Level Up Your API Testing

How to Move Beyond the Basics with Data-Driven Testing and Automation

SMARTBEAR

Contents

Introduction

Application Programming Interfaces, or APIs, facilitate communication between two applications. For example, a back-end application that interfaces with a database may want to convey data to both a web application and a mobile application. They have gone from a niche enterprise use case to the de facto standard for modern applications that need to communicate with various front-end user interfaces, including web and mobile apps.

Since they have become such an integral part of modern applications, it's important to move beyond basic endpoint testing and build automated test suites that ensure consistent responses and robust performance under stress. Automated discovery tools can provide a complete picture of an API to ensure complete test coverage, while automation tools can help ensure that test suites are run on a regular basis to catch errors or bottlenecks before production.

In this ebook, we will take a look at how to level up your API tests, from the planning stage to the automation stage.

Planning an API

There was a time when APIs were second class citizens. Most users accessed data through websites tied to a database and APIs were only developed for specific use cases where data had to be shared with non-web applications or external websites. For example, financial service providers may have developed an API for other websites to show the latest stock quotes. Or, airlines may have developed APIs for online travel agents, or OTAs.

The rise of JavaScript frameworks, like React and Vue, have made APIs a much more foundational part of web application development. At the same time, many large monolithic applications have been broken down into micro services that each involve their own APIs. Modern applications must also interact with a growing number of other services, including various mobile apps and internet of things devices.

APIs have become so important that Facebook developed GraphQL to simplify the way that clients consume APIs. Rather than hitting REST API endpoints and collecting pieces of data, GraphQL lets clients query for exactly the data that they need. These technologies could make APIs even more efficient in the future by eliminating unnecessary queries, reducing query sizes and improving overall API performance.

Let's take a look at what API-first development entails and how to start planning your API before writing a single line of code.

API-First Development

API-first development is the idea of starting development projects with API design. Using an API description language, development teams can

establish a contract for how the API is supposed to behave. Front- and back-end developers collaborate with stakeholders to develop the API design before any code is written and thereby minimize any potential issues, such as missing data, unnecessary bloat or other issues that could impact performance.

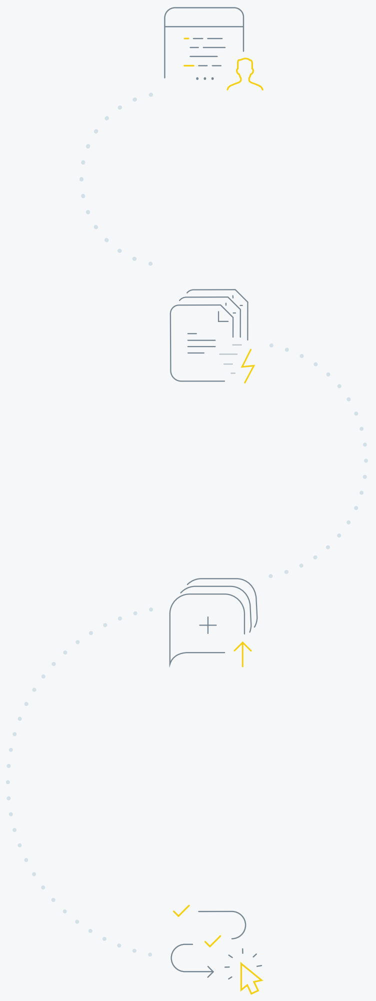
There are several key benefits to taking this approach:

- **Better Developer Experience:** Developers rely on APIs when developing front-end applications, which means that it's critical for the API to be reliable and up-to-date. Prioritizing API development ensures these goals are met.
- **Faster Documentation:** API documentation is critical to maintaining developer productivity. It's easy to ensure that documentation is up-to-date when it's the starting point for the entire development process.
- **Greater Adoption:** Outward-facing APIs require extensive documentation and reliability in order to encourage adoption. An API-first development approach helps ensure that these prerequisites are met to maximize adoption.
- **Better User Experience:** Well-tested APIs have fewer errors and performance bottlenecks that can degrade the user experience since they are caught before production when users can possibly see them.

