60 Test Cases For API Testing (With Template + API Testing Best Practices)

Application Programming Interface (API) is the backbone of the modern world. A lot of the activities you do on digital platforms leverage APIs, and testing those APIs is key to delivering good User Experience in software and applications. If you are trying to test APIs and don't know where to start, read on and discover **60 test cases for API testing** that you can use for references. We also include a test case template so that you can actually use those test cases in your work, as well as a detailed guide on how you should test APIs.

Read More: Top 10 API Examples You Should Know

What is API Testing?

API testing is a software testing practice that tests the APIs directly — from their functionality, reliability, performance, to security. As part of integration testing, API testing aims to validate the logic of the build architecture within a short amount of time. The goal of API testing is not to check the individual software component itself, but rather the **connection** between them.

API testing can be performed either manually or automatically with the help of API testing tools, each comes with its own pros and cons. Generally automated API testing is preferred due to the repetitive and data-driven nature of this testing type. Also, there are many aspects of API testing, but these are 3 most common categories:

- **Functional testing** (checking whether the API performs its function as expected)
- **Performance testing** (checking how the API performs under different levels of usage)
- **Security testing** (checking if the APIs have any vulnerabilities or have been protected against security threats)

large_banner4_850bc50b04.png

60 Test Cases For API Testing For Each Category

1. Test Cases For API Functional Testing

large_test_cases_for_API_33a09cd4eb.png

Functionality is the core of any Application Under Test (AUT), and API is no exception. Their most basic and foundational functionality is data retrieval and data sending, and API functional testing should revolve around those 2 domains. Check out the following functional test cases and see how you can apply them to your own testing project:

1. **Status Code Validation for Valid Requests**: Verify that the API consistently returns the expected response status code, such as "200 OK," for valid and properly formatted requests.
2. **Authentication Handling with Invalid Credentials**: Test the API's response when provided with invalid authentication credentials, ensuring it consistently returns a "401 Unauthorized" status code as expected.
3. **Graceful Handling of Missing or Invalid Parameters**: Verify that the API handles missing or invalid request parameters gracefully and returns clear and user-friendly error messages that aid in troubleshooting.
4. **Input Data Validation with Malformed Data**: Test the API's input validation by submitting various forms of malformed data, such as invalid email formats, and confirm that it properly rejects and responds to these inputs.
5. **Timeout Handling under Load**: Confirm that the API correctly handles timeouts by simulating requests that take longer to process, ensuring that it remains responsive and does not hang.
6. **Pagination Functionality Verification**: Test the API's pagination functionality by requesting specific pages of results and verifying that the responses contain the expected data and pagination information.
7. **Concurrency Testing without Data Corruption**: Verify that the API handles concurrent requests from multiple users without data corruption or conflicts, ensuring data integrity.
8. **Response Format Adherence (JSON/XML)**: Ensure that the API consistently returns responses in the specified format (e.g., JSON or XML) and adheres to the defined schema for data structure.
9. **Caching Mechanism Evaluation with Repeated Requests**: Evaluate the API's caching mechanism by making repeated requests and verifying that the cache headers are correctly set and honored.
10. **Rate Limiting Assessment**: Test the API's rate limiting by sending requests at a rate that exceeds the defined limits and checking for the expected rate-limiting responses, ensuring that limits are enforced.
11. **HTTP Method Support for CRUD Operations**: Verify that the API supports a variety of HTTP methods (GET, POST, PUT, DELETE) for Create, Read, Update, and Delete operations, and that it returns appropriate responses for each.
12. **Error Handling Capabilities for Meaningful Messages**: Evaluate the API's error-handling capabilities by intentionally causing errors, such as invalid inputs or unexpected situations, and confirm that it consistently returns meaningful error messages for troubleshooting.
13. **Conditional Request Handling (If-Modified-Since, If-None-Match)**: Test the API's support for conditional requests using headers like If-Modified-Since and If-None-Match, ensuring that responses are handled appropriately.
14. **Sorting and Filtering Validation for Resource Listings**: Verify that the API correctly sorts and filters resource listings based on specified parameters, maintaining data accuracy.
15. **Handling Long or Complex Data without Data Corruption**: Ensure that the API properly handles long or complex strings, such as URLs or text fields, without truncating or corrupting the data.
16. **Content Negotiation Support for Multiple Formats**: Test the API's support for content negotiation by specifying different Accept headers (e.g., JSON, XML) and

verifying that the response format matches the requested format.

17. **Resource Not Found Handling (404 Not Found)**: Confirm that the API consistently returns the appropriate "404 Not Found" response when attempting to access a non-existent resource.
18. **Response Time Measurement for Various Requests**: Measure the API's response time for different types of requests to assess its performance and responsiveness.
19. **Handling Large Payloads (File Uploads)**: Verify that the API can handle large payloads, such as file uploads, without encountering errors or significant performance degradation.
20. **Compatibility with Client Libraries and SDKs**: Evaluate the API's compatibility with different client libraries or SDKs to ensure seamless integration with various platforms and programming languages.

