## What Is an API?

### Applications and Interfaces

Hi, My name is Dan Appleman, and welcome to this Executive Briefing on APIs, or application programming interfaces. API is one of those terms in computer science that can sound very intimidating to those who aren't familiar with it, but it's actually very easy to understand. Let's start with the word application. You know what an application is. You use them all the time. For example, you might run Excel on your desktop, or laptop, or an app on your phone. You've used web applications. Some are complex, like Salesforce, that you may use to manage your business. And, of course, Pluralsight itself is an application. See? Application is easy. Now, let's skip over to the word interface. An interface defines how you work with an application. In Excel, you might enter a formula using a keyboard. In Salesforce, you might create a lead. In Pluralsight, you might do a search for one of your favorite authors. In each of these cases, you, the user, are working with the application's interface, the user interface, or UI, that is, the interface that just designed for users. People using applications, apps, or web applications is something everyone is familiar with. But if the person can use an application, how about one application using another application? Imagine if you have an application that could make a web request. In the previous example, searching for an author on Pluralsight, we entered the name in the search box, but take a look at the resulting URL. It has extra information in it that defines the search. An application could send that request directly to Pluralsight and get back a list of courses published by an author. That screen is easy enough for you to read, but take a look at what actually gets sent back to the application that made the request. Now, that's a lot harder to read. Trust me, it's not easy for software either. Most of what comes back has to do with appearance and user interaction, which is not what you're trying to retrieve.

### The Application Programming Interface

Applications are not users, so it makes no sense for them to use the user interface. They should have an interface that is designed just for them, one that an application can use that avoids all the extra content used for displaying information and interacting with users. We call that the application programming interface, or API. Let's look back at our Salesforce example. An application could retrieve a lead by entering its URL. But here, again, when you look at the raw HTML, what is returned by Salesforce? It's incredibly complex and subject to change at any time, so almost useless for other applications, but Salesforce publishes an API. Here is an example of the URL that an application might call to retrieve the same lead, which I'm entering into a workbench that is designed to help developers understand and build applications that use APIs. Take a look at what is returned. All the clutter needed to interact with users is gone. Only the data itself is there. In this case, it's in a format called JSON, JavaScript Object Notation, that is very easy for applications to process. Another common format is called XML, Extensible Markup Language. This particular API uses a protocol called REST, or Representational State Transfer. You can generally spot a REST interface by the fact that the operation is largely defined by the URL of the request. Though, there's a lot more to it that is beyond the scope of this course. REST interfaces almost always used JSON. Another older, more complex, but still useful protocol is called SOAP, or the Simple Object Access Protocol, in which a single URL can perform many different operations. SOAP interfaces always use XML. So there you are. An API is an interface that programmers creating software applications can use to interact with other software and applications. In fact, there are many cases where an application or software does not have a user interface at all, just an application programming interface. There are different kinds of APIs. Let's take a look at some examples. When you run an application on your phone or desktop, it uses the operating system's API, thousands of functions available to do things like creating windows, or storing, or retrieving data. For example, a Windows application uses the Windows API. Software developers building applications use APIs that are related to the language and environment that they are using. For example, a .NET developer would use the .NET Framework class library, which is what they call the .NET API. There are also many JavaScript frameworks and libraries, such as Angular and React, each of which has their own API. And of course, there are endless web applications and services that can be used by applications and by each other. In the next module, you'll learn about using or consuming APIs. In the final module, you'll learn about publishing APIs, which has its own set of issues. But before we get into that, there are two subjects that are essential to know about APIs, versioning and security.