The good news is that new tools and techniques have made it easier than ever to adopt an API-first development strategy in your organization. When combined with other Agile practices, these strategies can help ensure that your API is a well-thought-out foundation for your application rather than an unwieldy mess that will need to be refactored.

How to Plan Your API

API-first development means that API planning is the very first step of your entire development process. Before writing a single line of code, it's important to take a step back and work with every stakeholder involved with a project to come up with a plan to build the underlying API, including web developers, mobile developers and business analysts.



It's easy to get started with API-first development:

- **Brainstorm** the key features and capabilities that the application must include and write down potential endpoints based on those use cases.
- **Define** the stakeholders in the organization and ensure that you get buy-in from all of them when developing the API, including business and technical teams.
- **Design** the API contract that establishes the standards and best practices for API design. For example, the naming conventions, error codes, versioning strategy, etc.
- **Create a style guide** that puts these details into writing and consider using tools like SwaggerHub to guide and enforce these constraints throughout all APIs in your organization.
- **Implement** a governance process to ensure the integrity of the API. For example, you may want to include API-related items in code reviews to check compliance.
- **Automate** as much as possible to simplify the API development process. For instance, SwaggerHub can automate API documentation, style validation, API mocking and versions.
- **Track and manage** your API portfolio in order to avoid writing duplicate code. As the project grows, it becomes harder to track APIs and their various dependencies.
- **Create a portal** for internal developers where all information about an API is stored, including the specification, documentation and other information.

SwaggerHub is a great way to create these initial API standards before you start to write code. You can easily write API documentation that's automatically converted into an instant visualization that lets anyone interact with the API while you're still defining it. You can even generate server stubs and client libraries for your API in every popular language.

What's Next?

In the next section, we will take a look at how to get started writing API tests on a basic level before diving deeper into how to take a data-driven approach and introduce automation.

API Testing Basics

Most developers are familiar with test-driven development and the idea of unit and integration tests. When developing an API, backend developers will often use unit tests to ensure that methods return the correct results and front-end developers will use integration tests to confirm they work. Some APIs may also require security and load tests to ensure that the API is performant and secure from external attacks, which we will take a look at a bit later.

API tests are often written using the language and framework of the front or backend. For example, Ruby on Rails API tests may be written using Ruby and Rspec, and simply assert that a given endpoint returns a successful response header and the correct data structure. Similarly, front-end web applications may use Selenium, or a tool like TestComplete, to ensure that the APIs return the proper results. Mobile apps may use Xcode or a tool like Bitbar.

Let's take a look at the basic components of API tests and the tools that you need to get started.

API Testing Components

API tests contain many of the same components as conventional web applications, but there are some important differences to keep in mind. For example, API tests may include interoperability tests, compliance tests or penetration tests depending on the type of API and the use case. There may also be many different platforms involved rather than a single one.

The most common types of tests include:

- **Unit tests** ensure that each method or function returns the correct result on the backend. For example, a user sign-in API endpoint test might ensure that the correct credentials return the proper response header and JSON result.
- **Integration** tests ensure that the API produces results that appear correctly on the front-end. For instance, a web application might test that an API-based user sign-in works by simulating the user sign-in process with Selenium.
- **Performance** tests general load, stress and spikes to validate whether APIs can handle the traffic. For instance, a load test might simulate 10,000 hits to a given endpoint and verify that the error rate is within acceptable boundaries.
- **Security** tests look for cross-site scripting vulnerabilities, malformed XML, SQL injection and other potential issues. For instance, security tests may attempt an SQL injection into a POST route and look for an error message to appear.

Other less common types of API tests include:

- **Interoperability** tests ensure that SOAP APIs are accessible to the correct applications.
- **WS compliance** tests ensure that WS standards are properly implemented and utilized.

If you're adding tests to an existing application, you may want to prioritize your test coverage rather than trying to reach complete test coverage at the onset. A good example is focusing performance or load tests on instances where there's high usage, such as the add to cart process rather than a coupon code process for an ecommerce shop.

