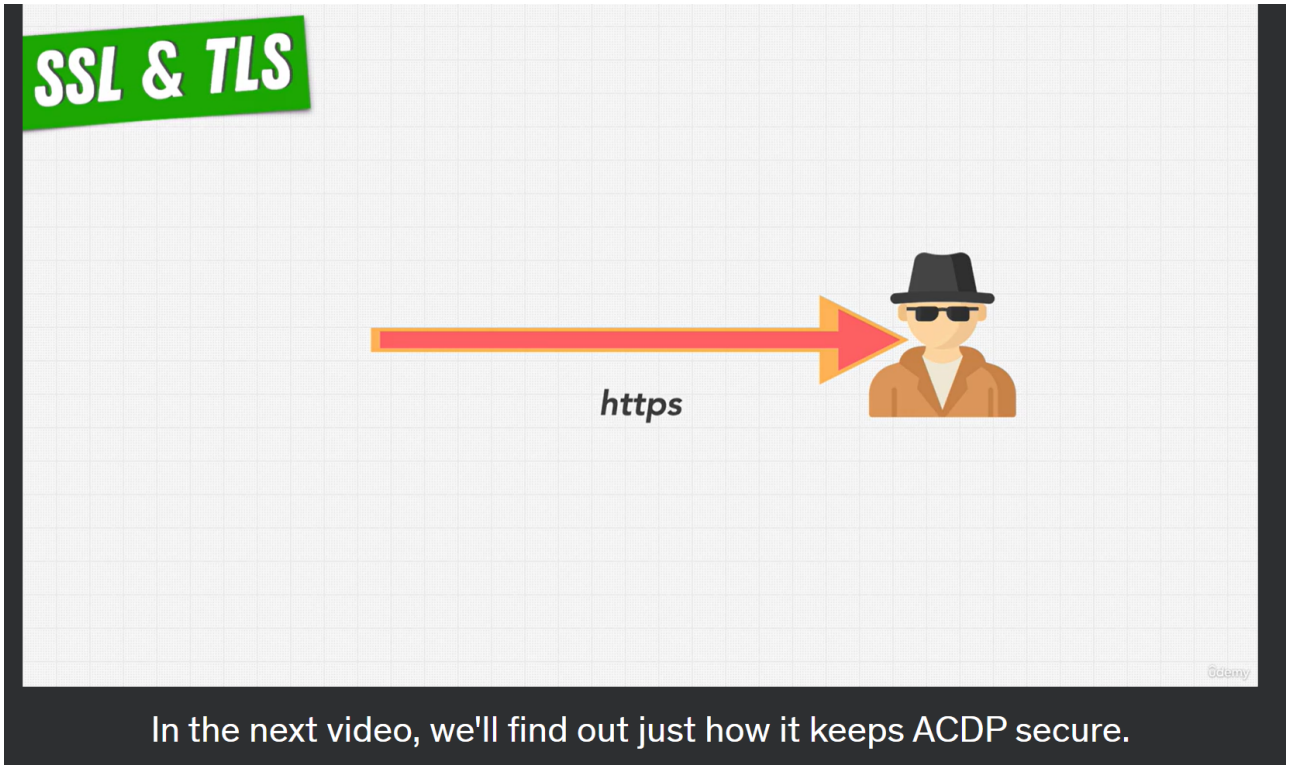
Fundamentals Node Security + Authentication

1. tls last version of the ssl

2. wrapping request tls encrypting



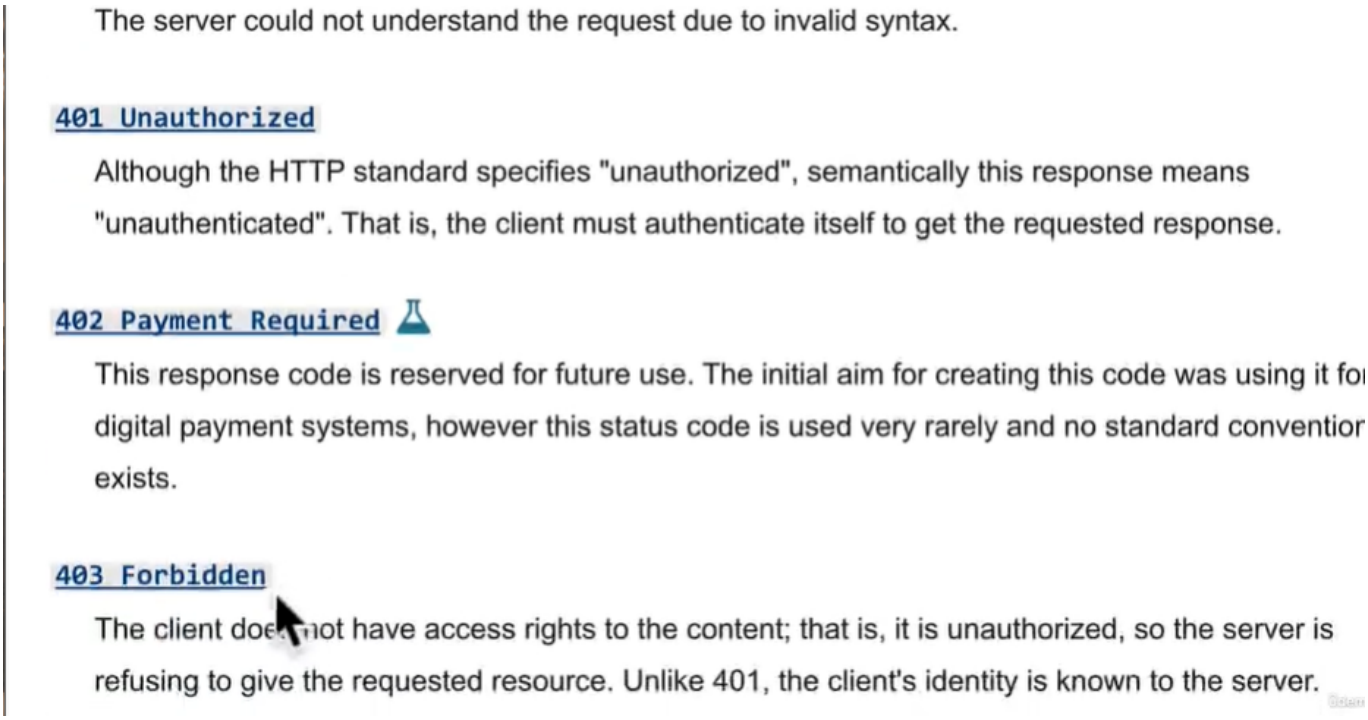
3. If need created ssl or tls certificate

Authentication

when need now who this user

Authorization

checks whether that user has permission to access a specific resource once they've been authenticated access control



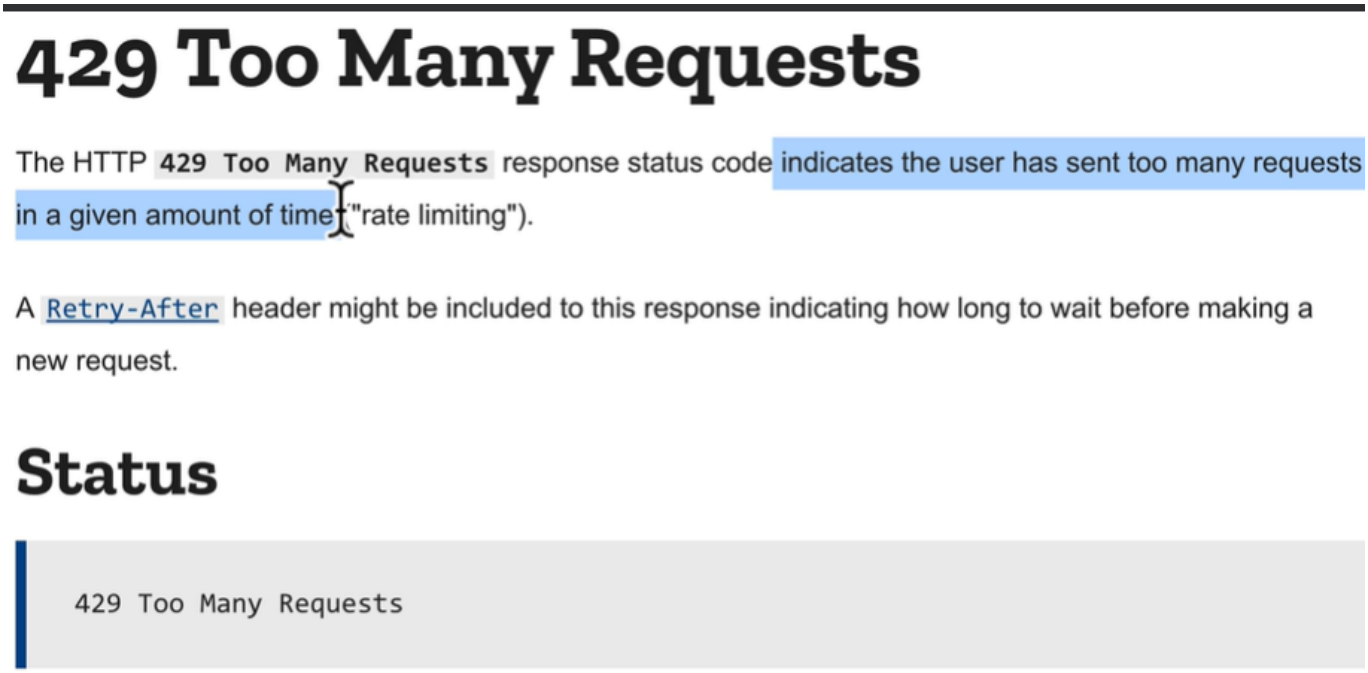
404 permission 401 authenticate

Api key

It's a string

passing as either a query parameter or as a header in http request

---



goole maps

Adding the API key to your request

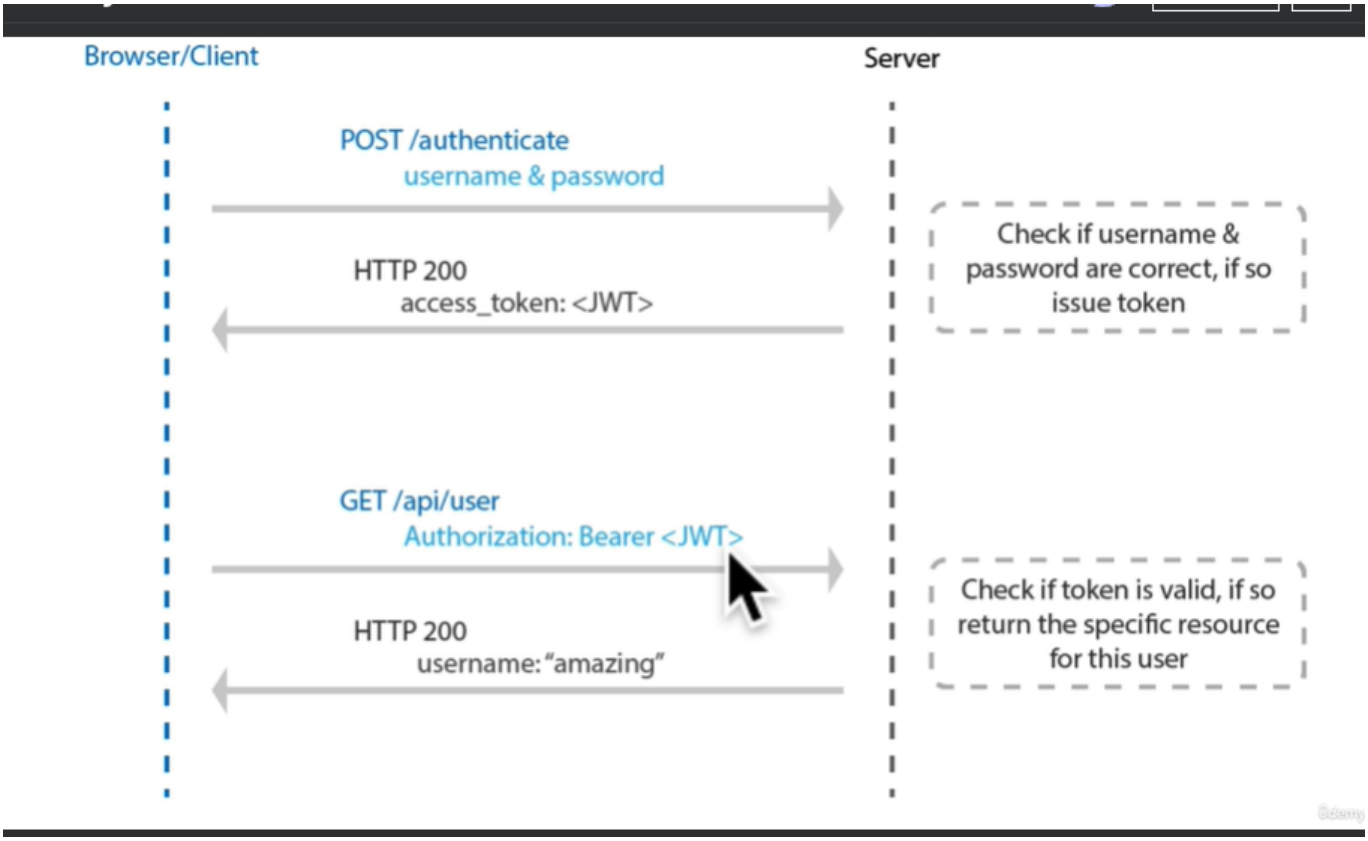
You must include an API key with every Maps JavaScript API request. In the following example, replace `YOUR_API_KEY` with your API key.

```
<script async defer src="https://maps.googleapis.com/maps/api/?key=YOUR_API_KEY&callback=initM
type="text/javascript"></script>
```

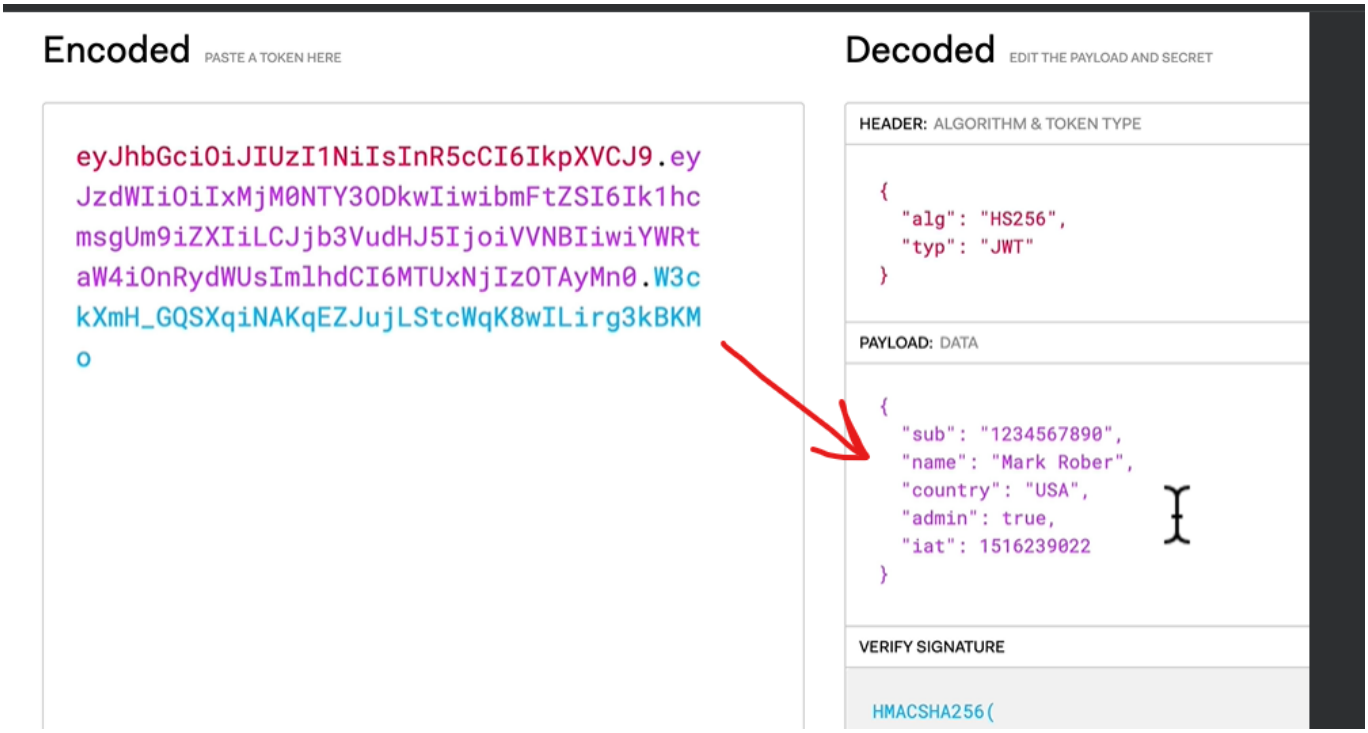
HTTPS is required for requests that use an API key, and recommended for requests that use a client ID. HTTPS is also required for applications that include sensitive user data - such as a user's location - in requests.

