Authentication / Authorization

- Authentication (Auth)
    - Who are you?
- Authorization (Authz)
    - What are you allowed to do?

Factors

A way of proving auth/authz

- Something you know
  - passwords, PIN
- Something you have
  - keycards, yubikey, RSA token, cellphone
- Something you are
  - fingerprints, iris, face

2FA is "two factor auth", MFA is "multi-factor (2+) auth"

Login

Authenticates, possibly authorizes

- Username
- Password

Send both. Per security discussion, server will compare hashed password+salt to stored salt+hash for that username.

But then what?

Beyond Stateless

Web requests are stateless

How do you let the server know a later request is from someone that has already authenticated?

Session Id

How to solve with session id

When user successfully authenticates:

- create a random string (session id)
- on the server:
    - connect the username and authz info with id
        - often a DB entry.
        - This course: just keep in memory
    - return this session id to the client
- on the client:
    - send this session id with any later requests

Other tokens

Session Id is a "token" that by itself is random

Other tokens contain usable info directly, but are "signed" to prove who created them

Example: JWT (JSON Web Token) ("jot")

Basically a text file saying "Whoever has this token has proven themselves to be X or be allowed to Y", with a digital signature to prove it came from the owner of some private key.

JWT

Advantages

- No DB check each time used
- Can be passed to others
    - This is how many 3rd party login systems work

Disadvantages

- Good for their lifetime, even if user "logs out"
- Don't want to store changing info in them

Ways to send auth token back and forth

- Cookies
    - header of a request/response
    - connected to a domain
    - server can "set" in a response
    - stored in browser
    - browsers auto send on later requests
    - Works across tabs/browser windows
        - Not across profiles/incognito mode
- Forms (rough)
    - send as hidden field
- Front end JS to include on requests

Cookie details

- "Remember me"?
  - sets cookie to expire(or not) after a long time
  - Otherwise cookie gone when browser exits
- Logout
  - Server can overwrite with new stored value/expiration date
- Other cookie config options
  - A path root (rare)
  - HTTPS only ("secure", recommended)
  - Be unseen by browser JS (confusing name)
  - Not interact with 3rd party pages (recommended)

Express cookie example

```
// express "middleware", this time as an extra library
const cookieParser = require('cookie-parser');
app.use(cookieParser());

// (skipping over other express stuff)
app.get('/', (req, res) => {
  const store = req.query.store;
  if(store) {
    res.cookie('saved', store);
  }

  const saw = req.cookies.saved;
  res.send(`<p>Request had cookie "saved": ${saw}</p>`);
});
```

Steps

1. Inside new project directory:
    - `npm init -y`
    - `npm install express`
    - `npm install cookie-parser`
2. Create the `server.js` (or whatever you call it) file
3. run `node server.js`
4. go to `localhost:3000` in the browser
5. use `?store=SOMEVAL` at end of url to set the cookie
6. DevTools-Network-Headers to see the `Set-Cookie` in the response and the `Cookie` in the request
7. DevTools-Application-Cookies (left) to see cookies

Changing the cookie example

Do you know how to:

- Store the cookie under a different name
  - not `"saved"`?
- Change the name of the query param you are sending to set the cookie value?
  - instead of `"store"`
- Redirect the user to `'/'` (no query param) after setting the cookie?