Starting with a Unit Test

Unit tests are designed to ensure that APIs return the correct HTTP status code and that the response is properly structured. In some cases, unit tests may also ensure that the correct response is returned (e.g. the result of a calculation), although those algorithms may already be covered by unit tests elsewhere in the code base. These unit tests are usually the starting point for most API tests and provide a baseline level of quality assurance.

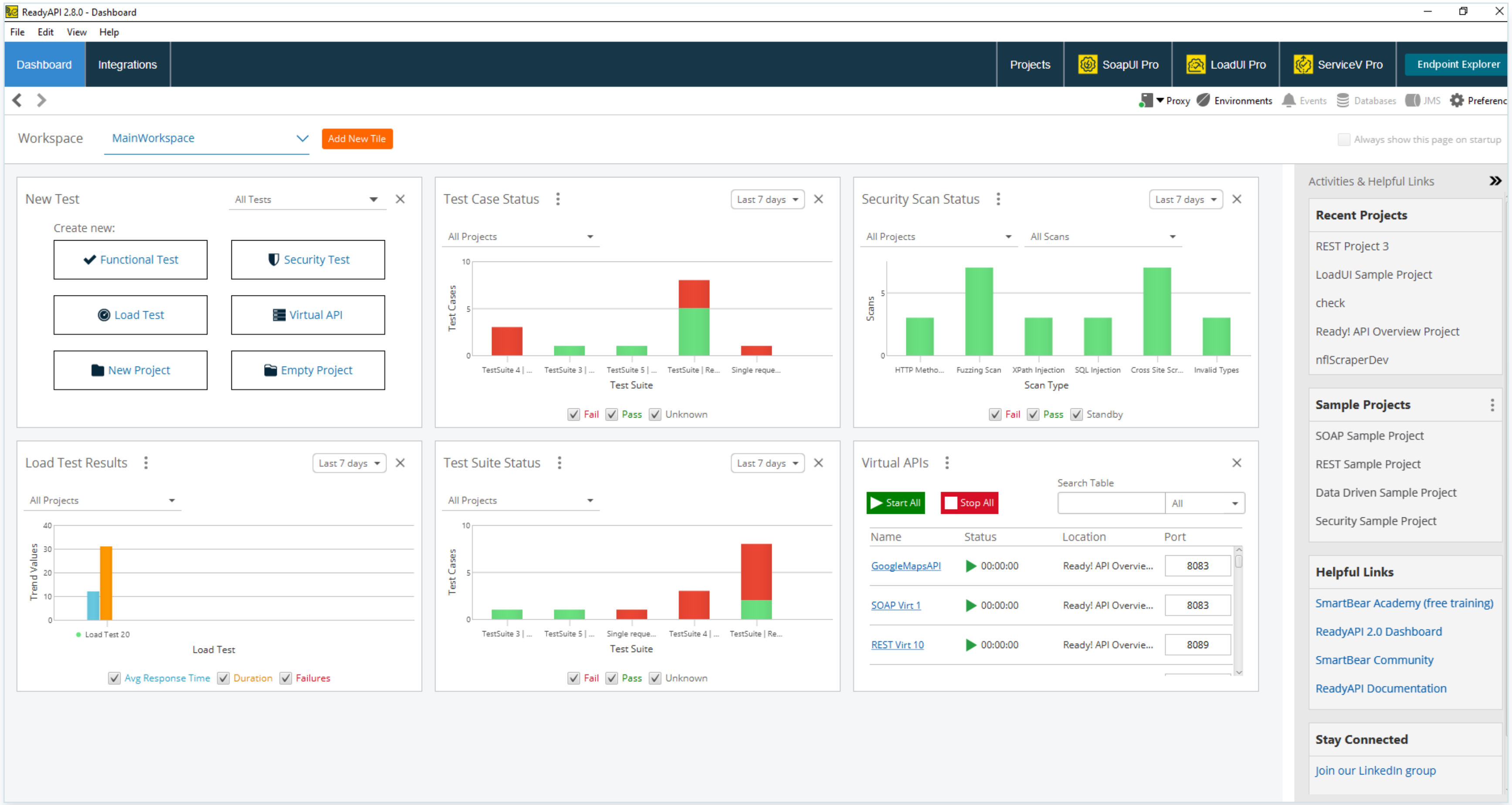
Let's take a look at a sample API unit test for a Ruby on Rails application using Rspec *(fig. 1)*.

