100+ API Testing Interview Questions with Answers

Sujal Rajput

Fundamentals (Questions 1-20)

1. What is an API and why is API testing important?

An API (Application Programming Interface) is a set of protocols and tools that allows different software applications to communicate with each other. API testing is important because it validates the business logic, ensures data accuracy, tests security vulnerabilities, and verifies that APIs work correctly before UI development.

2. What are the main differences between API testing and UI testing?

- API testing focuses on business logic layer, while UI testing focuses on presentation layer
- API testing is faster and can be automated earlier in development
- API testing doesn't require a GUI and tests data directly
- API testing provides better code coverage and can catch issues earlier
- UI testing is more prone to breaking with interface changes

3. Explain the difference between SOAP and REST APIs.

SOAP (Simple Object Access Protocol):

- Protocol-based with strict standards
- Uses only XML for data exchange
- Built-in error handling (WS-Security)
- More bandwidth-intensive
- Stateful operations

REST (Representational State Transfer):

- Architectural style with flexible guidelines
- Supports multiple formats (JSON, XML, HTML, plain text)
- Uses HTTP status codes for errors
- Lightweight and faster
- Stateless operations

4. What are the common HTTP methods used in RESTful APIs?

- GET: Retrieve data from server
- POST: Create new resources
- PUT: Update/replace entire resource

- PATCH: Partially update resource
- DELETE: Remove resources
- HEAD: Similar to GET but returns only headers
- OPTIONS: Describes communication options for target resource

5. What is the difference between PUT and PATCH methods?

PUT: Replaces the entire resource. If you send a PUT request with partial data, missing fields may be set to null or default values.

PATCH: Partially updates a resource. Only the fields included in the request are updated; other fields remain unchanged.

6. What are the standard HTTP status codes and what do they represent?

- 1xx: Informational responses
- 2xx: Success (200 OK, 201 Created, 204 No Content)
- 3xx: Redirection (301 Moved Permanently, 302 Found)
- 4xx: Client errors (400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found)
- 5xx: Server errors (500 Internal Server Error, 502 Bad Gateway, 503 Service Unavailable)

7. What is the difference between authentication and authorization in APIs?

Authentication: Verifies "who you are" - validates user identity through credentials like username/password, tokens, or API keys.

Authorization: Determines "what you can do" - controls access to resources based on permissions and roles after authentication is successful.

8. Explain what API endpoints are.

API endpoints are specific URLs where APIs can access resources. Each endpoint represents a specific function or resource in the API. Example: https://api.example.com/users/123 where /users/123 is the endpoint.

9. What is JSON and why is it commonly used in APIs?

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It's commonly used because:

- Human-readable and easy to understand
- Lightweight and faster to parse
- Native support in JavaScript
- Less verbose than XML
- Widely supported across languages

10. What is XML and how does it differ from JSON?

XML (eXtensible Markup Language) is a markup language for encoding documents. Differences:

- XML is more verbose with opening/closing tags
- JSON is more lightweight and faster to parse
- XML supports attributes and namespaces
- XML has better schema validation support
- JSON is easier to read and write

11. What are request headers and response headers?

Request headers: Metadata sent by client to server (Authorization, Content-Type, Accept, User-Agent).

Response headers: Metadata sent by server to client (Content-Type, Content-Length, Set-Cookie, Cache-Control).

12. What is a payload in API testing?

A payload is the actual data sent in the body of an HTTP request or response, typically in POST, PUT, or PATCH requests. It contains the information being transmitted, usually in JSON or XML format.

