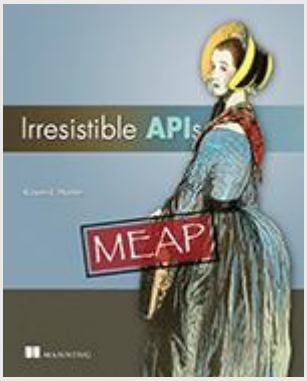

What is API First?

By Kirsten L. Hunter

In this article, I explain what API First is and how existing APIs are generally created.

API First is a design strategy for a company's entire product line, where APIs are the basis of every product instead of being a separate side product. To understand why API First is a good idea, you first have to understand how existing APIs are generally created.

There are various models for creating APIs, but generally the API accesses the backend directly in parallel with the main product. This means that if you want to make new applications, you have to either write more systems that access the backend or extend the API so that it supports both alternative products. Additionally, an API is frequently considered to be an "extra, nice to have" product rather than an important member of the product ecosystem. This creates serious problems with resource contention as the API competes against revenue producing products for engineering resources.

APIs as a Side Product

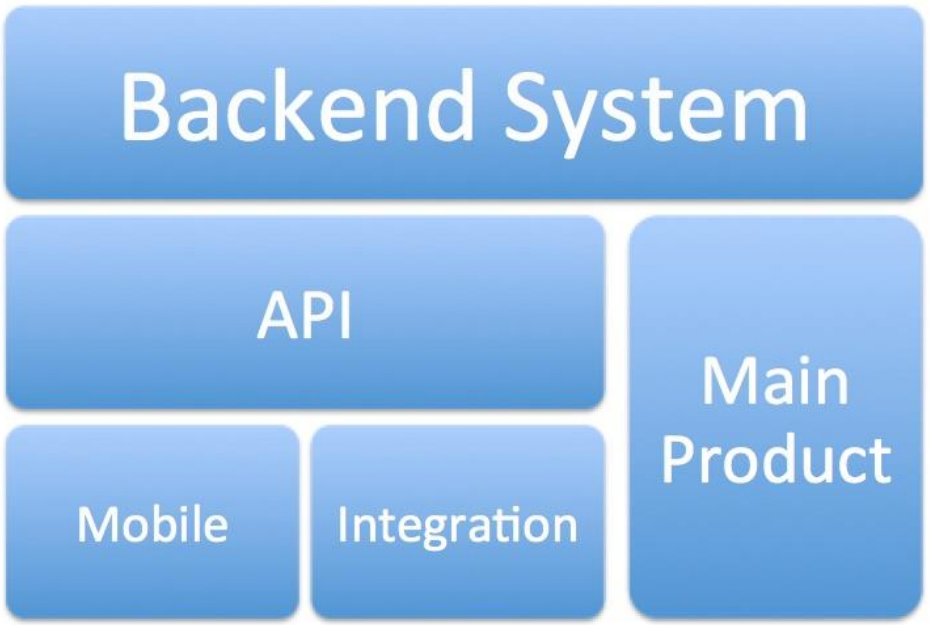

Figure 1 – Standard API Setup

Figure 1 shows an example of the "usual" setup for APIs – as you can see, the API is separate from the main product, and even if all secondary products run off of the API, there will be a mismatch between the two experiences.

Keeping everything entirely consistent is theoretically doable but a lot more work than simply restructuring the infrastructure to treat the API as a piece of core technology for the system. There is some thought that separating out the API from the main product can protect that product from attacks via the API, but truthfully the backend system is the critical piece, and separating out the clients in this way simply makes it harder to triage and fix problems that might occur in the product or the API. Keeping everything in the same pipeline helps assure consistency, reliability and as your product grows, it makes scaling much easier.

