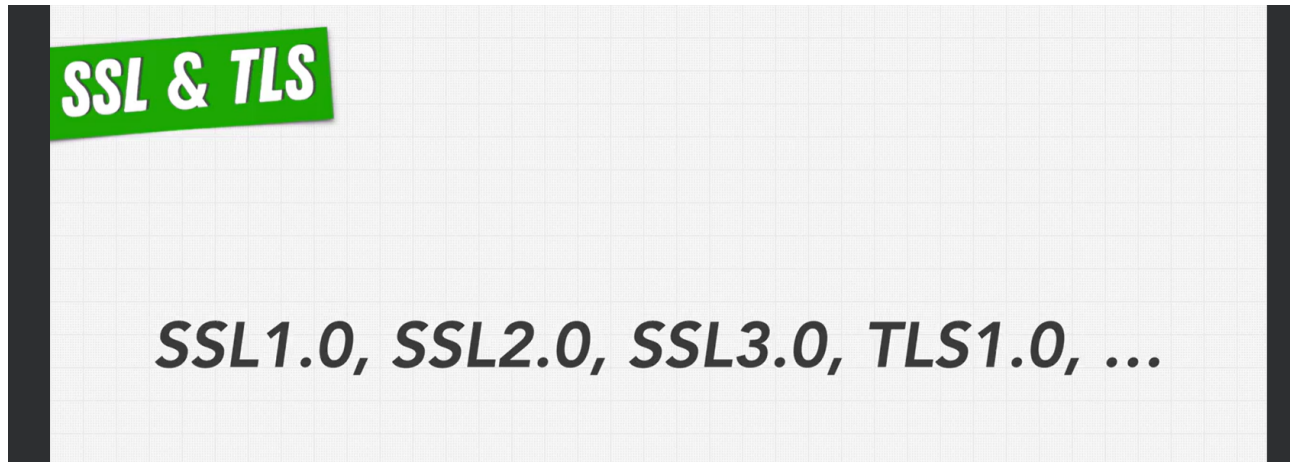


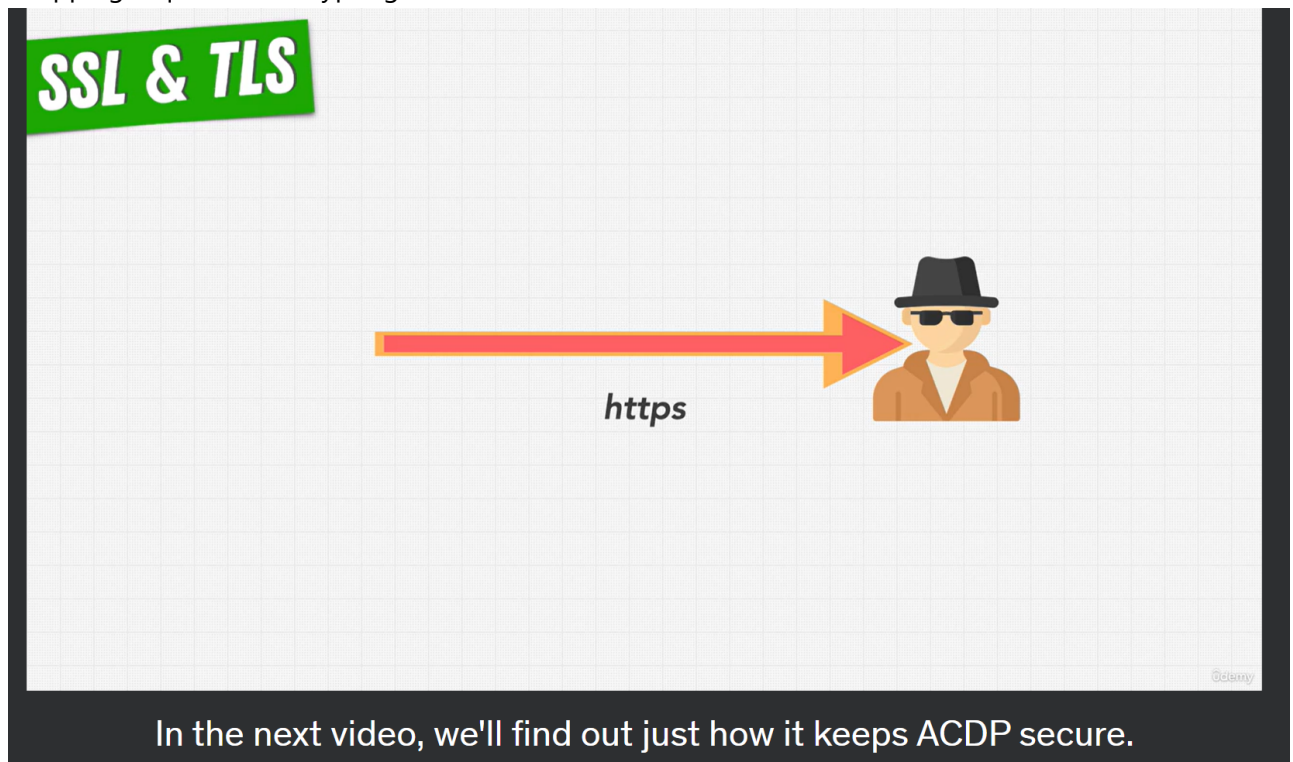
# Fundamentals Node Security + Authentication

