Verteilte Systeme 1

Technologien des World Wide Web

christian.zirpins@hs-karlsruhe.de

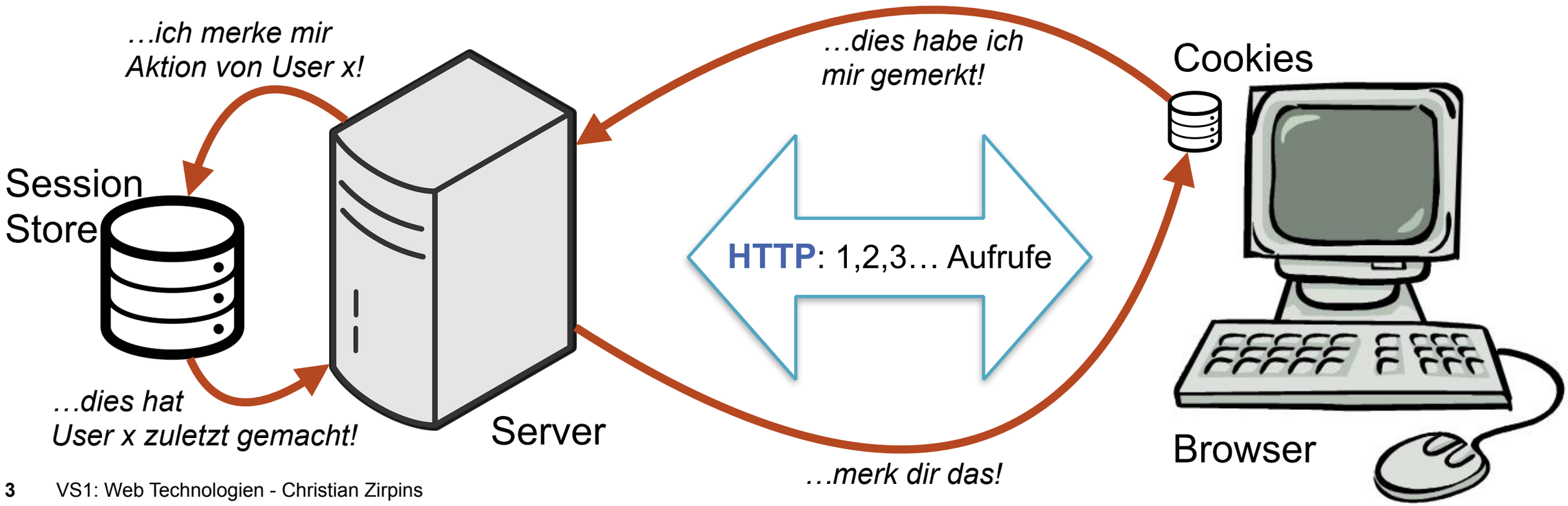
Web Apps Personalisieren

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Heutige Lernziele

Nach dieser Vorlesung können Sie…

- …für ein gegebenes Nutzungsszenario entscheiden, ob **Cookies** und / oder **Sessions** geeignet sind
- …Code zum Erstellen / Ändern / Löschen von **Cookies** implementieren
- …Code zum Erstellen / Ändern / Löschen von **Sessions** implementieren
- …**Third-Party-Authentifizierung** implementieren



Einführung von Cookies und Sessions

Erinnerung: das HTTP-Protokoll

- HTTP ist ein **zustandsloses** Protokoll
- Jede HTTP-Anfrage enthält **alle Informationen**, die für eine Antwort erforderlich sind
- Der Server muss sich die erfolgten Anfragen nicht merken

- Vorteil: Vereinfacht die Server-Architektur
- Nachteil: Clients müssen bei jeder Anfrage **die gleichen Informationen** neu senden

Viele Aktivitäten erfordern dauerhaften Zustand

- amazon.de merkt sich den Einkaufskorb, auch wenn man die Webseite verlässt
- JavaScript Spiele merken sich den Spielstand wenn man auf die Seite zurückkehrt
- Webseiten können Nutzern mitteilen, wie oft sie die Seite schon besucht haben
- Video Streaming (wenn Sie Zeit haben, schauen Sie sich dieses Video über die Vorzüge von JavaScript an) https://www.youtube.com/watch?v=bo36MrBfTk4

