# Cookie

By Ivan Wong

# Agenda

- Introduction to Cookie
- Why Cookie?
- Authentication Re-visit
- Hands-on Lab
  - Exploring Cookie in Chrome
  - Curl with Cookie
  - Cookies with Bash CGI backend and JavaScript frontend

# Introduction to Cookie

- HTTP Request Headers

| Header | Description | Example |
|---|---|---|
| `Accept` | Specifies which types of content the client can process. | `Accept: text/html` |
| `Accept-Charset` | Specifies which character sets the client can process or display. | `Accept-Charset: utf-8` |
| `Accept-Encoding` | Specifies which compressed formats are supported by the client. | `Accept-Encoding: gzip,deflate` |
| `Accept-Language` | Specifies which languages the client accepts. | `Accept-Language: en-US` |
| `Authorization` | Contains authentication data for HTTP authentication methods. | `Authorization: Basic bWF4bXVzdGVybWFubjp0b3BzZWNyZXQ=` |
| `Cookie` | Contains an HTTP cookie previously set by the server via the `Set-Cookie` response header. | `Cookie: user =johndoe;` |
| `Content-Length` | Specifies the length of the body in bytes. | `Content-Length: 348` |

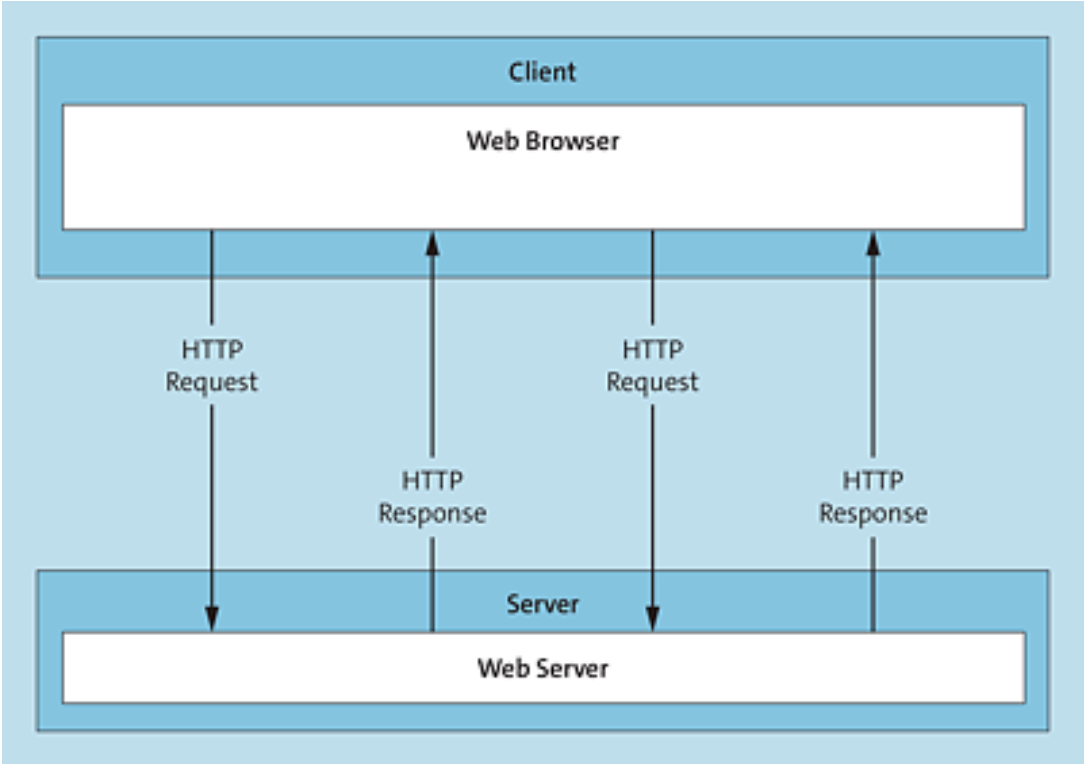
# Introduction to Cookie

- HTTP Response Headers

| Header | Description | Example |
|---|---|---|
| `Allow` | Allowed HTTP methods for the requested resource. | `Allow: GET, POST` |
| `Content-Language` | Language in which the resource is available. | `Content-Language: en` |
| `Content-Length` | Length of the body in bytes. | `Content-Length: 567` |
| `Content-Location` | Storage space for the requested resource. | `Content-Location: /examples.html` |
| `Content-Type` | MIME type of the requested resource. | `Content-Type: text/html;` |
| `Date` | Time of sending the response. | `Date: Mon, 06 Apr 2020 08:00:00 GMT` |
| `Last-Modified` | Time of the last change to the requested resource. | `Last-Modified: Mon, 06 Apr 2020 07:00:00 GMT` |
| `Server` | Details about the web server. | `Server: Apache` |
| `Set-Cookie` | Cookies set by the client. | `Set-Cookie: firstName=John; expires= Mon, 06 Apr 2020 23:00:00 GMT;` |

# Why Cookie?

- The HTTP protocol is a stateless protocol.
  - Each HTTP request from the client to a server is handled by the server independently of other requests.
  - As a result, the server cannot initially detect when a client makes a request for the second time