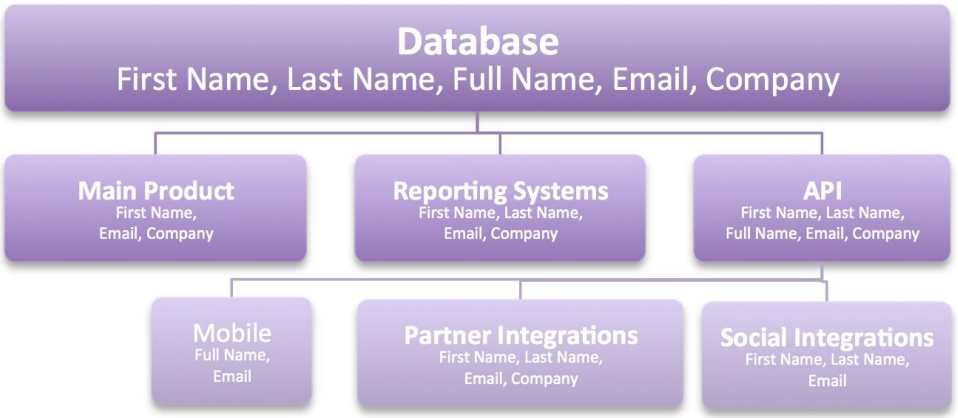

Figure 2 – API as One of Many Interfaces

Imagine that a company has a product for creating and updating contact information for users. The backend returns lists of users based on a database call from the client. Figure 2 demonstrates this case, where the main product communicates directly with the backend system, and is likely to retrieve data in a different way as a result.

When the backend system team adds a new "Location" field feature, it doesn't show up in the API until an engineer has time to add it there. This means that there will be a necessary lag between the addition of this field to the main product and availability within the API until and unless someone takes the time to write essentially duplicate code to retrieve the new fields.

There's a lot of technical debt incurred when you have multiple systems trying to reproduce a single interface. There's not likely to be a good process for comparing this API change to other APIs from the company – resulting in inconsistent results. In this case, the mobile app wouldn't allow or see locations, resulting in developer dissatisfaction and customer confusion and irritation. Once you've established how you want your users to interact with your system, it's best to support that everywhere.

When you have multiple teams creating products without a shared vision, you also tend to have poor communication between those teams. This can lead to bug fixes in one code base but not in the other, or inconsistencies between the items available from the system depending on which interface is being used.

API First Model

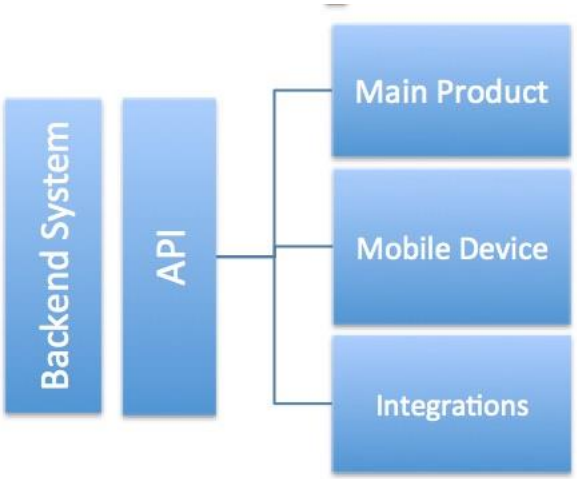

Figure 3 – API First model

In figure 3 we have an API First model. All of the products run off of the same interface. This ensures that each device, application or integration has the same resources available to use. These resources will be consistent across the entire product line. Note that just because an API resource is available within the system, you don't have to expose it to all the world – you can decide which of the API resources is internal, partner only, or open to anyone. It's still a great idea to have your API ready because when a major partner asks for access to some specific resources to support a use case, you have it ready to go.

API First also encourages communication between your backend team and each of the client engineering teams. Understanding use cases at a high level the helps create APIs that are usable for the use cases you understand up front, and more likely to support future use cases that come up. Once you're creating the API as a larger team, you'll find many places where different teams offer complementary resources adding to a much better structured system.

API First makes a lot of sense for any company – as soon as you have more than one product (for most companies that's going to be a website and a mobile application) you need to have a layer to protect the clients from changes on the server. A well-documented interface into the system, crafted with specific use cases in mind, allows you the freedom to change things around on the backend, as long as the interface doesn't change. Integrated testing is

easier, and the products running on the API will by their nature test the integrity of the system on a regular basis.

You can learn much more about APIs in my book Irresistible APIs, available from Manning Publications, Inc.