13. Explain the concept of API versioning and why it's important.

API versioning allows multiple versions of an API to coexist, enabling backward compatibility while introducing new features. Methods include:

- URI versioning: /api/v1/users
- Query parameter: /api/users?version=1
- Header versioning: Accept: application/vnd.company.v1+json

14. What are query parameters in an API request?

Query parameters are key-value pairs appended to the URL after a ? symbol, used to filter, sort, or paginate data. Example: /users?age=25&city=NYC

15. What is the difference between path parameters and query parameters?

Path parameters: Part of the URL path, required, identify specific resources. Example: /users/123

Query parameters: Optional, appear after ?, used for filtering/sorting. Example: /users?role=admin

16. What are the main components of an HTTP request?

- Request Line (method, URI, HTTP version)
- Headers (metadata)
- Body (payload for POST/PUT/PATCH)

17. What are the main components of an HTTP response?

- Status Line (HTTP version, status code, status message)

- Headers (metadata)
- Body (response data)

18. What is Content-Type header and why is it important?

Content-Type specifies the media type of the resource or data being sent. It tells the server how to parse the request body and tells the client how to interpret the response. Example: Content-Type: application/json

19. What is the Accept header used for?

The Accept header tells the server what content types the client can understand and prefers to receive in the response. Example: Accept: application/json

20. Explain what idempotency means in the context of REST APIs.

Idempotency means that making multiple identical requests has the same effect as making a single request. GET, PUT, DELETE, HEAD, and OPTIONS are idempotent, while POST is not.

API Testing Types and Approaches (Questions 21-35)

21. What are the different types of API testing?

- Functional testing: Validates API functions correctly
- Load testing: Tests performance under expected load
- Security testing: Identifies vulnerabilities
- Integration testing: Tests API integration with other systems
- Validation testing: Verifies responses, data types, error codes
- UI testing: Tests API through user interface
- Fuzz testing: Sends random/invalid data to test stability
- Negative testing: Tests error handling with invalid inputs

22. What is the difference between positive and negative testing?

Positive testing: Validates API works correctly with valid inputs and expected use cases.

Negative testing: Ensures API handles invalid inputs, edge cases, and error conditions gracefully.

23. What is contract testing in APIs?

Contract testing verifies that two services (consumer and provider) agree on the structure of their communication. It ensures the API provider delivers what the consumer expects, preventing integration issues.

24. Explain API integration testing.

API integration testing validates that multiple APIs or services work together correctly, testing data flow between different components, third-party integrations, and end-to-end workflows.

25. What is API mocking and why is it useful?

API mocking creates simulated API responses without calling the actual API. It's useful for:

- Testing when API is under development
- Avoiding dependencies on external services
- Creating specific test scenarios
- Faster test execution
- Cost reduction

26. What tools do you use for API testing?

- Postman: Manual and automated testing
- REST Assured: Java-based automation framework
- SoapUI: SOAP and REST testing
- JMeter: Performance and load testing
- Karate: BDD-style API testing
- Swagger/OpenAPI: Documentation and testing
- Newman: Command-line runner for Postman
- Insomnia: API client and testing tool

27. What is REST Assured and how is it used?

REST Assured is a Java library for testing RESTful APIs. It provides a domain-specific language for writing readable API tests with features like request/response validation, authentication support, and JSON/XML parsing.

28. How do you perform load testing on APIs?

Load testing involves:

- Defining expected concurrent users
- Creating realistic test scenarios
- Using tools like JMeter, LoadRunner, or Gatling
- Gradually increasing load to find breaking points
- Monitoring response times, throughput, and error rates
- Analyzing bottlenecks and performance degradation

29. What is API security testing?

API security testing identifies vulnerabilities like:

- Authentication/authorization flaws
- SQL injection attacks
- Cross-Site Scripting (XSS)

- Man-in-the-middle attacks
- Rate limiting issues
- Sensitive data exposure
- Improper error handling revealing system information

30. What is the difference between smoke testing and sanity testing for APIs?

Smoke testing: Quick preliminary tests to verify basic API functionality works before detailed testing.

Sanity testing: Focused testing after bug fixes or minor changes to verify specific functionality works correctly.

31. What is end-to-end API testing?

End-to-end testing validates complete workflows involving multiple APIs, testing entire business processes from start to finish, including data flow, integrations, and user scenarios.