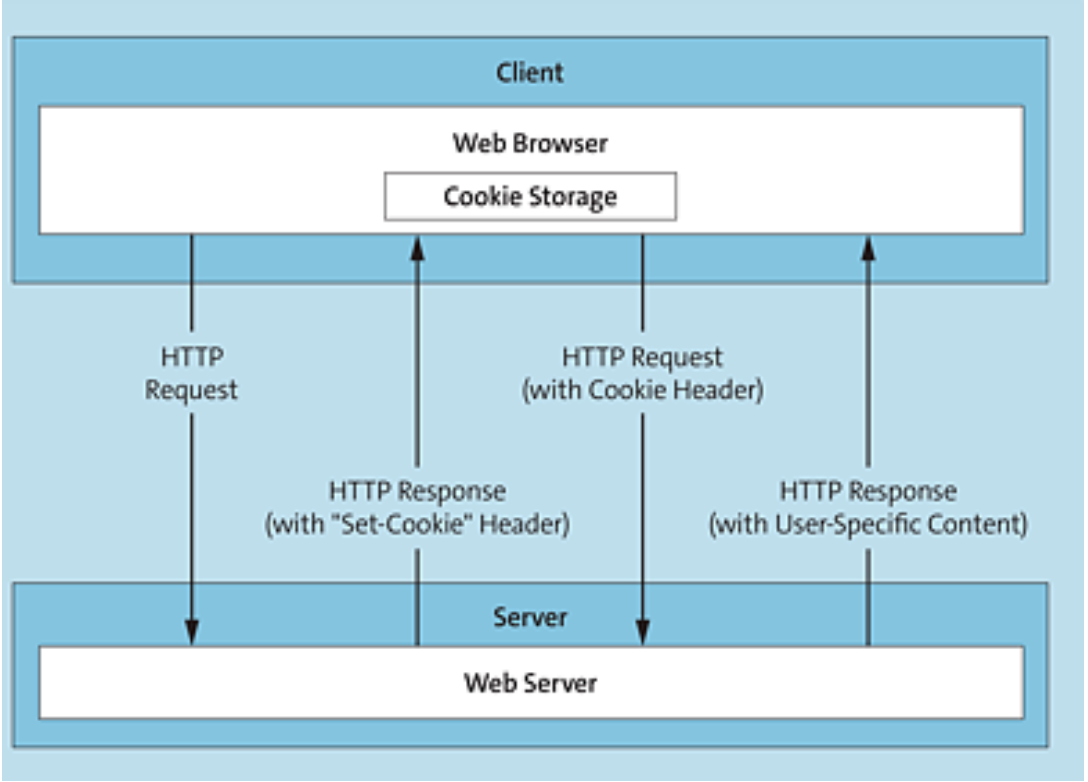


# Why Cookie?

- Nevertheless, in some use cases, detecting this second request is precisely what's needed, for instance, for knowing on the server side exactly which client is calling a web page.
  - Examples of this use case are all web pages where logon is possible and the server must keep track of whether the visitor to the web page has already logged on.
  - But store pages that allow users to add items to the shopping cart without logging on often also remember the products in the shopping cart without the user having to log on: If they call the web page the following day, those items are still in the shopping cart.

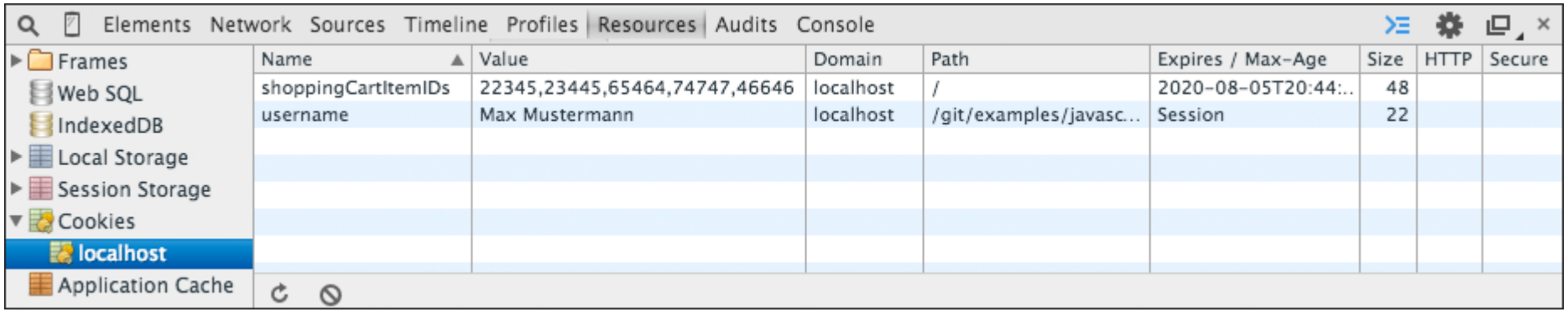
# Why Cookie?

- Cookies enable you to store small amounts of information on the client side and—when the user visits a web page again—to read this information on the server side.
- Cookies are key-value pairs (or name-value pairs) that the browser stores as text files on the user's computer.
- Of course, cookies are placed only if the user allows it and has not taken the appropriate precautions in the browser settings.
- The cookies are then transmitted to the server along with the HTTP request each time the associated web page is called



# Cookie Information

- Cookie files essentially contain the following information:
  - Name and value of the cookie
  - The domain of the server and the path on the server to which the cookie should be sent.
  - An expiration date until which the cookie is valid.
  - A security flag can be used to optionally specify whether a cookie should only be sent on connections that use Secure Sockets Layer (SSL)



