A Broadcom Company
technologies

White Paper

Ten Common API Errors and How to Avoid Them

BlazeMeter® Continuous Testing

Introduction

With the benefit of hindsight, it is easy to be wise. In 2000, Roy Fielding's doctoral dissertation presenting REST as an alternative to other distributed computing specifications now makes perfect sense. Fast forward a couple of years and we can all see why Jeff Bezos' mandate forcing every Amazon team to interact with each other through APIs was such a no-brainer.

Irrespective of academic milestones and mega-retailer epiphanies, one thing is clear: optimally exposing an organization's digital services through APIs in a well governed way is essential for success in an increasingly software-driven economy. But as we strive to decouple and define resources and make them available through APIs, there are many challenges. At the very least, improving interface support, ensuring discoverability, and extending monitoring beyond the simple up-down tests, towards determining whether an API is delivering the expected results. Testing is also critical, with methods needed to quickly validate and debug APIs, without impeding the flow of value to the business.

These requirements are well documented, but can be easily neglected when the easy to use frameworks like Flask for Python together with the Heroku or Zeit.now make developing and deploying an API seem as easy as serving fast food: fast, fun and (developer) friendly. But building a truly secure, resilient, supportable and highly performant API is no free lunch. Yes, we need to embrace the productivity benefits of new tools but without compromising API business goals. Analogous to a great chef who balances creativity with a fixation on detail and continuous taste testing, we need to deliver valuable API documentation, succinct error messages and great support, everything to meet a developer's expectations.

On the other side of the restaurant table, we have developers engaging our systems through the APIs we have prepared. These developers can make incorrect assumptions about how a particular end-point should work, neglect to read well-crafted documentation, or fail to correctly parse our error messages.

All this means that rigorous testing and monitoring is needed to quickly uncover API issues that would stay hidden without integration tests, or failing to accommodate how APIs will be used in the wild. Of course, all this demands great tools, but equally important is the ability for any tooling to incorporate knowledge and learnings from working with an active development community.

So, here are 10 common *gotchas* that can quickly relegate a well-intentioned API to the development food dumpster, why they can happen, and what to do to avoid them.

Ten Common API Mistakes

API Error #1: Making a Lousy Menu Choice

Just as we can choose a bad dish in a restaurant, we can sometimes use the wrong HTTP method, which can often be blamed on poor or unclear menus (documentation). The endpoint documentation often does not adequately detail what methods are supported, or the document uses the wrong verb entirely.

In this context, you should use tools with caution. Take curl, for example. If a user issues a GET request with the data option (-d), but forgets to specify -XGET, then it will automatically default to a POST request and include the `Content-Type:application/x-www -form-urlencoded` header.

In all cases, it is important that you carefully read documentation and not fall victim to making incorrect assumptions, especially when working with different tools and APIs. For example, the BlazeMeter® API uses a POST method when creating API test environments or steps, and a PUT method to modify them. On the other hand, the Stripe API uses a POST method to both create and update objects. While both are perfectly valid, the goal of an API provider is to remain consistent and keep the documentation clear and up to date.

API Error #2: Forgetting the Seasoning

Just like failing to add salt to a dish might cause a diner to wince, forgetting a single “s” can cause some surprises during API testing. Many APIs will only support HTTPS, while others may support a combo of HTTP for some endpoints and not others. But even if an API has the apparent flexibility to support both, there can still be problems. For example, some APIs will conveniently redirect HTTP requests to their HTTPS counterpart; however, not all frameworks are configured to follow a 302-status code.

An example is the Node.js ‘request’ module, which will follow a GET redirect by default, but has to be explicitly configured to follow non-GET responses (PUSH, POST and so on) as redirects to be explicitly configured to follow non-GET responses (PUSH, POST and so on) as redirects. It is also common for an API provider to stop supporting HTTP. Good providers will notify you well in advance through their websites, developer communities, and social media (a recent example is the Instant Payment Notification micro-site of PayPal); so, it is important to stay up to date and to integrate important update notices into your own developer communication channels.

On the API provider side, it is important that HTTPS strategies are in place to ensure the secure, private, and reliable connections that users, customers, and partners expect from your APIs. The process for getting certificates might previously have been used as an excuse for not moving to HTTPS, but now solutions like Let’s Encrypt, an open certificate authority, have helped lessen the pain.

