

Combining OAuth and JWT to gain performance improvements

by **Tosin Ogunrinde**¹⁰

on REST¹¹, Token¹², OAuth¹³, JWT¹⁴, Authentication¹⁵, Authorisation¹⁶, Architecture¹⁷

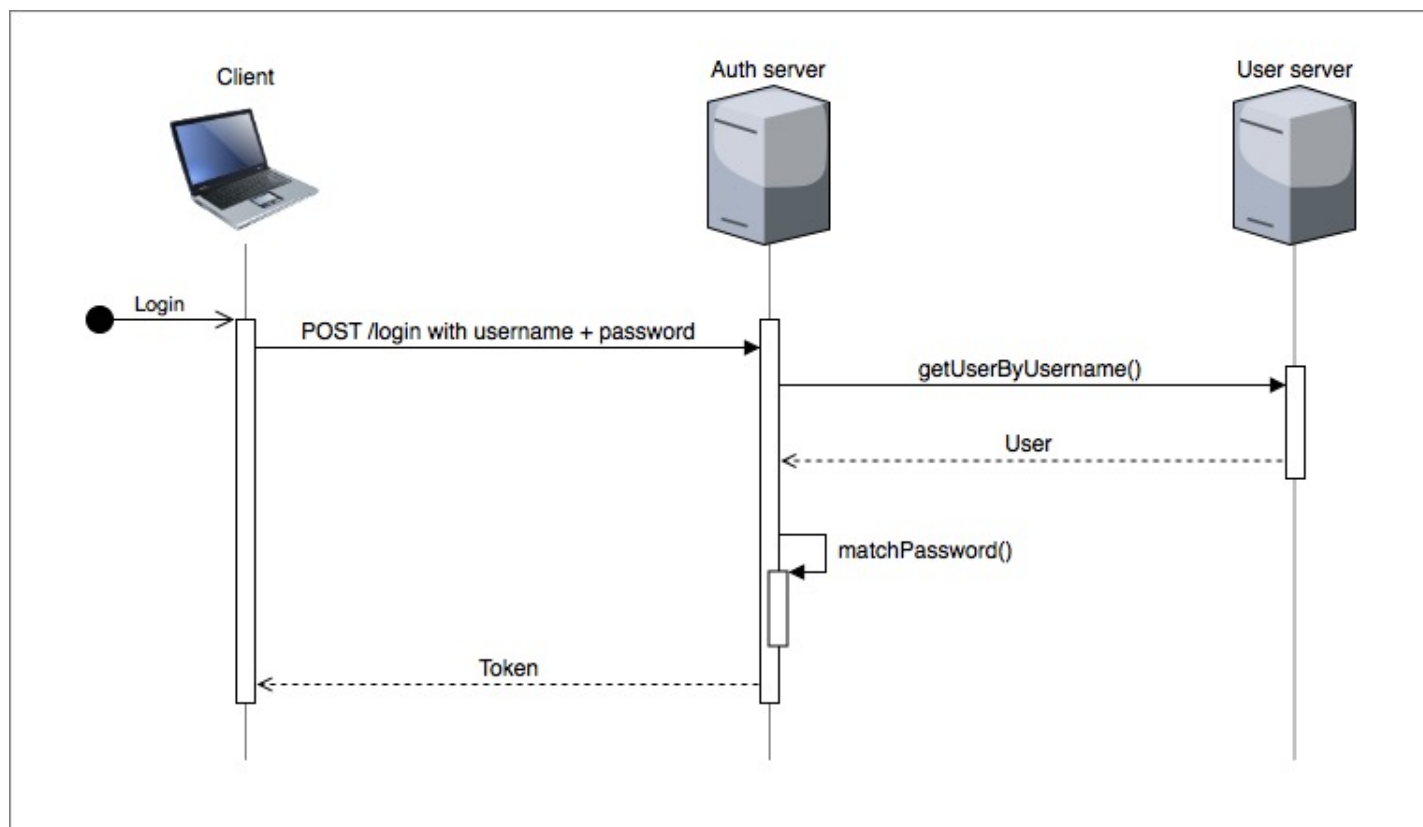
13 July 2018

Estimated reading time 4 minutes

For many years Simple Object Access Protocol (SOAP) was the standard approach for communicating with remote services, often via HTTP. The landscape has changed significantly in recent years with the increase in the adoption of Representational State Transfer (REST) APIs. There are still a number of use cases that suit SOAP, for example where stateful operations are required.

