Cookies and Authentication

Spring 2024

HTTP is stateless

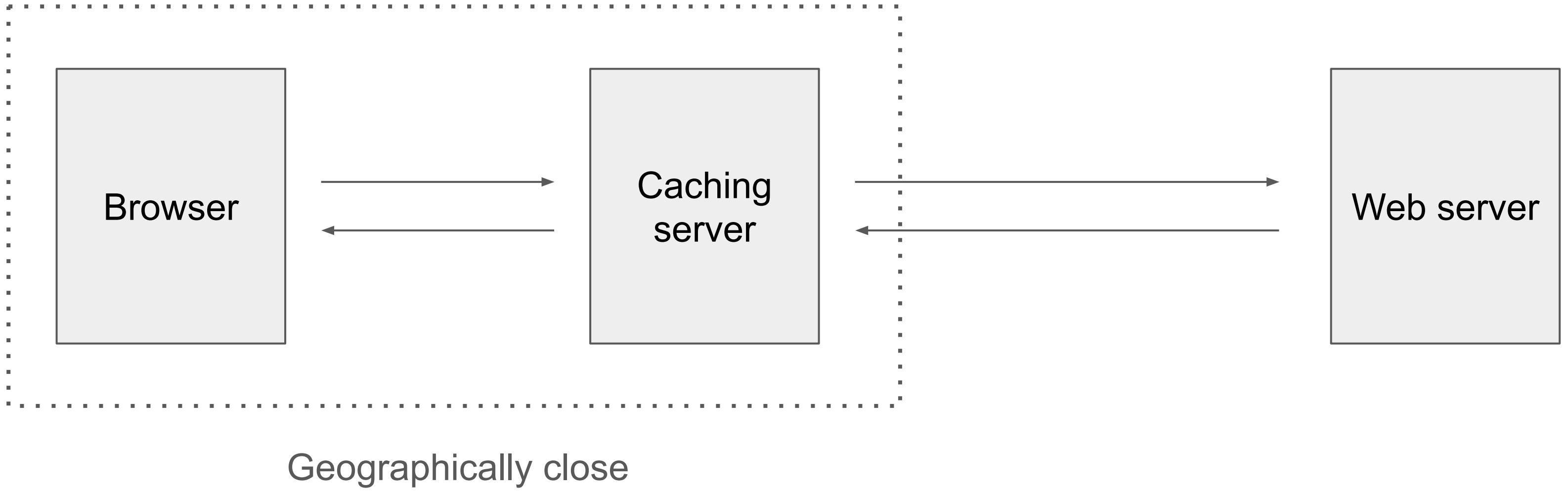
Every HTTP request is its own thing

- server should be able to respond to a request using only information contained in the request
- no "context" maintained between calls

