Below is a **clean, modern, internally-correct, mastery-level rewrite of Cookies in JavaScript**, rewritten **for deep understanding**, **NoteLM-friendly notes**, **interview readiness**, and **real-world accuracy**.

I've **fixed outdated parts**, **explained internals**, **highlighted security aspects**, and **added developer + interview insights**, while keeping it **easy to digest**.

🍪 Cookies in JavaScript — Deep Dive (From Scratch)

1️⃣ What Are Cookies?

Cookies are **small pieces of data** stored in the **browser** and automatically sent to the **server** with every HTTP request to the same domain.

They are stored as **key–value pairs** and mainly used for:

- Session management
- Authentication
- User preferences
- Tracking & analytics

📌 Cookies live **outside JavaScript memory** — they belong to the **browser storage layer**.

2️⃣ Why Cookies Exist (Very Important)

Problem: HTTP is Stateless

Each HTTP request is **independent**.

So the server:

- ❌ Does not remember who you are
- ❌ Does not know if you are logged in
- ❌ Does not know previous requests

Solution: Cookies

Cookies act as a **shared memory** between:

```arduino
Browser ↔ Server
```

Example:

1. You log in
2. Server sends a cookie: `sessionId=abc123`

3. Browser stores it
4. Every next request automatically sends it back
5. Server recognizes you

✅ **This is how login sessions work**

3️⃣ How Cookies Work (Internal Flow)

```
pgsql

1. Server sends: Set-Cookie header
2. Browser stores cookie
3. Browser attaches cookie to every request
4. Server reads cookie
5. Server identifies user
```

⚠️ JavaScript **does not send cookies manually** — the browser does it automatically.

4️⃣ Cookie Structure (Under the Hood)

A cookie is **not just key=value**. It has metadata.

```
pgsql

name=value; expires=DATE; path=/; domain=example.com; secure; samesite
```

Core Fields

Field	Meaning
`name=value`	Actual data
`expires`	When cookie dies
`path`	URL path restriction
`domain`	Domain restriction
`secure`	HTTPS only
`SameSite`	Cross-site control
`HttpOnly`	JS access blocked

5️⃣ Types of Cookies (Interview Favorite)

Type	Description
Session cookie	Deleted when browser closes
Persistent cookie	Has expiry date
Secure cookie	Sent only over HTTPS
HttpOnly cookie	Not accessible via JS
SameSite cookie	CSRF protection

6️⃣ Accessing Cookies in JavaScript

JavaScript accesses cookies via:

```
js

document.cookie
```

⚠️ Important:

- You **cannot access HttpOnly cookies**
- You **cannot read cookies from other domains**
- You **cannot get structured data automatically**

7️⃣ Setting a Cookie (Correct Modern Way)

```
js

document.cookie = "username=Anoop; path=/";
```

With Expiry

```
js

const expires = new Date();
expires.setDate(expires.getDate() + 7);

document.cookie = `username=Anoop; expires=${expires.toUTCString()}; path=/`;
```

📌 Without expiry → **session cookie**

8️⃣ Encoding Cookie Values (Mandatory)

Cookies **cannot safely store**:

- spaces
- semicolons
- special characters

Always encode values

```js
document.cookie = "user=" + encodeURIComponent("Anoop Yadav");
```

Decode while reading

```js
decodeURIComponent(value);
```

9️⃣ Reading Cookies Properly

`document.cookie` returns a **single string**:

```js
"key1=value1; key2=value2; key3=value3"
```

Safe Reader Function

```js
function getCookie(key) {
  const cookies = document.cookie.split("; ");
  for (const cookie of cookies) {
    const [k, v] = cookie.split("=");
    if (k === key) return decodeURIComponent(v);
  }
  return null;
}
```

🔟 Setting Multiple Cookies

```js
document.cookie = "cartItem=watch; path=/";
document.cookie = "price=10000; path=/";
```

⚠️ Each assignment sets **one cookie only**

1️⃣1️⃣ Updating a Cookie

Updating = **overwrite with same name**

```
js

document.cookie = "cartItem=bag; path=/";
```

✔️ Same key

✔️ New value

✔️ Old value replaced

1️⃣2️⃣ Deleting a Cookie (Correct Way)

Cookies are deleted by setting **expiry in the past**

```
js

document.cookie = "cartItem=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/";
```

⚠️ Path and domain must match the original cookie

1️⃣3️⃣ Cookie Expiry Explained

Expiry Type	Behavior
No expiry	Deleted on browser close
Future date	Persistent
Past date	Deleted

1️⃣4️⃣ Secure & HttpOnly Cookies (CRITICAL)

Secure Cookie

```
http

Set-Cookie: sessionId=abc; Secure;
```

✔️ Sent only over HTTPS

HttpOnly Cookie

```
http
```

```
Set-Cookie: sessionId=abc; HttpOnly;
```

❌ Cannot be accessed via JavaScript

✔️ Protects against XSS attacks

📌 **Authentication cookies must be HttpOnly**

1️⃣5️⃣ SameSite Attribute (Modern Web Security)

Value	Meaning
`Strict`	Only same-site
`Lax`	Default, safe
`None`	Cross-site allowed (requires Secure)

Example:

```
http

Set-Cookie: token=abc; SameSite=Strict;
```

1️⃣6️⃣ Cookies vs Other Storage (Interview Table)

Feature	Cookies	LocalStorage	SessionStorage
Sent to server	✅	❌	❌
Size limit	~4KB	~5MB	~5MB
Expiry	Yes	No	Session
Security	High (HttpOnly)	Low	Low
Use case	Auth	Preferences	Temporary UI

1️⃣7️⃣ Real-World Use Cases

✔️ Login sessions

✔️ Remember user preferences

✔️ Shopping cart tracking

✔️ CSRF protection

✔️ Analytics (limited use now)

❌ Not for large data

❌ Not for sensitive JS-accessible data

1️⃣8️⃣ Common Mistakes (Interview Traps)

❌ Storing JWT in JS-accessible cookie

❌ Forgetting path while deleting

❌ Using cookies instead of localStorage blindly

❌ Ignoring SameSite

❌ Not encoding values

1️⃣9️⃣ One-Line Rule (Very Important)

> **Use cookies only when data must travel with HTTP requests.**
> **Otherwise, use Web Storage APIs.**

2️⃣0️⃣ Final Mental Model

```ini

Cookies = Server ↔ Browser Memory
LocalStorage = Browser Only
SessionStorage = Tab Only
```

⏭️ What Should We Do Next?

Recommended next deep topics (in order):

1️⃣ **LocalStorage vs SessionStorage (Internals)**

2️⃣ **Authentication Flow: Cookies vs Tokens**

3️⃣ **CSRF vs XSS with Cookies**

4️⃣ **SameSite, Secure, HttpOnly (Real Attacks)**

Just reply with **1 / 2 / 3 / 4**.