API TESTING

API testing is a type of software testing that analyzes an application program interface (API) to verify it fulfills its expected functionality, security, performance and reliability. The tests are performed either directly on the API or as part of integration testing. An API is middleware code that enables two software programs to communicate with each other. The code also specifies the way an application requests services from the operating system (OS) or other applications.

Applications frequently have three layers: a data layer, a service layer -- the API layer -- and a presentation layer -- the user interface (UI) layer. The business logic of the application -- the guide to how users can interact with the services, functions and data held within the app -- is in the API layer. API testing focuses on analyzing the business logic as well as the security of the application and data responses. An API test is generally performed by making requests to one or more API endpoints and comparing the response with expected results.

API testing is frequently automated and used by DevOps, quality assurance (QA) and development teams for continuous testing practices.

How to approach API testing

An API testing process should begin with a clearly defined scope of the program as well as a full understanding of how the API is supposed to work. Some questions that testers should consider include:

- What endpoints are available for testing?
- What response codes are expected for successful requests?
- What response codes are expected for unsuccessful requests?

- Which error message is expected to appear in the body of an unsuccessful request?

Once factors such as these are understood, testers can begin applying various testing techniques. Test cases should also be written for the API. These test cases define the conditions or variables under which testers can determine whether a specific system performs correctly and responds appropriately. Once the test cases have been specified, testers can perform them and compare the expected results to the actual results. The test should analyze responses that include:

- reply time,
- data quality,
- confirmation of authorization,
- HTTP status code and
- error codes.

API testing can analyze multiple endpoints, such as web services, databases or web user interfaces. Testers should watch for failures or unexpected inputs. Response time should be within an acceptable agreed-upon limit, and the API should be secured against potential attacks.

Tests should also be constructed to ensure users can't affect the application in unexpected ways, that the API can handle the expected user load and that the API can work across multiple browsers and devices.

The test should also analyze the results of nonfunctional tests as well, including performance and security.

Types of API tests

Various types of API tests can be performed to ensure the application programming interface is working appropriately. They range from general to specific analyses of the software. Here are examples of some of these tests.

Validation testing includes a few simple questions that address the whole project. The first set of questions concerns the product: Was the correct product built? Is the designed API the correct product for the issue it attempts to resolve? Was there any major code bloat -- production of code that is unnecessarily long, slow and wasteful -- throughout development that would push the API in an unsustainable direction?

The second set of questions focuses on the API's behavior: Is the correct data being accessed in the predefined manner? Is too much data being accessed? Is the API storing the data correctly given the data set's specific integrity and confidentiality requirements?

The third set of questions looks at the efficiency of the API: Is this API the most efficient and accurate method of performing a task? Can any codebase be altered or entirely removed to reduce impairments and improve overall service?

Functional testing ensures the API performs exactly as it is supposed to. This test analyzes specific functions within the codebase to guarantee that the API functions within its expected parameters and can handle errors when the results are outside the designated parameters.

Load testing is used to see how many calls an API can handle. This test is often performed after a specific unit, or the entire codebase, has been completed to determine whether the theoretical solution can also work as a practical solution when acting under a given load.

Reliability testing ensures the API can produce consistent results and the connection between platforms is constant.

Security testing is often grouped with penetration testing and fuzz testing in the greater security auditing process. Security testing incorporates aspects of both penetration and fuzz testing, but also attempts to validate the encryption methods the API uses as well as the access control design. Security testing includes the validation of authorization checks for resource access and user rights management.

Penetration testing builds upon security testing. In this test, the API is attacked by a person with limited knowledge of the API. This enables testers to analyze the attack vector from an outside perspective. The attacks used in penetration testing can be limited to specific elements of the API or they can target the API in its entirety.

Fuzz testing forcibly inputs huge amounts of random data -- also called noise or fuzz -- into the system, attempting to create negative behavior, such as a forced crash or overflow.

Why is API testing important?

User interface tests are often inefficient for validating API service functionality and often do not cover all the necessary aspects of back-end testing. This can result in bugs left within the server or unit levels -- a costly mistake that can greatly delay the product release and often requires large amounts of code to be rewritten.

API testing allows developers to start testing early in the development cycle before the UI is ready. Any request that doesn't produce the appropriate value at the server layer will not display it on the UI layer. This enables developers to kill at least half of the existing bugs before they become more serious problems. It also enables testers to make requests that might not be possible through the UI -- a necessity for exposing security flaws.

Many companies are using microservices for their software applications because they allow software to be deployed more efficiently. If one area of the app is being updated, the other areas can continue functioning without interruption. Each application section has a separate data store and different commands for interacting with that data store. Most microservices use APIs; therefore, as more business adopt the use of microservices, API testing will become increasingly necessary to ensure all parts are working correctly.