→ Check out the top automated functional testing tools on the current market

2. Test Cases For API Performance Testing

If the API developed in your team receives high traffic, it is a good idea to incorporate performance testing into your daily routine. In fact, performance testing should start even before development begins since it provides valuable insights into the maximum stress level of the server, which can help the IT Ops team better allocate and optimize resources usage. Here are some common test cases when doing performance testing of your APIs:

1. **Baseline Response Time**: Measure the response time of a simple API request under normal conditions to establish a performance baseline.
2. **Stress Testing**: Send a large number of simultaneous requests to the API to assess its performance under heavy load.
3. **Concurrency Testing**: Evaluate how the API handles a specified number of concurrent requests without performance degradation.
4. **Ramp-up Testing**: Gradually increase the number of requests over time to identify the API's breaking point and performance limits.
5. **Peak Load Testing**: Test the API's performance at peak usage times to ensure it can handle maximum expected traffic.
6. **Endurance Testing**: Continuously send requests to the API for an extended duration to assess its stability over time.
7. **Scalability Testing**: Increase the load gradually and measure how the API scales by adding more resources (e.g., servers) to maintain performance.
8. **Resource Utilization Testing**: Monitor CPU, memory, and network utilization while conducting performance tests to identify resource bottlenecks.
9. **Response Time Distribution**: Analyze the distribution of response times to identify outliers and performance issues.
10. **Latency Testing**: Measure network latency between the client and the API server to ensure low latency for users.
11. **Throughput Testing**: Determine the maximum number of transactions the API can handle per unit of time while maintaining acceptable response times.
12. **Error Rate Testing**: Monitor and record the rate of errors or failed requests during load testing to assess error handling and resilience.
13. **Caching Performance**: Evaluate the impact of caching on response times and

resource utilization.

14. **Data Volume Testing**: Test the API with varying data volumes (e.g., small, medium, large payloads) to assess its performance with different data sizes.
15. **Geographical Load Testing**: Simulate requests from different geographic locations to assess the API's global performance and response times.
16. **Concurrency with Authentication**: Evaluate how the API handles concurrent requests with authentication, including token validation.
17. **Database Load Testing**: Assess the impact of API requests on the associated database by measuring query response times.
18. **Long-Running Transactions**: Test transactions that take a significant amount of time to complete and assess their impact on overall system performance.
19. **Rate Limiting Stress Testing**: Test how the API handles excessive requests when rate limiting is in place.
20. **Failover Testing**: Simulate server failures and test the API's ability to failover to backup servers while maintaining performance.

You May Also Like: Performance Testing vs Load Testing: A Complete Guide

3. Test Cases For API Security Testing

Finally, also make sure to check the security of your API, since this is the area where sensitive and high-value data is exchanged. APIs have always been a common target of attackers looking to gain unauthorized access to your systems. Some common API security test cases are:

1. **Authentication Testing**: Verify that the API enforces proper authentication for all endpoints.
2. **Authorization Testing**: Ensure that users can access only the resources they are authorized to access.
3. **Token Security**: Test the security of authentication tokens, including token encryption and expiration.
4. **Session Management**: Check for secure session management and handling of session cookies.
5. **SQL Injection**: Test for SQL injection vulnerabilities by injecting malicious SQL queries in API parameters.
6. **Cross-Site Scripting (XSS)**: Verify that the API is protected against XSS attacks by injecting malicious scripts.
7. **Cross-Site Request Forgery (CSRF)**: Test if the API is vulnerable to CSRF attacks by submitting unauthorized requests.
8. **Input Validation**: Ensure that the API validates and sanitizes user inputs to prevent injection attacks.
9. **Rate Limiting**: Test the API's rate limiting to prevent abuse and DoS attacks.
10. **Sensitive Data Exposure**: Verify that sensitive data, such as passwords or API keys, are not exposed in responses.
11. **HTTPS/TLS Testing**: Ensure that the API uses secure communication via HTTPS/TLS and checks for certificate validity.
12. **CORS (Cross-Origin Resource Sharing) Security**: Test for correct CORS headers to prevent unauthorized cross-origin requests.
13. **API Key Security**: Assess the security of API keys and their storage.

14. **JWT (JSON Web Token) Security**: Evaluate the security of JWTs used for authentication and authorization.
15. **Authentication Bypass**: Attempt to bypass authentication mechanisms and gain unauthorized access.
16. **Session Fixation**: Test if the API is vulnerable to session fixation attacks.
17. **Insecure Direct Object References (IDOR)**: Check for unauthorized access to resources by manipulating object references.
18. **Denial of Service (DoS) Testing**: Attempt to overload the API and test its resilience against DoS attacks.
19. **API Versioning**: Verify that the API supports versioning to prevent breaking changes from affecting existing clients.
20. **Security Headers**: Check for the presence of security headers such as Content Security Policy (CSP), X-Content-Type-Options, etc., in API responses.