Cookies, sessions and third-party authentication

Claudia Hauff

TI1506: Web and Database Technology

ti1506-ewi@tudelft.nl

Learning objectives

- **Decide** for a given usage scenario whether cookies or sessions are suitable
- **Explain and implement** cookie usage
- **Explain and implement** session usage
- **Implement** third-party authentication

Introduction to cookies and sessions

Recall: HTTP

- HTTP is a **stateless** protocol
- Every HTTP request contains **all information** needed to serve a response
- The server is not required to keep track of the requests issued
- Advantage: simplifies the server architecture
- Disadvantage: clients have to resend **the same information** in every request

We do a lot of things requiring a known state …

- bol.com keeps your *Winkelwagentje* full, even when you leave the website
- StatCounter (tracking users' visits) can exclude a particular visitor from being tracked
- JavaScript games can keep track of the game's status when you re-visit the game (website)
- Websites can tell you how many times you have visited

Cookies cannot …

- **Execute** programs
- Access information from a **user's hard drive**
- Generate spam
- Be **read by arbitrary parties**
  - Only the server setting the cookie can access it
  - But: beware of **third-party cookies**

Cookies

Cookies and sessions are ways to **introduce state** on top of the stateless HTTP protocol.

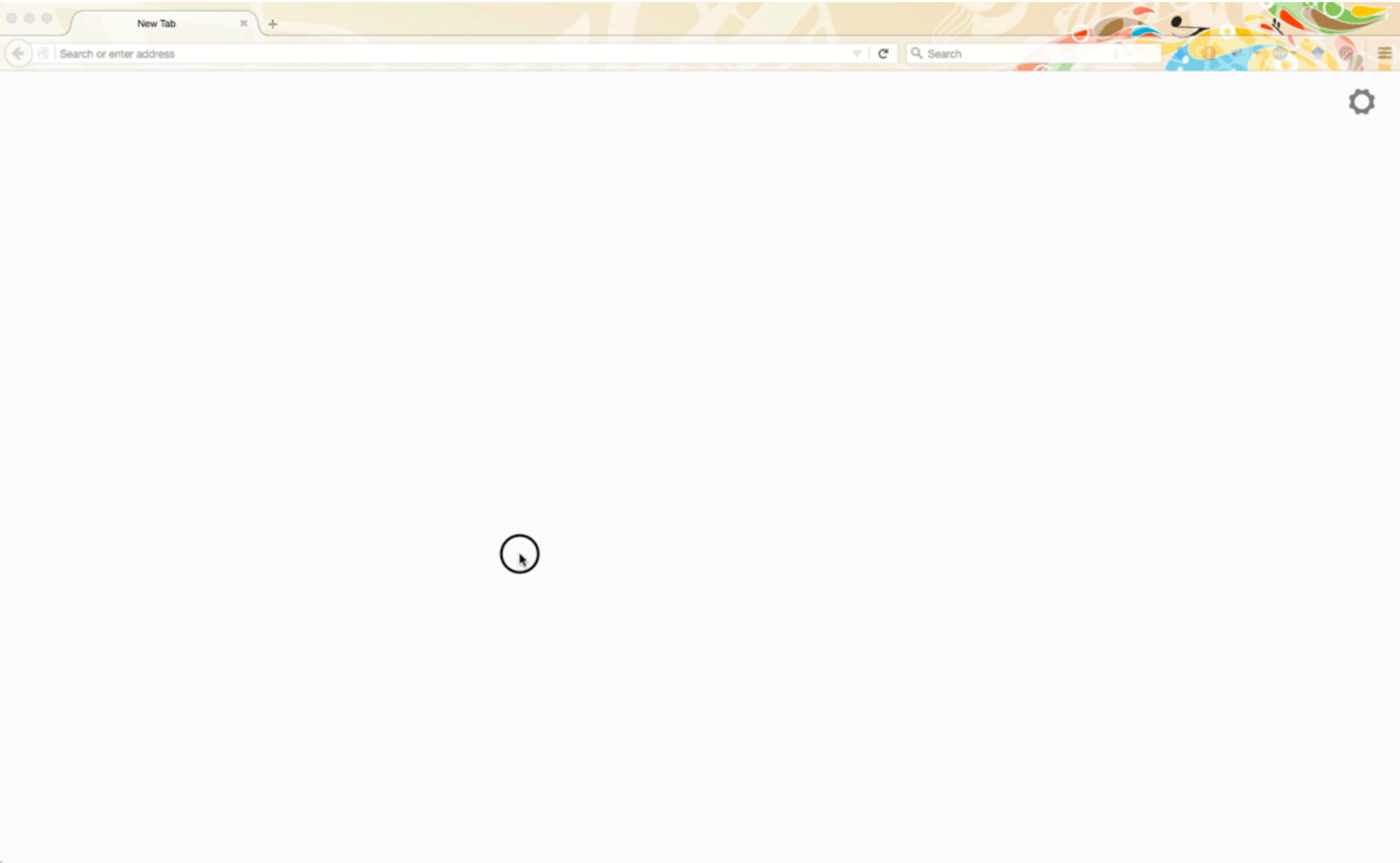
Cookie: a short amount of text **(key/value) sent by the server** and **stored by the client** for some amount of time.

Minimum client storage requirements (`RFC6265` from 2011)

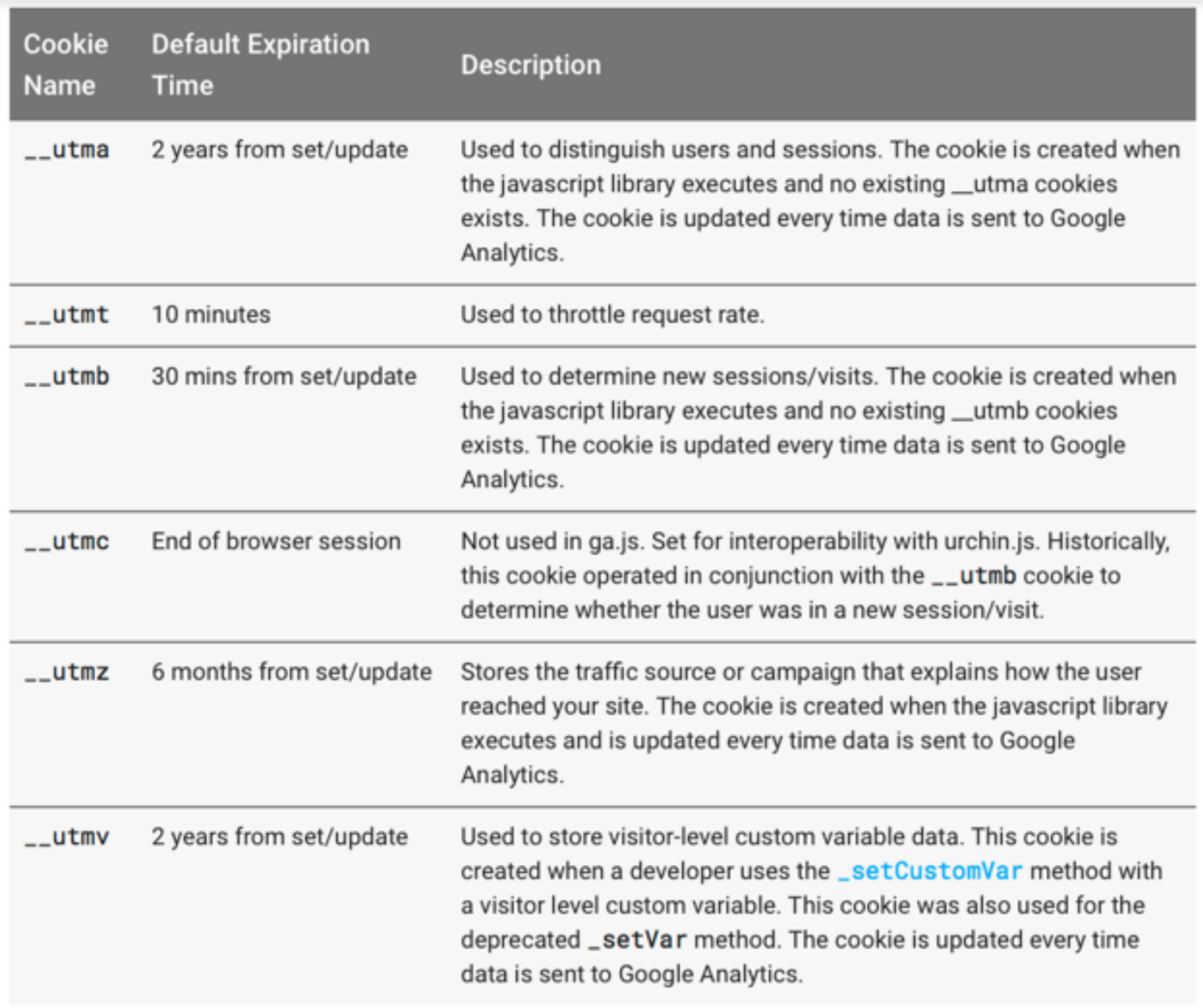
- Store at least 4096 bytes per cookie
- Store at least 50 cookies per domain
- Store at least 3000 cookies total.

“Servers SHOULD use **as few** and **as small** cookies as possible to avoid reaching these implementation limits and minimise network bandwidth”

Where can I find the cookies?



Why am I seeing the same cookies everywhere?



Cookie Name	Default Expiration Time	Description
`__utma`	2 years from set/update	Used to distinguish users and sessions. The cookie is created when the javascript library executes and no existing __utma cookies exists. The cookie is updated every time data is sent to Google Analytics.
`__utmt`	10 minutes	Used to throttle request rate.
`__utmb`	30 mins from set/update	Used to determine new sessions/visits. The cookie is created when the javascript library executes and no existing __utmb cookies exists. The cookie is updated every time data is sent to Google Analytics.
`__utmc`	End of browser session	Not used in ga.js. Set for interoperability with urchin.js. Historically, this cookie operated in conjunction with the `__utmb` cookie to determine whether the user was in a new session/visit.
`__utmz`	6 months from set/update	Stores the traffic source or campaign that explains how the user reached your site. The cookie is created when the javascript library executes and is updated every time data is sent to Google Analytics.
`__utmv`	2 years from set/update	Used to store visitor-level custom variable data. This cookie is created when a developer uses the `_setCustomVar` method with a visitor level custom variable. This cookie was also used for the deprecated `_setVar` method. The cookie is updated every time data is sent to Google Analytics.

Google Analytics › Tracking › analytics.js

Cookie & session basics

A very **old** piece of Web technology! Developed in **1994**.