API testing is also integral to Agile software development, in which instant feedback is necessary to the process flow. In Agile environments, unit tests and API tests are preferred over graphical user interface (GUI) tests because they are easy to maintain and more efficient. GUI tests often require intense reworking if they want to keep pace with the frequent changes in an Agile environment.

Overall, incorporating API tests into the test-driven development process can benefit engineering and development teams across the entire development lifecycle. These benefits are then passed along to customers in the form of improved services and better-quality products.

Benefits of API testing

API testing guarantees that connections among platforms are reliable, safe and scalable. Specific benefits include:

- API test automation requires less code than automated GUI tests, resulting in faster testing and a lower overall cost.
- API testing enables developers to access the app without a UI, helping the tester identify errors earlier in the development lifecycle, rather than waiting for them to become bigger issues. This also saves money because errors can be more efficiently resolved when caught early.
- API tests are technology and language independent. Data is exchanged using JSON or XML and it contains HTTP requests and responses.
- API tests use extreme conditions and inputs when analyzing applications. This removes vulnerabilities and guards the app from malicious code and breakage.

- API tests can be integrated with GUI tests. For example, integration can enable new users to be created within the app before a GUI test is performed.

While API testing presents these various benefits, it also produces challenges. The most common limitations found in API tests are parameter selection, parameter combination and call sequencing. Parameter selection requires the parameters sent through API requests to be validated -- a process that can be difficult. However, it is necessary that testers guarantee that all parameter data meets the validation criteria, such as the use of appropriate string or numerical data, an assigned value range and conformance with length restrictions.

Parameter combination can be challenging because every combination must be tested to see if it holds problems related to specific configurations. Call sequencing is also a challenge because every call must appear in a specific order to ensure the system works correctly. This quickly becomes a challenge, especially when dealing with multithreaded applications.

API testing tools

When performing an API test, developers can either write their own framework or choose from a variety of ready-to-use API testing tools. Designing an API test framework enbles developers to customize the test; they are not limited to the capabilities of a specific tool and its plugins. Testers can add whichever library they consider appropriate for their chosen coding platform, build unique and convenient reporting standards and incorporate complicated logic into the tests. However, testers need sophisticated coding skills if they choose to design their own framework.

Conversely, API testing tools provide user-friendly interfaces with minimal coding requirements that enable less-experienced developers to feasibly deploy the tests. Unfortunately, the tools are often designed to analyze general API issues and problems more specific to the tester's API can go unnoticed.

A large variety of API testing tools is available, ranging from paid subscription tools to open source offerings. Some specific examples of API testing tools include:

- **SoapUI.** The tool focuses on testing API functionality in SOAP and REST APIs and web services.
- **Apache JMeter.** An open source tool for load and functional API testing.
- **Apigee.** A cloud API testing tool from Google that focuses on API performance testing.
- **REST Assured.** An open source, Java-specific language that facilitates and eases the testing of REST APIs.
- **Swagger UI.** An open source tool that creates a webpage that documents APIs used.
- **Postman.** A Google chrome app used for verifying and automating API testing.
- **Katalon.** An open source application that helps with UI automated testing.

Examples of API tests

While the use cases of API testing are endless, here are two examples of tests that can be performed to guarantee that the API is producing the appropriate results.

When a user opens a social media app -- such as Twitter or Instagram -- they are asked to log in. This can be done independently -- through the app itself -- or through Google or Facebook. This implies the social media app has an existing agreement with Google and Facebook to access some level of user information already supplied to these two sources. An API test must then be conducted to ensure that the social media app can collaborate with Google and Facebook to pull the necessary information that will grant the user access to the app using login information from the other sources.

Another example is travel booking systems, such as Expedia or Kayak. Users expect all the cheapest flight options for specific dates to be available and displayed to them upon request when using a travel booking system. This requires the app to communicate with all the airlines to find the best flight options. This is done through APIs. As a result, API tests must be performed to ensure the travel booking system is successfully communicating with the other companies and presenting the correct results to users in an appropriate timeframe. Furthermore, if the user then chooses to book a flight and pays using a third-party payment service, such as PayPal, then API tests must be performed to guarantee the payment service and travel booking systems can effectively communicate, process the payment and keep the user's sensitive data safe throughout the process.

Best practices for API testing

API testing best practices include:

- When defining test cases, group them by category.
- Include the selected parameters in the test case itself.
- Develop test cases for every potential API input combination to ensure complete test coverage.
- Reuse and repeat test cases to monitor the API throughout production.
- Use both manual and automated tests to produce better, more trustworthy results.
- When testing the API, note what happens consistently and what does not.
- API load tests should be used to test the stress on the system.
- APIs should be tested for failures. Tests should be repeated until it produces a failed output. The API should be tested so that it fails consistently to identify the problems.
- Call sequencing should be performed with a solid plan.
- Testing can be made easier by prioritizing the API function calls.

- Use a good level of documentation that is easy to understand and automate the documentation creation process.
- Keep each test cases self-contained and separate from dependencies, if possible.