SYSTEM DESIGN INTERVIEW ASSESSMENT RUBRIC

FAANG-Level Evaluation for Distributed Systems & Architecture

1. OVERVIEW

System design interviews assess whether candidates can architect large-scale distributed systems. This evaluates breadth of knowledge, depth in specific areas, ability to make tradeoffs, and communication of complex ideas. Required for senior roles (L5+).

Expected Interview Flow (45-60 min)

1. Requirements	5-8 min	Clarify functional/non-functional requirements. Establish scope. Identify core features vs nice-to-haves.
2. Estimation	5-7 min	Back-of-envelope calculations: QPS, storage, bandwidth. Establish scale constraints.
3. High-Level Design	10-15 min	Draw main components: clients, servers, databases, caches. Show data flow.
4. Deep Dive	15-20 min	Detail 2-3 critical components. Discuss specific technologies, algorithms, data models.
5. Bottlenecks	5-10 min	Identify bottlenecks, single points of failure. Propose solutions. Discuss tradeoffs.
6. Evolution	5-10 min	How to scale 10x? Handle new requirements? Discuss monitoring, deployment.

Pass/Fail Thresholds

	STRONG PASS	Production-ready design. Senior/Staff level thinking.
	PASS	Solid design with minor gaps. Ready for senior role.
	BORDERLINE	Reasonable design but lacks depth. More experience needed.
	FAIL	Major gaps in understanding. Not ready for senior role.

	STRONG FAIL	Cannot design systems. Fundamental learning needed.

Critical: Any dimension below 3.0 = automatic FAIL. Senior engineers must show competency across ALL areas.

2. SCORING DIMENSIONS

2.1 Requirements Gathering & Scope Definition

Evaluates ability to clarify ambiguous requirements and establish appropriate scope. FAANG expects candidates to drive this conversation, not wait for direction.

Does not clarify requirements. Makes wrong assumptions. Scope undefined.	Few clarifying questions. Misses key requirements. Scope too broad or narrow.	Good clarification of functional requirements. Reasonable scope. Identifies main use cases.	Excellent requirements gathering. Clarifies both functional and non-functional. Prioritizes features. Sets clear scope.	Expert-level scoping. Uncovers hidden requirements. Perfect prioritization. Identifies cross-cutting concerns.

Feedback Templates:

- **Score 1-2:** You didn't establish clear requirements before designing. ALWAYS clarify: Who are users? What are core features? What scale (users, requests, data)? What are latency/availability requirements? What's in/out of scope?
- **Score 3:** Good functional requirements but missed [non-functional aspect]. Always ask about: expected QPS, latency SLA, availability target (99.9%?), consistency requirements, geographic distribution.
- **Score 4-5:** Excellent requirements gathering. You identified [hidden requirement] and correctly prioritized [feature] over [feature]. Your scope definition was clear and appropriate for the time.

Follow-up Questions:

- Who are the primary users?
- What are the core features we must support?
- What's the expected scale in terms of users/requests?
- What are the latency requirements?
- Do we need to support multiple regions?
- Is consistency or availability more important?
- What's out of scope for this discussion?

2.2 Capacity Estimation & Calculations

Evaluates ability to perform back-of-envelope calculations to inform design decisions. FAANG expects quantitative reasoning, not just hand-waving.

No capacity estimation. Cannot calculate basic metrics. Numbers make no sense.	Weak estimation skills. Math errors. Cannot translate to design decisions.	Reasonable estimates for QPS, storage, bandwidth. Some connection to design.	Strong calculations with clear methodology. Uses estimates to drive design choices. Considers growth.	Expert estimation. Precise methodology. Considers all dimensions. Numbers directly inform every design decision.

Feedback Templates:

- **Score 1-2:** You skipped capacity estimation. This must inform your design. Calculate: Daily/Monthly active users → QPS (peak = 2-3x average), Storage per user × users = total storage, Read:Write ratio, Bandwidth = QPS × request size.
- **Score 3:** Good estimation but didn't fully connect to design. When you calculated [X] QPS, that should have informed your choice of [database/cache]. Make numbers drive decisions.
- **Score 4-5:** Excellent capacity planning. Your calculation of [metric] directly justified [design decision]. Your growth projection of [X] informed the scalability approach.

Follow-up Questions:

- How many daily active users?
- What's the read:write ratio?
- How much storage per user? Total storage?
- What's the peak QPS?
- What bandwidth do we need?
- How does this grow over 5 years?

