design a Feature Flag Service (progressive rollout, kill switches).

# File / Media Systems

* Design a File Upload Service (upload/download/share, chunking, resume).
* Design a Video Streaming Service (store, transcode, stream via CDN).
* Design an Image Processing Pipeline (thumbnails, transformations, CDN).

# Real-time Applications

* Design a Real-time Chat Application (DMs, groups, presence, read receipts).
* Design a Ride-hailing System (matching, real-time tracking, pricing).
* Design a Live Notifications/Feed System (fan-out, personalization).
* Design a Multiplayer Game Leaderboard (real-time ranks, anti-cheat).

# General Distributed Systems

* Design a Distributed ID Generator (e.g., Snowflake-style unique IDs).
* Design a Simple Workflow Orchestrator (DAG, retries, scheduling).
* Design a Distributed Job Scheduler (no duplicate runs, HA, leader election).
* Design a Configuration Service / Spring Cloud Config replacement (versioned configs, rollout).

# Data & Consistency

* Design an Event Sourcing + CQRS-based Order System (write/read separation).
* Design a Data Sync Pipeline from OLTP DB to Analytics Store (CDC → Kafka → Data Lake/Warehouse).
* Design an Inventory Management System with Eventually Consistent Stock Levels.
* Design an Idempotent Payments API (idempotency keys, retries, exactly-once semantics best-effort).

# Appendix: How to answer in interviews (quick checklist)

1. Clarify functional & non-functional requirements; estimate rough scale (RPS, data size).
2. Sketch a high-level architecture: services, databases, cache, message queues, CDN.
3. Define data model (entities + key indexes); choose SQL vs NoSQL and justify.
4. Plan API endpoints/contracts and versioning; consider idempotency.
5. Discuss consistency model (strong vs eventual), transactions (Saga/Outbox), and concurrency control.
6. Address reliability: retries, circuit breaker, DLQ, timeouts, bulkheads, rate limiting.
7. Plan scalability: stateless services, horizontal scaling, partitioning/sharding, read replicas.
8. Add observability: logs, metrics, traces, dashboards, and alerting.
9. Cover security: authN/Z (JWT/OAuth2), input validation, secrets management, mTLS.
10. Wrap up with trade-offs and potential future improvements.