Nine Bugs that UI Testing *Could Not* Diagnose:

EBOOK | PRESENTED BY

APIFORTRESS

Introduction

In today's API economy, digital innovation decides which companies and brands rise to the top. Most IT and business leaders already understand this trend, and realize that release cycles must accelerate. However, with faster releases, many DevOps teams are seeing a decrease in software quality and process compliance. That is why solving the quality-at-speed challenge is at the heart of successful digital transformation.

It is also why most organizations now use a form of agile development, which has helped to solve for process compliance - but the issues of speed and quality remain. With that has come the proliferation of continuous integration and continuous delivery (CI/CD), which was specifically built to solve the speed and quality parts of the challenge.

Manual testing and unit testing are no longer sufficient to keep up with agile flows and CI/CD. Furthermore, APIs are inherently complex, which makes successful manual testing virtually impossible. Thus, implementing automated tools has quickly emerged as a necessary part of the Continuous Integration (automated testing) stage, now a best practice to solve the "quality" part of the challenge.

The new reality is that almost every modern app is a composite app, built from multiple services and microservices that depend on APIs. Only with [automated] API testing underwriting the entire modern software development lifecycle are DevOps teams able to derive immediate metrics about the fundamentals of quality: function, performance, reliability, security, and more. Now, teams can build in sprints with minimal bottlenecks.

Which means every delivery is now perfect.

Okay, that's far from the truth. Bugs will always exist: it's the reality of software development. The goal is to create processes that will minimize the amount of bugs that get through. With that said, not all CI/CD flows and automated tools are created equally.

In the following slides, we will explore nine of the most unique bugs that affected our customers. Bugs that went unchecked and were costing them time and money, until they started using API Fortress and upgraded their automated testing strategy.

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 inter-changeable with API, so we will continue that trend.

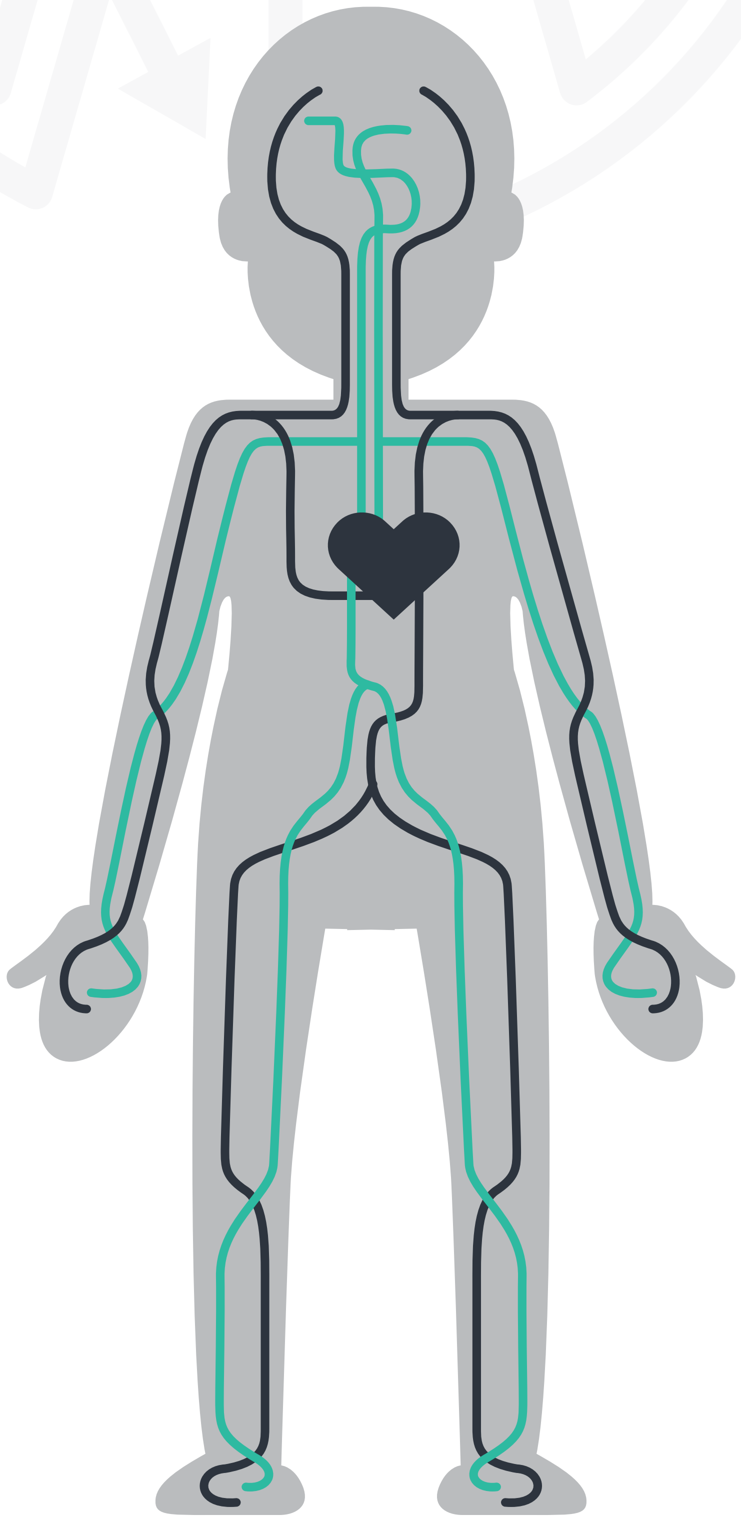
API Provider: The company that owns the API that is being used.