32. Explain regression testing in API context.

Regression testing ensures that new changes or updates don't break existing functionality. It involves re-running previously passed test cases after modifications to verify system stability.

33. What is API performance testing?

Performance testing evaluates API speed, scalability, and stability under various conditions, measuring response times, throughput, resource utilization, and identifying bottlenecks.

34. What is boundary value analysis in API testing?

Boundary value analysis tests API behavior at the edges of input ranges, testing minimum values, maximum values, and just beyond boundaries to catch edge case errors.

35. What is exploratory testing for APIs?

Exploratory testing involves simultaneously learning, designing, and executing tests without predefined scripts, discovering unexpected behaviors and edge cases through creative investigation.

Authentication and Authorization (Questions 36-45)

36. What are the common authentication mechanisms used in APIs?

- Basic Authentication: Username and password encoded in Base64
- Bearer Token Authentication: Token passed in Authorization header
- API Key Authentication: Unique key for each consumer
- OAuth 2.0: Token-based authorization framework
- JWT (JSON Web Tokens): Self-contained tokens with claims
- OAuth 1.0: Signature-based authentication
- Digest Authentication: Encrypted credentials

37. What is OAuth 2.0 and how does it work?

OAuth 2.0 is an authorization framework allowing third-party applications to access user resources without exposing credentials. Flow:

1. Client requests authorization from resource owner
2. Client receives authorization grant
3. Client exchanges grant for access token from authorization server
4. Client uses access token to access protected resources

38. Explain JWT (JSON Web Token) authentication.

JWT is a compact, self-contained token containing claims about a user. Structure:

- Header: Token type and algorithm
- Payload: Claims/data
- Signature: Verification hash

JWTs are stateless, scalable, and can be verified without database lookup.

39. What is API Key authentication and when should it be used?

API keys are unique identifiers passed in headers or query parameters to identify and authenticate API consumers. Best for:

- Server-to-server communication
- Internal APIs
- Simple authentication needs Not recommended for user authentication or sensitive operations.

40. How do you test API authentication?

- Test with valid credentials (should succeed)
- Test with invalid credentials (should fail with 401)
- Test without credentials (should fail)
- Test with expired tokens
- Test with revoked tokens
- Verify token refresh mechanisms
- Check for proper encryption (HTTPS)

41. What is the difference between 401 Unauthorized and 403 Forbidden?

401 Unauthorized: Authentication is required but not provided or failed. User needs to log in.

403 Forbidden: User is authenticated but doesn't have permission to access the resource. Authorization failure.

42. How do you test role-based access control (RBAC) in APIs?

- Create users with different roles
- Test each endpoint with each role
- Verify admin can access admin endpoints
- Verify regular users can't access restricted resources
- Test privilege escalation scenarios
- Verify horizontal access control (user can't access other user's data)

43. What is CORS and why is it important in API testing?

CORS (Cross-Origin Resource Sharing) is a security mechanism controlling which domains can access an API from browsers. Testing includes:

- Verifying allowed origins
- Testing preflight OPTIONS requests
- Checking CORS headers in responses
- Testing blocked origins receive errors

44. What security headers should you verify in API responses?

- Strict-Transport-Security: Enforces HTTPS
- Content-Security-Policy: Prevents XSS
- X-Content-Type-Options: Prevents MIME sniffing
- X-Frame-Options: Prevents clickjacking
- X-XSS-Protection: Enables XSS filtering

45. How do you test for SQL injection vulnerabilities in APIs?

- Send SQL commands in input fields
- Use special characters: ' OR '1'='1
- Test with union queries
- Verify API sanitizes inputs
- Check error messages don't reveal database structure
- Use automated security scanning tools

Data Validation and Testing (Questions 46-60)

46. How do you validate JSON schema in API responses?

Use JSON Schema validation to verify:

- Data types match specifications
- Required fields are present

- Field formats are correct (email, date, etc.)
- Enumerations contain valid values
- Array lengths are within limits Tools: JSON Schema Validator, REST Assured, Postman

