

Developer Experience (DX) is Key to a Successful API

A little while ago, I was asked to participate in a project that involved [comparing the APIs of leading electronic signature \(e-signature\) platforms](#). My role was to perform basic API testing using the [Postman REST Client](#) and to provide a summary of my experience using the APIs. While I'm experienced in using and integrating APIs such as [Twitter](#), [YouTube](#), [Google Charts](#), [Google Maps](#), [Open States](#), and others into websites and web applications, I had never before used the e-signature APIs that were to be tested and summarized for the project.

There were eight APIs that needed to be tested and reviewed. However, there was an immediate problem, two of those eight APIs were [SOAP](#). In fact, almost all of the e-signature APIs in the group were SOAP and only a few of the platforms provided a RESTful version of their API. I'm not experienced using SOAP and Postman is designed specifically for testing REST APIs / HTTP requests, so it was necessary to find a developer experienced in SOAP who would be able to test the two SOAP APIs for the project.

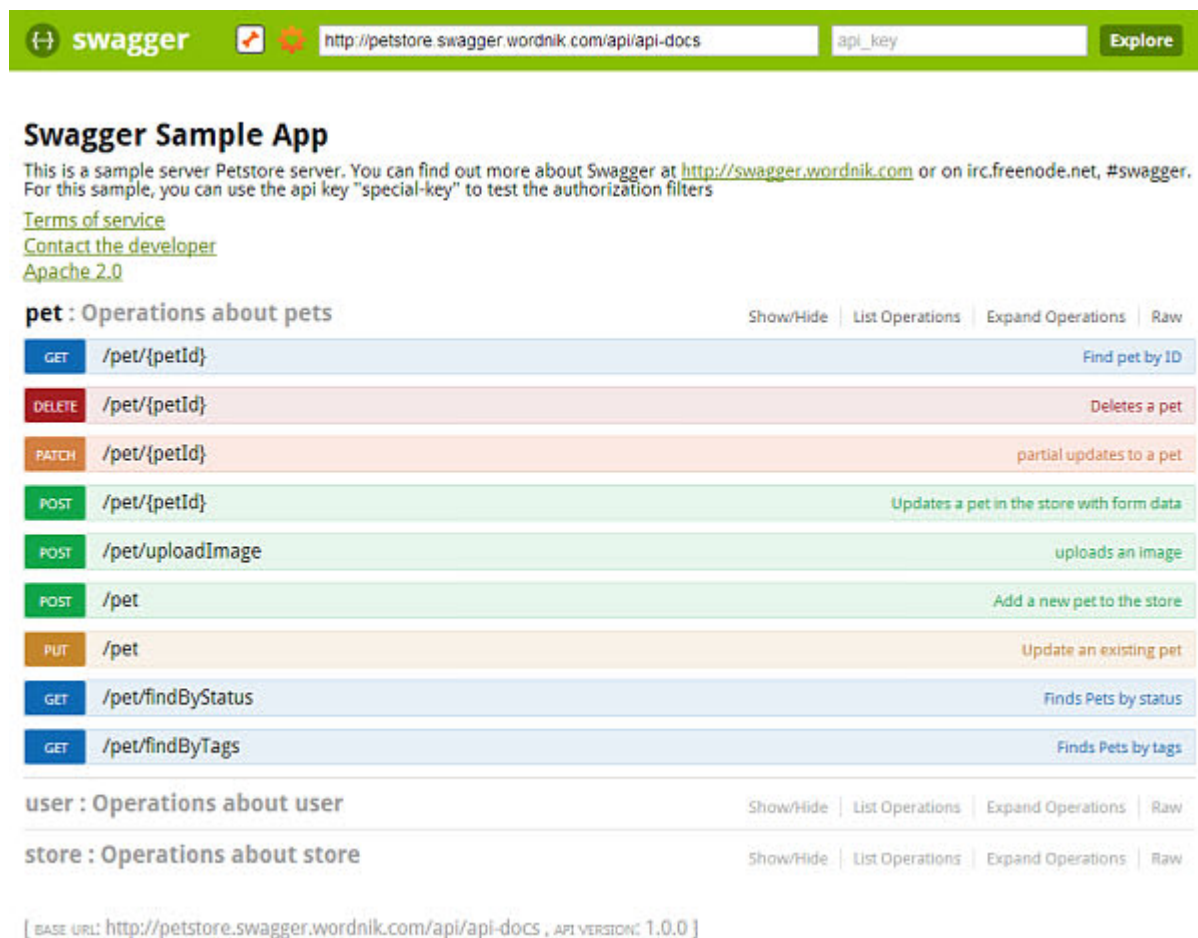
As I began reviewing the websites and documentation for the six remaining APIs, I found many obstacles—and in a few cases, complete roadblocks—to testing these APIs. In the end, I was only able to successfully make API calls with two of the six e-signature APIs.

