## 

## Why?

JSON Web Tokens (JWTs) make it easy to ***send read-only signed*** "***claims***" between services (both internal and external to your app/site). Claims are any bits of data that you want someone else to be able to read and/or verify but **not alter**.

**Note**: If that sounds buzz-wordy, don't worry, it will all become clear in the next 5 mins of reading!

## What?

"**JSON Web Token** (JWT) is a compact **URL-safe** means of representing claims to be transferred between two parties. The claims in a JWT are **encoded** as a **JSON object** that is **digitally signed** using JSON Web Signature (JWS). ~ IETF

### In English

To identify/authenticate people in your (web/mobile) app, put a **standards-based token** in the **header** or **url** of the page (or API endpoint) which proves the user has logged in and is allowed to access the desired content.

example: https://www.yoursite.com/private-content/?token=eyJ0eXAiOiJKV1Qi.eyJrZXkiOi.eUiabuiKv

**Note**: if this does not look "secure" to you, scroll down to the "[**security**](https://github.com/dwyl/learn-json-web-tokens#q-if-i-put-the-jwt-in-the-url-or-header-is-it-secure)" section.

### What does a JWT Look Like?

Tokens are a string of "url safe" characters which encode information. Tokens have **three components** (separated by periods) (shown here on multiple lines for readability but used as a single string of text)

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9 // header

.eyJrZXkiOiJ2YWwiLCJpYXQiOjE0MjI2MDU0NDV9 // payload

.eUiabuiKv-8PYk2AkGY4Fb5KMZeorYBLw261JPQD5lM // signature

#### 1. Header

The first part of a JWT is an encoded string representation of a simple JavaScript object which describes the token along with the hashing algorithm used.

#### 2. Payload

The second part of the JWT forms the core of the token. Payload length is proportional to the amount of data you store in the JWT. General rule of thumb is: store the bare minimum in the JWT.

#### 3. Signature

The third, and final, part of the JWT is a signature generated based on the header (part one) and the body (part two) and will be used to verify that the JWT is valid.

## Helper Methods

All the helper methods are kept in **/example/lib/helpers.js** The two most interesting/relevant methods are (simplified versions show here):

// generate the JWT

function generateToken(req){

var token = jwt.sign({

auth: 'magic',

agent: req.headers['user-agent'],

exp: Math.floor(new Date().getTime()/1000) + 7\*24\*60\*60; // Note: in seconds!

}, secret); // secret is defined in the environment variable JWT\_SECRET

return token;

}

Which **generates** our JWT token when the user authenticates (this is then sent back to the client in the **Authorization** header for use in subsequent requests),and

// validate the token supplied in request header

function validate(req, res) {

var token = req.headers.authorization;

try {

var decoded = jwt.verify(token, secret);

} catch (e) {

return authFail(res);

}

if(!decoded || decoded.auth !== 'magic') {

return authFail(res);

} else {

return privado(res, token);

}

}

Which **checks the JWT supplied by the client is valid**, shows private ("privado") content to the requestor if valid and renders the **authFail** **error** page if its not.

**Note**: Yes, both these methods are **synchronous**. But, given that neither of these methods require any **I/O** or **Network**requests, its pretty safe to compute them synchronously.

### FAQ:

### Q: If I put the JWT in the URL or Header is it secure?

Good question! The quick **answer** is: **No**. Unless you are using SSL/TLS (http**s** in your url) to encrypt the connection, sending the Token [**in-the-clear**](http://en.wikipedia.org/wiki/Plaintext) is always going to be insecure (the token can be intercepted and re-used by a bad person...). A naive"mitigation" is to add verifiable "claims" to the token such as checking that the request came from the **same browser** (user-agent), **IP address** or more advanced The solution is to either:

* use one-time-use (single use) tokens (which expire after the link has been clicked) **or**
* Don't use url-tokens where high degree of security is required. (e.g: don't send someone a link which allows them to perform a transaction)

**Use-cases** for a JWT token in a url are:

* account verification - when you email a person a link after they register on your site. https://yoursite.co/account/verify?token=jwt.goes.here
* password re-set - ensures that the person re-setting the password has access to the email belonging to the account.https://yoursite.co/account/reset-password?token=jwt.goes.here

Both of these are good candidates for single-use tokens (which expire after they have been clicked).