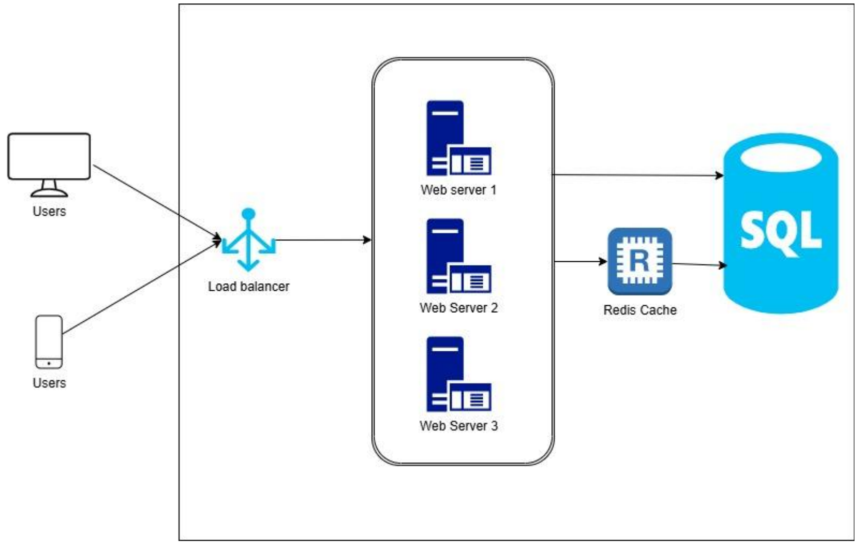
Mastering System Design

From Basics to Cracking Interviews

Rahul Rajat Singh

What is System Design?

- System Design is the process of defining the architecture, components, modules, interfaces, and data flow for a system to meet specific requirements.
- It involves making high-level decisions about scalability, reliability, performance, and maintainability.
- Used in real-world applications like designing scalable web apps, distributed systems, databases, and cloud infrastructures.



Why is System Design Important?

- Scalability & Reliability – Ensures systems handle millions of users without failure.
- Architectural Thinking – Goes beyond coding; involves trade-offs like CAP theorem, SQL vs. NoSQL.
- Career Growth – Essential for becoming a senior engineer or architect.
- Real-World Problem Solving – Helps in building actual systems, not just clearing interviews.
- Trade-Offs & Decision Making – Balances scalability, cost, speed, and complexity.
- Future-Proofing – Prevents bottlenecks and allows smooth evolution of software.

Note: Many learners approach system design just to crack interviews, but in reality, understanding system design is a critical skill for building scalable, reliable, and efficient software systems. If you deeply understand system design principles, interviews will naturally become easier—but more importantly, you'll be able to architect real-world solutions effectively.

The Evolution of System Design Over the Last 25 Years

- 1995-2005: Early Monolithic Web Applications
  - LAMP stack (Linux, Apache, MySQL, PHP) dominated
  - Stateless servers, single-node databases, basic MVC architectures
  - Server-side rendering, no real-time features
- 2005-2010: Scaling Challenges & Distributed Systems
  - Social media & e-commerce drove demand for better performance
  - Introduction of caching (Memcached, Redis), CDNs, and database replication
  - Horizontal scaling & load balancing became standard
- 2010-2015: The Cloud Revolution
  - AWS, Azure, GCP enabled on-demand infrastructure scalability
  - Shift to virtualized & containerized deployments
  - NoSQL databases (MongoDB, Cassandra) gained popularity
- 2015-2020: Microservices & Event-Driven Architectures
  - Monoliths evolved into microservices for agility
  - API gateways, service meshes, event-driven systems (Kafka, RabbitMQ)
  - CI/CD pipelines streamlined software delivery
- 2020-Present: Real-Time, AI-Driven & Edge Computing
  - Focus on low-latency (streaming, AI-driven recommendations)
  - Serverless computing, Kubernetes, and edge computing growth
  - Increased emphasis on security, compliance, and observability

How This Course is Structured?

- **Introduction** – Overview of system design and its real-world importance
- **System Design Fundamentals** – Core concepts, architecture patterns, and trade-offs
- **Scalability** – Handling growing traffic, load balancing, caching, and database sharding
- **Storage** – SQL vs. NoSQL, distributed databases, and data partitioning strategies
- **Performance** – Optimizing latency, throughput, and efficiency
- **Reliability** – Ensuring fault tolerance, high availability, and disaster recovery
- **Security** – Authentication, authorization, and data protection best practices
- **Putting It All Together** – Combining principles to design real-world systems
- **Approach for System Design** – Frameworks and methodologies for structuring solutions
- **Case Studies** – Deep dives into real-world architectures of companies like Netflix, WhatsApp, and Uber
- **Interview Tips** – Strategies to excel in system design interviews

Note: Each section consists of multiple lessons, covering concepts in depth.

How to Navigate This Course Effectively

- Start with Fundamentals — build depth & clarity
- Case Studies — simulate real interview discussions
- Use the 4-Step Design Framework to stay structured
- Don’t skip details — pause, research, resume
- Topics are interconnected — revisit the course once again