System Design Interview Study Guide

28-Day Preparation Plan for Staff-Level Interviews

1 Study Strategy: The Building-Blocks-First Approach

1.1 Why This Order Matters

System design interviews differ fundamentally from algorithm interviews. Instead of implementing solutions, you're architecting distributed systems. Success requires:

1. Foundational Knowledge: Understanding building blocks (load balancers, caches, databases) before combining them

2. Pattern Recognition: Recognizing when to apply specific architectures (microservices, CQRS, event sourcing)

3. Trade-off Analysis: Articulating why you choose one approach over another (CAP theorem, consistency vs latency)

4. Communication Skills: Explaining complex systems clearly with diagrams and examples

1.2 The Optimal Strategy: Weak Spots First, Then Sequential

Phase 1: Diagnostic (Day 1)

1. Review all 27 topics briefly (30 min each)
2. Identify 5-8 weak topics (unfamiliar concepts, can't explain trade-offs)
3. Mark as Priority 1 in the progress tracker
4. Topics most engineers struggle with: CAP theorem, sharding strategies, consistency models, capacity estimation

Phase 2: Intensive Practice on Weak Spots (Days 2-10)

1. Spend 80% of time on Priority 1 topics
2. For each topic:
   - Read notes thoroughly (1 hour)
   - Watch supplementary video if available (1 hour)
   - Practice mock design using that concept (1 hour)
   - Explain out loud as if teaching someone (30 min)
3. Example: If weak on "Sharding," do multiple designs requiring sharding (URL shortener, chat system)
4. Repeat each Priority 1 topic using spaced repetition: Day 2, 4, 7, 10

Phase 3: Sequential Study of Familiar Topics (Days 11-17)

1. Work through Priority 2 and 3 topics in order
2. Faster pace (1-2 hours per topic)
3. Focus on connecting concepts (how caching + CDN work together)
4. Build mental models of complete systems

Phase 4: Full System Designs (Days 18-24)

1. Practice complete designs end-to-end (URL shortener, chat, feed, ride-sharing, video streaming)
2. Time yourself: 45 minutes per design
3. Cover all phases: Requirements, API, Data Model, High-Level Design, Deep Dives, Operations
4. Practice explaining trade-offs clearly

Phase 5: Mock Interviews (Days 25-28)

1. Simulate real interviews with timer
2. Practice with peers or use Pramp/interviewing.io
3. Focus on communication and whiteboarding (or diagramming tools)
4. Review feedback and iterate

1.3 Falling Behind? Here's Your Recovery Plan

The 28-day plan is aggressive. If you're behind schedule, don't panic. Use this triage strategy:

If Behind by 3-5 Days (Minor Delay):

- **Skip Priority 3 topics entirely** - Focus only on P1 and P2
- **Reduce spaced repetition:** Day 1, 4, 14 only (skip Day 2 and 7 reviews)
- **Cut practice problems:** Do only Tier 1 designs (skip Tier 2 and 3)
- **Extend by 1 week:** Total 35 days is still excellent prep

If Behind by 1+ Weeks (Major Delay):

- **Focus on top 8 Priority 1 topics only:** CAP, Caching, Sharding, Partitioning, Message Queues, Rate Limiting, Monitoring, Consensus
- **Minimal spaced repetition:** Day 1 deep study + Day 7 review only
- **Practice 3 core designs:** URL Shortener (fundamentals), Chat System (real-time), News Feed (scale)
- **Extend to 6 weeks total:** Quality over speed - better to master 8 topics than rush 27

If Interview Is in 1 Week (Emergency Prep):

- **Day 1-2:** Memorize capacity formulas + RADEO framework
- **Day 3-4:** Do URL Shortener + News Feed designs 3x each (until smooth)
- **Day 5-6:** Review cheatsheet sections 1, 3, 6, 8 (Building Blocks, CAP, Capacity, Strategy)
- **Day 7:** Mock interview + rest
- **Reality Check:** This is Senior-level prep, not Staff. Manage expectations.

Key Principle: Depth on few topics beats shallow coverage of all. Interviewers value mastery over breadth.