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:

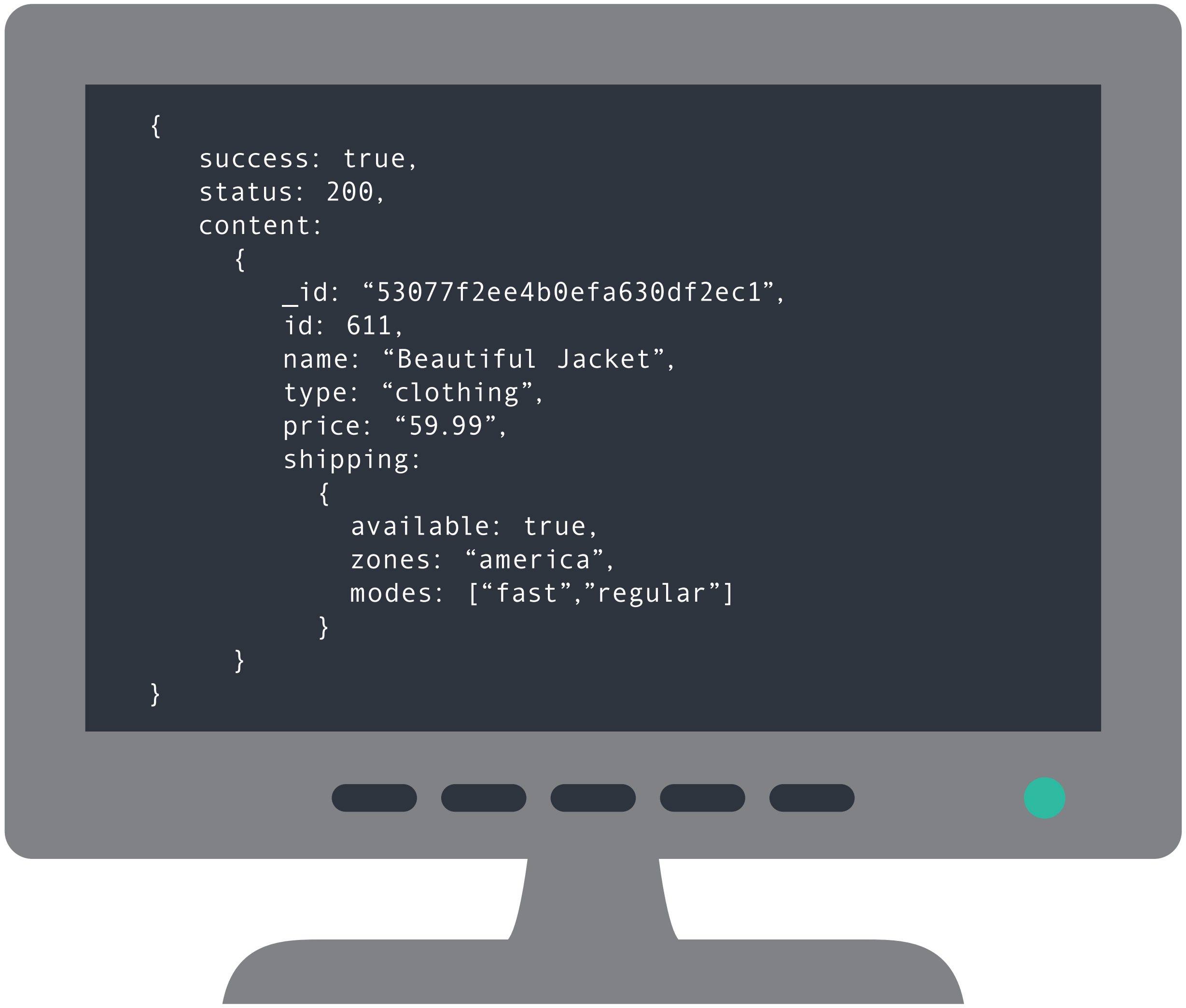
https://mastiff.apifortress.com/app/api/examples/retail/product?id=611



