Cookies, sessions and authentication

TI1506: Web and Database Technology
Claudia Hauff

Lecture 7 [Web], 2014/15

Course overview [Web]

1. http: the language of Web communication
2. Web (app) design & HTML5
3. JavaScript: interactions in the browser
4. node.js: JavaScript on the server
5. CSS: Lets make things pretty
6. Ajax: asynchronous JavaScript
7. **Personalisation: Cookies & sessions & authentication**
8. Securing your application

Learning objectives

- **Decide** for a given usage scenario whether cookies and/or sessions are suitable
- **Implement** code to create/change/delete cookies
- **Implement** code to create/change/delete sessions
- **Implement** third-party authentication

Introduction to cookies and sessions

Recall: the HTTP protocol

- 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.com (tracking users’ visits) can exclude a particular visitor from being tracked
- JavaScript games keep track of the game’s status when you re-visit the website
- Websites can tell you how many times you have visited them
- Video streaming (**if you have time, watch this video about the virtues of JavaScript!!**) https://www.youtube.com/watch?v=bo36MrBfTk4

Cookies

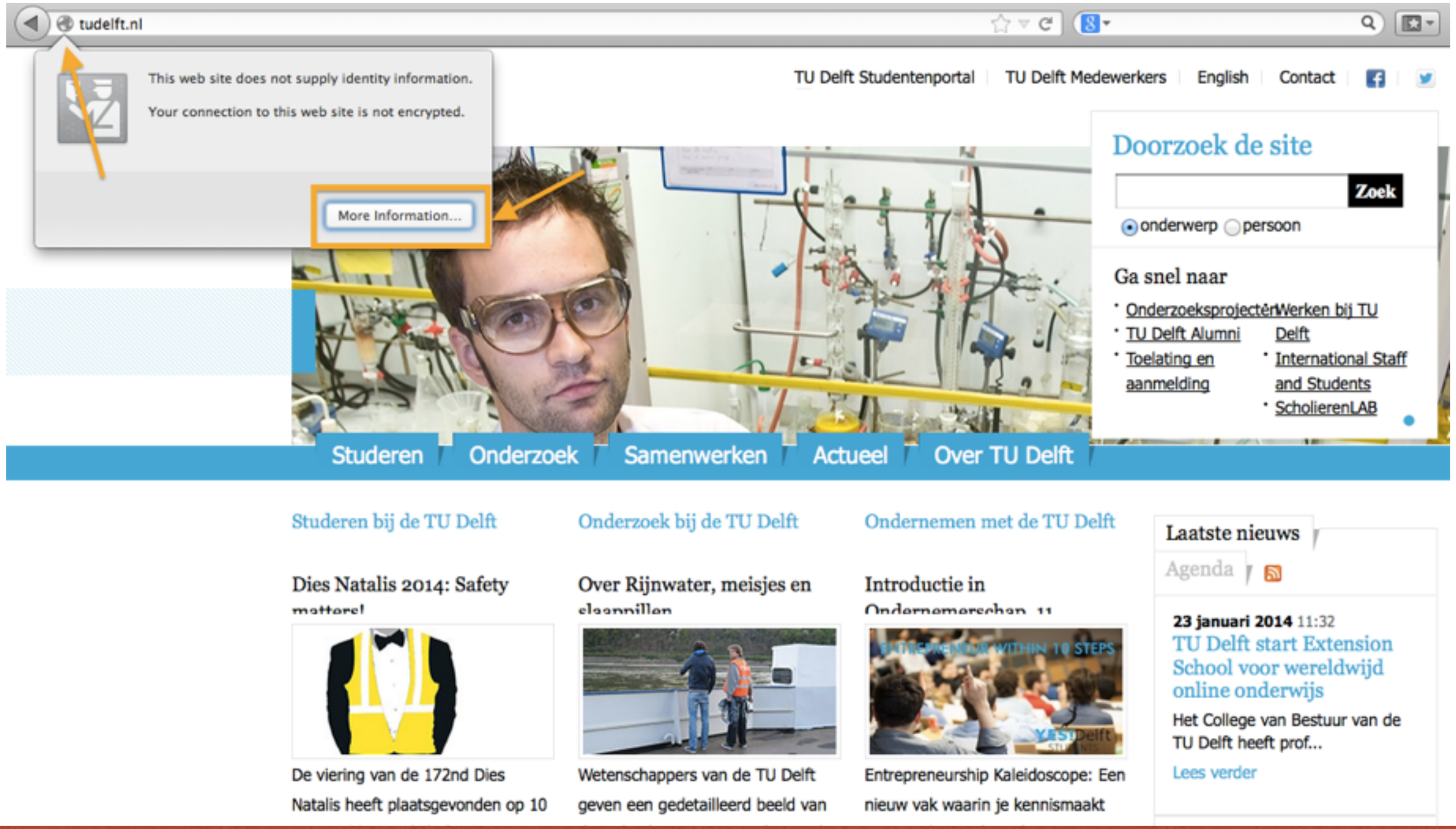
Cookies and sessions are ways to **introduce state** on top of the stateless HTTP protocol.

