Server-side Authentication and State

Achieving State in a Stateless Protocol

- There are a number of ways state can be maintained:
- Client-side:
  - Embedded in URL query parameters or paths
  - Embedded in hidden form fields
  - In the script running on a single page load
  - In localStorage / indexeddb storage that scripts can access between loads
- Server-side
  - By means of using cookies or other headers to remember user / session identifiers

Cookies

- The server can give the client a cookie.
  - Cookies are established in a **Set-Cookie response** header.
  - Cookies are name/value pairs with some restrictions.
- The client saves the cookie.
  - In the Set-Cookie header, the server instructs how long to save the cookie.
  - The cookie is stored *specific to the domain* and potentially *path* (if set in Set-Cookie).
- On subsequent requests to the the same domain name, the client gives the cookie back.
  - The cookies are sent back in a **Cookie request** header.

Secure Hashes

- Hashing is a one-way encoding of a secret
  - Algorithmic generation of a secure hash for a secret is easy once algorithm is verified/implemented.
    - Outside of scholarly, throw-away projects, never implement a secure hashing also yourself.
  - Exceedingly difficult to reproduce the secret from the hash
  - Given the secret again in the future, easy to check whether some hash is valid for the secret.
- Passwords are (or *should be*) stored as secure hashes!
  - When you register or reset a password, the server stores a secure hash in DB
  - When you login, and present your password again, the server checks validity of hash.
  - Your password should never be stored in plaintext by a 3rd party service (and you should never store your users' passwords in plaintext!)
- Which hash to use? A reasonable choice today is **bcrypt.** Don't use MD5/SHA1/etc. Take COMP535 for more detail!

Using bcrypt

- To add to our project:
  - npm install --save bcrypt
  - npm install --save-dev @types/bcrypt
- Hash:

```
const SALT_ROUNDS = 12;
let hash = await bcrypt.hash(req.body.password, SALT_ROUNDS);
```

- Test validity:

```
let isValid = await bcrypt.compare(req.body.password, process.env.ADMIN_PASSWORD_HASH);
```

Using cookies

- Cookie parser:
  - npm install --save cookie-parser
  - npm install --save-dev @types/cookie-parser
- To use encrypted cookies, we'll need to add a COOKIE_SECRET to our .env file
  - Fine choice to randomly generate a secret, such as a UUID
  - Online UUID generator: https://www.uuidgenerator.net/
  - Add variable COOKIE_SECRET=...uuid... to .env file
- Import cookie parser: import * as cookieParser from "cookie-parser";
- Register it as middleware for admin area: router.use(cookieParser(process.env.COOKIE_SECRET));

Signed Cookies

- Our cookie names and values will be sent in plain-text in headers
- We'd like to verify the value of the cookie was established by our server
    - We don't want the client to be able to set any value they'd like!
- By using a cookie library's signing capability, you can establish the veracity of cookies. Their contents come with a signature that is effectively a secure hash that *also encodes* server-side secret.
    - In our case, the COOKIE_SECRET env variable.

Add Authentication to our Project

- Let's add authentication to the admin area of your project!

Sessions and User Data Discussion