Cookies

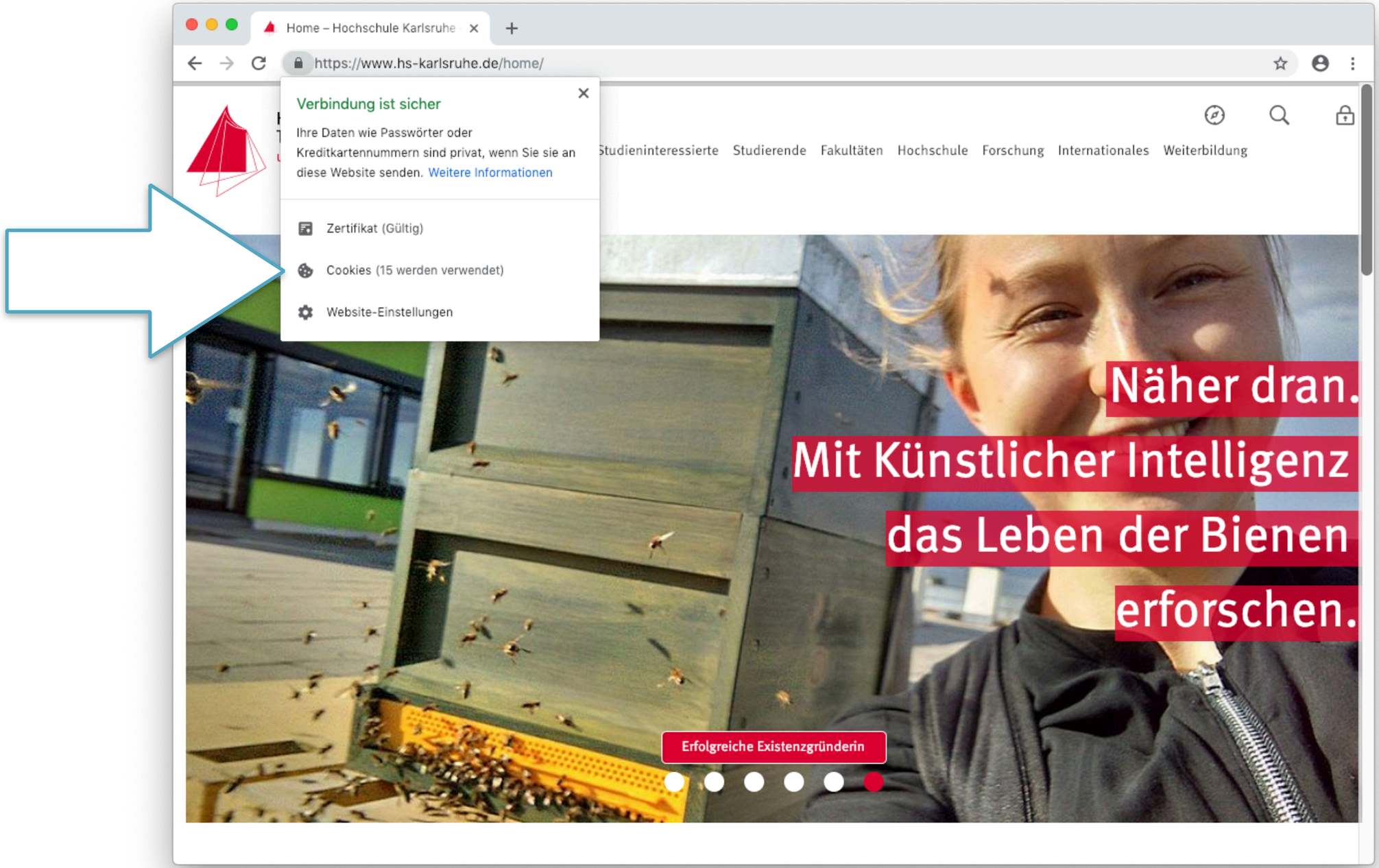
Cookies und **Sessions** sind Möglichkeiten, um **Zustand** auf Basis des zustandslosen HTTP Protokolls zu realisieren

Cookie: ein kurzer Text (Key/Value) der **vom Server gesendet** wird und **den der Client für einige Zeit speichert**

Minimale Anforderung für Browser bzgl. Menge/Umfang (RFC6265)

- Min. 4096 Bytes pro Cookie
- Min. 50 Cookies pro Domain
- Min. 3000 Cookies insgesamt

Wo (und wie) findet man Cookies?



Beispiel: Öffnen Sie die Hochschulseite https://www.hs-karlsruhe.de/ z.B. mit Chrome und Klicken Sie neben der Adresse auf die Verbindungsinfo

Cookie & Session Grundlagen

- Cookies sind **sichtbar** für die Benutzer (die es möchten)
  - Standardmäßig als Klartext gespeichert
- Clients (Benutzer, d.h. Sie!) Können Cookies **löschen** oder **verbieten**
- Cookies können vom Client **geändert** werden!
  - Öffnet eine Angriffslinie: Server sollten keine sensiblen Informationen in einfachen Cookies senden

Eine sehr alte Web-Technologie! Entwickelt 1994.

- **Sessions** sind Cookies vorzuziehen
  - Sessions selbst nutzen Cookies
  - Cookie enthält meist nur einen einzigen Wert (Session-ID), der Rest wird auf dem Server gespeichert

Cookies können nicht…

- **…Programme ausführen**
- …Informationen von der **Festplatte des Benutzers** zugreifen
- …Spam generieren
- **…von anderen Teilnehmern gelesen werden**
  - Nur der Server, der sie gesetzt hat, kann die Cookies lesen
  - Aber: passen Sie auf Third-Party Cookies auf!

Cookie Grundlagen

Client sendet den Cookie mit jedem Request!

- Kodiert im HTTP Header
- Web Frameworks haben spezielle Methoden, um mit Cookies zu arbeiten
- Cookies werden an den **Domain Namen** der Seite **gebunden** und nur bei Requests an diese spezifische Seite zurückgesendet

Was kann man in Cookies speichern?

- Cookies sind das **Kurzzeitgedächtnis des Servers**
- Der Server entscheidet über die Informationen im Cookie

- **Beispiele**:
  - Login Name des Benutzers
  - Historie der Seitenansichten
  - Eingaben in Formularfeldern

Session Cookies vs. Persistent Cookies

- **Session Cookies** (auch **Transient Cookie**)
  - Existieren nur im Speicher; beim Schließen des Browsers gelöscht
  - Cookies sind Session Cookies, wenn sie kein Verfallsdatum haben
- **Persistent Cookies**
  - Cookies bleiben beim Schließen des Browsers erhalten
  - Sie haben ein **maximales Alter**
  - Werden zum Server übertragen, solange sie gültig sind

Frage: Was bewirkt wohl das Klicken auf den Knopf "Dies ist ein öffentlicher Computer" vor dem Login einer Web App?

Cookie Verfallsdatum

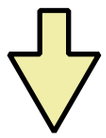
Einige beispielhafte Messungen der Cookie-Typen und Verfallsdaten bekannter Seiten (aus 2017)

- **Gmail**: 13 Cookies (2 Session), Verfallsdaten in 2017, 2019
- **Facebook**: 12 Cookies (3), Verfallsdaten in 2017, 2019
- **Amazon**: 12 Cookies (2), Verfallsdaten in 2017, 2035, 2036, 2037

Cookie HTTP Header: Parameter

- `Name=Value` (einziger geforderter Parameter)
- `Expires` (UNIX Timestamp) oder `Max-Age`
- `Domain`, an die der Cookie gebunden ist
- `Path`, für den der Cookie gilt (automatische Wildcard),
  z.B. gilt **/** für alle Seiten, **/info** für Seiten unterhalb von info etc.
- `Secure`, `HttpOnly`, `Signed` verschiedene Schalter

```
HTTP/2.0 200 OK
Content-type: text/html
Set-Cookie: id=a3fWa; Expires=Wed, 21 Oct 2025 07:28:00 GMT;
```



