Week 10 - Auth

Auth

Quick recap

- We generally don't want to give access to all users to all parts of our application
- Authentication x Authorization
- There are many ways to authenticate users
  - Passwords
  - Magic links
  - OIDC

> We will focus on the simplest token based authentication: `SessionID`

Auth with SessionID

- A user logs in
  - We generate a unique `SessionID` for them and store it **somewhere**
  - We send the `SessionID` to the user **somehow**

Auth with SessionID

> Where do we store the `SessionID`?

- Requirements:
  - Secure (Only the server should be able to read it)
  - Fast (We will be accessing this value a lot, disk access is slow, should be memory based)
  - (Optional) Distributed (We might need to be able to access it from multiple servers)

Auth with SessionID

> How do we send the `SessionID` to the user and maintain the session?

- Requirements:
  - Portability (Should work with the majority of clients)
  - Security (Should not be easily accessible by other websites or malicious scripts)
  - (Optional) Persistence (Should survive browser restarts)
  - (Optional) Expiration (Should not be valid forever)
  - (Optional) Ease of use

HTTP Cookies

- A cookie is a small piece of data stored on the client's computer by the web browser while browsing a website
- Can set the following attributes:
  - `Secure` - Only send over HTTPS
  - `HttpOnly` - Cannot be accessed by JavaScript
  - `SameSite` - Prevents the browser from sending the cookie along with cross-site requests
  - `Domain` - The domain the cookie is valid for
  - `Path` - The path the cookie is valid for
  - `Expires` - The expiration date of the cookie
  - `Max-Age` - The maximum age of the cookie in seconds

Auth with SessionID (express-session)

- A middleware for Express.js
- Creates and maintains sessionids for you
- Can store the sessionid in memory, in a database or in a cache (like Redis)

```
app.use(
  session({
    secret: "keyboard cat",
    resave: false,
    saveUninitialized: false,
    cookie: { secure: false, httpOnly: true }, // secure: false => http (not https), always us
    store: new RedisStore({ client: redisClient, prefix: "x-session:" }),
  })
);
```

Auth with SessionID (passport)

- An auth framework for Node.js
- Handles authentication strategies
  - `passport-local` - Local username and password
  - `passport-jwt` - JWT
  - `openid-client` - OpenID Connect
  - ... (many more)

```
passport.use(
  new LocalStrategy((username, password, done) => {
    // Fetch the user (from db or cache)

    // Check if the user exists and the password is correct
    if (!user) {
      return done(null, false, { message: "Incorrect username or password." });
    }
    // Check if the password is correct
    if (isValidPassword(user.password, password)) {
      return done(null, false, { message: "Incorrect username or password." });
    }
    return done(null, user);
  })
);
```

Password storage

- Never store passwords in plain text!
- Use secure hashing algorithms like `bcrypt` or `argon2` (not SHA, despite the name, lookup tables exist)
- Always use a salt (already included in `bcrypt` and `argon2`)

```
const hash = await argon2.hash("password");
// $argon2i$v=19$m=16,t=2,p=1$czhaaERxODRwZnFNaEFjbg$+eV7nw2kAE27VXgZL7+dSg
```

> If possible, try not to manage passwords yourself or be very careful

```
- Errors in password management are very costly
```

Protecting routes

- Use middleware to protect routes

```
const protected = (req, res, next) => {
  if (isRequestAuthorized(req)) {
    next();
  } else {
    res.status(401).send("Unauthorized");
  }
};

app.get("/protected", protected, (req, res) => {
  res.status(200).send("Protected route");
});
```

On the frontend

With cookies, server manages the session. You don't have to do anything on the frontend! (almost)

- Maybe just redirect the user to the login page if they are not authorized

Open source project shill

Insomnium - https://github.com/ArchGPT/insomnium

- A simple API testing tool, fork of Insomnia

Time to code!

JWT

- JSON Web Tokens (JWT) are an open, industry standard RFC 7519 method for representing claims securely between two parties.
- Explore JWTs at jwt.io
- JWTs do not need server side storage

> Instead of using sessionids, try to refactor your code to use JWTs instead with `passport-jwt`
> If you have time, try incorporating a third party authentication provider like Google, Facebook or GitHub for sign ins

On the frontend

- You will need to store the JWT in the browser
- You can use `localStorage`, `sessionStorage` or `cookies` again

With a JWT pair (access token and refresh token), you can implement a refresh token mechanism.

- Access token is short lived (minutes)
- Refresh token is long lived (days)
- When a request fails due to expired token, use the refresh token to get a new access token and retry the request

That's it!

What, you managed to finish this seminar in time?

State management

State management

- There are different types of `state` in an application
  - Local state
  - Data fetching state (data, loading, error)
  - Form state (input values, errors)
  - Routing state (current route, params)
  - ... not much left

State management

- For each of those, we have a specialized tool:
  - Local (or kindof local) state: `useState`, `useReducer`, `useContext`
  - Data fetching state: `@tanstack/react-query`
  - Form state: `react-hook-form`
  - Routing state: `react-router-dom`

> General rule of thumb: Use the simplest tool that gets the job done

> In the past, all-in-one solutions like Redux were popular, but they are not necessary anymore (and cumbersome to use)

State management

- But for whatever is left, we can use `jotai`!

Jotai

- Composable state management library for React
  - based on atoms
  - derived state through atoms/selectors (based on lib)
  - extremely performant!

Primitive atoms

```
import { atom } from "jotai";

const countryAtom = atom("Japan");

const citiesAtom = atom(["Tokyo", "Kyoto", "Osaka"]);

const animeAtom = atom([
  {
    title: "Ghost in the Shell",
    year: 1995,
    watched: true,
  },
  {
    title: "Serial Experiments Lain",
    year: 1998,
    watched: false,
  },
]);
```

Derived atoms

```
const progressAtom = atom((get) => {
  const anime = get(animeAtom);
  return anime.filter((item) => item.watched).length / anime.length;
});
```

> Derived atoms are updated when the atoms they depend on change

Atom extensions

```
import { atomWithStorage } from "jotai/utils";

// Set the string key and the initial value
const darkModeAtom = atomWithStorage("darkMode", false);
```

> Large number of extensions available

In React

- Just use a hook, anywhere!

```
const Input = () => {
  const [text, setText] = useAtom(textAtom);

  const handleChange = (e) => setText(e.target.value);
  return <input value={text} onChange={handleChange} />;
};
```

That's it! (for real this time)