gRPC - Practice Sheet

10 conceptual Q&A API Technologies

Q1. What is gRPC and what API communication problem does it solve?

A: gRPC defines a protocol or standard for structured communication between services. It addresses concerns like schema definition, payload format, versioning, or transport efficiency that plain HTTP with ad-hoc JSON does not solve on its own.

Q2. How does gRPC define its schema or contract?

A: gRPC uses an IDL or schema language to declare types, operations, and their inputs and outputs. This schema is the single source of truth from which documentation, validation, and client SDKs can be auto-generated.

Q3. How does gRPC handle authentication?

A: gRPC APIs commonly use Bearer tokens (JWT/OAuth2) in the Authorization header or API keys in custom headers. Transport-level security (TLS) is always required. Additional mechanisms (mTLS, HMAC) apply to high-trust scenarios.

Q4. How does gRPC handle API versioning?

A: Common versioning strategies include URL path (/v1/users), request headers (Accept: application/vnd.api+v2+json), and query params (?version=2). gRPC prefers one strategy and maintains backward compatibility within a major version.

Q5. How does gRPC handle errors and status codes?

A: gRPC uses HTTP status codes (200 OK, 400 Bad Request, 401 Unauthorized, 404 Not Found, 500 Internal Server Error) combined with a structured error body containing a code, message, and optional details field.

Q6. How do you implement pagination in a gRPC API?

A: Cursor-based pagination (next_cursor token) is preferred for large or frequently-updated datasets because it is stable. Offset-based pagination (page, limit) is simpler but can skip or duplicate rows if data changes between requests.

Q7. How do you document a gRPC API?

A: gRPC APIs use an API specification (OpenAPI, AsyncAPI, GraphQL SDL) to generate interactive documentation (Swagger UI, ReDoc, GraphiQL). Good docs include examples, error codes, and authentication instructions.

Q8. How do you test a gRPC API?

A: Contract testing (Pact) verifies the provider matches the consumer's schema. Integration tests call real endpoints against a test environment. Load tests (k6, Gatling) verify performance under production-like concurrency.

Q9. How do you protect a gRPC API from abuse?

A: Rate limiting (tokens per IP per minute) prevents DoS. Input validation rejects malformed requests before they reach business logic. Output filtering (response schema) prevents accidental data leakage. Monitor for anomalous usage patterns.

Q10. What monitoring and observability should a gRPC API have?

A: Instrument request count, latency histograms, and error rate with Prometheus or a cloud metrics system. Add distributed tracing with OpenTelemetry. Log structured request/response data. Set SLOs and alert when SLIs breach them.