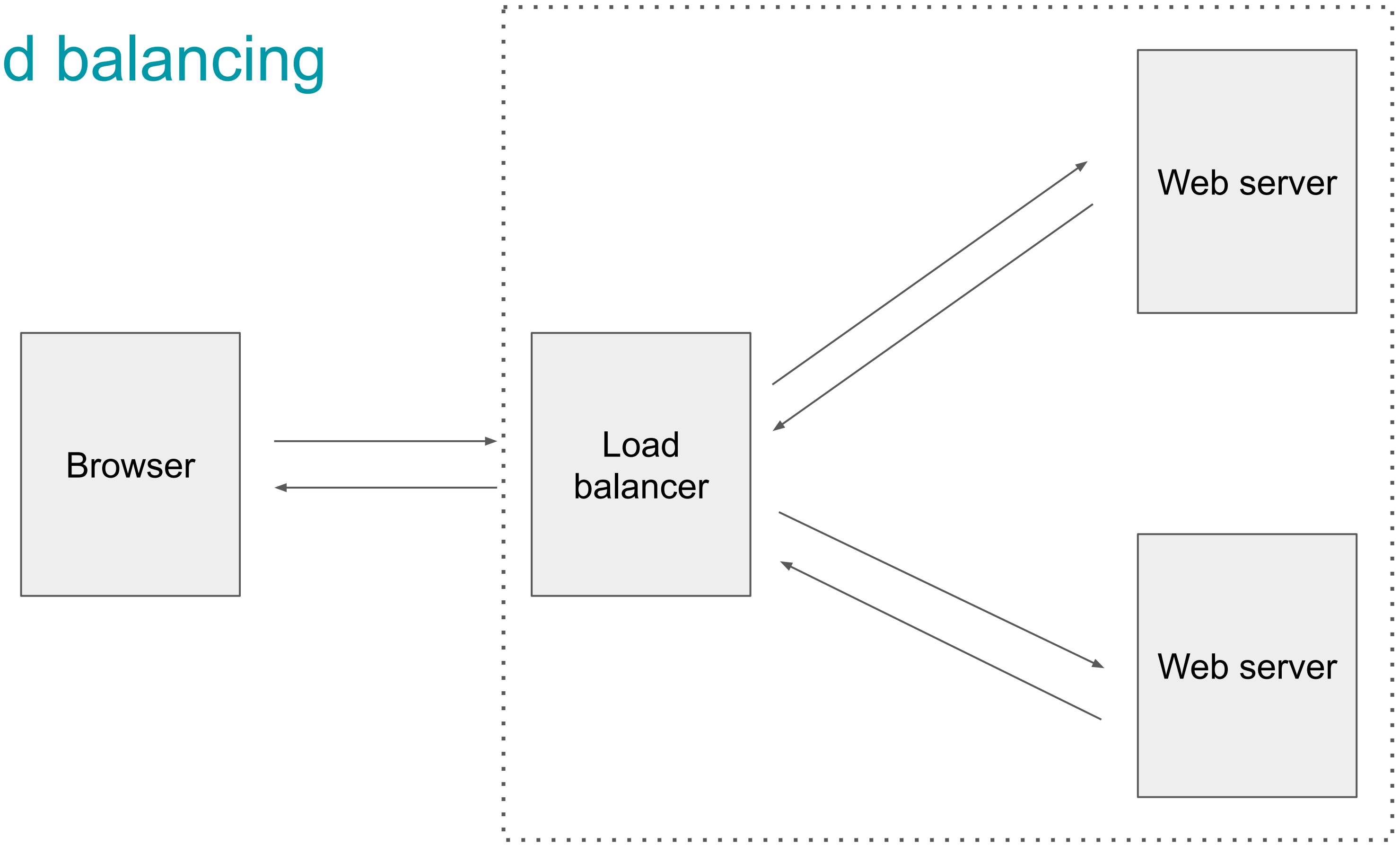
Advantages

- Can (mostly) cache responses to GET requests
- Can use load balancing servers

Caching



Load balancing



What about sessions?

Users are often interested in the notion of a **session**

- session = a span that represents a related sequence of interactions
- what defines a session is that **state** is maintained throughout the session
  - e.g., shopping cart, logged-in status

HTTP does not maintain context (= state) between calls

So we need to maintain state "manually" at the application level

Cookies

A **cookie** is just some state ***on the browser*** associated with a web server

- a cookie has a name and a value (a string)

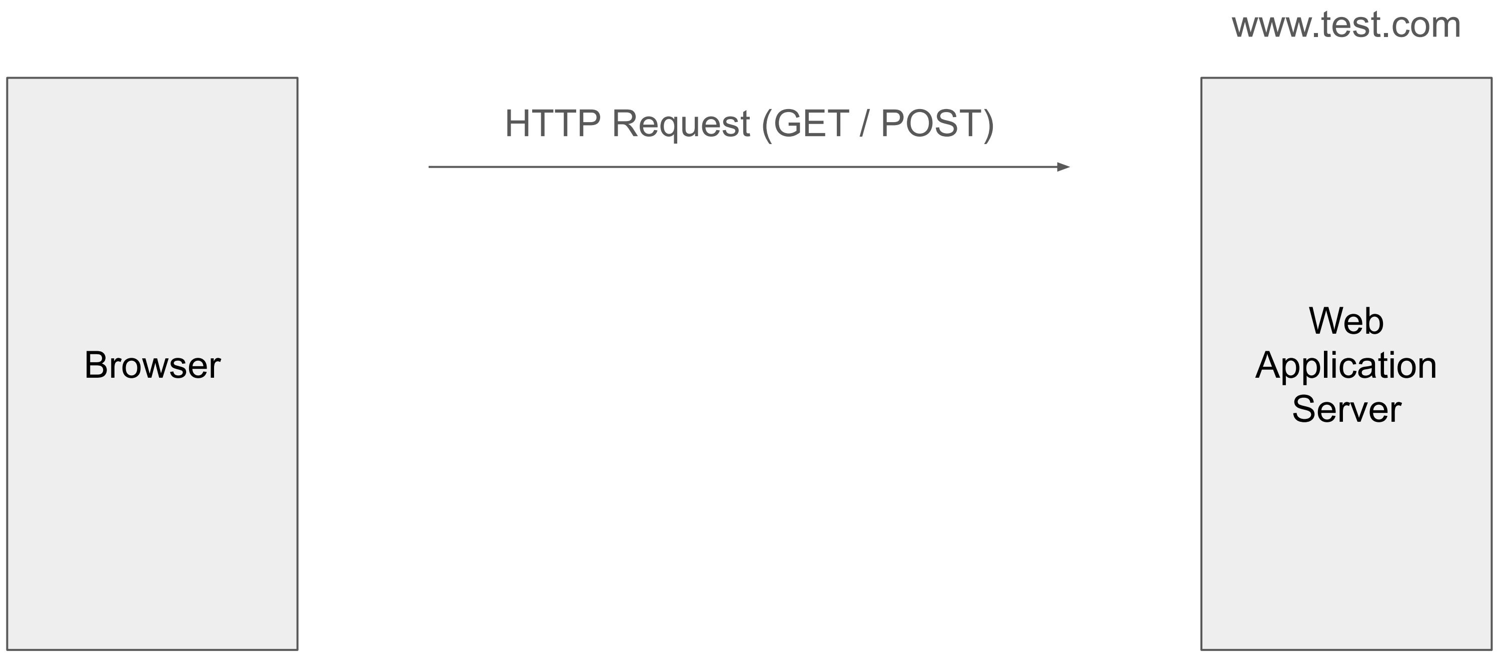
Created by a web server and sent to the client browser and kept in the browser

Automatically sent to the web server with every request from the browser

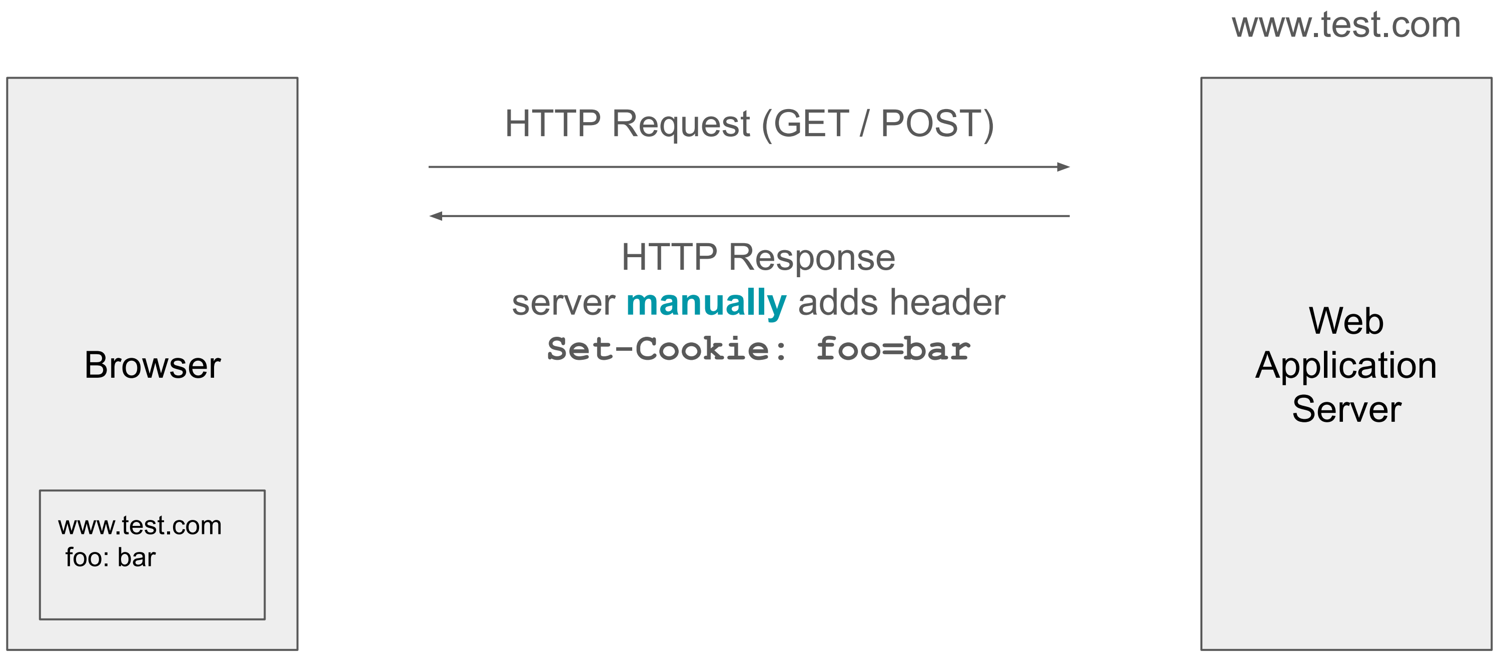
Web server may send new values for a cookie (= state update)