---

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

The server could not understand the request due to invalid syntax.

### 401 Unauthorized

Although the HTTP standard specifies "unauthorized", semantically this response means "unauthenticated". That is, the client must authenticate itself to get the requested response.

### 402 Payment Required

This response code is reserved for future use. The initial aim for creating this code was using it for digital payment systems, however this status code is used very rarely and no standard convention exists.

### 403 Forbidden

The client does not have access rights to the content; that is, it is unauthorized, so the server is refusing to give the requested resource. Unlike 401, the client's identity is known to the server.

404 permission 401 authenticate

Api key

It's a string

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

---

## 429 Too Many Requests

The HTTP **429 Too Many Requests** response status code indicates the user has sent too many requests in a given amount of time ("rate limiting").

A [Retry-After](#) header might be included to this response indicating how long to wait before making a new request.

## Status

429 Too Many Requests

google maps

## Adding the API key to your request

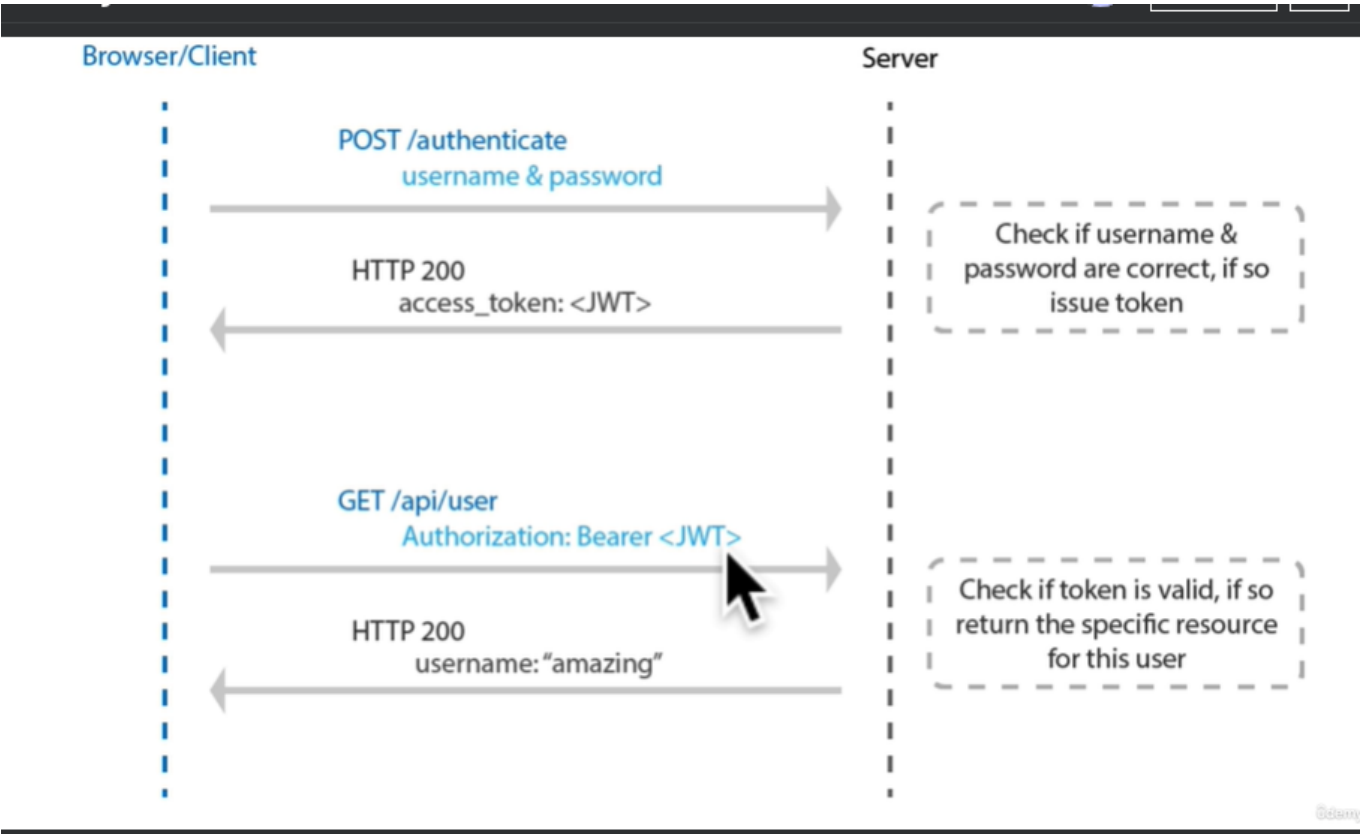
You must include an API key with every Maps JavaScript API request. In the following example, replace `YOUR_API_KEY` with your API key.

```
<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 tokens



Encoded

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkhlcmsgUm9iZXIiLCJjb3VudHJ5IjoiaVVNbIiwiaWYWRtYW4iOiOnRydWUsImIhdCI6MTUxNjIzOTAYMn0.W3ckXmH\_GQSXqiNAKqEZJuJLStcWqK8wILirg3kBKM

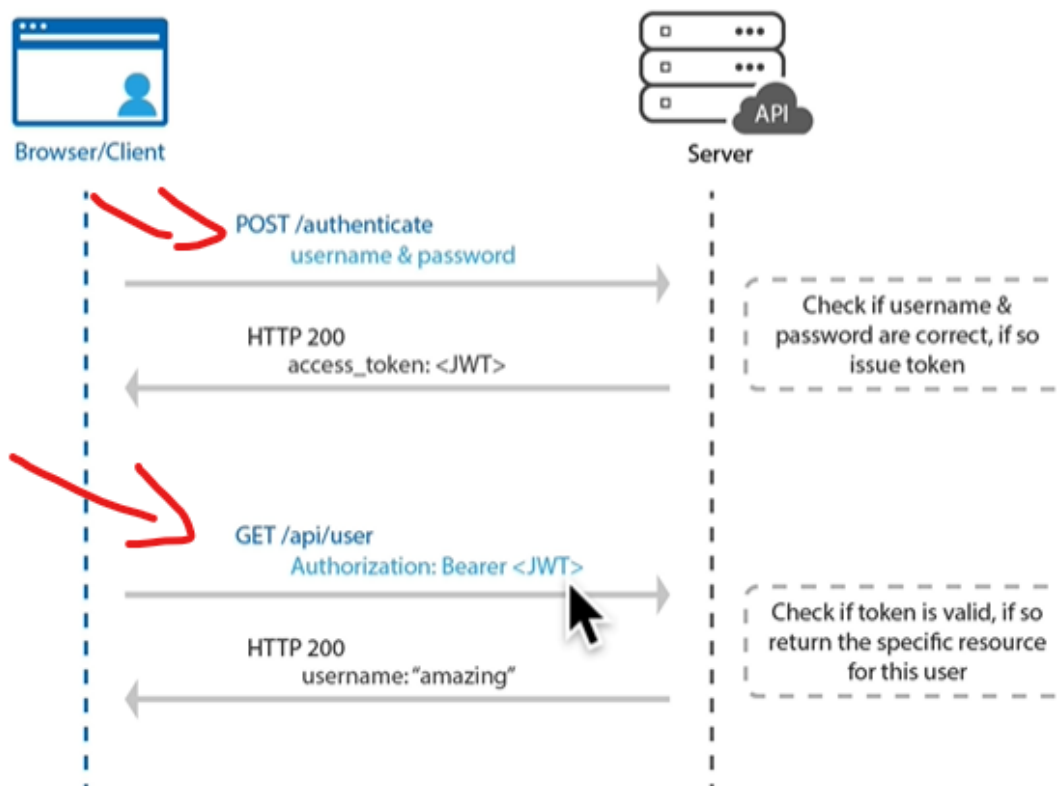
## EDIT THE PAYLOAD AND SECRET

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