2.3 High-Level Architecture & Component Design

Evaluates ability to design a coherent system architecture with appropriate components. FAANG expects clear diagrams with well-justified component choices.

Cannot draw coherent architecture. Missing critical components. No data flow shown.	Basic architecture with major gaps. Components poorly justified. Unclear data flow.	Reasonable architecture covering main components. Clear data flow. Standard patterns applied correctly.	Strong architecture with well-justified components. Clear separation of concerns. Elegant data flow. Considers failure modes.	Expert architecture. Innovative yet practical. Every component justified. Production-ready design. Considers operations.

Feedback Templates:

- **Score 1-2:** Your architecture was incomplete. Every system needs: Client → Load Balancer → Application Servers → Cache → Database. Plus: CDN for static content, Message Queue for async, separate read/write paths for scale.
- **Score 3:** Good basic architecture but missing [component]. Consider: Why separate services? Where does caching help? How does data flow for read vs write? What happens when [component] fails?
- **Score 4-5:** Excellent architecture. Your separation of [services] was well-reasoned. The use of [pattern] for [problem] showed deep understanding. The diagram was clear and complete.

Follow-up Questions:

- Why did you choose this architecture?
- What happens when this component fails?
- How does data flow for a write request?

- Where would you add caching?
- Why separate these into different services?
- How do components communicate?

2.4 Database Design & Data Modeling

Evaluates ability to design appropriate data models and choose suitable database technologies. FAANG expects deep understanding of SQL vs NoSQL tradeoffs.

No data model. Wrong database choice with no justification. No understanding of tradeoffs.	Basic schema but poorly designed. Database choice weakly justified. Doesn't consider scale.	Reasonable data model. Appropriate database choice with basic justification. Understands main tradeoffs.	Strong data model with proper normalization/ denormalization. Well-justified database choice. Considers sharding, replication.	Expert data modeling. Optimal schema for access patterns. Deep database knowledge. Production-ready data architecture.

Feedback Templates:

- **Score 1-2:** Your data design needs work. Consider: What are the main entities? What are access patterns (read vs write heavy)? SQL for ACID/joins, NoSQL for scale/flexibility. How to shard? How to handle hot spots?
- **Score 3:** Reasonable choice of [database] but didn't fully justify. Explain: Why this over alternatives? How does it handle your scale? What's the consistency model? How would you shard?
- **Score 4-5:** Excellent data design. Your choice of [database] for [reason] was spot-on. The sharding strategy by [key] handles hot spots. Your denormalization for [query pattern] was clever.

Follow-up Questions:

- Why SQL vs NoSQL for this use case?
- What's your sharding strategy?
- How do you handle hot partitions?
- What's the replication strategy?
- How do you ensure consistency?
- What indexes do you need?

2.5 Scalability & Performance

Evaluates ability to design systems that scale horizontally and perform under load. FAANG expects candidates to proactively address scalability.

No scalability consideration. Design cannot handle stated load. Single points of failure everywhere.	Mentions scaling but no concrete strategy. Major bottlenecks unaddressed. Limited understanding.	Reasonable scaling strategy. Identifies main bottlenecks. Uses standard patterns (caching, sharding).	Strong scalability design. Horizontal scaling throughout. Proactive bottleneck elimination. Understands CAP theorem.	Expert scalability. Handles 10-100x growth. Elegant solutions to hard problems. Deep performance optimization.

Feedback Templates:

- **Score 1-2:** Your design doesn't scale. Key patterns: Horizontal scaling (stateless services), Caching (CDN, Redis), Database sharding, Read replicas, Load balancing, Async processing with queues. Apply these systematically.
- **Score 3:** Good scaling approach but [bottleneck] remains. What happens at 10x load? Consider: cache hit ratio, database connection limits, network bandwidth. Every component needs a scaling story.
- **Score 4-5:** Excellent scalability design. Your approach to [bottleneck] using [solution] was sophisticated. The understanding of CAP tradeoffs when choosing [consistency/availability] showed senior-level thinking.

Follow-up Questions:

- How do you handle 10x the load?
- Where are the bottlenecks?
- How do you scale the database?
- What's your caching strategy?
- How do you handle a traffic spike?
- What's the read:write path optimization?

2.6 Reliability & Fault Tolerance

Evaluates ability to design resilient systems that handle failures gracefully. FAANG expects candidates to think about what can go wrong.