fig.1

```
require 'rails_helper'

describe Api::V1::UsersController do
    describe "GET #index" do
        before do
            get :index
        end

        it "returns success" do
            expect(response).to have_http_status(:success)
        end

        it "contains a list of users" do
            json_response = JSON.parse(response.body)
            expet(json_response.keys).to match_array([:id, :users])
        end
    end
end
```

In this test, we ensure that the users' controller returns a success response header and an array of users for the index action. The test confirms that the backend successfully returns a result and that the result is correctly formatted. However, it does not test that the data is correct or that the response is rendered correctly on the front-end.

ReadyAPI makes it easier to create these tests with an easy-to-use user interface rather than code. For example, you can define the route and create assertions for everything from HTTP response codes to service level agreement (SLA) requirements. The failures can be configured to report in the same way as a code-based test could handle the failures.



Name	Status	Location	Port
GoogleMapsAPI	00:00:00	Ready! API Overvie...	8083
SOAP Virt 1	00:00:00	Ready! API Overvie...	8083
REST Virt 10	00:00:00	Ready! API Overvie...	8089

Creating an Integration Test

Integration tests are designed to connect the dots and ensure that an API works on a functional level. For example, a unit test may ensure that a sign-in endpoint correctly returns a token for an authorized user, but an integration test ensures that a user can actually input their username and password, receive a token and be logged into the application. These tests are necessary to ensure that the entire piece of functionality using the API works correctly.

Let's take a look at a sample integration test for a front-end sign-in process using React *(fig. 2)*.

In this example, the front-end JavaScript application tests to ensure that the user can actually login by simulating the experience from their perspective. These tests may need to be repeated for each front-end application, including web applications, mobile applications or other API consumers in order to ensure that everything works.

Building It into a CI Server

Continuous integration and deployment servers run tests on an ongoing basis in order to ensure that

fig.2

```
it('signs in with the correct credentials', () => {
      wrapper = shallow(<Login/>);
      wrapper.find('input[type="text"]').simulate('change', {
         target: {
               name: 'username',
               value: 'johndoe@email.com'
      });
      wrapper.find('input[type="password"]').simulate('change', {
         target: {
               name: 'password',
               value: 'passw0rd123'
            }
      });
      wrapper.find('button').simulate('click');
      expect(wrapper.state('isSignedIn')).toBe(true);
});
```

no new code commits breaks existing functionality. That way, developers can be confident in making more frequent releases since they know that everything will work as expected, and stakeholders can be more confident in the final products.

API tests are often built into these processes to ensure that nothing breaks. Unit tests are often incorporated into the backend CI/CD pipeline, while integration tests may be included in the CI/CD processes associated with each front-end client project. Load tests, security tests and other API tests can also be built into CI/CD processes to ensure coverage.

We will discuss continuous integration and deployment in greater detail later in this guide.

What's Next?

In the next section, we will look at some advanced topics when it comes to API testing before diving into test automation and how ReadyAPI can help simplify your workflow.

Beyond the Basics

Basic REST API tests ensure that basic requests and responses work with unit and integration tests. However, real-world APIs involve a lot more variables that can make the testing processes more complex. For example, you may use non-REST APIs or require load testing to ensure that it can meet service level agreement (SLA) requirements.

Let's take a look at some advanced API testing topics, including testing alternative APIs (e.g. GraphQL) and implementing other types of tests (e.g. load tests).

Better Test Coverage

Many developers focus on writing simple API tests that ensure requests to an endpoint return the correct response. For example, they may ensure that a sign in endpoint returns a token if the credentials are correct. These tests ensure that the API functions properly, but they don't account for all of the possible scenarios that could occur.

