API Testing Guide

An automated approach to API testing transformation.
From Monolith to Microservice

The consumerization of IT has changed the way we write applications today. Instead of building a single, monolithic system that is installed on a server in a back office or call center, modern applications are scattered within mobile devices and web browsers so users can gain access to services at anytime from anywhere.

And modern applications are no longer self-contained; they are composed of multiple service components that are tied together at runtime. Therefore, the end-user experience is not powered by the elegant UIs we all use, but by APIs that facilitate interactions between code modules, applications and back-end IT systems.

Because of this, the way we test must change. Quality assurance (QA) and development teams must expand their capacity to test each layer of the application, as well as the end-to-end business transactions. Modern applications then require comprehensive API testing in addition to traditional UI testing. But API testing presents many challenges for testers. It requires a new approach—one that is automated wherever possible and inherently data-driven.

When testing APIs, testers are faced with numerous discrete units of work. Some of these APIs may have been created by a third party, while the possible ordering and combinations of different versions of APIs can cause the number of combinations that need to be tested to skyrocket to an impossible number.

The ability to identify these combinations and then reduce them to a realistic number without compromising quality requires testers to adopt a new model-based approach and invest in automation technologies.

Related Reading
Article from Professional Tester Magazine—Meeting the Challenge of API Testing
Your inclination may be to extend the same principals of UI testing to APIs—have developers build a UI in front of each API and put your “eyes on the glass,” using manual testing to input spreadsheets of data and watch the response. This approach, however, is not ideal, takes too long and will not scale. These challenges are exacerbated for several reasons:

**Use case proliferation**

Traditional UI testing is concerned only with the functionality of the overall application. A tester exercises an input and interprets the output against expected outcomes. But API testing is a different animal. Because APIs are the central hub of logic and the gateway to data for many applications, use cases are near-limitless. As a result, the number of required tests quickly surpasses the capabilities of the development and technical testers responsible for test case design.

**Asynchronous and synchronous processes**

Modern applications are complex and one API may link together several microservices and other APIs. As a result, a single call on an API may trigger any number of parallel and serial actions. For example, one service may look up a customer’s shipping address, while a second service looks up the price of an item. A third service then takes the output to calculate the tax based on the location of the purchaser and the cost of the item. This example assumes the application is calling one API. In reality, modern applications have APIs that get data from other APIs, that then rely on others. The complexity of an API can therefore grow exponentially as it is combined with other API calls. And this complexity only multiples as testers must also factor in the calling order of APIs in test case design. Consequently, the testing of APIs needs to be handled in both atomic, standalone units of work and in orchestration.
**The Challenges of API Testing, Continued**

**Access to dependent systems**

APIs pull data from multiple APIs, microservices and back-end systems. The resulting architecture can look like the roots of a tree spreading out deep and wide. And it’s unreasonable to think you will have access to every environment on this tree. Certain dependencies may still be under development, while third-party systems and mainframes may be too costly to test or their window and scope of access too limited. The simulation of unavailable resources is then key to avoiding testing bottlenecks. But the traditional way of building custom mocks and stubs is proving too problematic. For instance, stubs and mocks are usually written by developers, occasionally shared with QAs and very rarely shared with other development teams due to interoperability problems. What’s more, they cannot simulate race conditions for exhaustive performance and load testing.

**API versioning**

Versioning is a further cause of growing complexity in API testing. Most systems have a degree of deprecation, so an API must be able to handle an old version calling new versions, or a combination thereof. The API must recognize missing values and assign some kind of default to allow the old version to work. What’s more, it might be the case that some versions can be called by some versions but not others, and the numerous possible combinations must therefore be tested.

**Case in point:** In the following diagram, there are 127 API combinations which need to be tested. However, when the versioning is overlaid, there are 10,287 possible combinations which need to be tested, and it is not likely that manual scripting will cover a sufficient proportion of these.
A Model for API Testing Transformation

To overcome these challenges, you need to adopt a modern approach and set of tools that will enable you to automate API tests and incorporate them into a continuous delivery cycle.

- Generate API tests automatically and in multiple ways
- Edit and extend API tests without writing code
- Use one test and one tool across all layers and systems
- Run unlimited tests with service virtualization

“High-volume test automation requires test models, test-data generators and automatic oracles. Modelling, simulation, analytics, visualization and tool-supported decision making will become important capabilities of test architects and testing teams. Testers will have to learn how to create better test models and how to use them with more technical modelling and simulation tools.”