Beispiel: HTTP Nachrichten mit Cookie Header

```
GET /sample_page.html HTTP/2.0
Host: www.example.org
Cookie: id=a3fWa
```

Cookies robuster machen

- **Secure** Cookies
  - Ist das Attribut gesetzt, dürfen Cookies nur über HTTPS gesendet werden (d.h. sie sind bei der Übertragung verschlüsselt)
- **HttpOnly** Cookies
  - Cookies sind nicht außerhalb von HTTP zugreifbar (z.B. JavaScript)
  - Minimiert die Gefahr des Cookie Diebstahls
  - Nur bei kompatiblen Browsern
- **Signed** Cookies
  - Stellt sicher, dass Cookies vom Client nicht geändert wurden

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(3000);

app.get("/sendMeCookies", function(req, res) {
    console.log("Cookies sent");
    res.cookie("chocolate", "monster");
    res.cookie("signed_choco", "monster", {signed: true});
    res.send();
});

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

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

Cookie Parser Middleware

Cookie erzeugen

Cookie signieren

Cookies lesen

simple-cookies.js

Cookies in Express

- *EditThisCookie* für *Chrome*: erlaubt es, Cookies zu ändern (in neueren Versionen auch *Chrome Dev Tools*)
- Was ist der Unterschied zwischen signierten und nichtsignierten Cookies?
  - Prüfe Cookie Erstellung
  - Ändere nichtsignierte Cookies
  - Andere signierte Cookies (erst Name dann Wert)

simple-cookies.js

Selbst probieren:
https://github.com/zirpins/vs1lab/tree/master/Beispiele/nodejs_cookies

Cookies in Express

- Den Wert eines speziellen Key/Value Paares zugreifen

```
var val = req.signedCookies.signed_choco;
```

- Cookie löschen

```
res.clearCookie('chocolate');
```

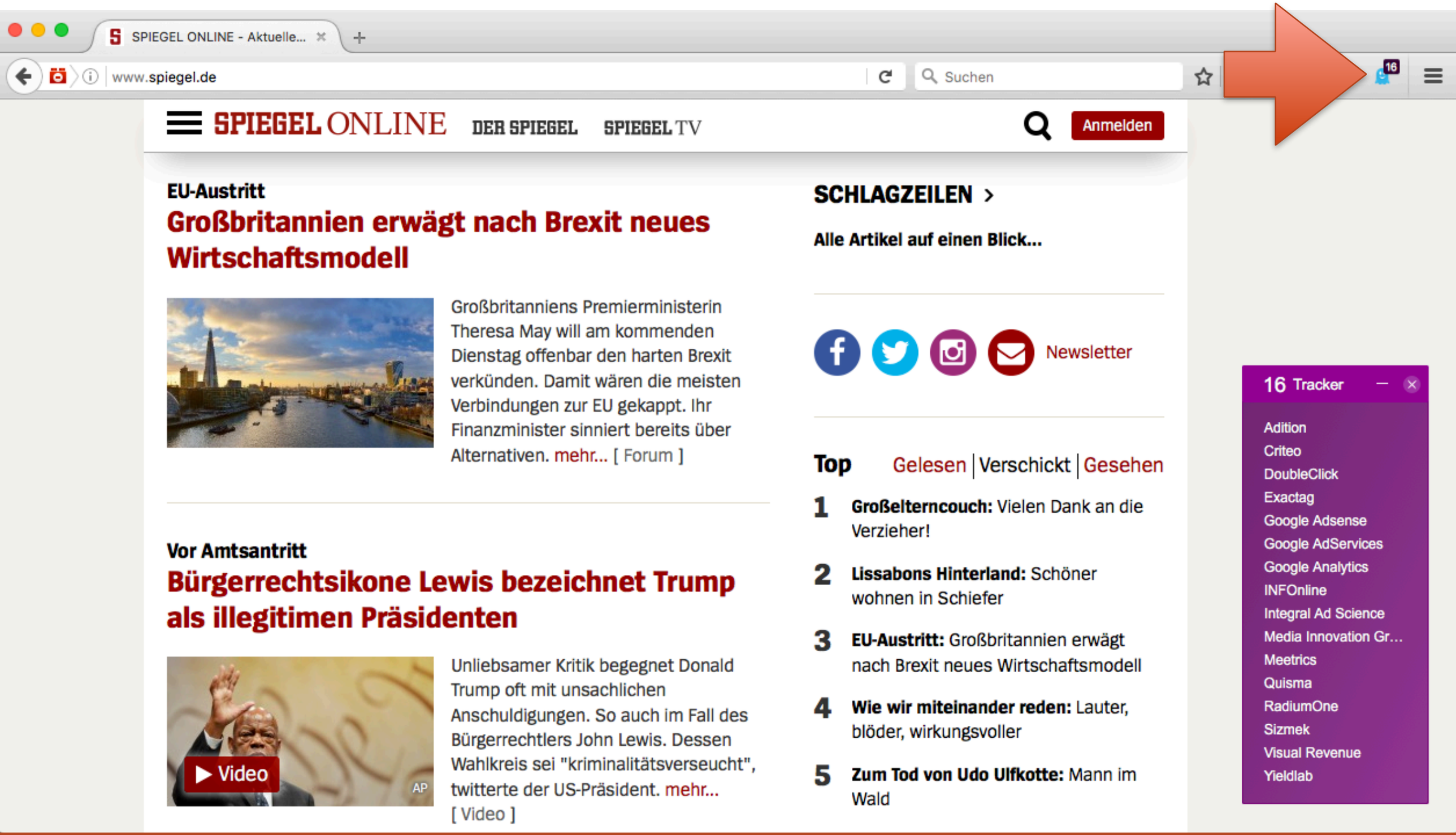
Cookie Key

Löschen im Response!

