System Design Mock Interview Guide: TinyURL

1. Clarify Requirements

Ask about custom aliases, user authentication, link expiration, analytics, and geographic distribution.

Example: 'Do we support custom short URLs? Should links expire?'

2. Define Use Cases

- Convert long URL to short URL
- Redirect from short to long URL
- (Optional) Expire links
- (Optional) Track analytics

3. Estimate Scale

- 100 million requests/day (90% reads, 10% writes)
- Store 10 million new URLs/day for 5 years approximately 18 billion entries
- Estimate ~11 TB of raw storage

4. API Design

- POST /tinyurl -> returns short URL
- GET /tinyurl/{shortCode} -> redirect
- (Optional) DELETE /tinyurl/{shortCode}
- (Optional) GET /tinyurl/{shortCode}/stats

5. Short Code Generation

- Use Base62 encoding
- Consider hash-based (e.g., MD5) or random generation
- Handle collisions with DB uniqueness checks

6. Storage Design

- Use RDBMS (PostgreSQL/MySQL)
- Table schema: id, short_code, long_url, created_at, expiration_time, click_count
- Use read replicas, then shard by id or short_code hash

7. Caching Layer

- Use Redis to cache hot shortCode -> longUrl mappings
- Add TTL and fallback to DB on cache miss

8. Scaling Strategy

- Load balancer (NGINX, ALB) to distribute traffic
- Scale app servers horizontally
- Read replicas and DB sharding for scale

9. Global Distribution

- Use CDN or DNS-based routing
- Replicate DBs across regions (CockroachDB, DynamoDB global tables)
- Cache per region and handle consistency trade-offs

10. Security & Abuse Prevention

- Rate limiting (IP-based, token bucket)
- Phishing/spam detection
- Expiration policies
- Support for user auth

11. Monitoring & Observability

- Track redirect latency and request volumes
- Log errors and monitor cache/DB health
- Set up alerts and dashboards

Wrap-Up

Summarize architecture and scaling path.

Mention optional features like analytics and user accounts.