—Paul Gerrard, Gerrard Consulting

Related Reading
- How Will the Internet of Things Affect Testers?
Generate API Tests Automatically and in Multiple Ways

In a perfect world, you could automatically derive all of the test cases you need to ensure adequate coverage, and then generate relevant test scripts while automatically provisioning the appropriate sets of data to run against those tests. Thanks to advances in testing technology, this ideal is not far off. But some degree of manual intervention is going to be required. Therefore, the question is: How do you minimize this requirement?

- Start by implementing a system to formally capture and manage all API requirements and change requests.
- Obtain the appropriate tools to auto-generate test cases and associated test scripts from these requirements and in multiple ways. For example, from contracts (Swagger), samples (individual messages) or observations from real API data.
- Store test cases in a library for reuse. It’s important to tie all test cases back to their requirements. Only then will testers have the context they need to understand what the API is doing and where the transaction is occurring in the overall workflow.
- Leverage test data management technology to pull and mask existing data from multiple back-end systems and, when needed, extract missing data synthetically for maximum test coverage.

70% of all testing is still manual.\(^2\)
Automated tests created with scripts are dependent on the system under test. Should APIs change, as they are known to do, automated tests may no longer work with future versions of the API, thus requiring additional scripting. In the end, this process may lead to large, unmanageable sets of code that are prone to defects. At the same time, modern applications are complicated and tests need to be validated across multiple application layers. This requires a much higher degree of test orchestration.

But with the right test automation tools and testing framework, you can edit and extend API tests without modifying a single line of code.

- Look for an API testing solution that allows for modularity to facilitate reusability from release to release. Configuration (environment, IPs, logins/passwords) should be stored separate from data, which should also be separate from invocation method (protocols such as web services, MQ, JMS, TIBCO, SQL, etc). Being able to change one of these areas separate from the other two will allow you to reuse the tests and adjust to the rapidly changing API landscape.

- Choose an API testing solution with Selenium-based web UI testing and a native visual editor. With such tools, you can load tests developed in Selenium UI and scale them across multiple browsers.

CA Application Test supports seamless integration with Selenium. You can import web-based recordings or test cases from Selenium Builder into CA Application Test. After you import the test, you can run the test in Mozilla Firefox, Google Chrome, Apple Safari or Microsoft Internet Explorer®.

View the online demonstration.

Related Reading
Automated Testing: Coping with Change
Tests need to work harder for you. One test should not only trigger and validate individual API calls and associated responses but also allow for non-functional testing and work with open-source technologies like Selenium.

To do this, you need an extensible testing framework that enables your teams to test modern Restful, SOA and ESBs along with more traditional protocols like JMS, MQ, Java™ and databases. It also requires collaboration between systems, designers and implementers to tie test cases together to validate a business process from end-to-end.

With the right testing framework, you can:

- Use the output from one test as the input for subsequent tests.
- Gain access to a rich feature set to facilitate collaboration of testing assets and data between developers and analysts, along with a powerful framework to invoke and verify requests across different services.
- Stitch together test cases into workflows to mimic full business processes.
- Reuse existing functional tests as performance tests.
- Run multiple tests simultaneously to verify functional paths and back-end APIs and services.
- Schedule tests to validate environment stability.

“In order to support continuous quality and faster development, especially for those on an agile and DevOps journey, testing must shift from a UI-only approach to a UI- and API-approach.”

—The Forrester Wave™: Modern Application Functional Test Automation Tools, Q2 2015

Full-stack, Automated and Scriptless API Testing

With CA Application Test, testers can use a single tool to test APIs, UIs and the layers that feed them.
Run Unlimited Tests With Service Virtualization

Service virtualization is the practice of simulating the behavior, data and performance characteristics of a dependent system that you can then use at all stages of development and testing. In contrast to stubs and mocks, virtual services are created by recording actual services by contract or by sample requests and responses. They are more dynamic and don’t require a developer to write them.

Using service virtualization, you can test for a virtually unlimited set of test cases and scenarios whenever you want. You can set up tests to verify all independent services and endpoints—not each in a vacuum, but all together. That is, you can simultaneously test the API that fetches inventory data, the shopping cart API and so on, as the customer will use them. Service virtualization frees you from call limits so you can keep testing and testing and testing until your app is completely debugged.

In fact, access to a virtualized API is actually better than the real thing for the simple reason that you can test all kinds of scenarios—varying levels of functionality, performance and maintenance levels—with a virtualized service that you never could with an API. You can make the virtualized API behave any way you want.

Virtualize Dependencies and APIs for True Testability

1. Virtualize dependencies

- Supply chain
- CRM
- Order MGMT

Create virtual copies of systems that can be used to test various scenarios on demand.

2. Drive headless tests

- SUT
- Isolate the system

Use virtual APIs for rigorous testing.

- API
- Increase test flexibility velocity by driving headless tests.

