Is “logout” useless on a REST API?

Asked 4 years, 1 month ago Active 2 months ago Viewed 8k times

7

8

Considering that, by definition, a REST API is stateless: is the "logout" operation useless?

I mean, I'm creating a REST API using encrypted JWT. Each token has an expiration time of, let's say, 60 minutes. If I save on a database table the last tokens generated by the API, the "logout" would be done deleting them from the table of valid tokens. But, **if I do that, I understand that the API will cease to be stateless, right?**

So, I understand that I shouldn't do that. The only solution that I'm thinking is make the JWT expiration time shorter, to 5 minutes, don't implement a "logout" operation and just let the tokens expire.

Is this the correct approach?

rest logout jwt api-design

asked Mar 29 '16 at 20:19

Mario S

1,484 ● 16 ● 26

6 Answers

Active Oldest Votes

7

> I mean, I'm creating a REST API using encrypted JWT

The *JSON Web Token (JWT)* tokens encodes all the data about the grant into the token itself. The most important advantage of this approach is that you do not need a backend store for token storage at all. One disadvantage is that you can't easily revoke an access token, so they normally are granted with short expiry and the revocation is handled at the refresh token. Another disadvantage is that the tokens can get quite large if you are storing a lot of user credential information in them. So if:

> If I save on a database table the last tokens generated by the API, the "logout" would be done deleting them from the table of valid tokens

Then you would lose the most important advantage of using JWT and also, still have all those disadvantages, which seems unreasonable to me.

> So, I understand that I shouldn't do that. The only solution that I'm thinking is make the JWT expiration time shorter, to 5 minutes, don't implement a "logout" operation and just let the tokens expire

In my opinion, if you're planning to use JWT, YES! it's better to rely on the token expiration. For more details on this approach you can check this question out.

> Is “logout” useless on a REST API?

Regardless of the fact that you're using JWT and similar to any other decent questions on computer science, the answer would be *It Depends*. The most important advantage of *Statelessness* is that your API would be more scalable. If you choose this path, probably, every request on your API should be authenticated, since you may need to search a backend store for the given token or decode a JWT token. So, in this case you may have some performance cost on a single node but in a big picture, you would still have the scalability. I guess what i'm trying to say is, if you do not need that scalability, you're better off to choose a *Stateful* approach. Otherwise, pure REST principles is the way to go.

edited May 23 '17 at 11:58
Community ♦
1 ● 1

answered Mar 29 '16 at 20:41

Ali Dehghani
34.2k ● 11 ● 123 ● 127

Scalability is the key... Relying on expiration time looks like the right path now. – Mario S Mar 29 '16 at 21:38

1 Then i recommend to read more about best practices of using JWT tokens..the linked question is very informative in this area...have fun – Ali Dehghani Mar 29 '16 at 21:42

3

Automatic token expiry is a separate concern from an explicit "log out" mechanism and, as such, they are *both* perfectly valid actions regardless of whether your API is ReSTful or not.

When a user logs out they are making a conscious decision to invalidate their access token - for example, if they're using a public computer or borrowing someone else's device temporarily.

Automated expiry is used to ensure that the user *must* revalidate, in some fashion, on a regular basis. This is good for server-side security.

Access tokens are not about sharing session state between client and server - it's entirely possible to implement an access token system without shared state and the token itself doesn't implement session state, it's only used to verify that the user is who they claim to be. As such, access tokens are not really anything to do with the statefulness of the API.

answered Mar 29 '16 at 20:28

sisyphus
5,088 ● 1 ● 17 ● 32

1

I would argue that your API is already stateful just by the sheer fact that you have a token around. I also wouldn't get too hung up on REST purity, meaning that everything *has* to be stateless come hell or high water.

I have a similar REST API that I support and I implemented a logout endpoint that is a DELETE call. It simply deletes the token information on the server side and clears any type of authentication for the logged in user.

TL;DR

No, a logout is not useless in a REST API. In fact, for APIs that require authentication, it is more or less a necessity.

answered Mar 29 '16 at 20:24

sma
8,452 ● 4 ● 45 ● 73

1

I think it depends on the behavior that you want for your application, and how secure you need it to be. Do you really need to invalidate the token? For instance, you could just remove your token from your frontend (browser or app). In theory, it is the only place that stores that particular token. If the token is compromised, it will still be valid until it expires, though.

If you really need to invalidate it server side, a common approach would be to create a blacklist with the token, and clear the expired entries from time to time.

But what if you need your application to accept just one token for each user, like in a bank app that you can only be logged in one device at time? For that purpose the blacklist won't do the job, so you will need to store a single token for each user and check if the passed token is the same. At logout, you would just clear that unique entry. Or you may just use sessions.

So, it is not useless, It just depends on your application.

answered Jun 21 '18 at 22:01

Edudjr
915 ● 9 ● 23

0

With a short expiration time on the token I would think for most applications deleting the token from the client on logout would be a good solution. Anything more would rely on the server and no longer be stateless.

answered Feb 17 at 13:10

T

Tri Dave
1 ● 1

-1

You can generate a new token that it already expired i.e. expiration is 1sec. and pass it to the user. Any upcoming request will be invalid. This is not optimal solution though..

answered Jun 30 '17 at 11:30

Adi
1