A common mistake when testing APIs is skipping past so-called sad paths — or ensuring that error messages are correct. Many client applications rely on HTTP error codes to display custom error messages or take certain actions, so it's important to ensure that you're sending the right error codes for the error that's occurring along with any messages.

In addition to testing sad paths, it's important to ensure that your testing with a wide enough range of data. An API endpoint that accepts and processes multiple file types should have tests that account for each individual file type rather than the most popular file type. That way, you know that the API will work for all users and not just the most common.

Additional Test Types

Most developers are familiar with unit and integration tests, but there are many other types of tests that are helpful to ensure that APIs are accurate and performant.

Performance tests ensure that your API can handle large numbers of visitors without an adverse impact to response times or error rates. For example, an ecommerce API may want to ensure that holiday traffic doesn't crash a server at an inopportune time or you may want to ensure that your API is capable of meeting service level agreement (SLA) requirements.

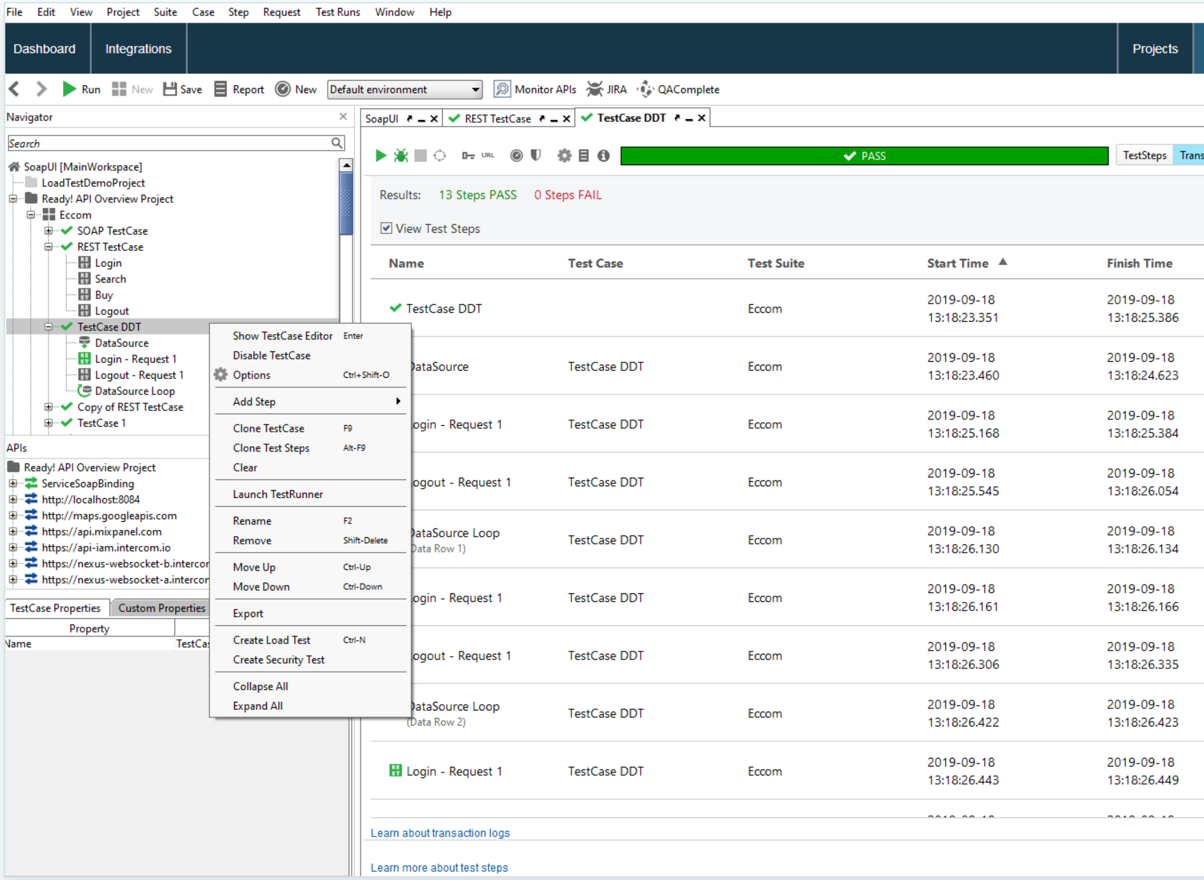
There are many different ways to load test APIs, but LoadUI Pro , part of the ReadyAPI suite, makes the process extremely straightforward. You can simply define the endpoint that you'd like to test, set the load you'd like to test and see the results when the load test has finished. You can create assertions and easily integrate the tests into your CI/CD workflows.