You can use cookies to carry context (= state) to the server

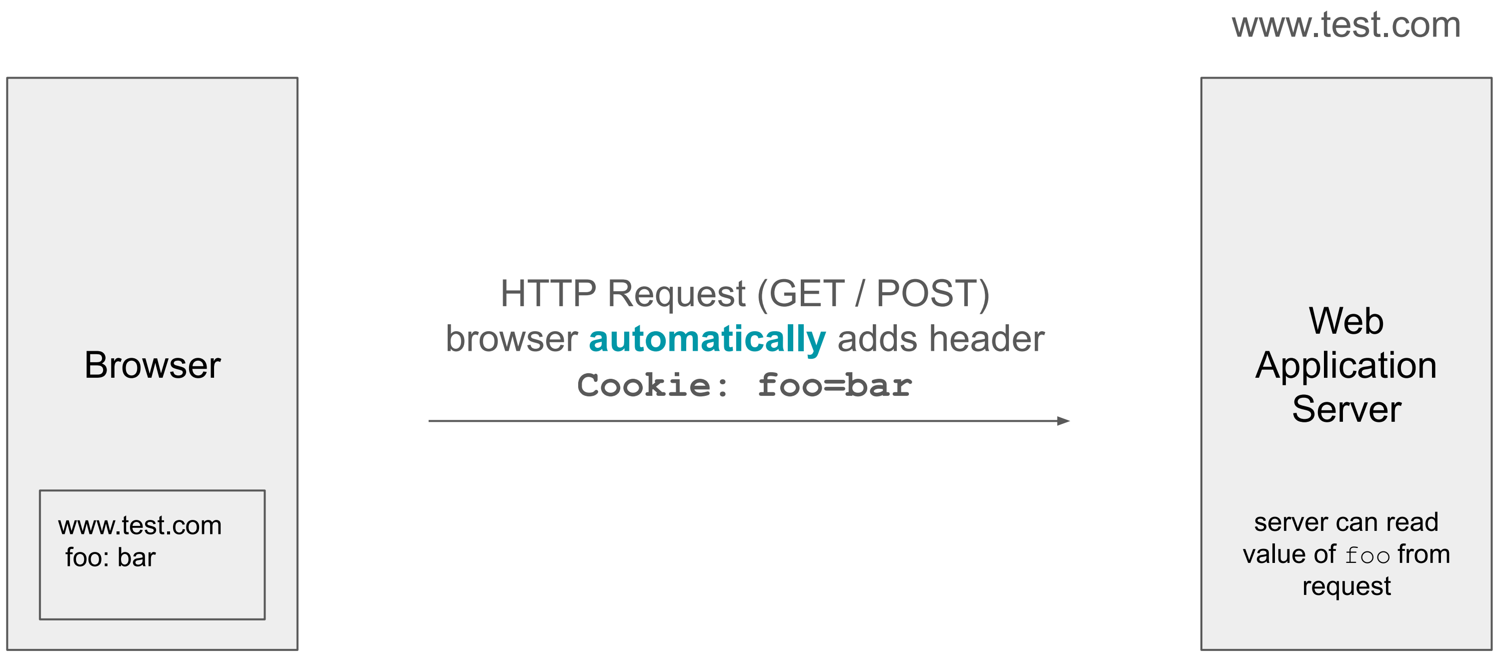
Cookies


www.test.com
HTTP Request (GET / POST)
Browser
Web Application Server

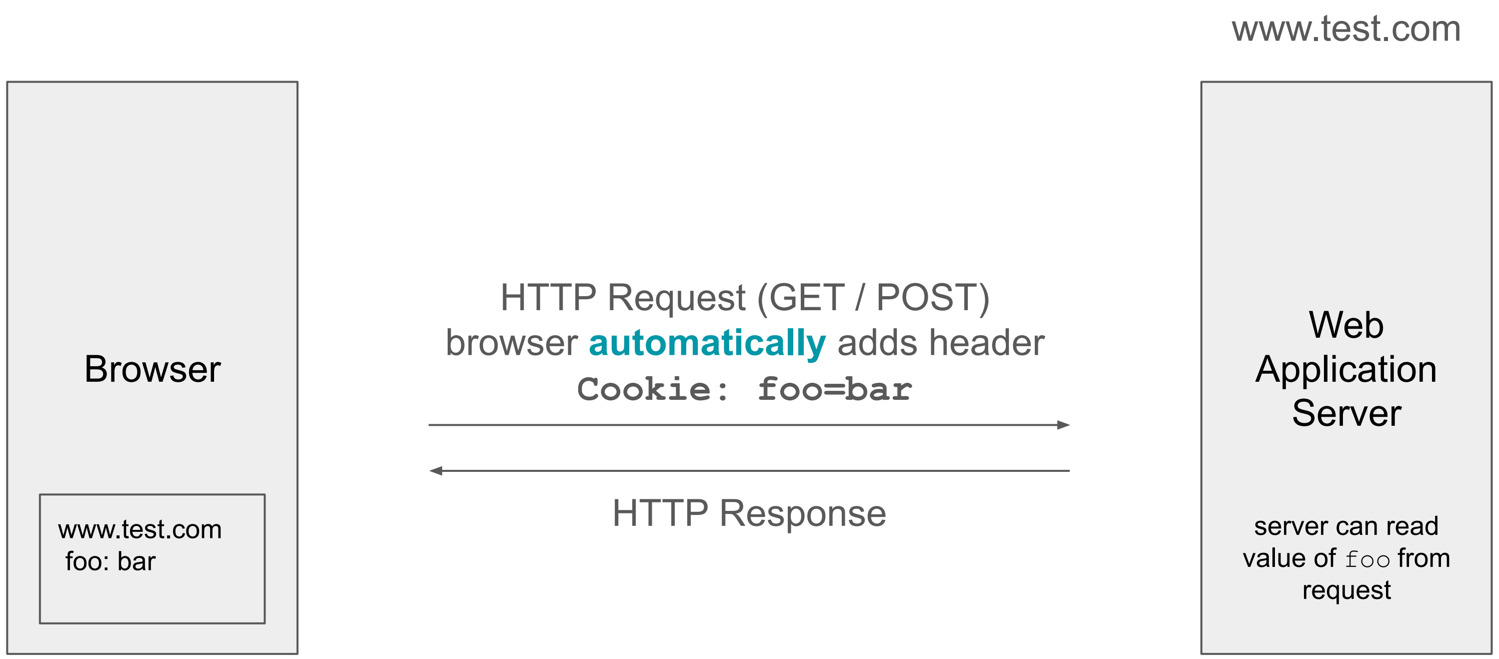
Cookies


www.test.com
HTTP Request (GET / POST)
HTTP Response
server **manually** adds header
`Set-Cookie: foo=bar`
Browser
www.test.com
foo: bar
Web Application Server

Cookies


www.test.com
Browser
www.test.com
foo: bar
HTTP Request (GET / POST)
browser **automatically** adds header
`Cookie: foo=bar`
Web Application Server
server can read value of `foo` from request

Cookies


www.test.com
Browser
www.test.com
foo: bar
HTTP Request (GET / POST)
browser **automatically** adds header
`Cookie: foo=bar`
HTTP Response
Web Application Server
server can read value of `foo` from request

Demo

Cookie for storing preferences

Cookies limitations

Cookie:

- limited to 4 Kb
- limited to 50 cookies per server
- limited to 3000 cookies total
- potential for staleness
- storing state on the client may be a security risk (even if encrypted)

Better to store session data on the server

But… HTTP is stateless?

