Cookies

Cookies are small pieces of data stored on the client (browser) by a website. They help websites remember information about users between requests, because HTTP itself is stateless.

What does it mean "HTTP is stateless"?

HTTP does not remember anything between two requests.

When you visit a webpage, the server sends a response and forgets about you immediately.

If you refresh or visit another page, the server treats it like a completely new request, as if it does not know who you are.

Example Without Cookies (Stateless Behavior):

Action	**What you expect**	**What Actually Happens (Because HTTP is Stateless)**
You log in to a website	You expect to stay logged in while browsing	Without cookies, you'd get logged out on every page load because the server doesn't remember you

What Data Is Typically Stored in Cookies?

Use Case	**Example Data Stored in Cookie**	**Example Format**
Authentication/Login session	Session ID or JWT token	sessionId=abc123xyz
User Preferences	Theme, language, layout settings	theme=dark; lang=en
Shopping Carts (Without Login)	Cart item IDs	cart=prod123,prod456
Analytics / Tracking	Unique user identifier	_ga=GA1.2.1892738123.1687349823
Personalization	Last visited page, UI state	lastVisited=/dashboard

What Should Not Be Stored in Cookies?

Cookies are not encrypted by default, so never store sensitive data like:

Passwords

Credit card info

Personal info (Aadhaar, phone number, etc.)

Large JSON objects (cookies have size limits ~4KB)

How Long Do Cookies Store Data?

Cookie Type	**Lifespan**	**How It's Defined**	**Example**
Session Cookie	Until browser/tab is closed	No expiration is set	Set-Cookie: theme=dark;
Persistent Cookie	Until a specific time or duration (minutes, days, months, years)	Uses Expires or Max-Age	Set-Cookie: token=abc123; Max-Age=86400;

Examples:

Session Cookie:

Set-Cookie: sessionId=abc123;

Persistent Cookie:

Persistent Cookie (Manually Set Duration)

How Cookies Are Distinguished?

Even if two cookies have the same name, the browser treats them differently based on their attributes, especially: **1. Expires / Max-Age, 2. Path 3. Domain**

1. Session Cookies

No Expires or Max-Age set

Stored in memory only

Deleted as soon as the browser/tab is closed

Example: Set-Cookie: theme=dark; Path=/; HttpOnly

Browser stores it temporarily

Closing the browser deletes it automatically

2. Persistent Cookies

Expires or Max-Age is set

Stored on disk (not just in memory)

Survives browser restarts until expiration date

Example: Set-Cookie: theme=dark; Path=/; Max-Age=2592000; HttpOnly

Stored for 30 days

Remains even if browser is closed and reopened

How Browser Knows the Difference?

When the browser receives a cookie:

1. If Expires / Max-Age is missing → session cookie
2. If Expires / Max-Age is present → persistent cookie

So the name alone doesn’t matter. Even if both cookies are called theme, their attributes decide their type.

Two Cookies with Same Name:

If two cookies have the same name but different Path or Domain, the browser stores them separately.

On a request, the browser sends the most specific cookie matching the URL.

When Does Auto Logout Actually Happen?

It varies depending on how developers configure things:

Scenario	**Auto Logout Triggered When**	**Controlled By**
Session Cookie (No expiry set)	When the browser is closed	No Expires or Max-Age in cookie
Persistent Cookie (Token with expiry)	After cookie expiry time passes	Max-Age or Expires in cookie
Server-Side Session (Stored in DB or Redis)	When session timeout is reached, even if cookie still exists	Server session expiration
JWT Token Stored in Cookie	When JWT token expires	exp field inside the JWT

Who Sets the Cookie?

Cookies can be set by either the Server or the Browser. And once set, the browser automatically sends them back to the server on each request (if they match the domain/path rules).

1. Server-Side Cookie Setting (Most Common — Used for Authentication):

When a user logs in, the server sets the cookie using the Set-Cookie HTTP response header.

