香港中文大學
The Chinese University of Hong Kong

CSCI2720 - Building Web Applications

Lecture 15: Cookie, Session, and Storage

Dr Colin Tsang

Outline

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

Application states

Cannot identify request / Don't know who the user is
Don't know the user have visited before /
Cannot determine which content is more suitable

- Our trouble: to remember states with the *stateless* HTTP.
- 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

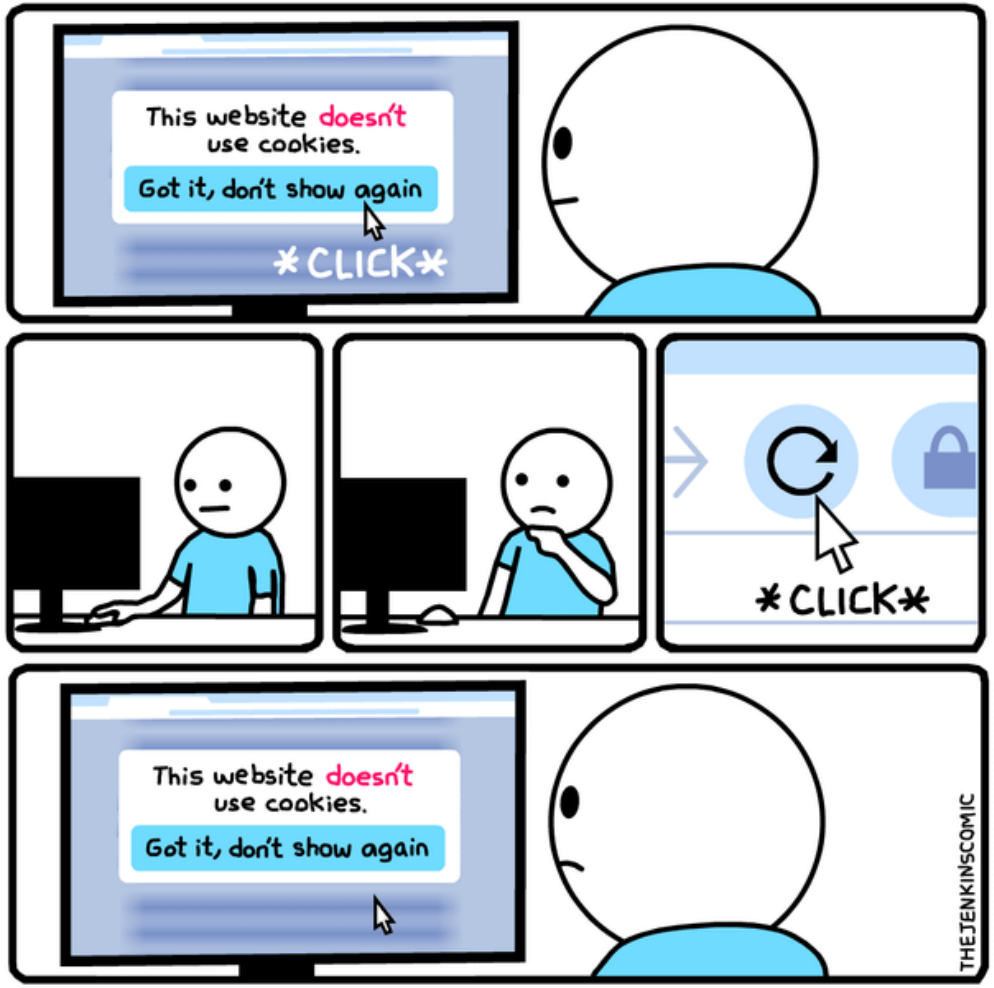
Cookies

Cookies is from the server, stored in the client machine

- 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 Stored in local 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

- European Union cookie law (GDPR, 2018), see: https://termly.io/resources/articles/cookie-law/
- Meanwhile, in HK: https://www.pcpd.org.hk/english/news_events/newspaper/newspaper_201911.html

