RESTful Web Services: Preventing Race Conditions

Alexandre Martins Follow
Oct 16, 2018 · 3 min read

One of the core premises of RESTful web services is that HTTP should be seen as an application protocol rather than just a transport protocol. It comprises a whole bunch of semantics that allows us to build robust distributed systems. And for some cases, when multiple consumers manipulate the same resource, therefore changing its state, the solution should be robust enough to prevent the system to get into a race condition.

But how HTTP could prevent that?

HTTP provides a simple but powerful mechanism for aligning resource states by making use of entity tag or ETag and conditional request headers. An ETag is anything that uniquely identifies an entity, such as the ID associated with a persisted resource, a checksum of the entity headers and body, etc. If this resource changes — that is, when one or more of its headers, or its entity body, changes — then the entity tag changes accordingly, reflecting this new resource state.

When a response contains an ETag associated to a resource state and you want to continue working with this same resource, it's recommended to use this tag in subsequent requests (called conditional requests), otherwise, the resource state might eventually become out of sync with service one, returning something like a `409 Conflict`.

Conditional requests happen when the current ETag is supplied to a conditional request header, such as `If-Match` or `If-None-Match`, when the user is requesting to update a resource for example. The service will then check the precondition, by comparing the current resource ETag with the one provided in the request. If it's satisfied, then the server proceeds and process the request, otherwise it concludes that the resource has changed and responds with a `412 Precondition Failed`.

Example

Given an online shop for home goods, where two employees — **Admin1** and **Admin2** — are responsible for administrating the contents. In this scenario, both administrators are trying to change the state of the same product (the Weber BBQ), around the same time. **Admin1** wants to lower the product price down to $300.00 and **Admin2** wants to change its state to "Not Available". Firstly, both administrators `GET` the current product state independently of one another by doing the following request:

```
GET /product/1 HTTP/1.1
Host: myshop.com
```

Returning the following content response. Note that the service's response contains an ETag header.

```
HTTP/1.1 200 OK
Content-Length: 265
Content-Type: application/json
ETag: "686897696a7c876b7e"

{
 "name": "WeberFamilyBBQ",
 "description": "Great for parties and cooks a neat roast too.",
 "price": 399,
 "status": "InStock"
}
```

When **Admin1** does a conditional PUT, including a `If-Match` header with the ETag value from the previous `GET`.

```
PUT /product/1 HTTP/1.1
Host: myshop.com
If-Match: "686897696a7c876b7e"

{
 "name": "WeberFamilyBBQ",
 "description": "Great for parties and cooks a neat roast too.",
 "price": 399,
```

```
 "status": "InStock"
}
```

And as the product state hasn't changed since the last request, then the request is thus successful! Notice that the response returns an updated ETag value, reflecting the new product state.

```
HTTP/1.1 204 No Content
ETag: "616898r96a8cy86b8eee11"
```

Little time after **Admin1** has updated the product, **Admin2** does another `PUT` request to the same product, including the same `If-Match` header with the ETag value from the `GET` request.

```
PUT /product/1 HTTP/1.1
Host: myshop.com
If-Match: "686897696a7c876b7e"

{
 "name": "WeberFamilyBBQ",
 "description": "Great for parties and cooks a neat roast too.",
 "price": 399,
 "status": "InStock"
}
```

The service then determines that someone is trying to change the same product, using an out-of-date resource representation (ETags are different!), and responds with a `412 Precondition Failed` code. No race conditions whatsoever!

```
HTTP/1.1 412 Precondition Failed
```

Conclusion

Although ETags and conditional request headers make up a powerful mechanism for dealing with concurrency, one thing to keep in mind is that, depending on the amount of

computation performed by the server to generate an ETag, response times might increase considerably. So use it only if you need it!

Special thanks to Jim Webber for helping me with this post. For more information on RESTful Web Services, check out his latest book (written together with Savas Parastatidis and Ian Robinson)— REST in Practice: Hypermedia and Systems Architecture.

API Restful Http Web Services