In a Frontend System Design interview, **Cookies** are treated differently than localStorage. While storage is for **Application Data**, Cookies are primarily for **Identity and Session Management**.

---

1. What is a Cookie?

A cookie is a small piece of data (limited to **4KB**) that a server sends to the user's web browser. The browser stores it and sends it back to the same server with every subsequent HTTP request.

2. Why do we need Cookies when we have Storage?

This is a common interview question. The key difference is **Automatic Transmission**.

- **Automaticity:** Unlike localStorage, which requires JS to manually read and attach data to a header, the browser **automatically** attaches cookies to every matching network request.
- **Server Access:** Browsers can't "see" your localStorage during an HTTP request. Cookies allow the server to know who you are (Session Management) before it even finishes processing the request.
- **Security:** Only cookies can be marked as HttpOnly, making them invisible to malicious scripts (XSS protection), whereas localStorage is always vulnerable to JS.

3. Who can Set and Access Cookies?

- **The Server:** Via the Set-Cookie HTTP response header.
- **The Client (JS):** Via document.cookie.
    - *Exception:* If a cookie is marked HttpOnly, JavaScript **cannot** read or write to it. This is a critical security feature.

4. How do Cookies Travel?

- **Server Response:** Server sends a header: Set-Cookie: sessionId=123; Path=/; Secure.
- **Browser Storage:** The browser stores this in its "Cookie Jar."
- **Future Requests:** Every time you click a link or fetch data from that domain, the browser adds a header: Cookie: sessionId=123.
- **The "Travel Agent":** The **Browser** handles the travel automatically based on the Domain and Path attributes.

---

5. Types of Cookies

- **Session Cookies:** Deleted when the "session" ends (usually when the tab/browser closes). They don't have an Expires or Max-Age attribute.
- **Persistent Cookies:** Stay on the device until a specific date (Expires) or duration (Max-Age).
- **First-Party Cookies:** Set by the site you are currently visiting.
- **Third-Party Cookies:** Set by a different domain (like an ad-tracker or a Facebook "Like" button). *Note: Browsers are currently phasing these out for privacy.*
- **Secure Cookies:** Only sent over encrypted **HTTPS** connections.

---

6 & 8. How to make a Cookie Invalid

You cannot technically "delete" a cookie from the server or client; you can only **expire** it.

- **The Trick:** Set the Expires date to a time in the past (e.g., Jan 1, 1970).
- **The Logic:** Set Max-Age: 0.

---

7. Performance Issues

Yes, cookies can significantly hurt performance:

- **Payload Bloat:** Since cookies are sent with **every single request** (including images, CSS, and JS), having 4KB of cookies can add massive overhead to your network traffic.
- **Unnecessary Data:** If you use a cookie to store a user's "Theme Preference," that data is sent to the server even though the server doesn't need it to render the HTML.
- **Solution:** Use **Cookie-free domains** (like static.example.com) to serve assets so cookies aren't wasted on images and scripts.

---

9. Clearing Storage from the Server: Clear-Site-Data

This is a modern, high-level SD topic. You can clear **all** frontend storage (Cookies, LocalStorage, Cache) using a single server header.

`Clear-Site-Data: "cache", "cookies", "storage"`

- **"cache"**: Clears the browser cache.
- **"cookies"**: Clears all cookies for that domain.
- **"storage"**: Clears localStorage, sessionStorage, and IndexedDB.

---

10. Critical Security Attributes

In a System Design interview, you **must** mention these flags:

Attribute	Effect
HttpOnly	Prevents JS access. Mitigates **XSS** attacks.
Secure	Cookie is only sent over **HTTPS**.
SameSite	Prevents **CSRF** (Cross-Site Request Forgery). Options: Strict, Lax (Default), or None.
Domain/Path	Restricts which URLs the cookie is sent to.

Most Asked Interview Q&A

Q: "What is the difference between Expires and Max-Age?"
A: Expires sets a specific calendar date (UTC). Max-Age sets a countdown in seconds. Max-Age is newer and preferred because it isn't affected by the user's system clock being wrong.

Q: "If I have 20 cookies of 4KB each, what happens?"
A: Browsers have limits on total cookies per domain (usually 50-100) and total size. If you exceed these, the browser will start evicting the oldest cookies to make room.

Q: "How do you prevent a CSRF attack using cookies?"
A: Use the SameSite=Strict or SameSite=Lax attribute. This ensures the cookie is only sent if the request originates from your own site, preventing malicious third-party sites from "borrowing" your login session.

In a Frontend System Design interview, distinguishing between **volatile (memory)** and **non-volatile (disk)** storage is key to explaining how an application survives a restart.

The answer depends entirely on the **type of cookie** and its **attributes**:

---

1. Session Cookies (Memory)

If a cookie **does not** have an Expires or Max-Age attribute, it is a **Session Cookie**.

