












Blockwoche: Web Programming Lab

Programm Blockwoche

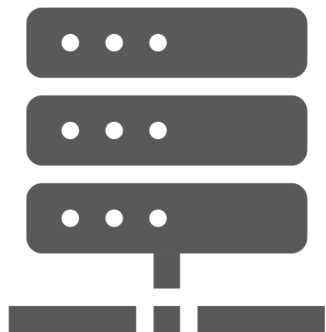
Montag 	Dienstag  	Mittwoch  	Donnerstag   	Freitag   
Architekturansätze von Web Anwendungen JavaScript Sprachkonzepte I	Client-Side- JavaScript I	Angular	Angular	Progressive Web Apps
JavaScript Sprachkonzepte II	Client-Side- JavaScript II	Angular	Server-Side- JavaScript REST	Authentication @ Web Apps

Agenda

1. CORS
2. Authentication @WebApps
 - Basic Authentication
 - Digest Authentication
 - JWT
 - HMAC Authentication (Amazon)
 - OAuth
 - FIDO
3. Abschluss

Input CORS

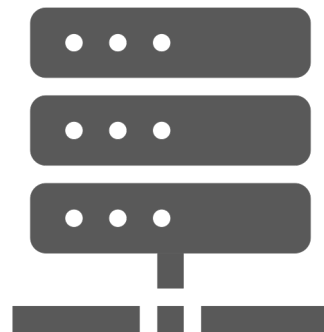




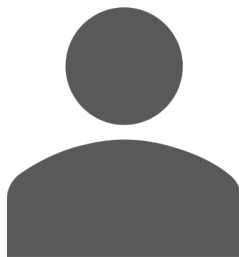
api.myapp.ch

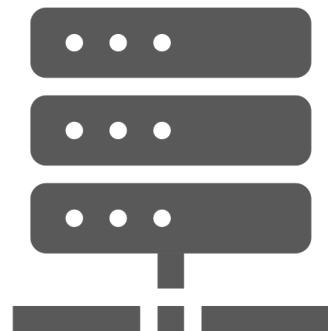


app. myapp.ch



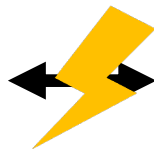
admin-api.myapp.ch



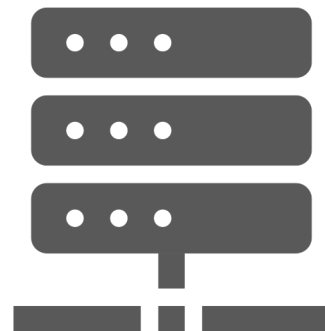
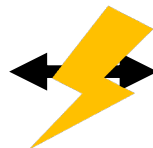


api.myapp.ch

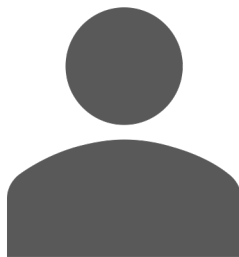
Same Origin Policy SOP



app.myapp.ch



admin-api.myapp.ch



Same Origin Policy SOP

- Sicherheitskonzept von Web Browsers
- Wurde 1996 von Netscape eingeführt
- Untersagt Clientseitigen Scriptsprachen auf Objekte von anderen Origins zuzugreifen.
- Möglichkeiten für Web Applications:
 - Frontend & Backend auf selber Origin
 - CORS

Cross-Origin Resource Sharing (CORS)

- Mechanismus, der Cross-Origin Requests ermöglicht
- CORS ist ein Kompromiss zugunsten größerer Flexibilität im Internet unter Berücksichtigung möglichst hoher Sicherheitsmassnahmen.
- Konfiguration via HTTP Headers:
`Access-Control-Allow-Origin`
`Access-Control-Allow-Headers`
`Access-Control-Allow-Methods`

Cross-Origin Resource Sharing (CORS)

