Cookie Storage

▼ What is a `cookie`?

A **cookie** (also known as a web cookie or browser cookie) is a small piece of data a server sends to a user's web browser. The browser may store cookies, create new cookies, modify existing ones, and send them back to the same server with later requests. Cookies enable web applications to store limited amounts of data and remember state information; by default the HTTP protocol is stateless. (source MDN)

Scope: Cookies can store data that persists across sessions, unlike `sessionStorage` and similar to `localStorage`, but they have additional capabilities such as being sent with HTTP requests.

▼ How does it work and how do we use it?

Cookies are sent by the server to the client using HTTP headers (Set-Cookie) and can be read, modified, or deleted by both the client and the server. Cookies can also be manipulated using JavaScript. Cookies are transmitted via HTTP call.

Cookies set by the client can always be read by the server however, the cookies set by the server based on some configuration(HttpOnly attribute etc.) may not be read by the client.

The cookies are stored as key-value pairs and can be set using JavaScript like this:

```
document.cookie = "yummy_cookie=choco";
document.cookie = "tasty_cookie=strawberry";

console.log(document.cookie);
// logs "yummy_cookie=choco; tasty_cookie=strawberry"

document.cookie = "yummy_cookie=blueberry";
```

```
console.log(document.cookie);
// logs "tasty_cookie=strawberry; yummy_cookie=blueberry"
```

The various attribute values can optionally follow the key-value pair, each preceded by a semicolon separator:

1. `;domain` : The host to which the cookie will be sent. Specifies the domain within which this cookie is valid. It allows subdomains to access the cookie if set properly.
2. `;expires=date-in-GMTString-format`: The expiry date of the cookie. If neither `expires` nor `max-age` is specified, it will expire at the end of session.
3. `;max-age=max-age-in-seconds`: The maximum age of the cookie in seconds (e.g., `60*60*24*365` or 31536000 for a year).
4. `;samesite`: The `SameSite` attribute of a `Set-Cookie` header can be set by a server to specify when the cookie will be sent. Possible values are `lax`, `strict` or `none`

- The `lax` value will send the cookie for all same-site requests and top-level navigation GET requests. This is sufficient for user tracking, but it will prevent many Cross-Site Request Forgery (CSRF) attacks. This is the default value in modern browsers.
- The `strict` value will prevent the cookie from being sent by the browser to the target site in all cross-site browsing contexts, even when following a regular link.
- The `none` value explicitly states no restrictions will be applied. The cookie will be sent in all requests—both cross-site and same-site.

5. `;secure`: Specifies that the cookie should only be transmitted over a secure protocol.
6. ;`HttpOnly`: If present, the cookie is inaccessible to JavaScript's `document.cookie` API, enhancing security.
7. ;`secure`: If present, the cookie will only be transmitted over secure protocols like HTTPS.

▼ Size Limit

Each cookie can store up to 4KB of data per domain.

▼ Performance

Cookies are limited in size (4KB per cookie), which restricts the amount of data that can be stored.

Cookies are sent with every HTTP request to the server, which can lead to increased latency and bandwidth usage if the cookies are large or numerous.

▼ Cookie Persistence

Session Cookies: expires when the session expires

Persistent Cookies: expiry of a cookie set by the expiry or max-age properties of the cookie

Cookie attributes such as max-age and expiry are used to manage the cookies' persistence

▼ Data Structure

Cookies are simple key-value pairs with optional attributes such as expiration date, path, domain, secure flag, HttpOnly flag etc.

▼ Security of Cookies

Various ways to secure the cookies are:

1. Use the `HttpOnly` attribute to prevent client-side scripts from accessing the cookies. This mitigates the risk of cross-site scripting (XSS) attacks.
2. Set an appropriate expiration time using the `Expires` or `Max-Age` attribute. This helps in managing the lifespan of cookies and ensures they do not persist longer than necessary.
3. Set the `Secure` attribute to ensure cookies are only sent over HTTPS connections. This prevents cookies from being transmitted over an unencrypted connection
4. The `SameSite` attribute helps prevent cross-site request forgery (CSRF) attacks. It can be set to `Strict`, `Lax`, or `None`.

5. Specify the `Domain` and `Path` attributes to restrict the cookie to a specific domain and path, reducing the risk of the cookies being accessed by unauthorised parts of your application.

▼ When to use?

- **Session Management**: Storing session identifiers or tokens for authenticated sessions.
- **Preferences**: Storing user preferences that need to persist across sessions (e.g., language, theme).

▼ When not to use?

- **Storing Sensitive Information**: Avoid storing sensitive information like passwords, personal data, or payment information.
- **Large Amounts of Data**: Cookies have a small storage limit, so they are not suitable for large data storage.