JSON Web token

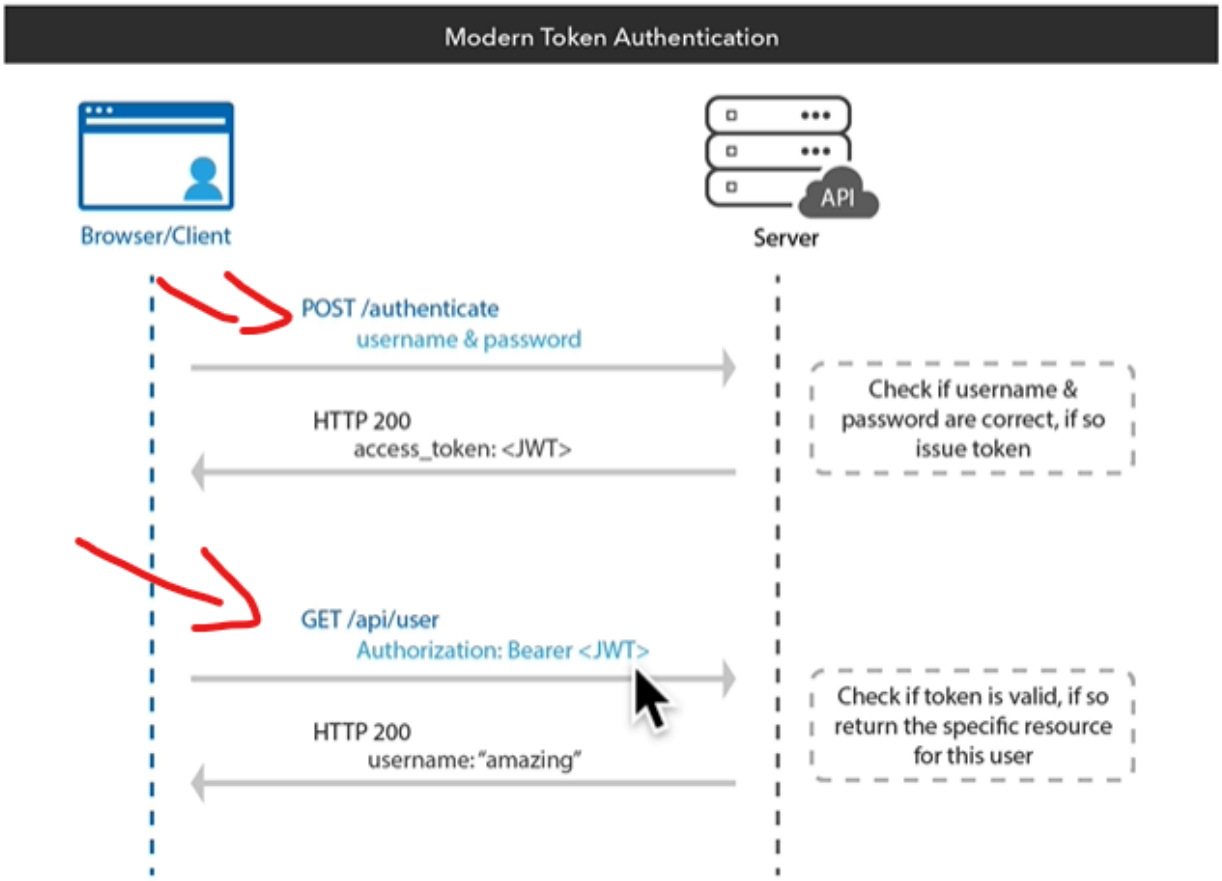
two types JWT || Opaque tokes



JWT encoded information



split to three part header -> alg (HS256) and type payload -> signature ->

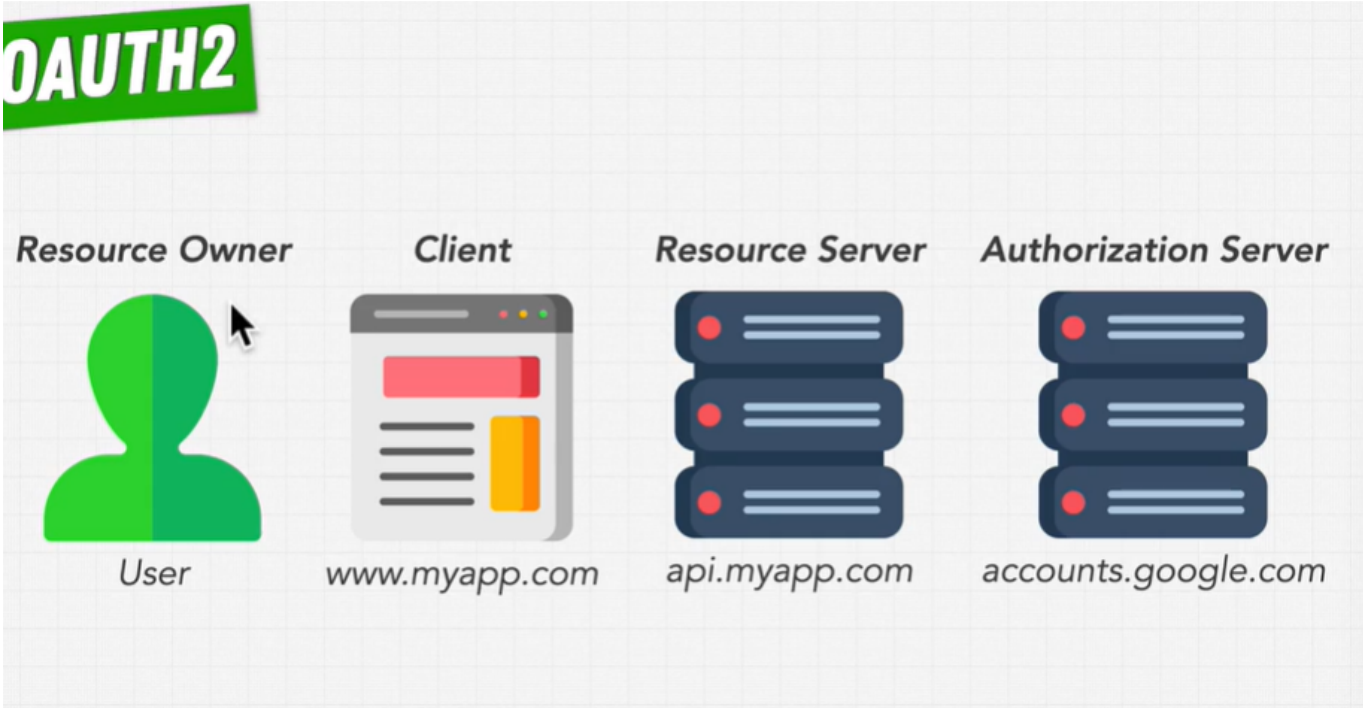


---

data that's constantly being passed around from client to the server.

The OAUTH 2.0 authentication standard

resource owner -> user client -> my website resource server -> your backend in your application
Authorization server -> goole service