### API Versioning

Let's take another look at Pluralsight's web application. It's easy enough for you to understand it. There are some menu items, a search bar, a sign in option, and other content. Now, through the magic of the Internet archive, let's take a look at Pluralsight's home page from early 2016. Looks different, doesn't it? But you can still understand it. The fact that it's changed doesn't impact your ability to use it. That's because you are a lot smarter than software. When an application uses an API to retrieve information or perform an operation, it expects it to work a certain way and return a certain set of data exactly the same way, every time. It's like a contract. If the software or application changes the way an API works, any program trying to use it will probably fail. We call this a breaking change because it breaks the contract, along with every application using that API. In order to allow applications to change their behavior or add new features to their API's, it's common to use some form of versioning. In other words, an application exposes or publishes different versions of their API, and the program can connect to a specific version. You actually saw this earlier in our Salesforce example. See that v48.0 in the URL? That means we're connecting to Version 48.0 of this particular API. I can change it to a different version number and see if there are any changes. In this case, it looks much the same. We'll go more into versioning later. It's an incredibly important topic. But for now, let's take a look at a topic that is even more important. Security.

### Security

There are some API's where security isn't much of an issue. When building an application that uses the Windows API or a JavaScript framework, there isn't really a need to secure the connection between a program and the operating system or a software library, that is effectively going to become an integral part of the application. But if your API is accessed through the Internet, security is a huge issue. For example, Google has a variety of API's that can be used to work with its services. They can't just work any time for anybody. For example, you wouldn't want anyone with your email address to be able to read your calendar. Google has made sure that every API request on your data has your permission to access it through that API. There are two parts to this, authentication and authorization. Authentication is making sure that you are the person who granted access. Authorization defines what you are allowing an application to do. For example, if I schedule a Zoom meeting, it gives me the option to add a calendar entry. Before it can do so, I get a prompt. I didn't get a log in prompt in this case because my account was already logged in with this browser. But I do have to grant permission for Zoom to use the Google API to access my calendar. Looking at our earlier Salesforce example, I had to log into Salesforce before I could use the workbench to retrieve the lead. Not only that, but my account had to have the necessary permissions to view the lead, and belong to a profile that allows it to access the API. So there you have it. An API is an interface, a contract, a way for one application to work with another application or piece of software. There are two key subjects that I'm going to return to in the next two modules, versioning, which is a way of allowing API's to change without breaking that contract; and security, which is critically important, especially on web API's. Now let's dig deeper into consuming, or using, API's.

## Consuming APIs

### APIs and Your Technology Stack

If you or your organization is creating software, you are using APIs. Modern software is built in layers from components. At the bottom is an operating system, Windows, iOS, Linux, Android, and so on. Then there are the various frameworks, software libraries that implement functionality that can be used by the main application. There may also be web applications, where the applications you build connect to external services through the internet, or you may be building applications directly on top of a cloud service or application. All of those elements make up what is called your technology stack, and each of these elements has its own API. This has some interesting ramifications. First, when you're hiring software developers, you'll rarely worry about whether a developer knows how to use APIs in general. They do because all modern software developers use APIs all the time. But you will worry about whether they know how to use a specific API, the ones that are part of your technology stack. That's because it can take a long time and a great deal of effort to become skilled at using a particular software, components or framework and their associated API. In fact, most of the effort of learning any new language is not in learning the language itself, but in learning the APIs from the frameworks and class libraries that are used with it. When it comes to choosing APIs, what you're really talking about is defining your technology stack, choosing the frameworks, components, and services that you will be using in your organization. Now, how to select elements of your technology stack is a subject that is far beyond the scope of this briefing. There are a number of courses on Pluralsight that can help you with that, including my own course, Building Software That Lasts ‑ A Guide to Maintainable Software, and the course How to Get the Technology Stack Right For Your Business. That said, there are issues that relate specifically to APIs that you should consider when choosing the elements of your technology stack. Let's look at those next.

### Evaluating an API

Examining the API of a framework software library or web application should be part of any selection process. There are a number of issues to consider. You learned about the importance of versioning in the last module. As a consumer of an API, you'll want to know if the publishers of that API are using versioning. You'll also want to know what steps they are taking to avoid breaking changes, updates that might impact your application. This applies to any type of API, but takes different forms. Consider an operating system. It's a huge problem when an operating system vendor makes changes that break your applications. If you use Windows, you face every update with a certain amount of fear and anxiety for exactly that reason. Though not common, there are definitely cases where Windows applications have suddenly stopped working correctly after a Windows update. When it comes to using frameworks or software libraries, you may have a choice as to whether to distribute a specific version of the library or automatically use the latest. For example, a web application you create might have the option to automatically download the latest version. While this might be okay for a noncritical in‑house system, you should never do so for a production or critical system. Always use a specific version of the library and create a process to evaluate updates before going live with them. For web APIs, you should be very concerned if the publisher or vendor is not versioning their APIs, it puts your application at significant risk. When it comes to security, most of the issues relate to the implementation of the software that is publishing the API, how it is built. That is another huge subject. But when it comes to consuming a web API, there are three issues that stand out that you should absolutely pay attention to. First, is the API using an encrypted connection, HTTPS? It may not be an issue for a public API, but for anything involving even remotely sensitive or personal information, it is essential. Another thing to look for is whether the publisher includes documentation, examples, and best practices for using their API securely. If they don't, that's a red flag because you don't want your developers, who may not be security experts, to have to figure out these things for themselves. And finally, remember, there is no such thing as perfect security, so you want to make sure that whoever is publishing the API is committed to quickly fixing security flaws as they are discovered.

### Updates and Maintenance

People often think about building software as a project. You have a set of requirements, you build the software, and at some point, you're finished. You can just use the software and only change it if the requirements change, say a new feature is desired. That may have been true in the earliest days of software development, but it is definitely not true with modern software development. Your technology stack needs to be maintained, and much of that maintenance is tied to the issues we keep coming back to, versioning and security. Even if your application is only using the operating system and software components that you distribute, over the long term, there is a chance that something will change to break it, an operating system update, or new security restrictions in browsers might come along and require you to update your application. If you are using web APIs, it becomes even more critical to stay up to date and watch for updates. You will periodically want to update your application to use newer APIs, even if you don't need any of their new features. That's because it's costly for publishers to maintain APIs, so they tend to deprecate and discontinue older ones. Sometimes they'll discontinue an API or service completely. One example I ran into was Google Image Charts that was deprecated in 2012 and can be shut down any time without notice. Fortunately, they published adequate notice, and I was paying attention. So I've long since migrated our software to newer charting technologies. Even publishers with the best intentions will sometimes blow it and release a breaking change. For this reason, you'll want to make sure that your application is built with robust error handling and diagnostics, so you can detect problems quickly and reach out to the publisher to resolve the issue. You've seen that consuming APIs is really about your technology stack, which is a big subject. We focused just on the API part, looking at versioning, security, and the scope of APIs, where it comes to selecting, using, and maintaining them. You may feel I'm a bit paranoid on the subject of maintenance and versioning when it comes to consuming APIs. I suppose I am, but trust me, it's justified, as I've experienced all of these issues first hand, And if you think consuming APIs has its challenges, just wait until the next module, where we'll look at publishing them.

## Publishing APIs

### Should You Publish an API?

While every organization that builds software consumes APIs, publishing APIs is much less common. Before you decide to publish an API, there are some questions you should consider. What are you trying to accomplish? Are you building a software library or a framework that you want developers to share? Is there actually a need for automation? Maybe a user interface is sufficient. Don't create an API just for the sake of creating an API. You should have a real need or opportunity before going down this path. Who is going to use it? Is it for in house use, or will outsiders be using it? If available to outsiders or even other departments or divisions within the same organization, the costs to develop and maintain an API can be significant. Consider for a moment what is involved in creating a software library or framework. You need good design. You need to decide how you will handle versioning. You need good documentation. You need to plan on supporting it. Since your library or framework probably uses other APIs, you need to do all of the things mentioned in the previous module with regards to maintenance. If you are creating a web application, you need all of those elements, plus a security strategy and the ability to maintain, monitor, and scale your application infrastructure, whether in house or in the cloud, and the funding to keep it running. I think you see my point. Publishing an API, especially a web API, is actually a big deal. It's not something one does lightly or casually, but if you still want to go ahead, the rest of this module will go deeper into the topics I just mentioned. Let's start with design.

### API Design

You might think that because every software developer knows how to consume APIs, that they are all qualified to design and publish them. That is definitely not the case. It's very easy to design a bad API. Let's take a look at some of the design issues that you or your team will need to consider. If you're building a software library or framework, you'll need to define the languages and platforms that you support. For example, a JavaScript library will obviously support JavaScript, but there's JavaScript and there's JavaScript. What versions of JavaScript will you support and which browsers and services? For web applications, you'll probably use a REST interface, but SOAP may be an option for some situations. You'll want to consider carefully what functionality you want to expose with your API. If you miss something the developers need, they may find your API useless and complain or turn to another API, perhaps one from a competitor. If you expose functionality that is rarely used, you'll have invested time and effort that is largely wasted and will continue to be wasteful because once published, you'll still need to maintain that functionality. The design will have a huge impact on how easy it is for developers to learn and use the interface. You'll want a clear and consistent naming convention, one that is as intuitive as possible to make functions and objects easier to learn and remember. You'll want to consider how the functions and objects in your API are organized, both in the interface and in the documentation. No software is perfect, and calls your API may result in errors. They may be the client's fault caused by invalid parameters to functions, or they may be due to bugs in your code, or in the case of a web application, they may be due to a failure or outage in the application itself. In all of these cases, you'll want to provide a mechanism to return error and diagnostic information to the caller. I haven't even discussed what goes into the architecture and design work that is needed before implementing the software that is exposing the API, and that's a huge subject, and it's far beyond the scope of this course. But as you can see, there's a lot that goes into designing a good API, and I haven't even gotten to the two most important topics. You guessed it, security and versioning.

### Security

Implementing a secure application programming interface is another one of those huge topics that I can't cover in great detail here, but when it comes to publishing the API, there are a number of issues that you will have to consider. First, you need to incorporate security into the design of the API itself. Consider a medical testing website where you can log in and see the results of your tests. The user interface is fairly straightforward. You log in, and you can see your test results. Now imagine the testing lab publishes an API that makes it easy for other applications to track your lab results using that API. You certainly don't want just anyone using the API to have access to all of the records. There has to be away for the calling application to certify to the labs application that it has permission to see and share your records. How this is done can be quite complex, but regardless, it has to be designed into the API from the very beginning. Making sure that an API handles authorization and authentication correctly and does not expose information that the caller does not have the rights to is essential. It is definitely an area where you will want to have someone with serious expertise on your team, as not every developer knows how to do this, and it's something you have to get right from the beginning. It's not unusual for API publishers to bring in expert consultants to help with this aspect of API design and development. By the way, this can be an issue even with software frameworks and class libraries, ensuring that they respect the permissions of whatever user is running the application. Now think about all the security features built into an operating system, and you'll see what I mean. The larger and more complex an API is, the greater the attack surface. That means the more ways an attacker can try to use the API to either extract data that it's not allowed to see or perform operations that should not be allowed. It is the responsibility of those implementing the API to validate parameters and write secure code, but beyond that, it's a good reason to keep your a API as small and simple as possible. Every feature and option you add is another potential security vulnerability. Finally, when you publish an API, you have to be prepared for the chance that people will find vulnerabilities in your application. You'll want to ensure that a reporting mechanism is in place and that your internal processes are designed to respond quickly and fix the issues. With luck, you can fix them in such a way that it does not break the applications using your API. Even using versioning won't help you here, as you may need to apply security fixes to older versions of APIs as well. I'll go into more detail on this in the next couple of clips. Right now, it's time to return to the subject of versioning, this time from the publisher's perspective.

