System Design Basics

---

1. What is System Design?

System design is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. It involves both **high-level architecture (HLD)** and **low-level design (LLD)**.

---

2. Types of System Design

- **High-Level Design (HLD):**
  Focuses on architecture, technologies, databases, and services.
- **Low-Level Design (LLD):**
  Focuses on classes, functions, algorithms, and relationships between components.

---

3. Key Components in System Design

- **Load Balancer**
- **Web Server**
- **Application Server**
- **Database (SQL/NoSQL)**
- **Cache (Redis, Memcached)**
- **Message Queue (Kafka, RabbitMQ)**
- **CDN (Content Delivery Network)**
- **Data Warehouse / Analytics Engine**

---

4. Important Concepts

Scalability

- **Vertical Scaling:** Add more power (CPU, RAM) to a single machine.
- **Horizontal Scaling:** Add more machines to distribute the load.

Latency vs Throughput

- **Latency:** Time to process a single request.

- **Throughput:** Number of requests processed per unit time.

Caching

- Reduces load on the backend/database.
- Types: Client-side, CDN, Application-level, Database caching.

Database Sharding

- Splits database into smaller, faster, more manageable parts called shards.

---

5. CAP Theorem

In any distributed system, you can only achieve two out of the following three:

- **Consistency** – Every read receives the most recent write.
- **Availability** – Every request receives a response.
- **Partition Tolerance** – System continues to operate despite network failures.

---

6. Design Principles

- **KISS (Keep It Simple, Stupid)**
- **YAGNI (You Aren’t Gonna Need It)**
- **DRY (Don’t Repeat Yourself)**
- **SOLID (for software design)**

---

7. Example: Designing a URL Shortener (e.g., Bit.ly)

Functional Requirements:

- Shorten a long URL
- Redirect to original URL
- Track number of clicks

Non-functional Requirements:

- High availability
- Low latency
- Analytics support

Components:

- REST API server
- Hashing service for URLs
- Database to store mappings
- Cache layer for fast lookups

---

8. Trade-offs to Consider

- **Consistency vs Availability**
- **Latency vs Accuracy**
- **Read-heavy vs Write-heavy optimization**
- **Monolithic vs Microservices**