Why is HATEOAS important?

In our *What is REST?* article we focused on a few of the important constraints of REST that are relevant to the SRP HTTP Framework. When discussing the *Uniform Interface* constraint, we provided an introduction to the concept of *Hypermedia As The Engine Of Application State* (or **HATEOAS** for short). In our summary of the concept of HATEOAS, we emphasized its importance but suggested the reader visit this present page to understand **why HATEOAS is important**.

Quick Answer

This Wikipedia article on HATEOAS nicely summarizes some key objectives and benefits of HATEOAS, so we will cite these here (the emphasis in the below citations are ours):

- Since "...application servers provide information dynamically through hypermedia. A **REST client needs little to no prior knowledge** about how to interact with an application or server beyond a generic understanding of hypermedia."
- "**HATEOAS**...decouples client and server **enables the server** functionality **to evolve independently**."
- "**All** future **actions the client may take are discovered within resource representations** returned from the server."
- "**RESTful interaction is driven by hypermedia**, rather than out-of-band information."

To be fair, there are plenty of people who appreciate REST in some respects, but ultimately do not see the significance or value of HATEOAS. This approach to RESTful APIs is often referred to as *pragmatic REST*. RESTful APIs that emphasize HATEOAS is sometimes referred to as *purist REST* (at times derisively by non-purist developers). This article from Ben Morris represents a well reasoned argument against the necessity of HATEOAS. He acknowledges the intent of HATEOAS such as those listed above, but ultimately concludes that the cost of HATEOAS outweighs the tangible benefits. Our intent is not to debate the subject or pick apart any arguments against HATEOAS. Rather, we prefer to just outline our case for HATEOAS so each developer can make up their own mind. The important part is for a developer to decide early on whether or not HATEOAS will be a normal part of the API to avoid shoehorning it in or excising it out later on in the life-cycle of API development. To facilitate developer choice, the SRP HTTP Framework does not enforce HATEOAS implementation but it does provide tools to implement HATEOAS relatively easy.

Digging Deeper

It might help to explore some of these concepts a little more and illustrate with a simple use case. We'll do this by looking at the core terms in the HATEOAS acronym: *Hypermedia*, *Engine*, and *Application State*.

Hypermedia

Hypermedia refers to content in a resource that is itself, or somehow contains, a URI link to another resource. The most common type of hypermedia is we experience is hypertext. It is arguably the most significant reason why the web so easy to navigate and explore. Consider any website that you load into a browser. How does one discover and explore additional content that is available? We all know the answer: through the URI links available on the page. In HTML, we usually encounter these links in the form of text or images that have been created using the anchor tag. Visitors do not require a road map nor or a manual of available URIs to access other content. The embedded hypermedia provides a visitor everything needed to continue. This addresses one important benefit of hypermedia: **self-documenting systems**. That is, assuming that the links are well labelled and clearly identified on the page, they serve as self-documenting elements that the users can easily understand.

Another benefit of hypermedia, although one that might not be obvious to the typical visitor, is that URIs can change without advertising this to the user. When a user clicks a link, the user normally only cares that they will arrive at the correct destination and not worry too much about the URI that got them there. If the link isn't broken then all is good. Therefore, sites are free to change their URIs as long as they allow the user to find what they are looking for. This introduces another import benefit of hypermedia: **client/server independence**. That is, web *servers* should be allowed to change as needed and not worry about breaking any client functionality (i.e., the web browser *client*) because the server is always providing the client with current URIs.

These two benefits can also apply to web APIs. If our resource objects contain hypermedia, then our APIs become *self-documenting*. If the system that communicates with our API understands that it will receive hypermedia then we will achieve *client/server independence*. Our APIs, then, can evolve much easier (as well as the client relying on our APIs). This Martin Nally article doesn't discuss REST or HATEOAS directly, but it does provide a very thoughtful and compelling reason for using hypermedia (aka links) in API responses.

Engine

In this context, *engine* is an abstract term meaning *the force that produces a result*. By itself it doesn't mean anything but it does connect the force (*hyperm edia*) with the result (*application state*).

Application State

For our purposes, *application state* refers to the resource content as it exists in the client at a specific point in time. As we have noted above when describing hypermedia, web servers (and web APIs) return digital content, but this content can be separated into two categories: *static* and *hypermedia*. Static data is what a client considers meaningful and directly related to the URI itself (e.g., `GET /orders/1234` is expected to return information related to order #1234, such as customer number, order date date, line item detail, etc.). Hypermedia data, however, informs the client what new states are available. For instance, our order resource might also include the following hypermedia:

- A link to the related customer resource (e.g., `GET /customers/1`).

- A link to each related item on the order (e.g., `GET /orders/1234/lineItems/1`).
- A link to the shipping vendor's tracking information (e.g., `GET https://tools.usps.com/go/TrackConfirmAction?tLabels=9374859697090312216947`).
- A link to cancel the order (e.g., `DELETE /orders/1234`).
- A link to see all open orders (e.g., `GET /customers/1/openOrders`).

Each of these links are capable of changing the *application state* (i.e., the resource content as it exists in the client at a specific point in time). Note, however, that some of these links simply request additional resource content (i.e., via the *GET* method) but links also exist to request a *change* to the resource state (such as cancelling the order). The important point is that all of these links were provided by the server at the time the resource was requested. This implies that the server used the *resource state* (i.e., the resource content as it exists in the server at a specific point in time) to determine which *application state* links were applicable when the request was made.

This describes a type of functional contract. Servers inform clients what can be done through hypermedia and clients inform servers what they want to do through the same hypermedia. This provides the basis of statelessness with our RESTful APIs. That is, servers don't track (and don't care about) the application state of each and every client. Rather, servers (i.e., our APIs) only cares about requested resource state changes when a client calls our API. When the request is made, our API returns resource content (static and hypermedia) that is appropriate at that time. Likewise, clients also should not care about the resource state of the server. They should only be concerned with the application state and any responses the server returns when a change in the resource state is made.

Let's revisit the `GET /orders/1234` API again. It was previously suggested that the resource might include a link to cancel the order. In order for HATEOAS to work as intended, the server needed to verify that order #1234 was capable of being cancelled. This could rely on a number of factors, such as permissions of the user, whether the product was already shipped, or terms and conditions of this order. In all cases, the server returns or withholds the link (hypermedia) order cancellation link as needed.

In Summary

HATEOAS is an important design element of APIs that contributes to the *uniform interface* constraint of REST. However, HATEOAS will be underutilized if clients aren't programmed to expect and utilize hypermedia content. In these cases, RESTful APIs are not much more than ways to access static resource content. Therefore, in order for HATEOAS to be worthwhile, both clients and servers need to support hypermedia content and use it request changes in the resource state.