# Shortcoming of Cookies

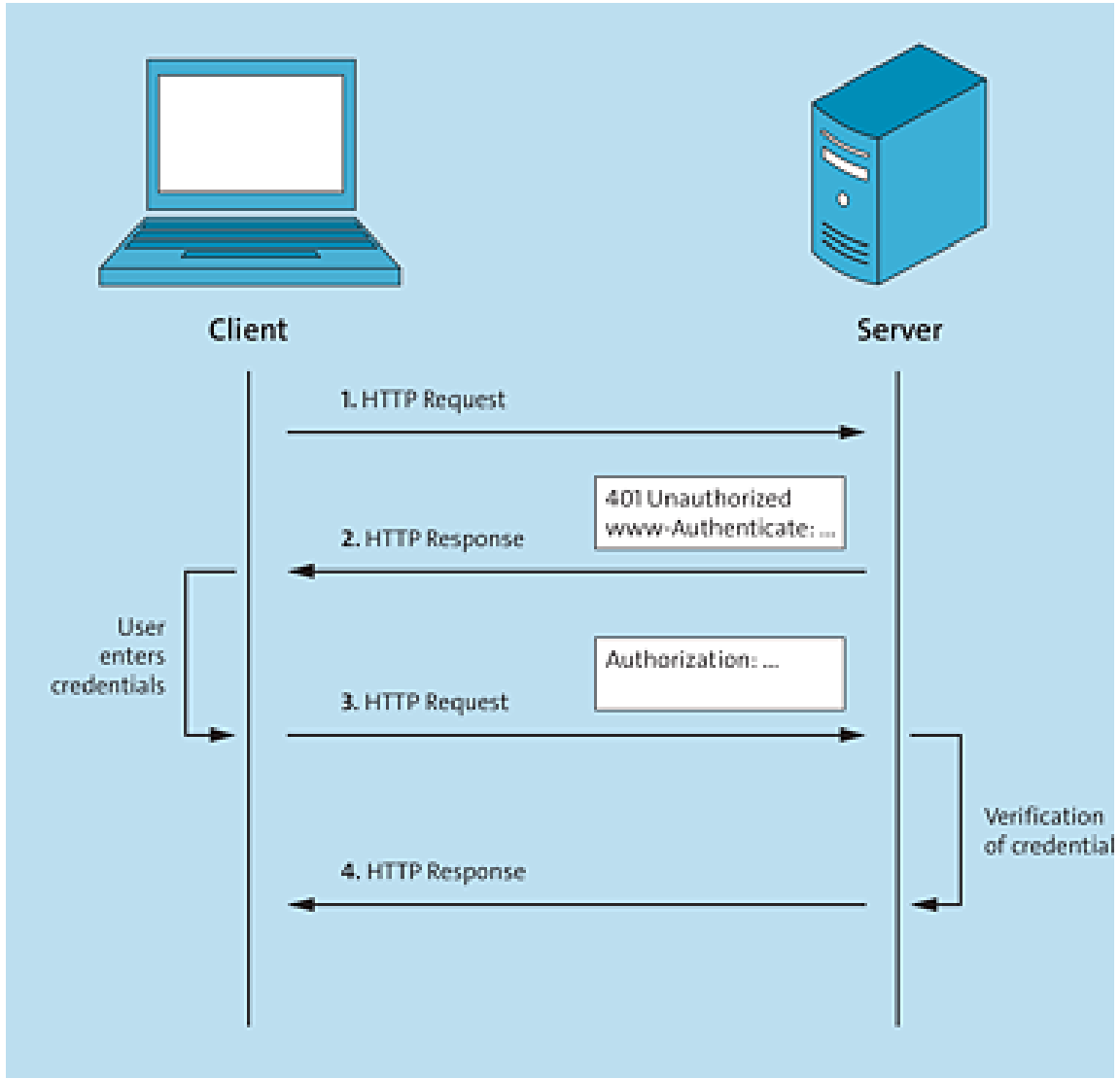
- Cookies are sent along with requests to the server via the cookie header.
- Cookies are suitable for many purposes, but they also have disadvantages:
- On one hand, cookies for the corresponding domain and path are sent along with each request, which has an overall impact on data volume.
- In addition, cookies that are sent via the HTTP protocol (and not via the secure HTTPS protocol) are transmitted unencrypted, which poses a security risk depending on the type of information transmitted.
- Also, the amount of data you can store via cookies is limited to 4 kilobytes.

# Authentication Revisit

- Authentication
  - Checking whether a user is who he or she claims to be
- Authorization
  - Checking whether a user is allowed to perform a certain action
- Various strategies exist for authentication:
  - Basic authentication
  - Session-based authentication
  - Token-based authentication

# Basic Authentication

1. The client sends an HTTP request to the web server, whereupon the server checks whether the requested resource is publicly accessible or requires authentication.
2. If authentication is required, the web server sends a corresponding HTTP response to the client, which contains status code 401 (“Unauthorized”) and the WWW-Authenticate header.
3. On the client side, this approach ensures that the credentials are requested. If the client is a browser, a corresponding browser dialog box is displayed to the user. Then, the client sends the request to the web server again, passing the credentials as the Authorization header.
4. The web server checks the credentials.