Ein pessimistischerer Blick auf Cookies

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Wir werden oft ohne unser Wissen verfolgt



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

Third-Party Cookies

Set-Cookie: x.org

Web Server x.org

First Party

http://x.org/

Einge-bettete Inhalte (Videos, Karten, Analyse, Werbung etc)

Third Party

Set-Cookie: ads.agency.com

Globale Werbeagentur ads.agency.com

Von der gleichen Agentur

Benutzer besucht drei verschiedene Websites

http://y.org/

Set-Cookie: ads.agency.com

http://z.org/

Set-Cookie: ads.agency.com

Basierend auf den Cookies können Profile erstellt werden

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: https://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: https://samy.pl/evercookie/

Clientseitige Cookies

Cookies in JavaScript

- Cookies müssen nicht immer vom Server kommen
- Cookies können im Browser gesetzt werden
- Meist, um Formulareingaben zu erinnern

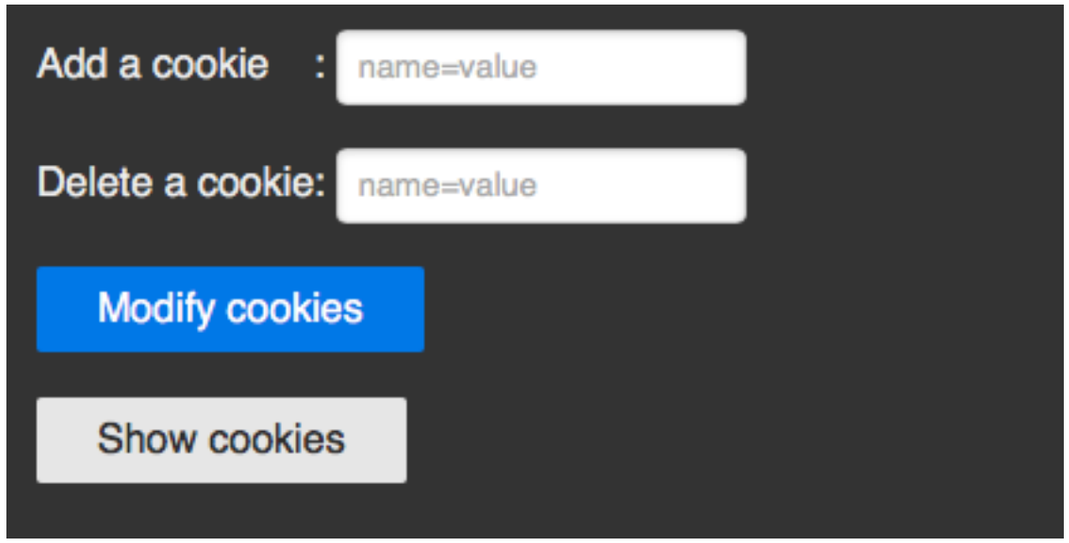
```
//set a cookie
document.cookie = "name=value";
document.cookie = "name=value; expires=Fri, 24-Jan-2014 12:45:00 GMT";

//delete a cookie
document.cookie = "name=value; expires=Fri, 24-Jan-1970 12:45:00 GMT";
```

`document.cookie` ist anders

```
// drei Cookies hinzufügen
document.cookie = "couponnum=123";
document.cookie = "couponval=20%";
document.cookie = "expires=60";

// Cookie löschen
// document.cookie=null oder document.cookie="" ohne Wirkung
document.cookie = "name=value; expires=Thu, 01-Jan-1970 00:45:00 GMT";
```



Selbst probieren:
https://github.com/zirpins/vs1lab/tree/master/Beispiele/browser_cookies

Cookies in JavaScript lesen

- Cookies lesen ist schwierig
- `document.cookie["firstname"]` funktioniert nicht
- String als Rückgabe von `document.cookie` parsen

```
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: jQuery Cookie Plugin (118 LOC)

Sessions

Session Übersicht

- **Szenario**: Benutzer interagieren für einige Zeit kontinuierlich mit einer Website (Sitzung bzw. **Session**)
- **Ziele**:
  - Benutzer während der Interaktion begleiten, ohne sich dabei (zu sehr) auf unzuverlässige Cookies verlassen zu müssen
  - Es können größere Datenmengen gespeichert werden
- **Problem**: Ohne Cookies kann der Server die Clients nicht auseinander halten
- **Lösung**: hybrider Ansatz zwischen clientseitigen Cookies und Serverseitig gespeicherten Daten

Sessions auf einer Folie

- Cookies werden verwendet, um ID auf dem Client zu speichern
- Übrige Benutzerdaten werden auf dem Server gespeichert
- Alternative Konfiguration ("URL Decoration") auch möglich

Eine Session Aufbauen

1. Client fordert erste Seite vom Server an
2. Server erstellt eindeutige Sitzungs-ID und initiiert Speicherung der Session-Daten für Client
3. Server sendet eine Seite mit Cookie, das die Session-ID enthält
4. Nun sendet der Client Requests zusammen mit dem Cookie
5. Server kann die ID verwenden, um Antworten zu personalisieren
6. Session endet, wenn keine Requests mit der ID kommen (Timeout)

Sessions in Express mit Memory Store

- Einfach zu realisieren in Express
- Gleicher Nachteil wie jeder **"In-Memory"-Store**: nicht persistent bei Ausfall von Maschinen
- Eine Middleware Komponente hilft: `express-session`: https://github.com/expressjs/session

Sessions in Express mit Memory Store

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

app.use(sessions({
    secret: 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!");
    }
});
```

Session Konfiguration

Client Session Objekt

Session existiert

Session existiert noch nicht

Bemerkung am Rande: Express Konfiguration

- `app.use()`
  - Füge Middleware Komponenten zur Anwendung hinzu
  - Entscheide, auf welchen Teil der App die Komponente beschränkt ist
- `app.get()`
  - Request Routing bei HTTP GET Operation
  - Jeder Pfad (URL), der öffentlich zugreifbar sein soll, sollte derart deklariert werden

Session bringen am meisten für…

- **Authentifizierung**
  - Einmal anmelden und dann für einen gewissen Zeitraum angemeldet bleiben