1.4 Spaced Repetition for Rusty Topics

For each Priority 1 topic, review on:

- **Day 1:** Initial deep study (3 hours)
- **Day 2:** Quick review + practice problem (1 hour)
- **Day 4:** Apply in full system design (1 hour)
- **Day 7:** Teach concept out loud (30 min)
- **Day 14:** Verify mastery with mock interview (45 min)

Research shows spaced repetition increases retention by 200% compared to cramming. Missing a weak topic review is unacceptable for Staff-level prep.

2 Common Weak Spots

Based on thousands of system design interviews, these topics trip up even senior engineers:

1. **CAP Theorem & Trade-offs:** Many can state "pick 2 of 3" but struggle to apply in designs. Must articulate: "I'm choosing availability over consistency here because..."

2. **Capacity Estimation:** Engineers skip this or guess wildly. Practice calculating QPS, storage, bandwidth, memory for every design. Interviewers test rigor here.
3. **Sharding Strategies:** Knowing "hash-based sharding" isn't enough. Must discuss: resharding, hot spots, cross-shard joins, consistent hashing.
4. **Consistency Models:** Confusion between strong, eventual, causal consistency. When to use read-your-writes vs linearizability?
5. **Caching Strategies:** Cache-aside vs write-through vs write-back. Eviction policies. Cache invalidation ("hardest problem in CS").
6. **Message Queues:** When to use point-to-point vs pub-sub? Delivery guarantees? Ordering? Kafka vs RabbitMQ vs SQS?
7. **Rate Limiting:** Token bucket vs leaky bucket vs sliding window. Implementation details (Redis-based).
8. **Monitoring & Operations:** Many designs omit this. Must discuss: metrics (QPS, latency, errors), alerting, failure modes, rollback strategies.

If you're rusty on any of these, mark them Priority 1 and attack them first.

3 Progress Tracking Table

Use this table to track your study progress across all 27 topics. Color coding indicates priority:

- **Priority 1 (P1):** Weak spots - Master these first with spaced repetition
- **Priority 2 (P2):** Moderate - Solidify with practice
- **Priority 3 (P3):** Familiar - Quick review sufficient

Note: On B&W printers, priorities show as: P1 (darkest gray), P2 (medium gray), P3 (lightest gray)

3.1 How to Assign Your Own Priorities

The table below pre-assigns priorities based on common weak spots, but **customize to your experience:**

Mark as Priority 1 (Red) if:

- You can't explain the concept clearly in 2 minutes
- You've never implemented it in production (or it's been 3+ years)
- You struggle to articulate trade-offs ("When would you use X vs Y?")
- It's a known interview hot topic (CAP, sharding, caching, consistency models)

Mark as Priority 2 (Yellow) if:

- You understand the concept but haven't practiced explaining it recently
- You know the theory but lack hands-on experience
- It's important but not your current weak spot

Mark as Priority 3 (Green) if:

- You've used it extensively in production (within last 2 years)
- You can explain trade-offs confidently without notes
- You've discussed it in past interviews successfully

Pre-assigned Priorities Explained:

- **P1 (Red):** CAP theorem, caching strategies, sharding, partitioning, message queues, rate limiting, monitoring, consensus - these trip up 70%+ of Staff candidates
- **P2 (Yellow):** Full system designs (URL shortener, chat, feed, etc.) + intermediate topics (microservices, search, big data) - require practice but less conceptually dense
- **P3 (Green):** Fundamentals most senior engineers know (load balancing, basic DB concepts, API design, replication) - quick review sufficient

Customization Example: If you're a backend engineer who's built caching layers, change "Caching Strategies" from P1 to P3. If you've never touched Kafka/RabbitMQ, keep "Message Queues" as P1.

Instructions:

1. **Week column:** Check off when you complete initial study
2. **Day 2, 4, 7, 14 columns:** For Priority 1 topics only, check off spaced repetition reviews
3. **Mock column:** Check off when you successfully use this concept in a mock interview
4. **Notes:** Track specific problems practiced, resources used, key insights