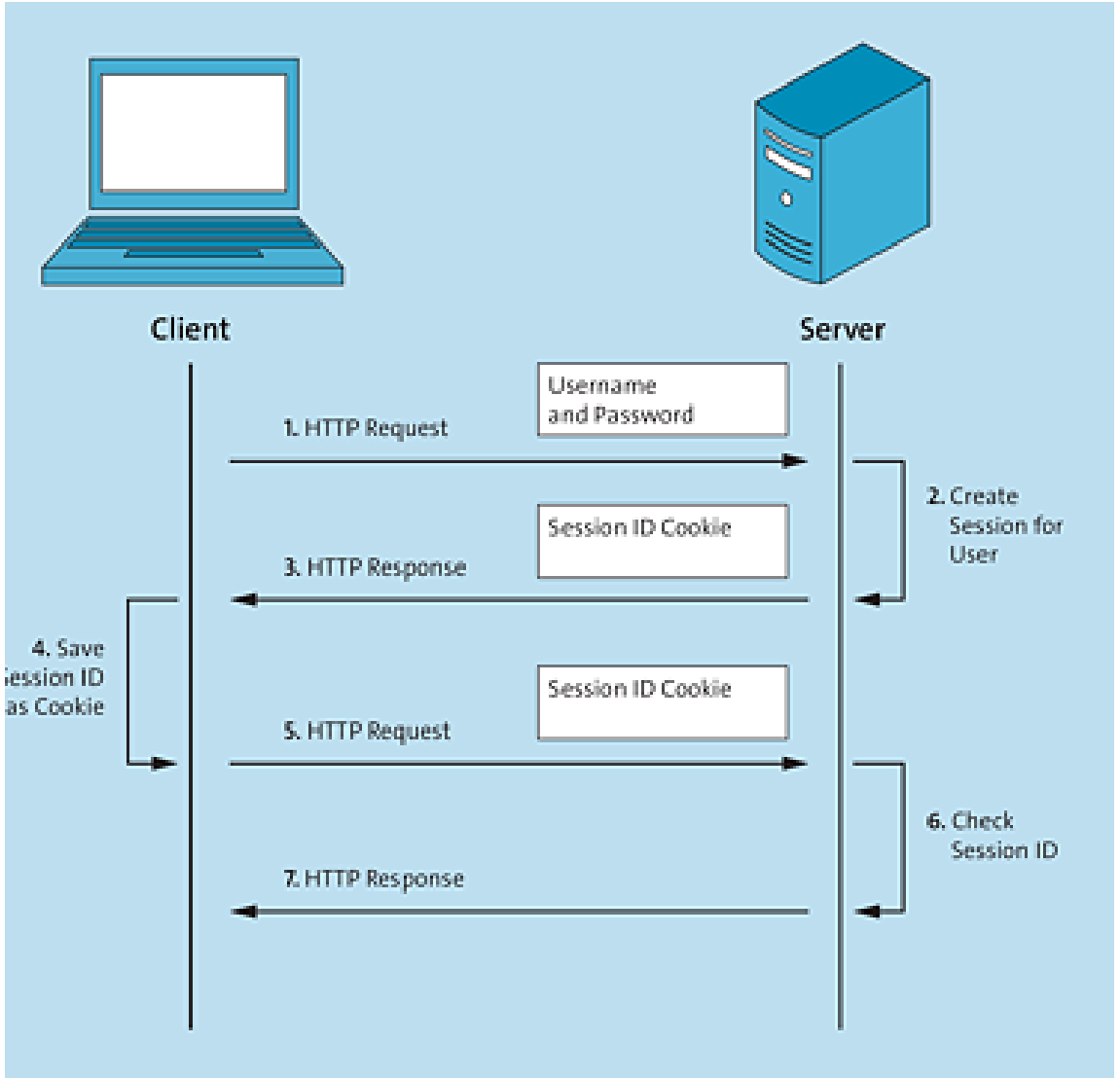


# Basic Authentication

- After successful authentication, the Authorization header, including the credentials, is sent to the server with any additional request (as long as the client is logged on to the server, of course).
- So, the credentials are not persisted anywhere on the server side.
- In the case of HTTPS, all headers and thus also the credentials are also transmitted in encrypted form.

# Session-Based Authentication

1. Once the user has logged on to the web server using a user name and password, the server creates a session for the user.
   - This session is uniquely identified by an ID, called the session ID, which, in turn, is sent back to the user by the web server and stored as a cookie in the user's browser.
2. As long as the user remains logged on to the web server, the cookie is sent along with each subsequent request.
3. The web server can then compare the session ID stored in the cookie with the session information stored on the server side to verify the identity of the user.