No consideration of failures. Single points of failure. No redundancy.	Mentions redundancy but incomplete. Major failure modes unaddressed. No recovery strategy.	Reasonable fault tolerance. Redundancy for critical components. Basic failure handling.	Strong reliability design. No single points of failure. Graceful degradation. Clear recovery procedures.	Expert reliability. Handles cascading failures. Chaos engineering mindset. Production-hardened design.

Feedback Templates:

- **Score 1-2:** Your design has critical single points of failure. Every component needs: redundancy (multiple instances), health checks, automatic failover, graceful degradation, retry logic with backoff. Design for failure.
- **Score 3:** Good redundancy but what happens when [failure scenario]? Consider: How do you detect failures? How fast is failover? What's the blast radius? How do you prevent cascading failures?
- **Score 4-5:** Excellent fault tolerance. Your circuit breaker pattern for [service] prevents cascading failures. The graceful degradation strategy for [feature] maintains user experience. This is production-ready thinking.

Follow-up Questions:

- What happens if this database goes down?
- How do you handle cascading failures?
- What's your failover strategy?
- How do you detect failures?
- What's the blast radius of this failure?
- How do you handle data corruption?

2.7 API Design & System Interfaces

Evaluates ability to design clean, usable APIs and define clear component interfaces. FAANG expects thoughtful API design.

No API design. Interfaces undefined. Cannot articulate how components communicate.	Basic API but poorly designed. Inconsistent. Missing error handling. Tight coupling.	Reasonable API design. RESTful or appropriate style. Main endpoints defined. Basic error handling.	Strong API design. Clean, consistent interfaces. Proper versioning. Comprehensive error handling. Loose coupling.	Expert API design. Elegant, intuitive interfaces. Backward compatible. Idempotent operations. Production-ready.

Feedback Templates:

- **Score 1-2:** Define clear APIs. Include: endpoints (GET/POST/PUT/DELETE), request/response formats, authentication, rate limiting, error codes, pagination. Think about how clients will actually use this.
- **Score 3:** Good API structure but consider: How do you version? How do you handle partial failures? Is this idempotent? What about pagination? Rate limiting? Authentication?
- **Score 4-5:** Excellent API design. Your use of [REST/GraphQL/gRPC] was appropriate for [reason]. The pagination strategy and error handling showed attention to developer experience.

Follow-up Questions:

- What does the API look like for this operation?
- How do you handle errors?
- How do you version the API?
- Is this operation idempotent?
- How do you handle pagination?
- What's your authentication strategy?

2.8 Tradeoff Analysis & Decision Making

Evaluates ability to identify, articulate, and navigate tradeoffs in system design. FAANG expects nuanced understanding that there's no perfect solution.

Cannot identify tradeoffs. Makes decisions without justification. One-dimensional thinking.	Identifies obvious tradeoffs only. Weak justification for decisions. Doesn't consider alternatives.	Identifies main tradeoffs. Reasonable justification. Considers 2-3 alternatives for key decisions.	Strong tradeoff analysis. Articulates multiple dimensions. Well-justified decisions. Considers long-term implications.	Expert tradeoff navigation. Nuanced analysis of competing concerns. Optimal decisions given constraints. Anticipates future tradeoffs.

Feedback Templates:

- **Score 1-2:** Every design decision has tradeoffs. For each choice, explain: What are the alternatives? What's the tradeoff (e.g., consistency vs availability, latency vs throughput)? Why is this the right choice for THIS system?
- **Score 3:** Good decision on [choice] but explain the tradeoff more clearly. What did you give up? When would you choose differently? How does this choice affect future evolution?

- **Score 4-5:** Excellent tradeoff analysis. Your explanation of choosing [X] over [Y] because [reason] despite [downside] showed mature engineering judgment. The awareness that [future scenario] might require revisiting was insightful.

Follow-up Questions:

- What's the tradeoff with this approach?
- What alternatives did you consider?
- Why this over [alternative]?
- What are you sacrificing with this decision?
- Under what conditions would you choose differently?
- How does this decision affect future scaling?

2.9 Communication & Diagram Clarity

Evaluates ability to communicate complex system designs clearly. FAANG expects candidates to explain their thinking to diverse audiences.