“I think the most valuable feature is the ability to quickly create simulations of APIs that you use to disconnect you from your dependencies as part of your testing process.”

—Sam Detweiler, Sr. SDLC Architect
A Fully Integrated, End-to-End API Solution

API testing is complex. An automated approach is needed to introduce the required rigor needed to support scalable testing frameworks that can keep up with the rate of change. CA delivers the tools your teams need to develop, manage and test APIs simultaneously.

CA Test Data Manager and CA Agile Requirements Designer

CA Test Data Manager enables you to create production-like test data and keep it synchronized across all the systems it touches during the various test phases. This eliminates much of the manual effort and time spent subsetting, masking and tailoring data for each time and use case every time there is a data refresh.

CA Agile Requirements Designer is an automated modeling solution enabling you to adapt to changing API requirements and designs. It allows you to identify what tests to run with the proper number of use cases, and only in areas of application change. This allows you to maximize test coverage without over testing.

When an API specification changes or test cases are updated, CA Agile Requirements Designer can quickly update virtual data to retain maximum coverage. When a new variable is added to the flowchart, its exact impact on the data model is identified automatically. Any new combinations of variables needed for 100-percent coverage can be created using CA Test Data Manager, while any broken or invalid combinations will be removed or repaired. This eliminates the time wasted on manual data maintenance, providing your testers with the up-to-date data and environments needed to rigorously test APIs.

A Modern Toolset for API Testing Transformation

<table>
<thead>
<tr>
<th>CA Test Data Manager</th>
<th>CA Agile Requirements Designer</th>
<th>CA Application Test</th>
<th>CA Service Virtualization</th>
<th>CA API Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find, create and provision test data automatically and fast.</td>
<td>Automatically generate and maintain the smallest set of test cases needed for maximum coverage.</td>
<td>Edit and extend API tests without writing code, and test the full technology stack.</td>
<td>Virtualize third-party APIs and unavailable systems for agile testing.</td>
<td>Manage APIs and create a directory of virtual APIs for secure access.</td>
</tr>
</tbody>
</table>

Related Reading

Rigorously Test Composite Applications Faster with CA Test Data Manager and CA Agile Requirements
CA Application Test is a multiprotocol testing platform, which supports web services, message queue technologies and other endpoints such as file endpoints, file transfer endpoints and databases. With CA Application Test, you can:

- **Execute comprehensive test coverage:** Cover REST, JSON and SOAP along with a variety of messaging interfaces and other endpoints such as databases, flat files, EJBs, etc.
- **Easily create and manage API tests:** Use a graphical visualizer for JSON-based payloads and path generation or an easy-to-use wizard for generating web service API test cases.
- **Collaborate with high-productivity visual tests:** Test “headless” middle tiers with no UIs.
- **Customize load testing:** Set up parameters, test REST and other services and run hundreds of simulations at once.

With CA Application Test, testers can use a single tool to test UIs, APIs and the implementation layers that feed them. Users can model a single test case with steps that invoke and verify an API’s behavior and performance, as well as validate database calls, SOAP messages and many more types of transactions.

CA Application Test is not the automated testing solution your QA organization grew up thinking about. It enables you to:

- Distribute the tests across the architecture.
- Remove layers of complexity by testing as close to logic as possible.
- Start functional testing before development is complete.
- Decompose big response times into little pieces.
A Fully Integrated, End-to-End API Solution

CA Service Virtualization and CA API Management Suite

Together, [CA Service Virtualization](#) and [CA API Management](#) accelerate application delivery providing solutions across the Plan, Build and Run phases of the application and service lifecycle.

The [CA API Developer Portal](#) gives developers the ability to discover virtual services through a directory, improving the consumption of externally available services. Services and APIs can now be discovered faster and with better documentation to plan development.

CA Service Virtualization combined with [CA API Gateway](#) offer intelligent policy-based routing of traffic between “real” and “virtual” services. CA API Gateway also enables organizations to leverage security standards such as OAuth, allowing developers to focus on business logic versus building security directly into the application.

CA API Gateway and CA Service Virtualization deliver a real-time feed from production to development, enabling faster creation of virtual services with greater fidelity. The more comprehensive, contextual and accurate the data, the better the virtualized environment that can be created for development and testing.

CA commissioned Forrester Consulting to conduct a Total Economic Impact™ (TEI) study and examine the potential ROI enterprises may realize by deploying CA Service Virtualization. Forrester interviewed four organizations that had implemented this solution in their enterprise. Taken as a whole, this composite company reported:

- Early defect detection in QA, leading to $1 million in average annual savings
- Development and testing efficiencies equating to $306,000 per year in savings
- Testing environment infrastructure cost avoidance of $2,747,971

[Read the full study](#)
To learn more, visit: