

[Total Economic Impact of Auth0](#) ▶

## Related Posts

### BLOG



# Cookies vs Tokens. Getting auth right with AngularJS

Using a token-based authentication design over cookie-based

authentication.



Alberto Pose

January 07, 2014

0

0

391

overs the same topic. You can find it here: [Cookies vs Tokens:](#)

## Introduction

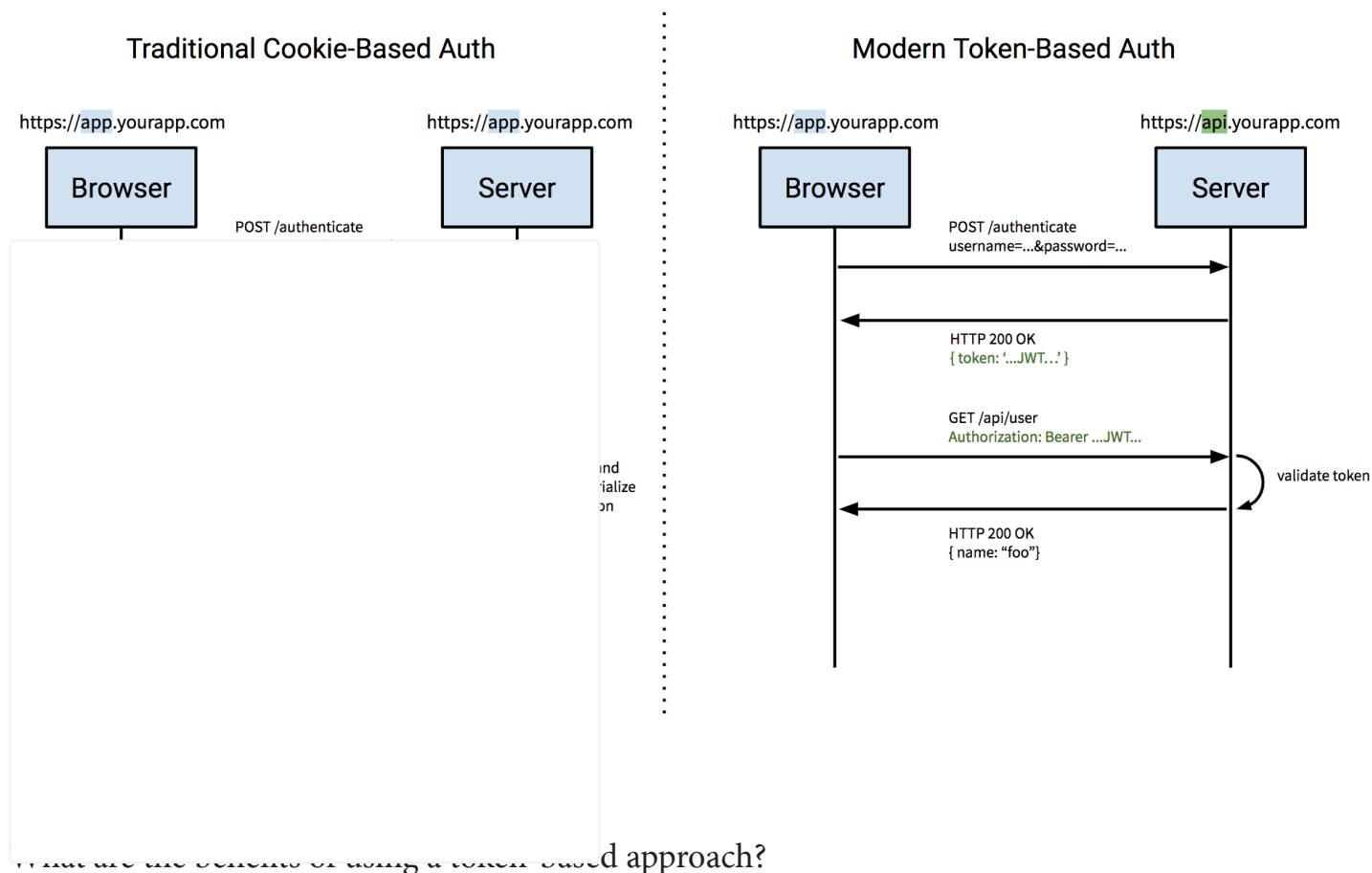
There are basically two different ways of implementing server side authentication for apps with a frontend and an API:

The most adopted one, is **Cookie-Based Authentication** (you can find an example here) that uses server side cookies to authenticate the user on every request.

## Related Posts

## Token based vs. Cookie based

The following diagram explains how both of these methods work.



**Cross-domain / CORS:** cookies + CORS don't play well across different domains. A token-based approach allows you to make AJAX calls to any server, on any domain because you use an HTTP header to transmit the user information.

**Stateless (a.k.a. Server side scalability):** there is no need to keep a session store, the token is a self-contained entity that conveys all the user information. The rest of the state lives in cookies or local storage on the client side.

## Related Posts

**Decoupling:** you are not tied to a particular authentication scheme. The token might be generated anywhere, hence your API can be called from anywhere with a single way of authenticating those calls.

**Mobile ready:** when you start working on a native platform (iOS, Android, Windows 8, etc.) cookies are not ideal when consuming a secure API (you have to deal with cookie containers). Adopting a token-based approach simplifies this a lot.

**CSRF:** since you are not relying on cookies, you don't need to protect against cross site requests (e.g. it would not be possible to `<iframe>` your site, generate a POST request and re-use the existing authentication cookie because there will be none).

hard perf benchmarks here, but a network roundtrip likely to take more time than calculating an HMACSHA256 ints.

are using Protractor to write your functional tests, you r login.

is a standard JSON Web Token (JWT). This is a standard (.NET, Ruby, Java, Python, PHP) and companies se, Google, Microsoft). As an example, Firebase allows on mechanism, as long as you generate a JWT with

Subscribe to more awesome content!

your@email.com

certain pre-defined proper

USE AUTH0 FOR FREE


shared secret to call their API.

What's JSON Web Token? **JSON Web Token (JWT, pronounced *jot*)** is a relatively new token format used in space-constrained environments such as HTTP Authorization headers. JWT is architected as a method for transferring security *claims based* between parties.

**AngularJS Authentication Screencast Series - Part 1**

new token format used in space-constrained environments such as HTTP Authorization headers. JWT is architected as a method for transferring security *claims based* between parties.

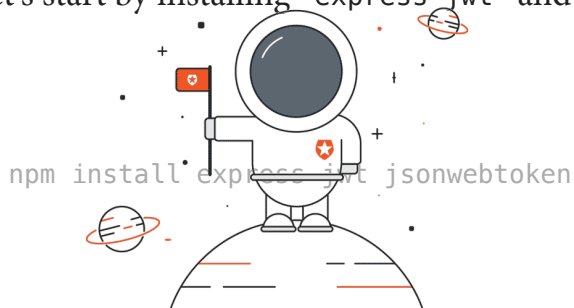
Using JSON Web Tokens as

 Ryan Chenkie

Related Posts Assuming you have a node.js app, below you can find the components of this architecture.

## Server Side in the Auth0 Ambassador Program

Let's start by installing `express-jwt` and `jsonwebtoken` :



Configure the express middleware to protect every call to `/api` .

[Learn More ►](#)

```

    -jwt');
    ;

    routes with JWT
    t: secret}));
  
```

The angular app will perform a POST through AJAX with the user's credentials:

```

app.post('/authenticate', function (req, res) {
  //TODO validate req.body.username and req.body.password
  //if is invalid, return 401
  if (!(req.body.username === 'john.doe' && req.body.password === 'foobar')) {
    res.send(401, 'Wrong user or password');
    return;
  }
}
  
```

---

## Related Posts

```
first_name: 'John',  
last_name: 'Doe',  
email: 'john@doe.com',  
id: 123  
};
```

```
// We are sending the profile inside the token  
var token = jwt.sign(profile, secret, { expiresInMinutes: 60*5 });  
  
res.json({ token: token });  
});
```

ed is straight forward. Notice that the credentials check are.

```
req, res) {  
  l + ' is calling /api/restricted');
```

## Angular Side

The first step on the client side using AngularJS is to retrieve the JWT Token. In order to do that we will need user credentials. We will start by creating a view with a form where the user can input its username and password.

---

## Related Posts

```
<form ng-submit="submit()">
  <input ng-model="user.username" type="text" name="user" placeholder="Username" />
  <input ng-model="user.password" type="password" name="pass" placeholder="Password" />
  <input type="submit" value="Login" />
</form>
</div>
```

And a controller where to handle the submit action:

```
($scope, $http, $window) {
  , password: 'foobar'};

  ser)
  , headers, config) {
    = data.token;

    headers, config) {
      r fails to log in

delete $window.sessionStorage.token;

// Handle login errors here
$scope.message = 'Error: Invalid user or password';
});
};
});
```

---

## Related Posts

header we are going to use `Bearer <token>` .

`sessionStorage` : Although is not supported in all browsers (you can use a [polyfill](#)) is a good idea to use it instead of cookies ( `$cookies` , `$cookieStore` ) and

`localStorage` : The data persisted there lives until the browser tab is closed.

```
myApp.factory('authInterceptor', function ($rootScope, $q, $window) {
  return {
    || {}];
    n) {
    = 'Bearer ' + $window.sessionStorage.token;

    user is not authenticated

    onse);

  };
});

myApp.config(function ($httpProvider) {
  $httpProvider.interceptors.push('authInterceptor');
});
```

After that, we can send a request to a restricted resource:

---

### Related Posts

```
console.log(data.name); // Should log 'foo'
});
```

The server logged to the console:

```
user foo@bar.com is calling /api/restricted
```

10 Things you should know about Tokens | AngularJS seed app.



10 Things you should know about Tokens | AngularJS seed app.

Token based authentication in realtime frameworks like Socket.io

10 Things you should know about Tokens

## Bottom Line

When building Single Page Applications, consider using a token-based authentication design over cookie-based authentication. Leave a comment or discuss on HN. Learn more about AngularJS




## Related Posts

## Aside: Securing AngularJS Apps with Auth0

Securing applications with Auth0 is very easy and brings a lot of great features to the table. With Auth0, we only have to write a few lines of code to get solid identity management solution, single sign-on, support for social identity providers (like Facebook, GitHub, Twitter, etc.), and support for enterprise identity providers (Active Directory, LDAP, SAML, custom, etc.).

To learn about how easy it is to secure AngularJS applications with Auth0, this guide walks you through setting up authentication and authorization.



Authentication that just works.  
Any device. Hosted anywhere.  
GET AUTH0 FOR FREE

 Arefe ▾ Recommend 3  Share

Sort by Best ▾



Join the discussion...

**Michael Kariv** • a year ago

What is the right way of implementing blacklist for users if I use JWT and passport? What should happen on the server to invalidate a token and block the user from even logging in again? I am prepared to change JWT to any other kind of bearer token, but blacklist is a must have for me. Any advice? Anybody?

17 ^ | ▾ • Reply • Share ›

## Related Posts

process.

After your server has successfully verified the JWT, you make sure it matches to something valid in your cache, or you drop it from the client's storage.

So blacklisting your JWT will come down to deleting or tagging their corresponding session's data in the server cache. Up to you.

If you use hapijs (<https://hapijs.com>) instead of express, there is a hapi-auth-jwt2 plugin (<https://www.npmjs.com/package/hapi-auth-jwt2>) that is configured with a validateFunc function to do just that.

^ | v • Reply • Share ›

**Kim Maida** Mod ➔ **Michael Kariv** • a year ago

JWT with Passport does not directly support blacklisting or whitelisting users because you're not storing tokens in a database. Instead, it's recommended to keep the lifetime of intended to expire rather than to be blacklisted. However, questions for alternatives to invalidate JWTs prior to expiration:

year ago

lifetime ?

implement remember me strategy, right ?

re ›

code • 5 months ago

re the OpenID Connect / OAuth 2 Authorization Server even if the token life times are short. You would have to  
ent renew to get new token while the user is active  
[cod.com/2017/....](#) Since the Open ID Connect Provider /

OAuth2 Authorization Server can use a marker cookie to remember the user, the user could be logged in automatically from verified clients if that's supported.

^ | v • Reply • Share ›

adobot Mod ➔ ironcode • a year ago

The best answer here is that it depends on your particular use case. If you are building an app that will store sensitive data - then you probably want a short lifetime. Most banks in the US for example will log you out after 5 to 10 minutes of inactivity.

If your app does not store sensitive data - there a long-lived token that lasts days to weeks or even months may be a good strategy.

## Related Posts

**Dilame** → adobot • 3 months ago

Ok, i am not a bank, and i want long lifetime of token. And we return to the question asked by **@Michael Kariv**

"What is the right way of implementing blacklist for users if I use JWT and passport? "

^ | v • Reply • Share ›

**Bruno S. Krebs** Mod → Dilame • 3 months ago

Hi there, have you read our article about this? <https://auth0.com/blog/blac...>

^ | v • Reply • Share ›

**David Sun** • 5 months ago

Nice article on how to use JWT. But the title "Cookies vs Tokens" is conceptually misleading.

Cookie is a mechanism enabling browser clients to store and exchange small data. It's a SessionId, JWT token or some other value. JWT is a token. There is a non-mutually exclusive relationship between it and Cookie. You can exchange JWT between client and server using Cookies, and use Cookie as a mechanism for the token on the client side. Of course, this is not a silver bullet. It has its own security caveats and counter-measures (google 'csrf' or 'sessionStorage' immune to security concern. IMO the title "Session vs Tokens" is more appropriate.

• 5 months ago

That's what you are saying and I agree with most parts. The thing is that it's the same problem with JWTs or with Cookies. Hence the

**ironcode** • a year ago

Thanks for your post, I hoped you would go deeper. The main issue with JWT is: how do you really validate a JWT ?

I am not talking about checking the signature and so on, it is done with `jwt.verify(...)`.

How do you make sure that the payload is still valid ? What I call payload here is your profile object. You may generate a JWT today, with an admin claim set to true, but tomorrow, the guy is not admin anymore, how do you invalidate his JWT ?

1 ^ | v • Reply • Share ›

**adobot** Mod → ironcode • a year ago

Hi ironcode,

---

**Related Posts** minutes of inactivity. You can use refresh tokens to get a new token as well which would have the updated rules.

---

There is always the option of going stateful and making a call against your database to verify the payload matches the backend data.

These are just some things to consider - but really it all comes down to what will satisfy your requirements in the best way possible.

^ | v • Reply • Share ›

**ironcode** ➔ adobot • a year ago

Hi adobot,

Thanks again for your post. I just searched for "refresh" here:

<https://www.npmjs.com/packa...>, but I found nothing. Please can you point me to a doc explaining how to create and use refresh tokens with jwt ? or maybe there are

ons for nodejs.

›

oncode • a year ago

[auth0.com/docs/toke...](https://auth0.com/docs/toke...) :)

• Share ›

ot • a year ago

ve it a try, but my actual option is stateful JWT, with stored on server cache (Redis or Mongo).

• Share ›

ode • 5 months ago

scale then every single node will have to lookup the session

^ | v • Reply • Share ›

**Alex .B** ➔ ironcode • 8 months ago

You can simply have the userId in you payload and call your database to retrieve the user on request. Retrieving a row in a db with an index would have a negligible performance impact.

^ | v • Reply • Share ›

**adobot** Mod ➔ ironcode • a year ago

Sure thing. In many instances it does make sense to have stateful JWT so if your use case calls for it - feel free to do it that way

^ | v • Reply • Share ›

## Related Posts

<https://www.npmjs.com/package...>

^ | v • Reply • Share ›

**Danny Suarez** • a year ago

Thank you very much for this, It helped me a lot, what color syntax do you use for snippet? pretty cool!!

1 ^ | v • Reply • Share ›

**Ribeiro** • a year ago

The cons: Token revocation and replay attack (both solutions make JWT stateful).

1 ^ | v • Reply • Share ›

**Joe Kolba** • 2 years ago

It is in your best interest to NOT save the JWT token in session storage. It is always a better idea to use HttpSession cookies and send requests using credentials. You leave yourself open to XSS

ens.

rs ago

self open to XSS attacks"... I mean, come on, you leave  
oy not escape inputs. And this is an old problem with your  
where to store tokens.

s • 2 years ago

XSS as possibility of browser malware getting access to your  
does a little beyond your own code I guess.