- **Where it lives:** It is stored in the **RAM (System Memory)** of the browser process.
- **Persistence:** Because it lives in RAM, it is wiped clean the moment the browser process is terminated (closing the browser). It never touches the hard drive.

2. Persistent Cookies (Hard Disk)

If a cookie **has** a future Expires or Max-Age attribute, it is a **Persistent Cookie**.

- **Where it lives:** The browser writes these to a local database file (usually a **SQLite database**) located on the **Hard Disk** (User's local profile folder).
- **Persistence:** It survives browser restarts, system reboots, and power failures. The browser reads this file back into memory when it launches.

---

3. The "Browser Cache" Confusion

It is a common mistake in interviews to say cookies are stored in the "Browser Cache." In technical terms:

- **Browser Cache (HTTP Cache):** Stores **files** (images, JS, CSS, HTML).
- **Cookie Store (The Cookie Jar):** A separate storage mechanism specifically for key-value pairs with metadata (Domain, Path, Flags).
- **The Overlap:** While both are stored on the disk, they are managed by different subsystems of the browser engine.

---

4. Deep Dive: Why does this matter for System Design?

Performance (I/O Overhead)

Reading from the **Hard Disk** is significantly slower than reading from **RAM**.

- If your site has 50 persistent cookies, every time a user opens their browser and hits your site, the browser must perform "Disk I/O" to retrieve those cookies before it can even send the first network request.
- **Design Tip:** Keep persistent cookies to a minimum to ensure the fastest possible "Cold Start" for your application.

Security (Data Forensics)

- **Session Cookies** are harder to steal if a device is stolen while turned off (because they aren't on the disk).
- **Persistent Cookies** are sitting in a file on the disk. If an attacker gains physical access to the machine's filesystem, they can copy the SQLite file and potentially hijack the session.

Summary Comparison Table

Feature	Session Cookie	Persistent Cookie
Storage Location	**RAM (Memory)**	**Hard Disk (File)**
Survival	Dies on Browser Close	Survives Reboots
I/O Speed	Extremely Fast	Slower (Disk Access)
Trigger	No Max-Age / Expires	Has Max-Age / Expires

---

1. The 4KB Limit: What exactly is counted?

The **4KB limit (4096 bytes)** is per individual cookie. This limit includes:

- **The Name** (key)
- **The Value**
- **The Attributes** (Domain, Path, Expires, etc.)

If you have a very long Domain attribute or a complex Path, you have even less room for your actual data. If the total string (Name + Value + Attributes) exceeds 4096 bytes, the browser will typically **ignore the cookie entirely** and won't store it.

---

2. The Domain Limit (The "Other" 4KB)

While each cookie can be 4KB, browsers also enforce a limit on the **total number of cookies per domain** (usually between **50 to 180** depending on the browser).

- If you hit the maximum number of cookies, the browser will evict an old one (usually the oldest or least recently used) to make room for the new one.
- This is dangerous because you might accidentally evict the user's session_id cookie to make room for a less important theme_color cookie, effectively logging the user out.

---

3. Why this is "Critical Info" for System Design

If you are designing a high-traffic system, you must consider the **"Cookie Tax"**:

- **The Math:** If you have 20 cookies, each 4KB, that is **80KB of data** added to **every single HTTP request**.
- **The Impact:** If a user is on a 3G connection, uploading 80KB of headers just to fetch a 1KB JSON response will make the app feel incredibly slow.
- **The Interview Solution:** If the interviewer asks how to optimize this, you suggest **Cookie-Free Domains**. You host your static assets (images, JS, CSS) on a different subdomain (e.g., static.example.com) that doesn't have these cookies set. This saves 80KB of upload bandwidth for every image and script.

---

4. Comparison of Limits

Feature	Cookie	LocalStorage	IndexedDB
Individual Item Limit	**~4KB (including meta)**	No specific limit	No specific limit
Total Origin Limit	**~200KB - 400KB** total	**~5MB - 10MB**	**GBs (Percentage of Disk)**
Eviction Policy	Automatic (Oldest deleted)	None (Error thrown)	Automatic (by OS if disk full)

---

5. Most Asked Interview Q&A (The "Missing" Details)

Q: "What happens if I try to set a cookie of 5KB?"
A: The browser will silently fail or reject the Set-Cookie header. It will not be stored, and you won't get a JavaScript error; the cookie simply won't exist in the next request.

Q: "How do you store more than 4KB of identity data?"
A: You don't store the data in the cookie. You store a Session ID (a small string) in the cookie and keep the large data (user profile, permissions, etc.) in a Server-Side Cache (like Redis). This keeps the "Cookie Tax" low.

Q: "Is the 4KB limit for the whole site or per subdomain?"
A: It is per cookie. However, cookies set on .example.com are sent to api.example.com and app.example.com. This can lead to "Header Overflow" errors where the server (like Nginx) rejects the request because the headers are too large.
