TIBCO® APIFORTRESS

# Success Guide for a Healthy API Program

Are your APIs performing optimally, and as they were designed? Do reliability and scalability concerns keep you up at night? Are you confident you can rapidly recover from a catastrophic data failure? Just about every customer we talk to has these concerns. The good news is that no matter where your API program is today, the right tools can support your program's health and empower you to remain a leader in your industry.

Tools like TIBCO Cloud™ Mashery® for cloud-native API management, and API Fortress for continuous testing, help large and small enterprises evolve toward a healthy and scalable API program.

The partnership between TIBCO and API Fortress expedites innovation throughout the full lifecycle of APIs. We have collaborated for a seamless experience for creating, testing, productizing, securing, and analyzing APIs, and also for defining the keys areas for a healthy API program. We hope you benefit from this collaboration, and take your API program to the next level.

## TIBCO CLOUD™ MASHERY® + API FORTRESS = HIGH-PERFORMING APIS DELIVERED WITH SPEED AND QUALITY

Review the material in this lookbook to learn more:

- Key Problems and Solutions for Healthy API Programs
- The Healthy API Lifecycle: A Checklist
- Learn from these API Management and Monitoring Mistakes

## 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 www.apifortress.com.

## ABOUT TIBCO

TIBCO fuels digital business by enabling better decisions and faster, smarter actions through the TIBCO Connected Intelligence Cloud. From APIs and systems to devices and people, we interconnect everything, capture data in real time wherever it is, and augment the intelligence of your business through analytical insights. Thousands of customers around the globe rely on us to build compelling experiences, energize operations, and propel innovation. Learn how TIBCO makes digital smarter at www.tibco.com.

TIBCO®

# Key Problems and Solutions for Healthy API Programs

Although it comes with a deep cost to the enterprise, the fast pace of change in technology makes for exciting and revolutionary changes in business operations and growth. One of the downsides of charging forward with an "innovate faster" ethos is that some legacy investments become difficult to update, which reduces their reliability.

Well-designed and well-managed APIs are a solution to this and other challenges, but become a problem when implemented without proper planning and process. As microservice architectures begin to reach maturity, organizations are finding that some systems are becoming stagnant, adding to the engineering organization's technical debt.

It's now a best practice to approach any new development with an API-first mindset to preserve or improve the agility and reusability of systems and information—and testing and monitoring all deployed code becomes critical to long-term success.

## EVOLUTION OF APIS

Though test-driven development and test-first programming are recommended, most organizations adhere closely to a "build first, test later" mentality. Developers dislike writing additional code to test the code they're about to write, and many engineering managers hesitate to assign sprints to generate test code when they're being evaluated on how quickly they can deliver new features. Many existing API programs are stuck with legacy code that has little test coverage because of this.

## EARLY STEPS

In the early days, many API programs were created for a single purpose: to serve data to apps installed on mobile devices. Though this code was intended for production, it was often treated as a proof of concept with less adherence to strict QA, documentation, and traditional development standards.

## REALIZING BUSINESS VALUE

As APIs proved their utility beyond initial purposes, organizations expanded into full API programs to take advantage of the open developer ecosystem, leveraging shared data and functionality for strategic partnerships.

## ADOPTING API FIRST

An API-first strategy is the logical next step for any mature API program. As staff engineers begin to understand the value of consuming their own APIs and abstracting proprietary vendor systems from their code, refactoring existing code and increasing test coverage becomes as important as developing new functionality and products.

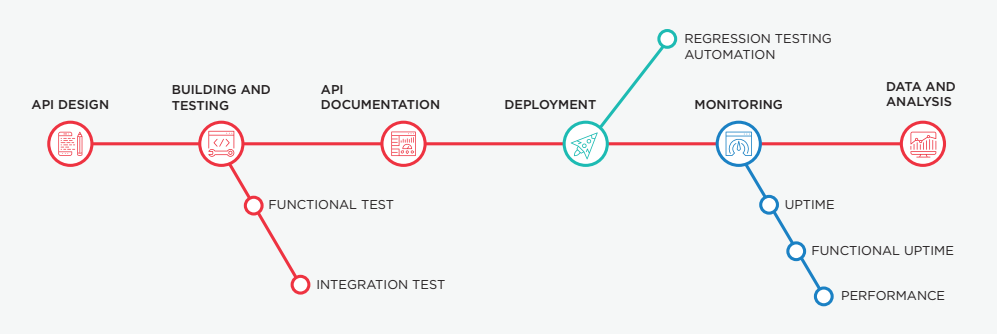
## ADDRESS PROBLEMS NOW

Regardless of where your organization is in its lifecycle, you should implement and adhere to the best practices of building to a pre-defined spec, ensuring maximum test coverage of all developed code, and monitoring your APIs on both the front and back end.