Topic	Pri	Week	D2	D4	D7	D14	Mock	Notes / Insights
Week 1: Fundamentals								

Topic	Pri	Week	D2	D4	D7	D14	Mock	Notes / Insights
1. Scalability Fundamentals	P3	☐					☐	Vertical vs horizontal, stateless design
2. Load Balancing	P3	☐					☐	L4 vs L7, algorithms, session persistence
3. Caching Strategies	P1	☐	☐	☐	☐	☐	☐	Cache-aside, write-through, eviction policies
4. Database Fundamentals	P3	☐					☐	RDBMS vs NoSQL, ACID, BASE
5. CAP Theorem	P1	☐	☐	☐	☐	☐	☐	CP vs AP, PACELC, real-world examples
6. Design: URL Shortener	P2	☐		☐		☐	☐	End-to-end practice, capacity estimation
Week 2: Communication								
7. Message Queues	P1	☐	☐	☐	☐	☐	☐	Point-to-point vs pub-sub, Kafka, RabbitMQ
8. API Design	P3	☐					☐	REST vs RPC, versioning, pagination
9. Microservices	P2	☐		☐		☐	☐	Service discovery, API gateway, saga pattern
10. Data Partitioning	P1	☐	☐	☐	☐	☐	☐	Hash, range, geo sharding, consistent hashing
11. Replication	P3	☐					☐	Master-slave, multi-master, sync vs async
12. Design: Chat System	P2	☐		☐		☐	☐	WebSocket, message queue, group chat
Week 3: Advanced Topics								
13. Database Sharding	P1	☐	☐	☐	☐	☐	☐	Strategies, resharding, cross-shard joins
14. CDN & Caching	P3	☐					☐	Edge locations, cache invalidation, TTL
15. Search & Indexing	P2	☐		☐		☐	☐	Elasticsearch, inverted index, ranking
16. Big Data Processing	P2	☐		☐		☐	☐	MapReduce, Spark, batch vs stream
17. NoSQL Deep Dive	P3	☐					☐	Key-value, document, column, graph
18. Design: Social Feed	P2	☐		☐		☐	☐	Fan-out on write/read, ranking, pagination
Week 4: Production Systems								
19. Security	P3	☐					☐	Auth, encryption, rate limiting, HTTPS
20. Monitoring & Observability	P1	☐	☐	☐	☐	☐	☐	Metrics, logs, traces, alerting, SLOs
21. REST DAY	–							Take a break or review weak spots
22. Rate Limiting	P1	☐	☐	☐	☐	☐	☐	Token bucket, leaky bucket, sliding window
23. Consensus & Coordination	P2	☐		☐		☐	☐	Paxos, Raft, ZooKeeper, leader election
24. Design: Ride Sharing	P2	☐		☐		☐	☐	Geospatial index, matching, real-time updates
25. Design: Video Streaming	P2	☐		☐		☐	☐	Transcoding, CDN, adaptive bitrate
26-27. Mock Interviews	–	☐					☐	Full end-to-end with timing
28. Final Review	–	☐						Weak spots, common mistakes

4 Capacity Estimation Practice

System design interviews always include capacity estimation. Practice these formulas until automatic:

4.1 Standard Assumptions

- **1 day = 86,400 seconds** (memorize this!)
- **1 month** $\approx$ 30 days = 2,592,000 seconds
- **Peak traffic** = 3-5x average (use 3x for conservative estimate)
- **80/20 rule:** 20% of data generates 80% of traffic
- **Cache hit rate:** 80-90% typical for hot data
- **Replication factor:** 3x for high availability
- **Headroom:** Add 30% for safety margin

4.2 Core Formulas

1. QPS (Queries Per Second):

$$\text{QPS} = \frac{\text{Daily Active Users} \times \text{Requests per User}}{\text{86,400 seconds}}$$

$$\text{Peak QPS} = \text{Average QPS} \times 3$$

2. Storage:

$$\text{Storage} = \text{Items} \times \text{Size per Item} \times \text{Replication Factor}$$

$$\text{5-Year Storage} = \text{Daily Items} \times 365 \times 5 \times \text{Size}$$

3. Bandwidth:

$$\text{Read Bandwidth} = \text{Read QPS} \times \text{Avg Response Size}$$

$$\text{Write Bandwidth} = \text{Write QPS} \times \text{Avg Request Size}$$

4. Memory (for Caching):

$$\text{Cache Memory} = 0.2 \times \text{Daily Requests} \times \text{Avg Response Size}$$

(Using 80/20 rule: cache 20% of data for 80% of traffic)

