Back-End Developer

Interview Guide

Prepared from the perspective of a Hiring Manager at a Top IT Company

Covers Core Programming, Databases, APIs, System Design, Security, DevOps & Behavioral Questions

1. Core Programming & Language Fundamentals

Q1. What is the difference between synchronous and asynchronous programming?

Synchronous code executes line by line, blocking further execution until the current operation finishes. Asynchronous code allows the program to continue running while a long-running operation (like a network call or file read) completes in the background, using callbacks, promises, or async/await, and is essential for building responsive, non-blocking servers.

Q2. What is the difference between multithreading and multiprocessing?

- Multithreading: Multiple threads run within a single process, sharing the same memory space — lightweight but requires careful synchronization to avoid race conditions.
- Multiprocessing: Multiple independent processes run in parallel, each with its own memory space — more overhead but safer isolation and true parallelism across CPU cores.

Q3. Explain the concept of garbage collection.

Garbage collection is an automatic memory management process where the runtime identifies and frees memory occupied by objects that are no longer reachable/referenced by the program, preventing memory leaks without requiring manual deallocation (as in C/C++).

Q4. What are design patterns? Name a few commonly used in backend systems.

Design patterns are reusable solutions to common software design problems.

- Singleton: Ensures a class has only one instance (e.g., a database connection pool).
- Factory: Creates objects without specifying the exact class to instantiate.
- Observer: Notifies dependent objects automatically when state changes (used in event-driven systems).
- Repository: Abstracts data access logic away from business logic.
- Strategy: Encapsulates interchangeable algorithms behind a common interface.

Q5. What is the difference between an interface and an abstract class?

An interface defines a contract (method signatures) with no implementation, and a class can implement multiple interfaces. An abstract class can contain both implemented and unimplemented methods, and a class can extend only one abstract class. Interfaces favor composition and flexibility; abstract classes are useful when sharing common code across related classes.

2. Databases (SQL & NoSQL)

Q6. What is the difference between SQL and NoSQL databases?

SQL databases (MySQL, PostgreSQL) are relational, use structured schemas with tables and enforce ACID properties, ideal for structured data with complex relationships. NoSQL databases (MongoDB, Cassandra, Redis) are non-relational, offer flexible schemas, and are optimized for scalability and specific data models like documents, key-value pairs, or graphs — often trading strict consistency for availability and partition tolerance.

Q7. What are ACID properties in a database?

- Atomicity: A transaction is all-or-nothing — it either fully completes or fully rolls back.
- Consistency: A transaction brings the database from one valid state to another, respecting all constraints.
- Isolation: Concurrent transactions don't interfere with each other's intermediate states.
- Durability: Once a transaction is committed, it remains so even after a system failure.

Q8. What is database indexing and how does it improve performance?

An index is a data structure (commonly a B-tree) that allows the database to locate rows quickly without scanning the entire table. It significantly speeds up read/query performance at the cost of additional storage and slower writes (since indexes must be updated on every insert/update/delete). Indexes should be added on columns frequently used in WHERE, JOIN, or ORDER BY clauses.

Q9. What is database normalization? What are the common normal forms?

Normalization is the process of organizing tables to reduce data redundancy and improve data integrity by dividing large tables into smaller related ones.

- 1NF: Eliminate repeating groups; ensure atomic column values.
- 2NF: Remove partial dependencies (non-key attributes depend on the whole primary key).
- 3NF: Remove transitive dependencies (non-key attributes depend only on the primary key, not on other non-key attributes).

Q10. When would you denormalize a database?

Denormalization intentionally introduces redundancy to optimize read-heavy workloads, reducing the need for expensive joins. It's common in reporting/analytics systems, read replicas, or high-traffic APIs where read performance matters more than write efficiency or storage cost.

Q11. Explain database transactions and isolation levels.

A transaction is a sequence of operations executed as a single logical unit of work. Isolation levels control how transaction changes become visible to other concurrent transactions:

- Read Uncommitted: Transactions can see uncommitted changes from others (dirty reads possible).
- Read Committed: Only committed changes are visible (default in many databases).
- Repeatable Read: Guarantees the same data is read if queried multiple times within a transaction.
- Serializable: Strictest level, transactions execute as if run one at a time sequentially.

Q12. What is a database deadlock and how do you prevent it?

A deadlock occurs when two or more transactions hold locks that the others need, and each waits indefinitely for the other to release theirs. Prevention strategies include acquiring locks in a consistent order, keeping transactions short, using timeouts, and letting the database's deadlock detector automatically roll back one of the transactions.

