System Design Fundamentals

1. What is scalability in system design?
Scalability is the ability of a system to handle increasing workload or growth in usage, data, or computational needs without a major drop in performance or requiring a complete redesign. A scalable system can efficiently adapt to rising demands by allocating more resources and maintaining reliability and speed.[geeksforgeeks+1]

2. How would you scale a simple URL shortener to support millions of users?
Scale using horizontal scaling—add more web servers and database nodes, use load balancers, and implement caching to distribute requests and data. Also, consider partitioning the database (sharding) and optimizing the hash function to avoid hotspots for frequently accessed short URLs.[statsig]

3. What's the difference between horizontal and vertical scaling?
Vertical scaling upgrades a single server's resources (CPU, RAM, disk), offering fast improvement but with physical and cost limits. Horizontal scaling adds more servers/nodes, distributing workload but needing careful design for load balancing and consistency, and enabling far bigger growth.[cloudzero+1]

4. Why is availability critical for a web service like TinyURL?
Availability ensures users can reliably access the service to redirect and shorten URLs. High availability directly impacts user trust and business potential, as downtime stops all redirects and new URL creation, hurting the core functionality and reputation of the service.[catchpoint+1]

5. How does redundancy improve system availability?
Redundancy adds backup systems—like multiple web or database servers—so if one fails, traffic automatically shifts to others. This avoids downtime and data loss, keeping the URL shortener operational even during hardware or software failures.[aws.amazon+2]

6. What's the basic trade-off between latency and throughput in a web service?
Latency measures response time (how quickly a single request is processed); throughput measures the number of requests processed per unit time. Lower latency is ideal for real-time experiences; higher throughput is key for bulk operations. Sometimes, optimizing one comes at the expense of the other—e.g., batching requests can increase throughput but add latency.[aws.amazon+1]

7. Give examples of user operations that are sensitive to latency in a URL shortener.
Redirection after clicking a short URL (should be quick), and short URL creation (instant feedback) are latency-sensitive. Users expect results in milliseconds for these actions, impacting perceived service quality.[redhat]

8. What techniques ensure high availability when a database goes down?
Use replication, failover strategies, redundant servers, backup databases, and automated monitoring. Load balancing and automated recovery mechanisms help reroute traffic or switch databases on failure, reducing downtime.[ibm+1]

System Design Trade-offs

9. When would you choose consistency over availability? Give a URL shortener scenario.
If it's vital that newly created short URLs are instantly accessible and never yield "not found," prioritize consistency—ensuring every read sees all writes, but possibly serving fewer users (downtime during errors). For example, on heavy traffic, sacrificing availability might mean users see delays but never stale/missing data.designgurus+1

10. What types of failures could affect only availability but not consistency?
Network failures or server crashes can make part of the system unavailable (users cannot access service), but once recovered, the data remains consistent and intact.

11. Why might you denormalize data in a highly scalable system?
Denormalization involves duplicating data to cut down on joins, speeding up reads and writes under heavy load, at the cost of storage space and possible update complexity. For a URL shortener, denormalized lookup tables can improve redirect and analytics performance.

12. How could sharding help in scaling the database for a busy URL shortener?
Sharding splits the database across multiple machines, each handling a subset of data (e.g., by short code prefix). This distributes the load, prevents bottlenecks, and enables scale to billions of URLs without overloading a single server.statsig

---

URL Shortener Technical Questions

13. Explain the main steps for shortening a URL.
User submits a long URL. Server generates a unique short code (using hashing or a sequence), saves the mapping in the database, and returns the short URL. When accessed, the server looks up the short code, retrieves original URL, and redirects the user.designgurus

14. How do you generate a unique short code for each URL?
Use hashing algorithms (MD5, SHA), base62 encoding, or incrementing counters; apply collision checks to avoid duplicate codes. Hashing with salts or random strings can reduce predictability.

15. What hashing techniques can you use for URL shortening?
MD5, SHA-256, or base62 encoding after hashing. You may take the first few characters of the hash, but it's important to check for collisions.

16. What happens if two different URLs produce the same hash (collision)?
Detect collision on insertion. Either try a different hash, add randomness, or append a unique sequence number to resolve it. If using a database, check if the code already exists before completing.designgurus

17. How can you ensure that generated short links aren't easily predictable?
Use random or cryptographically secure hash functions, add salt, time, or unique entropy per request. Avoid simple sequential codes to reduce risk of enumeration attacks.designgurus

18. What happens if someone tries to shorten the same URL twice?
If designed to minimize duplication, check if the URL already has a short code and return the existing one. Alternatively, each request could generate a new code. This is a business decision impacting UX and analytics.

