



香港中文大學
The Chinese University of Hong Kong

COOKIE, SESSION, AND STORAGE

CSCI2720 2022-23 Term 1

Building Web Applications

Dr. Chuck-jee Chau and previous contributors
chuckjee@cse.cuhk.edu.hk

OUTLINE

- The struggle of application states in the stateless HTTP
- Cookies
- Session
- Local storage

APPLICATION STATES

- Our trouble: to remember states with the *stateless* HTTP protocol
- Examples
 - Whether a user is currently logged in
 - Who the current user is
 - How many items the current user prefers to view per page
 - What items are in the shopping cart
- HTTP server does not know the identity and simply entertain requests one by one

APPLICATION STATES

- Standalone programs
 - Keep states in variables (memory) or in files
- Web applications
 - Client side
 - Embedded in URL query
 - Cookies
 - Client-side storage
 - Tokens
 - Server side
 - Files
 - Database
 - Sessions

COOKIES

- HTTP cookies are data which a server-side script sends to a web client to persist for a period of time
- Cookies are embedded in **HTTP headers**
- Cookies are stored on the client device within the browser
- On every subsequent HTTP request, the web client automatically sends the cookies back to server
 - Unless cookie support is disabled, or the cookies have expired

COOKIES



TYPICAL USE OF COOKIES

- Personalization
 - Retaining user preferences (e.g., theme colour, number of items to show)
- Session Management
 - Keeping a session ID for identifying the user
- Tracking
 - Remembering activities of a user on a website
 - Tracking the user across websites using third-party cookies – a cookie set by a website not directly visiting
 - e.g., when a person visits a website with a Facebook Like button, Facebook can set cookies as the browser retrieves relevant files from Facebook

DRAWBACKS OF USING COOKIES

- Clients can temper with cookies as plain text files
 - Modify cookie (directly or with JavaScript), etc.
- One set of cookies per browser
 - All browser windows and tabs share the same cookies
 - Users using the same browser share the cookies
- Limited number of cookies (~20) per server/domain
- Limited data size (~4k bytes) per cookie
- Increased HTTP request header size for every request
 - Including requests for static resources
- ***Important:*** *Never store sensitive info in cookies!*

SETTING COOKIES

```
HTTP/1.0 200 OK
Content-type: text/html
Set-Cookie: foo=1234
Set-Cookie: bar=5678; Expires=Sat, 1 Jan 2023 01:00:00 GMT
Set-Cookie: baz=abcd; Expires=Wed, 1 Jan 2022 01:00:00 GMT
```

[page content]

e.g.: setting cookies in the response header

```
GET /somepage.html HTTP/1.1
Host: www.example.com
Cookie: foo=1234; bar=5678
```

e.g.: sending cookies in the request header

Note: Here cookie **baz** has expired and is thus deleted by the browser

COOKIE ATTRIBUTES

- Each cookie in the **Set-Cookie** header begins with a name-value pair, followed by some optional attributes
 - **Name**
 - Can be any ASCII characters except control characters, spaces, or tabs
 - Must not contain a separator character like these:
() < > @ , ; : \ " / [] ? = { }
 - **Value**
 - Similar rules apply but many frameworks automatically url-encode/decode the value
- See: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie>

COOKIE EXPIRATION

- **Expires**

- Expiry date and time of the cookie in GMT
- If not specified, cookie is treated as a *session cookie*
 - i.e., the cookie is deleted when the client shuts down
 - Now many web browsers allow *restoring* a session, reopening all tabs when using the browser again
 - Cookies will also be present, and it simply seems the browser has never closed
- When an expiry date is set, the time and date is relative to the client, but not the server

- **Max-Age**

- Number of seconds until the cookie expires
 - A zero or negative number will expire the cookie immediately
- If both (**Expires** and **Max-Age**) are set, **Max-Age** will have precedence

COOKIE SCOPE

- **Domain**

- Specifies those hosts to which the cookie will be sent
- If not specified, defaults to the host portion of the current URL
- If a domain is specified, subdomains are always included
 - e.g., if a domain is specified as **example.com**, then **web1.example.com** and **web2.example.com** are included

- **Path**

- Indicates a URL path that must exist in the requested resource before sending the Cookie header
 - e.g., if **path=/docs**, then **/docs**, **/docs/Web/**, or **/docs/Web/HTTP** will all be matched

COOKIE SECURITY

- **Secure**

- If exists, the cookie will only be sent to the server when a request is made using SSL and the HTTPS protocol

- **HttpOnly**

- If exists, the cookie will not be accessible via JavaScript through the **Document.cookie** property, the **XMLHttpRequest** and Request APIs to mitigate attacks against cross-site scripting (XSS)

SETTING COOKIES IN EXPRESS

- **res.cookie(name, value [, options])**
 - Sets cookie name to **value**
 - **options** is an object with the following properties:
 - **expires** – A date in GMT
 - If not specified or set to 0, a session cookie is created
 - **maxAge** – In milliseconds
 - Convenient way to set the value of expires to **(now() + maxAge)**
 - Does not correspond to the actual cookie's **Max-Age** attribute
 - **domain** – Defaults to the domain name of the app
 - **path** – Defaults to /

SETTING COOKIES IN EXPRESS

- **secure** – Whether cookie is sent only via HTTPS
- **httpOnly** – Whether cookie is accessible only at server-side
- **signed**
 - Indicate if the cookie should be *signed*
 - Signed cookies will have a signature attached to it, so that a server-side script can detect if the cookie has been modified by the client