Q13. What is database sharding?

Sharding is a horizontal partitioning technique where a large database is split across multiple servers (shards), each holding a subset of the data (e.g., by user ID range or region). It improves scalability and performance for very large datasets but adds complexity for cross-shard queries and joins.

Q14. What is the difference between a primary key, foreign key, and unique key?

- Primary Key: Uniquely identifies each row in a table; cannot be null, only one per table.
- Foreign Key: A column that references the primary key of another table, enforcing referential integrity.
- Unique Key: Ensures all values in a column are distinct, but unlike primary keys, it can allow one null value and a table can have multiple unique keys.

Q15. What is connection pooling and why is it important?

Connection pooling maintains a cache of reusable database connections rather than opening/closing a new one for every request. This reduces the overhead of establishing connections (which is expensive), improves throughput, and prevents exhausting the database's maximum connection limit under high load.

3. APIs & Web Services

Q16. What is REST and what are its key principles?

REST (Representational State Transfer) is an architectural style for designing networked applications using stateless communication over HTTP.

- Statelessness: Each request contains all information needed; the server stores no client session state.
- Resource-based URLs: Endpoints represent resources (nouns), not actions (e.g., /users/123, not /getUser).
- Standard HTTP methods: GET, POST, PUT, PATCH, DELETE map to CRUD operations.
- Uniform interface & statelessness enable scalability and cacheability.

Q17. What is the difference between REST and GraphQL?

REST exposes multiple fixed endpoints, each returning a predefined data shape, which can lead to over-fetching or under-fetching of data. GraphQL exposes a single endpoint where clients specify exactly what data they need via queries, reducing over-fetching but adding complexity in caching and query optimization on the server.

Q18. Explain the difference between PUT, PATCH, and POST.

- POST: Creates a new resource; not idempotent (calling it twice creates two resources).
- PUT: Replaces an entire resource; idempotent (calling it multiple times has the same effect as once).
- PATCH: Partially updates a resource; may or may not be idempotent depending on implementation.

Q19. What is idempotency and why does it matter in API design?

An idempotent operation produces the same result no matter how many times it's performed. It matters because clients may retry failed requests (due to network issues) — idempotent endpoints ensure retries don't cause duplicate side effects, like double-charging a customer.

Q20. What are common HTTP status codes and what do they mean?

- 200 OK: Request succeeded.
- 201 Created: Resource successfully created.
- 400 Bad Request: Client sent invalid data.
- 401 Unauthorized: Authentication required or failed.
- 403 Forbidden: Authenticated but not authorized to access the resource.
- 404 Not Found: Resource doesn't exist.
- 429 Too Many Requests: Rate limit exceeded.
- 500 Internal Server Error: Unexpected server-side failure.

Q21. How do you version an API?

Common strategies include URL versioning (/api/v1/users), header versioning (Accept: application/vnd.myapi.v1+json), or query parameter versioning (?version=1). URL versioning is the most common due to its simplicity and visibility, though header versioning keeps URLs cleaner for long-term API evolution.

Q22. What is gRPC and how does it differ from REST?

gRPC is a high-performance RPC framework built on HTTP/2 that uses Protocol Buffers for serialization, enabling faster, more compact binary communication compared to REST's typically text-based JSON over HTTP/1.1. It supports bidirectional streaming and strongly typed contracts, making it popular for internal microservice-to-microservice communication, while REST remains more common for public-facing APIs due to its simplicity and human-readability.

Q23. What is API rate limiting and how would you implement it?

Rate limiting restricts how many requests a client can make within a time window, protecting the server from abuse or overload. Common algorithms include Token Bucket, Leaky Bucket, and Fixed/Sliding Window Counters, often implemented using an in-memory store like Redis to track request counts per client (by API key or IP).

Q24. What is webhooks and how do they differ from polling?

Polling means a client repeatedly asks the server if new data is available. Webhooks reverse this: the server proactively sends an HTTP POST request to a client-provided URL whenever an event occurs, reducing unnecessary requests and providing near real-time updates.

4. System Design & Architecture

Q25. What is the difference between monolithic and microservices architecture?

A monolith is a single deployable application containing all business logic, which is simpler to develop and deploy initially but harder to scale and maintain as it grows. Microservices split the application into small, independently deployable services, each owning its own data and business capability, improving scalability and team autonomy at the cost of increased operational complexity (networking, monitoring, data consistency).

Q26. How would you design a URL shortener (like bit.ly)?