For nearly all of the e-signature platforms, the three biggest stumbling blocks to using the APIs were:

- **Bad documentation** - Incomplete, inaccurate, very confusing and hard to follow, lack of interactive documentation, documentation in PDFs, etc.
- **No SDKs / code samples in my language** - SDKs not available for the most commonly used programming languages such as JavaScript, PHP, Python, Ruby, Java, Objective-C, C# etc.
- **Only SDKs available** - Could not make calls using the API directly.

Other problems in getting started and using some of these APIs included lack of a free trial or freemium developer account, API key not instantly accessible, overly complex SDKs (one required a database), and installation of additional frameworks needed. While I ran into a multitude of problems trying to use these e-signature APIs, the issues regarding documentation and SDKs were the ones that caused the most headaches. I spent many hours searching in [StackOverflow](#), API groups, developer forums and Googling, trying to troubleshoot API problems and trying to figure out how to get the APIs to work.

All in all, my developer experience (DX) trying to set up and use these e-signature APIs was far from great



The screenshot displays the Swagger UI for a sample Petstore API. At the top, there's a green header with the Swagger logo, a search bar containing 'http://petstore.swagger.wordnik.com/api/api-docs', an 'api_key' input field, and an 'Explore' button. Below the header, the title 'Swagger Sample App' is followed by a descriptive paragraph and links for 'Terms of service', 'Contact the developer', and 'Apache 2.0'. The main content is organized into sections: 'pet : Operations about pets', 'user : Operations about user', and 'store : Operations about store'. Each section lists various HTTP methods (GET, DELETE, PATCH, POST, PUT) and their corresponding endpoints, such as '/pet/{petId}', '/pet/uploadImage', and '/pet'. Each endpoint is color-coded and includes a brief description of its function. At the bottom, a status bar indicates the base URL and API version: '[BASE URL: http://petstore.swagger.wordnik.com/api/api-docs , API VERSION: 1.0.0]'.

Swagger makes it possible to easily create nicely designed, interactive API documentation - Image Credit: [Reverb / Swagger](#)

