


The Most Common API Errors to Watch for *in* Retail

PRESENTED BY

APIFORTRESS

Introduction



Did you know that most APIs suffer some sort of failure everyday? A lot of people are not aware of this fact because they are not testing for it. Uptime and functional uptime are two very different things. Unless a customer notifies you, or there is a catastrophic failure, most API issues are quietly eating away at your bottomline.

This is true of your internal APIs, and perhaps more critically, your affiliate APIs. APIs are as vulnerable to flaws and irregularities as your websites, apps, and firewalls; and should be receiving the same amount of attention.

API Fortress has been working in the testing field since 2013. We have seen thousands of errors, and to help API providers and consumers we have listed some of the most common API issues we see regularly. These are all real world examples of our customers in the retail industry.



What Is An **API**?

TERMS

APIs: Think of them as pipes that carry data between systems. It is how the Instagram app can send and receive pictures from the Instagram servers.

Webservice: Has unfortunately become mostly interchangeable with API, so we will continue that trend.

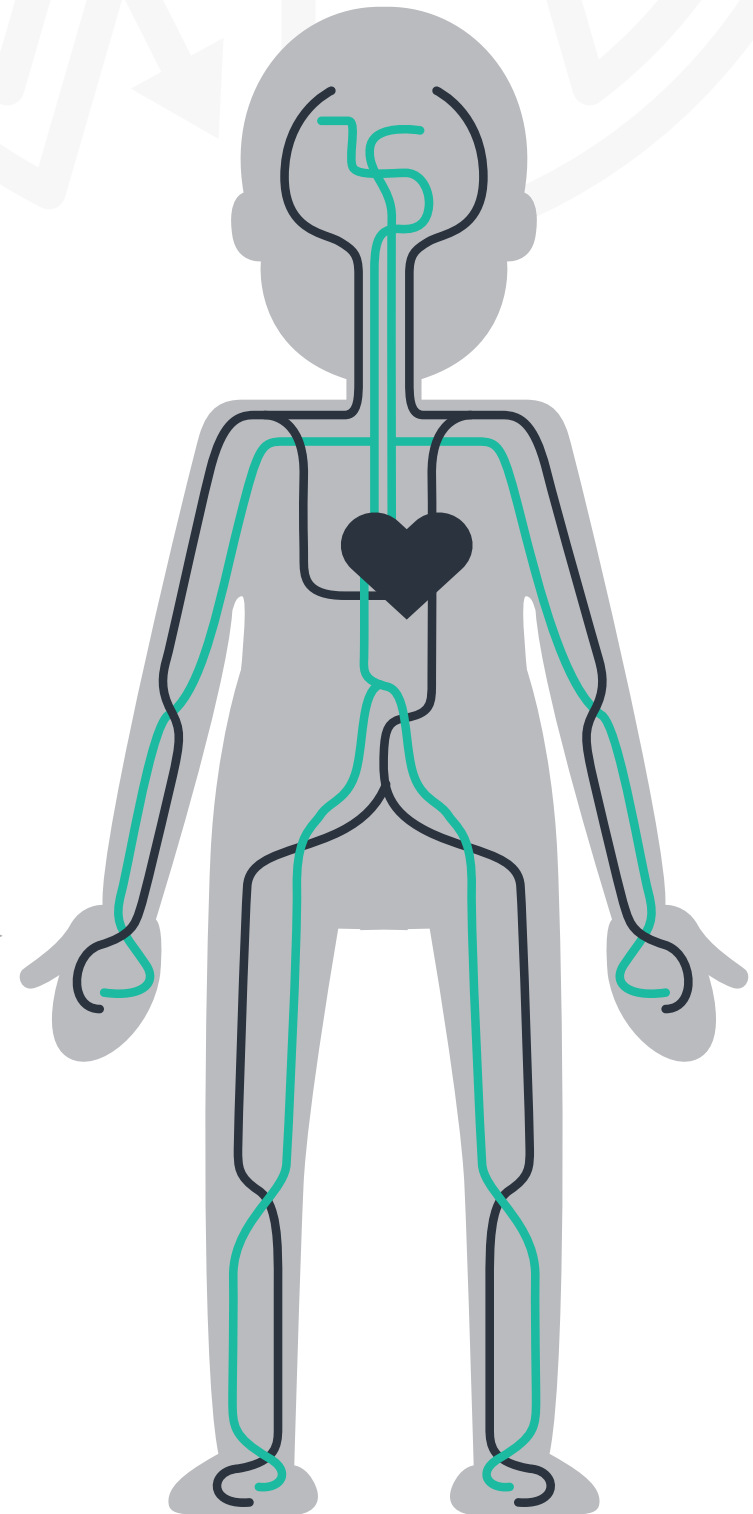
API Provider: The company that owns the API that is being used.