In the other whitepapers in this lookbook, we dive into the details and offer suggestions you can implement immediately to bring your API program up to speed. The diagram provides an overview of what we'll be covering.

Tools like TIBCO Mashery® for API management, and API Fortress for testing and monitoring, help large global organizations burdened with tons of antiquated backend technology to migrate their legacy processes to efficient APIs. This method expedites innovation with little additional engineering effort, opening new opportunities to monetize data.

## THE HEALTHY API LIFECYCLE




This lifecycle is cyclical

TIBCO®

**Global Headquarters**
3307 Hillview Avenue
Palo Alto, CA 94304
+1 650-846-1000 TEL
+1 800-420-8450
+1 650-846-1005 FAX
www.tibco.com

TIBCO fuels digital business by enabling better decisions and faster, smarter actions through the TIBCO Connected Intelligence Cloud. From APIs and systems to devices and people, we interconnect everything, capture data in real time wherever it is, and augment the intelligence of your business through analytical insights. Thousands of customers around the globe rely on us to build compelling experiences, energize operations, and propel innovation. Learn how TIBCO makes digital smarter at www.tibco.com.



04/11/19

TIBCO®

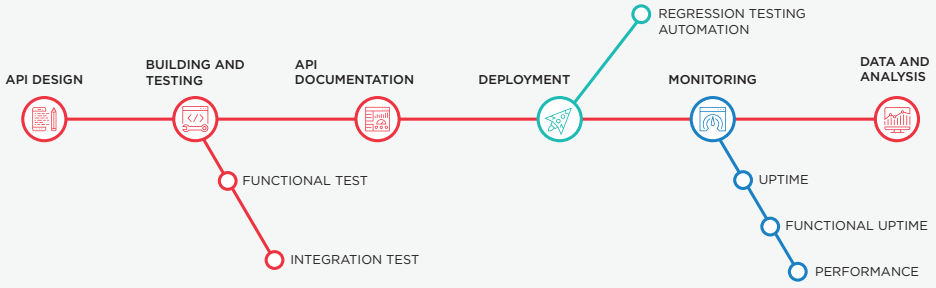
# The Healthy API Lifecycle: A Checklist

Though APIs have been around for a while, developers still treat them like they're something special and unique. In truth, there's not much difference between serving APIs and building an application with a graphical user interface. Each should be designed with the end user in mind, fully tested on both the backend and frontend before release, and continually monitored to ensure they are performing within expected limits.

One key difference specific to APIs is the availability of mature solutions to make all of these operations easier. API management tools like TIBCO Cloud™ Mashery® provide the infrastructure that can help you scale, secure, and support your API program, while API-specific testing and monitoring tools, such as those provided by TIBCO partner API Fortress, can help you adhere to best practices, building services that are performant and secure.

Even with good API management, testing, and monitoring, IT leaders express anxiety over the stability of their APIs. The interconnected nature of APIs can lead to the appearance of complexity that cause operational bottlenecks or potential security threats. However, the worst offenses are often due to poor testing, inefficient coding, and a general lack of knowledge of how APIs operate. Understanding the healthy API lifecycle and treating it as a checklist can help address the biggest of issues and should calm all but the most irrational fears.

## THE HEALTHY API LIFECYCLE




### API DESIGN CHECKLIST

- Identify key resources.
- Address limitations.
- Define system expectations (through user stories).
- Ensure exposed data can be contained for providing to developers.
- Appropriately limit data exposure.
- Use model and mock-up tools to flesh out resources, relationships, and actions.
- Collaborate and confirm the design.

## API DESIGN

You should put the same thought, care, and effort into designing your APIs as you do for any of your websites or visual interfaces. A well-designed API will not only best serve its primary function, it will reduce development time and costs, improve performance on both the backend and frontend, improve reusability, and leave you with a more robust platform with plenty of room for growth as the business changes.

