ETags in REST

00:00: Welcome to lesson one of module 10: ETags in a REST API. In this lesson, we are going to start talking about entity tags and we're going to discuss what they're useful for. We are then going to look at the client's side and we're going to touch on the kinds of requests that the clients need to do in order to benefit from entity tags. We are then going to start talking about the solution and we are going to talk about deep versus shallow tags. And of course, finally we are going to look at the Spring support and we are going to do an implementation. Alright, so let's get right into it.

00:36: At the very simple level, the ETag is an HTTP header. Now, that header is primarily used for HTTP caching. It's a caching mechanism and it involves conditional requests. These are specific types of requests that actually lay out specific conditions for the response. So we are going to be looking at what these are and we are going to be looking at how to form a conditional request and how to send that over to an API that is ETags-enabled. So let's do that with a simple example. We're going to send a GET request to this API, and we are going to simply retrieve a full resource by ID.

01:12: And here's where we're first seeing the ETag support. So we're getting back the 200 Okay. We're getting back JSON because, of course, we were asking for JSON. Notice the accept header on the request. And now we're getting this extra HTTP header. We're getting the ETag. And now that we have the ETag, the next time we're retrieving this particular resource, we're going to make use of that ETag and we're going to send a conditional request. Alright. So let's jump to this second conditional request and let's start understanding how we can form and send this type of request.

01:48: So, first of all, what is a conditional request? This kind of request requires a specific type of HTTP header. These are the if headers. So we are going to add the if header on a standard GET request and that is going to basically turn that request into a conditional request. And the two if headers that we're mainly concerned with here are the If-None-Match header and the If-Match header. So these are the two headers that we're going to use to actually make use of this ETag's functionality.

02:20: Because keep in mind that up until this point, yes, we did receive the ETag's data but we didn't really do anything with it yet. So this is exactly what we're going to do now. We're going to make good use of the If-None-Match header and we're going to basically request the same full resource and we're going to use this header to instruct the server to instruct the API, basically, to only return that resource if it has changed in the meantime. So between the time that we last retrieved it and this time, if the resource has changed, then the server should actually return it. If the resource has not changed, then we already have that representation. So there is no point in having the server give us the resource again because we do already have it.

03:09: Alright. So let's have a look at the example. So, as you can see, we are sending the same exact request. We are expecting JSON and we are asking for the same full resource, only this time we are also specifying this If-None-Match header and we're passing in the ETag's value that we got last time. So as we've been discussing, this is going to tell the server not to return the representation of this resource if the ETag of that resource is going to match because we already have that representation, so there is no need for the server to return it again. However, if the ETag doesn't match, which would mean that the resource has changed and we basically have an older version of that resource, then the server should and will return the new representation.

03:54: And here we go. This is exactly what we were expecting. Because the resource has not changed and basically the ETag is the same, the server, the API, will simply return the 304 Not Modified, which is simply telling the client that the resource has not been modified. However, notice the critical thing here. The response from the server is very, very lightweight. There is no body on that response because there is no representation of the resource to be returned. And that is really where ETags can be critical. Using ETags properly can make some types of APIs, specifically heavyweight APIs that have large representations for resources, it can make those APIs a lot more lightweight. Because all of a sudden, there is no more data going back and forth over the wire if resources don't change very often. So that is why ETags can be so helpful. Alright. Now that we understand the basics of ETags, let's have a look at the potential ways of actually getting them implemented.

05:00: At a very high level, we have two alternatives of doing the ETag's implementation. We have a deep implementation and we have a shallow implementation. So let's start with the shallow implementation because that is definitely the easiest of the two. Now, the shallow implementation is very simple to understand. Basically, the API will process the ETag request just as a normal request and at the very end of the request response cycle, right before the server marshals the response and sends that over to the client, at that point, a hash will be calculated out of that response and the ETag's value will be set. So the reason this is called a shallow implementation is that most of the work that the API has to do is going to be done anyway. So ETags really don't help with not executing the request. The request still gets fully-executed. It consumes all of the normal resources that it would consume if it was not an ETag request. And it's only at the very, very end that the ETag actually gets calculated.

06:05: Now, it's important to understand that we are still getting a lot of benefits here, namely the fact that at the very end, when the ETag gets calculated, if it matches the value that the client sent, then of course, the response will no longer be marshalled and so we're still getting the huge benefit of less data over the wire. However, there is an alternative. There is a way to actually benefit even more from using the ETag and potentially, if we have a mechanism to do that, skipping the execution of the entire request. So that is the deep implementation.

