**System Design & Architecture (Microservices, Distributed Systems)**

**1. How would you design a URL shortener like Bit.ly?**

✅ **Answer:**

* Use **Base62 encoding** to generate short unique IDs.
* **Database partitioning** to handle large-scale requests.
* Implement **caching (Redis)** for frequently accessed URLs.
* **Rate limiting** to prevent abuse.
* **Asynchronous processing** for analytics and logging.

**2. How do you handle inter-service communication in microservices?**

✅ **Answer:**

* **Synchronous:** REST APIs (simple, but can cause latency).
* **Asynchronous:** Kafka, RabbitMQ, or gRPC (better scalability, avoids blocking).
* **Service discovery:** Use **Eureka, Consul, or Kubernetes Service** for dynamic discovery.
* **API Gateway:** Centralized request routing and security (e.g., Kong, Apigee).

**3. How would you design a real-time notification system for a banking app?**

✅ **Answer:**

* **Push notifications (Firebase, APNs) for mobile** and **WebSockets for real-time updates.**
* **Event-driven architecture** (Kafka, RabbitMQ).
* **User preferences management** (email, SMS, push).
* **Rate limiting** to avoid spam.

**4. What strategies can be used to ensure high availability in microservices?**

✅ **Answer:**

* **Auto-scaling** (Kubernetes Horizontal Pod Autoscaler).
* **Load balancing** (Nginx, HAProxy).
* **Database replication** (Read replicas, Multi-region DB).
* **Circuit Breaker pattern** (Resilience4j, Hystrix).
* **Failover mechanisms** (Backup services, retry policies).

**5. How do you manage data consistency in microservices?**

✅ **Answer:**

* **Eventual consistency** (use event-driven architecture with Kafka).
* **Distributed transactions** using **Saga pattern**.
* **Idempotency** to handle duplicate requests.
* **Optimistic locking** to prevent race conditions.

**Performance & Optimization**

**6. How would you optimize a slow-running REST API?**

✅ **Answer:**

* Use **caching (Redis, CDN, HTTP Cache-Control)**.
* **Optimize database queries** (Indexing, Query tuning, Avoid N+1 problem).
* Implement **pagination** for large datasets.
* **Asynchronous processing** for long-running tasks.
* Use **Gzip compression** for smaller payloads.

**7. How do you identify memory leaks in a Java application?**

✅ **Answer:**

* Use **heap dump analysis** (jmap, VisualVM, YourKit).
* Check for **static collections holding references**.
* Monitor **Garbage Collection (GC logs, -XX:+PrintGCDetails)**.
* Use **WeakReference & SoftReference** for caching.

**8. What are the best practices for database indexing?**

✅ **Answer:**

* Index **frequently queried columns**.
* Avoid **too many indexes** (affects insert/update speed).
* Use **Composite Indexes** for multiple search conditions.
* **Clustered Index** for primary key performance.
* **Covering Index** for queries that retrieve specific columns.

**9. How do you handle high traffic spikes in a web application?**

✅ **Answer:**

* Implement **CDN caching** for static content.
* Use **load balancers** (Nginx, HAProxy).
* Enable **horizontal scaling** (Kubernetes, AWS Auto Scaling).
* Optimize **database queries & use caching** (Redis).
* Implement **rate limiting** to prevent abuse.

**10. What is the CAP theorem, and how does it apply to distributed databases?**

✅ **Answer:**

* **Consistency (C):** Every read gets the latest write.
* **Availability (A):** Every request gets a response (not necessarily the latest data).
* **Partition Tolerance (P):** System functions even if network fails.
* **Trade-off:** In distributed systems, you can **only guarantee 2 out of 3 (CA, CP, or AP).**
* **Example:**
  + **CP:** MongoDB (Consistent, Partition-tolerant).
  + **AP:** Cassandra (Available, Partition-tolerant).
  + **CA:** Relational databases (Consistent, Available, but no partition tolerance).

**Security & API Protection**

**11. How do you prevent SQL injection in Java applications?**

✅ **Answer:**

* Use **Prepared Statements & Parameterized Queries**.
* **Input validation** to reject malicious inputs.
* **ORM frameworks** (Hibernate, JPA) to prevent direct query execution.

**12. What are the best practices for securing REST APIs?**

✅ **Answer:**

* Use **JWT for authentication**.
* Implement **OAuth2** for third-party access.
* **Enable HTTPS** to encrypt communication.
* **Rate limit requests** to prevent abuse.
* **Validate input data** to avoid injections.

**13. How do you prevent Cross-Site Scripting (XSS) attacks?**

✅ **Answer:**

* **Sanitize user inputs** before rendering on a webpage.
* Use **Content Security Policy (CSP)** to block unauthorized scripts.
* Encode HTML responses (<script> → &lt;script&gt;).

**14. How do you prevent Cross-Site Request Forgery (CSRF)?**

✅ **Answer:**

* Use **CSRF tokens** in forms.
* Implement **SameSite cookies**.
* Validate **Origin and Referer headers**.

**15. How do you implement rate limiting for APIs?**

✅ **Answer:**

* Use **API Gateway throttling (Kong, Apigee)**.
* Implement **Token Bucket or Leaky Bucket algorithms**.
* Track requests per **IP/user**.

**Database & Caching**

**16. How do you design a caching strategy for a web application?**

✅ **Answer:**

* **In-memory cache** (Redis, Memcached) for frequent lookups.
* **Cache expiration & eviction policies** (LRU, LFU).
* **Use write-through or write-back caching** for DB updates.

**17. When should you use NoSQL over SQL?**

✅ **Answer:**

* **NoSQL (MongoDB, Cassandra)**:
  + Unstructured/semi-structured data.
  + High scalability.
  + Flexible schema.
* **SQL (PostgreSQL, MySQL)**:
  + Strong ACID transactions.
  + Structured data.
  + Relational dependencies.

**Messaging & Event-Driven Systems**

**18. What are the advantages of using Kafka in microservices?**

✅ **Answer:**

* **Asynchronous processing** (reduces API response time).
* **Event-driven architecture** (decouples services).
* **Scalability & fault tolerance** (partitioning, replication).

**19. What are Dead Letter Queues (DLQ) in Kafka?**

✅ **Answer:**

* **Stores messages that fail processing after multiple retries.**
* Prevents blocking the main queue.
* Helps in debugging failed messages.

**Other Scenarios**

**20. How do you monitor a Spring Boot application in production?**

✅ **Answer:**

* **Spring Boot Actuator (Health, Metrics, Logs)**.
* **Prometheus + Grafana** for visualization.
* **Distributed tracing (Zipkin, OpenTelemetry)**.

**21-25:**

* **Explain the difference between synchronous and asynchronous processing.**
* **How do you implement Circuit Breaker in microservices?**
* **What is the difference between Monolithic vs. Microservices deployment?**
* **How do you prevent data loss in event-driven architectures?**
* **What are the trade-offs of eventual vs. strong consistency?**