Key considerations: generating unique short codes (e.g., base62 encoding of an auto-incrementing ID, or hashing), a fast key-value store mapping short codes to original URLs (Redis or a NoSQL DB), handling redirects with HTTP 301/302, ensuring high read throughput via caching, and considering analytics tracking and expiration policies.

Q27. What is load balancing and what algorithms are commonly used?

A load balancer distributes incoming traffic across multiple servers to prevent any single server from being overwhelmed, improving availability and scalability.

- Round Robin: Requests distributed sequentially across servers.
- Least Connections: Routes to the server with the fewest active connections.
- IP Hash: Routes based on client IP, useful for session stickiness.
- Weighted algorithms: Account for servers with different capacities.

Q28. What is caching and what are common caching strategies?

Caching stores frequently accessed data in fast storage (memory) to reduce load on the primary data source and improve response times.

- Cache-aside (lazy loading): Application checks cache first, loads from DB on miss, then populates cache.
- Write-through: Data is written to cache and DB simultaneously, keeping them in sync.
- Write-behind: Data is written to cache first and asynchronously flushed to DB later.
- TTL (Time-to-live) expiration: Prevents stale data by expiring entries after a set duration.

Q29. What is the CAP theorem?

The CAP theorem states that a distributed system can only guarantee two of the following three properties simultaneously: Consistency (all nodes see the same data at the same time), Availability (every request receives a response), and Partition Tolerance (the system continues to operate despite network failures between nodes). Since network partitions are unavoidable in distributed systems, real systems must choose a trade-off between consistency and availability during a partition.

Q30. What is a message queue and why would you use one?

A message queue (e.g., RabbitMQ, Kafka, SQS) enables asynchronous communication between services by decoupling the producer of a task from its consumer. It's used for handling background jobs, smoothing out traffic spikes, retrying failed operations, and enabling event-driven architectures without services needing to know about each other directly.

Q31. How would you design a scalable notification system?

Discuss: an event producer publishing notification events to a message queue, worker services consuming from the queue to send notifications via email/SMS/push providers, a database to track notification status and preferences, retry/backoff logic for failed deliveries, and horizontal scaling of consumers to handle high volume.

Q32. What is the difference between horizontal and vertical scaling?

- Vertical scaling (scale up): Adding more resources (CPU, RAM) to an existing server — simpler but has a hardware ceiling and a single point of failure.
- Horizontal scaling (scale out): Adding more servers/instances to distribute load — more complex (requires load balancing, statelessness) but offers near-limitless scalability and better fault tolerance.

Q33. What is eventual consistency?

Eventual consistency is a model used in distributed systems where, after an update, all replicas will converge to the same value over time, but may temporarily return stale data. It's a trade-off many NoSQL databases make to achieve higher availability and partition tolerance, acceptable for use cases like social media likes but not for financial transactions.

Q34. How would you design a rate limiter for an API gateway?

Discuss maintaining request counters per client (in Redis for shared state across servers), choosing an algorithm (token bucket is common for allowing bursts while enforcing an average rate), returning 429 status codes with a Retry-After header when limits are exceeded, and applying different limits per subscription tier.

5. Security

Q35. What is the difference between authentication and authorization?

Authentication verifies who a user is (e.g., login with credentials). Authorization determines what an authenticated user is allowed to do (e.g., access permissions, roles). Authentication always happens first; authorization decisions depend on the authenticated identity.

Q36. What is JWT (JSON Web Token) and how does it work?

A JWT is a compact, self-contained token consisting of a header, payload, and signature, used to securely transmit claims (like user identity) between parties. The server signs the token with a secret or private key; on each request, the server verifies the signature to trust the claims without needing to look up a session in a database, enabling stateless authentication.

Q37. What is the difference between session-based and token-based authentication?

Session-based authentication stores session state on the server (often in memory or a database) and gives the client a session ID cookie; it requires server-side lookups and doesn't scale as easily across multiple servers without shared session storage. Token-based authentication (like JWT) is stateless — the server verifies the token's signature without storing session data, making it easier to scale horizontally, though token revocation before expiry is harder to manage.

Q38. What is SQL Injection and how do you prevent it?

SQL Injection is an attack where malicious SQL is inserted into a query through unsanitized user input, potentially exposing or corrupting data. Prevention: always use parameterized queries or prepared statements (never string-concatenate user input into SQL), use ORM libraries that escape input by default, and apply the principle of least privilege to database accounts.

Q39. What is CSRF and how do you protect against it?

Cross-Site Request Forgery tricks an authenticated user's browser into submitting unwanted requests to a site they're logged into. Protection includes using anti-CSRF tokens (unique per session/form), checking the Origin/Referer headers, and setting cookies with SameSite=Strict or Lax attributes.