47. What is schema validation and why is it important?

Schema validation ensures API responses conform to expected structure and data types. It's important for:

- Catching data structure changes early
- Ensuring contract compliance
- Preventing integration issues
- Validating data integrity

48. How do you test API pagination?

- Test first page retrieval
- Navigate through multiple pages
- Test last page
- Verify page size limits
- Test invalid page numbers
- Check total count accuracy
- Test with different page sizes
- Verify navigation links (next, previous)

49. What checks should you perform on API response data?

- Status code correctness
- Response time within acceptable limits
- Data type validation
- Required fields presence
- Data format correctness
- Boundary values
- Error messages clarity
- Response headers

50. How do you test APIs that return large datasets?

- Use pagination to limit data size
- Test with filters to reduce results

- Verify performance with full dataset
- Check memory usage
- Test timeout scenarios
- Validate data consistency across pages
- Use streaming for very large responses

51. How do you handle dynamic data in API testing?

- Use regular expressions for pattern matching
- Extract and store dynamic values (IDs, tokens)
- Use variables to chain requests
- Implement data-driven testing
- Focus on data structure rather than exact values
- Use relative assertions (greater than, contains)

52. What is data-driven testing in API automation?

Data-driven testing separates test data from test scripts, allowing the same test to run with multiple data sets. Data is stored in external files (CSV, Excel, JSON) and iterated through test execution.

53. How do you test file upload APIs?

- Test with various file types
- Test file size limits (minimum, maximum, exceeding)
- Test with corrupted files
- Test with empty files
- Verify file content after upload
- Test concurrent uploads
- Check for malicious file uploads
- Verify proper error messages

54. How do you test API that returns different data formats?

- Test with different Accept headers (JSON, XML, CSV)
- Verify correct Content-Type in response
- Validate data structure in each format
- Test default format without Accept header
- Test with unsupported formats
- Verify data consistency across formats

55. What is API contract testing with examples?

Contract testing ensures provider meets consumer expectations. Example:

Consumer expects:

json

```
{
  "userId": "string",
  "name": "string",
  "email": "string"
}
```

Provider must always return this structure. Tools: Pact, Spring Cloud Contract.

56. How do you test date and time handling in APIs?

- Test various date formats (ISO 8601, Unix timestamp)
- Test timezone conversions
- Test date ranges and validations
- Test invalid dates (Feb 30)
- Test leap years
- Test daylight saving time transitions
- Verify UTC vs. local time handling

57. How do you validate error messages in API responses?

- Verify appropriate status codes
- Check error message clarity and usefulness
- Ensure no sensitive information exposure
- Validate error response structure consistency
- Test all error scenarios
- Verify proper error codes/identifiers
- Check localization if applicable

58. What is the importance of testing API response time?

Response time affects:

- User experience
- System scalability
- SLA compliance

- Resource utilization
- Application performance

Typical benchmarks: <200ms excellent, 200-1000ms acceptable, >1000ms slow

59. How do you test APIs with nested JSON objects?

- Validate each nesting level separately
- Use JSONPath or similar to extract nested values
- Verify array within objects
- Test optional nested objects
- Validate nested object schema
- Test with missing nested objects

60. What is the difference between synchronous and asynchronous API testing?

Synchronous: Client waits for immediate response. Test response directly.