Third-Party Authentifizierung

Übersicht

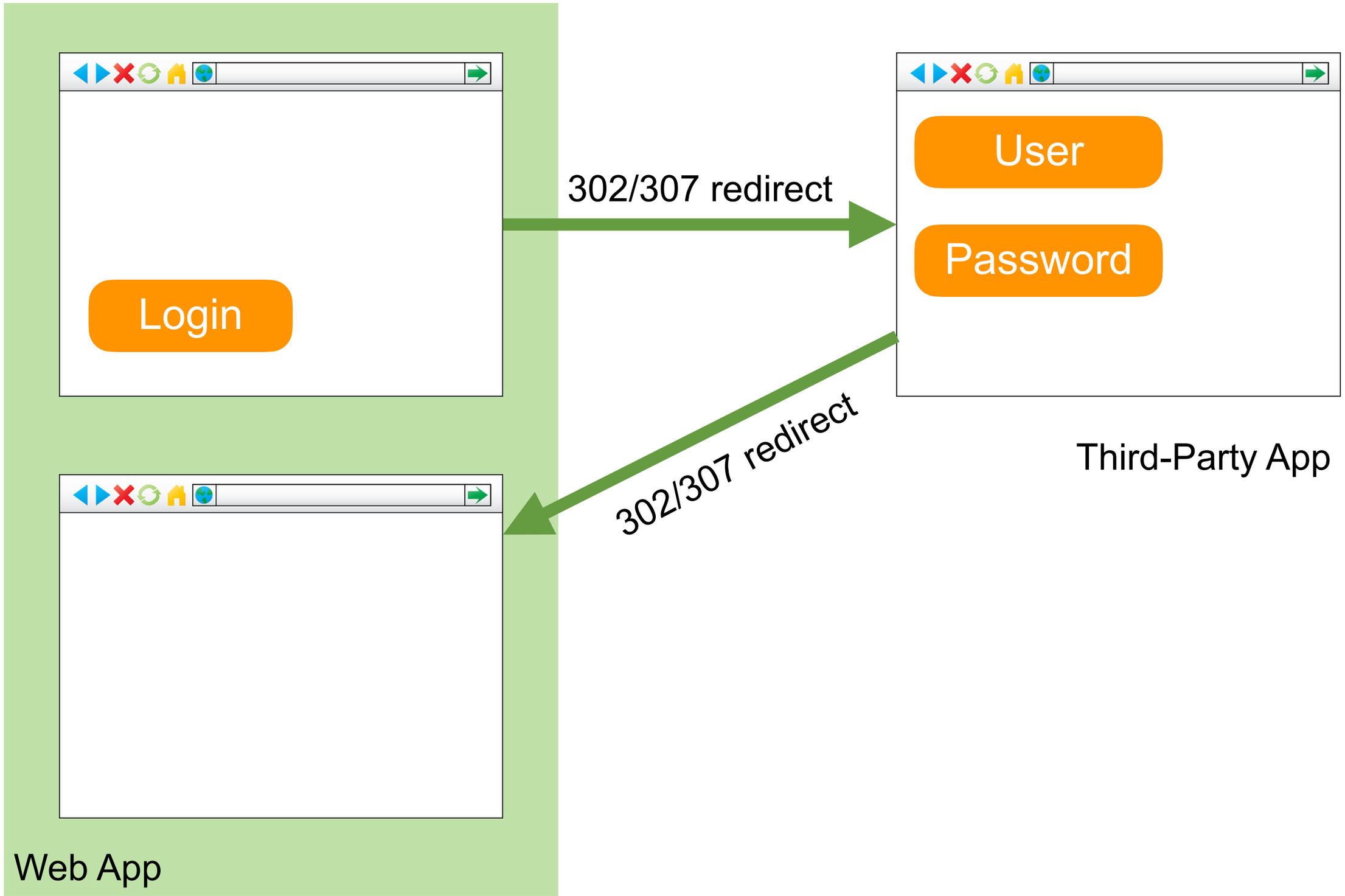
Authentifizierung: die Identität eines Benutzers verifizieren

- Schwächstes Glied einer authentifizierten App ist das Passwort
- **Entscheidung der App**
  - Braucht die App Authentifizierung?
    Sind Cookies/Sessions genug?
  - Wenn Authentifizierung gebraucht wird, soll diese durch Dritte durchgeführt werden (**Third-Party Authentication**)?
    - Geringe kognitive Last für den Benutzer