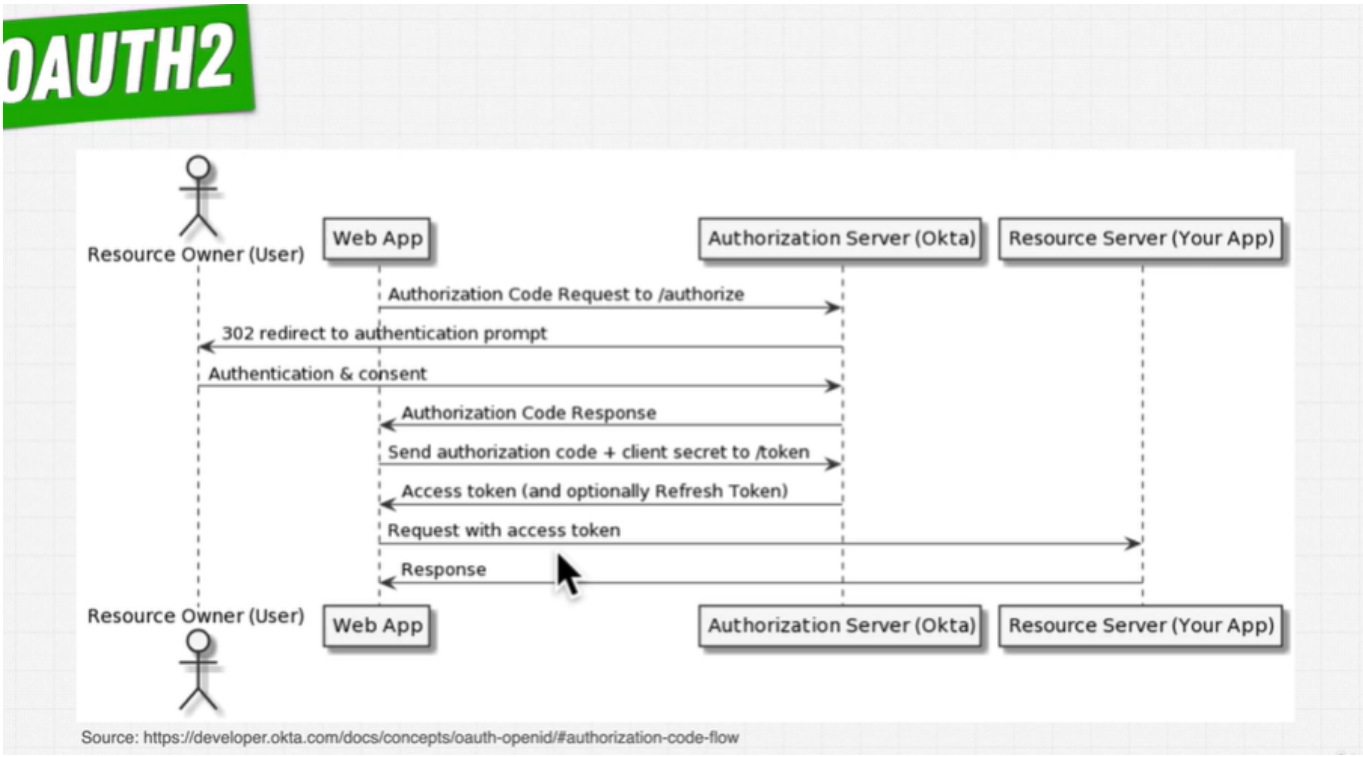


1. has difference flows

The table shows you which OAuth 2.0 flow to use for the type of application that you are building.

Type of Application	OAuth 2.0 flow
Server-side (AKA Web)	Authorization Code flow