Cannot explain design. No diagrams or unreadable diagrams. Disorganized presentation.	Weak explanation. Confusing diagrams. Jumps around topics. Hard to follow.	Clear explanation of main concepts. Readable diagrams. Logical flow. Answers questions adequately.	Excellent communication. Clear, well-organized diagrams. Anticipates questions. Explains to any audience level.	Outstanding communication. Compelling narrative. Perfect diagrams. Could present to executives or junior engineers equally well.

Feedback Templates:

- **Score 1-2:** Your communication needs work. Structure your presentation: Start with requirements, then high-level diagram, then deep dives. Label all components. Show data flow with arrows. Explain as you draw.
- **Score 3:** Good communication but [diagram was cluttered / jumped between topics / didn't explain X]. Practice: narrate while drawing, use consistent notation, explain WHY not just WHAT.
- **Score 4-5:** Excellent communication. Your diagram was clear and well-organized. The way you explained [complex concept] simply was impressive. You anticipated and addressed questions proactively.

Follow-up Questions:

- Can you explain that component in more detail?
- Can you walk me through a request flow?
- What does this arrow represent?
- Can you summarize the key design decisions?
- How would you explain this to a PM?

3. SYSTEM DESIGN BUILDING BLOCKS

Candidates should know when and how to use these components:

Load Balancer	Distribute traffic, health checks, SSL termination	L4 vs L7, round-robin vs weighted, sticky sessions

API Gateway	Authentication, rate limiting, routing	Single entry point, can become bottleneck
CDN	Static content, edge caching, reduced latency	Cache invalidation, cost at scale
Cache (Redis/Memcached)	Reduce DB load, sub-ms latency	Cache invalidation, consistency, cold start
Message Queue (Kafka/ SQS)	Async processing, decoupling, buffering	Ordering, exactly-once, dead letter queues
SQL Database	ACID, complex queries, joins	Scaling limits, schema rigidity
NoSQL (DynamoDB/ Cassandra)	Horizontal scale, flexible schema	Eventual consistency, limited queries
Search (Elasticsearch)	Full-text search, analytics	Indexing lag, resource intensive
Blob Storage (S3)	Large files, cheap storage	Eventual consistency, egress costs
Rate Limiter	Prevent abuse, fair usage	Algorithm choice, distributed coordination

4. RED FLAGS

Starts drawing without gathering requirements	Requirements capped at 2.5
No capacity estimation / numbers	Estimation capped at 2.0
Single database with no scaling strategy	Scalability capped at 2.5
No consideration of failures	Reliability capped at 2.0
Cannot explain tradeoffs when asked	Tradeoff Analysis capped at 2.0
Says 'just use X' without justification	All technical dimensions reduced by 0.5
Design doesn't meet stated requirements	Architecture capped at 2.5
Cannot deep-dive on any component	Architecture capped at 3.0
Dismisses interviewer questions	Communication capped at 2.0
Copies a known design without adaptation	All dimensions capped at 3.0

5. ASSESSMENT TEMPLATE

Requirements Gathering	[X.X]
Capacity Estimation	[X.X]
High-Level Architecture	[X.X]
Database Design	[X.X]
Scalability & Performance	[X.X]
Reliability & Fault Tolerance	[X.X]
API Design	[X.X]
Tradeoff Analysis	[X.X]
Communication & Diagrams	[X.X]
OVERALL AVERAGE	**[X.X]**

System Designed: [Name]
Scale: [QPS, Storage, Users]

Key Decisions: [List 3-5]
Verdict: [PASS/FAIL]

6. LLM INSTRUCTIONS

Interview Flow

- Present open-ended problem (e.g., 'Design Twitter')
- Let candidate drive; intervene if they skip phases
- Ask about requirements if not clarified in 3 min
- Push for numbers if no estimation by 10 min
- Request deep-dive on 2-3 components
- Challenge decisions: 'Why not [alternative]?'
- Ask about failure modes for critical path

Probing Depth

- If surface-level: 'Can you go deeper on [component]?'
- If no tradeoffs: 'What's the downside of this approach?'
- If hand-waving: 'What specific technology would you use and why?'
- If ignoring failures: 'What happens when [component] goes down?'
- If no data model: 'What does the database schema look like?'

Scoring Rules

- 3.0 = adequate design, not impressive
- 5.0 = could be built and shipped to production
- Partial designs can still score 3.5+ if depth shown on covered parts
- Breadth without depth: max 3.5
- Depth without breadth: max 3.5
- Must demonstrate BOTH breadth and depth for 4.0+