HOW MANY API CALLS SHOULD YOU DO?

Overview: APIs provide access to valuable mainframe data, but deciding whether to add to an existing API or create a new one can be a difficult task. Evaluating a variety of factors can help make it easier to determine which option makes sense.

As applications evolve, they will often need access to data and processing on the mainframe. If the necessary APIs are available, this becomes an easy task, however the architecture isn't always there, creating a roadblock. Teams are faced with choices on how to deliver value quickly and set up the architecture to address future challenges.

Assuming you have calls that may not deliver all the information you need, the choice is whether to add on to the existing call or produce a new call. How many calls should you have to do? When does it make sense to take the development time to make just one call? Conceptually speaking, one must look at the structure/design and make decisions based on what is already in place. Questions may arise, such as which applications are currently using this API and how changes will affect them. On that note, one must also keep in mind the complexity of adding additional APIs, whether it is on the side that is consuming the API or on the opposite side that is producing it. Your goal may be to always have just one call for information. That might be desirable but isn't always necessary. Don't let this desire for one-call purity stop you from innovating.

> *Just because it isn't "right" doesn't mean that it's "wrong".*

In most cases, you will find teams extending an existing call, in order not to expect anyone consuming the API to be forced to change theirs. When thinking of whether to go with one or multiple calls, look at the core functionality and determine whether there are cases where breaking down APIs truly makes sense or whether a single call is enough. Always think about the future.

There is a lot that goes into this decision but considering the following factors will help you to make the right decision.

When you should take the time to streamline it with one call:
• **When it's frequently used** – putting this in place will open up many more uses. Instead of duplicating the API work on the front-end, the front-end can just call the API to handle most of the work. Thus, reducing code duplication and bugs that may be caused by changing the behavior in one place and not the other. One centralized location that fits the needs of every API call.
• **Performance** – adding simple logic. If you are only adding something simple like a new field or some simple logic to an existing call, it can have better performance than creating a new call.
• **Saving development time** – putting it in now could save much-needed time in the future. If you see that soon there will be a request for an API call that handles everything, then it might be worthwhile investing the time to do it right the first time.

When you may be able to use more than one call:
• **When time is of the essence** – there is a critical need right now. We've all been here before, the so-called software dilemma where we need speed, quality, and features… now pick two. Sometimes the picks are speed and features and there isn't much time to perfect the API.
• **If the API is not used a lot** – when it's a smaller cog in the machine. Going back to the software dilemma, in this case it's like picking speed and quality. It may only introduce a small number of features that will not be further expanded in the future.
• **When testing a prototype in the field** – testing with users gives feedback and time to respond. Create something quick that can be streamlined later down the line. Feedback on the flow of work is more important here than low-level process of the API.
• **If separating the API calls** – it may allow for a modular approach, where in future work, using only a partial number of the calls may be needed. This is more common than you'd think. Especially dealing with large amounts of data, it may be better suited to break the API into parts.
• **Performance** – dealing with distinct operations. Performance on the back end is usually better than on the front-end, especially dealing with any CRUD (create, read, update, delete) type operations. Creating a call for each operation will perform better than having a call perform multiple operations.
• **When a part of the API will eventually be replaced** – if there are cases where there is a temporary solution in place, then it may be wiser to create a separate API for this temporary part of the call so it can be replaced in the future without causing further issues.

Expanding mainframe usage is important but can be daunting—don't let it be. Sometimes getting something now is better than nothing. Incremental improvement based on feedback is what Agile is all about. You can use this as a guide for your developers in how to approach these decisions. The important thing is to innovate and extend the value of the mainframe for the benefit of your users.