Single-Page Application	Authorization Code flow with PKCE or Implicit flow when the SPA that you are building runs in older browsers that don't support Web Crypto for PKCE
Native	Authorization Code flow with PKCE
Trusted	Resource Owner Password flow

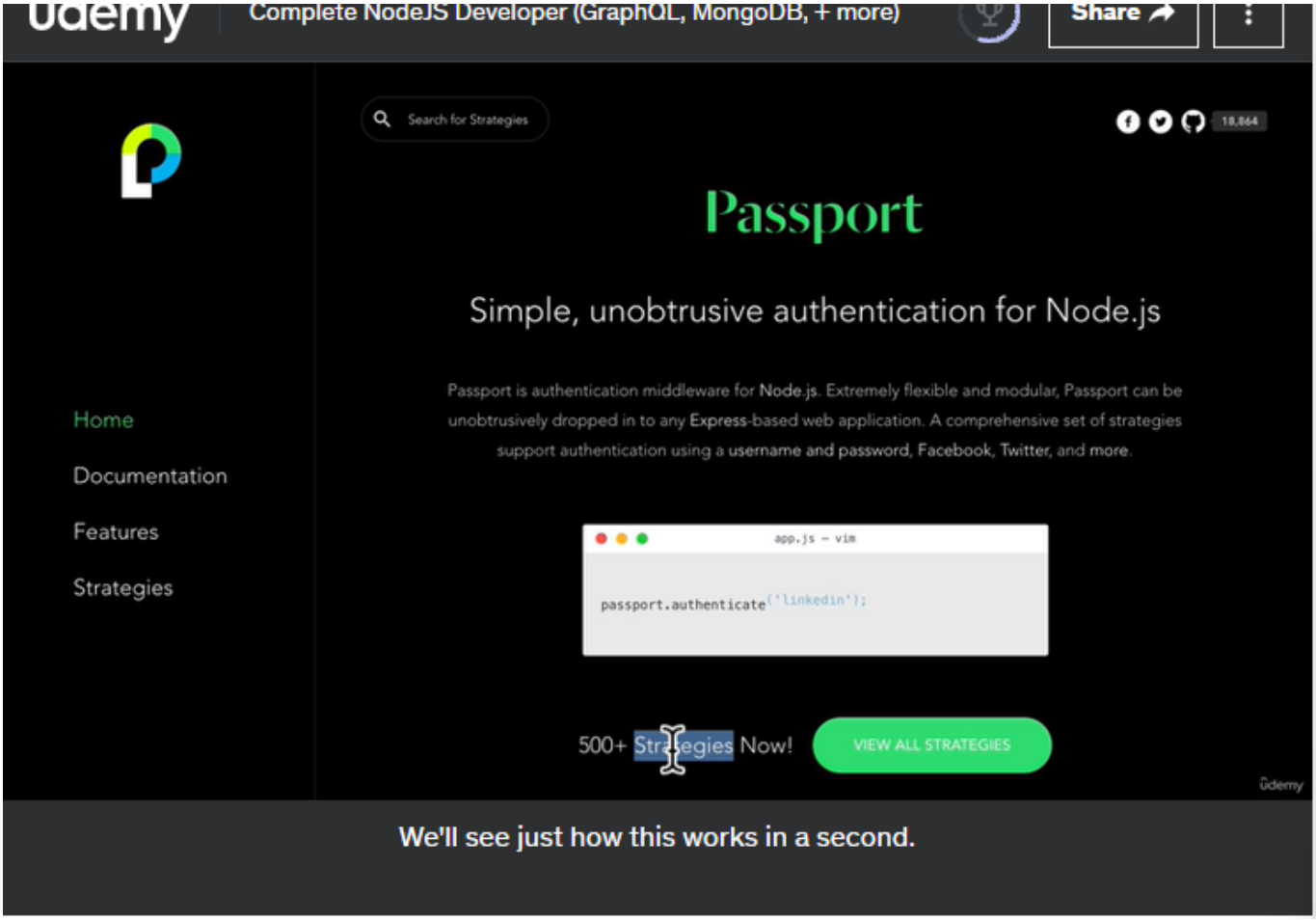
Server-side (AKA web)



