Resource Naming

HTTP Methods
Idempotence

In addition to utilizing the HTTP verbs appropriately, resource naming is arguably the most debated and most important concept to grasp when creating an understandable, easily leveraged Web service API. When resources are named well, an API is intuitive and easy to use. Done poorly, that same API can feel klutzy and be difficult to use and understand. Below are a few tips to get you going when creating the resource URIs for your new API.

Essentially, a RESTful API ends up being simply a collection of URIs, HTTP calls to those URIs and some JSON and/or XML representations of resources, many of which will contain relational links. The RESTful principal of addressability is covered by the URIs. Each resource has its own address or URI—every interesting piece of information the server can provide is exposed as a resource. The constraint of uniform interface is partially addressed by the combination of URIs and HTTP verbs, and using them in line with the standards and conventions.

In deciding what resources are within your system, name them as nouns as opposed to verbs or actions. In other words, a RESTful URI should refer to a resource that is a thing instead of referring to an action. Nouns have properties as verbs do not, just another distinguishing factor.

Some example resources are:

- Users of the system.
- Courses in which a student is enrolled.
- A user's timeline of posts.
- The users that follow another user.
- An article about horseback riding.

Each resource in a service suite will have at least one URI identifying it. And it's best when that URI makes sense and adequately describes the resource. URIs should follow a predictable, hierarchical structure to enhance understandability and, therefore, usability: predictable in the sense that they're consistent, hierarchical in the sense that data has structure—relationships. This is not a REST rule or constraint, but it enhances the API.

RESTful APIs are written for consumers. The name and structure of URIs should convey meaning to those consumers. It's often difficult to know what the data boundaries should be, but with understanding of your data, you most-likely are equipped to take a stab and what makes sense to return as a representation to your clients. Design for your clients, not for your data.

Let's say we're describing an order system with customers, orders, line items, products, etc. Consider the URIs involved in describing the resources in this service suite:

Resource URI Examples | Resource Naming Anti-Patterns | Pluralization

To insert (create) a new customer in the system, we might use:
POST http://www.example.com/customers

To read a customer with Customer ID# 33245:
GET http://www.example.com/customers/33245 The same URI would be used for PUT and DELETE, to update and delete, respectively.

Here are proposed URIs for products:
POST http://www.example.com/products for creating a new product.

GET|PUT|DELETE http://www.example.com/products/66432
for reading, updating, deleting product 66432, respectively.

Now, here is where it gets fun... What about creating a new order for a customer? One option might be: POST http://www.example.com/orders And that could work to create an order, but it's arguably outside the context of a customer.

Because we want to create an order for a customer (note the relationship), this URI perhaps is not as intuitive as it could be. It could be argued that the following URI would offer better clarity: *POST http://www.example.com/customers/33245/orders* Now we know we're creating an order for customer ID# 33245.

Now what would the following return?
GET http://www.example.com/customers/33245/orders
Probably a list of orders that customer #33245 has created or owns. Note: we may choose to not support DELETE or PUT for that url since it's operating on a collection.

Now, to continue the hierarchical concept, what about the following URI?
POST http://www.example.com/customers/33245/orders/8769/lineitems
That might add a line item to order #8769 (which is for customer #33245). Right! GET for that URI might return all the line items for that order. However, if line items don't make sense only in customer context or also make sense outside the context of a customer, we would offer a POST www.example.com/orders/8769/lineitems URI.

Along those lines, because there may be multiple URIs for a given resource, we might also offer a GET http://www.example.com/orders/8769 URI that supports retrieving an order by number without having to know the customer number.

To go one layer deeper in the hierarchy:
GET http://www.example.com/customers/33245/orders/8769/lineitems/1
Might return only the first line item in that same order.

By now you can see how the hierarchy concept works. There aren't any hard and fast rules, only make sure the imposed structure makes sense to consumers of your services. As with everything in the craft of Software Development, naming is critical to success.

Look at some widely used APIs to get the hang of this and leverage the intuition of your teammates to refine your API resource URIs. Some example APIs are:

- Twitter: https://developer.twitter.com/en/docs/api-reference-index
- Facebook: http://developers.facebook.com/docs/reference/api/
- LinkedIn: https://developer.linkedin.com/apis