Q40. How do you securely store passwords?

Never store passwords in plain text. Use a slow, salted, one-way hashing algorithm designed for passwords (bcrypt, scrypt, or Argon2), which incorporates a unique salt per password and a configurable work factor to resist brute-force and rainbow-table attacks.

Q41. What is HTTPS and why is it important?

HTTPS encrypts data in transit between client and server using TLS/SSL, preventing eavesdropping, tampering, and man-in-the-middle attacks. It also verifies server identity via certificates issued by trusted Certificate Authorities, which is essential for protecting credentials, tokens, and sensitive data over the network.

Q42. What is the principle of least privilege?

It states that any user, process, or service should be granted only the minimum permissions necessary to perform its function — nothing more. This limits the potential damage from compromised credentials or bugs, and is a foundational concept in access control design (e.g., database users, IAM roles, API scopes).

6. Performance, Testing & DevOps Basics

Q43. How would you identify and fix a performance bottleneck in a backend service?

Approach: use monitoring/APM tools (e.g., New Relic, Datadog) to identify slow endpoints, profile the code to find CPU/memory hotspots, check database query performance (missing indexes, N+1 queries), review external API call latencies, and consider caching or asynchronous processing for expensive operations. Always measure before and after any optimization to confirm improvement.

Q44. What is the N+1 query problem and how do you avoid it?

The N+1 problem occurs when code fetches a list of N records, then makes an additional query for each record to fetch related data, resulting in N+1 total queries instead of one. It's avoided using eager loading/joins (e.g., JOIN FETCH in JPA, includes in ActiveRecord) to retrieve related data in a single query.

Q45. What is the difference between unit testing, integration testing, and end-to-end testing?

- Unit testing: Tests individual functions/components in isolation, often using mocks for dependencies.
- Integration testing: Tests how multiple components/modules work together (e.g., service + database).
- End-to-end (E2E) testing: Tests the entire application flow from the user's perspective, simulating real-world usage.

Q46. What is CI/CD and why is it important?

Continuous Integration is the practice of frequently merging code changes into a shared repository, with automated builds and tests running on every commit to catch issues early. Continuous Deployment/Delivery automates releasing that code to staging or production environments. Together, CI/CD reduces manual errors, shortens release cycles, and improves software quality and team velocity.

Q47. What is containerization and how does Docker help in backend development?

Containerization packages an application with all its dependencies and configuration into a lightweight, portable unit (a container) that runs consistently across different environments. Docker is the most popular containerization tool — it eliminates "it works on my machine" issues, simplifies deployment, and pairs well with orchestration tools like Kubernetes for scaling containers across a cluster.

Q48. What is the difference between logging, monitoring, and alerting?

- Logging: Recording discrete events (errors, requests, state changes) for later debugging and auditing.
- Monitoring: Continuously tracking system metrics (CPU, memory, latency, error rates) to understand health over time.
- Alerting: Automatically notifying engineers when monitored metrics cross defined thresholds, enabling fast incident response.

Q49. How do you handle backward compatibility when deploying changes to a live API?

Strategies include: never removing or renaming existing fields (add new ones instead), using API versioning for breaking changes, deploying with feature flags to gradually roll out new behavior, and running database migrations in a backward-compatible way (e.g., expand-contract pattern: add new columns, migrate data, then remove old columns in a later release).

7. Behavioral / Soft Skill Questions (Common in Top Companies)

Q50. Tell me about a time you had to optimize a slow or failing system in production. What was your approach?

(Answer should demonstrate a calm, methodical incident-response approach: triaging severity, checking logs/metrics, forming and testing hypotheses, communicating status to stakeholders, and following up with a post-mortem and preventive fix.)

Q51. Describe a time you had to make a trade-off between shipping quickly and writing perfect code.

(Look for pragmatic judgment: understanding business context, communicating technical debt clearly, and having a plan to revisit and improve the solution later rather than leaving it unaddressed indefinitely.)

Q52. How do you approach designing a new backend feature from scratch?

(Look for a structured process: clarifying requirements, considering data models and API contracts, thinking about scalability/security early, writing tests, and getting design feedback from peers before heavy implementation.)

Interview Tips for Candidates

- Think out loud about trade-offs (performance vs. simplicity, consistency vs. availability) rather than just naming a solution.
- For system design questions, clarify scale and constraints before diving into an architecture.
- Be ready to justify database and technology choices with reasoning, not just familiarity.
- Relate answers to real production experience — incidents, migrations, or scaling challenges — wherever possible.