```
{
  "sub": "1234567890",
  "name": "Mark Rober",
  "country": "USA",
  "admin": true,
  "iat": 1516239022
}
```

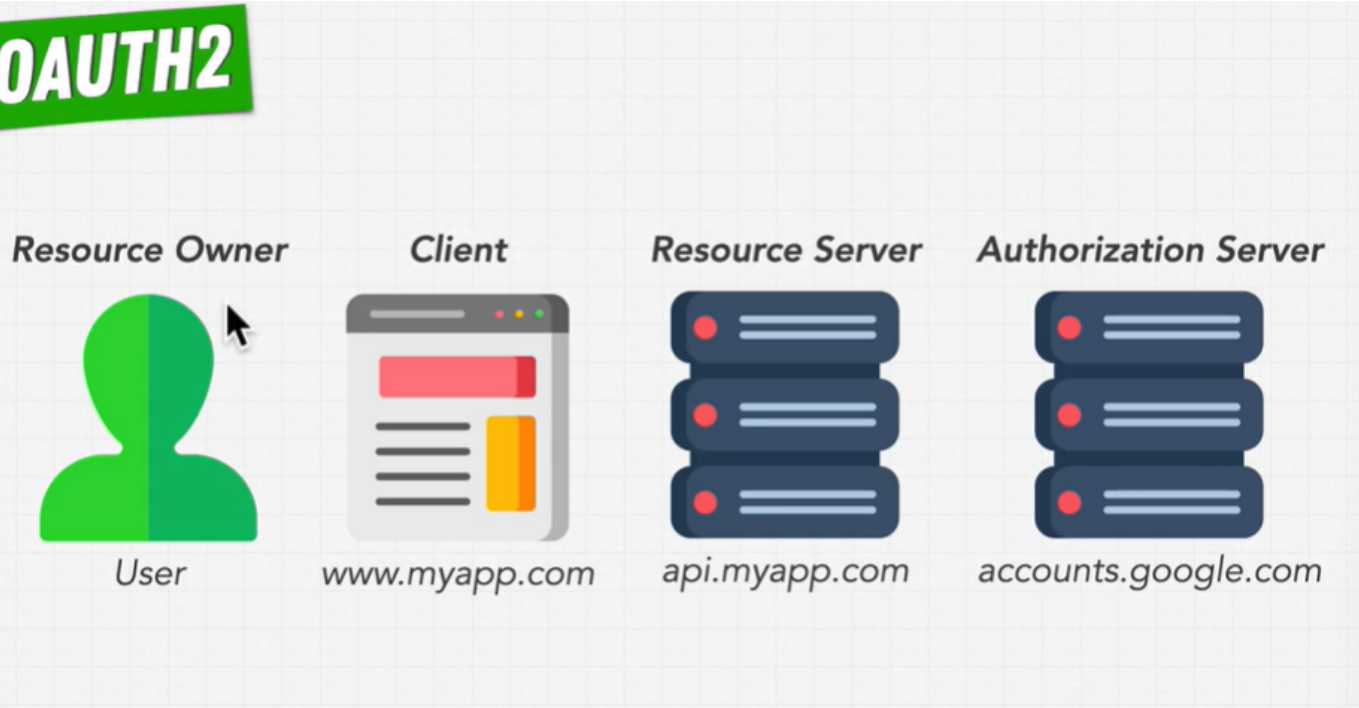
HMACSHA256 (

## Modern Token Authentication



## The OAUTH 2.0 authentication standard

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

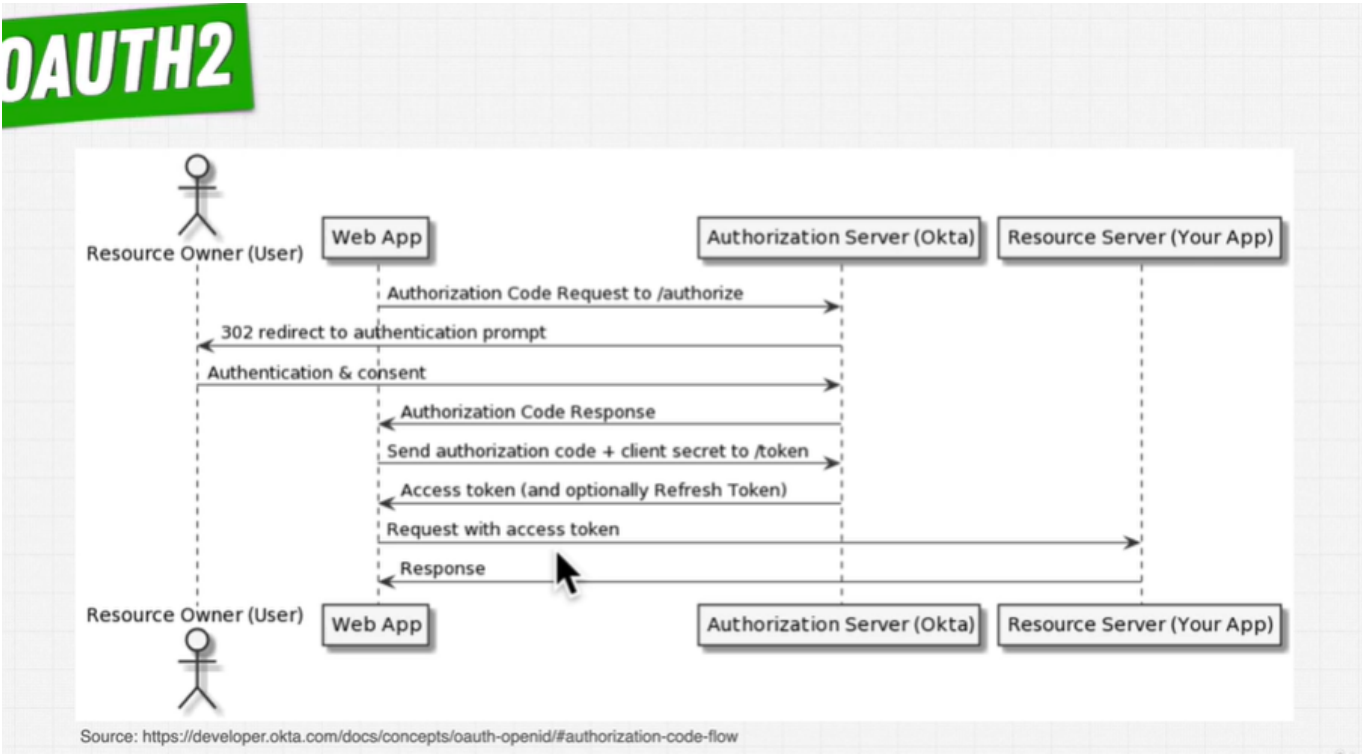


1. has difference flows  
The table shows you which OAuth 2.0 flow to use for the type of application that you are building.