Many teams start the API design discussion debating different protocols, such as REST, SOAP, or GraphQL. The best APIs are designed to serve the needs of the business by identifying key resources and data parameters required to solve the problems at hand. These will determine your protocol selection.

Approaching API design with this mindset can set you up to identify potential security and performance issues. If the resources you need are not easily accessible or are tightly coupled behind the scenes, you can begin to address these limitations before writing a line of code. One approach is to start with user stories that identify the actor, the action being taken, and the purpose of the action. One example of a user story for an e-commerce application might be:

"As a customer, I would like to search for a product by category so it's easier to find what I am looking for."

You may wind up with dozens — potentially hundreds — of stories, but they are tremendously valuable when defining your system expectations. User stories uncover the resources, actions, and relationships that will ultimately make up your API system. In the story above, we can see that the nouns — "customers," "products," and "categories" — are all resources we'd expect to find in this system. The verb "search" indicates an action that should be implemented on these resources. The phrase, "to search for a product by category" also indicates a relationship between a product and a category resource. In this case, those products belong to a category.

It's important to pay attention to the audiences who are going to use these APIs and ensure the exposed data can be contained to provide it to developers. A common security issue with APIs is exposing more data than needed in your resources, potentially leading to unintended data leaks when you widen your audience. A good rule of thumb is to assume your APIs will be exposed to third-party developers, so limit exposure of sensitive data to specific endpoints.

As you develop more use cases, the full nature of each resource, its relationships, and the actions you can take will begin to guide your API design and protocol choice.

Strong API management systems can help you decide how best to build these APIs by providing modeling and mock-up tools. By using existing specification formats such as Swagger/OpenAPI, you can use visual editors to rapidly design standardized APIs for your organization according to the resources, actions, and relationships. They also allow you to collaborate with your developers to confirm that your design is headed in the right direction. Once you've agreed on the general design of your APIs and how they might work once launched, you can start building.

### API BUILD CHECKLIST

- Write test code and continually test code changes to confirm that service transformations and new services are working as expected.
- Use pure functions whenever possible.
- Ensure autonomous pieces work in harmony. Use API Fortress to write intermediate integration tests to check for semi-complete artifacts and address them.
- Use a load testing tool at the end of specific sprints to identify weaknesses in request and response services.
- Document your API using a standardized format, such as the OpenAPI Spec (Swagger), to clearly define request and response data, which forms the basis of your automated integration test plan.

## BUILDING

There are several ways to build your APIs: transforming existing services, creating new services from scratch, using low-code API creation tools, etc. With all of these methods, you need to make sure they are working as expected, continuously testing them when changes are made. Testing and test code is an invaluable part of software development, crucial to a healthy API program. That being said, it should be done at this stage of the lifecycle.

Developers can guarantee that a function they write performs as expected. Such functions should be determinate, always producing the same output given the same input parameters and state of the application. A function that interacts primarily with the state of the system may be hard or impossible to test, while a function that receives an input, interpolates it, and produces an output (a.k.a. pure functions) is easier to test. Whether you're a functional developer or not, leverage pure functions wherever you can. This not only guarantees that the code is working but will verify that code changes do not modify behavior unexpectedly.

Unit tests can ensure internal consistency in standalone code but do little to ensure the integrity of distributed, interconnected services.

Unit tests can ensure internal consistency in standalone code but do little to ensure the integrity of distributed, interconnected services, such as those found in a microservices architecture. In such an environment, the risk for a "perfectly functioning" piece of software that passes its own internal tests but still disrupts the entire system is high. It's common for a single API call to trigger a cascade of multiple sub-calls to other dependent services in the system. Write intermediate integration tests with platforms like API Fortress to work as checkpoints for semi-complete artifacts. Validating the data returned from these dependent services is vital to ensure autonomous pieces of the flow work in harmony.

Identifying these interdependencies can be tricky, but following the design process outlined above can identify them early and highlight any potential risks. Mocking and modeling these calls can also help bring up potential issues, allowing your team to address them before they happen. There are open source mocking tools available, along with the features available in TIBCO and API Fortress products.