- **Simple CORS**

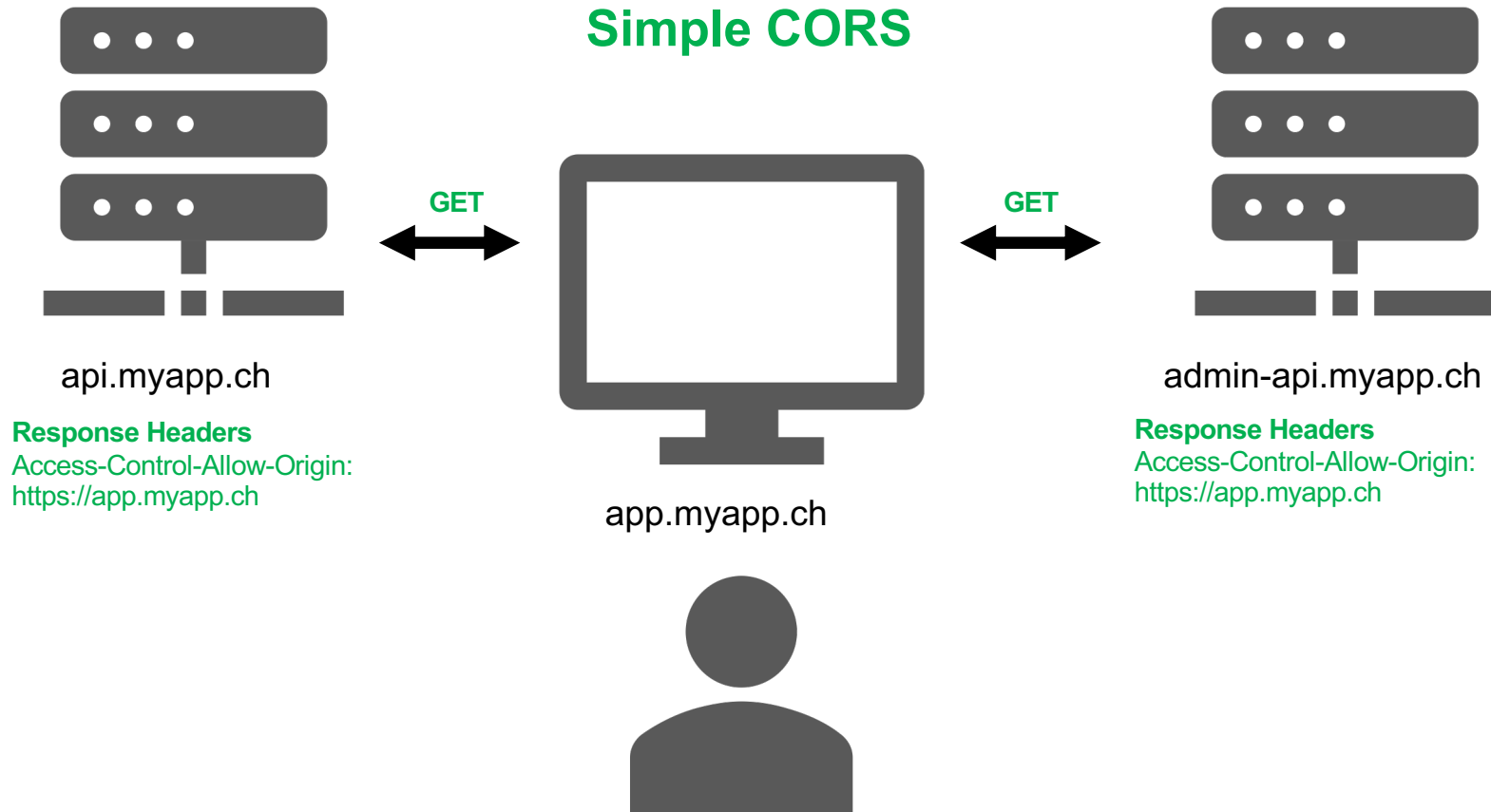
GET & POST mit begrenzten Headers

- **CORS Preflight Requests**

Für weitere HTTP Methoden oder zusätzliche Headers

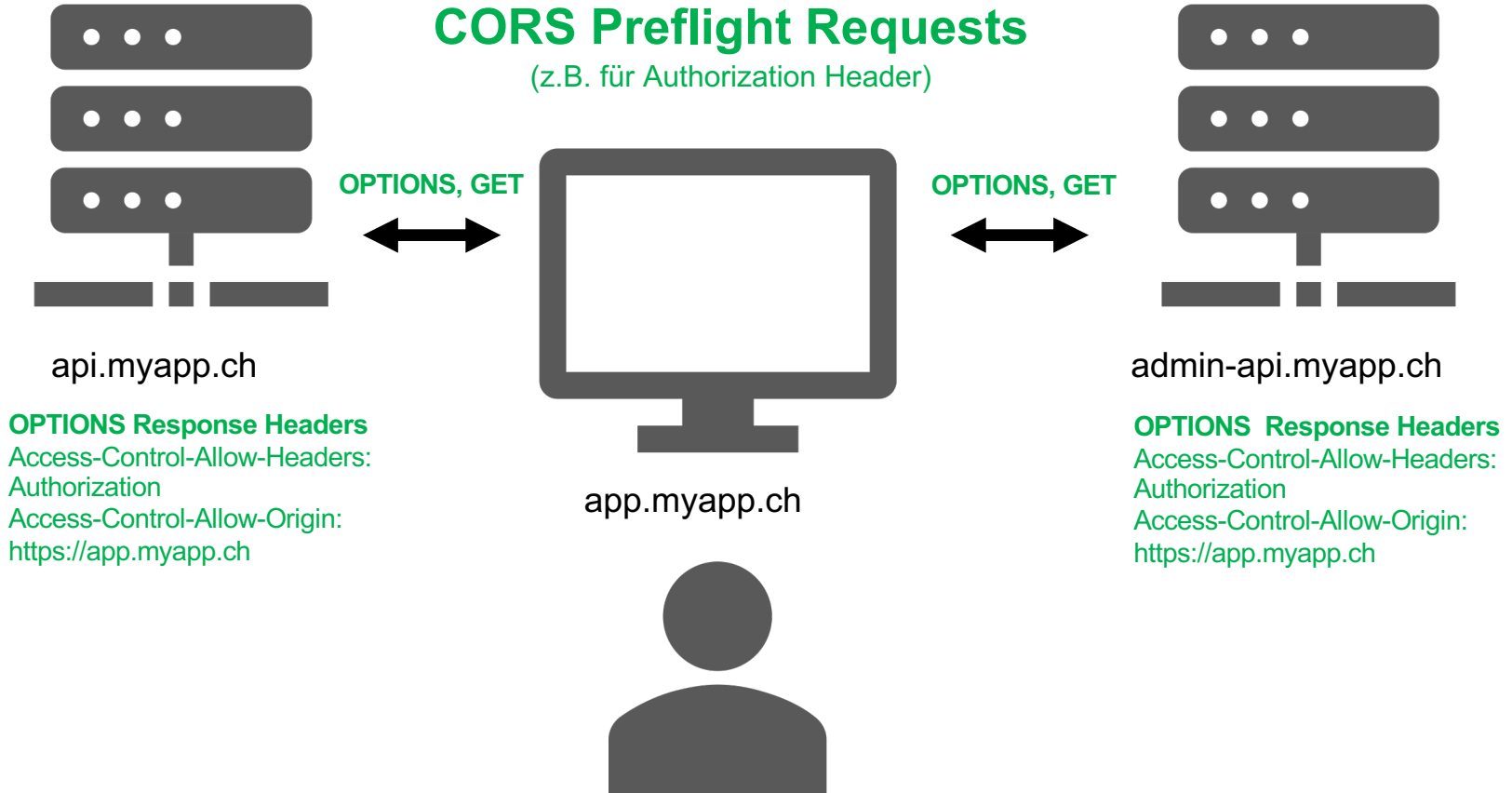
(z.B. Authorization Header)

Simple CORS



CORS Preflight Requests

(z.B. für Authorization Header)



Demo

Input

Authentication @WebApps



Authentifizierung

Wer bist Du?

Autorisierung

Was darfst Du?

Wie vorgehen?

- **Selbst implementierte Auth Lösung**
- **Bekannte Lösung einsetzen**

Selbst implementierte Auth Lösung

Security through obscurity



Selbst implementierte Auth Lösung

Security through obscurity



Security through complexity



Selbst implementierte Auth Lösung

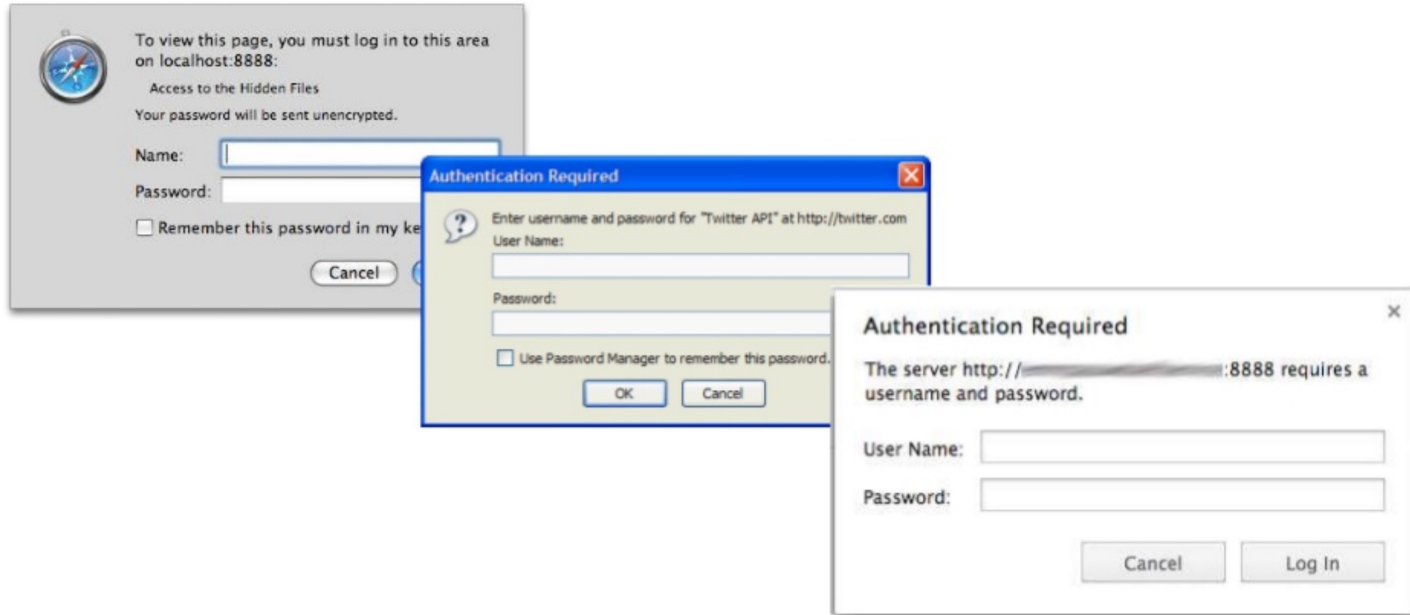
- Regel #1



Sicherheit für «Services»

- **Optionen:**
 - Basic / Digest
 - HMAC ("The way Amazon does it...")
 - JSON Web Token (JWT)
 - OAuth 2.0
- **Und weitere:**
 - OAuth 1.0a
 - WS Security
 - SAML

Basic Authentication



Basic Authentication

- **Idee:**
 - Usernamen und Passwort werden (bei jedem) HTTP Request mitgesendet.
- **Einsatzgebiete:**
 - Einfache Authentication Szenarios mit HTTPS (z.B. Absicherung einer Website mit Benutzername/Passwort)
- **Implementation:**
 - Über HTTP Headers

Basic Authentication

1. Client greift auf URI zu
2. WebServer fordert Authentifizierung für einen Bereich (Realm) im HTTP Response Header an.

```
HTTP/1.1 401 Unauthorized  
WWW-Authenticate: Basic realm="RealmName"
```

3. Client sendet Username/Password im HTTP Request Header im Format "Username:Passwort" Base64 kodiert.

```
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

Übung 1: Basic Authentication

- Erstelle eine Node.js App mit integrierter Basic Authentication.
 - Basic Auth geschützte Ressource «/topsecret» (GET)
 - Login via Browser Popup
 - Verwende Node mit express
 - Plugin für Basic Authentication: express-basic-auth
<https://www.npmjs.com/package/express-basic-auth>
- Repo: <https://github.com/web-programming-lab/node-authentication-seed>
- Branch: basic-authentication

Basic Authentication

*The Basic authentication scheme is **not a secure method of user authentication**, nor does it in any way protect the entity, which is transmitted in cleartext across the physical network used as the carrier. HTTP does not prevent additional authentication schemes and encryption mechanisms from being employed to increase security or the addition of enhancements (such as schemes to use one-time passwords) to Basic authentication. **The most serious flaw in Basic authentication is that it results in the essentially cleartext transmission of the user's password over the physical network.***

(RFC 2617, Internet Engineering Task Force, 1999)

Basic Authentication

- **Vorteile:**
 - Einfache Möglichkeit zur Authentisierung von Websites & Web Applications.
- **Nachteile:**
 - Passwort wird als Plaintext gesendet
 - Passwort wird bei jedem Request gesendet
 - Passwort wird im Browser gecached

Digest Authentication

- **Idee:**
 - Passwort wird nicht gesendet, sondern nur der “Digest” (MD5 Checksum), der aus einer Reihe von Werten berechnet wird.
 - Server verwendet zusätzlich einen “nonce” Wert und einen “opaque” Wert.
- **Einsatzgebiete:**
 - Einfache Authentication Szenarios (z.B. Absicherung einer Website mit Benutzernamen/Passwort)

Digest Authentication

1. Client greift auf URI zu
2. Server fordert Authentifizierung für einen Bereich (Realm) und sendet nonce sowie opaque Werte an den Client.
3. Client berechnet Digest Werte und fügt sie dem Request hinzu.
4. Server prüft Digest, falls erforderlich wird Zugriff auf URI gewährt und der Server sendet (optional) Informationen zur Authentifizierung (z.B. nächstes nonce).

Digest Authentication: HTTP Response Header

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Digest
realm="testrealm@host.com",
qop="auth,auth-int",
nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

- Nonce: Eindeutiger, einmal erzeugter String
- Opaque: Vom Server definierter String, der wieder zurückgeschickt wird.
- Qop: quality of protection: "auth" oder "auth-int"

Digest Authentication: HTTP Request Header

```
Authorization: Digest username="Mufasa",  
realm="testrealm@host.com",  
nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",  
uri="/dir/index.html",  
qop=auth,  
nc=00000001,  
cnonce="0a4f113b",  
response="6629fae49393a05397450978507c4ef1",  
opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

Digest Authentication: HTTP Request Header

- response: Berechneter Digest Wert
- username: Benutzername
- uri: URI (wiederholt wegen Proxies)
- qop: quality of protection
- cnonce: nonce vom Client (wird für Digest verwendet)
- nc: Anzahl Requests, die der Client mit diesem Nonce geschickt hat.

Berechnung des Digest

1. MD5 Hash von "username:realm:passwort" wird berechnet als "HA1"

```
HA1 = MD5( "Mufasa:testrealm@host.com:Circle Of Life" )
```

2. MD5 Hash von "method:uri" wird berechnet als "HA2"

```
HA2 = MD5( "GET:/dir/index.html" )
```

3. MD5 Hash von "HA1:nonce:nc:cnonce:qop:HA2" wird berechnet und als digest response an den Server gesendet.

```
Response =
```

```
MD5("HA1:dcd98b7102dd2f0e8b11d0f600bfb0c093:00000001:0a4f113b:auth:HA2")
```

```
= 6629fae49393a05397450978507c4ef1
```

Digest Authentication

*Digest Authentication **does not provide a strong authentication mechanism, when compared to public key based mechanisms, for example.** However, it is significantly stronger than (e.g.) CRAM-MD5, which has been proposed for use with LDAP [10], POP and IMAP (see RFC 2195 [9]). It is intended to replace the much weaker and even more dangerous Basic mechanism. Digest Authentication **offers no confidentiality protection beyond protecting the actual password.** All of the rest of the request and response are available to an eavesdropper.*

(RFC 2617)

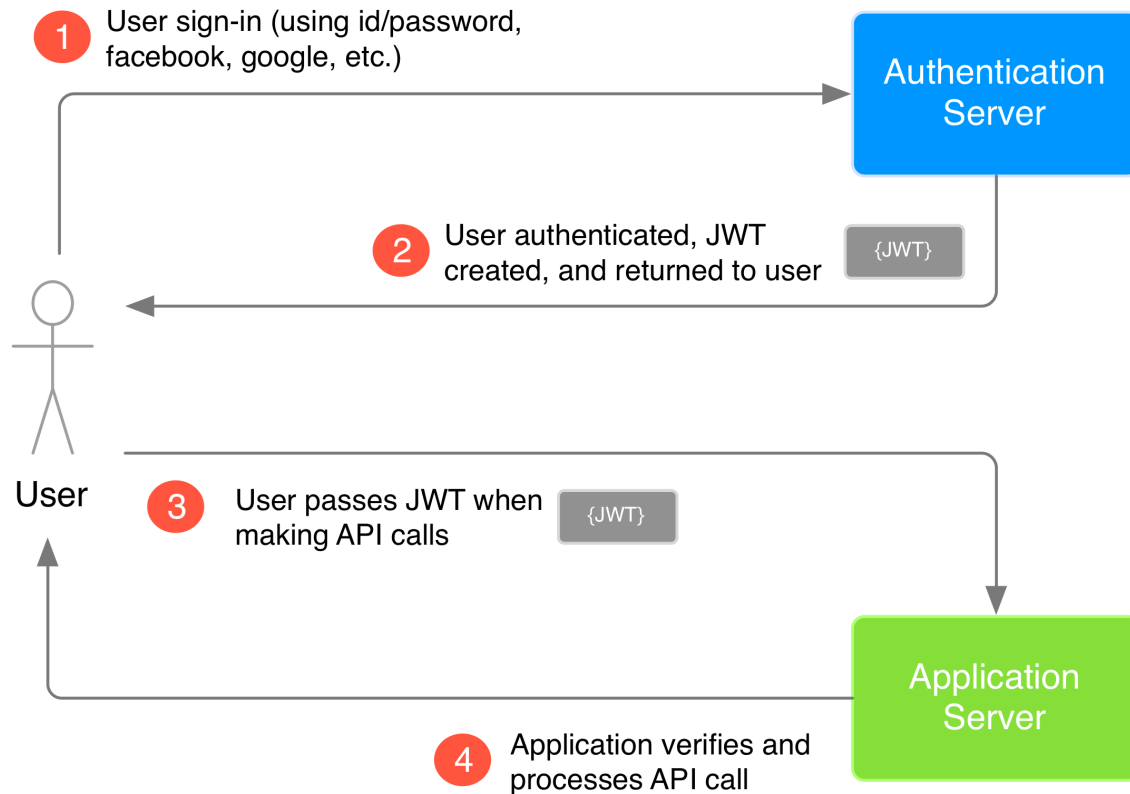
Digest Authentication

- **Vorteile:**
 - Passwort wird nicht als Plaintext übertragen
 - Kein HTTPS notwendig
- **Nachteile:**
 - Kein zusätzlicher Schutz neben dem Passwort (z.B. Request Body)

JSON Web Token (JWT)

- **Idee:**
 - Sichere Übertragung von JSON Objekten
 - Basiert auf Standard (RFC7519)
 - Header, Payload, Signature
- **Einsatzgebiete:**
 - Authentication für (verteilte) Web Anwendungen
 - Single Sign On

JWT Authentication Flow



JWT Header

- JSON Format
- Gibt Algorithmus aus:
 - HS256 = HMAC SHA-256
 - RS256 = RSASSA-PKCSI-v1_5 SHA-256
 - ES256 = ECDSA P-256 curve SHA-256

```
{"typ":"JWT", "alg":"HS256"}
```

JWT Payload

- Benutzerdefinierte Attribute (Claims)
- 3 Typen von Attributen:
 - Standard
 - Public
 - Private

```
{  
  "iss": "www.hslu.ch",  
  "sub": "tcschuer",  
  "name": "Andreas Schürmann",  
  "admin": true  
}
```

JWT Standard Claims

- iss: Issuer
- sub: Subject
- aud: Audience
- exp: Expiration Time
- nbf: Not Before
- iat: Issued at (Time)
- jti: JWT Id

JWT Signature

- Header und Payload sind base64 encoded
- Signature über Header und Payload

```
Signature =  
HMACSHA256(  
    base64UrlEncode(header) + "." + base64UrlEncode(payload)  
    , secret)
```

JWT Token

- String mit Header (base64), Payload (base64) und Signature durch “.” getrennt.

```
Header    = eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
Payload   = eyJpc3MiOiJ3d3cuaHNsdS5jaCI6InN1YiI6Inpha29sbGVyIiwibmFt
           ZSI6IlRob21hcyBLb2xsZXIiLCJhZG1pbI6dHJ1ZX0
Signature = gP07XZYZ7Au83MHRuIMtq41L0l8fWYBG4-ylb4f2c4

Token = Header.Payload.Signature
```


JWT Beispiel

- <https://jwt.io>

ALGORITHM

HS256

Encoded

PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpFuZHI6IjE5MTUxMjM0NTY3ODkwIiwiaWF0IjE1MTUxMjM0NTY3ODkwLCJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.MuXEvRPOM85bHVWBygHbTUNJxUVAhShpZr3CMTcn0oM
```

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{  "alg": "HS256",  "typ": "JWT"}
```

PAYLOAD: DATA

```
{  "sub": "1234567890",  "name": "Andreas Schürmann",  "iat": 1516239022}
```

VERIFY SIGNATURE

```
HMACSHA256(  base64UrlEncode(header) + "." +  base64UrlEncode(payload),  your-256-bit-secret)  ☐ secret base64 encoded
```

JWT

- **Vorteile:**
 - Kompakt & self-contained
 - Passwort wird nur einmalig übertragen
 - Speichern von zusätzlichen Informationen im Token Payload
 - Dank JWT Signatur kann der Token-Issuer verifiziert werden
- **Nachteile:**
 - Je mehr Daten im Payload, je grösser wird das Token
 - Erfordert HTTPS beim Login mit Benutzernamen/Passwort

JWT Authentication Servers

- Keycloak
- Gluu
- Auth0
- ...

Resourcen zu JWT

- <https://jwt.io>
- <https://tools.ietf.org/html/rfc7519>
- <https://www.npmjs.com/package/express-jwt>

Übung 2: JWT Authentication

- Erstelle eine Node.js App mit folgenden Endpoints:

/login User Login mit Username und Passwort

/ Nur mit gültigem JWT aufrufbar

- Die Endpoints müssen via CURL ansprechbar sein

(siehe <https://gist.github.com/anschuermann/16c62d7f65c28085b9486e236aff4df0>)

- Tipp: jsonwebtoken Dependency einsetzen

(siehe <https://github.com/auth0/node-jwebtoken>)

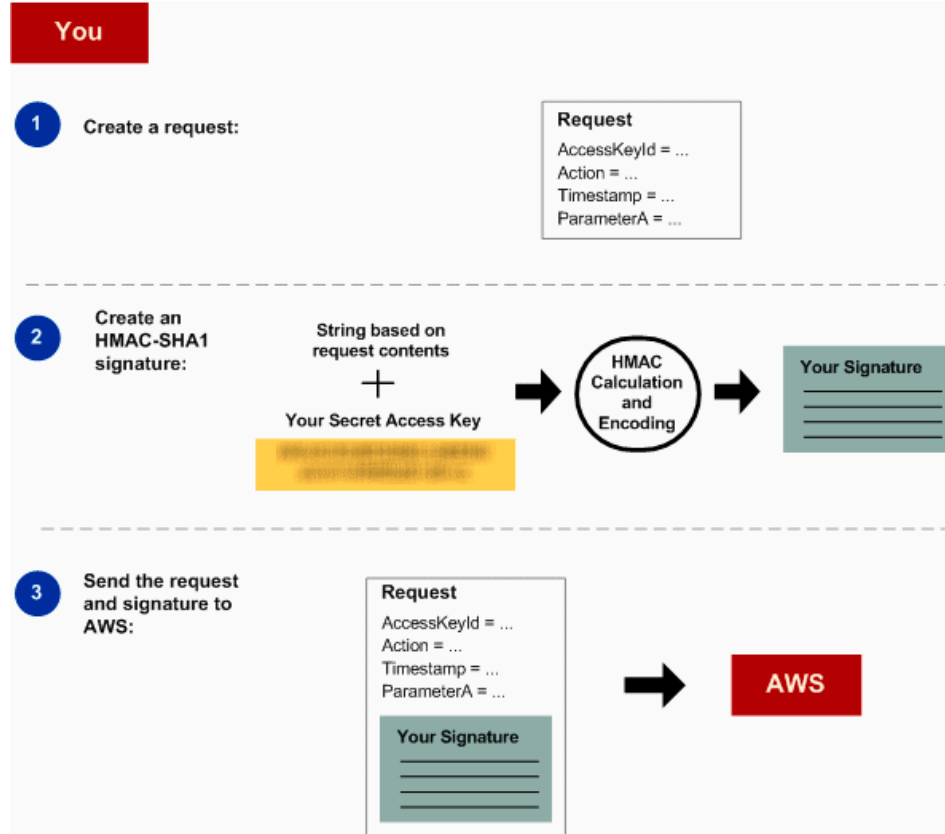
- Repo: <https://github.com/web-programming-lab/node-authentication-seed>

- Branch: jwt-authentication

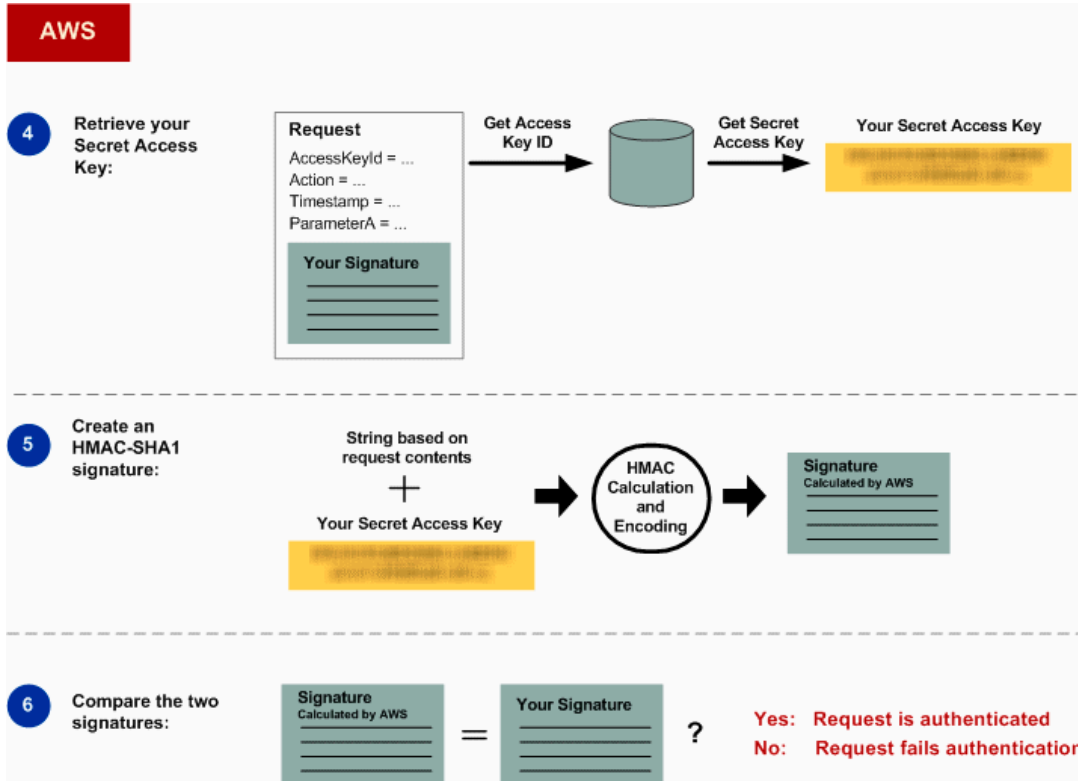
HMAC Authentication (Amazon)

- **Idee:**
 - Verwendung von signiertem Digest Wert (HMAC) zur Authentisierung
 - HMAC = Hash-based message authentication code
 - Digest berechnet aus “SecretAccessKey” und String
 - String enthält HTTP-Verb, **Content**, Content-Type, Date, Headers, Resource Info
 - Verschiedene Versionen: S3 API, Signature Version 4, ...

HMAC Authentication (Amazon)



HMAC Authentication (Amazon)



HMAC

- **Vorteile:**
 - Der gesamte Request wird signiert inkl. Body
 - HTTPS nicht notwendig
- **Nachteile:**
 - Kein Standard resp. konsistente Implementation (ausser Amazon)
 - "Fragil", wenn z.B. Headers noch hinzugefügt werden
 - Aufwändiger in der Integration

OAuth

- Authorization Framework
- Ziel: Freigabe von Daten an Dritte, ohne Username und Passwort freizugeben
- Token basiert
- Versionen
 - OAuth 1.0a
 - **OAuth 2**

OAuth 2 Developer Sicht

- Zugang zu Daten über **Access Token**
- Access Token ist zeitlich begrenzt gültig (Durchschnittlich 1h)
- Verschiedene Typen von Access Tokens
- Access Token kann mit Hilfe eines **Refresh Tokens** erneuert werden
→ Das Refresh Token ist länger gültig (z.B. 30 Tage)

OAuth 2 Rollen

- **Resource Owner / End User**

Besitzer der Ressourcen auf dem Resource Server

- **Resource Server:**

Enthält geschützte Ressourcen

- **Client:**

Anwendung, die auf die Ressourcen zugreifen möchte (z.B. SPA oder Web-Server)