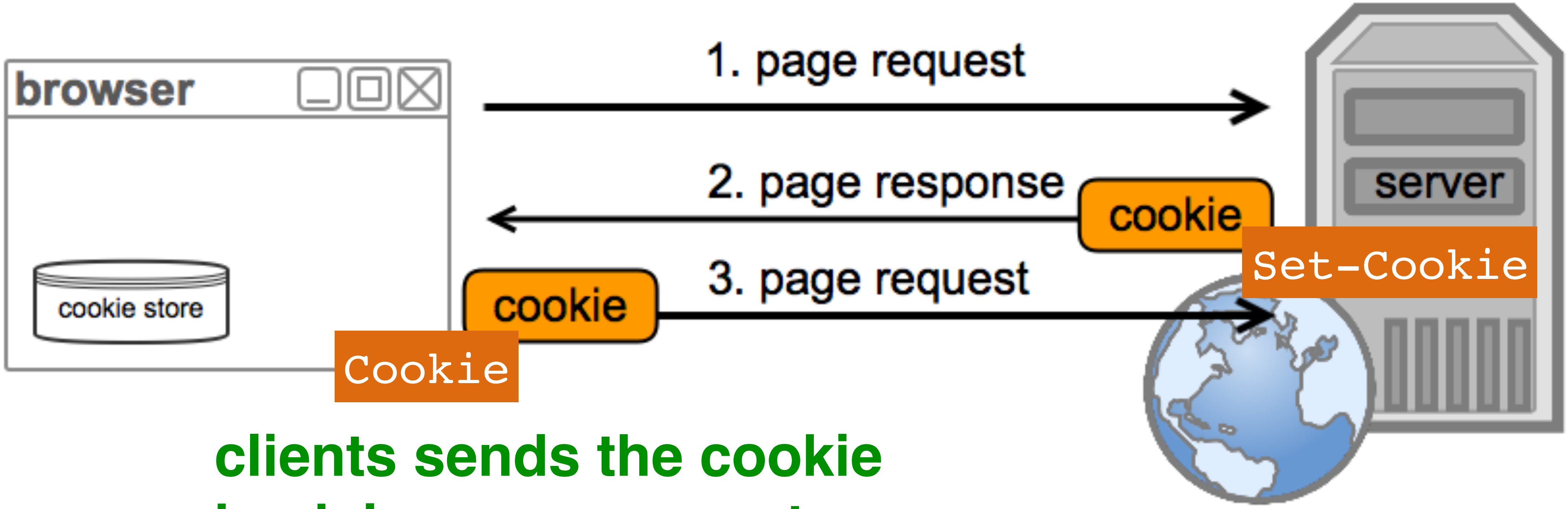
- Cookies are **visible** to the users (who make the effort)
  - By default, stored in the clear
- Clients (users, i.e. you!) can **delete/disallow** cookies
- Cookies can be **altered by the client**
  - Opens up a line of attack: **servers** should not send sensitive information in simple cookies
- **Sessions** are preferable to cookies
  - Sessions themselves make use of cookies
  - Cookie usually contains a single value (session ID), the rest is stored on the server

A word of warning: RFC6265

> “This document defines the HTTP `Cookie` and `Set-Cookie` header fields. These header fields can be used by HTTP servers **to store state** (called cookies) at HTTP user agents, letting the servers maintain a **stateful session** over the mostly stateless HTTP protocol.
>
> **Although cookies have many historical infelicities that degrade their security and privacy, the `Cookie` and `Set-Cookie` header fields are widely used on the Internet.** ”

Cookie basics

**server sends a cookie once;
resends when key/value changes**



**clients sends the cookie
back in every request**

- Encoded in the **HTTP header**
- Web frameworks have designated methods to work with cookies
- Cookies are **bound** to a **site domain name**, are only sent back on requests to this specific site

What can be stored in cookies?

- Cookies are the **server's short term memory**
- Information in a cookie is decided by the server

- **Examples**:
  - History of page views
  - Settings of form elements (can also be fully client-side)
  - Tracking of user’s UI preferences
  - …

Session vs. persistent cookies

Not the tab or window!

- **Session** (or transient) cookies:
  - Exist in memory only, are deleted when the browser is closed
  - Cookies are session cookies if **no expiration date** is defined.
- **Persistent** cookies:
  - Cookies remain intact after the browser is closed
  - Have a **maximum age**
  - Are **send back to the server** as long as they are **valid**

Cookie fields

- **`Name=value`** the only **required** field
- **`Expiration`** date (UNIX timestamp) or **`max age`**
- **`Domain`** the cookie is associated with; cookies can only be assigned to the **same domain** the server is running on
- **`Path`** the cookie is applied to (automatic wildcarding): `/` matches all pages, `/todos` all pages within `todos`, etc.
- Three flags (add a layer of robustness)
  - **`Secure`**
  - **`httpOnly`**
  - **`Signed`**

Making cookies more robust

- **Secure** cookies:

  Secure setting via **HTTP**: the cookie will **not** be **sent**

  - Setting the secure attribute ensures that the cookies are sent via HTTPS (i.e. encryption across the network)
- **HttpOnly** cookies:
  - Cookies are **not accessible** to non-HTTP entities (e.g. **JavaScript**)
  - Minimises the threat of cookie theft
  - Applies to session management cookies, not browser cookies

  Hash Message Authentication Code

- **Signed** cookies (appended `HMAC[value]`):
  - Ensures that the **value** has not been **tampered** with by the client (offers **no privacy**)

```
s%3Amonster.TdcGYBnkcvJsd0%2FNcE2L%2Bb8M55geOuAQt48mDZ6RpoU
```

Cookie domain

- **Origin**: **request domain** of the cookie (a cookie is always applicable to its origin server)
  `GET http://www.my_site.nl/todos` → `www.my_site.nl`
  - Port or scheme can differ, the received cookie is also applicable to `https://www.my_site.nl:3005`
- **Domain attribute**: a cookie's `Domain` attribute has to cover the origin domain
  - If not set, a cookie is only applicable to its origin domain (a cookie from `www.my_site.nl` is not applicable to `my_site.nl`)
  - If set, a cookie is applicable to the domain listed in the attribute and all its **subdomains**

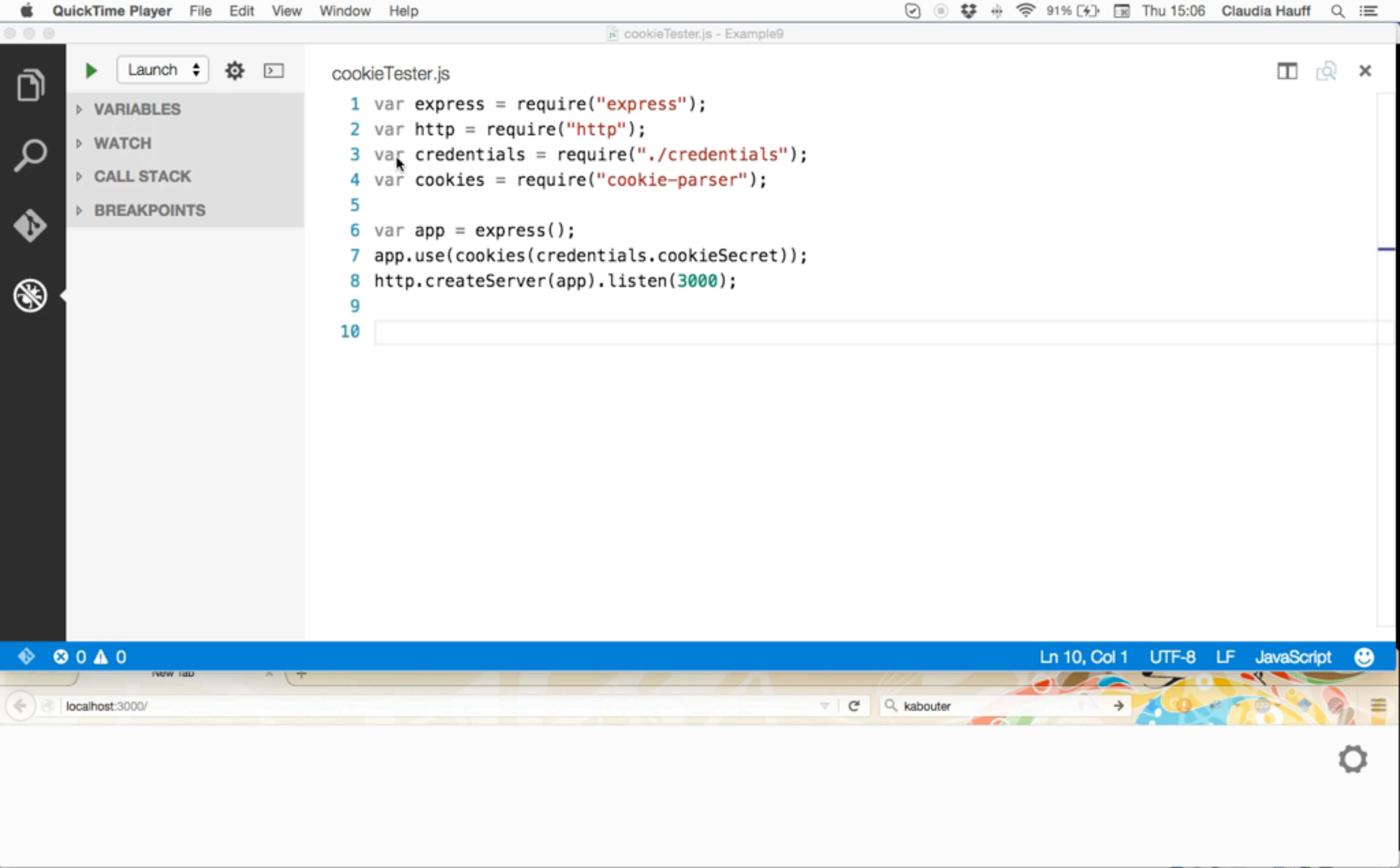
```
GET http://www.my_site.nl/todos
Set-Cookie: name=value; Path=/; Domain=my_site.nl
```

applicable to `www.my_site.nl` `todos.my_site.nl` `serverA.admin.todos.my_site.nl`

Domain attribute cannot be a public suffix (.com, .nl, …)

Example 9

Example: (1) send cookies to a client that requests them; (2) list all cookies sent by the client.



```
var express = require("express");
var http = require("http");
var credentials = require("./credentials");
var cookies = require("cookie-parser");

var app = express();
app.use(cookies(credentials.cookieSecret));
http.createServer(app).listen(3000);

```

npm install cookie-parser

Cookies in express

```
var express = require("express");
var http = require("http");
var credentials = require('./credentials.js');
var cookies = require("cookie-parser");

var app = express();
app.use(cookies(credentials.cookieSecret));
http.createServer(app).listen(port);

app.get("/sendMeCookies", function (req, res) {
    console.log(“Handing out cookies");
    res.cookie("chocolate", "kruemel");
    res.cookie("signed_choco", "monster", { signed: true});
    res.send();
});

app.get("/listAllCookies", function (req, res) {
    console.log("++++ unsigned ++++");
    console.log(req.cookies);
    console.log("++++ signed ++++");
    console.log(req.signedCookies);
    res.send();
});
```

cookie-parser middleware

creating cookies

signing a cookie

reading cookies

Cookies in express

```
npm install cookie-parser
```

```
module.exports = {
    cookieSecret: 'abc'
};
```

```
var express = require("express");
var http = require("http");
var credentials = require('./credentials.js');
var cookies = require("cookie-parser");

var app = express();
app.use(cookies(credentials.cookieSecret));
http.createServer(app).listen(port);

app.get("/sendMeCookies", function (req, res) {
    console.log(“Handing out cookies");
    res.cookie("chocolate", "kruemel");
    res.cookie("signed_choco", "monster", { signed: true});
    res.send();
});

app.get("/listAllCookies", function (req, res) {
    console.log("++++ unsigned ++++");
    console.log(req.cookies);
    console.log("++++ signed ++++");
    console.log(req.signedCookies);
    res.send();
});
```

Accessing and deleting cookies in express

- **Accessing the value** of a particular key/value pair:
  ```
  var val = req.signedCookies.signed_choco;
  ```
  cookie key

- **Deleting** a cookie:
  ```
  res.clearCookie('chocolate');
  ```
  delete in the **response**!

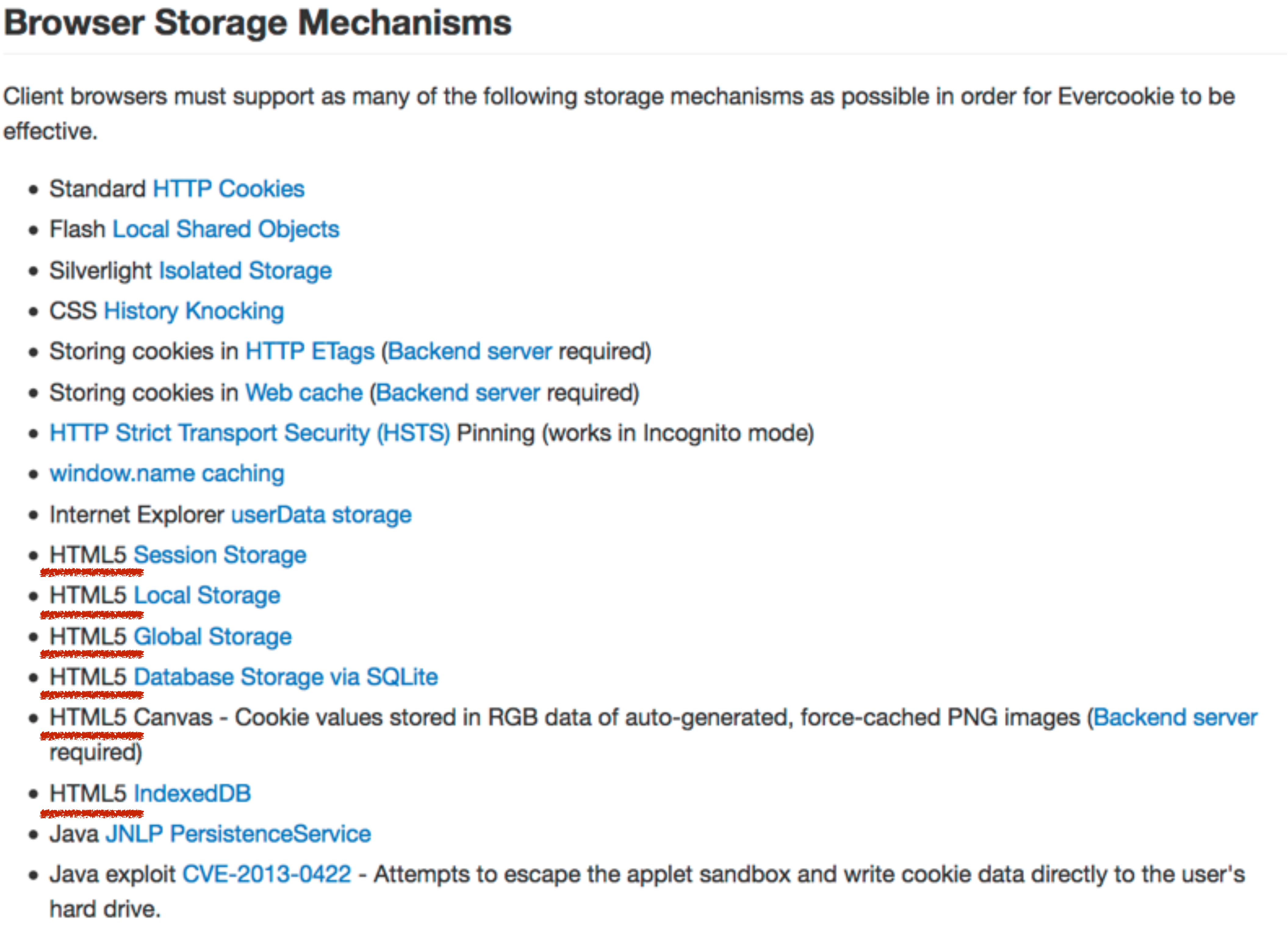
A more pessimistic view on cookies

Evercookie

> “`evercookie` is a javascript API available that produces **extremely persistent cookies** in a browser. Its goal is to **identify a client** even **after they've removed standard cookies** […]
> `evercookie` accomplishes this by storing the cookie data in **several types of storage mechanisms** that are available on the local browser.
> Additionally, if `evercookie` has found the user has removed any of the types of cookies in question, it **recreates** them using each mechanism available.”

Source: http://www.samy.pl/evercookie/

Evercookie



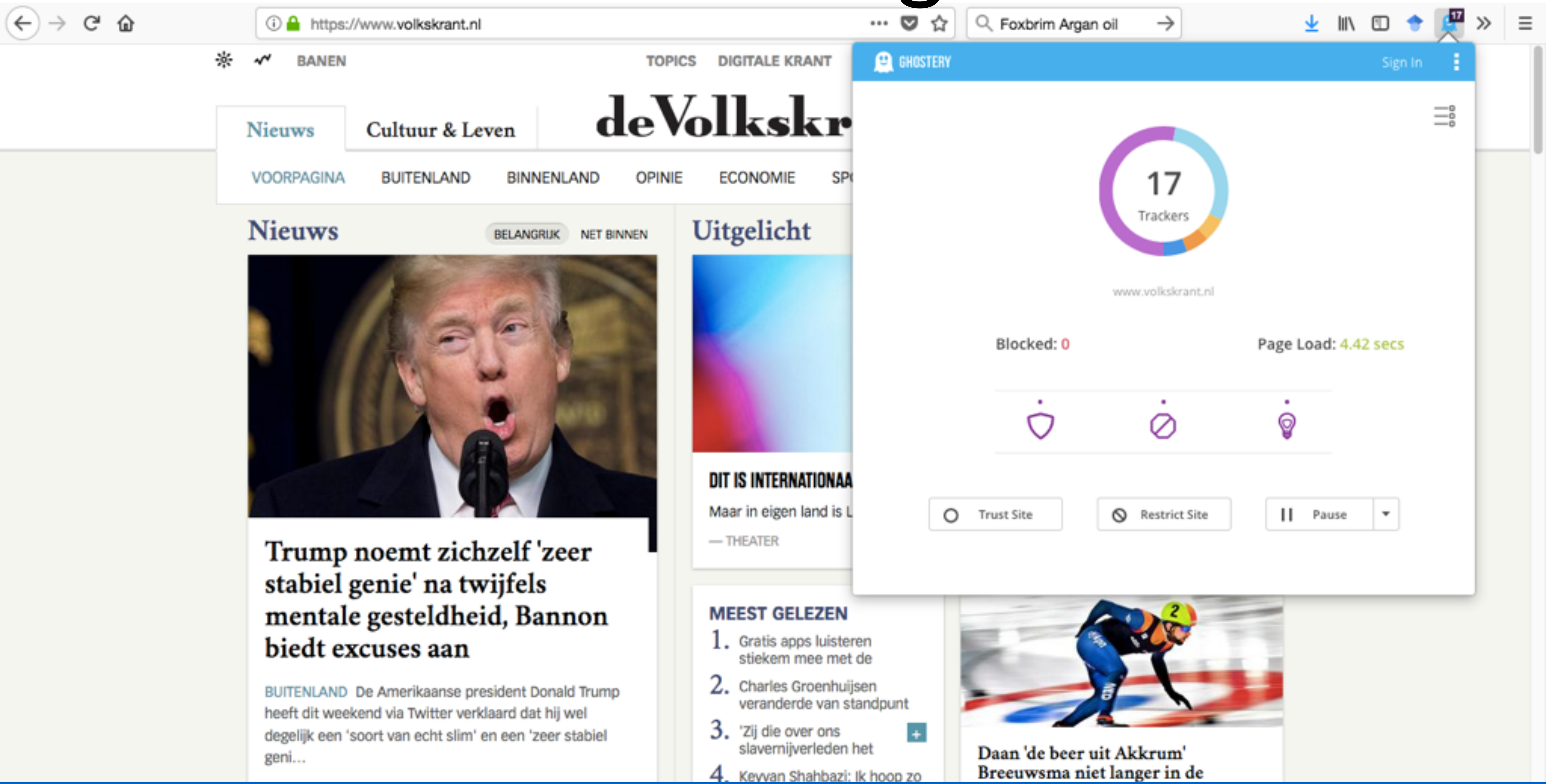
Browser Storage Mechanisms

Client browsers must support as many of the following storage mechanisms as possible in order for Evercookie to be effective.

- Standard HTTP Cookies
- Flash Local Shared Objects
- Silverlight Isolated Storage
- CSS History Knocking
- Storing cookies in HTTP ETags (Backend server required)
- Storing cookies in Web cache (Backend server required)
- HTTP Strict Transport Security (HSTS) Pinning (works in Incognito mode)
- window.name caching
- Internet Explorer userData storage
- HTML5 Session Storage
- HTML5 Local Storage
- HTML5 Global Storage
- HTML5 Database Storage via SQLite
- HTML5 Canvas - Cookie values stored in RGB data of auto-generated, force-cached PNG images (Backend server required)
- HTML5 IndexedDB
- Java JNLP PersistenceService
- Java exploit CVE-2013-0422 - Attempts to escape the applet sandbox and write cookie data directly to the user's hard drive.

Source: https://github.com/samyk/evercookie

Often though, we are tracked without our knowledge



Nieuws

Uitgelicht

Trump noemt zichzelf 'zeer stabiel genie' na twijfels mentale gesteldheid, Bannon biedt excuses aan

BUITENLAND De Amerikaanse president Donald Trump heeft dit weekend via Twitter verklaard dat hij wel degelijk een 'soort van echt slim' en een 'zeer stabiel geni...

DIT IS INTERNATIONAA

Maar in eigen land is L

— THEATER

MEEST GELEZEN

1. Gratis apps luisteren stiekem mee met de
2. Charles Groenhuijsen veranderde van standpunt
3. 'Zij die over ons slavernijverleden het
4. Keyvan Shahbazi: Ik hoop zo