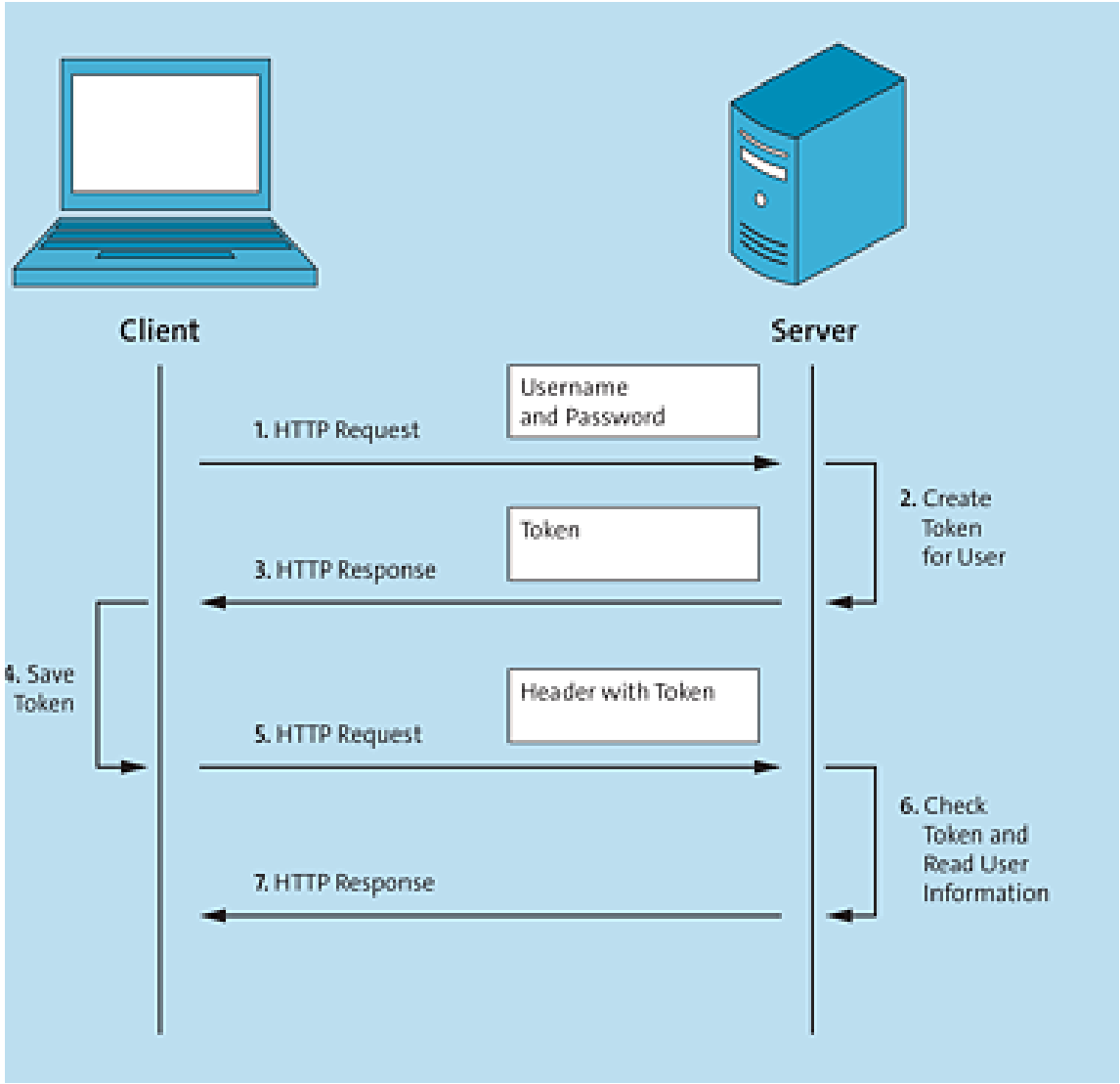


# Session-Based Authentication

- Session-based authentication is already more secure than basic authentication
- It still has several disadvantages: First, for applications that are used simultaneously by a large number of users, you must ensure that the server has sufficient memory to store the session IDs and associated sessions.
- Second, session-based authentication is not suitable for authentication against web services due to the underlying cookies.

# Token-Based Authentication

- Once the user has logged on to the web server using a user name and password, the web server creates a token (which can also contain the entire session information) and sends this token to the client.
- The client in turn stores the token (for example, in the local memory) and sends it along with each request to the web server as an Authorization header.
- The web server can then use the token to verify the identity of the client without having to memorize sessions internally.