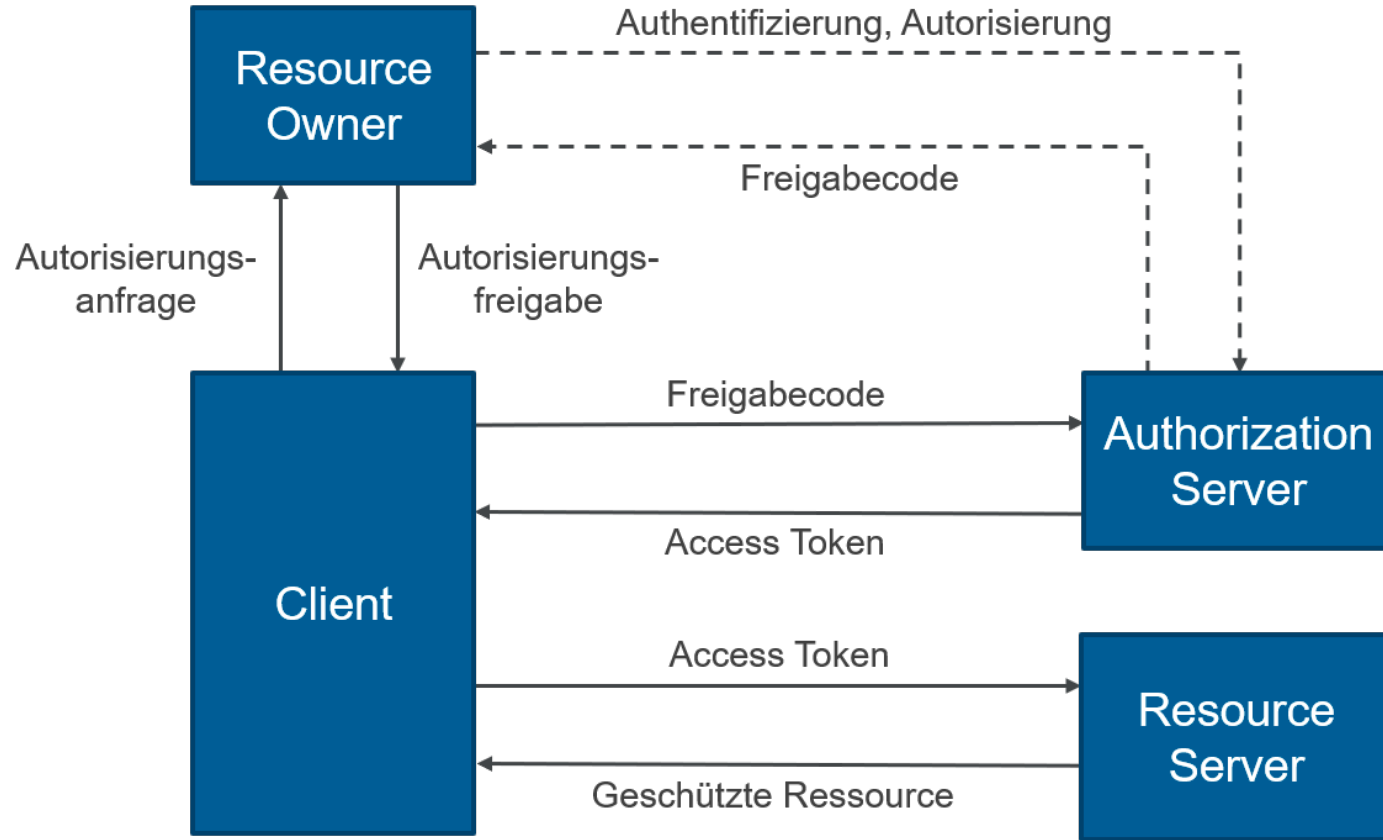
- **User-Agent:**

z.B. Browser

- **Authorization Server**

Authentisiert Resource Owner und stellt Tokens für den Client zur Verfügung

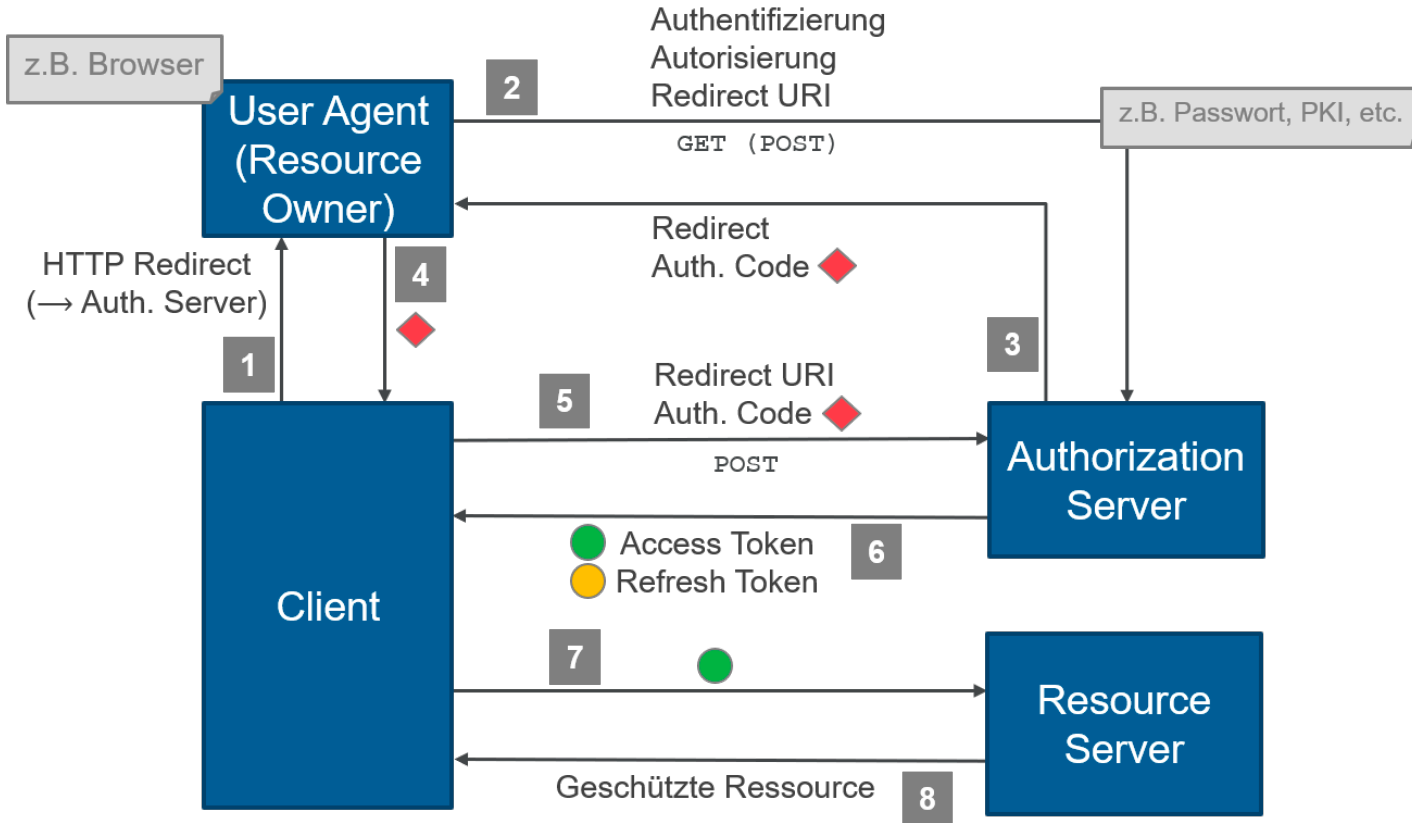
OAuth 2 Basic Flow



OAuth 2 Grant Types

- **Authorization Code**
 - Für Web-Server basierte Anwendungen.
- **Implicit**
 - User-Agent-Based Application – Client Secret und Token nicht sicher (z.B. Browser Apps, Third-party mobile Apps)
- **Resource Owner Password Credentials**
 - Native Application: Anmeldung über User-Login Daten
- **Client Credentials**
 - Für Machine2Machine Kommunikation

OAuth 2 Authorization Code Flow





OAuth 2 Authorization Code Flow User Sicht




Konto verknüpfen



 Über Google anmelden



adhook.io benötigt Zugriff auf Ihr Google-Konto

 an.schuermann@gmail.com

Dadurch erhält **adhook.io** diese Berechtigungen:

- AdWords-Kampagnen verwalten ⓘ
- Produkteinträge und Konten für Google Shopping verwalten ⓘ
- Ihre Brancheneinträge auf Google abrufen, bearbeiten, erstellen und löschen ⓘ

adhook.io vertrauen?

Eventuell teilen Sie vertrauliche Informationen mit dieser Website oder App. In der [Datenschutzerklärung](#) von adhook.io erfahren Sie alles zum Umgang mit Ihren Daten. In Ihrem [Google-Konto](#) können Sie die Zugriffsberechtigungen jederzeit einsehen oder entfernen.