Daan 'de beer uit Akkrum' Breeuwsma niet langer in de

17 Trackers

www.volkskrant.nl

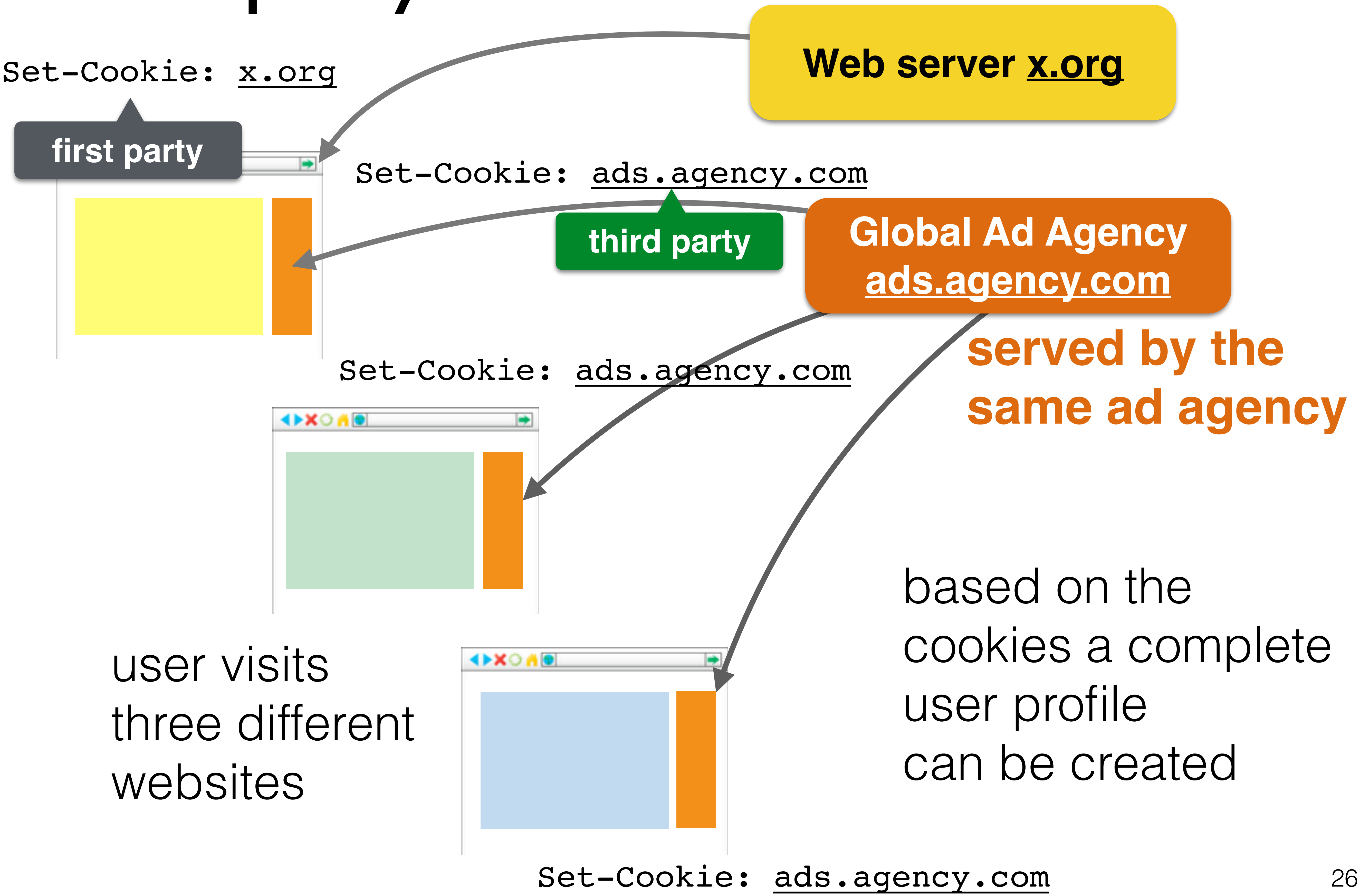
Blocked: 0

Page Load: 4.42 secs

https://www.ghostery.com/

First-party cookies are cookies that belong to the same domain that is shown in the browser's address bar.

Third-party cookies



Web server x.org

Set-Cookie: x.org

first party

Set-Cookie: ads.agency.com

third party

Global Ad Agency
ads.agency.com

served by the same ad agency

Set-Cookie: ads.agency.com

Set-Cookie: ads.agency.com

user visits three different websites

based on the cookies a complete user profile can be created

Client-side cookies

Cookies in JavaScript

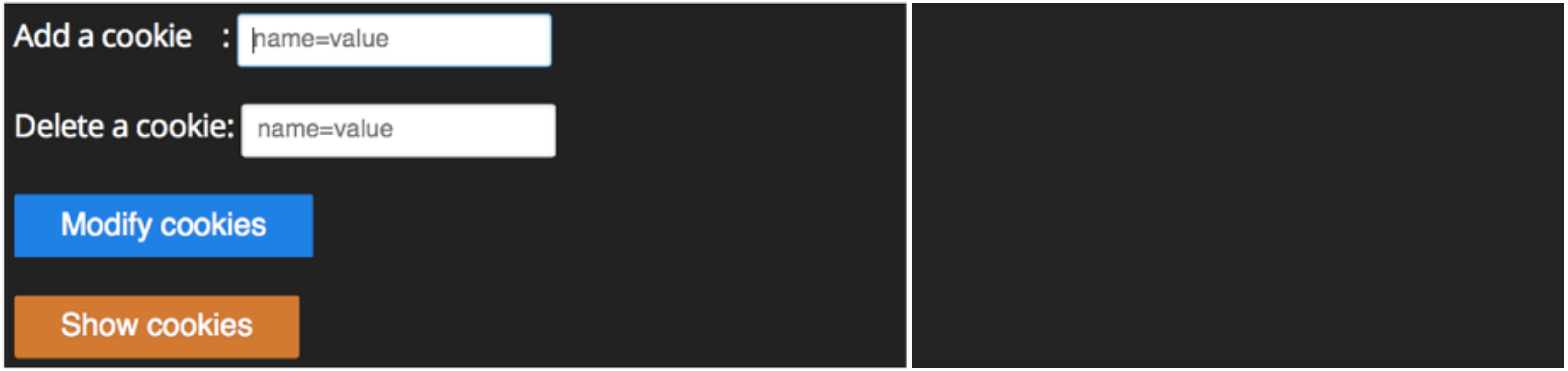
- Not always necessary to receive cookies from a server
- Cookies can be **set in the browser**
- Standard use case: remember form input

```
//set TWO(!) cookies
document.cookie = "name1=value1";
document.cookie = "name2=value2; expires=Fri,
                          24-Jan-2019 12:45:00 GMT";

//delete a cookie by RESETTING the expiration date
document.cookie = "name2=value2; expires=Fri,
                          24-Jan-1970 12:45:00 GMT";
```

`document.cookie` is unlike any other

Example 12

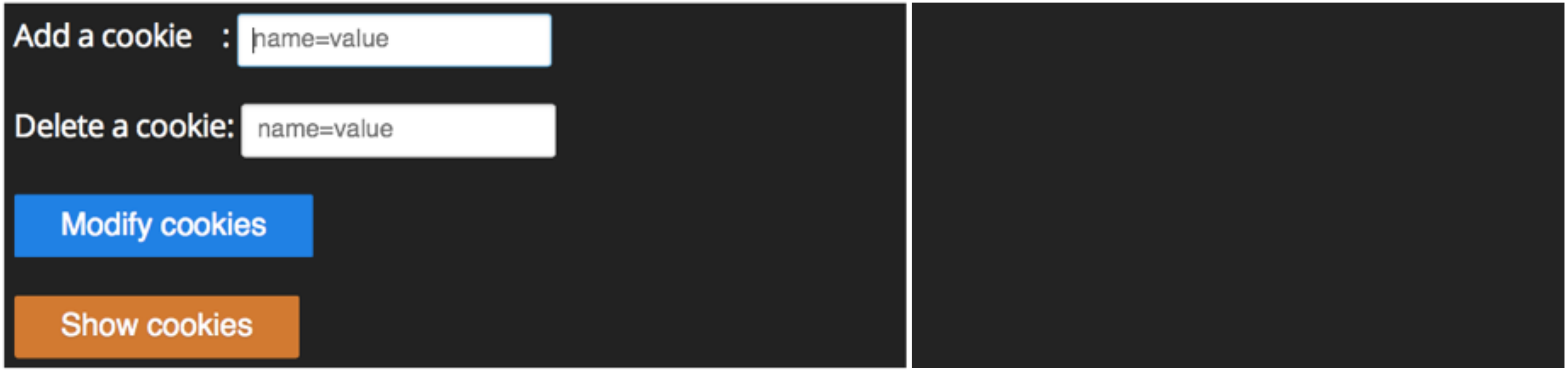
```
//adding three cookies
document.cookie = "couponnum=123";
document.cookie = "couponval=20%";
document.cookie = "expires=60";

//delete a cookie
//document.cookie=null or document.cookie="" has no effect
document.cookie = "name=value; expires=Thu,
                          01-Jan-1970 00:45:00 GMT";
```



`document.cookie` is unlike any other

Example 12

```
var toadd = document.getElementById('addCookie').value;

 if( toadd.length > 0) {
     document.cookie = toadd;
 }
 var todel = document.getElementById('deleteCookie').value;

 if( todel.length > 0) {
     document.cookie =
          todel+'; expires=Thu, 01-Jan-1970 00:00:01 GMT';
 }
```



Reading cookies in JavaScript

- `document.cookie["firstname"]` does **not work**
- **String** returned by `document.cookie` needs to be parsed `couponnum=123; couponval=20%; expires=60`

```
var cookiesArray = document.cookie.split("; “);
var cookies=[];

for(var i=0; i < cookiesArray.length; i++) {
  var cookie = cookiesArray[i].split("=");
  cookies[cookie[0]]=cookie[1];
}
```

- **Alternative**: `js-cookie` (140 lines of code)

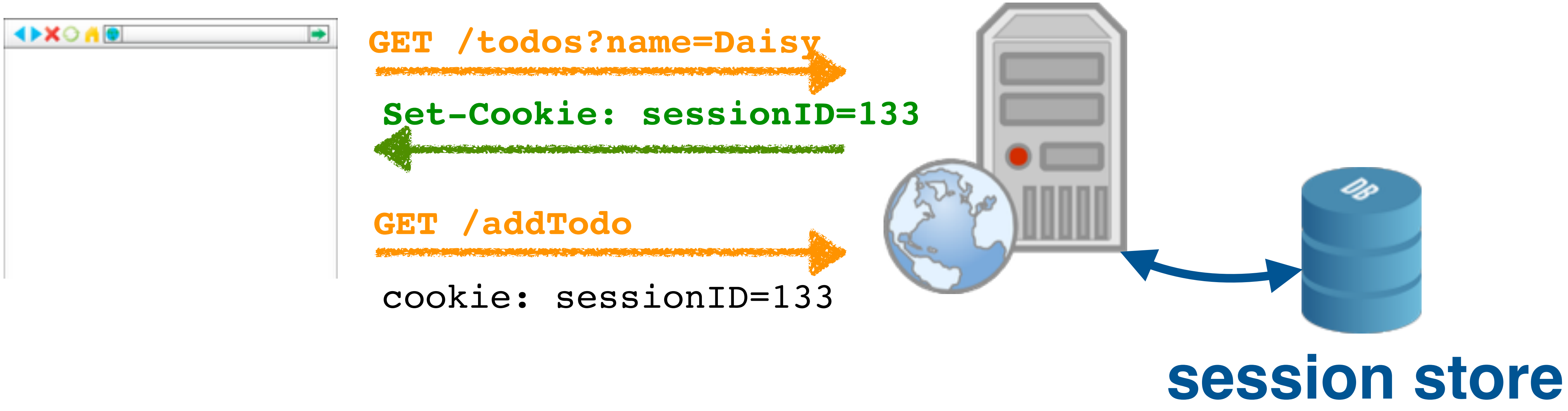
https://github.com/js-cookie/js-cookie

Sessions

Establishing a session

- **Common scenario**: short period of time that users interact with a web site (a session)
- **Goals**:
  - Track the user without relying (too much) on unreliable cookies
  - Allow larger amounts of data to be stored
- **Problem**: without cookies the server cannot tell clients apart
- **Solution**: **hybrid approach** between cookies and server-side saved data

Sessions in one slide



- Cookies are used to store a **single ID** on the **client**
- **Remaining user information** is stored **server-side** in memory or in a database

Establishing a session

1. **Client** requests a first page from the server
2. Server creates **unique session ID** and initiates the storage of the session data for that client
3. Server sends back a page with a **cookie** containing the session ID
4. From now on, the client sends **page requests together with the cookie**
5. Server can use the **ID to personalise** the response
6. A **session ends** when no further requests with that session ID come in (timeout)

Sessions in express with memory stores

- Easy to set up in express
- Same drawback as any in-memory storage: not **persistent** across machine failure
- A middleware component is helping out: **`express-session`**: https://github.com/expressjs/session
- Most common use case: **authentication**

Authentication: verifying a user's identity

Example 10

Sessions in express with memory stores

```
npm install cookie-parser
npm install express-session
```

```
var express = require("express");
var http = require("http");
var credentials = require("./credentials");
var cookies = require("cookie-parser");
var sessions = require("express-session");

var app = express();
app.use(cookies(credentials.cookieSecret));
app.use(sessions(credentials.cookieSecret));
http.createServer(app).listen(3001);

app.get("/countMe", function (req, res) {
    var session = req.session;
    if (session.views) {
        session.views++;
        res.send("You have been here " +
               session.views + " times (last visit: " + session.lastVisit + ")");
        session.lastVisit = new Date().toLocaleDateString();
    }
    else {
        session.views = 1;
        session.lastVisit = new Date().toLocaleDateString();
        res.send("This is your first visit!");
    }
});
```