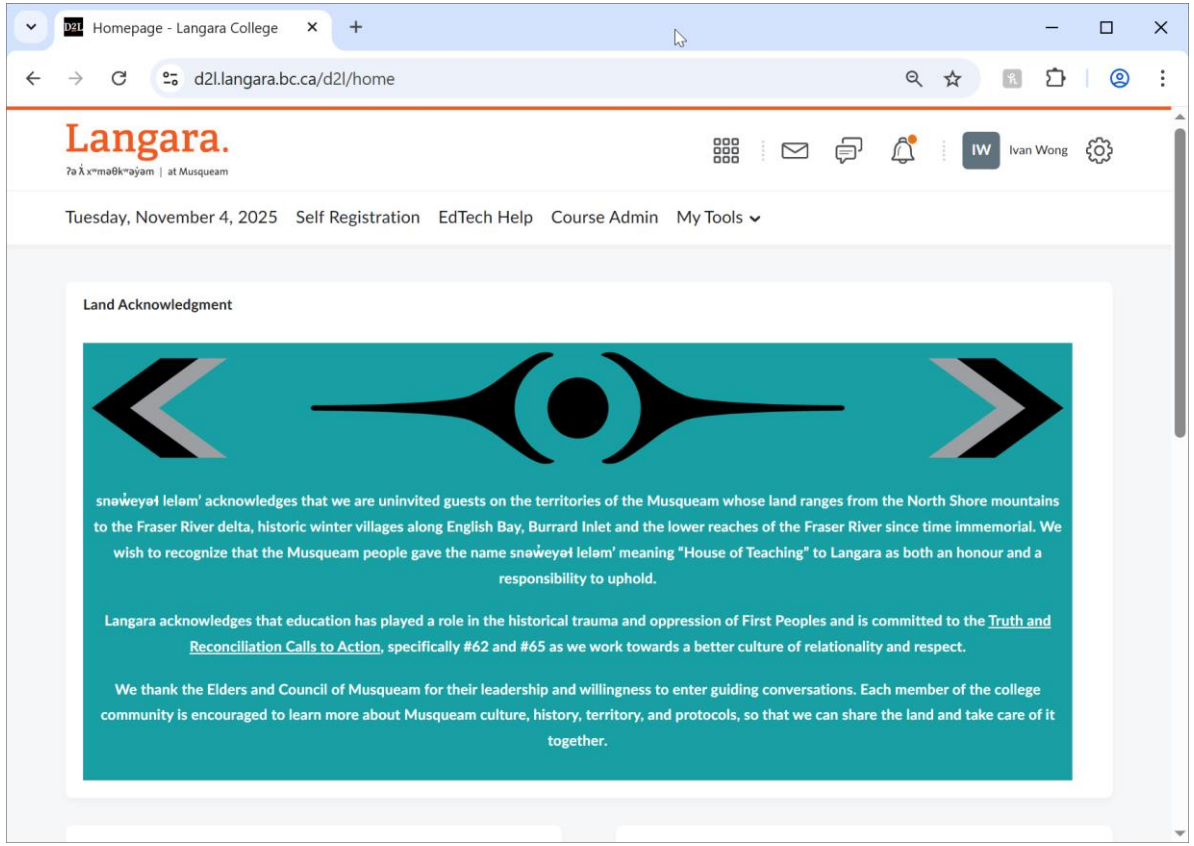
# Token-Based Authentication

- The difference between session-based authentication and token-based authentication is that the user's status is not stored on the server side (in a session) but on the client side (in the token).
- Since the sessions on the server side are omitted, the server does not have to use any storage space for this purpose. In other words, token-based authentication scales better than session-based authentication.
- Various standards exist for the implementation of tokens, for example, Simple Web Tokens (SWT), Security Assertion Markup Language Tokens (SAML Tokens), and JSON Web Tokens (JWT).
  - Today, JWT is common because it is more secure than SWT and also more compact than the XML-based SAML.