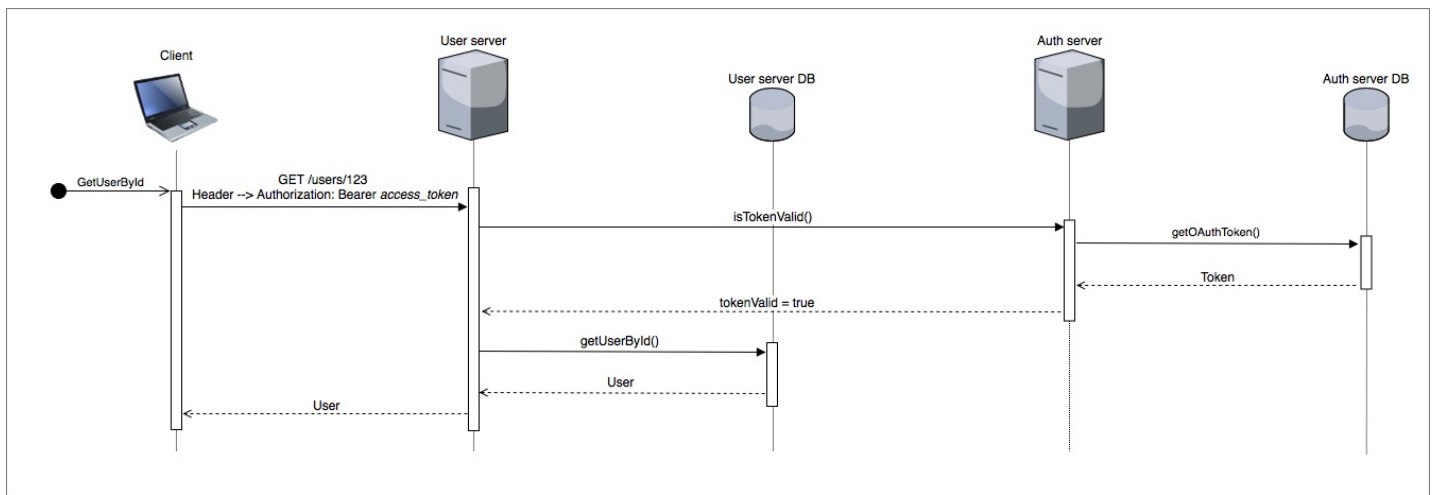
One of the six guiding architectural principles of REST is statelessness. Every API request from the client to a server must contain all the necessary information necessary to serve the request. The server maintains neither state nor context. How do we then authorise access to protected REST APIs? Say hello to OAuth¹⁸ and JSON Web Token (JWT)¹⁹. OAuth and JWT are two of the most widely used token frameworks or standards for authorising access to REST APIs. In this blog post I consider how both OAuth and JWT can be combined to gain performance improvements.

OAuth enables an application to obtain limited access to an HTTP service. While JWT is a compact, URL-safe means of representing claims to be transferred between two parties. OAuth has a number of grant types. So whenever I refer to OAuth in this blog post, I am referring to the OAuth 2.0 Resource Owner Password Credentials Grant type²⁰. A user is required to authenticate or login to obtain a token. A typical authentication flow is shown in the sequence diagram below.