Example 10

Sessions in express with memory stores

```
npm install cookie-parser
npm install express-session
```

```
var express = require("express");
var http = require("http");
var credentials = require("./credentials");
var cookies = require("cookie-parser");
var sessions = require("express-session");

var app = express();
app.use(cookies(credentials.cookieSecret));
app.use(sessions(credentials.cookieSecret));
http.createServer(app).listen(3001);

app.get("/countMe", function (req, res) {
   var session = req.session;
   if (session.views) {
      session.views++;
      res.send("You have been here " +
            session.views + " times (last visit: " + session.lastVisit + ")");
      session.lastVisit = new Date().toLocaleDateString();
   }
   else {
      session.views = 1;
      session.lastVisit = new Date().toLocaleDateString();
      res.send("This is your first visit!");
   }
});
```

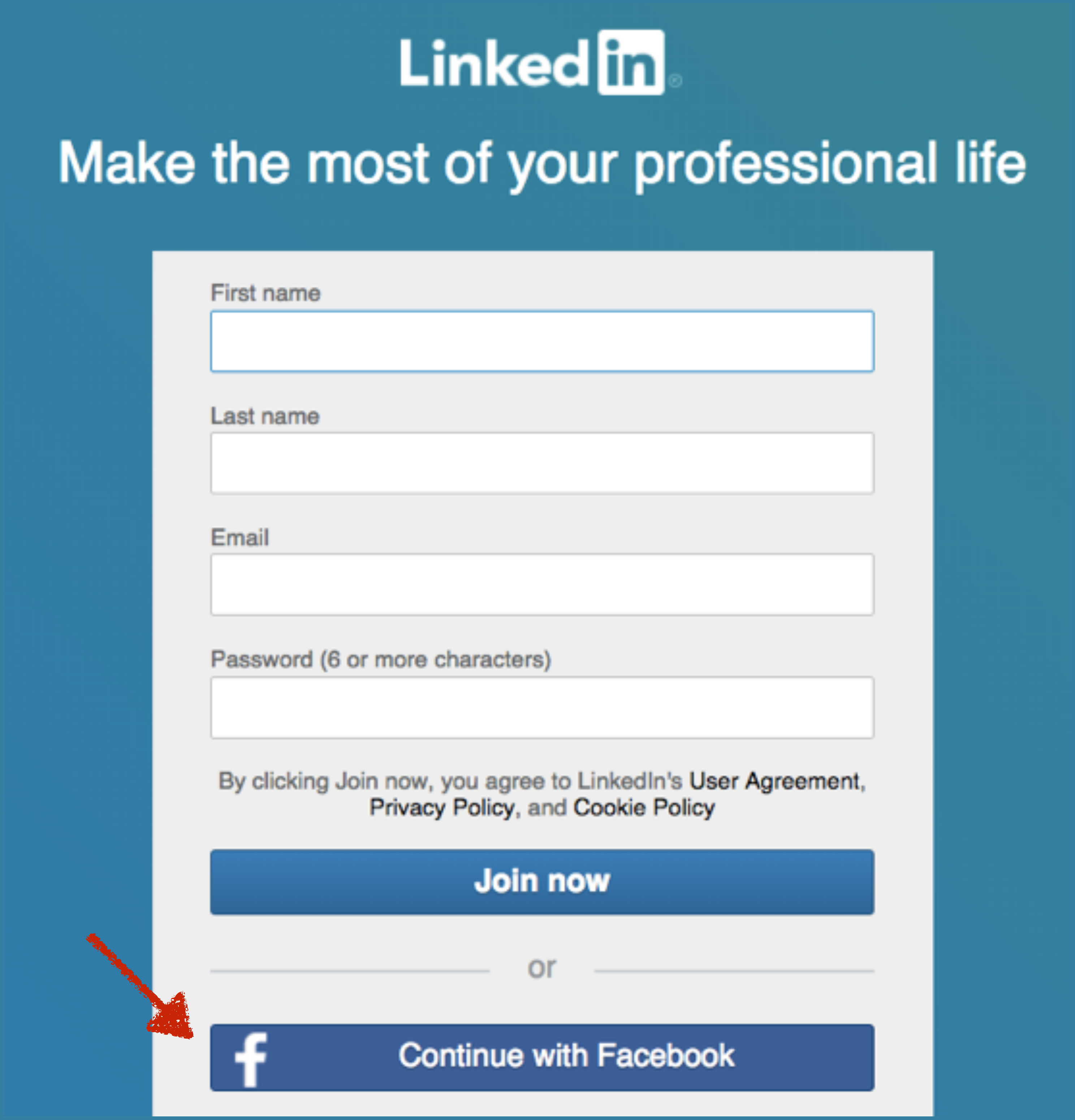
cookie & session setup

session object available on `req` object only

session exists!

session does not yet exist

Third-party authentication



Linked in

Make the most of your professional life

First name

Last name

Email

Password (6 or more characters)

By clicking Join now, you agree to LinkedIn's User Agreement, Privacy Policy, and Cookie Policy

- Twitter
- Google
- Facebook
- Yahoo
- LinkedIn
- ...

Overview

- **Weakest link** in an authenticated application is the **user's password**
- **Application-based decision**
  - Does the application need authentication?
  - Are cookies/sessions enough?
  - If authentication is needed, should third-party authentication be used? (low cognitive burden for the user)

Third-party authentication

- Authenticating users through popular social Web services (Twitter, Facebook, Google, LinkedIn, etc.)
- **Easy** to develop for popular platforms
- **Trusted** social Web platforms **provide authentication**, no need to store passwords or employ particular security measures
- **However**: some users may not use social Web platforms or do not like to hand over their data

OAuth 2.0 Authorization Framework

> "The OAuth 2.0 authorization framework enables a **third-party application** to obtain **limited access** to an HTTP service, either **on behalf** of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf."

Source: https://tools.ietf.org/html/rfc6749

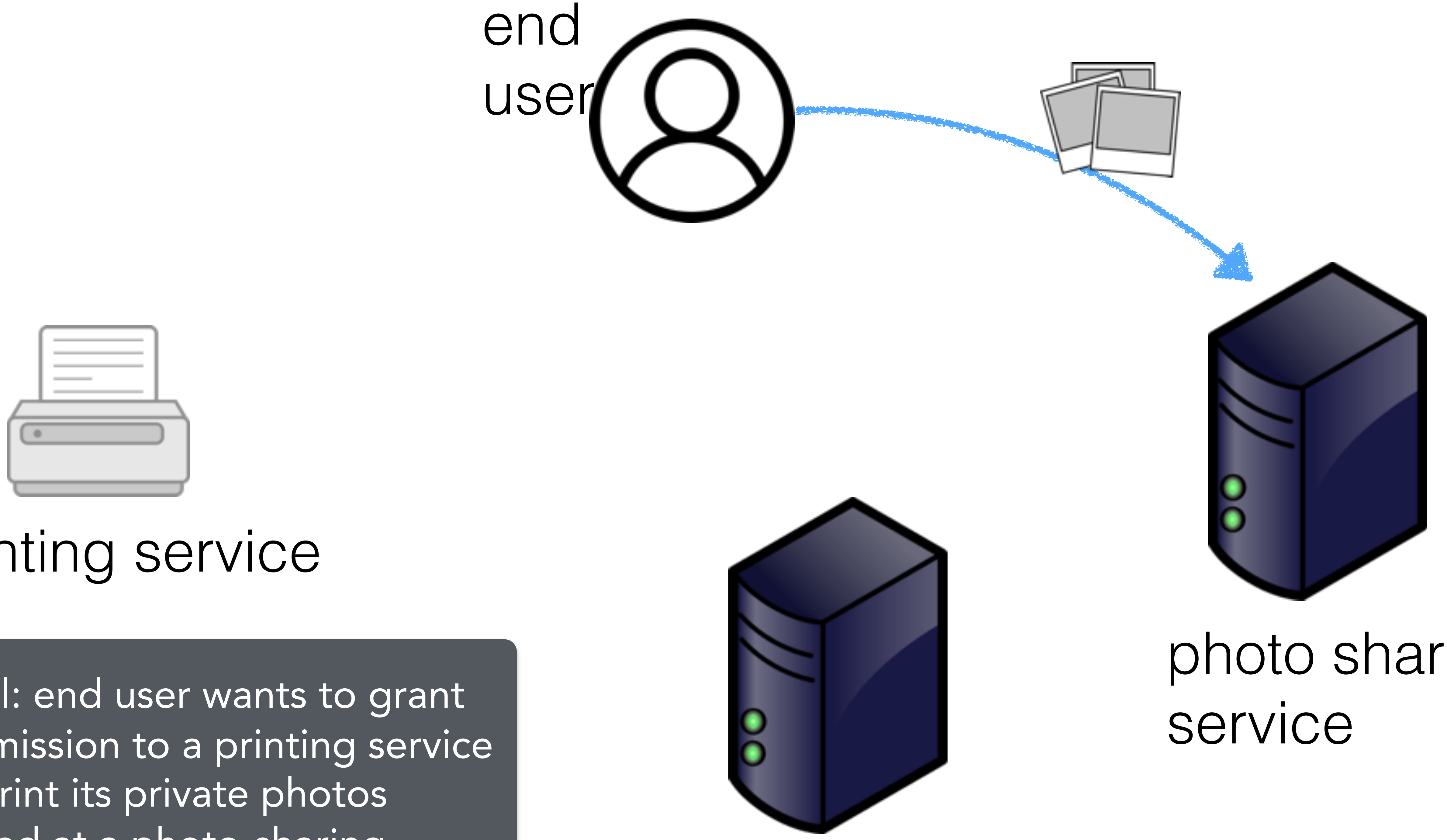
OAuth 2.0 roles

- **Resource owner**: entity that grants access to a protected resource
- **Resource server**: server hosting the protected resources, capable of accepting and responding to protected resource requests using **access tokens**.

a string denoting a specific scope, lifetime and other access attributes

- **Client**: an application making protected resource requests on behalf of the resource owner **and with its authorisation**
- **Authorization server**: server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization

OAuth 2.0 roles exemplified



end user

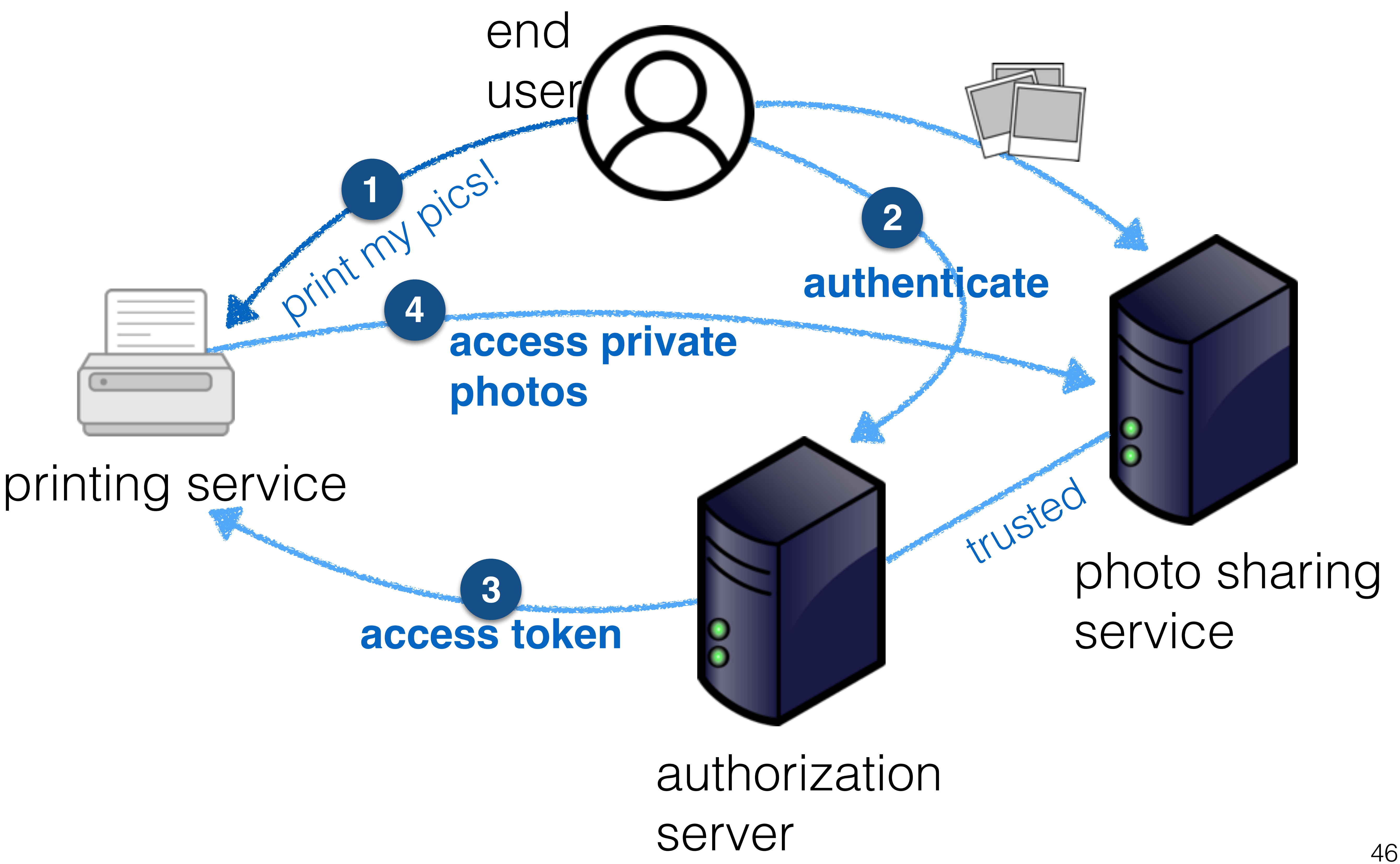
printing service

photo sharing service

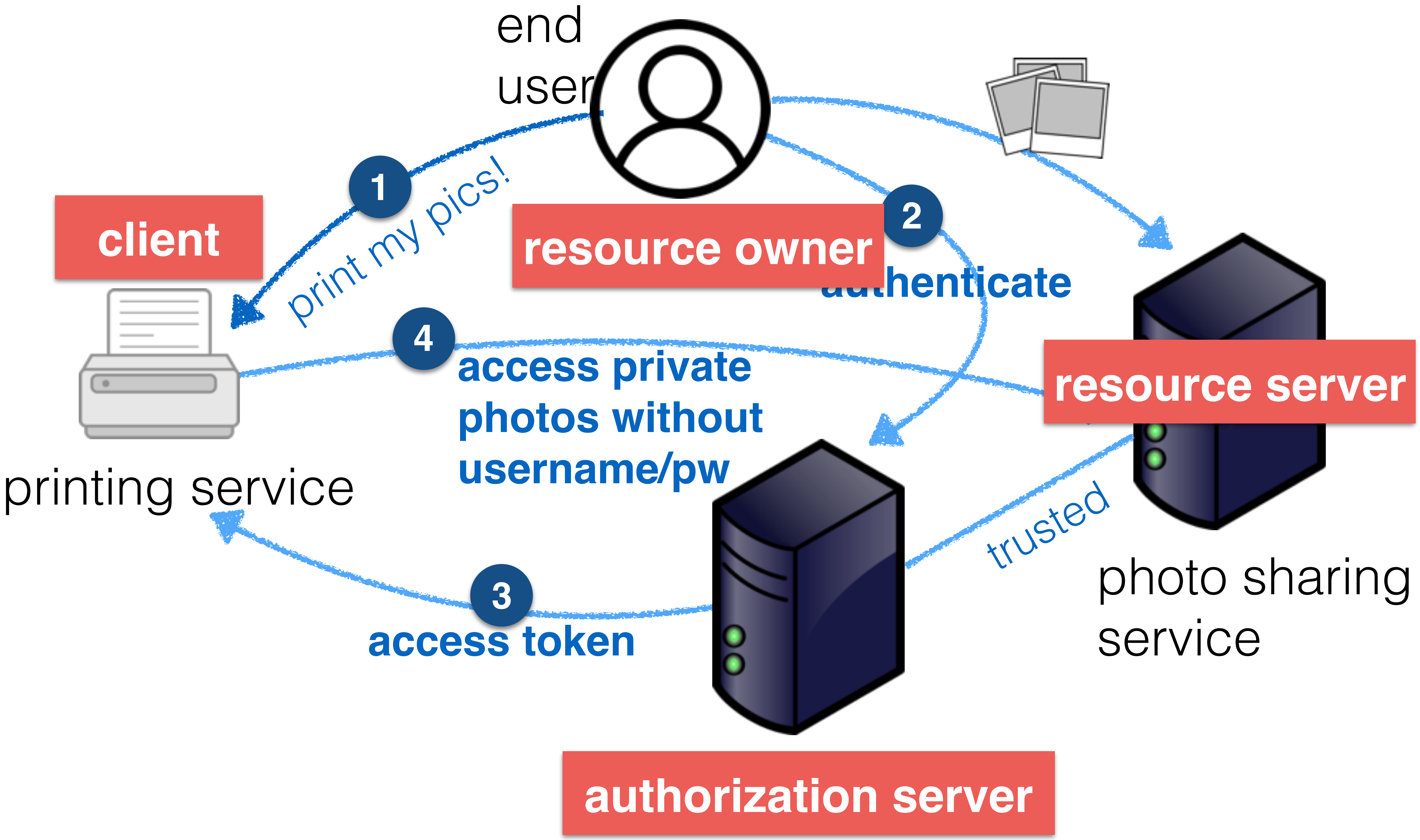
authorization server

Goal: end user wants to grant permission to a printing service to print its private photos stored at a photo-sharing service **without giving away her username/password**.

OAuth 2.0 roles exemplified



OAuth 2.0 roles exemplified



end user

resource owner

client

printing service

1 print my pics!