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.

“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.

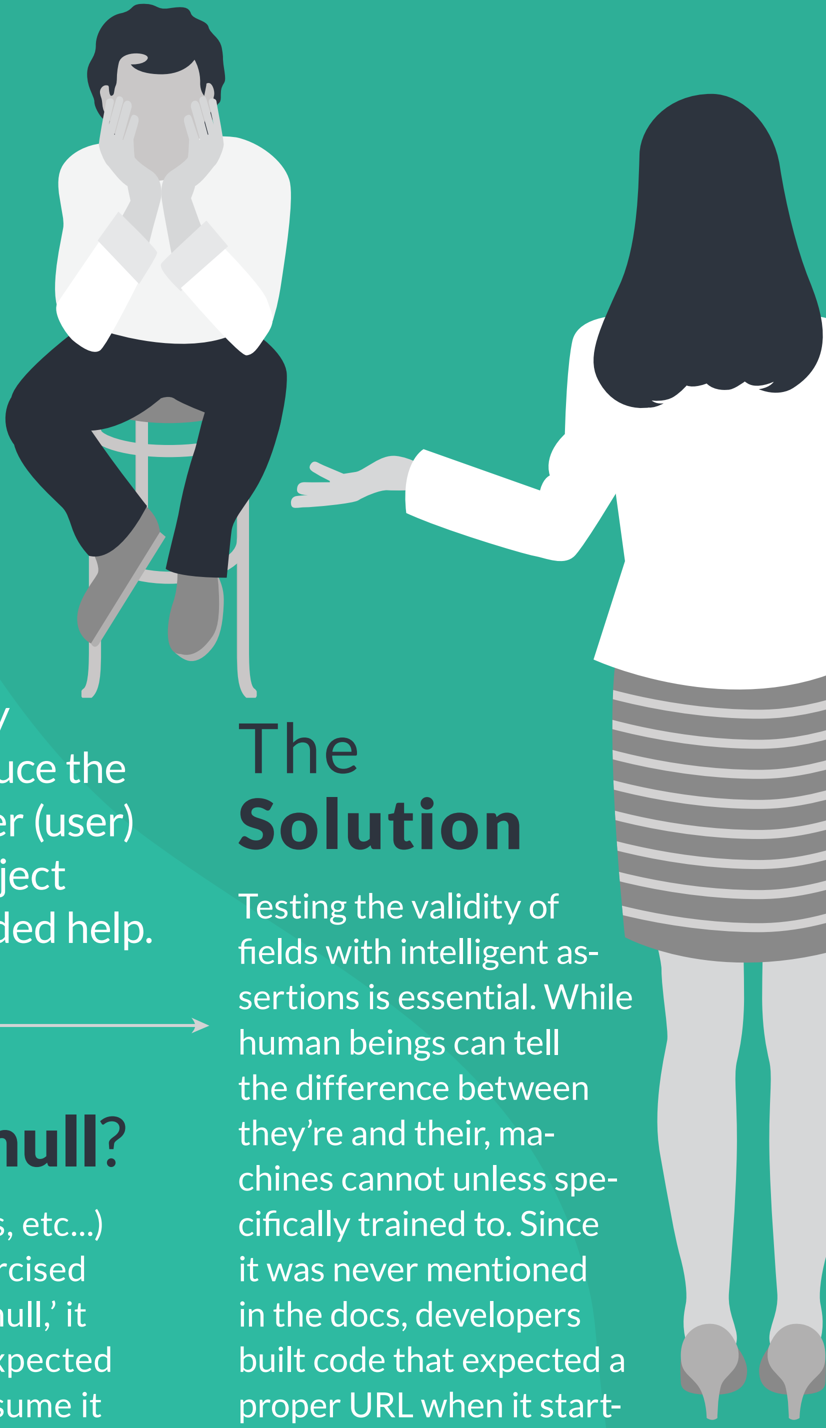
It’s Not You, *It’s Me*

“EVERYONE IS COMPLAINING, BUT NO ONE CAN PROVE IT’S REAL.”

We had a customer that was getting multiple reports of URL failures from their API consumers. As hard as they tried, they could not reproduce the issue. Without being able to reproduce the problem they could not decipher if it was a bug or a developer (user) issue. The API documentation clearly stated that the URL object should be either a properly formatted link or ‘null.’ They needed help.

What Happens When a URL is **http:null**?

URL problems can come in two flavors - error (encoding, double slashes, etc...) or a failure (absolute vs relative, URL vs URI). When we thoroughly exercised their API we saw that instead of the URL being properly formatted or ‘null,’ it would sometimes return as ‘http:null.’ Developers using the platform expected ‘null’ or a URL. If the field started with HTTP, then many apps would assume it was a proper URL. This is what led to the user complaints.



The **Solution**

Testing the validity of fields with intelligent assertions is essential. While human beings can tell the difference between they’re and their, machines cannot unless specifically trained to. Since it was never mentioned in the docs, developers built code that expected a proper URL when it started with ‘HTTP.’

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.