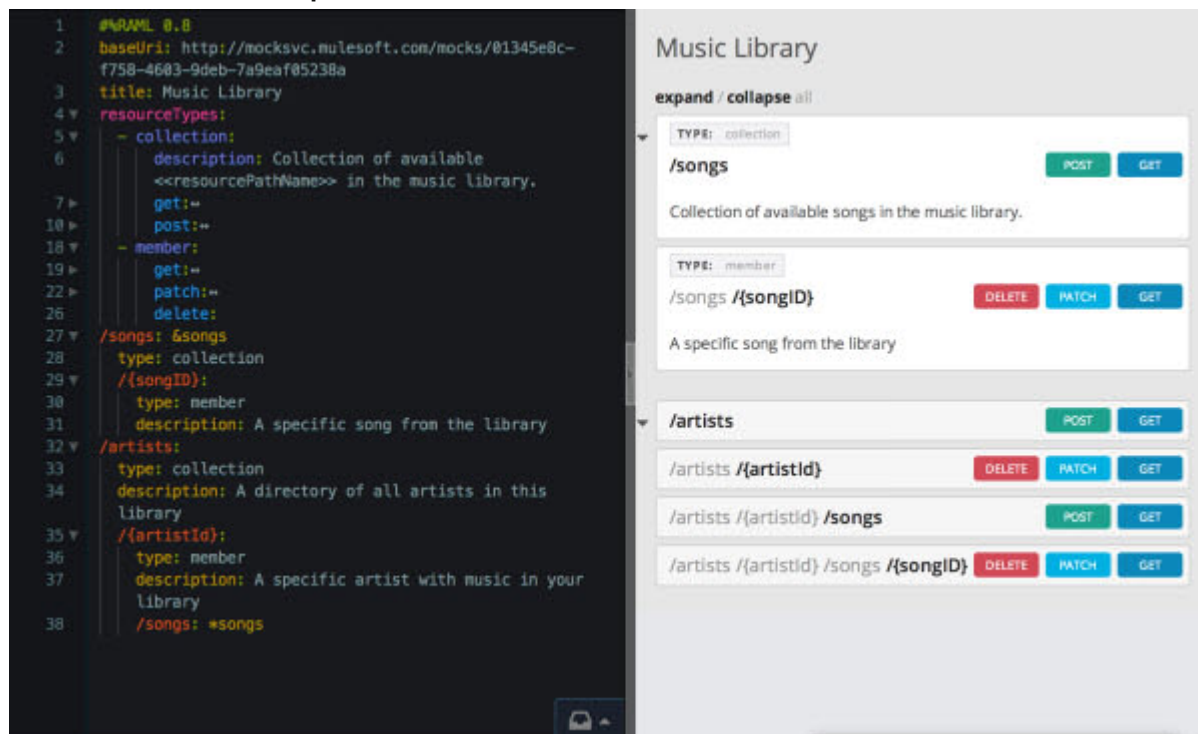
Documentation

Great documentation is the most important factor when it comes to ensuring a good developer experience with an API. Documentation must be complete and up to date, easy to read and follow, and include code samples for commonly used programming languages such as JavaScript, PHP, Python, Ruby, Java, Objective-C, C#, etc.

Documentation should also provide a detailed explanation of what the API does, list and describe all endpoints for the API, and show how to cURL the endpoint when appropriate. Easy-to-use, interactive documentation should

also be available for developers to not only learn about the API, but try out the API as well.

There are many tools available, such as [Swagger](#), [RAML](#), and [Apigility](#), for creating nicely designed, interactive documentation. *ProgrammableWeb* published an [article](#) by Romin Irani that covers great API documentation. Additionally, many API management vendors including [Mashery](#), [3scale](#), [MuleSoft](#), [Apigee](#) and [WSO2](#) among others offer API providers interactive documentation out of the box.



RAML helps you design your API, and auto-generates API documentation with an intuitive, interactive API Console - Image Credit: [RAML.org](#)

SDK vs. API

In recent years, API providers have been shifting to SDKs as the sole means of API consumption as opposed to allowing for direct calls to the API. Holger Reinhardt, senior principal of business unit strategy at CA Technologies and director of emerging technologies at Apiacademy.co,

aptly describes the new meaning of API SDK in a recent [blog post](#) where he writes:

There used to be a time, not so long ago, when "SDK" meant documentation, code samples, build scripts and libraries all bundled together. Today, you find all the former on the Web and usually only the library remains. When I use the term SDK, I mean a programmatic API (e.g. JS, Ruby, PHP, Python) on top of the Web API (usually REST/JSON).

For many developers, it is much easier to consume an API by making direct calls than having to install and use an SDK. In addition, in many cases, an SDK is simply overkill. What if the developer wants to use an API to create a simple, one-page, client-side web app? Having to use an SDK means the inclusion of unused functionality and many lines of unnecessary code. Many SDKs also require the installation of additional frameworks, causing the developer to spend significantly more time setting up and using the API SDK.

SDKs are also problematic in that there are many different programming languages available today. If an SDK is not available in the developer's preferred programming language, the API SDK becomes unusable. However, if calls can be made using the API directly, then the choice of programming language becomes moot.

Conclusion

Whether an API is designed to be integrated with complex enterprise systems or simple third-party apps, it is the developer experience (DX) that determines whether the API will be widely adopted or shunned by the developer community. If developers have a bad experience with an API,

they will not want to use it again and will look for a better, easier API to use (aka the competition). If developers have a good experience with an API, then they are more likely to recommend the API to other developers, and may even become evangelists for that API.

The bottom line: The success of an API depends on the support of the developer community. Make sure your API is easy to use and the barriers to entry are as low as possible.