2 authenticate

3 access token

4 access private photos without username/pw

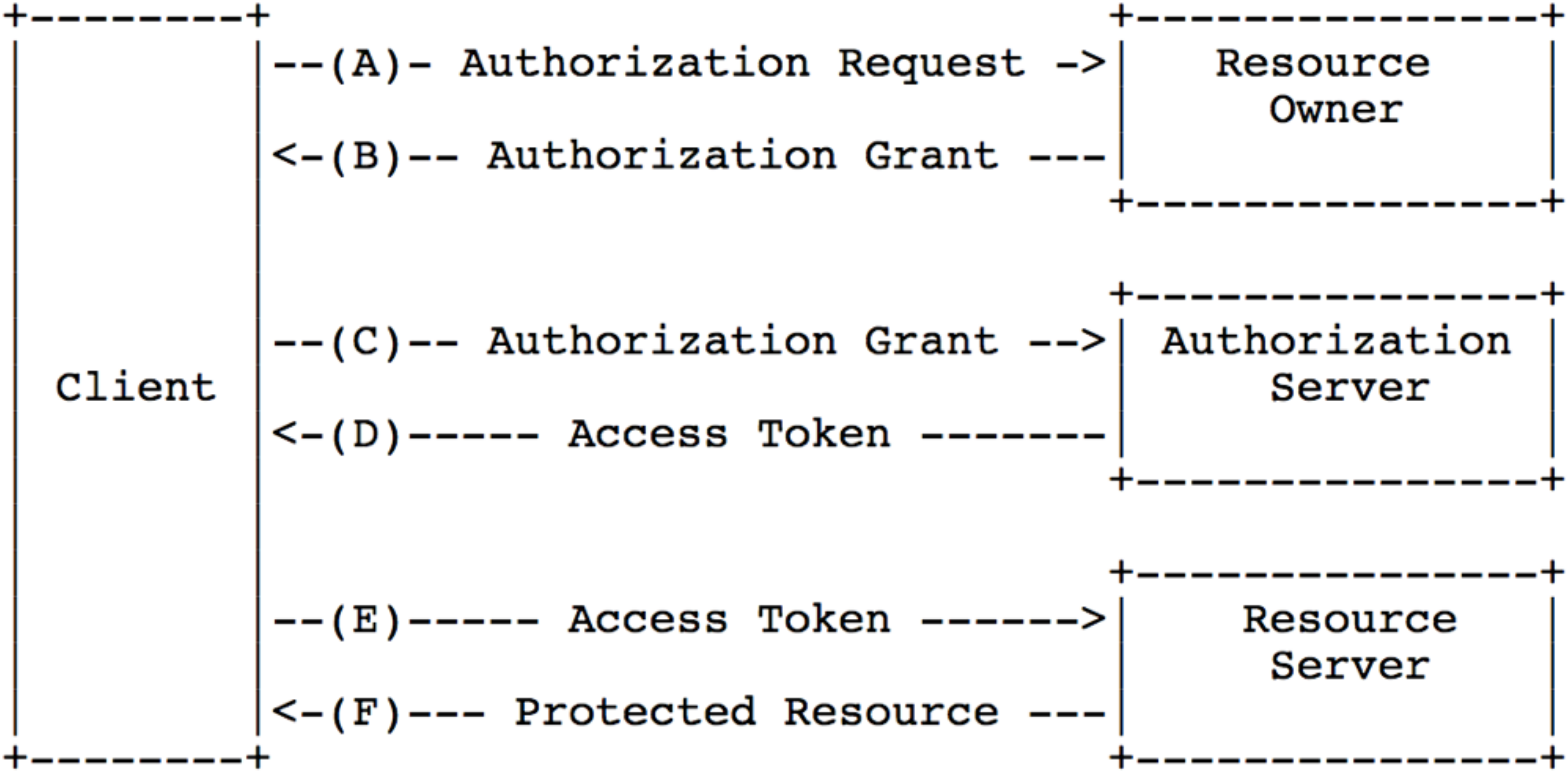
resource server

photo sharing service

trusted

authorization server

Abstract protocol flow



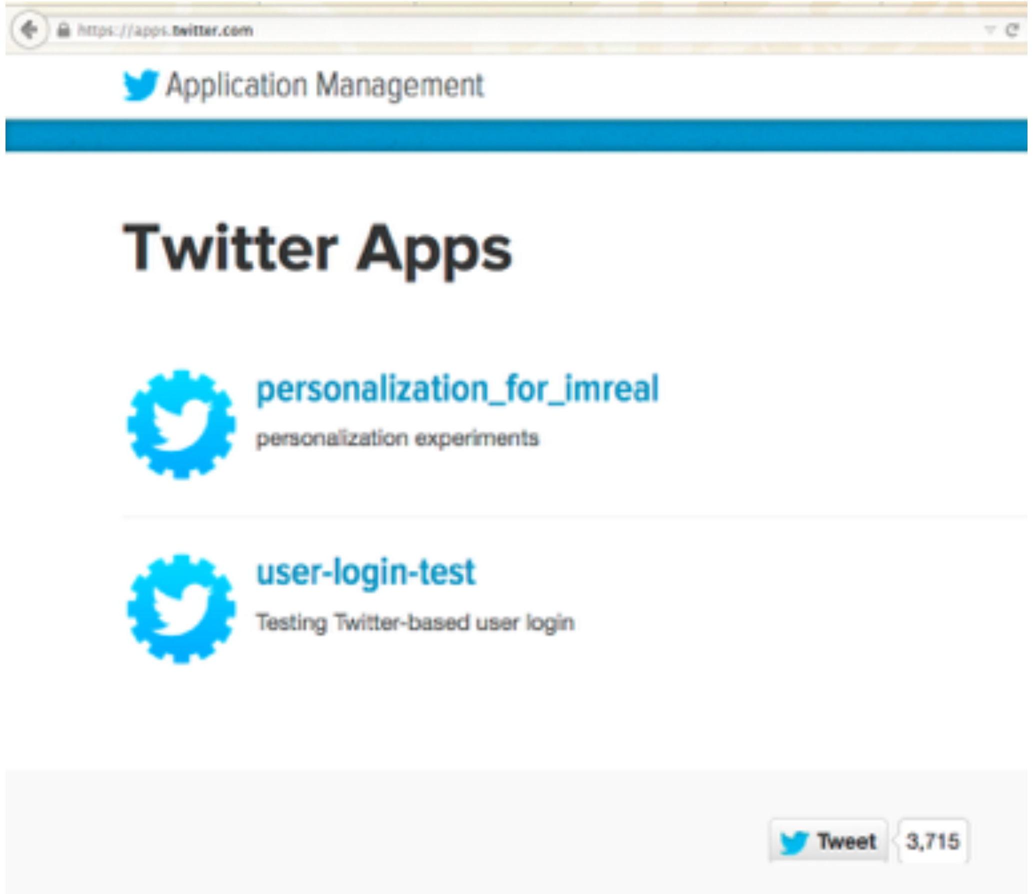
```
+--------+                               +---------------+
|        |--(A)- Authorization Request ->|   Resource    |
|        |                               |     Owner     |
|        |<-(B)-- Authorization Grant ---|               |
|        |                               +---------------+
|        |
|        |                               +---------------+
|        |--(C)-- Authorization Grant -->| Authorization |
| Client |                               |     Server    |
|        |<-(D)----- Access Token -------|               |
|        |                               +---------------+
|        |
|        |                               +---------------+
|        |--(E)----- Access Token ------>|    Resource   |
|        |                               |     Server    |
|        |<-(F)--- Protected Resource ---|               |
+--------+                               +---------------+
```

Source: https://tools.ietf.org/html/rfc6749

Third-party authentication
Twitter example

Goal: “Sign in with your Twitter account”

- Works similarly (but not in exactly the same way) across different services
- Starting point: **create** an “app” (Twitter app, Facebook app, etc.)



https://apps.twitter.com

Application Management

Twitter Apps

personalization_for_imreal

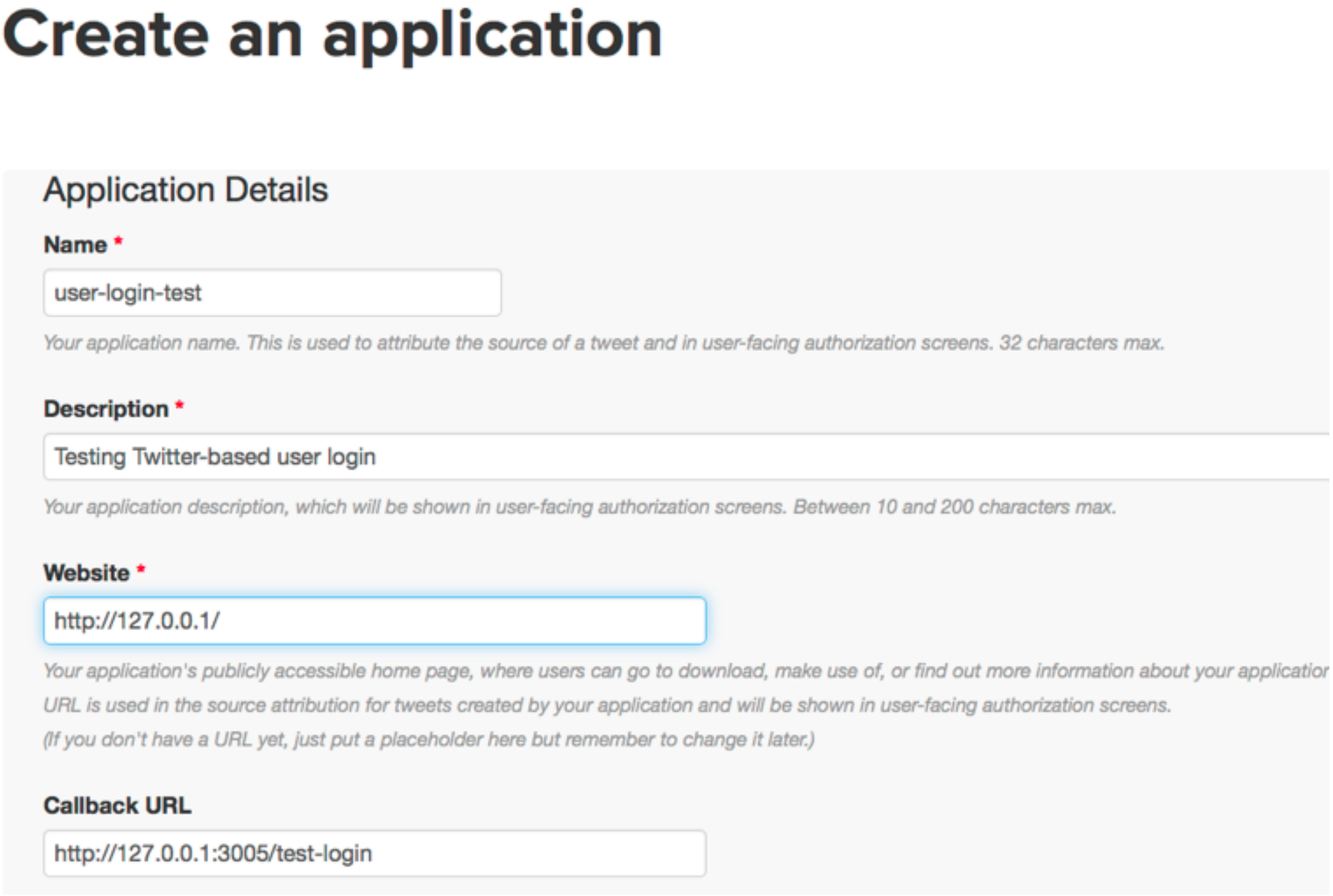
personalization experiments

user-login-test

Testing Twitter-based user login

https://apps.twitter.com/

Third-party authentication Twitter example



Create an application

Application Details

Name *

user-login-test

Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.

Description *

Testing Twitter-based user login

Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.

Website *

http://127.0.0.1/

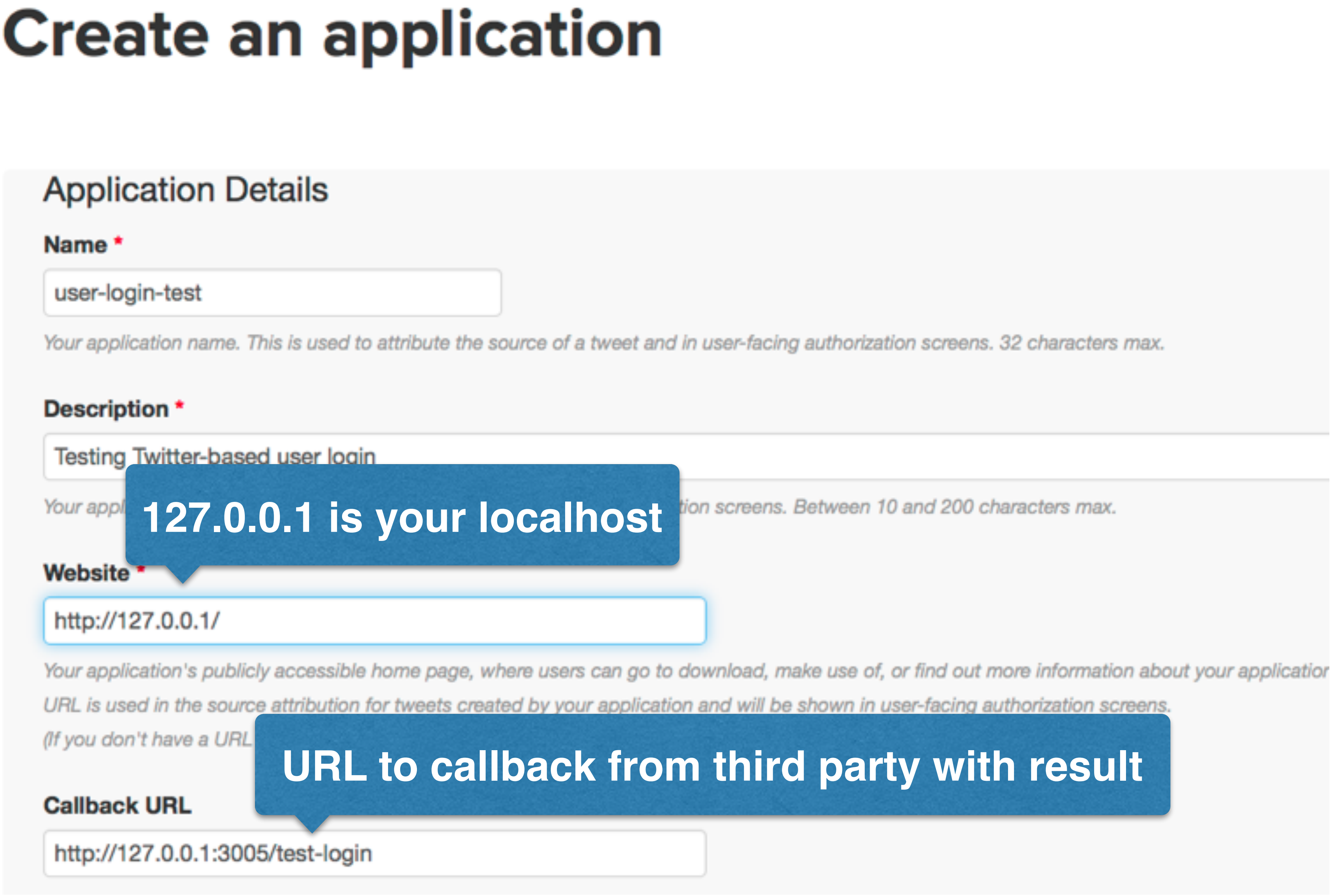
Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your applicatior
URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens.
(If you don't have a URL yet, just put a placeholder here but remember to change it later.)

Callback URL

http://127.0.0.1:3005/test-login

https://apps.twitter.com/

Third-party authentication
Twitter example



Create an application

Application Details

Name *

user-login-test

Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.

Description *

Testing Twitter-based user login

Your appl ion screens. Between 10 and 200 characters max.

127.0.0.1 is your localhost

Website *

http://127.0.0.1/

Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your applicatior URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens.