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

Server-side (AKA web)





White passport

<https://www.passportjs.org/>

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

udemy

Complete NodeJS Developer (GraphQL, MongoDB, + more)

Share

Search for Strategies

18,864

Passport

Simple, unobtrusive authentication for Node.js

Passport is authentication middleware for Node.js. Extremely flexible and modular, Passport can be unobtrusively dropped in to any Express-based web application. A comprehensive set of strategies support authentication using a username and password, Facebook, Twitter, and more.

app.js ~ vim

passport.authenticate('linkedin');

500+ Strategies Now!

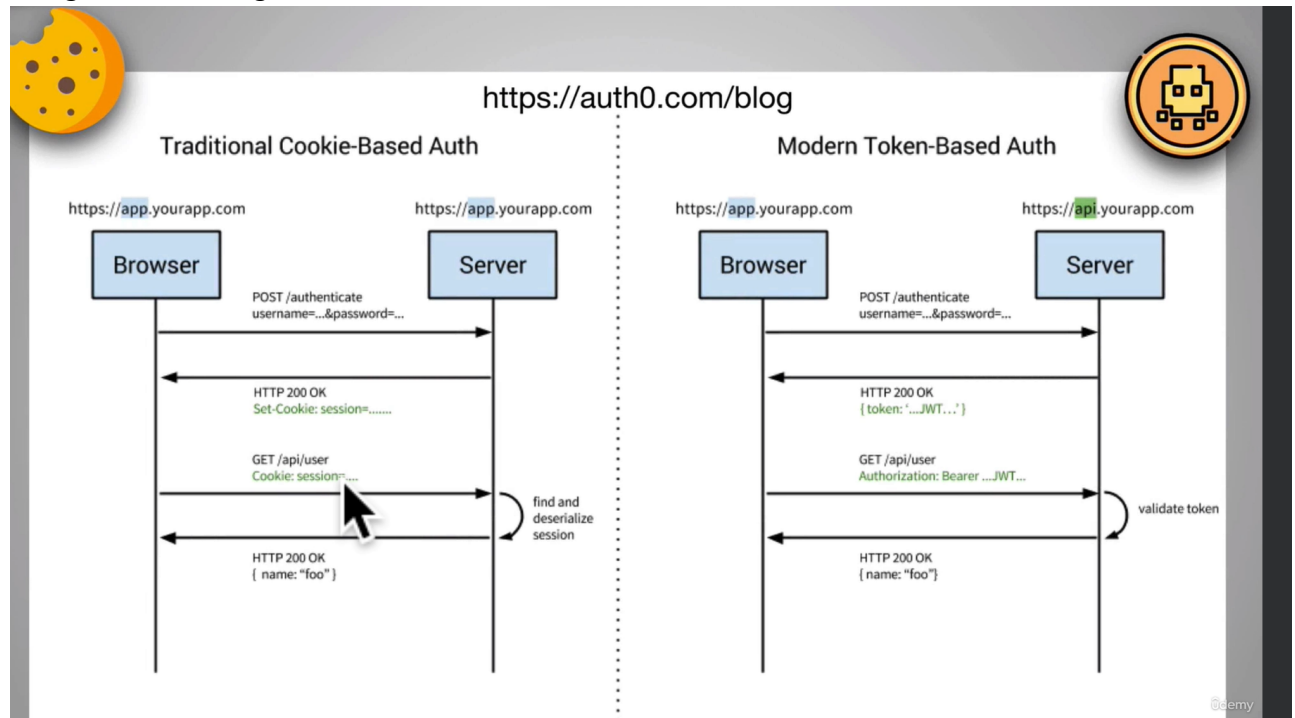
VIEW ALL STRATEGIES

We'll see just how this works in a second.

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](#)