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

Minimum size requirements (RFC2625 from 2011)

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

Where can I find the cookies?



Lets take a look at the cookies served at bol.com and tudelft.nl.
We can change the cookies: Cookies Manager+

Cookie & session basics

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

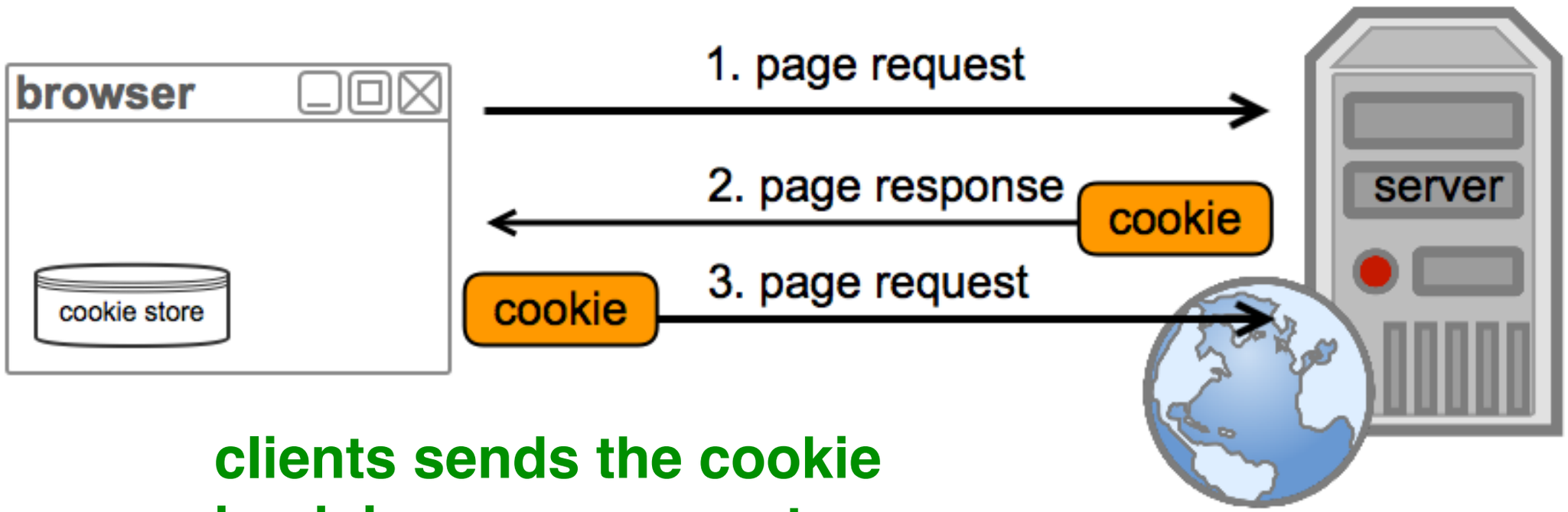
- Cookies are **visible** to the users (who make the effort)
  - By default, stored in the clear
- Clients (users, i.e. you!) can **delete/disallow** cookies
- Cookies can be **altered by the client**
  - Opens up a line of attack: **servers** should not send sensitive information in simple cookies
- **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

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

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**:
  - User’s login name
  - History of page views
  - Settings of form elements (can be fully client-side)

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. "

Source: http://tools.ietf.org/html/rfc6265

Session vs. persistent cookies

- **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

Difference in action: clicking the ***This is a public computer*** button before logging in.

Cookie expiration dates

- **Gmail**: 15 cookies (4 session),
  expiration dates in 2014, 2015 and 2016
- **Facebook**: 13 cookies (5),
  expiration dates in 2014, 2015 and 2016
- **Volkskrant**: 6 cookies,
  expiration dates in 2014, 2016 and 2017
- **Amazon**: 21 cookies (1),
  expiration dates in 2014-17, 2028, 2034/36/38 and 2082

Cookie fields

- **Name=value** (only required field, the rest has defaults)
- **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.
- **Secure** flag
- **httpOnly** flag
- **Signed** flag

Making cookies more robust

- **Secure** cookies:
  - 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
- **Signed** cookies:
  - Ensures that the value has not been tampered with by the client

Secure setting via HTTP: cookie will not be set

Cookies in express

```
1  var express = require("express");
2  var http = require("http");
3  var credentials = require('./credentials.js');
4  var cookies = require("cookie-parser");
5
6  var app = express();
7  app.use(cookies(credentials.cookieSecret));
8  http.createServer(app).listen(port);
9
10 app.get("/sendMeCookies", function (req, res) {
11     console.log("Cookies sent");
12     res.cookie("chocolate", "monster");
13     res.cookie("signed_choco", "monster", { signed: true});
14     res.send();
15 });
16
17 app.get("/listAllCookies", function (req, res) {
18     console.log("\n++++ /listallcookies (unsigned) ++++");
19     console.log(req.cookies);
20     console.log("\n++++ /listallcookies (signed) ++++");
21     console.log(req.signedCookies);
22     res.send();
23 });
```

```
module.exports = {
    cookieSecret: 'my_secret',
};
```

cookie-parser middleware

creating cookies

all that is needed to sign a cookie

reading cookies

simple-cookies.js

Cookies in express

- Cookies Manager+ for Firefox: allows you to easily view/ edit cookies
- What is the difference between signed and unsigned cookies?
  - Check cookie creation
  - Change unsigned cookie
  - Change signed cookie (first name, then value)

simple-cookies.js

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

Often though, we are tracked without our knowledge

Ghostery: https://www.ghostery.com/

Third-party cookies

Set-Cookie: x.org

first party

Web server x.org

Set-Cookie: ads.agency.com

third party

Global Ad Agency
ads.agency.com

served by the same ad agency

Set-Cookie: ads.agency.com

user visits three diff. websites

based on the cookies a complete user profile can be created

Set-Cookie: ads.agency.com

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

Specifically, when creating a new cookie, it uses the following storage mechanisms when available:

- **Standard HTTP Cookies**
- **Local Shared Objects (Flash Cookies)**
- **Silverlight Isolated Storage**
- **Storing cookies in RGB values of auto-generated, force-cached PNGs using HTML5 Canvas tag to read pixels (cookies) back out**
- **Storing cookies in Web History**
- **Storing cookies in HTTP ETags**
- **Storing cookies in Web cache**
- **window.name caching**
- **Internet Explorer userData storage**
- **HTML5 Session Storage**
- **HTML5 Local Storage**
- **HTML5 Global Storage**
- **HTML5 Database Storage via SQLite**
- **HTML5 IndexedDB**
- **Java JNLP PersistenceService**
- **Java CVE-2013-0422 exploit (applet sandbox escaping)**

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

Client-side cookies

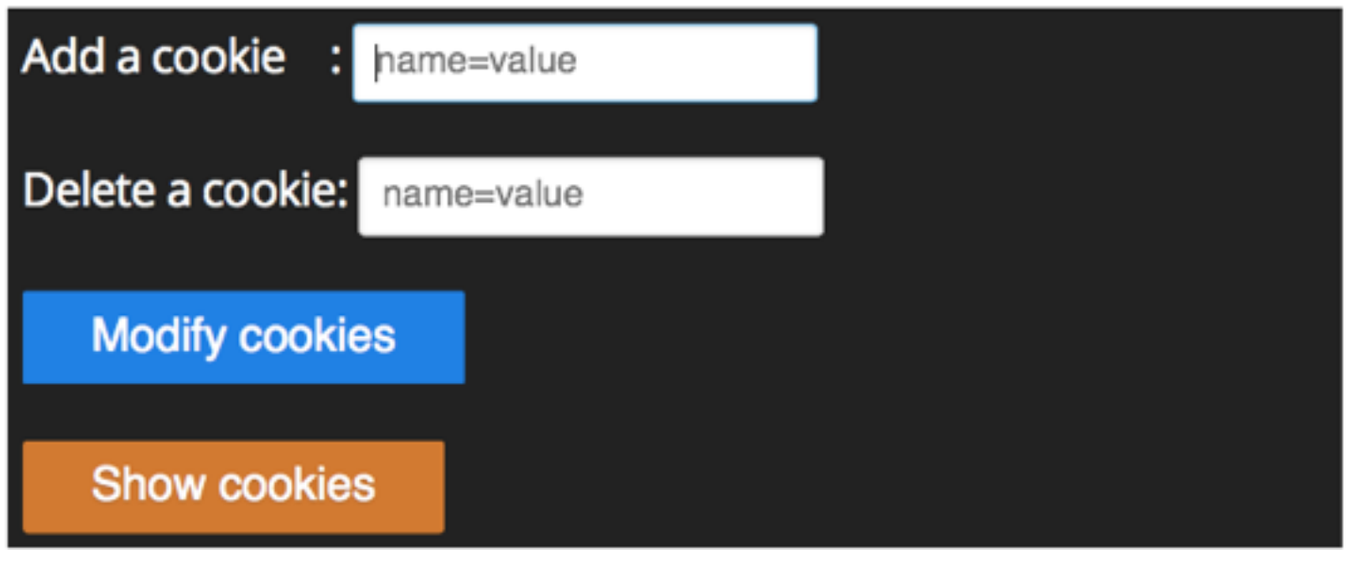
Cookies in JavaScript

- Not always necessary to receive cookies from a server
- Cookies can be set in the browser
- Usually to remember form input

```
1 //set a cookie
2 document.cookie = "name=value";
3 document.cookie = "name=value; expires=Fri,
                              24-Jan-2014 12:45:00 GMT";
4
5 //delete a cookie
6 document.cookie = "name=value; expires=Fri,
                              24-Jan-1970 12:45:00 GMT";
```

`document.cookie` is unlike any other

```
1 //adding three cookies
2 document.cookie = "couponnum=123";
3 document.cookie = "couponval=20%";
4 document.cookie = "expires=60";
5
6 //delete a cookie
7 //document.cookie=null or document.cookie="" has no effect
8 document.cookie = "name=value; expires=Thu,
                            01-Jan-1970 00:45:00 GMT";
```



Example: simple-cookie-example.html

Reading cookies in JavaScript

- Reading cookies is hard
- `document.cookie["firstname"]` does not work
- String returned by `document.cookie` needs to be parsed

```
1 var cookiesArray = document.cookie.split("; ");
2 var cookies=[];
3
4 for(var i=0; i < cookiesArray.length; i++) {
5   var cookie = cookiesArray[i].split("=");
6   cookies[cookie[0]]=cookie[1];
7 }
```

- Alternative: jQuery cookie plugin (118 lines of code)

jQuery cookie plugin

Create session cookie:

```
$.cookie('name', 'value');
```

Create expiring cookie, 7 days from then:

```
$.cookie('name', 'value', { expires: 7 });
```

Create expiring cookie, valid across entire site:

```
$.cookie('name', 'value', { expires: 7, path: '/' });
```

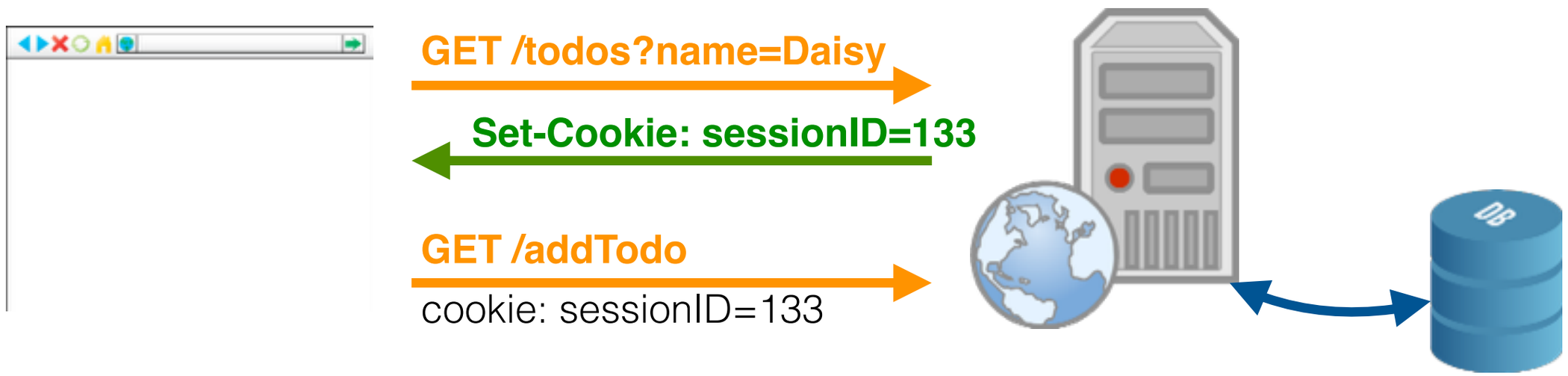
.....

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 client-side cookies and server-side saved data

Sessions in one slide



session store

- Cookies are used to store a single ID on the client
- Remaining user information is stored server-side in memory or a database
- Alternative setup (via URL decoration) is also possible

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

- Very 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

Sessions in express with memory stores


```
1  var express = require("express");
2  var http = require("http");
3  var credentials = require('./credentials.js');
4  var cookies = require("cookie-parser");
5  var sessions = require('express-session');
6  var app = express();
7  app.use(cookies(credentials.cookieSecret));
8  app.use(sessions(credentials.cookieSecret));
9  http.createServer(app).listen(3006);
10
11 app.get("/countMe", function (req, res) {
12   var session = req.session;
13   if(session.views) {
14      session.views++;
15      res.send("You have been here "+session.views+" times!");
16   }
17   else {
18      session.views = 1;
19      res.send("This is your first visit!");
20   }
21 });
```

cookie & session setup

client's session object

session exists!

session does not yet exist

A side node on express

- `app.use()`
  - Add middleware components to your application
  - Decide to which part of the application to limit the component to
- `app.get()`
  - Request routing via GET
  - Every path (URL) you want to make publicly accessible should be defined this way

Sessions are most useful for …

- **Authentication**
- Log in once, and remain logged in for some amount of time

Third-party authentication

Overview

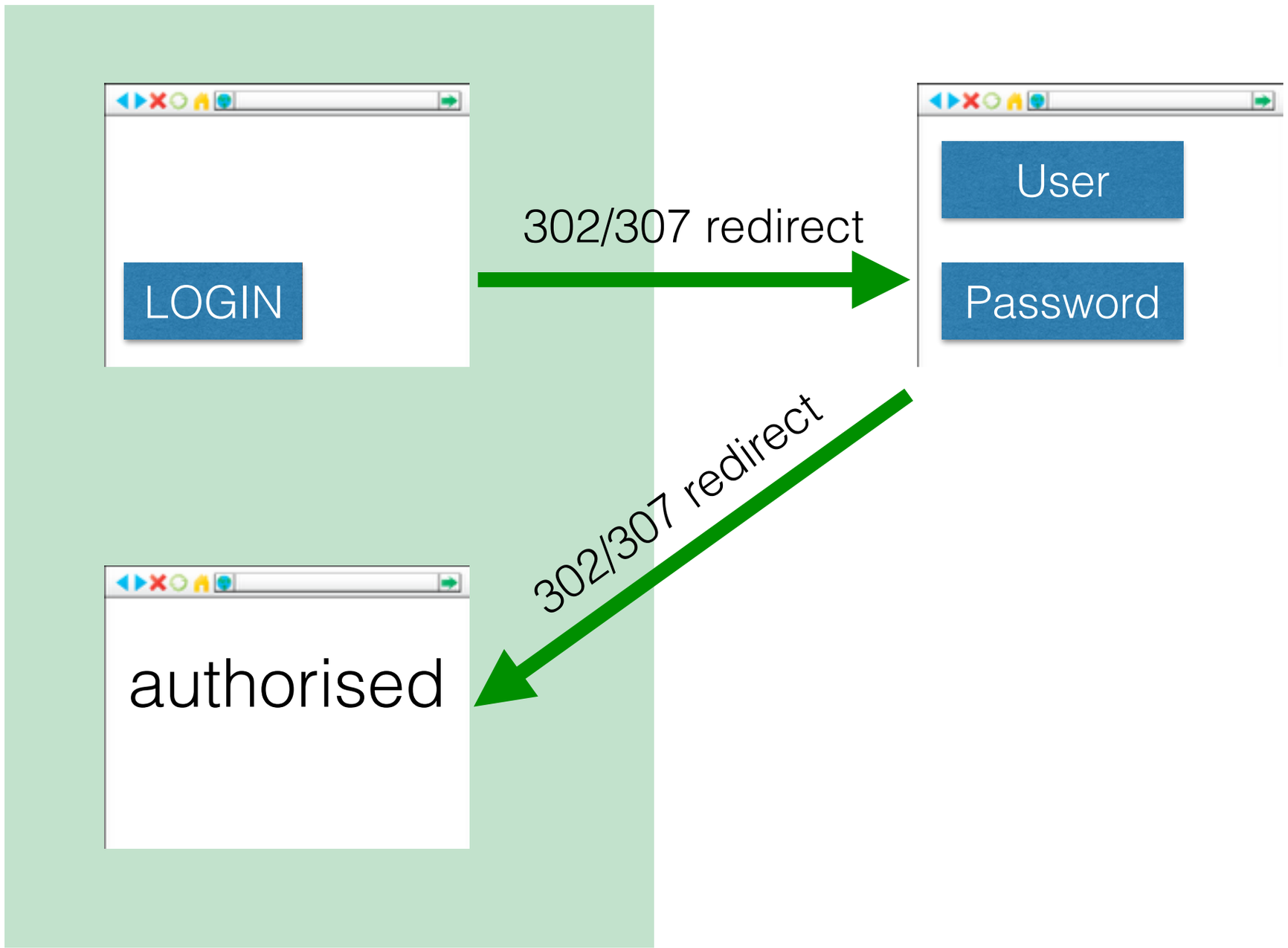
Authentication: verifying a user's identity

- Weakest link in an authenticated application is the user's password [a whole research field by itself!]
- **Application-based decision**
  - Does the app 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
  - node.js packages exist
- **Trusted** social Web platforms **provide authentication**, no need to store passwords or employ particular security measures
- **However**: there are people who do not use social Web platforms or do not want to hand their data out

Third-party authentication depicted



Third-party authentication depicted in detail

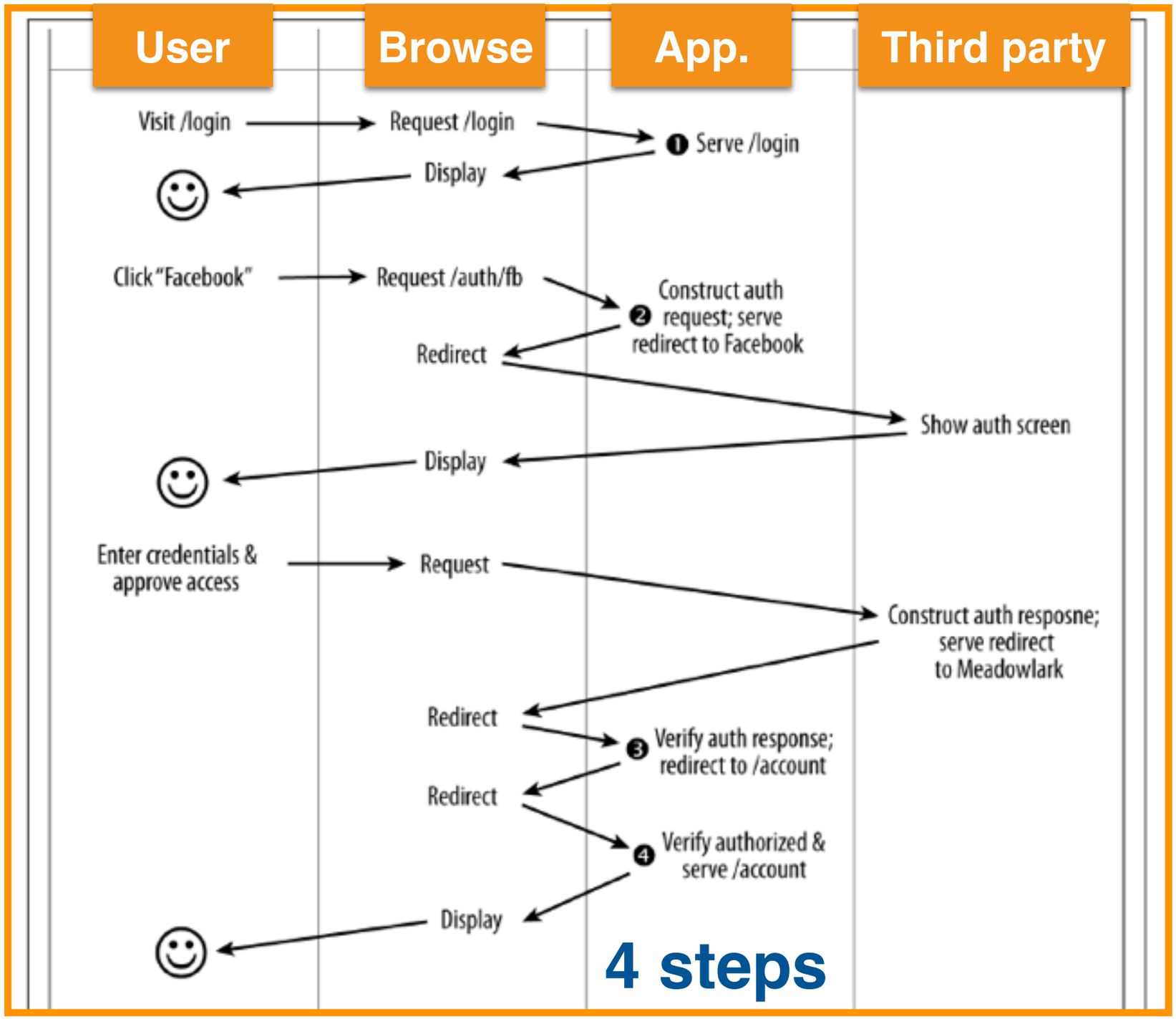


Image source: Web development with Node and Express, p. 222

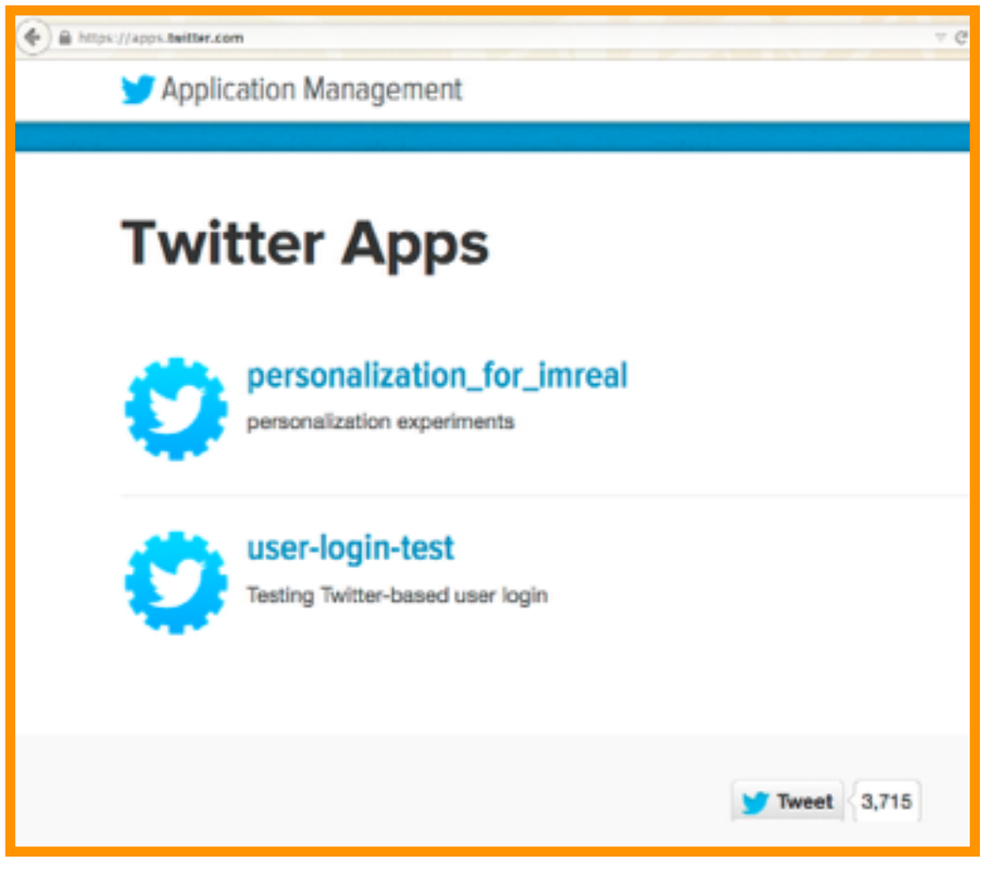
Third-party authentication stepwise

- **Login method**: browser displays options and the user selects one
- **Authentication request construction**: construct request to be send to the third party
  - You can ask for more (name, email, etc.)
  - Requests differ considerably between services
- **Verifying the authentication response**: distinguish between authorised (i.e. valid auth response) and unauthorised access
  - Authorised users should be given a session
- **Verifying the authorisation**: is verified user X allowed to access Y?
  - Store access rights in a database