19. What security issues might arise in a public URL shortening service?
Phishing, malicious redirections, spam URLs, privacy (users can mask real links). Add abuse monitoring, link previews, blacklist, and authentication for users.

20. How would you expire or delete URLs after a given time?
Add an expiration timestamp to database entries, and periodically purge expired URLs. Schedule cleanup jobs that scan and delete stale records.

21. How do you track analytics (like click counts) for shortened URLs?
Update a counter field in the database per access and optionally store access logs with time/user info for further analysis.

Database (SQL vs NoSQL) Choices

22. What would a basic SQL schema for a URL shortener look like?
Table: `urls`
Columns:

- `id` (primary key)
- `short_code` (unique, indexed)
- `long_url` (text)
- `created_at`
- `expires_at` (optional)
- `click_count` (optional).designgurus

23. What would a basic NoSQL (document-based) schema look like?
Document example:

```
json
{
  "short_code": "abc123",
  "long_url": "https://www.example.com/page",
  "created_at": "2025-10-05",
  "expires_at": "2025-12-05",
  "click_count": 107
}
```

Each document holds mapping and metadata.

24. Compare pros and cons of using SQL for this use case.
Pros:

- Strong consistency
- Support for complex queries, joins, and integration with analytics.designgurus
  Cons:
- Scalability and performance limits for huge data sets or high write load.

25. Compare pros and cons of using NoSQL here.
Pros:

- Easily scales out (horizontal scaling)
- Good for simple key-value/payload storage, rapid access.designgurus
  Cons:
- Possible inconsistency or eventual consistency
- Fewer complex querying options.

26. How would you handle database migrations if traffic increases dramatically?
Implement sharding or partitioning, migrate to more distributed databases, and use data replication. Gradually redirect traffic to new nodes, and run tests to ensure no data loss or downtime.

27. How do you prevent duplicate short codes in the database?
Use a unique constraint/index on the `short_code` column and always check for existence before insertion. Retry hashing or use random generation if a collision is detected.

REST API Design & Best Practices

28. What would the REST endpoint for creating a short URL be?
`POST /api/shorten`
Request: `{ "long_url": "https://www.example.com" }`
Response: `{ "short_code": "abc123", "short_url": "http://short.ly/abc123" }`

29. What would be a RESTful way to handle redirection for a short code?
`GET /{short_code}`
Server looks up code and responds with HTTP 301/302 redirect to the long URL.

30. How would you design an endpoint for analytics or stats retrieval?
`GET /api/stats/{short_code}`
Returns: `{ "click_count": 107, "created_at": "...", "last_accessed": "..." }`

31. What status code should you return if a short code does not exist?
HTTP 404 Not Found.

32. Why use POST vs GET in your API design for this service?
POST is for creating new resources (short URLs) and should not be idempotent; GET is for reading existing resources (redirections or data) and is always safe and repeatable.

33. What data validations would you enforce in your API?
Check valid URL format, length limits, allowed protocols (e.g., HTTP/HTTPS only), prevent duplicate requests, validate expiration dates and optional parameters.

Bonus — Design, Optimizations, and Reflection

34. How can you cache frequent redirects for even lower latency?
Use an in-memory cache (like Redis or Memcached) for popular short codes, reducing database lookups and serving instant redirects.

35. How do you handle the case when the most popular short URLs slow down the database (hotspotting)?
Introduce caching, use load balancers, or partition hot keys across cache/database nodes to distribute load. Monitor for "hot" codes and scale resources for them.designgurus

36. If you support custom short URLs (user-chosen), how would your design change?
Add a field for custom alias, check for uniqueness before assignment, and handle reserved keywords/abusive alias attempts. Adjust validation and API.

37. How would your system change if you had to comply with the EU's GDPR data deletion requirements?
Implement explicit delete functionality for users, track metadata to identify personal data, and ensure all logs or analytics data related to user are ERASED on request.designgurus

38. Explain the end-to-end flow for a user creating and using a short link, touching on each component.
User submits long URL via API; server generates unique (random/custom) code, checks for collision, stores mapping in DB, and returns short URL. When someone uses the short link, server looks up code—first checks cache, then DB—redirects user, then increments click analytics. If expired or deleted, returns 404. System checks validations, applies security measures, and logs activity for monitoring