Validating request and response data is vital, but many errors don't make themselves known until the system is under more intensive use. These errors may show themselves through race conditions that produce strange behavior or may emerge as performance issues. Programmatic functional load testing at the end of specific sprints may help identify these weaknesses before your services are deployed to production. The API Fortress platform includes a load testing tool, which uses existing functional tests to perform this work. This helps make sure your APIs work under stress, capturing unexpected problems such as memory leaks.

The structured, deterministic nature of APIs is a boon for your testing plan once your services are in production. Documenting your APIs using a standardized format such as the OpenAPI Spec (a.k.a "Swagger") will clearly define the expected data that both go into your API as a request and returns as a response. These definitions not only document your APIs in a flexible way that is friendly to automation tools, they should form the basis of your automated integration testing plan and can be supplied to your continuous integration systems and tests they should enforce. Platforms like API Fortress can automate test generation from these spec files, saving you time and money.

### API DOCUMENTATION CHECKLIST

- Write or automatically generate functional documentation on the API's expected behaviors, including every endpoint and piece of data it supports.
- Produce static documentation covering common use cases, anticipated support questions, and a narrative to walk developers through your API program's capabilities.
- Verify that your documentation and actual API behavior/program are in sync, and if not, fix any discrepancies.
- Prepare developers to take advantage of modifications you make to the API by communicating about updates as soon as they are deployed.
- Consider including a "deprecated" or "update" field in your API response payloads to alert developers of a change.

## API DOCUMENTATION

The value of good API documentation cannot be overstated. While good resource naming conventions and data response design can go a long way in improving the usability of your APIs, well-formatted, searchable documentation, and consistent communication can eliminate support issues before they arise. More importantly, good documentation explains how your APIs are expected to act, which can help both your developers and your code testers quickly identify and address any issues they may find.

Good documentation starts with the developers creating your APIs. The best strategy is often to start by documenting your APIs' expected behaviors and build to those specs. The mock and modeling tools discussed in the API Design section often produce functional documentation written using one of the more popular documentation formats, such as the OpenAPI Initiative's Specification format (OAS), formerly Swagger. While there are many formats to choose from, such as the Blueprint spec from Apiary or the RESTful API Markup Language (RAML), the OAS has seen wider adoption across the industry and is the most recommended.

This form of documentation describes every endpoint and every piece of data your APIs can support on a request and return on a response, allowing it to become the single source of truth to provide guidance to your client developers, configure API management and testing tools, and even generate client code and SDKs to speed development time. Tools like Stoplight and SwaggerUI, as well as those embedded in API management systems like TIBCO Cloud Mashery, can turn your OAS definitions into browser-based interactive documentation, allowing developers to experiment with your API without writing a line of code.

While your functional documentation should be the primary source for information about how your APIs are designed, your static documentation should provide information about how they are intended to be used. Rather than acting as a data dictionary and laying out each endpoint and parameter used by your API — a task better suited for your functional documentation — your static documents should focus on covering common use cases, addressing anticipated support questions, and providing a narrative to walk developers through the capabilities represented by your API program. These often take the form of tutorials illustrated using sample code from the languages your developer audience is most likely to use.

Both your functional and static documentation is tremendously valuable as you test your APIs, as they should document expected behaviors. Any behavior that strays from what you have advertised must either be fixed or properly documented as soon as possible to ensure your developer audience always has the most recent information available.

One frequently overlooked aspect of documentation is active and consistent communication with your developer audience. As your API program matures, you will likely add more endpoints and possibly alter existing APIs to make them better suited to the needs of your audience. You should update all of your documentation to reflect these changes as prior to deploying APIs are deployed. It's crucial to inform developers of these changes as early as possible to help them prepare and let them take advantage of your modifications.

The communication strategy you adopt will depend largely on how your target audience prefers to receive communications. The best plans are those that span several channels at once. Use social media outlets such as Twitter and tools like email to drive developers back to a blog or documentation page that details what has changed and how long older versions will live until they are deprecated.

One of the more effective and interesting means of communication lies directly in your APIs themselves. You may consider designing your response payloads to include an optional response section with a field name of "deprecated" or "update." The data in this field should include a short message describing the change made to the endpoint with a link to a static blog entry or web page that details exactly what has changed and how the developer should respond.