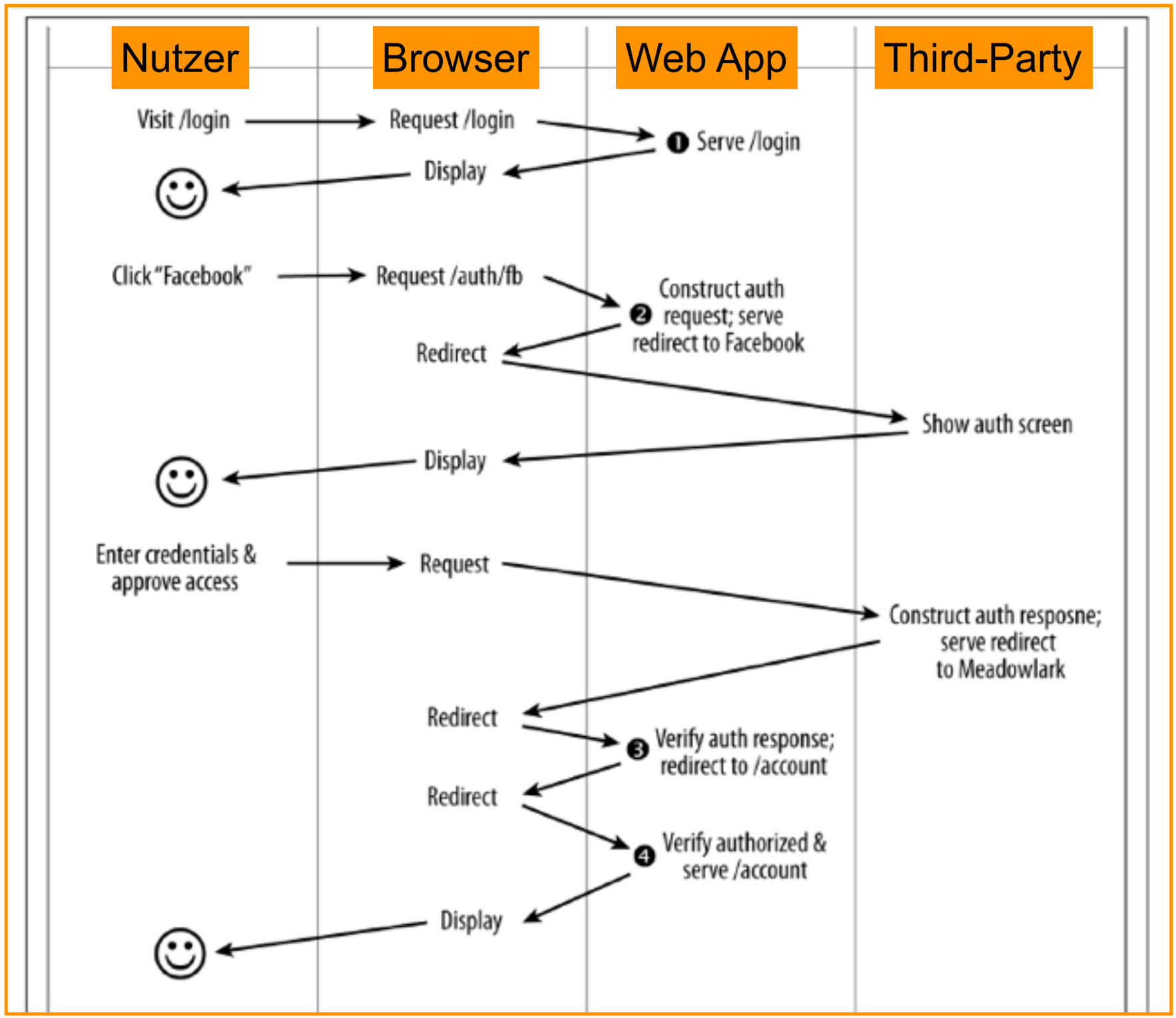
Third-Party Authentifizierung

- Benutzer durch Social Web Dienste authentifizieren (Twitter, Facebook, Google, LinkedIn, etc.)
- **Einfach** zu entwickeln
  - Es gibt entsprechende node.js Pakete
- Vertrauenswürdige Social Web Plattformen **erbringen Authentifizierung**, keine Notwendigkeit zur Speicherung von Passwörtern oder für spezielle Sicherheitsmaßnahmen
- **Aber**: einige Benutzer verwenden keine Social Web Plattformen oder wollen ihre Daten nicht preisgeben
  - Wenn etwas im Web nichts kostet, dann sind die Benutzer (also wir!) nicht Kunde sondern Produkt