Third-party authentication
Twitter example

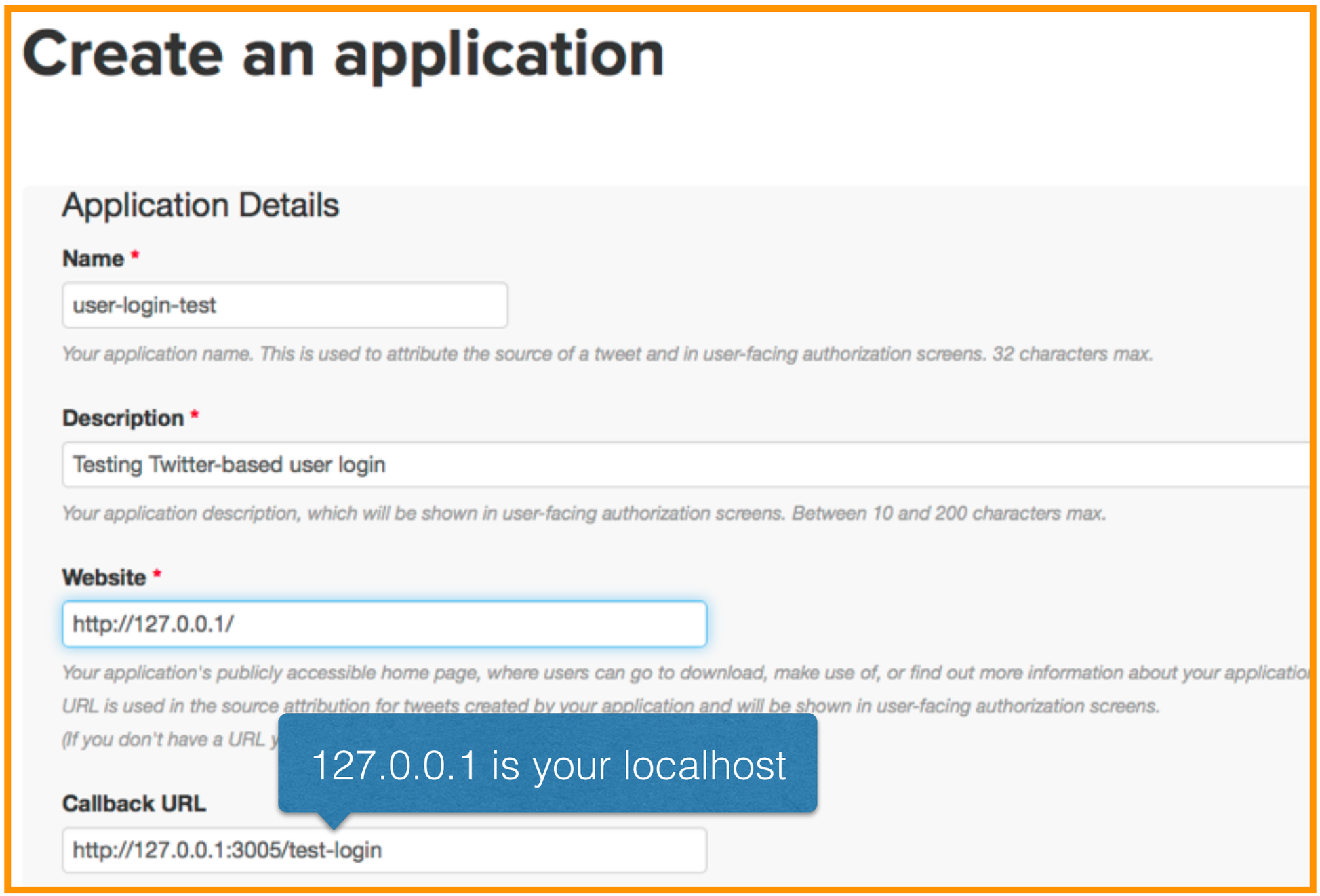
Goal: "Sign in with your Twitter account"