The major benefit of this approach is that client developer teams can choose precisely how they would like to handle these updates as they see them. As developers change jobs, email addresses and social media followers may not be aware of who to reach in light of the changes. Embedding it in the API itself — and clearly documenting how it is expected to work in your static documentation — allows developers to create code that reacts immediately when an update is seen by forwarding the information to an internal mailing list, adding a message to an internal chat channel, filing a ticket in their support system, or any other way they see fit.

### API CHECKLIST

- Write or automatically generate functional and integration tests.
- Write detailed tests that also validate the intended logic (business logic) of the API program.
- Run tests automatically as part of a CI/CD pipeline.
- Schedule tests to run as functional uptime monitors.

## TESTING

Verifying that your code and systems work as expected is not a one-off activity. In massively distributed architectures where parts are maintained and deployed by several teams — in some cases, not even part of the same organization — continuous testing at every stage becomes critical to the success and performance of your API program. These organizations put CI/CD pipelines in place to automate this process, using platforms such as Jenkins, Bamboo, and Travis CI. These tests can be broken down into two types: functional testing and integration testing (also known as end-to-end tests).

Functional testing is meant to ensure that your API call responds as expected, validating everything from the header, to the resources, to the data associated with each resource. Providing your OAS definitions to an automated testing platform reduces much of the load, freeing you to focus on the creative aspects of test development and business logic validation.

Consider as an example an API for e-commerce clothing. A general search may return a resource representing many items across multiple categories, such as shirts and shoes, each item with a parameter for size. For shirts, you'd might expect this parameter to include values small, medium, large, etc. For shoes, you might expect a numeric size cast as a string. Your OAS definition only shows that the system should return a string value, but additional tests should also confirm the actual returned values match expectations.

Integration tests allow you to validate the workflows and interdependencies of your APIs. Most applications will make subsequent calls to your system, using the data from one return endpoint to set the request parameters in another. Using our e-commerce example again, a customer might perform a search based on some criteria that returns multiple products, then select one of those products to get more details. If they choose to purchase the product, they may add it to their cart and, finally, complete the checkout by providing shipping and payment details. Where functional testing ensures that each of these calls conforms to expected behavior, integration testing helps ensure the complete flow of one call to the next also works as expected.

Your developers can write these integration tests as part of their development workflow, or you can use a platform like API Fortress to make it easy for developers or testers to create them. Once deployed, these tests should be handed off to automated testing tools that run them at least each time a new piece of code is deployed to the system. This ensures that the whole system is always performing as expected and there are no regressions.

### API DEPLOYMENT CHECKLIST

- Run a full series of regression tests each time a new service is deployed to production.

## DEPLOYMENT

Whether you deploy at scheduled intervals or run a true continuous deployment program, it's critical to run a full series of regression tests each time you move new services to production.

Whether you deploy at scheduled intervals, or run a true continuous deployment program, it's critical to run a full series of regression tests each time you move new services to production. As more code is deployed, the number of tests can quickly add up and build on each other. It's important to update the test code that no longer reflects current functionality and remove any tests designed for obsolete functionality.

### API MONITORING CHECKLIST

- Ensure the API manager is providing an extra layer of security by monitoring for unusual activity spikes and other events.
- Consider using visualized analytics such as provided by Spotfire® to help understand unusual activity and what might be causing it.
- Be sure to also monitor for functional uptime to ensure the API is performing within expected parameters.

## MONITORING

No API program in active use can be considered static, and there's no such thing as perfect test coverage. The performance of your APIs is determined by the code itself, expected usage on the systems supporting it, and the actions of your users—situations well beyond your anticipation or control.

API managers can provide for much of this monitoring by noting unusual spikes in activity, an increase in certain HTTP response errors, and/or strange usage patterns from specific users or applications. Monitoring for these activities provides an extra layer of security when a bad actor or bad code gets into your API. By establishing baselines early on, you can set thresholds that point to unusual behavior.

While a good API management system can identify such unusual activity, it often requires human intervention to determine whether the behavior is bad or not. Any kind of monitoring needs to be followed by analysis, which means getting all of the data into a format that's easy to understand and parse. The data provided by TIBCO Cloud Mashery, for example, can be streamed directly into a TIBCO Spotfire® instance, allowing you to create visualizations that help you best understand what the data represents.