Third-Party Authentifizierung: Funktionsprinzip



Third-Party Authentifizierung: im Detail



Quelle: Web development with Node and Express, p. 222

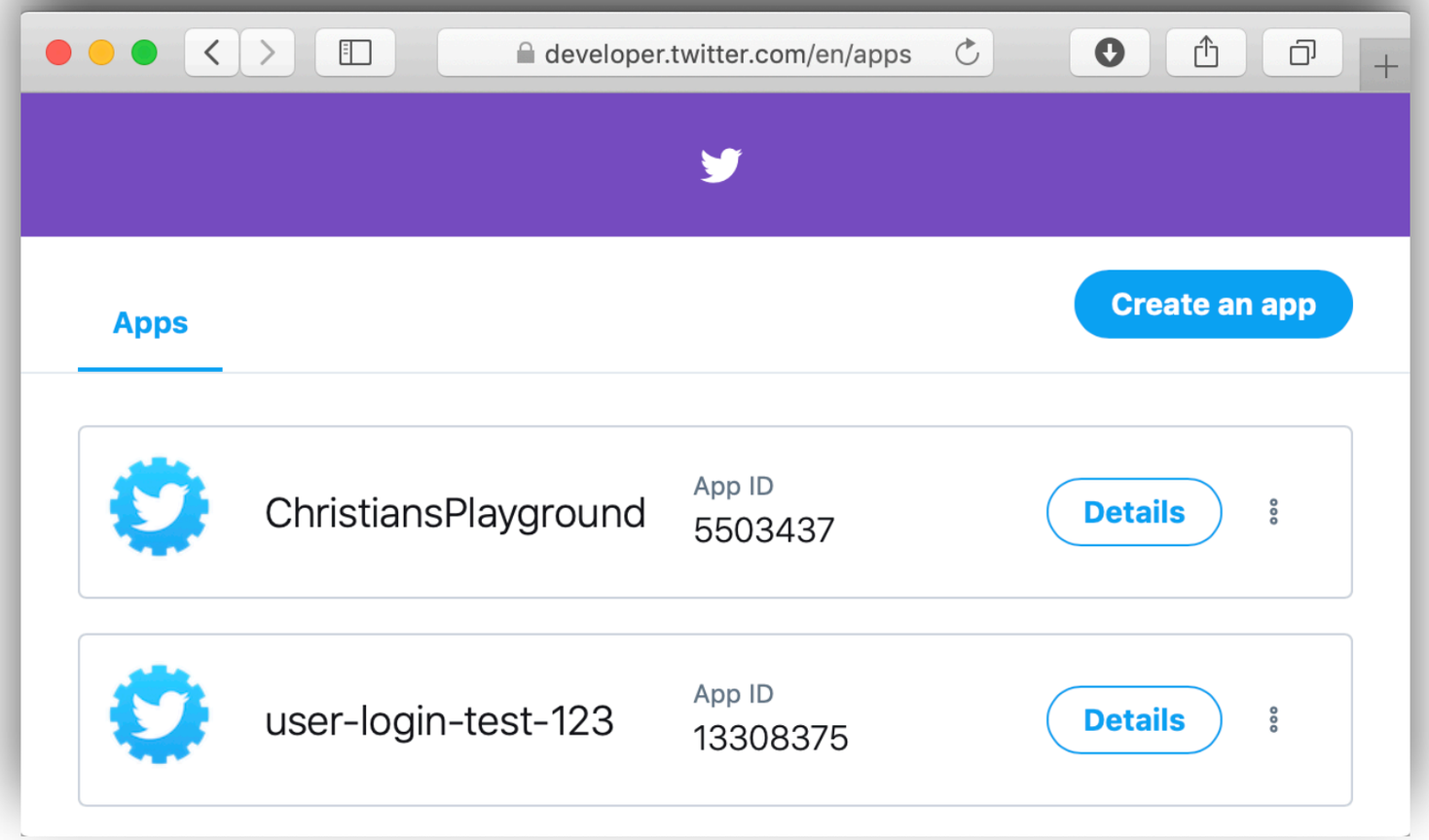
Third-Party Authentifizierung: Schritte

- **Anmeldemethode**: Browser zeigt Optionen / Benutzer wählt eine
- **Authentication-Request erstellen**: zum Senden an Third Party
  - Es kann mehr abgefragt werden (Name, E-Mail usw.)
  - Anfragen unterscheiden sich erheblich zwischen den Diensten
- **Überprüfen der Authentication-Response**: Unterscheide autorisierten (gültige Auth-Response) und nicht autorisierten Zugriff
  - Autorisierte Benutzer sollten eine Sitzung erhalten
- **Überprüfen der Berechtigung**: ist verifizierter Benutzer X berechtigt, auf Y zuzugreifen?
  - Speichern von Zugriffsrechten in einer Datenbank

Third-Party Authentifizierung: Twitter

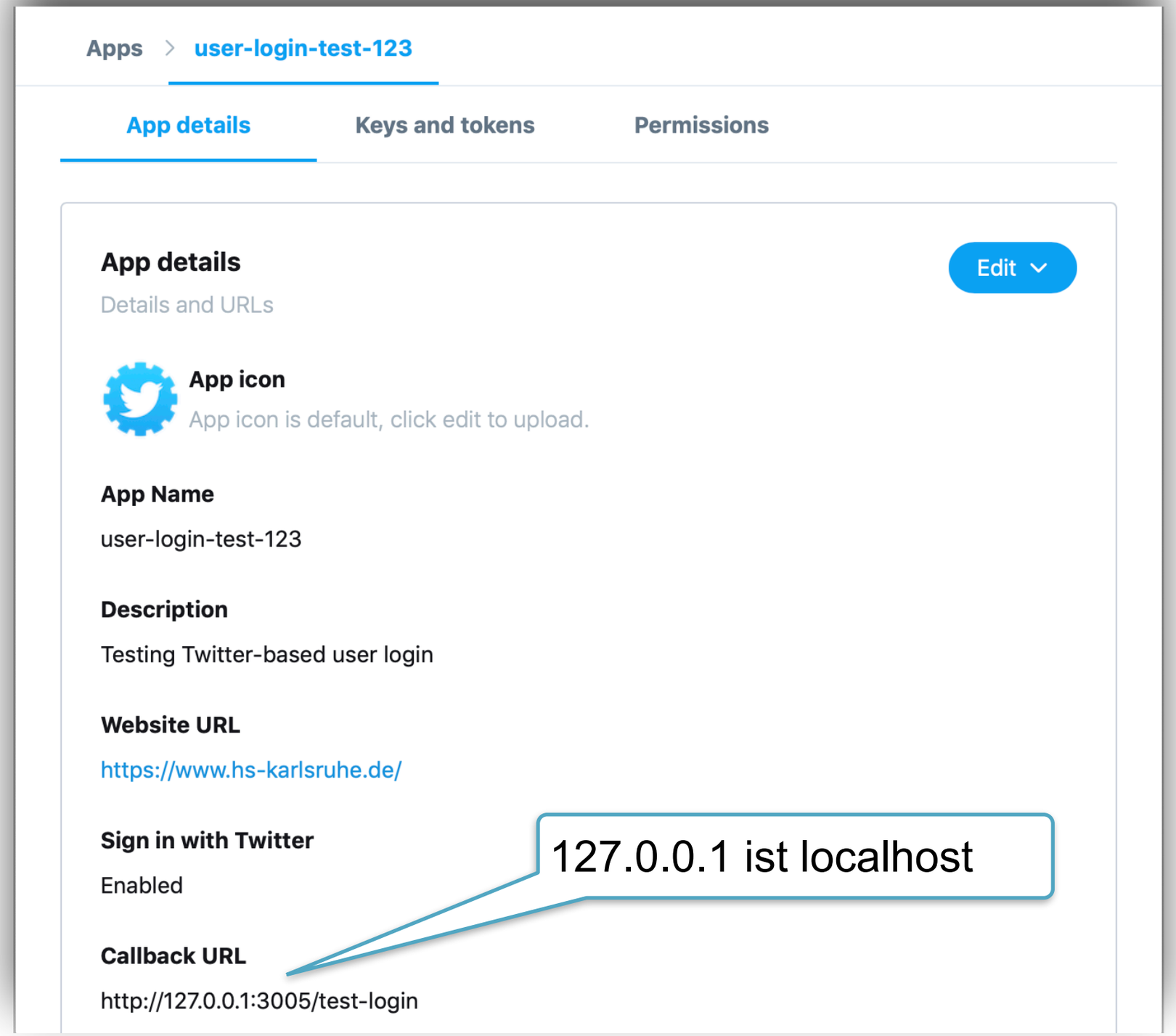
Ziel: "Anmelden mit Twitter Konto"

- Funktioniert ähnlich bei verschiedenen Diensten
- Startpunkt: **erzeuge** eine „App" (Twitter App, Facebook App etc.)