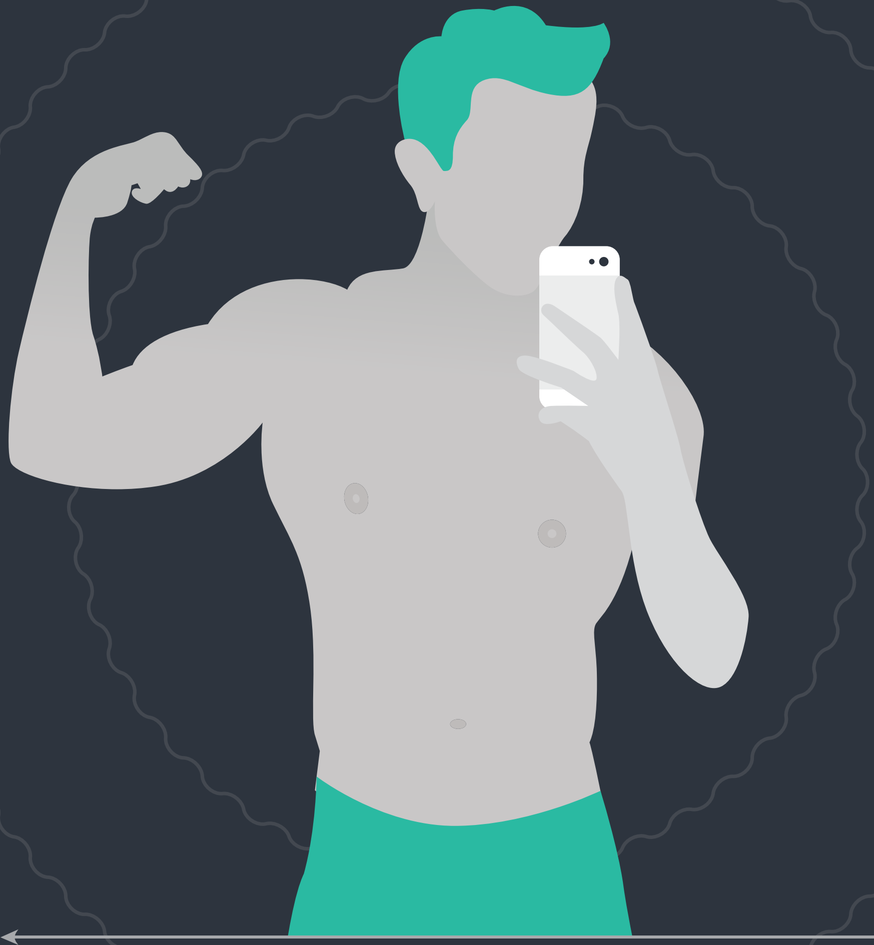
The **Solution**

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.

Am I Your *DataTYPE?*

"THIS WILL *NEVER* GO PUBLIC."

A naive thought from celebrities that take candid photos of themselves, and from internal webservice teams. This stock image company decided to jump on the "make it publicly consumable" bandwagon, and pushed their team to open internal services to the world. Soon after going live their forums were filled with news of a bug - "edited is not returning correctly." This confused their team so we were brought in to help.



What **Filter** Should I Use?

One of the more common endpoints of this customer's API had an object named 'edited.' There was limited documentation, but developers saw that it always seemed to return as 'False' and therefore built their apps with that expectation.

The problem was in communication. The company had built the 'edited' field to return one of two responses - False or an epoch (timestamp) of the last time it was updated. Developers saw this epoch as a random set of numbers and reported it as a bug. Developers are used to seeing errors and are quick to place blame. This was technically not a bug, but it led to just as much of a headache.

The **Solution**

There are no 100% standards in the API industry, but there are guidelines that are commonly accepted. Do not overthink your API. When possible stick to what people are used to - REST, JSON, consistent datatypes. Those three things will make life easier for yourself and future developers.

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 tested 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...

Je Fail Souvent, *Mais Il Est Un Secret*

Sometimes there are moments when something can be both boring and interesting at the same time. With that said, it is likely always important. When an API is streaming it must have a defined character set. Two important notes are that these ‘charsets’ do not contain every character in the set, and that the receiver will use this ‘charset’ to decode the API. Usually the characters are similar among western countries, but problems arise when an unexpected character comes along.

