

M183 Applikationssicherheit Implementieren # 6

By Jürg Nietlispach

Recap # 5?

Recap # 5

1. 2-Factor Authentication: TOTP (vs. OTP)
2. Kurzintro Single Sign On

2-Factor-Authentication – the perfect solution?

Consider: one user has many tools to use ...

What are the issues? And Why?

Usability Issues with 2-Factor-Authentication

1. Separate Credentials for every Portal? -> In practice NO!
2. Time consuming to do 2-Factor-Authentication for every tool the user uses
3. N Tools -> N Logins! (-> Minimize Attack Surface Area!)
4. N Tools -> N Logins -> N-times Password- and Profilechanges!
5. ...

Solution?

Single Sign On!

(in combination with 2-Factor-Auth)

“Single sign-on (SSO) is a property of [access control](#) of multiple related, yet independent, [software](#) systems. With this property, a user [logs in](#) with a single ID and password to gain access to a connected system or systems without using different usernames or passwords, or in some configurations seamlessly sign on at each system.”

Single Sign On - Types

- **Portal Solutions (SAML, OAuth2, OpenID Connect, traditional HTTP-Cookie)**
- Local Solutions (z.B. Passwordmanager)
- Ticketing System (Kerberos)
- ...
- (Public Key Infrastructure: Digitales Zertifikat)

Single Sign On – Portal Solution 1

NON-SSO SCENARIO



Motivation: Already Logged-In User on domain1.com automatically login on domain2.com

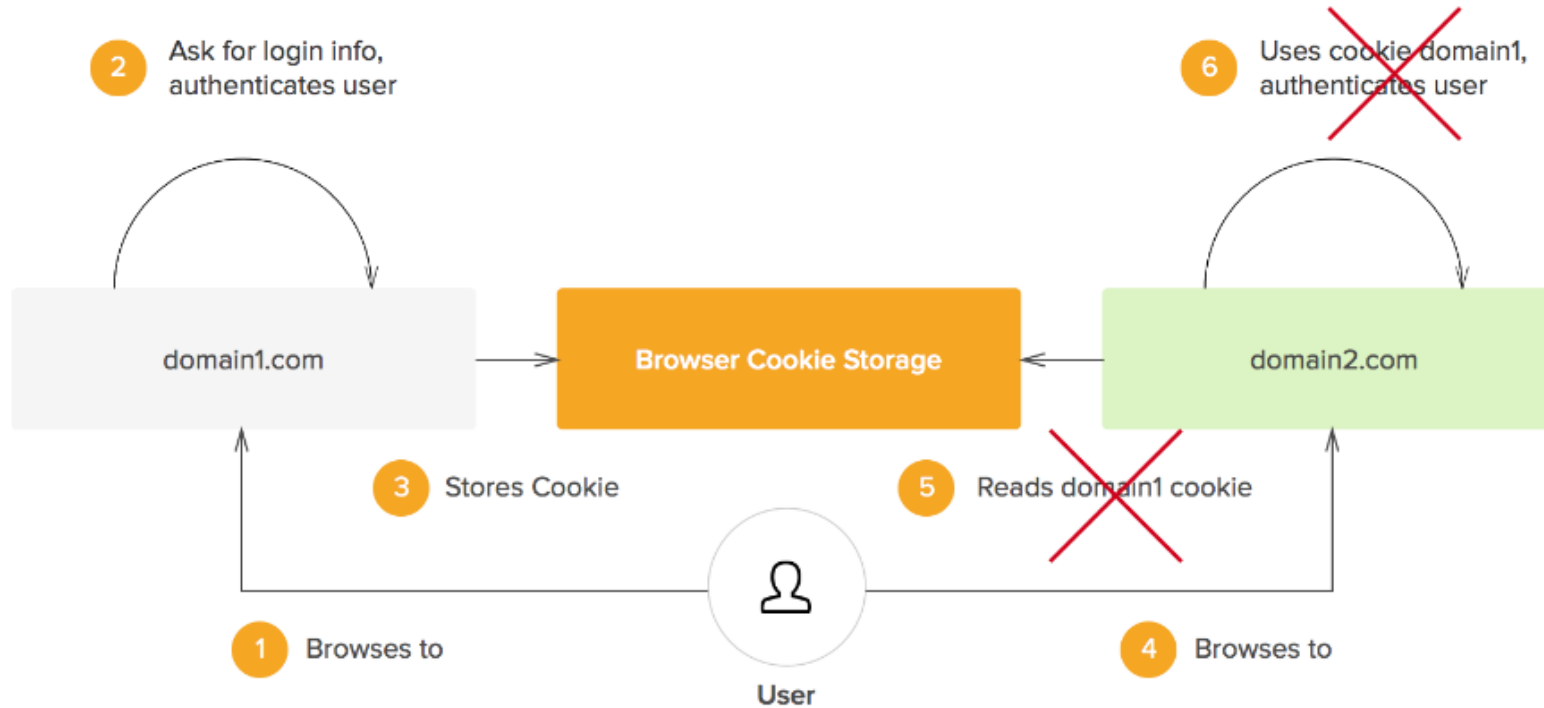
Solution: **share session information** across different domains

Single Sign On – Portal Solution 1

Problem?

Single Sign On – Portal Solution 2

SAME-ORIGIN-POLICY FORBIDS THIS



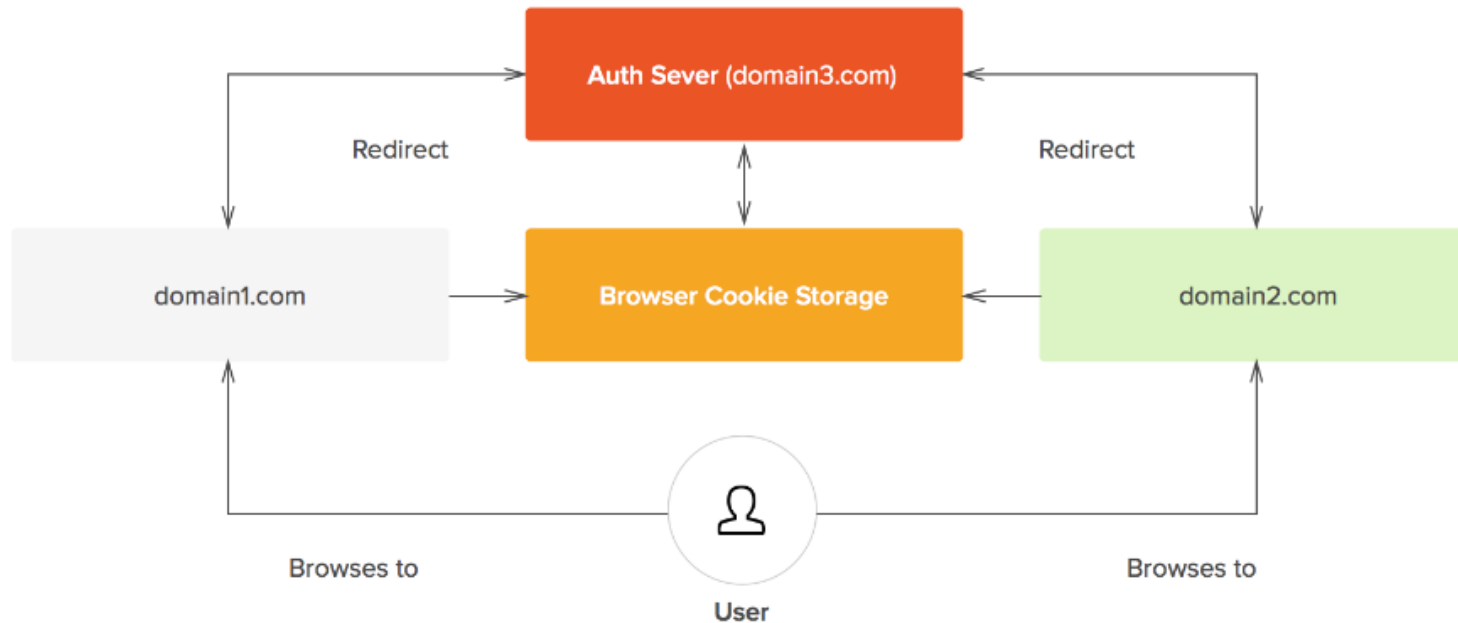
Problem: *same origin policy*.
This policy dictates that cookies (and other locally stored data, JWT) can **only be accessed by its creator!** (i.e. domain1.com, subdomain.domain1.com etc.)

Single Sign On – Portal Solution 2

Solution?