API Consumer: A developer, company, or app that uses an API.

How Do I “**Call**” an API

For this White Paper we will be sticking to web APIs that are called using HTTP. Here is an example of an API, feel free copy and paste it into your browser’s URL bar:

<https://mastiff.apifortress.com/app/api/examples/retail/product?id=611>

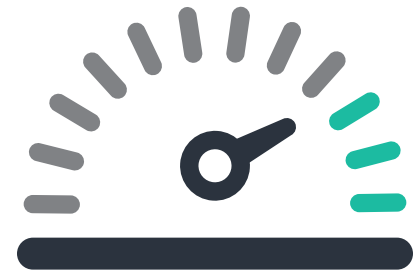


What Does an API Look Like?

Use the link above and see for yourself! Also, here is it. It is usually human-readable text.

Here is an example of an e-commerce product API for a specific jacket:

```
{
  success: true,
  status: 200,
  content:
    {
      _id: "53077f2ee4b0efa630df2ec1",
      id: 611,
      name: "Beautiful Jacket",
      type: "clothing",
      price: "59.99",
      shipping:
        {
          available: true,
          zones: "america",
          modes: ["fast","regular"]
        }
    }
}
```



How Important Are APIs?

Think of APIs as the veins in a human body. While the heart is the most important organ, without vessels to carry the blood (data) to all the other critical organs the heart can not do its job. APIs connect your platform to its websites, apps, and kiosks.

When NULL Costs You *Thousands*

“WHAT DO YOU MEAN THERE ARE 3,000 ITEMS MISSING FROM OUR WEBSITE?!”

When we first onboarded this customer’s API the product listing endpoint had failures immediately. Somehow there were over 3,000 errors every time the test executed. The customer had never had an issue with that endpoint before, and when testing manually it seemed to work fine.

An API With a Leak

Upon deeper investigation of the error reports, the problem became clear. Each product has an object called ‘recipient.’ It is what they use to categorize products, and should be one of 17 options including men, women, babies, etc... In their listing of 50,000 products, over 3,000 items had ‘NULL’ for the category. That means “nothing” in API speak.

That meant 3,000 items with no category associated. The category option is how the website navigation can find and show you products under Men > Accessories > Scarves. That is thousands in potentially lost sales because of an API that never caused them concern before.

The Solution

It is critical to test up and downstream. Test that the API is working as expected, and then also test the end result where live data meets the API. It could be an easy way to find thousands in hidden revenue, much like this customer did.



When a 200 *Is NOT OK*

“WE JUST WANT YOU TO CONFIRM OUR 99.9% UPTIME.”

Junk food lobbies pay for research that, unsurprisingly, often find evidence that perhaps candy is *not that bad* for you. There are occasions where we are brought in under a similar guise, and that was the case with this large media company. Unfortunately, almost immediately we started firing errors to their Alerts channel on Slack.



The Hose Works But the Outdoor Faucet *Is Not On*

Our tests for this customer were checking performance, uptime, and diving into the payload itself. The API's response had a Status Code of 200, which should mean everything is fine. This is why their team did not think there was an issue

Unfortunately, the monitoring team was not aware of a decision made by the API development team. They chose to have all responses return a 200, even if there was an error. These are often called “soft error codes.” If the database feeding the API was down, which it was in this case, the API would still respond 200. Upon closer inspection you would notice the payload was mostly blank with an error message. This leads to inaccurate uptime numbers, and potentially caching a payload with an error.

The Solution

Testing for uptime is important, but even more important is testing for *functional uptime*. More work is required to get a truly accurate view of your API's health than a Status Code check.



Automatically Generated *Inaccuracies*

“THAT’S NOT MY JOB.”

API developers, documentation writers, and users are often three different teams. This makes proper communication of changes pivotal, and unsurprisingly, leads to many breakdowns. A customer’s eCommerce analytics team noticed a significant problem in sales and asked us for help. Specifically, new earrings being listed were not selling, but older listings saw no dip in sales.

Oh God, *I Lost My Earring!*

We reviewed the API and quickly discovered that earrings were being listed as both ‘earrings’ and ‘jewelry.’ We spoke with the API development team and learned that a few weeks prior a developer had made a small “improvement” to the ‘type’ of products available.