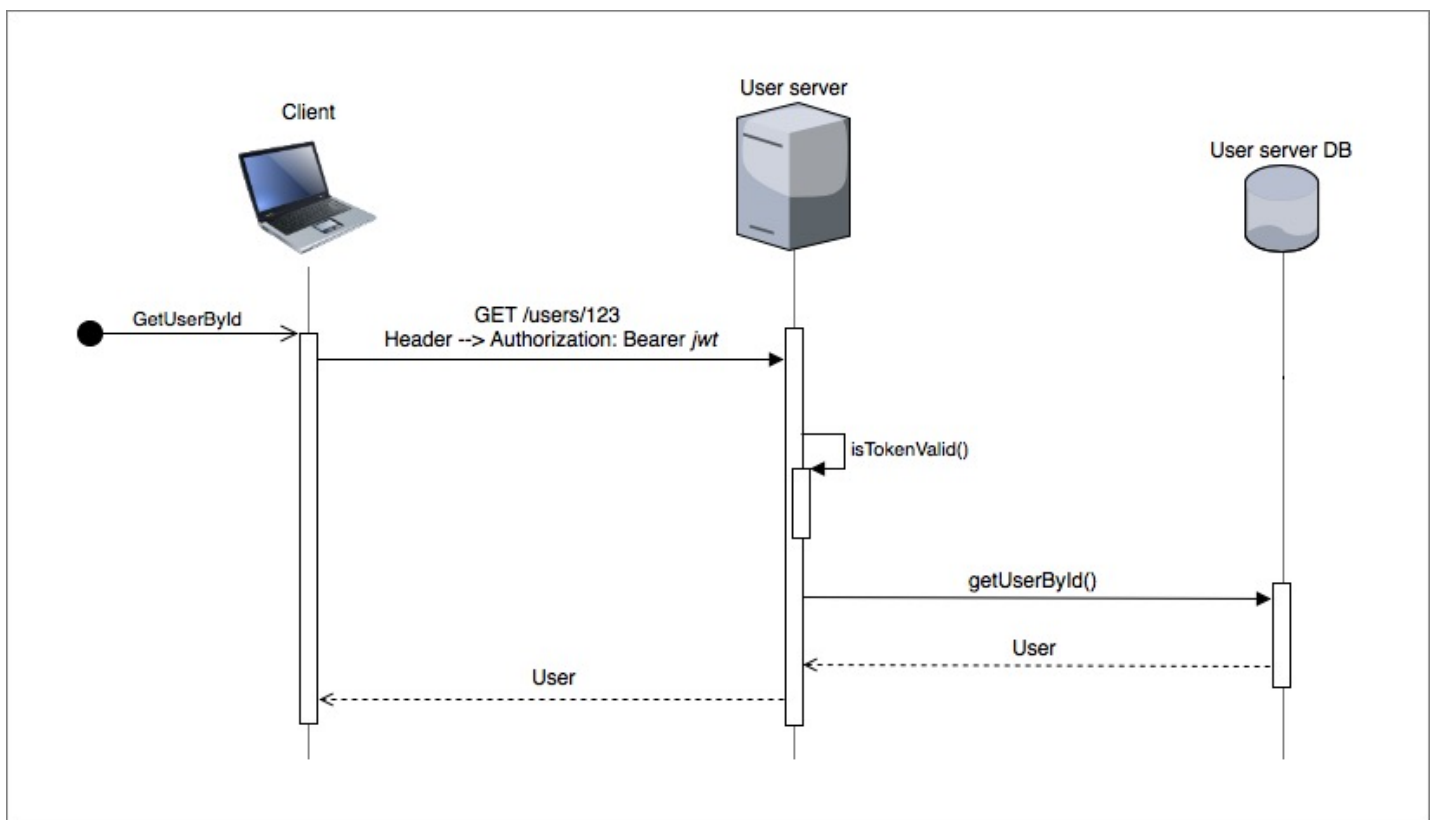


A user will enter their username and password via a client (which could be a mobile device or PC), and at the end of the authentication process the user will be supplied with a token. The client will then include the token with every subsequent API request to a resource server (like the User server). To compare what the authorisation flow for both OAuth and JWT will look like, let's consider an example where we make an API request to *GET the authenticated user*.

The OAuth flow to *GET the authenticated user whose ID is 123* will typically look like the sequence diagram below.



While, the JWT flow to *GET the authenticated user whose ID is 123* will typically look like the sequence diagram below.



The JWT implementation is less chatty and more performant compared to OAuth. This is because JWT enables a resource server to verify the token locally. In its compact form, JWT consist of three parts: the header, payload and signature. The signature is the result of signing the base64Url encoded header and the base64Url encoded payload with a key. The resource server uses the signature to verify that the token has not been tampered with.

The JWT payload contains the claims. The claims are statements about an entity (typically, the user excluding private information of course) and additional metadata like the expiry time. JWT however has a drawback in that once it has been issued it will allow its holder to gain access to a resource server until the expiry time is lapsed. It looks like we need a way to revoke JWTs. So let's go back to OAuth.

OAuth is chattier compared to JWT. This is because OAuth requires the Auth server to verify the validity of the token and the Auth server in turn relies on the information it has stored in a database to make this judgement. OAuth however does have an advantage over JWT in that tokens can be easily revoked. This is particularly a good feature if instant access revocation is desired. The basic OAuth token response is shown below.

```
{
  "access_token": "foo",
  "token_type": "example",
  "expires_in": 3600,
  "refresh_token": "bar"
}
```

The relevant attributes are described in the table below.

Attribute	Description
access_token	The short-lived token that is typically valid for about an hour.
expires_in	The lifetime of the access token in seconds.
refresh_token	The long-lived token which are used to obtain new access tokens. Typically valid for a number of days or months.

As shown in the example OAuth flow above, the client will include the access token in every API request to a resource server. The client will then make use of the refresh token to obtain a new access token. How can we benefit from the inherent performance advantages associated with JWT and the limited access capability provided by OAuth? By issuing an OAuth token with JWT in both access token and refresh tokens as depicted below.

```
{
  "access_token": "aShortLivedJWT",
  "token_type": "example",
  "expires_in": 3600,
  "refresh_token": "aLongLivedJWT"
}
```

The client will include the short-lived JWT for every call to the resource server, and will make use of the long-lived JWT to obtain a new access token. The short-lived JWT is validated locally. We do however need to keep a record or blacklist of the revoked refresh tokens till they expire. This blacklist will be checked only when the client wishes to refresh the OAuth token. A new access token will not be granted if the refresh token is found in the blacklist. The blacklist should ideally contain refresh tokens associated with users who have logged out and users whose account have been disabled. Although the access token is not immediately revoked, it is meant to be a short-lived token.