5. Server Count:

$$\text{Servers} = \frac{\text{QPS}}{\text{QPS per Server}} \times 1.3$$

(Typical: 1K-10K QPS per web server, 100-1K per DB query server)

4.3 Practice Problem 1: URL Shortener

Requirements:

- 100 million new URLs per month
- 10:1 read-to-write ratio
- Each URL record = 500 bytes (short code + long URL + metadata)
- Plan for 5 years of storage

Solution:

Step 1: Write QPS

$$\text{Write QPS} = \frac{100M}{30 \times 86400} = \frac{100,000,000}{2,592,000} \approx 39 \text{ writes/sec}$$

Step 2: Read QPS

$$\text{Read QPS} = 39 \times 10 = 390 \text{ reads/sec}$$

$$\text{Peak Read QPS} = 390 \times 3 = 1,170 \text{ reads/sec}$$

Step 3: Storage (5 years)

$$\text{Total URLs} = 100M \times 12 \times 5 = 6 \text{ billion URLs}$$

$$\text{Storage} = 6B \times 500 \text{ bytes} = 3 \text{ TB (raw)}$$
$$\text{With 3x replication} = 3 \times 3 = 9 \text{ TB}$$

Step 4: Bandwidth

$$\text{Write Bandwidth} = 39 \times 500 = 19,500 \text{ bytes/sec} \approx 20 \text{ KB/sec}$$
$$\text{Read Bandwidth} = 390 \times 500 = 195,000 \text{ bytes/sec} \approx 200 \text{ KB/sec}$$

Step 5: Cache Memory (80/20 rule)

$$\text{Daily Reads} = 390 \times 86,400 = 33.7M \text{ reads/day}$$
$$\text{Cache } 20\% = 0.2 \times 33.7M \times 500 = 3.37 \text{ GB}$$

Step 6: Servers

- **Web Servers:** Peak 1,170 QPS / 5,000 per server = 1 server (+ 1 for redundancy = 2)
- **Database:** 9 TB / 3 TB per server = 3 shards (+ replicas = 9 DB servers)
- **Cache:** 4 GB fits in single Redis (+ 1 replica = 2 cache servers)

4.4 Practice Problem 2: Chat System

Requirements:

- 50 million daily active users
- Each user sends 20 messages per day on average
- Each message = 200 bytes (text + metadata)
- Store message history for 2 years

Your Turn: Calculate write QPS, storage, bandwidth, cache memory, and server count. Use the formulas above.

(Hint: Write QPS = 50M × 20 / 86,400 ≈ 11,574 messages/sec)

4.5 Practice Problem 3: Video Streaming

Requirements:

- 10 million daily active users
- Each user watches 30 minutes of video per day
- Video: 720p @ 2 Mbps bitrate (average)
- Store videos for 10 years
- 100,000 new videos uploaded per day, average 10 minutes each

Your Turn: Calculate watch QPS, upload QPS, storage (10 years), bandwidth (separate read/write).

(Hint: Convert minutes to bytes - 1 minute @ 2 Mbps = 2 Mbps × 60 sec / 8 bits/byte = 15 MB)

5 Mock Interview Preparation Checklist

5.1 Week 4: Mock Interview Readiness

By Week 4, you should be ready to simulate real interviews. Use this checklist:

Before Mock Interview:

☐ Set up whiteboard or diagramming tool (Excalidraw, draw.io, or physical board)
☐ Prepare timer (45 minutes for design phase)
☐ Have cheatsheet printed for reference (but don't rely on it)
☐ Pick a random design problem (see list below)
☐ Record yourself or practice with peer

During Mock Interview (Follow RADEO Framework):

☐ **Requirements (5 min):** Ask clarifying questions, define functional/non-functional requirements
☐ **API Design (5 min):** Sketch 3-5 key endpoints with request/response
☐ **Data Model (5-10 min):** Database choice, schema, relationships, indexes
☐ **High-Level Design (10 min):** Draw components (client, LB, app, DB, cache, queue), data flow
☐ **Deep Dives (15 min):** Identify bottlenecks, discuss trade-offs, add sharding/caching/replication
☐ **Operations (5 min):** Monitoring, failure modes, security, scaling

Communication Checklist:

☐ Think out loud (don't go silent)
☐ Ask clarifying questions (don't assume)
☐ Discuss trade-offs explicitly ("I'm choosing X over Y because...")
☐ Draw clear diagrams with labels
☐ Use numbers (QPS, storage, latency) to justify decisions
☐ Acknowledge when you don't know something ("I'm not familiar with X, but I'd approach it by...")