Security tests ensure that your API is secure from external attacks — especially if you provide public APIs. These tests typically look for vulnerabilities in the form of SQL injection, where malicious values can be passed into an API query to execute against a database. Endpoints should also be locked down with the proper permissions.

SoapUI Pro, also part of the ReadyAPI suite of API quality tools, enables you to create and run a wide range of security tests for your APIs to ensure that they're not vulnerable to malicious attacks. These tests may try to exploit bad handling of values outside of defined ranges in JSON POST requests, attempt to exploit bad handling of the attached files or exploit bad database integration code.

SOAP, GraphQL & Other APIs

HTTP REST APIs may be the most popular APIs in production, but there are many different types of APIs protocols and architectures. Each type of API



Results: 13 Steps PASS 0 Steps FAIL

Name	Test Case	Test Suite	Start Time ▲	Finish Time
✓ TestCase DDT		Eccom	2019-09-18 13:18:23.351	2019-09-18 13:18:25.386
ataSource	TestCase DDT	Eccom	2019-09-18 13:18:23.460	2019-09-18 13:18:24.623
ogin - Request 1	TestCase DDT	Eccom	2019-09-18 13:18:25.168	2019-09-18 13:18:25.384
ogout - Request 1	TestCase DDT	Eccom	2019-09-18 13:18:25.545	2019-09-18 13:18:26.054
ataSource Loop ata Row 1)	TestCase DDT	Eccom	2019-09-18 13:18:26.130	2019-09-18 13:18:26.134
ogin - Request 1	TestCase DDT	Eccom	2019-09-18 13:18:26.161	2019-09-18 13:18:26.166
ogout - Request 1	TestCase DDT	Eccom	2019-09-18 13:18:26.306	2019-09-18 13:18:26.335
ataSource Loop (Data Row 2)	TestCase DDT	Eccom	2019-09-18 13:18:26.422	2019-09-18 13:18:26.423
Login - Request 1	TestCase DDT	Eccom	2019-09-18 13:18:26.443	2019-09-18 13:18:26.449

may require unique tests to ensure that it's functioning properly. It's important to include these tests in order to confirm that all aspects of the API are working correctly rather than just the responses.

Simple Object Access Protocol, referred to as SOAP, is a protocol that has more standards than conventional HTTP REST APIs, such as security and messaging. The key difference is that SOAP APIs generally have tighter security, standardized messaging functionality and ACID compliance to protect databases by defining exactly how transactions can interact with databases.

GraphQL is a query language for APIs and a runtime for fulfilling those queries with existing data. By providing a complete and understandable description of data within an API, clients can ask for exactly what they need and nothing more, which makes it easier to maintain APIs, reduce overhead costs and enable powerful developer tools.

What's Next?

In the next section, we will look at how to take a data-driven approach to testing your API before diving into test automation and how ReadyAPI can ease your workflow.

Data-Driven Approaches

Most development teams strive for complete test coverage, but few come close to achieving it. Unless a project began with diligent test-driven development, it's hard to wrap your head around what needs to be done. The good news is that there are new data-driven tools that can make it easier to achieve complete test coverage and automate the testing process.

Let's take a look at some strategies for automating test-driven development and taking a data-driven approach to decision making when writing tests.

Discovering APIs and Usage

A big challenge for existing APIs that lack test coverage is understanding all of the endpoints. Many REST APIs do not have descriptive definitions outlining their expected behaviors, which means test engineers or developers must poke around to explore the correct behaviors in order to create assertions that can form the basis for an automated test suite.