(If you don't have a URL

URL to callback from third party with result

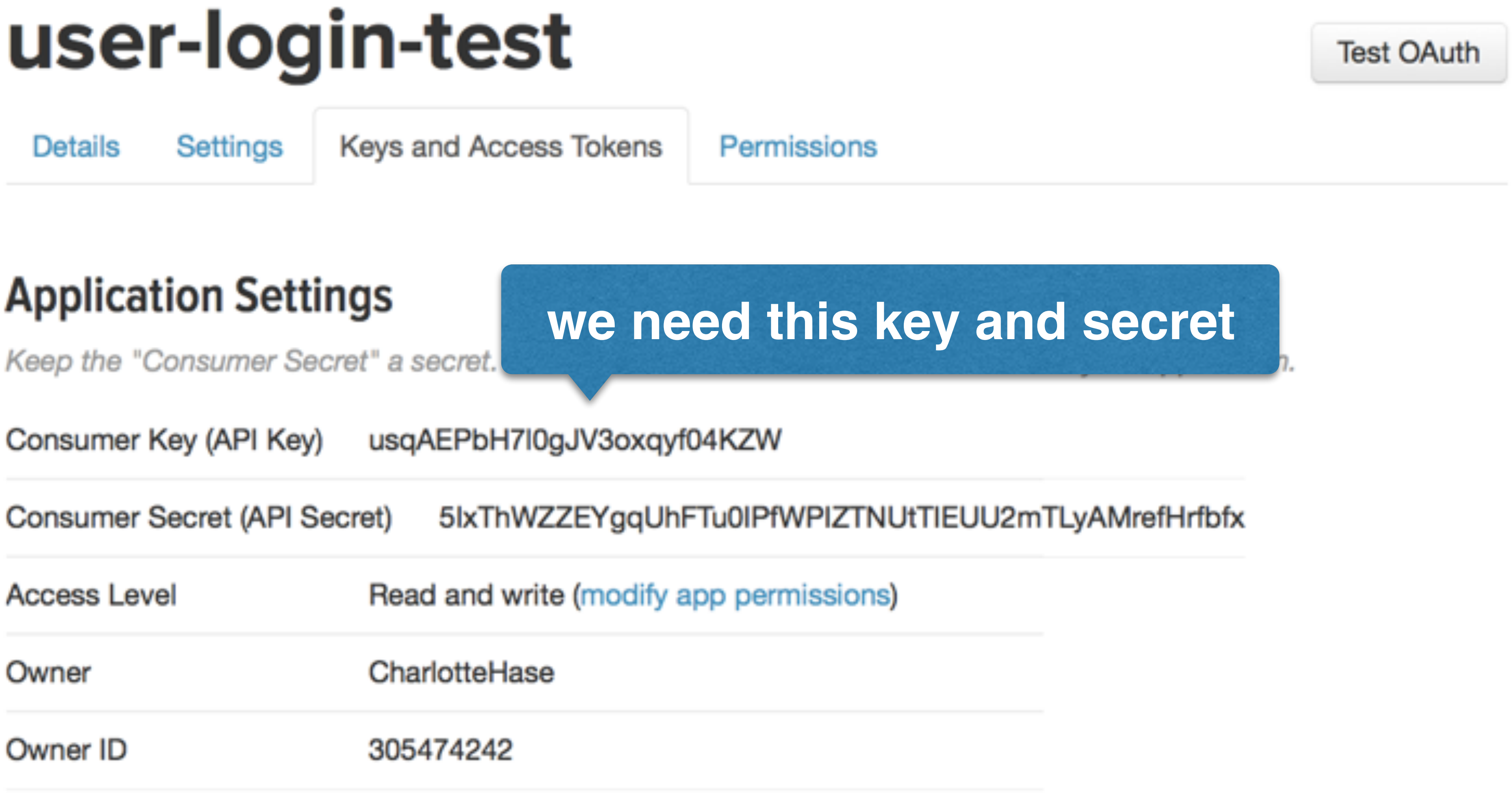
Callback URL

http://127.0.0.1:3005/test-login

https://apps.twitter.com/

Third-party authentication Twitter example cont.

- In application settings, check “Allow this application to be used to Sign in with Twitter”
- Read out the **access tokens**



user-login-test

Application Settings

we need this key and secret

Keep the "Consumer Secret" a secret.

Consumer Key (API Key)	usqAEPbH7l0gJV3oxqyf04KZW
Consumer Secret (API Secret)	5lxThWZZEYgqUhFTu0IPfWPIZTNUtTIEUU2mTLyAMrefHrfbfx
Access Level	Read and write (modify app permissions)
Owner	CharlotteHase
Owner ID	305474242

Third-party authentication Twitter example cont.

- Express can make use of `passport`, one of the most popular **authentication middleware components**
  - 300+ authentication **strategies**
  - Supports OpenID and OAuth

- `Passport` hides a lot of complexity from you

```
$ npm install passport
```

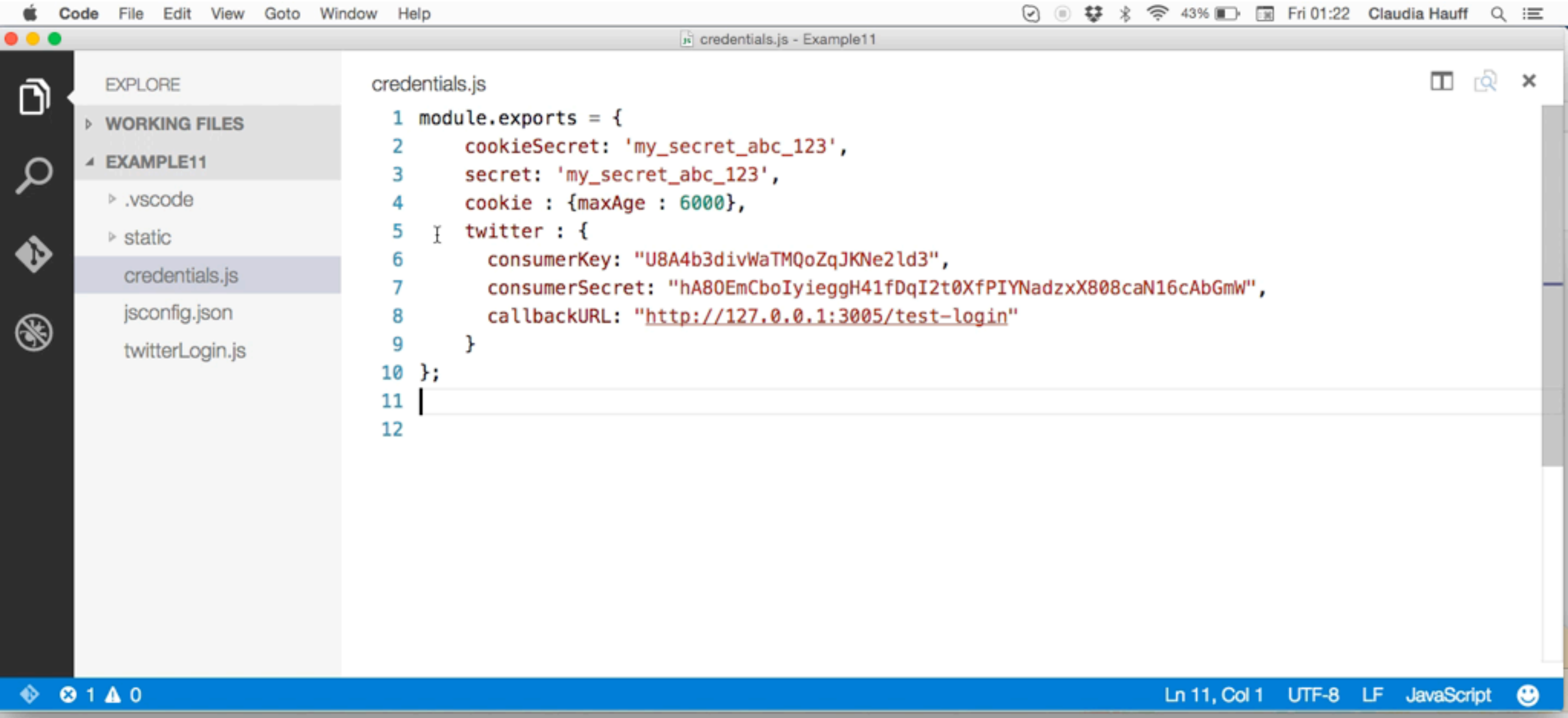
```
$ npm install passport-twitter
```

Installing a strategy

http://passportjs.org/

Example 11

Example: authenticating through Twitter

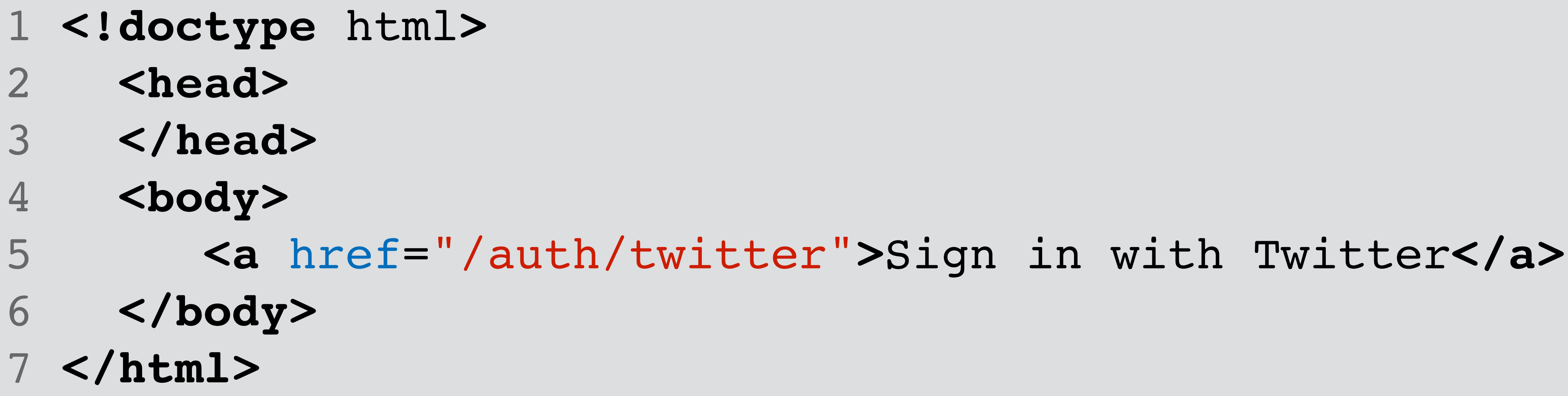


```
module.exports = {
    cookieSecret: 'my_secret_abc_123',
    secret: 'my_secret_abc_123',
    cookie : {maxAge : 6000},
    twitter : {
      consumerKey: "U8A4b3divWaTMQoZqJKNe2ld3",
      consumerSecret: "hA8OEmCboIyieggH41fDqI2t0XfPIYNadzxX808caN16cAbGmW",
      callbackURL: "http://127.0.0.1:3005/test-login"
    }
};

```

Third-party authentication Twitter example cont.

```javascript
// Redirect the user to Twitter for authentication.
app.get('/auth/twitter', passport.authenticate('twitter'));

// Twitter will redirect the user to this URL after approval.
app.get('/test-login',
  passport.authenticate('twitter', { failureRedirect: '/failure' }),
  function(req, res) {
    res.redirect('/success');
  });

app.get("/success", function (req, res) {
  console.log("Success!");
  res.send("User login via Twitter successful!");
});

app.get("/failure", function (req, res) {
  console.log("Failure!");
  res.send("User login via Twitter was unsuccessful!");
});
```

Third-party authentication
Twitter example cont.



```html
<!doctype html>
  <head>
  </head>
  <body>
     <a href="/auth/twitter">Sign in with Twitter</a>
  </body>
</html>
```

Summary

- Cookies
- Sessions
- Third-party authentication

End of Lecture