Example (Express.js / Node.js):

Code:

```
res.cookie("sessionId", "abc123", {
httpOnly: true,
secure: true,
sameSite: "lax",
maxAge: 3600000 // 1 hour
});
```

This is sent to the browser as: Set-Cookie: sessionId=abc123; HttpOnly; Secure; SameSite=Lax; Max-Age=3600

Browser stores it automatically

On every future request, browser sends: Cookie: sessionId=abc123

For the first 1 hour — browser automatically sends sessionId=abc123 on every request → ✅ User stays logged in.

After 1 hour — the cookie automatically disappears from the browser.

What Happens on Next Request After Expiry?

1. The browser will NOT send the cookie anymore
2. So when the user makes a new request (page reload / API request):

   The server won't find a session or token.

   The server responds with "Unauthorized" / logs out the user.

3. The user is then redirected to login page (or shown "Please log in").

Important Clarification:

The user does not instantly get logged out at the exact moment of expiry.

They get logged out only when they make the next request after expiry (like clicking, refreshing, or API call).

2. Client-Side (JavaScript) Cookie Setting:

Cookies can also be set by frontend code:

Code:

```
  document.cookie = "username=John; path=/; max-age=3600";
```

But JavaScript cannot set HttpOnly cookies, so this method is less secure (not preferred for login tokens).

So the role of cookie is to set the user logged in for some time... that's the only work?

No — the major role is maintaining state in a stateless protocol (HTTP).

Other Important Uses of Cookies:

1. **User Preferences (Theme, Language, Layout)**

Example: You switch a website to Dark Mode.

The site sets a cookie like: theme=dark; Max-Age=31536000 // 1 year

Even if you close your browser and reopen next week, it stays in Dark Mode — because the cookie stores your preference.

2. Shopping Cart for Guest Users (Without Login)

Ever added items to Amazon without logging in, then returned later and still saw them in your cart?

That’s because Amazon stores something like: cartItems=prod123,prod456;

Stored in a cookie until you log in or complete checkout.

3. Tracking & Analytics (For Page Statistics)

Tools like Google Analytics set cookies like _ga=182381923

This helps websites know:

Info Collected via Cookie	**Purpose**
Unique visitor ID	Count unique users
Time of visit	Track user engagement

Referral source Know if you came from Google or Instagram

No personal details are stored — just tracking identifiers.

4. A/B Testing (UX Experiments)

Websites test two versions of a page to see which gets more clicks.

Example:

variant=A; // You see Design A

variant=B; // Another user sees Design B

Cookies ensure you always see the same version so you don’t get confused.

5. Advertisement Personalization (Ad Targeting)

If you visit a shoe store website, and later see shoe ads everywhere, that’s because: ad_user_interest=shoes;

Advertisers (like Google Ads, Facebook Pixel) store interest cookies to show relevant ads

How Cookie works?

Step Who Does It? Action

User logs in Browser → Server User sends credentials (email/password)

Server verifies user Server If valid, server **creates a cookie** (e.g., sessionId or JWT token) |

Server sends cookie to browser Server → Browser Using Set-Cookie header

Browser stores cookie Browser Automatically, without JavaScript

Browser sends cookie on next request Browser → Server Server identifies the user

How Do We Assign Cookies in Backend?

Example:

```
res.cookie("sessionId", "abc123", {
  httpOnly: true, // Cannot be accessed by JavaScript → Prevents XSS attacks
  secure: true, // Only sent over HTTPS
  maxAge: 3600000, // 1 hour (in milliseconds)
  sameSite: "lax", // Controls cross-site behavior (prevents CSRF)
  path: "/", // Cookie valid for all routes
});
```

In JWT Authentication — How Do We Get a New Cookie After Token Expiry or Logout?

Case 1: Simple JWT Auth (No Refresh Token):

When the JWT expires, the user is effectively logged out.

Next time they send a request, server responds → "401 Unauthorized — Please log in again"

User must log in manually again, and then server issues a new cookie with a new JWT token.

✔ Easy, but user experience is not smooth.