API Error #3: Not Giving Good Service

A good waiter will find out if there is something wrong with your order and correct the problem with minimal fuss. Similarly, a good API error message will help developers quickly determine coding issues and what is needed to fix them.

HTTP provides 70+ status codes that you can use, but a pragmatic approach as outlined in a Steve Marx blog post suggests that you implement at least three status codes (200, 300, 500) and augment these with status codes that are consistent and have actionable meaning across multiple APIs. Another best practice is to use concise error messages with links guiding developers back to documentation.

As an API consumer, making assumptions can also cause problems. While a status code of 200 suggests that everything is okay, it is not always so clear cut. For example, Facebook's Graph API always returns status code 200, even when there is an error included in the response data. It is important, therefore, to be mindful when working with APIs. Make sure to carefully read the documentation and responses while you learn how they work.

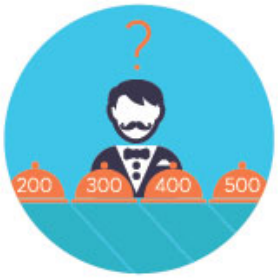
API Error #4: Do You Have a Reservation?

Have you ever tried to get a table at a popular restaurant in New York at 7:00 PM without a reservation? In the same way, some APIs require 'authorization' and you must be careful to not try to use 'authentication' instead. These and other word similarities can cause headaches. For example, OAuth 2 implementations usually require including an 'Authorization' header, but it is so easy to type 'Authentication' instead. It is a common problem. So, if a request keeps failing make sure that you are using the correct word.

Authorization header construction also has a few traps for the unwary. OAuth 2 tokens must be pre-pended with a "Bearer" to work, and with respect to authentication, common mistakes include:

- Forgetting the 'Basic' prefix (including a space).
- Not encoding 'username: password' or forgetting the colon.
- Falling foul to quirkiness, such as when an API provider only requires a username (often the API key), yet the provider still makes it necessary to place a colon after the username (even though there is no password).

API Error #5: Lost in Translation



Have you ever tried to order from a menu written in a foreign language when the waiter only speaks that language? It can be just as frustrating when you try to debug SSL errors when the returned messages are cryptic or do not provide indications on how to fix the problem. Complicating the situation is the way that many different technologies and tools handle SSL. But two of the most common SSL errors are easy to remedy: invalid certificates and incomplete certificate chains.

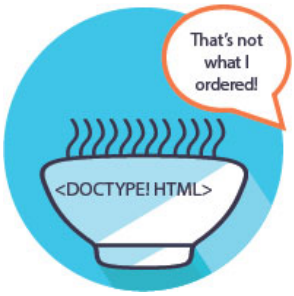
When you run into invalid certificate problems, first make sure that your API testing tool supports the certificate authority (CA) that created the SSL certificate. At BlazeMeter, we use the Mozilla CA Included Certificate List, but it is easy to find what your tool supports with a quick Google search. If the server's SSL certificate provider is on the list, the next step is to debug the issue with the SSLLabs SSL Server Test, which is a free tool that performs deep analysis of any public SSL web server.

API Error #6: Confusion in the Kitchen

You can never assume that the kitchen will be aware of your dietary requirements that you mentioned when you placed your order with the waiter. It is the same with APIs, especially when you are negotiating the type of information sent or received between client and servers through the Accept and Content-Type headers. While some APIs will conveniently accept requests that do not contain any of these headers (defaulting to good old JSON and XML), others will be stricter and return a 403 error when you are not being explicit.

This issue can cause confusion when you are testing using various tools. Curl, for example, together with tools such as Postman, will automatically include an 'Accept' header for any MIME type: '*/*' with every request, while BlazeMeter does not add a default Accept header. The net-net could mean that you get different results when testing the same endpoint with different tools, so be prepared to cover all the bases.

API Error #7: That Is Not What I Ordered



So, you thought that you ordered macaroni and cheese but the server places a bowl of nachos in front of you. It can be the same story with APIs. Sometimes an API might return an HTML error page <DOCTYPE! HTML> instead of the JSON you expected, causing your code to self-combust.

Unexpected returns from an API are often due to your not specifying the 'Accept' header with the request and the API not being sure what response format your code is expecting.On the provider side, many frameworks and web servers default to HTML; so if you are creating an API that has no good reason for dishing up HTML, always make sure to check the default error response.

Other complicating factors might have nothing to do with your API, but are the result of the software sitting in front of the API, such as load balancers. If for example, an *nginx* instance encounters a request time-out, it can return an HTML error page by default before your API ever gets the chance to determine what is cooking.