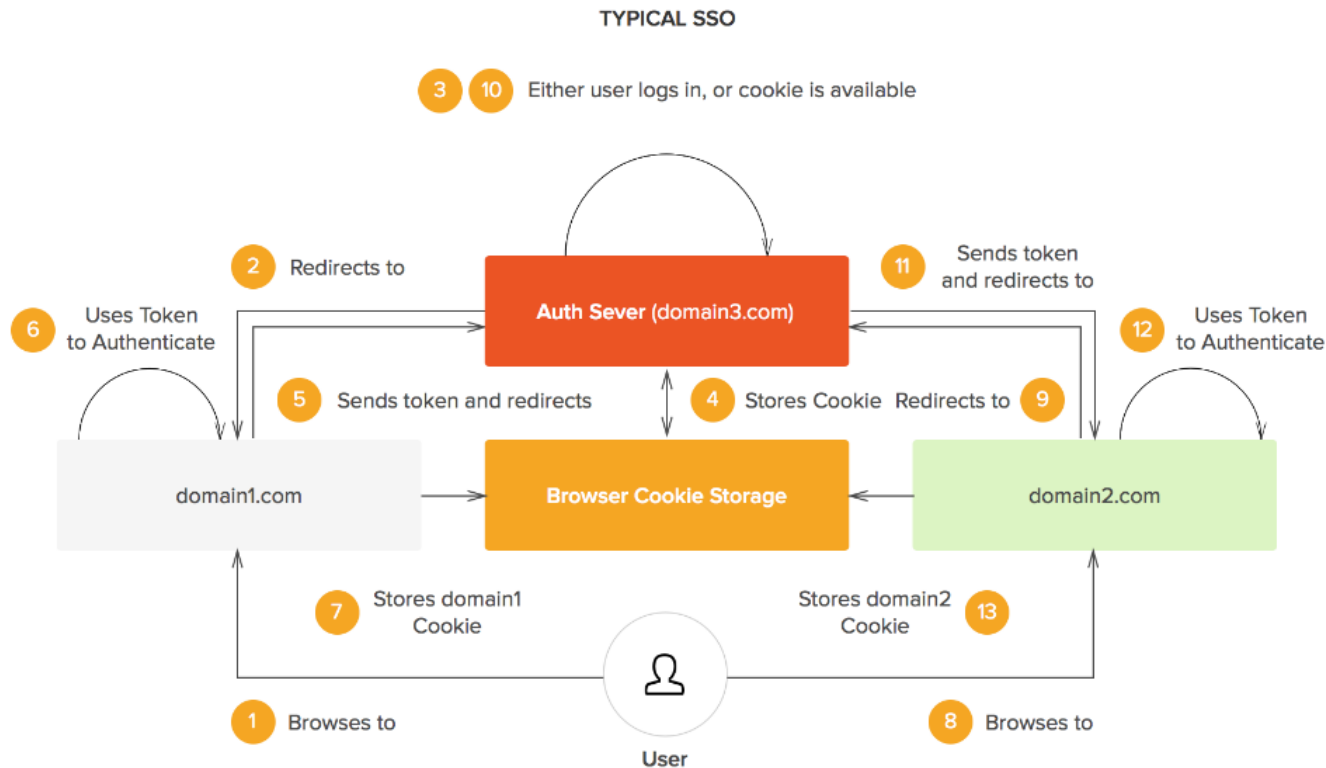
Single Sign On – Portal Solution 3

USING A CENTRAL AUTHENTICATION DOMAIN



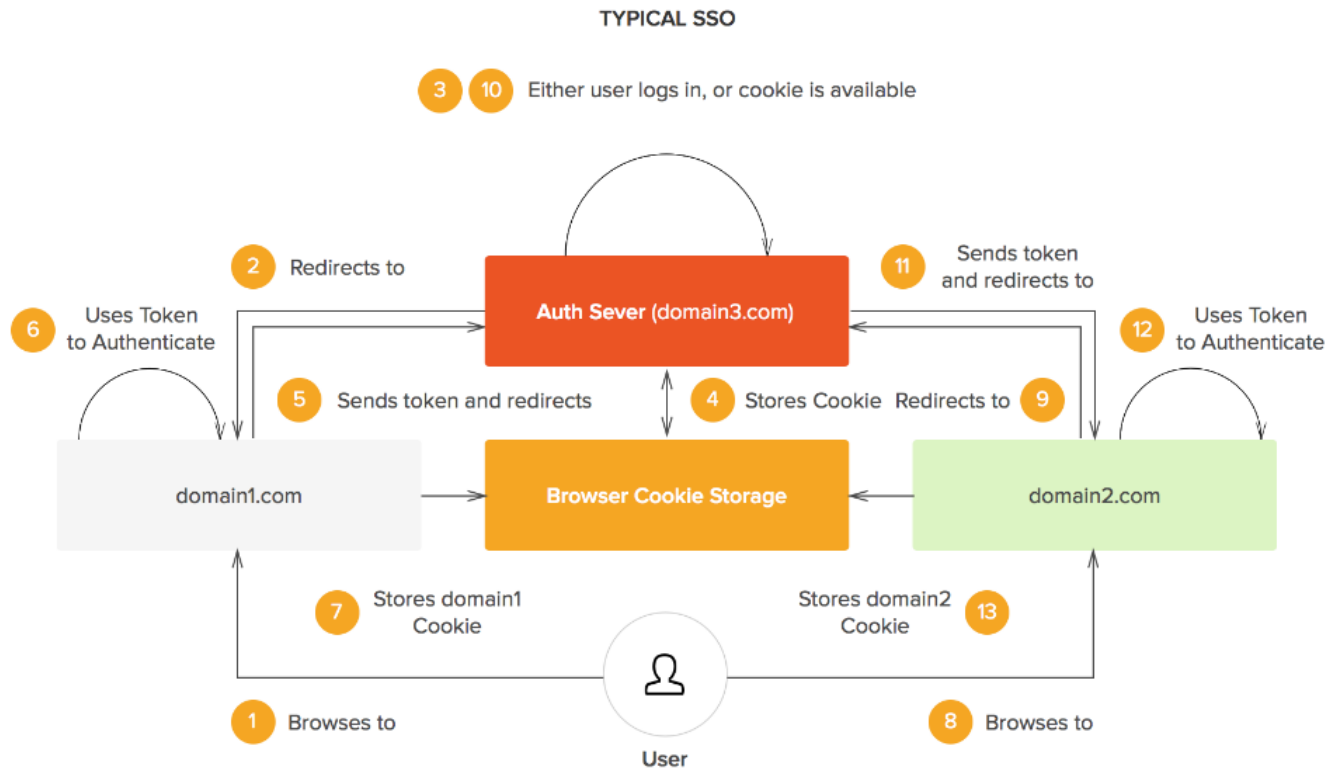
Solution 1: Central Authentication Provider!