White passport

https://www.passportjs.org/

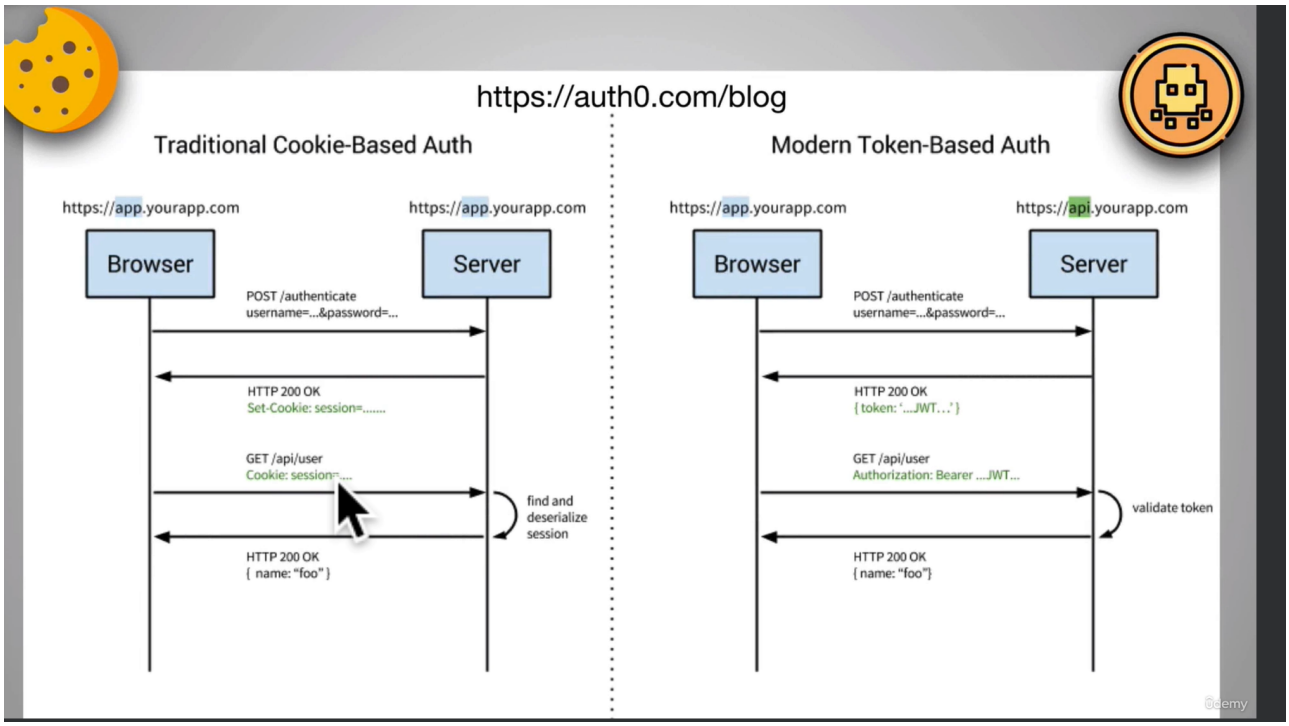
https://www.passportjs.org/packages/passport-google-oauth20/



Provide authenticate for node

Cookies

1. string of data storing in our browser



Two type of using cookies

1. stateful cookies
    1. store session in DB and send to client only reference (changes when a lot of users)
    2. if need keep in our session in a secret
2. stateless cookies
    1. all the session data lives in the client

size cookies limited about 40 kilobytes

Session

1. Are a way of storing data about the current active user.

Storing session data:

1. Server side session -> where user data lives in the server
2. client side session -> when user data lives in the site (cookies)

npm packages:

1. Server side session -> express-session
    1. only saves the Session ID in the cookie
    2. and session data save in DB and permission
2. client side session -> cookies-session
    1. storing all actual data in a cookies in a browser

Serialize its mean saving our user data to a cookie. Deserialize loading that user data from that cookie into a value that we can read.

Role-based access control

That allows them to do some specific set of task in our API Store that role inside a session

Node.js Security Cheat Sheet