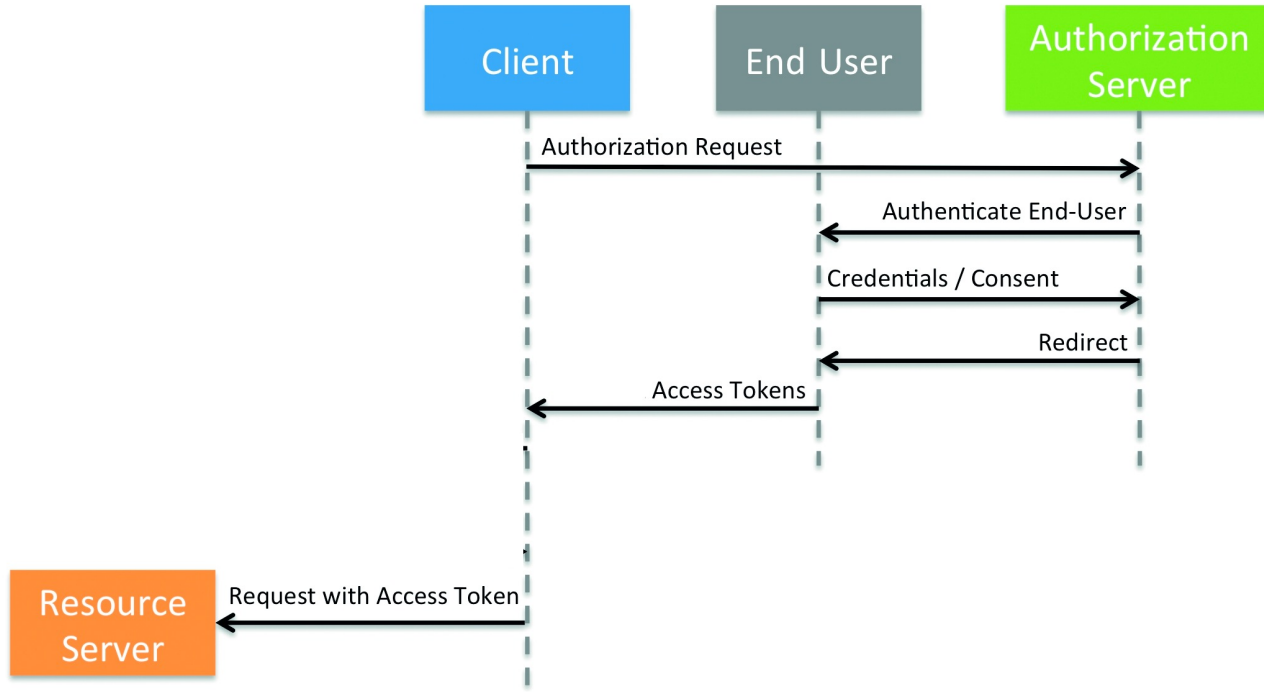
[Weitere Informationen zu den Risiken](#)

[Abbrechen](#) [Zulassen](#)

OAuth 2 Grant Types

- **Authorization Code**
 - Für Web-Server basierte Anwendungen.
- **Implicit**
 - User-Agent-Based Application – Client Secret und Token nicht sicher (z.B. Browser Apps, Third-party mobile Apps)
- **Resource Owner Password Credentials**
 - Native Application: Anmeldung über User-Login Daten
- **Client Credentials**
 - Für Machine2Machine Kommunikation

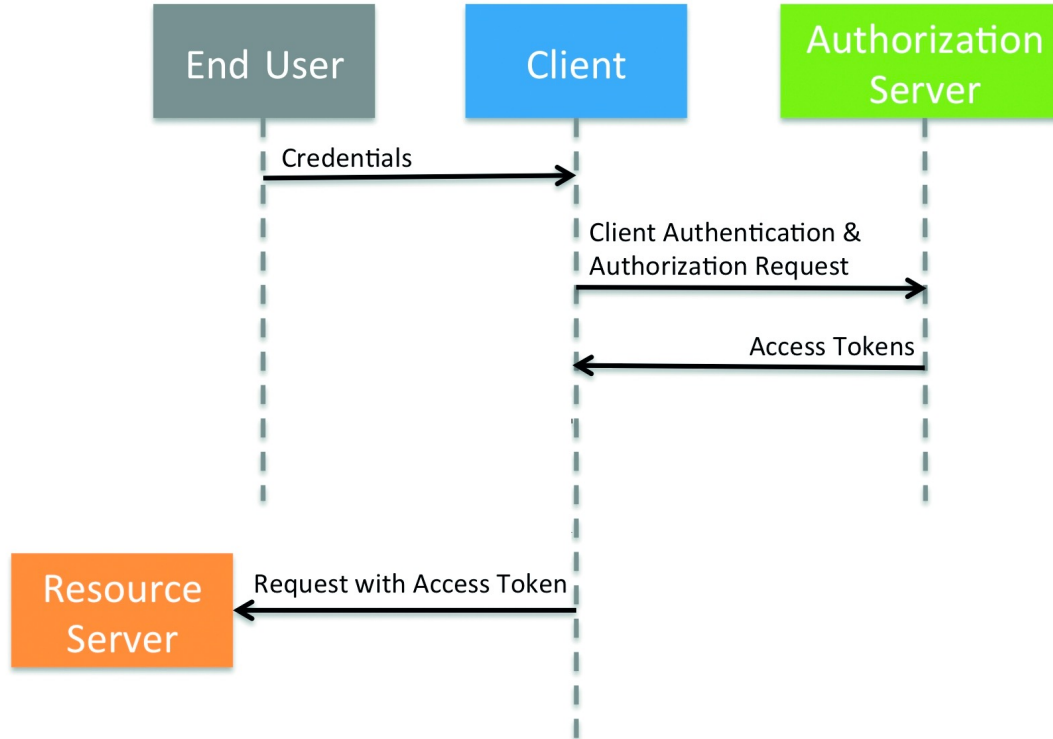
OAuth 2 Implicit Grant



OAuth 2 Grant Types

- **Authorization Code**
 - Für Web-Server basierte Anwendungen.
- **Implicit**
 - User-Agent-Based Application – Client Secret und Token nicht sicher (z.B. Browser Apps, Third-party mobile Apps)
- **Resource Owner Password Credentials**
 - Native Application: Anmeldung über User-Login Daten
- **Client Credentials**
 - Für Machine2Machine Kommunikation

