# Death To Cookies*!

## Long Live Tokens!

* My boss wrote the title (but not the content) of this talk. I'm not in favor of executing cookies.

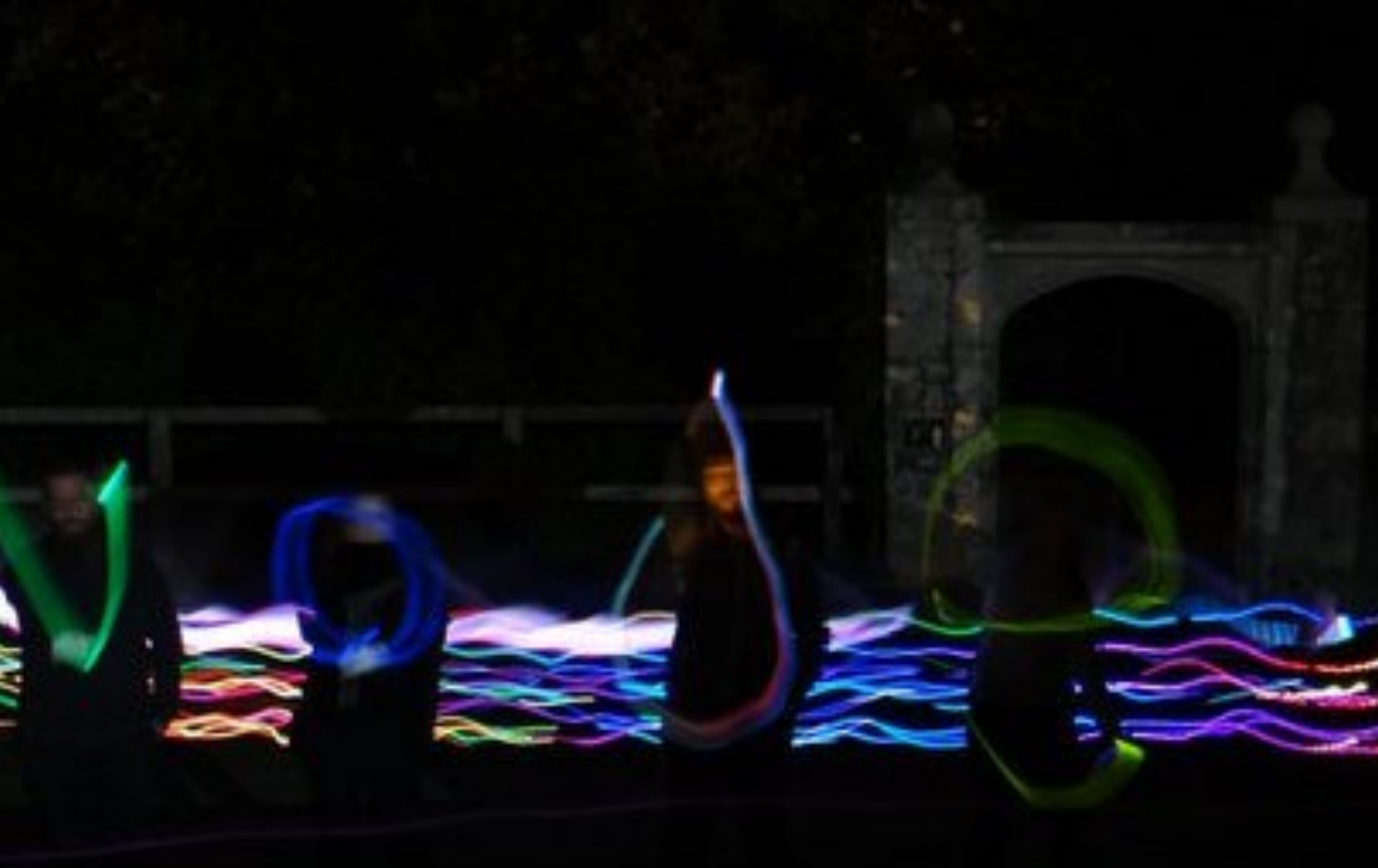
# Hi! I'm Kas

→ Developer Evangelist at **Auth0**

→ Austinite

→ Catparent

→ Builder



**But today, I'm here to talk about cookies and tokens!**

# What we'll cover

→ Why not cookies?

→ Wait, what are JSON Web Tokens?

→ Why tokens?

→ How do I use tokens?

→ What can go wrong?

# Why not cookies?

**The web is a diverse, wonderful cacophony of things:**

→ Regular web sites

→ Single-Page web applications

→ IoT devices

→ Hybrid mobile applications

**Cookies were designed with only one of these in mind!**

(Not due to lack of foresight, just a lack of a crystal ball!)

We need a more **flexible** solution for a more **flexible** definition of what is a web-enabled application

# Case 1a: Single-Page application

→ I need LOTS of data from the server!

→ I get...a session ID that I need to go get more data for.

# Case 1b: Single-Page Application

→ I need to make a call to this authenticated route on my server!

→ I have to wait for database calls to execute…

# Case 2: IoT

→ Um...how do I store cookies on an Arduino? A Particle Photon?

→ Also, do I really want to making more http calls than I need here?

# Case 3a: Microservices

→ I need authentication for ALL of these domains with one login!

→ Cookies only work for that subdomain!

# Case 3b: Microservices

→ Okay, I'll just pass the cookie around!

→ How? And will that even work? How safe is that?

Case 4: Third-party SSO

→ my cookie works for my service…

→ …but I have to go get the token (wait.)
for other services.

# Okay, Okay, we get it.

Cookies work **great for traditional client-server web apps**, but tend to **not be the right tool for the job outside of that.**

# But what are JSON Web Tokens?

# JSON Web Tokens

→ RFC 7519

→ Consist of a header, payload, and key

→ header and payload are made up of claims

→ key is a signed aggregate of the header and payload

→ signed cryptographically with a secret

# So let's make a JSON Web Token!

# Okay cool. But what's a Bearer Token?

→ Think of it like a passport

→ It's got your data on it

→ And it's signed by an authority that's recognized globally!

→ Goes in the Authorization header:
Bearer <token>

# Okay. But why tokens?

# Portable (between platforms)

→ If you can parse/stringify JSON, and sign with sha256, **you can use JWTs**!

→ JSON is a pretty universal data format

→ Plenty of libraries out there if you don't wanna code that yourself.

# Portable (between services and domains)

→ JWTs do not care about subdomains! As long as everyone shares the secret, you're good!

→ You can use them for SSO or just authenticating across several microservices

# Standardized

→ RFC 7519

# Wait. Hold On.

# What about encryption?

# RFC 7516-- JSON Web Encryption

→ allows you to encrypt your JSON Web Token data!

→ Protects your sensitive information during transit

→ (though I probably would just keep these out of the client-side!)

# But how do I use JWTs?

# Sending a JWT for Authentication

→ Your Authorization header should contain:

```
Authorization: Bearer <token>
```

You can also send the token as a query or body parameter in some cases, but this isn't recommended unless required by a third party.

# Using JWTs for authentication on a server.**

** Meaning live code time!

# But what are some possible pitfalls while using JWTs?

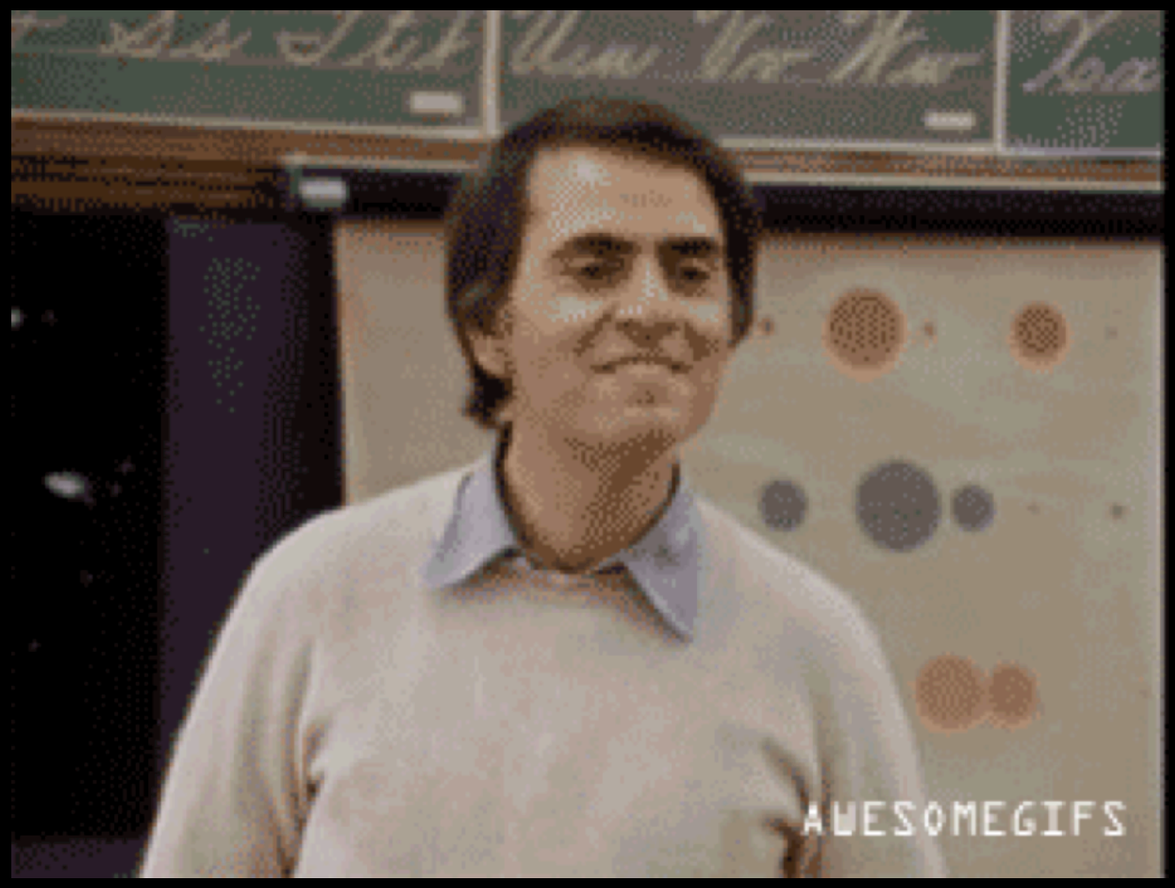
# Possible pitfalls

→ Keep tokens small

→ Keep tokens safe

→ If you can't keep them safe, keep sensitive info out!

# Thanks for listening!




@nodebotanist
kassandra.perch@auth0.com