Asynchronous: Client receives acknowledgment, actual response comes later via callback/webhook. Testing involves:

- Verifying initial acknowledgment
- Polling for completion
- Testing callbacks/webhooks
- Handling timeouts
- Testing notification mechanisms

Advanced Concepts (Questions 61-80)

61. What is GraphQL and how does it differ from REST?

GraphQL is a query language for APIs where clients specify exactly what data they need. Differences:

- Single endpoint vs. multiple REST endpoints
- Client-defined responses vs. server-defined
- Eliminates over-fetching/under-fetching
- Strongly typed schema
- Real-time updates with subscriptions

62. How do you test GraphQL APIs?

- Test query syntax and structure
- Validate field selection and nesting
- Test variables and arguments

- Verify error handling
- Test mutations for data changes
- Test subscriptions for real-time updates
- Validate schema introspection
- Test query complexity limits

63. What is API rate limiting and how do you test it?

Rate limiting restricts the number of API requests in a time period. Testing:

- Make requests below limit (should succeed)
- Exceed rate limit (should return 429 Too Many Requests)
- Verify rate limit headers (X-RateLimit-Limit, X-RateLimit-Remaining)
- Test rate limit reset
- Test different rate limits for different tiers

64. What are webhooks and how do you test them?

Webhooks are HTTP callbacks triggered by events. Testing:

- Set up webhook listener/mock server
- Trigger events that should send webhooks
- Verify payload structure and data
- Test retry mechanisms on failure
- Test webhook signatures/authentication
- Test timeout scenarios
- Verify idempotency

65. How do you implement retry logic in API testing?

Retry logic handles temporary failures:

- Implement exponential backoff
- Set maximum retry attempts
- Retry only on specific errors (500, 503)
- Log retry attempts
- Don't retry on client errors (4xx)
- Add jitter to prevent thundering herd

66. What is API throttling and how is it different from rate limiting?

Rate limiting: Hard limit on requests per time window

Throttling: Slows down request processing when limits approach, may queue requests rather than rejecting them. Both control API consumption but throttling is more gradual.

67. How do you test microservices APIs?

- Test each service independently
- Test inter-service communication
- Use contract testing (Pact)
- Test service discovery
- Test circuit breakers and fallbacks
- Test distributed tracing
- Test service mesh interactions
- Perform chaos engineering

68. What is API gateway and what should you test?

API gateway is a server that acts as entry point for microservices. Test:

- Routing to correct services
- Authentication/authorization
- Rate limiting
- Request/response transformation
- Load balancing
- Caching mechanisms
- Logging and monitoring
- SSL/TLS termination

69. Explain circuit breaker pattern in API testing.

Circuit breaker prevents cascading failures by monitoring for failures and "opening the circuit" to stop requests to failing services. States:

- Closed: Normal operation
- Open: Service failing, requests rejected immediately
- Half-Open: Testing if service recovered

Test all three states and transitions.

70. How do you test API versioning strategies?

- Test all supported versions simultaneously
- Verify backward compatibility
- Test deprecated version warnings

- Test version negotiation
- Ensure old clients work with new versions
- Test version-specific features
- Validate version migration paths

71. What is API documentation testing?

Validating that API documentation accurately reflects actual API behavior:

- Test all documented examples
- Verify endpoint URLs are correct
- Check parameter descriptions
- Validate response examples
- Test authentication instructions
- Verify error codes documentation
- Check for undocumented features

72. How do you test WebSocket APIs?

- Establish WebSocket connection
- Send and receive messages
- Test connection upgrades
- Test ping/pong heartbeats
- Test connection close handling
- Test message ordering
- Test reconnection logic
- Test concurrent connections

73. What is API orchestration and how do you test it?

API orchestration coordinates multiple API calls into a single workflow. Testing:

- Test complete workflow execution
- Test partial failures and rollback
- Test data dependencies between calls
- Verify transaction consistency
- Test timeout handling
- Test parallel vs. sequential execution

74. How do you perform chaos testing on APIs?

Chaos testing intentionally introduces failures:

- Randomly kill service instances
- Introduce network latency
- Corrupt data
- Fill disk space
- Exhaust resources
- Verify graceful degradation
- Test recovery mechanisms Tools: Chaos Monkey, Gremlin

75. What are the best practices for API test automation?

- Use page object model pattern
- Implement data-driven testing
- Use descriptive test names
- Maintain test independence
- Use proper assertions
- Implement logging and reporting
- Use environment-specific configurations
- Implement continuous integration
- Use version control
- Regular test maintenance

76. How do you handle test data management in API testing?

- Use test data factories
- Implement data cleanup after tests
- Use database snapshots/rollbacks
- Create isolated test environments
- Use realistic but synthetic data
- Implement data seeding scripts
- Avoid hard-coded data
- Use data generation libraries

77. What is API mocking vs. API stubbing?

Mocking: Simulates API with pre-programmed responses, can verify interactions were made correctly.

