System Design Cheat Sheet

The 45-Minute Interview Framework

Phase	Min	What to do	What interviewers listen for
1. Requirements	5	Clarify features, users, scale; state non-functional goals	Do you ask before you build?
2. Estimation	5	QPS, storage, bandwidth: use the numbers below	Can you size the problem?
3. High-level design	10	Main components; walk one request end to end	Is the skeleton sound and simple?
4. Deep dives	15	Data model, hardest component, scaling the bottleneck	Depth on demand, trade-offs said aloud
5. Wrap-up	5	Failure modes, monitoring, what you would improve	Do you think beyond the happy path?

Two rules throughout: say every trade-off out loud, and never change direction silently.

Numbers Every Engineer Should Know

Operation	Time
L1 cache reference	0.5 ns
Main memory reference	100 ns
Read 1 MB sequentially from memory	250 µs
SSD random read	150 µs
Read 1 MB sequentially from SSD	1 ms
Round trip inside a datacenter	0.5 ms
Read 1 MB from spinning disk	20 ms
Round trip across the Atlantic	150 ms

Quick Math

- **A day** is 86,400 s; round to 10^5. 10M requests/day ≈ 100-120 QPS average; plan 2-5x for peak.
- **Powers of two:** 2^{10} ≈ thousand (KB), 2^{20} ≈ million (MB), 2^{30} ≈ billion (GB), 2^{40} ≈ trillion (TB).
- **Availability:** 99.9% = 8.7 h down/year · 99.99% = 52 min · 99.999% = 5 min.
- **Storage:** 1M users × 1 KB/day ≈ 1 GB/day ≈ 365 GB/year.

The Five Questions (any blank whiteboard)

- **Flow:** who waits, and who shouldn't?
- **Storage:** what is truth, what is derived?
- **Speed:** what is read most, how stale may it be?
- **Failure:** per arrow: hangs, dies, answers twice?
- **Growth:** what doubles first, what breaks then?

System Design Trade-Offs at a Glance

Decision	Choose the first when	Choose the second when
SQL vs NoSQL	Relations, transactions, ad-hoc queries	Massive scale, flexible schema, key-based access
Vertical vs horizontal	Simplicity wins, scale is moderate	Open-ended growth, availability matters
Strong vs eventual consistency	Money, inventory, anything audited	Feeds, counters, profiles: availability first
Sync vs async	The user needs the answer now	Nobody waits: queues absorb spikes
Push vs pull (fan-out)	Ordinary audiences, many readers	Celebrity audiences: compute at read time
Normalize vs denormalize	Write-heavy, correctness-sensitive	Read-heavy: precompute the joins
Monolith vs microservices	Small team, early product	Many teams, independent deploys
Cache vs recompute	Reads repeat, staleness tolerable	Answers must be exact and fresh
Latency vs throughput	Interactive paths: optimize p99	Batch pipelines: optimize rows/hour
Batch vs streaming	Staleness budget of hours	Staleness budget of seconds

The Pattern Map 60+ patterns

Communication (7)
Request-Response · Message Queue · Pub/Sub · Event-Driven · Webhooks · SSE · WebSockets

Storage & Replication (6)
Primary-Replica · Sharding · Consistent Hashing · WAL · Event Sourcing · CQRS

Caching (5)
Cache-Aside · Read-Through · Write-Through · Write-Behind · Stampede Prevention

Reliability (7)
Timeout · Retry + Backoff · Idempotency · Circuit Breaker · Bulkhead · DLQ · Graceful Degradation

Scaling (5)
Vertical · Horizontal · Load Balancing · Auto-Scaling · Connection Pooling

Consistency (4)
Two-Phase Commit · Saga · Quorum · Vector Clocks

API & Edge (9)
Reverse Proxy · CDN · API Gateway · BFF · Rate Limiting · Cursor Pagination · Versioning · Sidecar · Service Mesh

Operations (5)
Health Checks · Distributed Tracing · Blue-Green · Canary · Feature Flags

Data-Intensive (7)
MapReduce · Stream Processing · Lambda/Kappa · CDC · Exactly-Once · Backpressure · Partitioned Consumption

AI-Era (7)
Feature Store · Model Serving · GPU Auto-Scaling · LLM Gateway · Semantic Caching · Vector DB Sharding · RAG

The AI and LLM Cheat Sheet

Pattern	What it does	Reach for it when
RAG pipeline	Model answers from fresh, permissioned documents, with citations	Knowledge changes often or is access-controlled
LLM gateway	One door for model traffic: keys, token budgets, failover	Multiple teams or model providers
Semantic caching	Caches answers by meaning, not exact match	Same question, many phrasings
Vector database	Similarity search over embeddings; shard by tenant	RAG retrieval, semantic search
Continuous batching	Requests share each GPU weight-pass: 20-30x throughput	Self-hosting LLMs
GPU auto-scaling	Warm pools + queue-depth signals vs minutes-long cold starts	Spiky GPU fleets
Feature store	One feature definition for training and serving	Offline and online numbers must match

Eight Patterns Behind 90% of Interview Questions

Pattern	The one-line answer
Cache-Aside	Check cache, load on miss, store; hit ratio and staleness budget rule everything
Sharding	Split rows by a key that is in every query, spreads load, keeps related data together
Message Queue	Work nobody waits on goes behind a queue; spikes become backlog, not outages
Circuit Breaker	Stop calling the dead; fail fast; probe to recover
Idempotency	Same request twice, effect once: keys make retries safe
Saga	Local steps with designed compensations; irreversible steps last
CQRS	Lean write model, purpose-shaped read models, synced from a change stream
Rate Limiting	Token buckets by tier; a polite 429 beats a crash