›

I am wondering how JWT improves performance by avoiding finding a session on database. I assume once a user logs in, they get a JWT token and they sends the token in http header. But I think validating the JWT Token without checking database or at least some server cache won't work.

1. if someone knows your secret used during generation of JWT token, he can do anything, so I don't think it's fully trust the data in JWT. (eg: payload may contain user id or role, but we should not trust it until we verify from database.)
2. The user state may be changed. Eg: he used to be an admin but now he is not.

To solve these issues, you have to access database, so using JWT does not help.

The original wording in you article:

"Performance is not something you have to sacrifice here, but a network roundtrip

Related Posts ▾ • Reply • Share ▾

**Bruno S. Krebs** Mod ➔ Jiyuan Zheng • 3 months ago

Well, knowing the private key that generates token allows them to generate whatever token they want. But this is pretty much the same of having the password to the server, or anything like that. You do have to be very careful on how you protect your keys (passwords or keys to generate tokens).

If you are afraid on how to secure your keys, perhaps you can subscribe to Auth0. We hold our customers key well protected.

Besides that, the user indeed may be an admin and then loose some privileges. To solve that, you generate token (jwts) that won't be valid for that long (just a couple of hours). Then, when generatiing a new one, you can check if the user has lost their privileges or not.

^ | ▾ • Reply • Share ▾

3 months ago

short-lived token?

,

Mod ➔ Dilame • 3 months ago

you an answer in the other question (where I talk shortly : store) that applies in this particular case as well... Hope

• Share ▾

ef logins and gets the new token. At this time, both I and he

Later, I change my password, at this time, the thief can still play with the server with his old token?

The question is, how the server can kicks old token out?

^ | ▾ • Reply • Share ▾

**Sebastian Peyrott** Mod ➔ Justin CAO • 9 months ago

You can revoke tokens at any time, either by blacklisting the stolen token (which would make that single token invalid), or by changing the signing key (which would invalidate all tokens for all users).

^ | ▾ • Reply • Share ▾

---

Related Posts

**Bruno S. Krebs** Mod ➔ Dilame • 3 months ago

You just have to be creative. For instance, you could have a key/value store that holds an object with two properties: ids from users that recently changed their password; and the date that this event occurred.

Then, for each request, you would take the token and check its "iat" ([claim that means issued at and contains a numeric date](#)) property to see if you should deny it.

^ | v • Reply • Share ›

**Ole Ersoy** ➔ Sebastian Peyrott • 5 months ago

You usually cannot blacklist the token unless you want to implement the Authorization Code flow where the client validates the token on every request ... somewhat defeating the purpose of a JWT ...

Sebastian Peyrott • 9 months ago

the server knows which is the old token?  
I think it's the best idea when bother my all users.

on express version?

a year ago

you using? This current implementation should work with  
recent releases since express hasn't had any major changes in a while.

Thanks,

Ado

^ | v • Reply • Share ›

**Chathu Vishwajith** ➔ adobot • a year ago

I'm using Express 4.x

...

```
app.use(express.json());  
app.use(express.urlencoded());
```

...

## Related Posts

[^](#) | [v](#) • [Reply](#) • [Share](#) ›**Nighty** ➔ Chathu Vishwajith • a year ago

download the body-parser middleware from the npm repo.

```
npm install --save body-parser
```

Then do this:

```
var bodyParser = require('body-parser')
```

```
...
```

```
app.use(bodyParser.json())
```

```
app.use(bodyParser.urlencoded())
```

[^](#) | [v](#) • [Reply](#) • [Share](#) ›**dzdengineer** • a year ago

"token-based approach" is pretty cool.

the Interceptor part!!

[✉ Subscribe](#) [D Add Disqus to your site](#) [Add Disqus](#) [Add](#) [🔒 Privacy](#)



---

## Related Posts

---



PRODUCT

Pricing

---

Why Auth0

Related Posts

COMPANY

About Us

Blog

Jobs

Press

SECURITY

Availability & Trust



EXTEND

Lock

WordPress

API Explorer

CONTACT

Related Posts

[Support Center](#)

Follow @auth0

12.8K followers

Like 14K

[Privacy Policy](#) [Terms of Service](#) © 2013-2016 Auth0 ® Inc. All Rights Reserved.