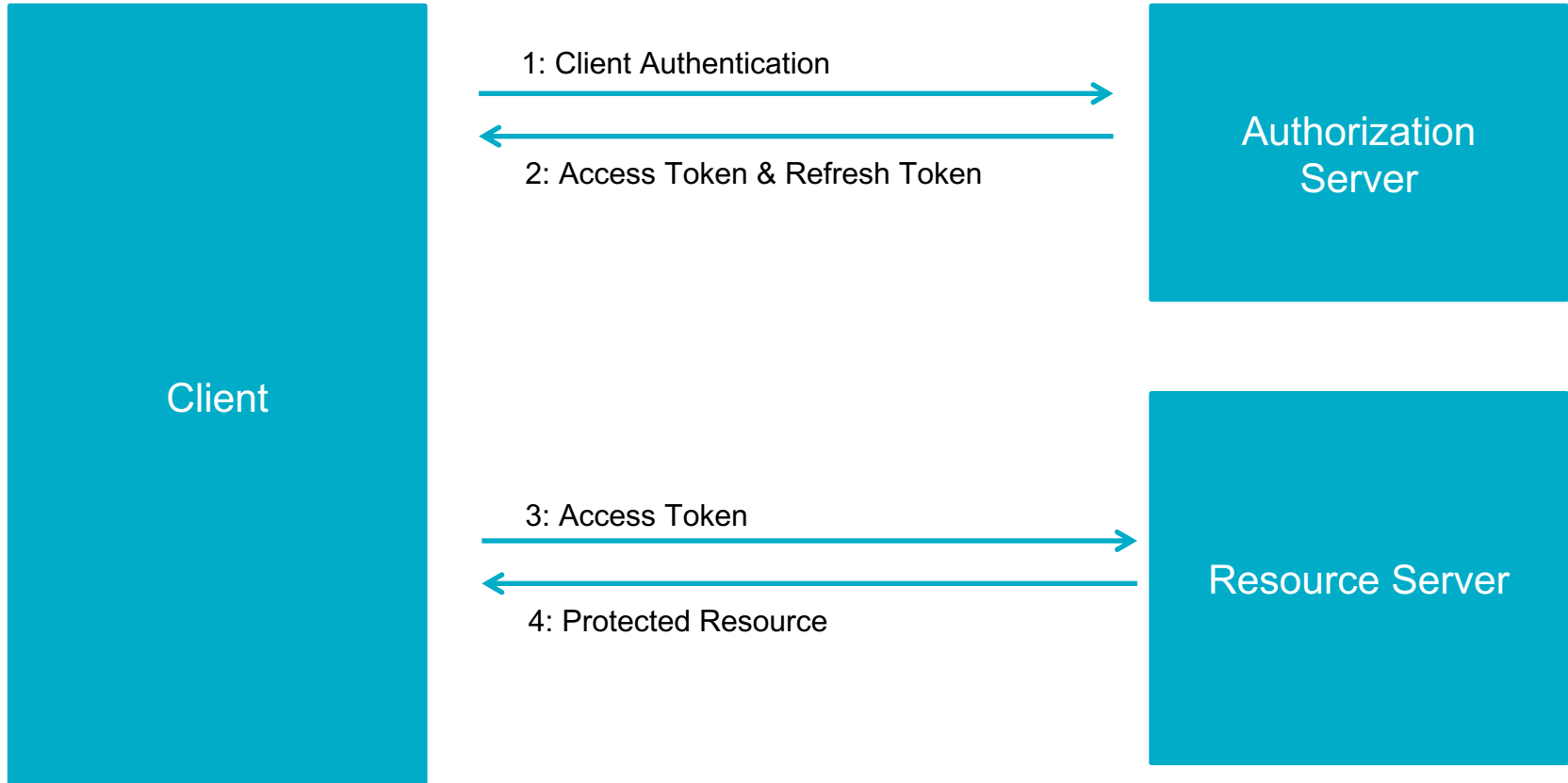
OAuth 2 Password Credential Grant



OAuth 2 Grant Types

- **Authorization Code**
 - Für Web-Server basierte Anwendungen.
- **Implicit**
 - User-Agent-Based Application – Client Secret und Token nicht sicher (z.B. Browser Apps, Third-party mobile Apps)
- **Resource Owner Password Credentials**
 - Native Application: Anmeldung über User-Login Daten
- **Client Credentials**
 - Für Machine2Machine Kommunikation

OAuth 2 Client Credentials Grant



OAuth 2 Authorization Servers

- Keycloak
- Gluu
- Auth0
- ...

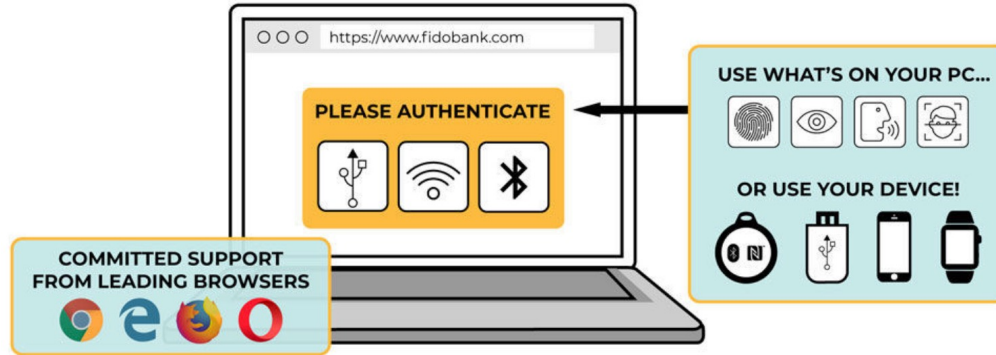
OAuth 2

- **Vorteile:**
 - Viele Authentication Möglichkeiten in einem Framework
 - Entkopplung von Authentication und Web Application
 - Stark verbreitet
- **Nachteile:**
 - Komplexität im Einsatz aufgrund der versch. Möglichkeiten
 - Unklarheiten in der RFC Spec

FIDO2: Fast Identity Online

- **Ziel:** Anmeldung an Webdiensten ohne Passwort («simpler stronger authentication»)
- Einsatz von USB-Token oder biometrischem Verfahren
- Fido alliance: Google, Microsoft, Mozilla, PayPal, Samsung u.a.

FIDO2



FIDO AUTHENTICATION: THE NEW GOLD STANDARD



Protects against phishing,
man-in-the-middle and
attacks using stolen
credentials

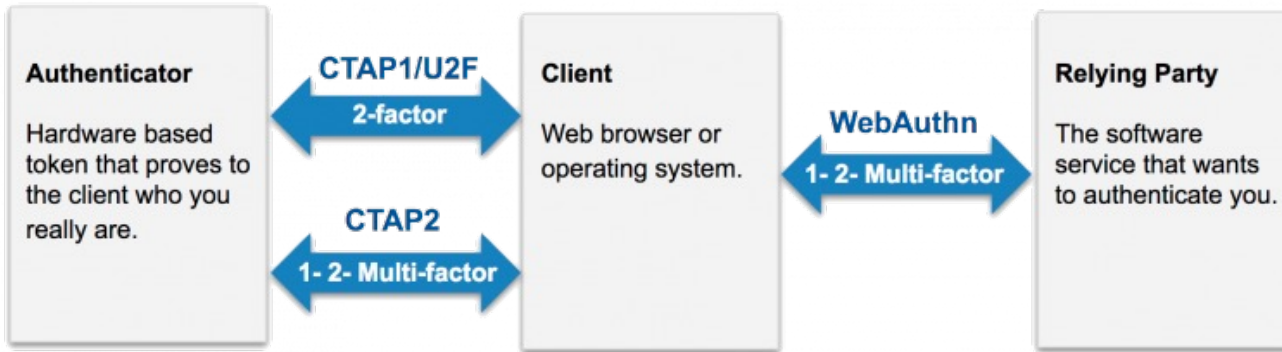


Log in with a single
gesture – HASSLE FREE!



Already supported in
market by top online
services

FIDO2 Bestandteile: CTAP & WebAuthn



FIDO2 Demo

- <https://webauthn.io>

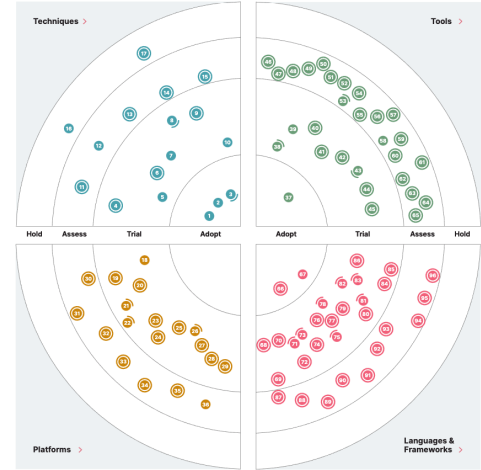
Abschluss

Organisatorisches



Projekt

- Einzelarbeit
- Projekt “Technologie-Radar” oder eigene Projekt-Idee
- Alle Informationen zum Web Programming Lab Projekt
- Fokus Software Engineering mit Web Technologien
 - > Idealerweise mit Technologien ohne Praxiserfahrung
 - > inkl. (lokales) Deployment & Testing



MEP

- **Projekt-Ergebnisse (ohne Präsentation):**

Abgabe bis Mittwoch, 8. März 2023 18:00

- **Projekt-Präsentation:**

Samstag, 11. März 2023 08:30 – ca. 11:00

(5 Minuten Präsentation, 1 – 2 Minuten Fragen)

Feedback

- Unsere fünfte Durchführung – vielen Dank für Eure Teilnahme!
- Feedback via [menti.com](https://www.menti.com)