Stubbing: Provides canned responses to calls, focuses on simulating state without verification.

78. How do you test API backward compatibility?

- Run old test suite against new version
- Verify old request formats still work
- Check deprecated features still function
- Test with old client libraries
- Verify response structure includes old fields
- Test optional new fields don't break old clients

79. What is continuous testing in API development?

Continuous testing integrates testing throughout the development pipeline:

- Automated tests run on every commit
- Multiple test environments
- Shift-left testing approach
- Parallel test execution
- Fast feedback loops
- Integration with CI/CD pipelines

80. How do you measure API test coverage?

Metrics include:

- Endpoint coverage: % of endpoints tested
- Method coverage: All HTTP methods tested
- Status code coverage: All response codes tested
- Parameter coverage: All parameters tested
- Scenario coverage: Business scenarios covered
- Error path coverage: Error conditions tested

Scenario-Based Questions (Questions 81-105)

81. Scenario: An API intermittently returns 500 errors. How would you troubleshoot?

Steps:

1. Check logs for error patterns and timestamps
2. Reproduce the issue with specific requests
3. Monitor server resources (CPU, memory, disk)
4. Check database connection pools

5. Verify third-party dependencies
6. Review recent deployments
7. Test with load to identify concurrency issues
8. Check for timeout issues
9. Implement retry logic with exponential backoff
10. Add detailed error logging

82. Scenario: You need to test a payment gateway API. What scenarios would you cover?

Test scenarios:

- Successful payment with valid card
- Payment with expired card (should fail)
- Payment with insufficient funds
- Payment with invalid CVV
- Payment with different currencies
- Refund processing
- Partial refunds
- Duplicate payment prevention (idempotency)
- Timeout handling
- Large transaction amounts
- Payment status verification
- Webhook notifications for payment events
- PCI compliance validation
- Rate limiting on payment attempts
- Concurrent payment requests

83. Scenario: API returns correct data in dev but wrong data in production. How do you investigate?

Investigation steps:

1. Compare environment configurations
2. Check database data differences
3. Verify API versions deployed
4. Review environment-specific variables
5. Check cache configurations

6. Verify external service endpoints
7. Compare request/response in both environments
8. Check for environment-specific feature flags
9. Review deployment logs
10. Test with production-like data in lower environment

84. Scenario: You need to test a search API. What test cases would you create?

Test cases:

- Search with exact match
- Search with partial match
- Search with special characters
- Search with empty string
- Search with very long query
- Case-insensitive search
- Search with filters applied
- Pagination of search results
- Sorting results by relevance/date
- Search with no results
- Search with wildcards
- Search performance with large datasets
- Faceted search
- Auto-complete functionality
- Search suggestions

85. Scenario: API response time is degrading over time. How would you approach this?

Approach:

1. Establish baseline performance metrics
2. Monitor database query performance
3. Check for N+1 query problems
4. Review caching strategy effectiveness
5. Analyze database indexes
6. Check for memory leaks
7. Profile application code

8. Review connection pool settings
9. Check for growing data without cleanup
10. Verify CDN and load balancer configuration
11. Implement pagination if returning large datasets
12. Add database query optimization
13. Consider implementing caching layers
14. Monitor third-party API dependencies

86. Scenario: You need to migrate from REST to GraphQL. How would you test the migration?

Testing strategy:

1. Run parallel testing (REST and GraphQL)
2. Compare responses for data consistency
3. Test all REST endpoints have GraphQL equivalent
4. Validate schema matches REST contract
5. Test query performance vs. REST endpoints
6. Test nested queries and relationships
7. Verify error handling parity
8. Test authentication/authorization
9. Load test both implementations
10. Create migration test suite
11. Test client libraries for both
12. Implement feature flags for gradual rollout

87. Scenario: API needs to handle file uploads up to 100MB. What would you test?

Test scenarios:

- Upload files of various sizes (1KB, 1MB, 50MB, 100MB)
- Upload exactly at 100MB limit
- Attempt to upload files exceeding 100MB
- Upload different file types (images, PDFs, videos)
- Upload corrupted files
- Test upload progress tracking
- Test resume on interrupted uploads
- Test concurrent uploads

- Test upload timeout
- Verify file integrity after upload
- Test malicious file uploads
- Test with slow network connections
- Validate file metadata
- Test storage capacity limits
- Test virus scanning integration

88. Scenario: Third-party API you depend on is down. How do you handle testing?

Handling strategy:

1. Implement API mocking for tests
2. Create stub responses based on documentation
3. Test circuit breaker activates correctly
4. Verify fallback mechanisms
5. Test error messages to users
6. Test retry logic
7. Implement timeout handling
8. Test graceful degradation
9. Cache previous successful responses
10. Monitor and alert on dependency failures
11. Test with different failure scenarios
12. Verify logging of dependency failures

89. Scenario: You find that API documentation doesn't match actual behavior. What do you do?

Actions:

1. Document the discrepancies with examples
2. Verify with actual API calls and responses
3. Create test cases proving the discrepancy
4. Report to development team with evidence
5. Check if it's intentional or a bug
6. Determine if it's breaking change
7. Update test automation to match actual behavior
8. Coordinate documentation updates

9. Verify other endpoints for similar issues
10. Check version history for when change occurred

90. Scenario: You need to test an API that processes asynchronous jobs. How would you approach it?

Testing approach:

1. Submit job and verify acceptance response (202 Accepted)
2. Implement polling mechanism to check status
3. Test job status endpoint
4. Verify job completion notifications/webhooks
5. Test job cancellation
6. Test multiple concurrent jobs
7. Test job priority handling
8. Test job failure scenarios
9. Test job timeout handling
10. Verify job results retrieval
11. Test job history/audit trail
12. Test queue overflow scenarios
13. Verify idempotency of job submission

91. Scenario: API needs to support multiple languages/localization. What would you test?

Test scenarios:

- Test with different Accept-Language headers
- Verify translated error messages
- Test date/time formatting for different locales
- Test currency formatting
- Test right-to-left languages
- Verify character encoding (UTF-8)
- Test with missing translations (fallback)
- Test sorting with different character sets
- Verify number formatting
- Test with language-specific validation rules
- Test timezone handling

- Validate text length differences across languages

92. Scenario: Your API test suite takes 2 hours to run. How would you optimize it?

Optimization strategies:

1. Parallelize test execution
2. Identify and remove redundant tests
3. Use API mocking where appropriate
4. Implement test categories (smoke, regression, full)
5. Run quick tests in CI, detailed tests nightly
6. Optimize test data creation
7. Use database snapshots instead of setup/teardown
8. Cache authentication tokens
9. Reduce test dependencies
10. Profile slow tests and optimize
11. Use headless execution
12. Implement smart test selection (run only affected tests)

93. Scenario: You discover a security vulnerability in the API during testing. What's your process?

Process:

1. Document the vulnerability with reproduction steps
2. Assess severity and potential impact
3. Report immediately to security team privately
4. Do not share details publicly
5. Create proof of concept if needed
6. Follow responsible disclosure process
7. Verify fix in subsequent testing
8. Update security test cases
9. Review similar patterns in other endpoints
10. Document lessons learned
11. Verify no data breach occurred

94. Scenario: API returns inconsistent data when called multiple times quickly. What could be wrong?

Possible issues:

- Cache synchronization problems
- Database replication lag
- Load balancer routing to different versions
- Race conditions in code
- Stale cache invalidation issues
- Eventual consistency in distributed systems
- Transaction isolation issues
- Multiple database instances out of sync

Testing approach: Call API repeatedly, compare responses, check timestamps, verify data sources, test under load.