After Mock Interview:

☐ Review recording or get peer feedback
☐ Identify gaps (concepts you struggled to explain)
☐ Update progress tracker with notes
☐ Redo the same design next day (should be much smoother)

5.2 Mock Interview Problem Bank

Practice these in order. First 5 are most common in real interviews:

Tier 1 - Must Practice:

1. URL Shortener (TinyURL, bit.ly)
2. Chat System (WhatsApp, Slack)
3. News Feed (Twitter, Instagram, Facebook)
4. Video Streaming (YouTube, Netflix)
5. Ride Sharing (Uber, Lyft)

Tier 2 - High Frequency:

6. Search Autocomplete (Google search suggestions)
7. Notification System (Push notifications, email alerts)
8. Web Crawler (Google crawler, Scrapy)
9. Distributed Cache (Memcached, Redis cluster)
10. Rate Limiter (API rate limiting)

Tier 3 - Good Practice:

11. Photo Sharing (Instagram, Flickr)
12. E-commerce Platform (Amazon product catalog)
13. Ticketing System (Ticketmaster, live event booking)
14. Food Delivery (DoorDash, UberEats)
15. Ad Click Tracking (Google Ads analytics)

6 Staff-Level Interview Expectations

6.1 What Distinguishes Staff from Senior?

Senior Engineer:

- Design functional systems that meet requirements

- Apply standard patterns (load balancer, cache, database)
- Explain basic trade-offs (SQL vs NoSQL, sync vs async)

Staff Engineer (Higher Bar):

- **Proactive Problem Identification:** Spot bottlenecks before interviewer asks
- **Quantitative Analysis:** Use numbers to justify every decision ("With 10K QPS, we need...")
- **Deep Trade-off Discussion:** Go beyond surface level (e.g., not just "use cache" but "LRU cache because 80/20 rule, invalidation strategy is TTL + manual purge on updates")
- **Failure Modes:** Discuss what breaks and how to recover (DB failure, network partition, cascading failures)
- **Operations Mindset:** Monitoring, alerting, deployment strategy, rollback plan
- **Cross-Team Considerations:** API contracts, backward compatibility, migration strategy
- **Alternative Approaches:** Present multiple solutions, compare pros/cons, justify final choice

6.2 Red Flags for Staff Level

These mistakes will hurt you at Staff level (even if acceptable at Senior):

- Jumping to solution without clarifying requirements
- Forgetting capacity estimation or guessing wildly
- Ignoring edge cases (what if service goes down?)
- Not discussing monitoring/alerting
- Overlooking security (auth, rate limiting, encryption)
- Failing to articulate trade-offs clearly
- Being too attached to one solution (not considering alternatives)
- Going silent for extended periods (not thinking out loud)
- Over-engineering simple problems
- Under-engineering scale problems

6.3 Staff-Level Response Example: Good vs Bad

Let's compare Senior vs Staff responses to: *"Design a URL shortener like TinyURL."*

BAD Response (Senior-Level Thinking):

"We need a database to store URLs. Let's use MySQL with two tables: one for URLs and one for users. When a user submits a URL, we generate a short code using Base62 encoding and store it. We'll use Redis for caching popular URLs. For high traffic, we'll add a load balancer and scale horizontally. That should handle the load."

Why This Is Insufficient for Staff:

- No requirements clarification (read-heavy? write-heavy? scale?)
- No capacity estimation (how much traffic? storage?)
- No trade-off discussion (why MySQL? why Base62? why Redis?)
- No failure modes (what if DB goes down?)
- No operations (monitoring? deployment?)

GOOD Response (Staff-Level Thinking):

"Let me clarify requirements first. Are we expecting 100M new URLs per month with 10:1 read ratio? Let me calculate capacity..."

