What is System Design?

Ashish Pratap Singh

4 min read

When you scroll through **Instagram**, stream your favorite show on **Netflix**, or shop on **Amazon**, you probably don't pause to think about what's happening behind the scenes.

But with every tap, click, or refresh, a **complex network of interconnected components** work seamlessly to deliver a smooth experience.

Behind this seamless experience lies the art and science of **System Design**.

---

1. What Is System Design?

At its core, **System Design** is the process of defining how different parts of a software system interact to meet both **functional** (what it should do) and **non-functional** (how well it should do it) requirements.

It's not about writing code, at least not yet. It's about making **high-level architectural decisions** that balance scalability, reliability, performance, and cost.

A Real-World Analogy: Designing a Skyscraper

Imagine you're an architect designing a skyscraper.

You don't start by laying bricks. You start by asking questions:

- How many floors will it have?
- How many people should it support?
- What kind of soil is it built on?
- What level of earthquake resistance is needed?

Once the requirements are clear, you create **blueprints** showing how everything fits together: the foundation, the structural supports, the plumbing, the electrical layout, and the elevator shafts.

You also consider how different systems **interact**, such as how plumbing might affect electrical layouts. You plan for **future expansion** (scalability) and think about how the building will handle unexpected issues (fault tolerance).

In the software world, this translates to:

- **Architecture:** The overall structure of the system. Should the system be built as a monolith, a set of microservices, or an event-driven system?
- **Components/Modules:** Databases, servers, load balancers, caches, message queues, and APIs.
- **Interfaces:** How these components communicate with each other (e.g., REST APIs, gRPC).
- **Data:** How data is stored, managed, accessed, and kept consistent.

---

2. 10 Big Questions of System Design

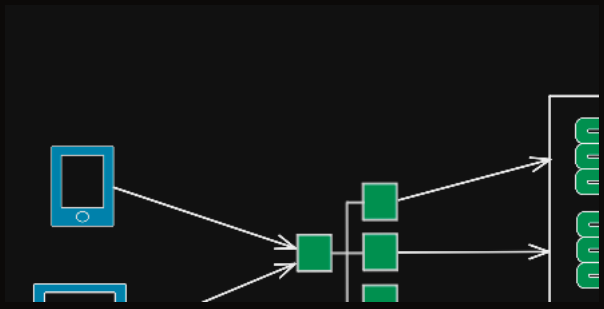
On a high level, system design revolves around answering these 10 big questions:

1. **Scalability:** How will the system handle a large number of users or requests simultaneously?
2. **Latency and Performance:** How can we reduce response time and ensure low-latency performance under load?
3. **Communication:** How do different components of the system interact with each other?
4. **Data Management:** How should we store, retrieve, and manage data efficiently?

5. **Fault Tolerance and Reliability:** What happens if a part of the system crashes or becomes unreachable?
6. **Security:** How do we protect the system against threats such as unauthorized access, data breaches, or denial-of-service attacks?
7. **Maintainability and Extensibility:** How easy is it to maintain, monitor, debug, and evolve the system over time?
8. **Cost Efficiency:** How can we balance performance with infrastructure cost?
9. **Observability and Monitoring:** How do we monitor system health and diagnose issues in production?
10. **Compliance and Privacy:** Are we complying with relevant laws and regulations (e.g., GDPR, HIPAA)?

---

3. Key Components of a System



A typical software system can be broken down into several key components:

- **Client/Frontend:** The part of the system that users interact with directly (e.g., web browsers, mobile apps). It is responsible for displaying information, collecting user input, and communicating with the backend.
- **Server/Backend:** The backend handles the core functionality of the system. It processes incoming requests, executes business logic, interacts with databases or services, and sends responses back to the client.
- **Database/Storage:** This component is responsible for storing and managing data. It can take various forms, including relational databases (SQL), non-relational stores (NoSQL), in-memory caches, or distributed object storage systems, depending on the needs of the application.
- **Networking Layer:** This includes components like load balancers, APIs, and communication protocols that ensure reliable and efficient interaction between different parts of the system.
- **Third-party Services:** These are external APIs or platforms that extend the system's capabilities. Common examples include payment processors, email or SMS

notification services, authentication providers, analytics tools, and cloud-based AI services.

---

4. The Process of System Design

Designing a system is not a one-size-fits-all approach. It's a step-by-step process that starts with understanding the requirements and ends with a detailed blueprint.

Here are the key steps:

Step 1: Requirements Gathering

Every design starts with a conversation. Before drawing diagrams or choosing technologies, focus on understanding what the system needs to do.

Ask questions like:

- What are the **functional requirements** (core features and workflows)?
- What are the **non-functional requirements** (scalability, availability, latency, consistency)?
- Who are the users, and how many are expected initially and at scale?
- What's the expected **data volume** and **traffic pattern**?
- Are there any **constraints** (e.g., specific technologies, budgets, or compliance rules)

Step 2: Back-of-the-Envelope Estimation

Next, estimate the **scale** of your system. Approximate numbers give you a sense of what you're designing for.

Estimate:

- Data size (storage requirements)
- Queries per second (QPS) or requests per second (RPS)
- Bandwidth needs
- Number of servers or instances required

These rough calculations help guide architectural decisions and ensure your design is grounded in realistic expectations.

Step 3: High-Level Design (HLD)

Now that you understand what you're building and how big it needs to be, start visualizing the system's **core components** and how they interact.

Define:

- The main modules and services

- Data flow between them
- External dependencies (e.g., third-party APIs, external databases)

At this stage, you're sketching the **architecture blueprint**, a bird's-eye view of the system.

Step 4: Data Model / API Design

Once the architecture is clear, move closer to the data and interfaces.

- Choose the right **database type(s)** — relational, NoSQL, time-series, etc.
- Define **schemas, tables, and relationships** to support your use cases.
- Design **APIs** for interaction between services (e.g., `POST /tweet`, `GET /timeline`).

Step 5: Detailed Design / Deep Dive

Zoom into each component and define:

- Internal logic, caching, and concurrency handling
- Scaling strategies (horizontal vs vertical scaling)
- Replication, partitioning, and fault tolerance

This is also where you address **non-functional requirements (NFRs)** like availability, reliability, and latency.

In other words, you go from *what* the system does to *how* it does it.

Step 6: Identify Bottlenecks and Trade-offs

No system is perfect. Every choice has trade-offs.

Ask yourself:

- Where could the system break under high load?
- What are the **single points of failure**?
- Can caching or replication help reduce pressure?
- Is it okay to choose **eventual consistency** for higher availability?

A strong design doesn't eliminate trade-offs, it makes them **explicit** and **justifiable**.

Step 7: Review, Explain, and Iterate

Finally, step back and evaluate. Explain your design decisions clearly—why you made certain choices and how they meet the requirements.

Be open to feedback, iterate on weak spots, and refine your design.

You don't need to get everything perfect on the first try.

What matters is your ability to **adapt, refine, and evolve** the design as you uncover new insights or constraints.

---

5. Conclusion

System design is one of the most important skills for building software that's **reliable, scalable, and maintainable**.

Whether you're designing a small web app or a massive distributed platform, understanding system design principles helps you make **better architectural decisions**, choose the right technologies, and optimize performance with confidence.

The first step in mastering system design is to understand its **core concepts and building blocks**.

In the next chapter, we will explore the **Top 30 System Design Concepts** that you will most commonly come across while designing large scale systems or preparing for system design interviews.