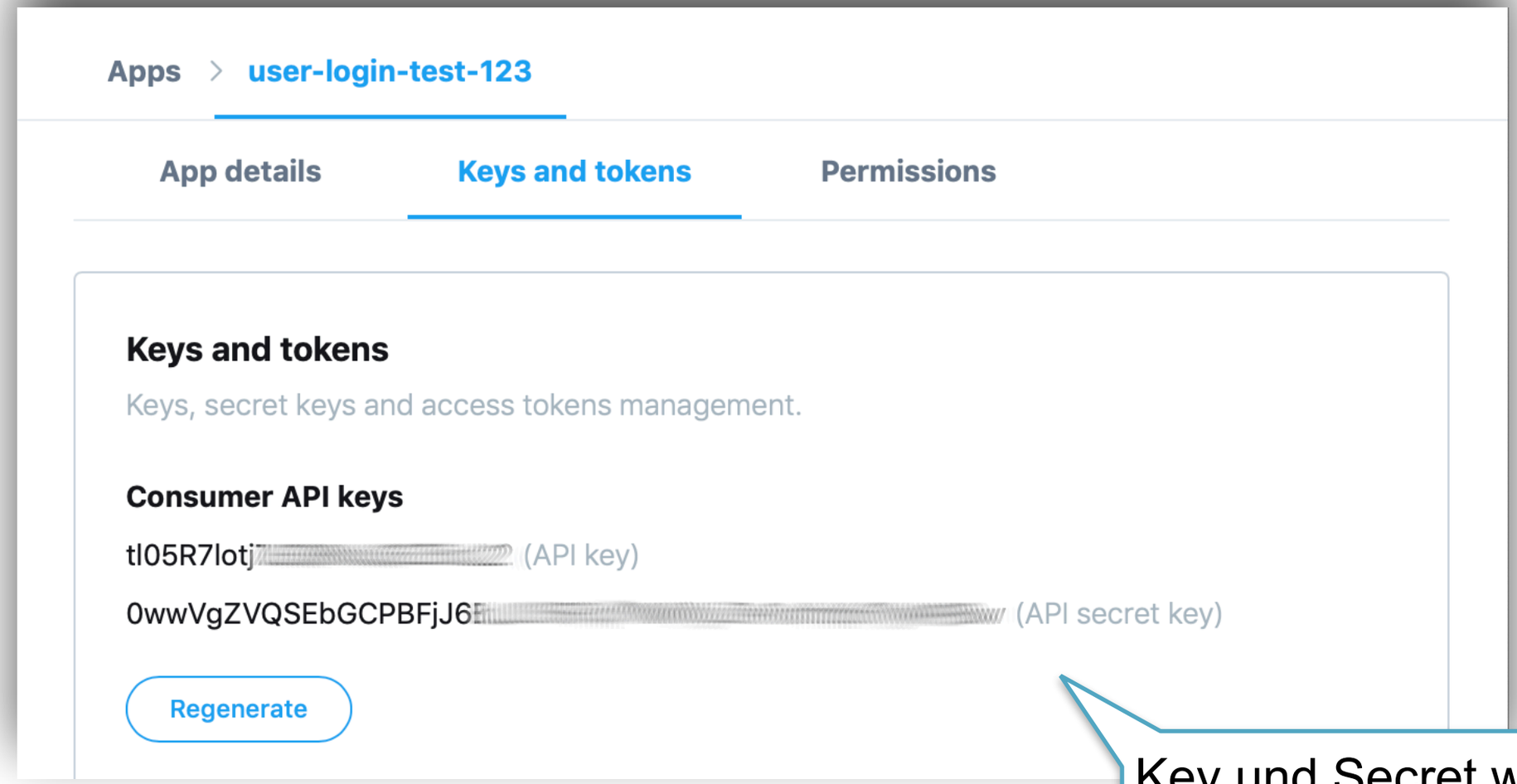
https://developer.twitter.com/

Third-Party Authentifizierung: Twitter



Third-Party Authentifizierung: Twitter

- In den App Settings auswählen: "Allow this application to be used to Sign in with Twitter"
- Access Tokens erzeugen



Third-Party Authentifizierung: Twitter

- Express kann `passport` nutzen, eine der populärsten Middleware Komponenten zur Authentifizierung
  - 140+ Authentifizierungsstrategien
  - Unterstützt OpenID und OAuth

- Twitter nutzt OAuth
  - `passport` versteckt die Details

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

Installiert eine Strategie

Third-Party Authentifizierung: Twitter

- `passport` hat viel Boilerplate Code (Copy & Paste)
- **Wichtig**: eigenen **Key** und eigenes **Secret** setzen
- **Wichtig**: Middleware Komponenten in der richtigen Reihenfolge aufrufen
- **Wichtig**: nicht `localhost` und `127.0.0.1` vermischen
- **Versuchen Sie es selbst (Beispiel auf Github)**
  - Erstellen Sie eine Twitter App
  - Passen Sie die Credentials an

Third-Party Authentifizierung: Twitter

```
// Leite Benutzer für Authentifizierung zu Twitter um
app.get('/auth/twitter', passport.authenticate('twitter'));

// Nach Bestätigung leitet Twitter Benutzer zu dieser URL um
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!");
});
```

Auszug aus dem kompletten node.js Skript

Third-Party Authentifizierung: Twitter

```
<!doctype html>

<head>
</head>

<body>
    <a href="/auth/twitter">Sign in with Twitter</a>
</body>

</html>
```

OAuth2

"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: offener Standard für Autorisierung
- OAuth 2 wurde 2012 fertiggestellt
- OAuth 2 ist nicht Rückwärtskompatibel zu OAuth 1
- OAuth 1 & 2 werden beide noch genutzt

Zusammenfassung

Themen heute waren

- Cookies
- Sessions
- Third-party Authentifizierung

Literatur

- A. Barth, *"HTTP State Management Mechanism"*, IETF 6265, 2011 https://www.ietf.org/rfc/rfc6265.txt
- Ethan Brown, *"Web development with Node and Express",* O'Reilly, 2014
- OAuth, https://oauth.net