Another critical aspect to monitor is functional uptime. Beyond merely checking whether the API is responding, functional uptime also helps ensure the data provided in those responses matches expectations. This is where your choice of API testing platform is key. A tool that only shows you that your API has been up 100 percent of the time does nothing to tell you whether it's actually performing within expected parameters. Your testing platform should be able to schedule existing integration and regression tests against your live API to ensure it remains functionally accurate and consistent. Leading platforms like API Fortress also provide a status page that can be viewed by all stakeholders — internal and external — to help rule out potential issues as it tests your APIs calls.

### API DATA AND ANALYSIS CHECKLIST

- To manage and grow your program, regularly review API program stats collected by your API management system: calls made, errors found, detailed developer and app usage.
- Pay attention to developer engagement, identifying which applications are making the most calls and which show little to no activity. It may be helpful to integrate that data with information from other systems to determine KPIs for your business as well as for your users.

## DATA AND ANALYSIS

Continuous automated testing can help you identify potential problems as they happen, but a regular manual review of your API data provides numerous benefits as you manage and grow your program. The information regarding the calls made, errors found, and detailed usage of your APIs by developers and applications give insight into how well your services are helping your client developers perform their tasks.

For example, you may find some applications making regular successive calls to a single endpoint to get the same data. This may indicate that applying a caching layer or perhaps modifying the call could ensure that the data they receive is always the most up to date. You may find several patterns where multiple calls are made to perform a single task, which could indicate an opportunity to optimize and potentially reduce that number.

It's also important to pay attention to developer engagement, identifying which applications are making the most calls and which show little to no activity. You may want to form tighter relationships with your best users and ask them for regular feedback on your system to identify feature enhancements that will help your API usage grow among a similar cohort of applications. You may also consider reaching out to those who are not using your APIs as often as expected to find out whether any changes are necessary to make their tasks easier to perform.

Your API management system can help you collect most of this information, but it may be helpful to integrate that data with information from other systems to indicate KPIs from your APIs. The business KPIs you track will often differ from the performance KPIs, but the sources for the data driving them may be the same. Get with your business team early to identify their KPIs and build them into your analysis to ensure you're not only driving value for your customers, but also for your business.

## SUMMARY

When it comes to monitoring the health and performance of your API program, there's no single solution. A combination of a powerful API management system, such as TIBCO Cloud Mashery, integrated with a full-featured API testing platform, such as API Fortress, can help ensure your API program remains robust and performant, no matter how you design, build, and deploy your services.

API Fortress's integration with TIBCO Cloud Mashery complements and improves the efficiencies of both platforms. Login to your existing Mashery account from within API Fortress to generate regression tests in seconds and start rapidly innovating with confidence.

TIBCO®

**Global Headquarters**
**3307 Hillview Avenue**
**Palo Alto, CA 94304**
**+1 650-846-1000 TEL**
**+1 800-420-8450**
**+1 650-846-1005 FAX**
**www.tibco.com**



04/11/19

TIBCO®

# Learn from these API Management and Monitoring Mistakes

By now, you should have a clear understanding of just how critical good API management, testing, and monitoring is to the health of your API program. While we've gone into great detail to explain the hows and whys of applying these processes to your API lifecycle, we find that real-world examples often provide the best explanation of what can go wrong and how you can avoid similar pitfalls. The TIBCO Cloud™ Mashery® and API Fortress teams have more than 10 years worth of such stories. We're going to share a few of them in the hopes of keeping you from making the same mistakes.

## LIFE WITHOUT API MANAGEMENT OR MONITORING

*"One of your API users just brought down our entire system!"*

Let's start with an example of a successful API program that did not have proper API management or monitoring applied. In a matter of minutes, the operations team for a marketing company noticed an unusual amount of traffic causing severe growth and performance issues. At first, they believed they were the target of a coordinated denial of service attack, but the data payloads appeared to be valid.

After some digging, they discovered that the source of the issues was a single API call to add email recipients to a list one at a time. The system was designed to provide both individual email adds as well as batch list uploads. The developer documentation clearly laid out when to use each one; if you were only adding less than 10 contacts to your list, you should use the individual add call, otherwise, you use the batch call.

The ops team reached out to the engineering team responsible for building the APIs. They then reached out to the person responsible for managing the API program to find out exactly which client application was the culprit. Without an API management system, there was no easy way to find out where the calls were originating from, and every minute the performance of the entire web application was degrading, affecting every single customer.