ReadyAPI's API Discovery feature lets you watch HTTP traffic in the ReadyAPI internal browser or through an external browser that uses ReadyAPI as a proxy. After capturing all of the REST traffic, the platform writes it to a traffic log that you can browse through to find specific types of traffic. This eliminates the need to explore an API to find behaviors.

The data from the discovery process can be used to create test cases, create virtual services or configure data quickly without starting from scratch. In other words, developers can automate the exploration process and focus on actually writing tests, which translates to less time wasted and more test coverage for mission-critical APIs.

Generate Request Parameters

API tests require request parameters in order to ensure that an endpoint works. While there are some tools to generate fake data, it can be challenging to come up with request parameters that accurately mimic the production environment — especially if there's little API documentation that covers what request parameters should look like.

ReadyAPI can automatically generate values for parameters in each request test step. The value type is deduced from the type of the parameter that is specified in the service definition, or if that's not possible, from the parameter name. When a test case is run, ReadyAPI randomly generates values from the specific type and inserts them as request parameters.

Using Smart Assertions

A related challenge is understanding what API endpoints should be returning in order to create an assertion. Test engineers or developers typically have to run an API endpoint with a tool like Postman, see what is returned and then write an assertion that can be used in a test case. The good news is that this process can be easily automated.

ReadyAPI's Smart Assertion feature automatically creates validation and assert points against your API. It also gets smarter over time, intelligently recognizing dynamic values that change adjusting assertions. That way, developers can spend less time writing assertions and achieve greater test coverage during the time they have to devote.

These assertions can be easily created in a GUI interface that's faster and easier than writing code. Rather than dealing with syntax errors or other code-related issues, your team can quickly write effective tests that integrate with common test automation tools to work toward complete test coverage.

What's Next?

In the next section, we will take a look at how to implement test automation before going full circle and showing you how ReadyAPI can improve every part of the process.

Test Automation

Test automation is the best way to realize the benefits of test-driven development practices. By running the test suite before each merge or deploy, you can ensure that new code doesn't break existing code and confidently deploy applications to production. The key is efficiently incorporating test automation into your Agile development workflow.

Let's take a look at how to setup API test automation and some best practices to keep in mind.

Continuous Integration & Delivery

Continuous integration (CI) is the process of merging code from multiple developers into a shared mainline several times per day. The goal is to avoid integration issues or merge conflicts that can arise when many different developers are working on massive changes to the same code base, as well as to take advantage of continuous delivery (CD).

Most CI/CD workflows look something like this:

- A developer checks out a master branch and starts a new branch with his or her changes.
- The developer finishes those changes pushes the branch to a remote shared code repository.
- The push triggers a continuous integration server that checks for any merge conflicts and runs the test suite to ensure that the new code doesn't break any functionality.
- The new code is merged into the master branch if everything passes or must go back for changes if something broke.

There are many different continuous integration servers out there, but Jenkins is one of the most popular options. The open-source platform can be used as a simple CI server that runs tests or operates as a continuous delivery hub that deploys passing code on production servers. There are also many plugins that interface with common testing and delivery tools.

API tests can be automated in the same way as many other tests. For instance, unit and integration tests covering API endpoints can be easily added to a test suite that's triggered by a merge and run on a CI server like Jenkins. Performance tests are often run on a less frequent basis since they involve greater resources in many cases.

Best Practices to Consider

There are countless different test automation approaches, techniques, frameworks tools and strategies. With so many options, it's easy to see the process as more of a problem than a solution. The good news is that there are several standards that can help you get started on the right foot.

Some test automation best practices include:

- Avoid Code Duplication - Many API tests share the same setup processes, such as connecting to the server, authenticating a user or evaluating a request. It's a good idea to reuse as much as possible to avoid excessive code.
- Create Useful Reports - The goal of API testing is to help developers with debugging, so it's important to provide helpful messages and configure test reports following each CI run that can help trace potential issues.
- Ensure They're Flexible - Write tests that can run under different SUT configurations to ensure that they aren't limited by the system. That way, it's easier to update the tests when there are changes to the infrastructure.
- Segregate the Tests - Unit and integration tests may be run with each merge, but performance or security tests may only need to run before a major deploy. These tests may have longer run times or greater resource usage.