### Versioning and Culture

It should be no surprise that as a publisher you'll want to use versioning for your APIs, whether it is a versioned web API or maintaining older versions of a software library or framework. But this is just the first step. Publishing an API demands a very specific software development culture, and it all comes down to a very simple commandment. No breaking changes! You must never break your client's application. The only exception to this would be a critical security fix, and we'll talk about that scenario in the next clip. There are many different software methodologies and cultures. For example, companies such as Facebook have taken pride in the slogan Move Fast and Break Things, a mantra of constant innovation. And as you learned earlier in this module, breaking or changing a user interface isn't a terrible thing. Some users might be annoyed and frustrated, others excited about the improvement, but most will adapt. But applications using an application programming interface can't adapt. An API is a contract. A culture of move fast and break things for a development team responsible for an API is a sure road to disaster. When publishing an API, the top priority over everything other than security must be no breaking changes. If you want to change something, it has to be versioned in such a way that current consumers of the API continue to work. No breaking changes means no breaking changes. It means you can't even fix a bug if fixing it would change the behavior of the API. And when I say behavior of the API, I don't mean just the documented behavior of the API. Trust me, people will use your APIs in ways you did not expect, and that usage, even if undocumented, should not be changed without very careful consideration. When you have the right culture, you do the right things. Other teams might cut corners on automated testing. A development team publishing an API makes automated testing a priority. It's a great way to reduce the risk of accidentally creating a breaking change. It's also a good idea to provide early access to beta, pilot or pre‑release versions of builds so that those using your API have a chance to validate their application on the new builds before they are released. It goes without saying that you'll need to update your API documentation for each release, and ideally provide pre‑release documentation beforehand. But even with your best efforts, you may release a breaking change, which brings us to support and maintenance.

### Support and Maintenance

Once you publish an API, those who use it will be relying on it. It becomes a part of their technology stack. It's a safe bet that you will hear from them. Some of the things you hear will be good and welcome, others less so. There are always bugs, not only because developers make mistakes, but also because, as mentioned earlier, people will do things with your API that you never imagined. While it's nice to see your API used that way, it creates a real predicament. You learned in the previous clip that some bugs can't be fixed because they would be a breaking change. If fixing a bug would change an undocumented behavior, it too can break someone's application. Still, bugs do need to be fixed, so you'll want to establish a careful review process to determine whether a bug can be safely fixed or if the fix needs to be incorporated into the next version of the API. Then there are security vulnerabilities. Those need to be fixed, and in many cases they cannot be versioned. In other words, you may need to go back and fix older versions of your libraries or frameworks or your web API. In some cases, this may lead to breaking changes. This puts you in a terrible position. Don't fix the issue, and those using your API are vulnerable. Fix it, and their applications break. This means that you need to put into place an alert system, a way that users can subscribe to critical information so that they know what they need to do. If it is a software library, they'll need to quickly download the updated software, fix and validate their own application, and make it available to their customers or users. If it is a web application, they'll need to drop everything and validate that their application can handle the change. The possibility that security issues may require you to patch older versions of your APIs means that you have to maintain those versions. That can become costly as time goes on and versions proliferate. To minimize this cost, you'll want to establish a deprecation policy, essentially putting your customers on notice that you will stop supporting older versions of the API at a certain time. Here again, a way to notify users is essential. It may seem like publishing an API is a big deal. That's because it is. It's a commitment, and comes with a great deal of responsibility. But in today's interconnected world, the availability of APIs to automate and integrate applications has become expected. Not having one puts your software or services at a disadvantage. My point is not to discourage you from publishing APIs, rather to encourage you to invest the time and effort needed to do it properly so that both your users and your organization will succeed. That means focusing on design, security, and a development culture that emphasizes reliability in all of its forms, from avoiding breaking changes, versioning APIs, and strong documentation and communication.