Cookies

Cookies are typically used for storing stateful information for the session, such as whether the user is logged in, user's preferences/settings, items in a shopping cart, etc. You typically set a time limit, so the cookie will eventually expire and the data will be erased.

Benefits

- The biggest benefit is the persistence of cookies. It can easily be used to store information for repeat visitors even if they close the browser.
- They work without the user having to know anything about their use.

Downsides

- If a user disables cookies, it can cause problems or even break the application if the cookies are required for your site to work properly.
- Each cookie is limited in information stored; about 4 kilobytes per cookie (1 byte represents one letter, so about 4 thousand letters per cookie).
- Because of the above, they're almost always limited to string information.
- Most browsers limit the number of cookies per site. This, combined with the small amount of data sent, can cause problems if you need to send a lot of data via cookies.
- Easily accessible and readable if the user finds them. Also vulnerable to XSS (Cross-site scripting) attacks and CSRF (Cross-site request forgery) attacks. Here are two mozilla docs that talk about these: (https://developer.mozilla.org/en-US/docs/Glossary/Cross-site_scripting) (https://developer.mozilla.org/en-US/docs/Glossary/CSRF)

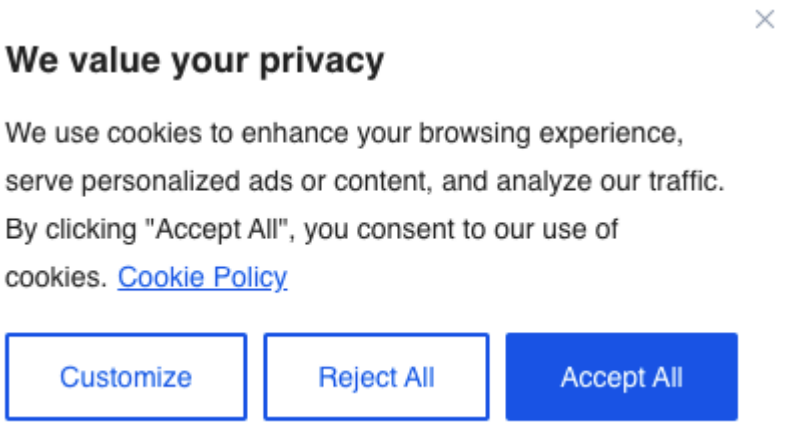
Basically, cookies are really useful for storing simple information that does not need to be secure.

Make sure you're following the law

If you use cookies, it's best practice to implement cookie management and consent management on your site in order to comply with privacy laws in certain countries.

1. Include an opt-in disclosure, in which the user must specifically agree to the use of cookies.
2. Include an opt-out disclosure, in which the user is given the option of blocking some, or all, cookies.
3. Include an implied consent disclosure, in which the user is informed that their use of the website implies consent to the use of the cookies.

Here's a good example of a cookie consent banner/modal:



Implementation

While using express, you can use a library called `cookie-parser` to manage cookies. There is also another library called `js-cookie` that is used on the front-end, but this one is more secure.

To make a cookie:

1. Give it a name
2. Give it some data
3. Give it a maxAge property. This is a time limit on how long the cookie should last (in miliseconds). As of August 2022, cookies can no longer last more than 400 days.

```
import express from 'express';
import cookieParser from 'cookie-parser';

const app = express();
app.use(cookieParser()); // Enables cookie-parser

app.post('/login', (req, res) => {
    const { username, password } = req.body;  // Assuming these are correctly sent
from the login page

    // Validate credentials (this example assumes validation is successful)
    if (username === 'admin' && password === 'password') {
        // Making a new cookie called username, storing their username, with an
age of 7 days
        res.cookie('username', username, { maxAge: 604800000, httpOnly: true });
        res.cookie('isLoggedIn', 'true', { maxAge: 604800000, httpOnly: true });
        res.send('Login successful!');
    } else {
        res.status(401).send('Invalid credentials');
    }
});

// By setting httpOnly to true, you make the cookie inaccessible to JavaScript
running in the browser, mitigating the risk of a client-side script accessing the
protected cookie.

// Whenever you need to check if a user is logged in or retrieve the username, you
```

```
can access the cookies from the request object provided by cookie-parser.

app.get('/dashboard', (req, res) => {
    if (req.cookies.isLoggedIn && req.cookies.username) {
        res.send(`Welcome back, ${req.cookies.username}!`);
    } else {
        res.status(401).send('Please login to view this page');
    }
});

// To log out a user, you can clear the cookies by setting their expiration date
to a past time.
// You can also just use the res.clearCookie function.

app.get('/logout', (req, res) => {
    res.clearCookie('username');
    res.clearCookie('isLoggedIn');
    res.send('Logged out successfully');
});
```

If you want to read information in a cookie from the front-end, you would make a fetch request to whichever endpoint is sending that cookie information (in this example, it's /dashboard). The only difference is that you need to set credentials to 'include' like this:

```
fetch('http://localhost:3000/dashboard', {
    method: 'GET',
    credentials: 'include'
})
// ... now add all the .then functions like normal
```

This makes sure that the cookies are actually being sent in the request so that the server can access the cookies.