# Hands-on

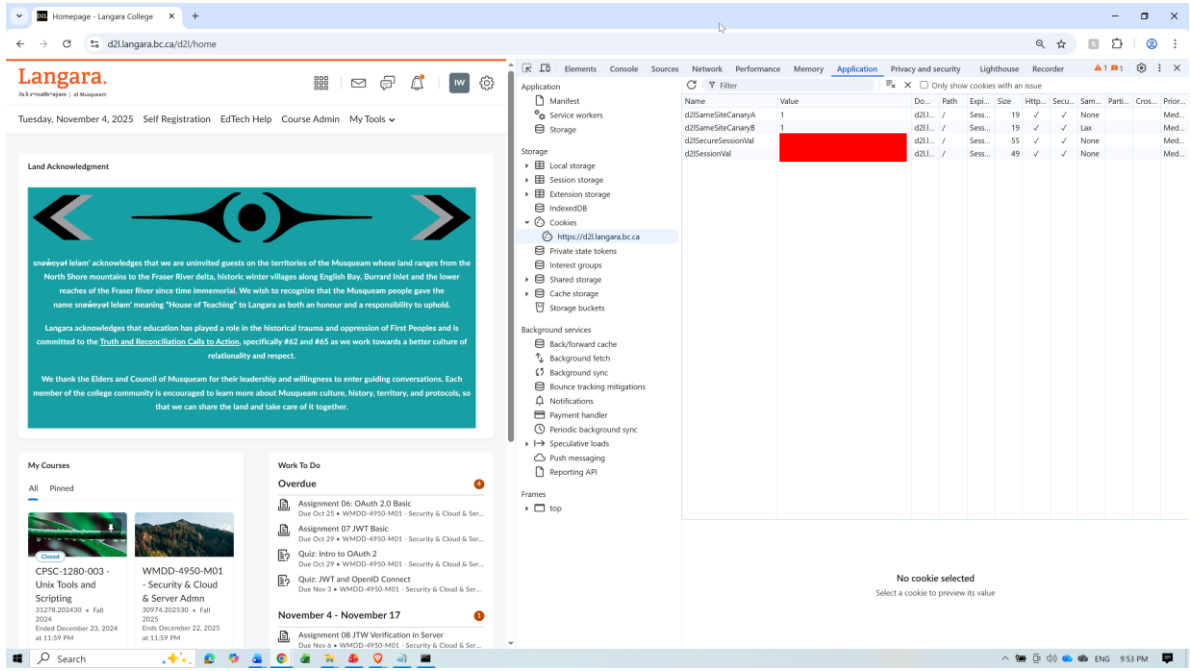
# Exploring Cookie in Chrome

- Go to a Website (e.g., d2l.langara.ca)
- Log in to the Website

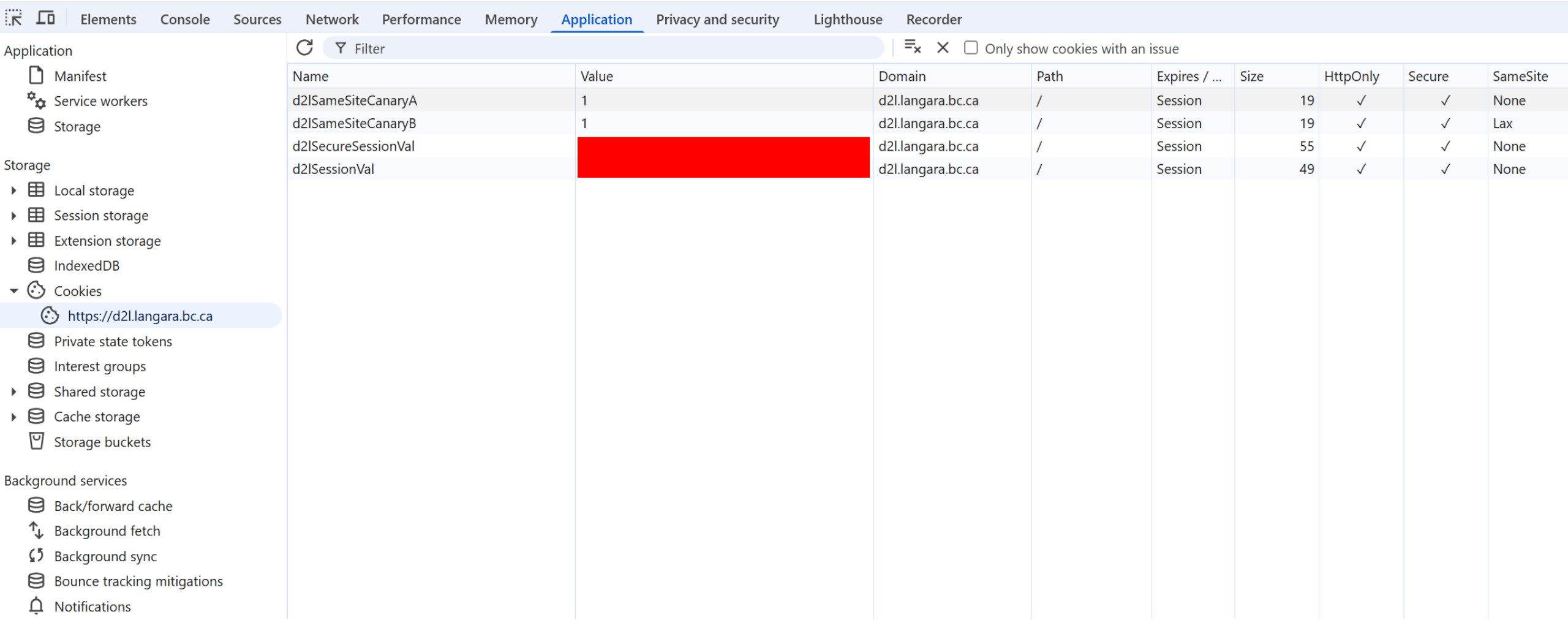


# Exploring Cookie in Chrome

- After login in, open Development Tools by pressing F12
- Go to Application Tab
- One the left menu, choose Cookies and highlight the Website.

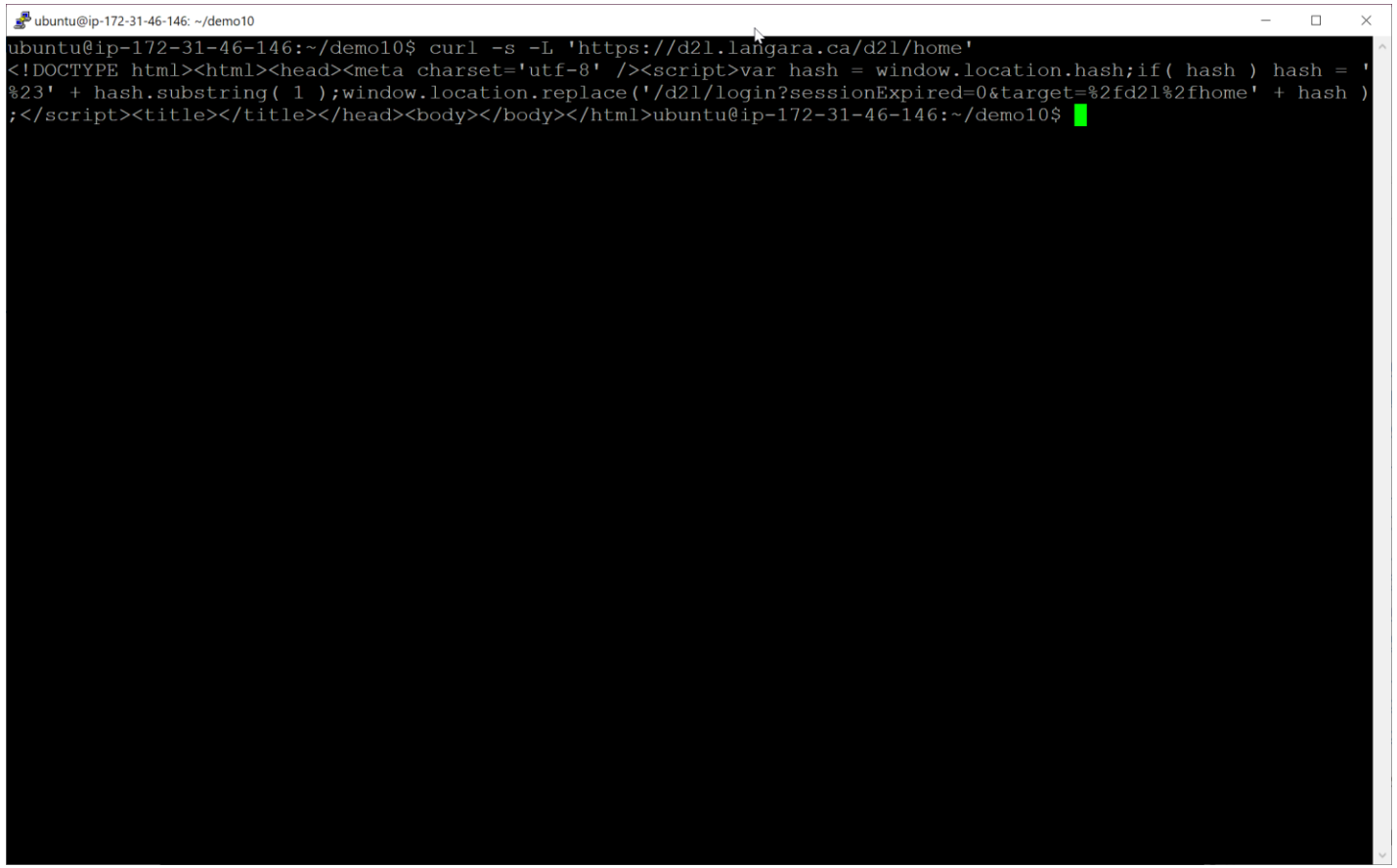


# Exploring Cookie in Chrome



# curl with Cookie

- curl -s -L 'https://d2l.langara.ca/d2l/home'