Storing session data on the server

Server holds session state in a **dictionary**

key → state

Server puts the key (a session ID) in a cookie

- client sends the session ID every time it sends an HTTP request
- server can access session state by looking up the session ID in the dictionary

Problem: doesn't work in a multi-server scenario

Solution: store the dictionary in a shared database (Redis)

Authentication

Intuitively:

- provide credentials to a server to prove that you are who you claim you are

How do you prove your identity?

- know a secret (password)
- possess a unique device (authentication token)
- biometrics

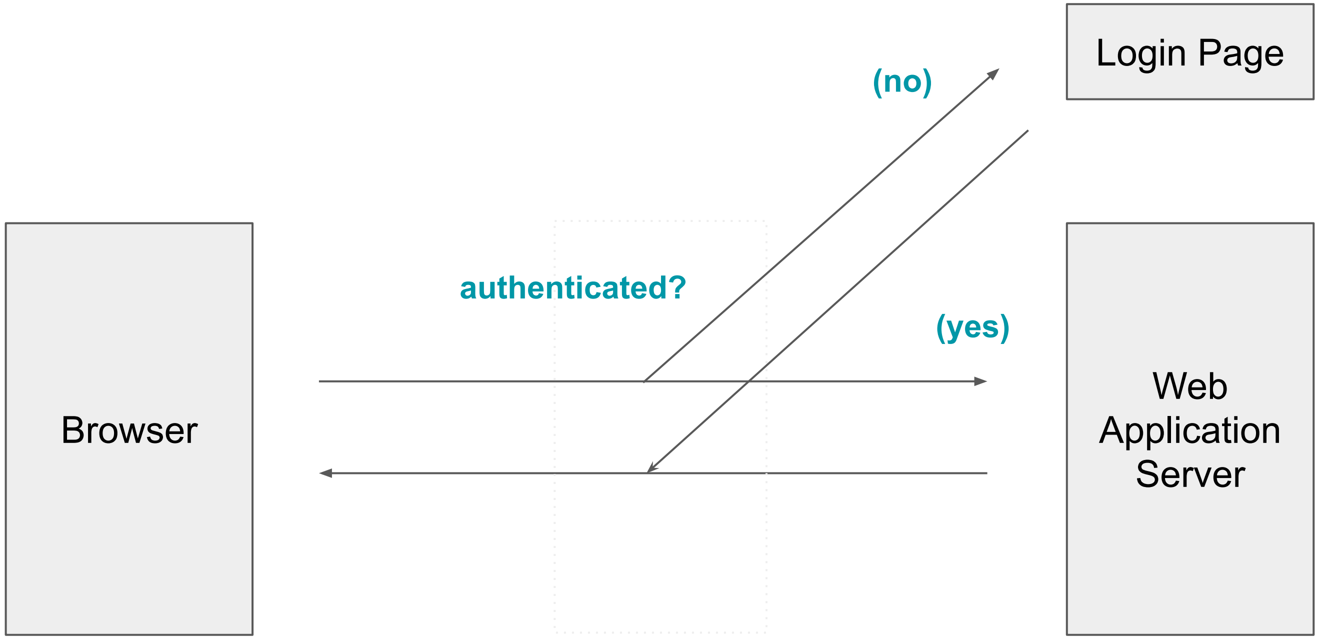
Authentication

Implementing authentication in a system is a two-steps process:

1. Proving your identity at the beginning:
   - page that lets you enter credentials (login page)
   - needs to be accessible even if you're not authenticated (duh!)

2. Keep a token that shows you've proved your identity
   - cookie that you can only have if you've proved your identity

Common setup



Session authentication versus token authentication

Same question as last time:

- do we keep authentication information on the browser (**token auth**)
    - generally done with JWT (JSON Web Token)
    - useful for SSO [login to one site, provide access to other sites]

- or do we keep authentication information on the server (**session auth**)
    - session state kept on the server
    - sessionId kept in the browser

Demo

Content of session cookie:
    sessionId

Session state on the browser:
    for each sessionId:
        name of (authenticated) user
        can also add user preferences

A user is authenticated if they have a session cookie with a valid sessionId