SETTING AND RETRIEVING COOKIES

```
// cookie-parser is installed with npm separately  
const cookieParser = require('cookie-parser');  
app.use(cookieParser());  
  
app.get('/', (req, res) => {  
  // The cookies values are accessible through req.cookies  
  if (req.cookies['visited'] === undefined) {  
    res.cookie('visited', 'yes', { maxAge: '1200000' });  
    res.send('Your first visit!');  
  } else {  
    res.send('Welcome back!');  
  }  
});
```


DELETING COOKIES

```
app.get('/', (req, res) => {  
  
  // the API to clear a cookie  
  res.clearCookie('bar');  
  
  // An expiration date in the past also deletes a cookie  
  res.cookie('bar', '', { expires: new Date(1) });  
  
  // Alternative approach to set expires  
  res.cookie('bar', '', { maxAge: -1000; });  
  
  ...  
});
```

HTTP SESSION

- How can we ensure two HTTP requests are *related*?
 - *e.g., initiated from the same client by the same user*
 - IP can be dynamically assigned to different machines
 - A browser on a computer can be shared by multiple users
- The same copy of server-side scripts is used to serve all requests
- How can these scripts share data between *related* requests?
 - The share data can be login status or items in a shopping cart

HTTP SESSION

- Typical approach to relate multiple HTTP requests
 1. Generate a unique session ID for each user
 - in the 1st visit or after the user has successfully logged in
 2. Keep the session ID at the client side
 3. For each subsequent request, embed the session ID in the request
 - Embedded in cookies or query string

USING SESSION

- The first time a web client visits a server, the server sends a unique session ID to the web client for the client to keep
 - Session ID is typically stored in a cookie
 - Session ID is used by the server to identify the client
- For each session ID created, the server also creates a storage space
 - Typically a map-like data structure
 - Server-side scripts that receive the same session ID share the same storage space
- Different implementations have different strategy to delete expired session storage space

USING SESSION

```
sessionID = Retrieve session ID (e.g., from cookies)

if (sessionID does not exist or has already expired) {
    if (sessionID has expired) {
        Destroy or clean up the session data;
    }
    sessionID = Create a new session ID
    Create and initialize the corresponding session data structure
} else { // Session is still active
    Restore the session data structure with the saved data
    (e.g., from memory, file or database)
}

// Application code can now read/write session data here ...

// At the end of the current request-response cycle
if (session data has been modified)
    Save the modified session data (to memory, file or database)
```

EXPRESS SESSION

- **req.session.destroy(callback)**
 - Destroys the session and unsets **req.session**, and then call the given **callback** function
- **req.session.id**
 - Unique ID associated with the current session
- **req.session.cookie**
 - An object storing the cookie attributes of session ID
 - Defaults to
`{ path: '/', httpOnly: true, secure: false, maxAge: null }`
- See: <https://www.npmjs.com/package/express-session>

ENABLING SESSION IN EXPRESS

```
const express = require('express');
const app = express();
// Require npm module "express-session"
const session = require('express-session');

// Enable session support for all requests
app.use(session({
  secret: 'foobarbazz', // A value for signing cookie ID
  cookie: { maxAge: 1200000 } // Expires in 20 min
  // If not set, defaults to null (until browser closes)
})));
```

```
app.get('/', (req, res) => {
  let S = req.session;
  // If current user is a returning visitor
  if (S.visitedCount !== undefined) {
    S.visitedCount++;
    res.send('<p># of visits: ' + S.visitedCount + '</p>' +
      '<p>expires in: ' + (S.cookie.maxAge / 1000) +
      's</p>');
  } else { // First timer
    S.visitedCount = 0;
    res.redirect('/'); // Force reloading this page
  }
});
const server = app.listen(3000);
```

CLIENT-SIDE WEB STORAGE

- **`window.localStorage`** and **`window.sessionStorage`**
- Allows a web application to store data within the browser via JavaScript
- The storage allowed is at least **5MB**, and the stored data is never transferred to the server
- All pages from the **same origin** can store and access the data in the same local storage
- Data are stored as name-value pairs
 - Value need to be a string or a JSON encoded string

WEB STORAGE

// storing a piece of content

<https://codepen.io/chuckjee/pen/wvo0mqm>

```
let text = document.querySelector("#text").innerText;
```

```
localStorage.setItem('content', text);
```

// retrieve the stored content

```
alert( localStorage.getItem('content') );
```

// Since it's all plain text, you can easily manipulate

// the local storage using other API

WEB STORAGE

- **Storage.setItem(name, value)** sets the item under **name** to be **value**
 - Also allowed in these forms:
 - **Storage.name = value**
 - **Storage[name] = value**
- **Storage.getItem(name)** returns the item under **name**
- **Storage.removeItem(name)** removes the item under **name**
- **Storage.clear()** empties the entire storage object for the domain
- **sessionStorage** *expires* when browser is closed
- **localStorage** persists *always*

A QUICK SUMMARY

- ***Cookies***

- Retain data ***at the client side*** for a period of time
- Automatically return to the server on every request
- Not suitable for keeping sensitive data or large amount of data

- ***Session***

- Keep temporary data ***at the server side*** that are shared among server-side scripts for related requests

- ***Local storage***

- Storing arbitrary data at the client side



MDN HTTP Cookies

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies>

MDN Web Storage

[https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API/Using the Web Storage API](https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API/Using_the_Web_Storage_API)

READ FURTHER...