95. Scenario: You need to test an API that integrates with 5 external services. How do you manage dependencies?

Management strategy:

1. Create mocks for each external service
2. Use contract testing (Pact)
3. Implement service virtualization
4. Test each integration independently
5. Test with real services in staging
6. Create dependency health checks
7. Test cascade failure scenarios
8. Implement circuit breakers
9. Test fallback mechanisms
10. Use feature flags to isolate services
11. Monitor external service SLAs
12. Test timeout handling for each service

96. Scenario: Business wants to deprecate an old API version. How do you validate impact?

Validation process:

1. Analyze API usage logs for old version traffic
2. Identify clients still using old version
3. Communicate deprecation timeline
4. Provide migration guide and support
5. Test that new version supports old use cases

6. Implement warnings in old version responses
7. Monitor error rates during migration
8. Test parallel running of both versions
9. Verify backward compatibility
10. Plan phased deprecation
11. Keep old version in read-only mode initially

97. Scenario: API needs to handle burst traffic (10x normal load). How would you test this?

Testing approach:

1. Establish baseline normal load metrics
2. Use load testing tools (JMeter, Gatling)
3. Gradually increase to 10x load
4. Monitor response times and error rates
5. Test auto-scaling triggers
6. Verify rate limiting behavior
7. Test queue overflow handling
8. Monitor database connections
9. Test cache effectiveness under load
10. Verify alerting thresholds
11. Test recovery after burst
12. Identify bottlenecks
13. Test sustained high load

98. Scenario: You need to validate that sensitive data is properly encrypted. How do you test this?

Testing approach:

1. Verify HTTPS/TLS for all endpoints
2. Check that sensitive data not logged
3. Test password hashing (not stored plain text)
4. Verify encryption at rest
5. Test that tokens are encrypted
6. Inspect network traffic with tools like Wireshark
7. Check database to ensure encrypted storage
8. Verify PII is masked in responses

9. Test that API keys are not exposed in URLs
10. Validate certificate strength and validity
11. Test that sensitive headers are not cached
12. Verify no sensitive data in error messages

99. Scenario: API is experiencing memory leaks causing crashes after running for several days. How would you identify and test for this?

Identification and testing approach:

1. Monitor memory usage over extended periods
2. Use profiling tools (JProfiler, VisualVM)
3. Run long-duration tests (soak testing)
4. Check for unclosed connections/resources
5. Review code for proper resource disposal
6. Monitor garbage collection patterns
7. Test with realistic load for 24-48 hours
8. Check for growing collections/caches
9. Review database connection management
10. Analyze heap dumps before crashes
11. Test with different load patterns
12. Verify proper cleanup in finally blocks
13. Check for circular references preventing GC
14. Monitor thread pool growth
15. Test resource limits and thresholds

100. Scenario: You need to test an API that implements eventual consistency across distributed databases. What challenges would you face and how would you test it?

Challenges:

- Data may not be immediately consistent across regions
- Different read replicas may return different results
- Race conditions in distributed updates
- Network partitions causing temporary inconsistencies

Testing approach:

1. Write data and immediately read from different regions
2. Test acceptable consistency windows (seconds/minutes)

3. Verify conflict resolution strategies
4. Test write-after-write consistency
5. Test read-after-write consistency per user
6. Simulate network partitions
7. Test concurrent updates to same data
8. Verify eventual convergence timeframes
9. Test version vectors or timestamps
10. Validate business logic handles stale reads
11. Test causal consistency if required
12. Monitor replication lag metrics
13. Test failover scenarios
14. Verify data reconciliation processes
15. Test with anti-entropy mechanisms