Hi all, I have tried to kept this document the same way that I have been teaching you, if you need something specific apart form this, please let shivani know, will update more documents, apologies for a bit of delay folks! Wish you all the best and love from my end! <3

The 5-Step Framework for Nailing System Design Interviews

Alright class, here's the game plan. A system design interview isn't about getting the one "right" answer. It's about showing the interviewer *how* you think. Follow these five steps, and you'll be able to tackle any problem they throw at you.

Step 1: Clarify Requirements & Scope the Problem 🗺️

This is the most important step. Don't start designing anything until you know exactly what you're building. Your job is to act like a product manager before you act like an engineer.

Ask questions to understand the constraints and features.

- **Functional Requirements (What does it do?):**
    - "What are the absolute must-have features for version 1? For example, for a YouTube design, is it just video uploading and viewing, or do we also need comments and live streaming?"
    - "Who are the users? Are they mobile, web, or both?"
- **Non-Functional Requirements (How well does it do it?):**
    - "What kind of scale are we talking about? A few thousand users or hundreds of millions?"
    - "Does this system need to be real-time with very low latency?" (like a chat app)
    - "Is it okay if data is slightly delayed, meaning is eventual consistency acceptable?" (like a social media feed)

Never assume anything. State your assumptions clearly if the interviewer doesn't have an answer. **Folks focus on this point the most in the interview**. A brilliant design for the wrong problem is a complete failure.

Step 2: Estimate the Scale (Back-of-the-Envelope Math) 🔢

Now that you know *what* to build, you need to figure out *how big* it needs to be. This step justifies all your technology choices later on.

- **Calculate Traffic (QPS):** Estimate how many read and write requests per second the system will get.
    - *Example:* "If we have 100 million daily users and they each read 10 articles, that's 1 billion reads per day. This gives us a rough QPS to design for."

- **Calculate Storage:** Estimate how much data you'll need to store over a few years. Pay special attention to media like images and videos.
- **Calculate Bandwidth:** Estimate how much data will be moving in (ingress) and out (egress) of your system.

You don't need to be perfect. The goal is to get a general idea of the scale (is it terabytes or petabytes?) to prove you're not designing in a vacuum.

---

3. Design the High-Level Architecture (The Whiteboard Sketch) ✏️

This is where you draw the big picture. Sketch out the main components and how they connect.

1. **Start with the User:** Draw the client (web/mobile).
2. **Add the Entry Point:** Show how traffic comes in through a **Load Balancer** and an **API Gateway**.
3. **Break it Down:** Divide the system into logical **Microservices** (e.g., `User Service`, `Search Service`, `Review Service`). Don't go into too much detail yet, just the main responsibilities.
4. **Choose Data Stores:** For each service, identify the right type of database.
    - Need structured data and transactions? Use a **Relational DB** (like PostgreSQL).
    - Need to scale writes and handle simple lookups? Use a **NoSQL DB** (like Cassandra).
    - Need fast lookups and caching? Use an **In-Memory Cache** (like Redis).
    - Need complex text search? Use a **Search Engine** (like Elasticsearch).
5. **Add Messaging:** If services need to communicate asynchronously, add a **Message Queue** (like Kafka or RabbitMQ).

Folks focus on this point the most in the interview. A clean, logical, and well-explained diagram shows you can think about a complex system from a high level.

---

4. Deep Dive into Key Components ⚙️

You can't explain every single detail of the system, so pick one or two interesting parts and go deep. This is your chance to shine and show off your detailed knowledge.

Good areas for a deep dive include:

- A complex data flow (e.g., "How does a new review get processed and update the average rating for a business?").
- A real-time component (e.g., "How does the chat system deliver messages to online vs. offline users?").
- A specialized service (e.g., "How does the search ranking algorithm work?").

Explain the specific technologies, data models, and protocols you would use for that component.

---

5. Identify Bottlenecks & Discuss Trade-offs ⚖️

No system is perfect. The final step is to show you understand the weaknesses of your design and how you might improve it in the future.

- **Identify Bottlenecks:**
  - "A popular business could create a 'hotspot' in our database. We can mitigate this with aggressive caching."
  - "The Fan-out Service could become a bottleneck if a group has millions of members."
- **Discuss Trade-offs:** Every decision has a trade-off. Talk about them.
  - "We chose Cassandra for its write performance, but that means we lose some query flexibility compared to SQL."
  - "Using a CDN reduces latency and cost, but it adds complexity to cache invalidation."
- **Future Improvements:** Briefly mention how you would scale the system further, add new features, or improve monitoring and security.