Cookies

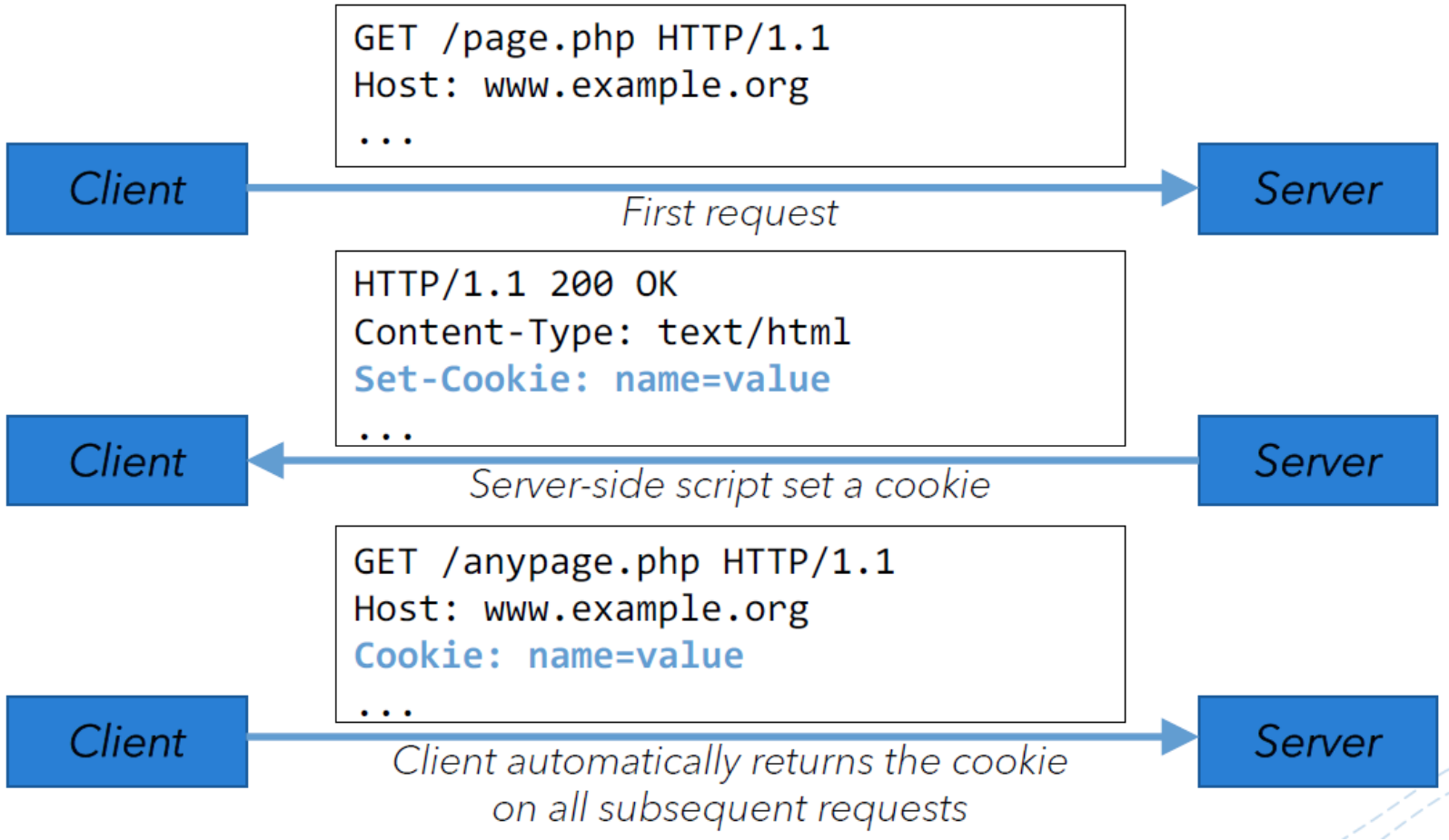


https://www.reddit.com/r/ProgrammerHumor/comments/l5gg3t/this_website_doesnt_use_cookies/

Cookies

Both Client & Server have control on cookies

When client has 1st request, server will ask for 1st cookie



Client can change / deleted

There is cookie everytime

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 CUHK's website with a Facebook like button (the button is not from CUHK), Facebook can set cookies as the browser retrieves relevant files from Facebook.
    - This usage of cookies is usually under law regulation.

Drawbacks of using cookies

- Client 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
- Limited data size (~4k bytes) per cookie
- Increased HTTP request header size for every request
  - Including requests for static resources
- ***Important:*** never store sensitive data in cookies!

Setting cookies

- Setting cookies in the response header:

```
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]
```

The cookie will be automatically deleted after expired date by the browser

- Setting cookies in the request header:

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

Cookie attributes

- Each cookies in the **Set-Cookie** header begins with a *name-value pair*, follows by some 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 framework automatically url-encode the value.
- See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie

Cookie expiration

- **Expires**
  - Expiry data 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 tabs when using the browser again
    - Cookies will also be present, and it simply seems the browser has never closed
- **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 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 scope

- **Domain**
  - Specifies the domain to which the cookie should be sent by the browser
    - E.g., you may use it when your server host multiple domain
      - *example1.com/* and *example2.com/*
  - 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
    - i.e., the cookie with comeback to server when the request contain this path
      - E.g., *http://server-address/event/123/loc/shb130*
    - E.g., if *path=/docs*, then */docs*, */docs/Web*, or */docs/Web/HTTP* will all be matched

Setting cookies in Express

- **res.cookie(name, value [, options])**
  - Set cookie *name* to *value*
  - *Options* is an object with the following properties:
    - **expires**
    - **maxAge**
      - In milliseconds
    - **domain**
    - **path**

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 cookies

```
app.get('/', function (req, res) {
  // Set a cookie to be returned by the client when the URL
  // matches *.example.com/admin/*, which expires in an hour
  res.cookie('foo', '123', { Only name-value pairs are required
    domain: 'example.com',
    path: '/admin',
    expires: new Date( Date.now() + 3600000 )
  });

  // Set a session cookie for the current domain
  res.cookie('bar', '456');

  ...
});
```

Optional

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 machine
  - 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 shared data can be login status or items in a shopping cart

HTTP session

- Typical approach to relate multiple HTTP requests
  - Generate a unique session ID for each user
    - In the 1st visit or after the user has successfully logged in
  - Keep the session ID at the client side
  - 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 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)
```

Using session in Express

- **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
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

✡ Use “name” - “value” pair

- 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

Further readings

- 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