API Error #8: Tea, Anyone?

No waiter worth his or her salt would crack a joke after telling you that the kitchen messed up your order. You would think it would be the same in development, right? Wrong! Back in 1998, a group of developers jokingly proposed the Hyper Text Coffee Pot Control Protocol and error 418 *I'm a teapot* (which is not a recognized standard server error type). Perhaps because of its mega-meme status, many development teams have implemented the *I'm a Teapot* error code in their applications and APIs, often using the code as a catch-all for bugs that are difficult to classify. But, however well intentioned, jokes can fall flat. For example, on May, 2018, a worldwide failure of NPM JavaScript package manager greeted developers with *ERR! 418 I'm a teapot*, when they attempted to install or update a JavaScript Node.jspackage.

Humor is a great thing, but it should not compromise professionalism. Teams should be judicious in their use of error codes. In those cases where more detail is needed, this can be provided with sub-status error codes and messages. Twilio is a great example of a product using the best practice in this regard. Twilio provieds clear sub-status codes, messages, and links to more information.

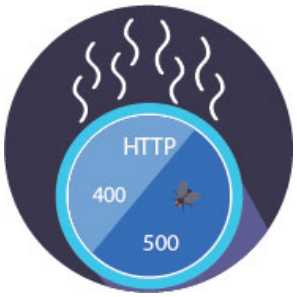
API Error #9: That is Not How I Expected it to Taste

When you order your favorite dish at a new restaurant you can get unexpected tastes and ingredients. Failing to acknowledge the quirks and details of how different APIs and tools behave differently can also cause you to get unexpected results. This is especially true in the case of HTTP redirects. Redirects can be tricky and can also be triggered by small details. As we mentioned before, HTTP API endpoints might redirect to their HTTPS counterparts. Another common occurrence is endpoints with trailing slashes, such as https://yourapihere.com/users/ being redirected to https://yourapihere.com/users or vice-versa. And often, after an API call goes through a redirect, the HTTP method for that request will default to GET, no matter what the original request method was.

This behavior in regard to HTTP redirects can vary between APIs and can also vary between tools. Again, we will use curl to illustrate. With curl, specifying -L to follow HTTP redirects together with an -X option (which is actually superfluous) will result in that method also being used on the redirected-to requests, which might not be what the server asks for at all. As Internet protocols guru and inventor of curl, Daniel Stenberg, points out in his blog post on the unnecessary use of curl-X, dropping the -X makes curl adhere to what the server asks for. If you want to alter what method to use on a redirect, curl has options to cover this: "--post301,--post302" and "--post303."

Staying ahead by reading and keeping up-to-date with documentation is absolutely critical, especially when you are using open source tools that are updated regularly.

API Error #10: Waiter, There is a Fly in My Soup



Problems happen. API data problems can be caused by the following errors:

- Content-length mismatched to actual content length causing connections to be prematurely closed and invalid data returned or parsed.
- Unexpected gzip content because of missing Content-Type in the response header.
- Unstructured response data or different structures from similar endpoints.
- Putting an auth token in the wrong place (header -v- URL -v- payload).

Unfortunately, the above problems and many others can cause developer frustration with lengthy attempts to validate data using suspect coding hacks and poor workarounds.Tools play an important role in resolving data issues, ranging from simple problems to the hair pulling variety. Tools can include simple assertions for JSON and XML, header and response body validation. Tools can also extend to support more complex API use-cases, such as: chained-requests and dynamic data insertion (such as the insertion of IDs and timestamps into requests).

Conclusion

The ten mistakes and errors outlined in this paper do not represent a definitive list. However, based on our experience working with many customers, they are common enough to cause thousands of hours in wasted effort, resulting in low productivity, missed deadlines, and lost opportunity. And, whether you are consuming or providing API goodness, these *gotchas* can easily go unnoticed, causing immense pain to your co-workers, partners, and customers.

Bad redirects, unexpected error codes, and invalid content types can quickly derail efforts, which is why you need comprehensive API monitoring to quickly and accurately pinpoint issues from internal and third-party APIs.But nothing is ever static. Just as great chefs continuously taste their culinary masterpieces, a great testing and monitoring approach operates on a continuous schedule; so that you can identify, prevent, and solve simple and complex problems fast.

For more information about BlazeMeter Continuous Testing, go to blazemeter.com.