How to Integrate with CI/CDs

Most unit and integration tests can be easily incorporated into a CI/CD process via the command line. For example, a Ruby on Rails application that serves an API may include the command `bundle exec rspec` that runs the entire test suite, including unit and integration tests that are built into the application. The results may be published to a report or simply output on a CLI.

In addition to unit and integration tests, many performance tests can be automated in a similar way. JMeter, a popular open-source load testing framework, can be configured to run on Jenkins or other CI servers via command line or using the JMeter plugin for Jenkins. Many other load testing frameworks operate in a similar way via the CLI or plugins.

ReadyAPI simplifies the process even more through built-in integrations with common CI/CD tools like Jenkins. Using TestRunner, you can control what environments test run on, run specific test cases or entire suites, run specific subsets of tests using tags or provide parameters from a previous continuous integration step, such as dynamic host names.

What's Next?

In the next section, we will take a look at how ReadyAPI can help simplify API testing and help you achieve complete coverage, as well as answer some common questions about the service and API testing in general.

How ReadyAPI Simplifies API Testing

SMARTBEAR
ReadyAPI

ReadyAPI Makes API Testing Simple

Start Free Trial

APIs have become increasingly important as a communication tool connecting databases and application servers with mobile apps, web apps, IoT devices and other clients. While most development teams at least test endpoints, it's important to move beyond these basic tests toward more complete test coverage that includes performance tests, security tests and other types of tests, as well as configure them within a CI/CD pipeline.

In this course, we have discussed many of the strategies and best practices surrounding these topics, from the smart planning at the onset of a project to effective continuous integration and deployment processes.

ReadyAPI streamlines many of these processes with an easy-to-use, yet comprehensive, GUI-based interface for running a wide range of API tests. The platform's automation capabilities make it easy to understand an API's endpoints, generate request parameters and create smart assertions with a few clicks. You can also integrate with many different CI/CD tools to build API tests into your existing test automation workflow.

The ReadyAPI platform features:

- Projects to create and maintain test cases, data sources and object components in a single place for easy reusability and management.
- Dashboards that provide real-time, end-to-end insights into your functional, security and load tests with test runs, results and usage details.
- Integrations with nearly any CI/CD workflow, enabling teams to run their API test suites each time that new code is deployed.
- Scripting support to crate functional, load or security tests using either Groovy or JavaScript as a scripting language.
- Discovery tools that make it easy to listen to live API traffic, record responses and reuse the data to create test cases, virtual services or configure data.

Questions & Answers About ReadyAPI

How can I get started with ReadyAPI?

You can get started with ReadyAPI by starting a free trial today! If you'd like more information, you can also schedule a free demo to learn more about the product and how it could fit into your existing workflow.

What level of technical experience is required?

ReadyAPI unifies the way development and QA teams collaborate on API quality by making otherwise complex tasks a matter of a few clicks. With features crafted to simplify monotonous tasks and authentication, improve coverage, and maximize test reusability across the software lifecycle, ReadyAPI helps teams of all ability levels to standardize their approach to API testing.

What type of reporting will this tool provide?

ReadyAPI has a host of pre-built reports and reporting templates to help you deliver on just the information that matters to you the most. More than just visual reports, the product provides numerous export formats so that your continuous integration results can be stored or processed into another system for later analytics.

What support options are available?

ReadyAPI includes extensive documentation that covers all of our basic functionality, as well as email and phone support for anything above and beyond.

Leveling Up Your API Testing with *Automation*

The ReadyAPI platform accelerates functional, security, and load testingof RESTful, SOAP, GraphQL and other web services right inside your CI/CD pipeline.results with unprecedented accuracy.

Try ReadyAPI for Free

SMARTBEAR