[Does math on whiteboard: 39 writes/sec, 390 reads/sec, 9TB storage over 5 years]

"For this scale, I'll use Base62 encoding with 7 characters (62^7 = 3.5 trillion combinations, plenty of headroom). For the database, I'm choosing a key-value store like DynamoDB over MySQL because:

- Simple schema (short_code -¿ long_url)
- High write throughput (39 writes/sec easily handled)
- Built-in replication and auto-scaling

Trade-off: We lose relational queries, but we don't need them here. For caching, I'm using Redis with LRU eviction to cache the top 20% of URLs (80/20 rule). That's about 3.4GB of cache memory based on daily reads.

For failure modes, if Redis goes down, we fall back to DynamoDB (slower but still works). If a DynamoDB partition goes down, we have 3x replication for high availability. For monitoring, I'd track: write QPS, read QPS, cache hit rate (should be 80%+), p99 latency (target ¡50ms), and error rate.

Why This Is Staff-Level:

- Clarified requirements upfront
- Used numbers to justify decisions
- Articulated specific trade-offs (DynamoDB vs MySQL)
- Discussed failure modes and fallback
- Included operations (monitoring metrics)
- Mentioned specific algorithms (Base62, LRU) with reasoning

Key Lesson: Staff engineers justify every decision with *why*, not just *what*. Use numbers, discuss trade-offs, anticipate failures.

6.4 Common Mistakes with Examples

These are the most frequent mistakes in Staff-level interviews, with before/after corrections:

Mistake 1: Vague Capacity Estimation

Before: "We'll need a few database servers and some cache."

After: "With 10K writes/sec and 100K reads/sec, each DB server handles 1K QPS, so we need 10 write servers + 100 read replicas. Cache holds 20% of 1TB dataset = 200GB, which fits in 4x Redis instances (50GB each)."

Why It Matters: Staff engineers quantify everything. Vague estimates suggest lack of production experience.

Mistake 2: Forgetting to Discuss Cache Invalidation

Before: "We'll use Redis to cache user profiles for fast reads."

After: "We'll use Redis to cache user profiles. For invalidation, I'd use a hybrid approach: TTL of 1 hour for passive expiry + event-based invalidation via message queue when profile updates. Trade-off: Adds complexity but ensures consistency within seconds."

Why It Matters: Cache invalidation is notoriously hard. Staff engineers show they've dealt with stale data issues.

Mistake 3: Not Addressing Failure Modes

Before: "We'll replicate the database for high availability."

After: "We'll replicate the database 3x across AZs. If master fails, we promote a replica (30-60 sec downtime using automated failover). During network partition, we enforce quorum (2 of 3 nodes) to prevent split-brain. If entire region fails, we have cross-region async replication (5 min RPO)."

Why It Matters: Staff engineers anticipate failures. Describing failure scenarios shows operational maturity.

Mistake 4: Ignoring Hot Shard Problem

Before: "We'll shard users by user_id using consistent hashing."

After: "We'll shard users by user_id using consistent hashing. For celebrities with 100M+ followers (hot shard), we'll detect high QPS (¿10x average) and move them to a dedicated cache layer with separate infrastructure. This prevents one celebrity from overloading a shard."

Why It Matters: Hot shard is a classic Staff question (Twitter/Instagram interviews). Ignoring it is a red flag.

Mistake 5: Weak Trade-off Articulation

Before: "SQL is good for structured data, NoSQL is good for unstructured data."

After: "I'm choosing Cassandra (NoSQL) over PostgreSQL here because: (1) Write-heavy workload (10K writes/sec) favors LSM-tree, (2) No complex joins needed, (3) Built-in geo-replication. Trade-off: We lose ACID transactions, but we don't need them for this use case (eventual consistency is fine for social feed)."

Why It Matters: Staff engineers show deep understanding of internals (LSM-tree) and justify with use case specifics.

Mistake 6: Not Discussing Monitoring

Before: "The system should be scalable and reliable."

After: "For observability, I'd track: (1) Golden metrics: QPS, latency (p50/p95/p99), error rate, (2) Business metrics: feed load time, post success rate, (3) SLO: 99.9% availability (43 min downtime/month), p99 latency ¡500ms. Alerts fire on SLO violations. We'd use Prometheus + Grafana + PagerDuty."

Why It Matters: Staff engineers own production. Concrete metrics show you've run services at scale.

