Three Ways to Better Understand Your APIs

Janet Wagner — Read time: 5 minutes

Designing and building high-quality APIs that developers like to use is not an easy task.

You need to design APIs that solve real-world problems. You must work with a wide range of stakeholders, many of whom have a vested interest in the success of the APIs you build. And you need to make sure every API provides a great developer experience (DX).

All of these things require that you have a deep understanding of every API you build.

So today we're highlighting three things you can do to gain a better understanding of your APIs.

1) Talk About API Architectural Styles

We talk about API specification standards like OpenAPI quite a bit on our blog. But before you choose an API specification standard for your API, you need to figure out which API architectural style to use, and each one comes with a set of constraints. Discussing the pros and cons of different API styles will help you understand how your APIs should work for different use cases. Among the API styles used today are:

- **RPC –** With a Remote Procedure Call (RPC) style, you can use almost the same code whether the procedure is executed in a local environment or an external one. This type of API style is often used in distributed systems. JSON-RPC and XML-RPC are the most used RPC styles today.
- **gRPC –** This is an RPC style that uses the HTTP/2 protocol for transport, and its features include bi-directional streaming and pluggable authentication. gRPC is open source and was initially developed by Google. Today it is an incubating project at the Cloud Native Computing Foundation.
- **SOAP –** Many developers consider Simple Object Access Protocol (SOAP) to be an API architectural style. However, SOAP is actually a protocol specification. It provides a strict specification about the exchange of structured information between systems and applications. SOAP follows the RPC style and uses XML for its data format. Despite its age, SOAP is still used in many traditional enterprise systems and applications today.
- **REST –** Nearly all REST APIs rely on the HTTP protocol for transport and focus on resources. However, what constitutes a Representational State Transfer (REST) architectural style differs among developers. The ongoing REST API debate tends to revolve around Roy Fielding's definition of REST outlined in this https://grpc.io/dissertation. According to Fielding, for an API to be REST it must adhere to the Hypermedia as the Engine of Application State (HATEOAS) constraint. Today you will find APIs with different elements of REST, but most of them don't conform to HATEOAS. Some developers use the Richardson Maturity Index to determine an API's level of REST compliance, and Phil Sturgeon has created an updated version of this model.
- **Hypermedia –** This style is based on REST and focuses on embedding links and providing its own URIs for additional resources. Most Hypermedia APIs tend to conform to HATEOAS. For more on hypermedia, check out this API Intersection podcast episode which dives

deep into the pros and cons of hypermedia, why developers and the business side benefit from working together, and how to create a shared vocabulary that becomes the foundation of your organization's API design.

We didn't include GraphQL in this list because it is a query language and runtime for APIs, not an API architectural style. There are several reasons you might adopt GraphQL for some of your APIs.

Discussing which styles will work best for different use cases is the first step. Once you know which style to use, you can choose an API specification and then start designing your API. When your design is complete, you can use visualizations to gain an even better understanding of your API.

2) Visualize Your API Designs

Collaborating on designs using visualization can help everyone involved in the design process understand the APIs in development. Many non-technical people find working on API designs with a text editor difficult. However, with a visual editor, they can participate in the API design process and provide feedback regardless of technical skill. And visualization can help you notice things in your designs that you might not notice using a text editor.

For example, the team at Highmark uses Stoplight's visual editor to collaborate on API designs. The visual editor helped the team see that the payload for the "ID card API response model" included too much information. The technical and non-technical team members at Highmark understood the problem thanks to visualization and fixed it before the team began writing any code. Seeing the payload in a visual format instead of text made it easier to see that the payload was too large and not what the team wanted.

Collaborating on API designs using a visualization tool can help you find problems that you wouldn't necessarily have found using a text editor. You can also visualize API and code dependencies to track changes that might break your APIs. Visualization can provide valuable insights, but if you want to gain an even better understanding of your APIs, you need to get critical feedback from stakeholders.

3) Get Lots of Feedback

Getting feedback from stakeholders throughout the entire API lifecycle will help you understand and improve your API designs. And getting feedback from your peers and end-users will bring you new perspectives about your APIs and inspire API design ideas you may not have thought of on your own.

You can use a tool like Stoplight Studio to gather feedback from other developers on your API designs and a code hosting platform like GitHub, GitLab, or BitBucket to conduct peer code reviews. And you can gather feedback from end-users in a number of ways:

- Survey end-users directly about use cases relevant to your API.
- Ask client developers about potential problems with your API.
- Use customer-centric product management software like ProductBoard to convert end-user and developer feedback into valuable insights.
- Ask your customer support team if they know of any API-related problems or issues.

Once you've gathered information from end-users you can turn that feedback into user stories and use those stories to create your Jobs to be Done. The Jobs to be Done facilitates the rest of your API planning and design process. If you can get feedback from end-users early in the design process when you

have mock APIs, you can improve your API design long before you write any code. Designing your APIs in a collaborative way that includes constructive feedback and criticism will ensure you deliver high-quality APIs and a great developer experience.

A Collaborative Effort

Designing consistent, high-quality APIs requires that you continually learn about API best practices and new trends in API design. It also requires a collaborative approach to design where you leverage the knowledge and viewpoints of all stakeholders. A tool for APIs like Stoplight can help you not only design great APIs for developers and end users but also gain a better understanding of your APIs through effective collaboration.