What Happens When **Francois** and ***François*** Collide?

For this eCommerce customer the name ‘François’ was in the data, but the declared charset could not decode the cédille character. When a platform can not decode a character it might replace it with a slew of different characters depending on the system. This time it changed the word to ‘Fran?ois.’ It may seem insignificant, but that character led to a problem with categorization, usernames, and mailing orders.

The **Solution**

Use proper payload monitors. A good payload monitor will verify every item in the payload, and make sure that all that data is appropriately readable for the receiving platform.

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.

About API Fortress

API Fortress is a continuous testing platform for APIs. It is the final piece to complete your continuous integration vision. One platform to test functionality, performance, and load. Save time with automated test generation, benefit from true cross-team collaboration, leverage your existing version control system, and seamlessly integrate with any CI/CD platform. Catch problems before they are pushed live—automatically. To learn more about why companies are switching to API Fortress, visit apifortress.com.

Enterprises and technology consulting service providers that register for a free trial of API Fortress (cloud or on-premises) are eligible to request a free API Testing Risk Assessment. A report will be generated with insights about your organization's measures to solve the quality-at-speed challenge. Your assessment will also include best practice guidelines for how to safely accelerate digital transformation journeys.

Contact Us

Will Hart
Vice President, Global
Strategic Accounts
API Fortress
will@apifortress.com
(203) 520-4296

Patrick Poulin
CEO
API Fortress
patrick@apifortress.com

Jason Loannides
Director, Sales Engineer
API Fortress
jason@apifortress.com