It took more than an hour to dig through logs and database entries to find the account responsible, then additional time to track that account to the developer responsible for the application. Upon speaking with the developer, the team learned he had built a simple email newsletter registration form that had gone massively and unexpectedly viral. Each time someone signed up, his code added the single contact to an existing list, which was exactly how the API documentation indicated it should be done. What no one expected was that this code was making this call dozens of times per second, highlighting the inefficiencies not only in the API, but in the code supporting it. The team immediately blocked the account from making further calls by updating a row in the user database by hand and working with the developer to rewrite his code to use the batch method instead.

With proper API monitoring, the operations team would have found the performance degradation much faster because the spike in unexpected calls on that endpoint would have triggered alerts to the right people. API management could have helped the company find the culprit in a matter of minutes, limiting the impact on the system. This would have been as simple as reducing the number of calls they could make per second rather than shutting them off completely. Proper integration and performance testing during the API's development and deployment could have identified these issues well before the code was in production.

Needless to say, the company immediately sought an API management solution that would help them avoid this in the future, and adopted best practices in their testing and monitoring across the entire API lifecycle.

## AN EXPENSIVE NULL

*"What do you mean there are 3,000 items missing from our website?!?"*

When we first on-boarded a particular retailer's API, we created tests based on their documentation. The product_listing endpoint, which contained all available products at that moment, had more than 3,000 errors. The retailer had never had an issue with that endpoint before, and manual testing indicated that it seemed to work fine.

When we reviewed the error report, the source of the problem became immediately clear. Each product had an object called category, which should have been one of 17 options including "men," "women," "babies," etc. In their listing of about 50,000 products, more than 3,000 items returned "null" for the category.

This was a critical miss because the category is how their main API consumer, their website, organized the products. Those null values meant that more than 3,000 products were not showing up in customer searches, leading to potentially thousands of dollars in lost sales. They designed tests to make sure the API was responding with the correct data, but had not validated that the information returned met expectations.

After further investigation, they discovered the web team that was responsible for the tool that allowed vendors to post their products for sale had not run a validation test to ensure the category field was populated before sending it to the API. Further, the API team failed to add the requirement that the category field be populated by client applications.

Good integration testing could have caught this before the code was deployed. Applying those same tests using the platform running functional uptime monitoring on the production environment would also have caught the problem if the flaw had been introduced in a subsequent deployment.

This is a common problem we see in large organizations; cross-team collaboration is a worthy goal, but often fails to catch errors. Automated testing can pick up the slack and help ensure everything works as advertised.

## 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 6:00 am from a book publisher. When we looked at our reports, it showed the customer's product details endpoint was frequently returning a 404 "resource not found" error. Manual testing showed no issues at all.

A look at the testing reports once again quickly identified the problem: only the integration tests were failing. These tests were written to follow the common flows that their API partners would experience. In this test, we first hit their ISBN listing endpoint, which gave us all active product IDs. Using those results, the platform dove into hundreds of products individually. The integration tests showed the same 404 errors customers were reporting.

So why were so many items missing for customers using the API outside the API manager when all internal manual tests were fine? Good API management gives you the ability to cache endpoints every few hours and substantially increase performance for your customers and partners. When a major update is made to a database, the API cache should be reset in order to use that new data. The team performed a database refresh biweekly but never updated the API cache. As a result, the problem had been building for months.

API management systems are incredibly powerful, but some features need to be planned for carefully, and it can be easy to miss some things. A proper API testing strategy can help you uncover these potential pitfalls early and give you continued confidence that your APIs are responding quickly and accurately.

## SUMMARY

We have countless tales of APIs gone wrong, and the vast majority of problems are not due to bad development teams but simple human error. Good API testing, management, and monitoring can't fix every potential problem you encounter, but a good strategy can increase your chances of finding and fixing problems before your customers do.

### Give TIBCO Cloud Mashery a try at cloud.tibco.com.

Learn more about API Fortress at **https://apifortress.com**, or **sign up for a free trial**.

TIBCO®

**Global Headquarters**
**3307 Hillview Avenue**
**Palo Alto, CA 94304**
**+1 650-846-1000 TEL**
**+1 800-420-8450**
**+1 650-846-1005 FAX**
**www.tibco.com**