# curl with Cookie

- curl -s --cookie 'd2lSecureSessionVal=58I... ... ; d2lSessionVal=Uil... ...' -L 'https://d2l.langara.ca/d2l/home'

# Cookies with Bash CGI backend and JavaScript frontend

- File structure

```
/var/www/html/cookie_demo/
  index.html
  script.js

/usr/lib/cgi-bin/cookie_demo/
  set_cookie.sh
  read_cookie.sh
  clear_cookie.sh
```

# /var/www/html/cookie_demo/index.html

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Cookie Lab – Bash CGI + JS</title>
</head>
<body>
  <h1>Cookie Lab – Bash CGI + JavaScript</h1>

  <section>
    <h2>Set a cookie (server sets it)</h2>
    <form id="setForm">
      <label>Username:
        <input name="username" id="username" required>
      </label>
      <button type="submit">Set Cookie via CGI</button>
    </form>
    <pre id="setResult"></pre>
  </section>
  <section>
    <h2>Read cookie (server side)</h2>
    <button id="readServer">Ask server to read Cookie</button>
    <pre id="serverCookie"></pre>
  </section>
  <section>
    <h2>Read cookie (client side via document.cookie)</h2>
    <button id="readClient">Read document.cookie</button>
    <pre id="clientCookie"></pre>
  </section>

  <section>
    <h2>Clear cookie</h2>
    <button id="clearCookie">Clear cookie (server)</button>
    <pre id="clearResult"></pre>
  </section>

  <script src="script.js"></script>
</body>
</html>
```

# /var/www/html/cookie_demo/script.js

```
document.getElementById('setForm').addEventListener('submit', async (e) => {
  e.preventDefault();
  const username = document.getElementById('username').value;
  // Send as POST to CGI that sets cookie
  const res = await fetch('/cgi-bin/cookie_demo/set_cookie.sh', {
    method: 'POST',
    headers: { 'Content-Type': 'application/x-www-form-urlencoded' },
    body: `username=${encodeURIComponent(username)}`,
    credentials: 'same-origin' // important for cookie exchange
  });
  const text = await res.text();
  document.getElementById('setResult').textContent = text;
});
document.getElementById('readServer').addEventListener('click', async () => {
  const res = await fetch('/cgi-bin/cookie_demo/read_cookie.sh', { credentials: 'same-origin' });
  const text = await res.text();
  document.getElementById('serverCookie').textContent = text;
});
document.getElementById('readClient').addEventListener('click', () => {
  document.getElementById('clientCookie').textContent = document.cookie || '(no cookie visible via JS)';
});
document.getElementById('clearCookie').addEventListener('click', async () => {
  const res = await fetch('/cgi-bin/cookie_demo/clear_cookie.sh', { credentials: 'same-origin' });
  const text = await res.text();
  document.getElementById('clearResult').textContent = text;
});
```

# /usr/lib/cgi-bin/cookie_demo/set_cookie.sh

```
#!/usr/bin/env bash
# Reads POST body -> username=value and sets a cookie with that username.

# Read Content-Length and body
read -N "${CONTENT_LENGTH}" POST_DATA

# Parse username from urlencoded body
# decode function
urldecode() { echo -e "$(sed 's/+/ /g; s/%/\\x/g' <<<"$1")"; }

username=$(sed -n 's/.*username=\([^&]*\).*/\1/p' <<<"$POST_DATA")
username_decoded=$(urldecode "$username")

# Cookie attributes: Path=/ ; Max-Age=3600 (1 hour) ; HttpOnly omitted so JS can read
cookie_value=$(printf '%s' "$username_decoded" | sed 's/[^A-Za-z0-9._-]/_/g')

# Create headers (Set-Cookie). For demo: not HttpOnly so JS can read via document.cookie
echo "Content-Type: text/plain"
echo "Set-Cookie: lab_user=${cookie_value}; Path=/; Max-Age=3600; SameSite=Lax"
echo ""

echo "Server set cookie lab_user=${cookie_value}"
```

# /usr/lib/cgi-bin/cookie_demo/read_cookie.sh

```
#!/usr/bin/env bash
# read_cookie.cgi
# Read the HTTP_COOKIE environment variable and return it.
echo "Content-Type: text/plain"
echo ""
if [ -z "${HTTP_COOKIE}" ]; then
  echo "No Cookie header sent to server."
else
  echo "Cookie header received by server:"
  echo "${HTTP_COOKIE}"
  # parse lab_user value
  user=$(echo "${HTTP_COOKIE}" | tr ';' '\n' | sed -n
's/^[[:space:]]*lab_user=\(.*\)/\1/p')
  if [ -n "$user" ]; then
    echo ""
    echo "Parsed lab_user value: ${user}"
  fi
fi
```

# /usr/lib/cgi-bin/cookie_demo/clear_cookie.sh

```
#!/usr/bin/env bash
# clear_cookie.cgi
echo "Content-Type: text/plain"
# To clear cookie, set Max-Age=0
echo "Set-Cookie: lab_user=deleted; Path=/; Max-Age=0; SameSite=Lax"
echo ""
echo "Cookie lab_user cleared (server instructed browser to delete it)."
```

# Summary

- Cookie is a HTTP headers saving key-value pairs
- Cookie can streamline the authentication in Web applications
- Token-Based authentication integrates encrypted tokens (e.g., JWT) with cookie to provide a more secure and a more scalable authentication.
- Practise:
  - Browsing cookies in Chrome’s developer tools
  - Curl with cookies
  - Cookies in Javascript and BASH