- 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/

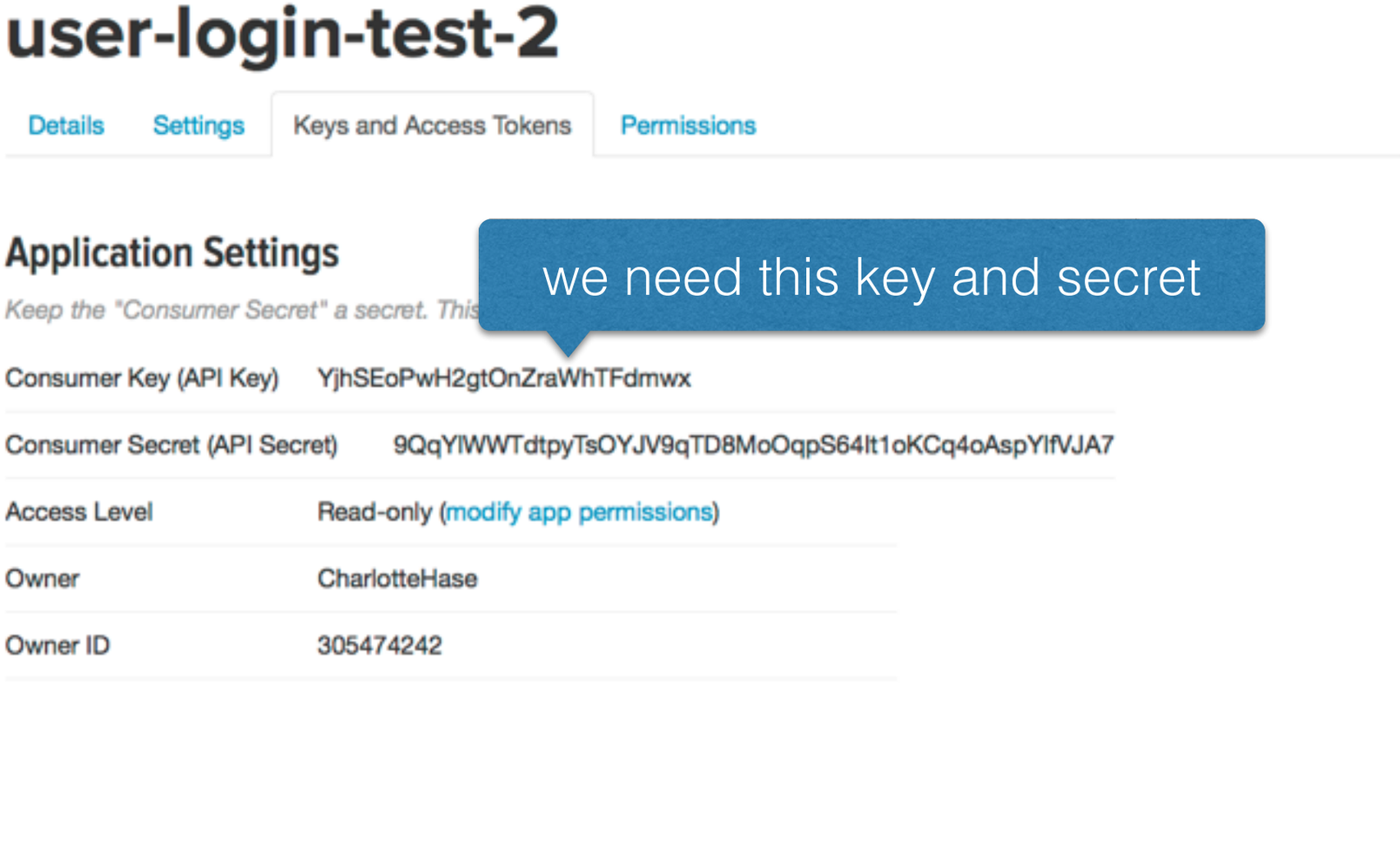
Third-party authentication
Twitter example



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”
- Create access tokens



Third-party authentication Twitter example cont.

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

- Twitter still uses OAuth 1.0, other services use 2.0
  - **`passport`** hides this complexity from you

Installing a strategy

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

Third-party authentication Twitter example cont.

- `passport` has a lot of boilerplate code (copy & paste)
- Ensure that you set your own key and secret
- Ensure that you call the middleware components in the right order (otherwise errors will occur)
- Ensure that you do not mix “localhost” and 127.0.0.1
- **Write yourself**:
  - Server-side node.js script
  - Client-side HTML

Third-party authentication Twitter example cont.

```
1  // Redirect the user to Twitter for authentication.
2  app.get('/auth/twitter', passport.authenticate('twitter'));
3
4  // Twitter will redirect the user to this URL after approval.
5  app.get('/test-login',
6    passport.authenticate('twitter', { failureRedirect: '/failure' }),
7    function(req, res) {
8      res.redirect('/success');
9    });
10
11 app.get("/success", function (req, res) {
12   console.log("Success!");
13   res.send("User login via Twitter successful!");
14 });
15
16 app.get("/failure", function (req, res) {
17   console.log("Failure!");
18   res.send("User login via Twitter was unsuccessful!");
19 });
```

Excerpt from the full node.js script

Third-party authentication
Twitter example cont.

```
1 <!doctype html>
2    <head>
3    </head>
4    <body>
5       <a href="/auth/twitter">Sign in with Twitter</a>
6    </body>
7 </html>
```

OAuth2

RFC 6749

> "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. "

- OAuth: Open Standard to Authorization
- OAuth 2 was finalised in 2012
- OAuth 2 is not backwards compatible with OAuth 1
- Both OAuth 1 & 2 are still in use

Summary

- Cookies
- Sessions
- Third-party authentication

End of Lecture