Practice Exercise:

Review your last 3 mock designs. For each, identify:

- Did I quantify capacity? (servers, storage, bandwidth)
- Did I discuss cache invalidation?
- Did I describe failure modes?
- Did I mention hot shard/hot key problems?
- Did I explain trade-offs with specific reasons?
- Did I include monitoring metrics and SLOs?

If you answered "no" to any, redo that design with corrections. This is the difference between Senior and Staff.

6.5 Final Week Checklist

7 Days Before Interview:

☐ Complete all Priority 1 topics (no gaps)
☐ Practice all 5 Tier 1 designs end-to-end
☐ Comfortable with capacity estimation (under 5 min per problem)
☐ Can draw clean diagrams quickly
☐ Have 2-3 "stories" ready (past projects, scale challenges you solved)

3 Days Before Interview:

☐ Do 2-3 full mock interviews (timed, with peer or online platform)
☐ Review feedback from mocks, fix weak spots
☐ Read cheatsheet cover-to-cover (refresh memory)
☐ Practice explaining trade-offs out loud (to yourself or others)

1 Day Before Interview:

☐ Light review only (don't cram new material)
☐ Skim cheatsheet for key formulas and numbers
☐ Do ONE mock design to stay sharp (pick an easy one: URL shortener)
☐ Prepare questions to ask interviewer (about team, tech stack, challenges)
☐ Get good sleep (seriously - fatigue kills performance)

Day of Interview:

☐ Review capacity estimation formulas (5 min)
☐ Review CAP theorem and trade-offs (5 min)
☐ Remind yourself: Think out loud, ask clarifying questions, discuss trade-offs
☐ Arrive 10 min early, calm and confident

7 Additional Resources

7.1 Recommended Books

- *Designing Data-Intensive Applications* by Martin Kleppmann (the bible)
- *System Design Interview* by Alex Xu (Vol 1 & 2) (practical, interview-focused)
- *Web Scalability for Startup Engineers* by Artur Ejsmont

7.2 Online Platforms

- **Mock Interviews:** Pramp, interviewing.io, Exponent
- **Practice Problems:** SystemDesignPrimer (GitHub), DesignGurus.io
- **Video Courses:** Educative.io "Grokking System Design," Coursera "Cloud Computing"

7.3 Companion Cheatsheet

This study guide pairs with *system-design-templates-cheatsheet.tex* cheatsheet. Print both:

- **Cheatsheet:** Quick reference during practice (building blocks, formulas, patterns)
- **Study Guide:** Strategic planning, progress tracking, capacity practice

How to Use Cheatsheet During Study:

Phase 1-2 (Days 1-10): Deep Learning Mode

- **Before:** Read cheatsheet section thoroughly (15-20 min)
- **During:** Keep cheatsheet open while practicing mock designs
- **After:** Close cheatsheet, try to recreate key concepts from memory
- **Goal:** Internalize patterns, not memorize text

Phase 3-4 (Days 11-24): Quick Reference Mode

- **Before:** Close cheatsheet, attempt design unaided
- **When Stuck:** Glance at cheatsheet for specific concept (30 sec lookup)
- **After:** Review full cheatsheet section you struggled with
- **Goal:** Build confidence, identify remaining weak spots

Phase 5 (Days 25-28): No-Cheatsheet Mode

- **During:** Simulate real interview (no cheatsheet at all)
- **After:** Review cheatsheet to see what you missed
- **Goal:** Prove mastery, calibrate readiness

Cheatsheet Sections by Study Phase:

- **Building Blocks (Sec 1):** Reference daily Week 1-2
- **Observability (Sec 2):** Reference Week 4 for Operations discussion
- **CAP/Consistency (Sec 3):** Master by Day 5 (Priority 1 topic)
- **Scaling Patterns (Sec 4):** Reference daily Week 2-3 (sharding, replication)
- **Capacity Estimation (Sec 6):** Practice formulas daily Week 1-4
- **Interview Strategy (Sec 8):** Read before every mock interview

7.4 Final Thoughts

System design mastery takes time. Don't rush. The 28-day plan is aggressive but achievable if you:

- Follow spaced repetition religiously
- Focus on weak spots first
- Practice out loud (explaining matters more than reading)
- Do full mock interviews (timing and communication are skills)

You've got this. Staff-level is within reach with disciplined preparation. Good luck!