Finally, there may be scenarios where this behaviour is not desired but ultimately, it depends on the requirements of the system. This approach enables the resource server to validate the OAuth access token locally and only requires interaction with the Auth server when we need to get a new OAuth access token. It is this reduction in the interaction with the Auth server that gives us the performance improvements.

Related Posts

The Best of Capgemini Engineering Blog 2016

25

Looking Forward to the Spring eXchange 2014 with Capgemini Software Engineering

26

Ten Steps towards Cloud Native

27

The Conservation of Complexity in Software

How you can never remove complexity in a system, you can only move it

28

Thoughts on SkillsMatter Meetup: Plugin Architecture

Uncle Bob describes plugin architecture and why it's so powerful

29


Join our Architecture team³⁰


Find out more ³⁰



Tosin Ogunrinde ¹⁰

Tosin is a Solution Architect at Capgemini. He's a technologist who enjoys creating clean, elegant and scalable solutions to otherwise complex problems.

 Follow @tosin_ogunrinde on Twitter ³¹

 Meet Tosin on LinkedIn ³²

 tosin-ogunrinde on GitHub ³³

Posts by Tosin

CQL Statement Builder

³⁴

Opinions expressed on this blog reflect the writer's views and not the position of the Capgemini Group.

All content copyright Capgemini⁷ © 2020 • All rights reserved • Privacy Policy³⁵

Proudly published with Jekyll³⁶

Links

1. <https://capgemini.github.io/architecture/combining-oauth-and-jwt-to-gain-performance-improvements/#main-content>

2. <https://capgemini.github.io/>
3. <https://capgemini.github.io/architecture/combining-oauth-and-jwt-to-gain-performance-improvements/#>
4. <https://capgemini.github.io/presentations>
5. <https://capgemini.github.io/categories>
6. <https://capgemini.github.io/tags>
7. <https://capgemini.github.io/about>
8. <https://capgemini.github.io/authors>
9. <https://capgemini.github.io/feed.xml>
10. <https://capgemini.github.io/authors#author-tosin-ogunrinde>
11. <https://capgemini.github.io/tags/index.html#REST>
12. <https://capgemini.github.io/tags/index.html#Token>
13. <https://capgemini.github.io/tags/index.html#OAuth>
14. <https://capgemini.github.io/tags/index.html#JWT>
15. <https://capgemini.github.io/tags/index.html#Authentication>
16. <https://capgemini.github.io/tags/index.html#Authorisation>
17. <https://capgemini.github.io/tags/index.html#Architecture>
18. <https://tools.ietf.org/html/rfc6749>
19. <https://tools.ietf.org/html/rfc7519>
20. <https://tools.ietf.org/html/rfc6749#page-37>
21. [https://twitter.com/intent/tweet?text="Combining OAuth and JWT to gain performance improvements"%20https://capgemini.github.io/architecture/combining-oauth-and-jwt-to-gain-performance-improvements/%20by%20@tosin_ogunrinde%20via%20@CapgeminiUK](https://twitter.com/intent/tweet?text=)
22. <https://www.facebook.com/sharer/sharer.php?u=https://capgemini.github.io/architecture/combining-oauth-and-jwt-to-gain-performance-improvements/>
23. <https://plus.google.com/share?url=https://capgemini.github.io/architecture/combining-oauth-and-jwt-to-gain-performance-improvements/>
24. <https://www.linkedin.com/shareArticle?mini=true&url=https://capgemini.github.io/architecture/combining-oauth-and-jwt-to-gain-performance-improvements/&title=Combining OAuth and JWT to gain performance improvements>
25. <https://capgemini.github.io/blog/best-of-2016/>
26. <https://capgemini.github.io/development/spring-exchange-2014/>
27. <https://capgemini.github.io/cloud/cloud-native-steps/>
28. <https://capgemini.github.io/architecture/The-Conservation-of-Complexity-in-Software-Architecture/>
29. <https://capgemini.github.io/design/clean-architecture/>
30. https://www.capgemini.com/gb-en/careers/job-search/?search_term=Architecture&utm_source=githubmsblog&utm_medium=referral&utm_campaign=tracking
31. https://twitter.com/tosin_ogunrinde
32. <https://www.linkedin.com/in/tosin-ogunrinde>
33. <https://github.com/tosin-ogunrinde>
34. <https://capgemini.github.io/apache%20cassandra/cql-statement-builder/>

35. <https://www.capgemini.com/privacy-policy>
36. <https://jekyllrb.com/>
37. <http://twitter.com/CapgeminiUK>
38. <http://github.com/capgemini>
39. <http://facebook.com/CapgeminiUK>
40. <https://www.drupal.org/capgemini>
41. <https://capgemini.github.io/architecture/combining-oauth-and-jwt-to-gain-performance-improvements/>