Case 2: JWT with Refresh Token:

This is the more professional flow used by real-world apps.

Token Type	**Stored In**	**Expiry**	**Purpose**
Access Token (JWT)	HttpOnly Cookie	Short (15min–1hr)	Used on every request
Refresh Token (JWT or Random)	HttpOnly	Cookie Long (7-30 days)	Used to request a new Access Token

Flow After Access Token Expires

1. User makes a request → Access Token expired -> Server says 401 Unauthorized
2. Browser automatically sends Refresh Token cookie -> Server checks if valid
3. If Refresh Token is valid → Server generates a NEW Access Token
4. Server sets a NEW cookie with the fresh Access Token
5. Request is retried automatically or client resends

When user clicks Logout:

Server clears both access & refresh token cookies. Now even if old cookies exist, server won't accept refresh token anymore.

How a Cookie with Token Is Set

1. **Backend Creates the Token**

After the user logs in, the backend generates a JWT token (or any token).

Example in Node.js/Express:

```
const jwt = require("jsonwebtoken");

const token = jwt.sign(

  { userId: "user_123" },

  "mySecretKey",

  { expiresIn: "1h" } // Access token valid for 1 hour

);
```

2. Backend Sends the Token in a Cookie

The backend uses Set-Cookie to assign the token to the browser.

```
res.cookie("accessToken", token, {
  httpOnly: true, // JS cannot read it
  secure: true, // Only over HTTPS
  sameSite: "lax", // Protects against CSRF
  maxAge: 3600000 // 1 hour in milliseconds
});
res.send("Logged in successfully!");
```

Here's what happens under the hood:

Browser receives the Set-Cookie header.

Browser stores it automatically.

No JavaScript intervention is needed on the client side (unless you want to read it, which is not recommended for HttpOnly).

3. Browser Sends the Cookie Automatically

On every subsequent request to the same domain:

```
GET /dashboard HTTP/1.1
Host: example.com
Cookie: accessToken=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

Server reads the cookie and validates the token.

4. Backend Validates the Token

```
const token = req.cookies.accessToken;
try {
  const decoded = jwt.verify(token, "mySecretKey");
  console.log(decoded.userId); // Authenticated user
} catch (err) {
  return res.status(401).send("Invalid or expired token");
}
```

Key Points:

Backend creates the token (JWT or session ID).

Backend sends it as a cookie via Set-Cookie.

Browser automatically stores and sends it on future requests.

Backend validates the token on each request.

Important Notes

Always use HttpOnly + Secure for authentication cookies.

SameSite helps prevent CSRF attacks.

Cookie lifetime (maxAge) controls auto-logout.

what is the difference between cookie lifetime and jwt token? how are they similar or different?

1. **Cookie Lifetime**

What it is: How long the browser stores the cookie.

Where it's set: Set-Cookie header from backend, using:

```
res.cookie("accessToken", token, { maxAge: 3600000 }); // 1 hour
```

Effect:

After this time, the browser automatically deletes the cookie.

If the cookie is gone, the browser cannot send the token to the server anymore → user is effectively logged out.

Scope: Browser-side. Doesn't care if JWT inside is valid or expired; if cookie is deleted, it's gone.

2. JWT Token Expiry

What it is: How long the token itself is valid, encoded inside the JWT (exp claim).

```
const token = jwt.sign({ userId: "user_123" }, "secret", { expiresIn: "1h" });
```

Effect:

Even if the cookie still exists in the browser, the server will reject the token after it expires.

User cannot access protected routes until a new token is issued.

Scope: Server-side enforcement (token is validated using secret/signature).

3. How They Interact

Both must be coordinated for smooth UX:

Cookie lifetime ≥ token expiry → ensures token is sent before it expires.

Often:

Access Token → 15 min

Cookie lifetime → 15 min (or slightly longer to allow sending request)

Refresh tokens can have a longer lifetime (days/weeks) in separate cookies.

In a secure JWT-based authentication system, the access token and refresh token are usually stored in separate cookies.