06:42: Alright. So now that we discussed the shallow implementation and we briefly touched on the deep implementation as well, let's actually go through the deep implementation at the high level. So with the deep implementation, an ETag enable request comes into the server, it starts processing. However, based on that ETag value, there are several options, several types of mechanisms that we could use to simply determine if the ETag has changed.

07:10: So if the ETag has not changed, then we really don't need to hit the persistence layer and consume all of those resources. We could simply return back the three or four to the client without doing any of that. But of course, the main point here is that we need the mechanism to be able to verify if the ETag has changed or not. Remember that in the shallow implementation, it was much easier to calculate the ETag because we were doing that at the very end when the response was already computed. So at that point, it's easy to apply a hashing algorithm over that response and calculate the ETag value. In the beginning, however, of course, it's much more difficult. And so the implementation does have a lot of potential but we actually need to implement it. We need to integrate that into our application. And that is why this is called the deep implementation versus just the shallow implementation, where we're not really affecting our internal logic and we just come in at the very end.

08:14: And that is it. That is the foundation of what ETags are and now it's time to actually jump to a proper implementation and see how these work in action. Alright. So now that we are over the theory part of the lesson, let's actually start doing the Spring implementation. If we're going to do the shallow implementation and, of course, that is what we're going to do here, we are not going to have to do that manually. We're going to use the Spring implementation and that, more specifically, is called the shallow ETag's header filter. So this filter right here which, of course, does exactly what the name says. It's going to add the ETag's header on the body of the response right before the response gets marshalled. So basically, the shallow implementation that we already discussed.

09:01: Now, keep in mind that we are in a boot project. So we are going to benefit from the way that boot allows us to set up a filter but, of course, this is just a standard servlet filter. So, at the end of the day, there is nothing special about setting this up. You can do it in whatever you're using to set up your servlet configuration. So that may be the web XML, that may be a Java initializer or it may be Spring boot. Alright. So let's go back to the web configuration here and let's add in our filter configuration.

09:35: And there we go. We have the filter defined as a bean and then we have this filter registration bean that basically helps us to register the bean in the servlet configuration. So notice that we're adding the URL pattern that this filter is going to be applied to and notice that we're setting the order. So pretty much the exact same set of details that we would set up in the web XML. And that is literally it. That is all it takes to set up the shallow configuration. So you can now start seeing why this is called the shallow configuration because it really is shallow. It really does come at the very end of the HTTP request and it doesn't need any integration with the actual application code, which, of course, make it a very easy implementation but still also very powerful.

10:24: Alright. So now let's start up the system and let's see this implementation in action. Let's see that on the client side. When we send out the GET request, we are actually getting the ETag's HTTP header. Alright. So now with the system running, we're simply going to send this GET request and we're basically going to try to retrieve a role by ID. So nothing very complicated, just hitting the roles API and just retrieving one of the roles resources by its ID.

10:53: So let's actually add the except header here, we want JSON data to come back from the API. And let's now send out the request. Okay. So because the credentials were cashed, security did not kick in. Of course, if you are sending this request for the very first time, you're going to have to enter the credentials. And now, let's actually have a look at the response. So we are now seeing the ETag value. We're seeing the new ETag header being returned on a standard GET response. So that is the first step. Now, of course, as we've discussed in the first half of the lesson, the ETag's header here is just the first step. We're not really benefiting from the ETag's functionality until we actually use it. So let's do that. And now we're going to send the second request and we're going to make use of the ETag value. So we're going to set this value with a custom header. We're using the If-NoneMatch header, exactly like we talked about in the first part of the lesson, and we're sending this exact request again.

12:05: Now, keep in mind that if the ETag's functionality works as expected, we're not going to get back a 200 Okay and we're certainly not going to get back any response body. So what we should get back is a 304 here, in no way should it be a response body. Okay. So let's send this request and let's make sure that everything is functioning as expected. And there we go. We are getting back the 304 and if we go to the response body, we're getting an empty HTTP response. So, this is exactly what we wanted. We wanted not to have this data sent over the wire again since the underlying resource has not changed. The underlying resource is exactly the same. The ETag's header does match, and so there is no reason that the server should send this data again over the wire. Alright. Hope you're excited. See you in the next one.