OpenLegacy

As appeared in
IBM Systems MAGAZINE

What to consider when considering APIs

Congratulations. You have decided to take the leap: You've heard about APIs, you've read about them. You realize that they are the new SOA and are the future of computing. You want your IBM i system to expose APIs and you want it as soon as possible. But what does that mean going forward, what do you have to take into account?

Application Programming Interfaces (APIs) are fast becoming the standard way for integration and systems architecture design. These well defined but lightweight interfaces enable unprecedented flexibility by allowing you to expose processes and data for anyone to use. Having a standard API to your IBM i based application means not only that it can be used by any existing web or mobile application but also that any new device, such as watches or glasses or Wifi-connected coffee mugs will be able connect to the same APIs in the same exact way.

That is all truly wonderful but moving to an API strategy does not come without its challenges. There are things to consider even beyond the technical challenge of creating an APIs layer. Like most thing these challenges are best confronted early on in the process where the cost of change is the lowest.

Security

The very nature of APIs suggest a certain lack of control over who will end up using them. That's their greatest strength and the reason they are so flexible and easy to use but that is also a source of possible security issues. When designing API security, it is important to note that there are different levels of security to consider, the first of which is authentication. Unlike services, API can be accessed by individual users as well as applications. This means that two sets of security standards might be applicable; oAuth for people and API keys for applications. Second is access control. As you might want to expose as APIs systems which were not initially designed for such use cases, you need to pay attention to what type of information your API actually expose. You may, for example, have far more information on you customers than you would have a 3rd party mobile app developers get. Finally and for the same reasons you might want to consider filtering the content of your data based on specific context so that only a controlled subset of your data can be pulled through the API.

Workload

While APIs themselves do not necessarily imply an increase of workload for your IBM i based application, you do need to consider the fact that it will now be much easier to access from mobile, web or any other platform out there. Mobile generated workloads especially might prove much more unexpected than you thought. Not only do usage patterns differ from web application (location based queries being sent tens of times a minute from each device) but development of these application is much more likely to be handled by developers in an outside application studio. These developers are not always aware of the importance of reducing back-end workloads.

Structure

Modern APIs usually mean REST APIs. REST is a great protocol for lightweight service activation based on HTTP, but in it's core it is not meant for services at all,

it is rather a resource centric protocol. The distinction is nuanced yet important. Those of us who remember the CORBA protocol may find some resemblance. The REST protocol describes a resource using a hierarchical URI (ie www.domain.com/customers/1002/accounts/233 means account 233 belonging to cusomer 1002) and through the HTTP method it describes what should be done with this resource (PUT for create, GET for read etc.) This is not the same as having a getCustomerAccount service. It requires a different design of the application and a different perspective of the system's design. Having a REST API which uses the URI line as a service identifier is technically possible (ie www.domain.com/getCusotmerAccount?cust=1002&account=223) but is conceptually wrong and will look unfamiliar and awkward to developers accustomed to well formed, resource-based REST APIs. This seemingly technical detail may prove quite significant as it is at the heart of the design of the application and is incredibly hard to change once the API solution is implemented.

Governance

Much like SOA services, APIs need governance strategies to keep them coherent and controlled. Unlike SOA services these strategies can not rely on a central services bus through which everything passes. Traditionally SOA was meant to connect different parts of the organization's IT, APIs on the other hand have a much more of an outward facing bias, aimed at outside users and 3rd party developers. This means that API management, the API equivalent to SOA governance needs to be both flexible and comprehensive. Everything, from lifecycle management to publication and description has to be done with external users in mind. It is important to avoid making assumption about the end user knowledge of the underlying systems. Additionally, much like the APIs themselves, the API management should be scalable. Having your API publishing web page unavailable to developers is not a great way to get ahead in the new API economy.

APIs are a great new way to think about your IT. While the concept of decoupling back-end systems from front-end ones isn't exactly new, it's implementation through APIs can bring new flexibility and agility to your organization, removing layers of complexity without compromising governance. But when considering the move to an API architecture, as in most things in life, planning ahead is the key to success.

OpenLegacy
www.openlegacy.com
sales@openlegacy.com

OpenLegacy USA
103 Carnegie Center, Suite 300
Princeton, NJ 08540
Tel: +1 609.608.0556