The web team had a stringent process where the navigation was specifically curated to maximize conversions. This meant the new ‘earring’ type was not built for, and therefore was missing from the website’s navigation.

The Solution

There are two learnings here. First, can API blueprinting and mocking platforms available today can help product teams plan how an API should function. That plan should be stuck to and test-ed against. There are tools today that allow you to create an API test from a definition file (Swagger, RAML, API Blueprint, etc...). Test your API using the original plan to **confirm it still conforms**.

Second, communication of API changes is key. Even for an API that is only internal, each change has to be communicated clearly to all parties in advance. Every change can be a major one.



A/R is Not *Quality Assurance*

“HEY, ACCOUNTING JUST TOLD US WE HAD NO REVENUE TUESDAY. ANY IDEAS?”

A large image company had an API that fed stock images to its customers, including a print-on-demand platform. This POD would help people print pictures of pugs on hats and mugs. Wednesday afternoon the image company received a frantic call from the printing company’s team with this message. They needed answers fast.



Did I Do That?

The image provider’s API team had pushed a small release Tuesday morning to clean up a few lines of code. In the process they had created a slight structural change. It did not affect all of their customers, but it was devastating to the POD. Their platform was written to read the API strictly, and was ill prepared to handle the change. This caused an error every time a user tried to search for an image.

The **Solution**

Continuous integration testing. Every new release should run through a series of tests on staging (pre-release) that confirm the results. They should be the same tests that confirm the live servers, so there is consistency. They can easily be made part of your deployment flow, especially if you are using tools such as Jenkins, Travis, CircleCI, etc...

How Do I *Mount* This Birthday Cake?

One of our customers has a very expansive catalog of items. They offer everything from car tires to apples. This can lead to a single search returning a series of products with very unique data and features. On a Thursday afternoon a bug with a succinct title was reported:

///////////////// **“EVERYTHING HAS MOUNTING INSTRUCTIONS.”**

It seemed funny but was a huge issue. No matter what you searched, once you clicked into a product it would contain all the correct information including “Mounting Instructions.” It required a vigilant QA to finally recognize the issue that many had seen but not consciously noticed.

Stop Trying to Mount Me

We came in to analyze the API and the tests they had created. The first thing we noticed was that the API did actually contain mounting instructions with every product. A clear API bug that led to a serious rollback of code.

The second was the testing methodology was severely flawed. While the customer did test the payload, their method of testing was lacking. The problem was that they were using a testing platform that had some constraints. The first was a lack of ability to search many different types of products, and the second was not being reactive to the type of product that was returned. A refrigerator and a dress should contain very different types of information, and your test should understand the expectations for both.


The Solution

Smart API tests allow for IF statements. Therefore, a great test would look at the product category and IF the category is a picture frame, then the mounting instructions should exist, if it is a shoe it should only contain shoe related information.



**“We Need A Full Blood Panel Workup”
- EVERY HOSPITAL TV SHOW EVER**

Cache Me If You Can



**“NONE OF OUR INTERNAL
ALERTING TOOLS ARE
REPORTING ANY ISSUES,
BUT YOURS HAS BEEN
GOING OFF FOR AN HOUR.”**

We received this message at 10pm from an online retailer. When we looked at our reports it showed the customer’s product details endpoint was often returning a 404 (which means ‘not found’). Yet, they were saying the endpoint had no issues when tested manually.

This required some diagnosis. First, we manually tried a few product IDs and they all passed. Then, we dove into the reports and saw that only a few of the IDs were failing. So what happened when we tested those specific product IDs by hand? They failed!

Most of our tests are full integration tests, meaning they go through full flows that an API consumer would experience. In this test, we first hit the product listing API, which gave us all active product IDs. Then, using those results, the platform dove into each product individually.

How could a call asking for all product IDs give us a bad one? The API manager was the culprit. It cached the listing endpoint, which is usually a good practice, but when not properly configured can lead to hours of outdated results.



The Solution

Merely exercising a singular endpoint is not enough. Always aim to reproduce an entire flow that your consumers may experience. Test everything *for* everything, and you will be able to uncover hard to catch issues such as this one.

About API Fortress

API Fortress is a complete API health platform. Test and monitor for performance, functional uptime, and accuracy. Test in production and as part of your continuous integration, on-premises or in the cloud. Get notifications, comprehensive reports, and a status page with the push of a button..

Learn more at apifortress.com, or signup for a [free account](#).