Single Sign On – OpenId Connect



1. User navigates to Login on domain1.com
2. domain1.com redirects to Auth Server (Login with Google-Button)
3. User logs in at Auth Server
 1. User gets Cookie for the Auth Server (future logins)
 2. User gets a signed Ticket for domain1.com
4. Client is redirected back to domain1.com with the signed Ticket
5. Information of the signed Ticket are used to check back at the Auth Server (from domain1.com) whether the user is logged in (ID Token)
6. domain1.com can send a session cookie to the user
7. domain1.com can request additional user information (Claims) using the ID Token

Single Sign On – OpenId Connect 2



8. User navigates to Login on domain2.com
9. domain2.com *redirects* to Auth Server (Login with Google Button)
10. User is **already logged in** in at Auth Server
8. User gets a signed Ticket for domain2.com
11. Client is *redirected* back to domain2.com with the signed Ticket
12. Information of the signed ticket are used to check back at the Auth Server (from domain2.com) whether the user is logged in (ID Token)
13. domain2.com can send a session cookie to the user
14. domain2.com can request additional user information (Claims) using the ID Token

Single Sign On – OpenId Connect

1. Create Link (i.e. login with google button) on domain1.com (containing redirect_uri, client_id, etc)

```
HTTP/1.1 302 Found
Location: https://openid.c2id.com/login?
    response_type=code
    &scope=openid
    &client_id=s6BhdRkqt3
    &state=af0ifjsldkj
    &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
```

1. This link redirects the user to the Auth Server User logs in at Auth Server
 1. The user gets Cookie for the Auth Server (future logins)
 2. Client is redirected back to domain1.com with a ticket

```
HTTP/1.1 302 Found
Location: https://client.example.org/cb?
    code=Splxl0BeZQQYbYS6WxSbIA
    &state=af0ifjsldkj
```

Single Sign On – OpenId Connect

1. Informations of the identity token are used to check back at the Auth Server whether the user is logged in

```
POST /token HTTP/1.1
Host: openid.c2id.com
Content-Type: application/x-www-form-urlencoded
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW

grant_type=authorization_code
&code=5plxl0BeZQQYbYS6WxSbIA
&redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
```

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store
Pragma: no-cache

{
  "id_token": "eyJhbGciOiJSUzI1NiIsImtpZCI6IjFlOWdkazcifQ.ewogImlzc  
yI6ICJodHRwOi8vc2VydmVybWV4YW1wbGUuY29tIiwKICJzdWIiOiAiMjQ4Mjg5  
NzYxMDAxIiwKICJhdWQiOiAic2ZCaGRSa3F0MyIsCiAibm9uY2UiOiAibi0wUzZ  
fV3pBMk1qIiwKICJleHAiOiAxMzExMjg5OTcwLAogImldCI6IDEzMTEyODANz  
AKfQ.ggw8hZ1EuVLuxNuuIJkX_V8a_OMXR0EHR9R6jgdqrOOF4daGU96Sr_P6q  
Jp6IcmD3HP990bi1PRs-cwh3LO-p146waJ8IhehcwL7F09JdijmBqkvPeB2T9CJ  
NqeGpe-gccMg4vfKjkM8FcGvnzZUN4_KSP0aAp1tOJ1zZwgjxqGBYKH1OtX7Tpd  
QyHE5lcMiKPXFIEQILVq0pc_E2DzL7emopwoaoZTF_m0_N0YzFC6g6EJB0EoRoS  
K5hoDalrcvRYLSrQAZZKFlyuVCyixEoV9GFnQC3_osjzw2PAithfubEEBLuVVk4  
XUVrWOlrLL0nx7RkKU8NXNHq-rvKMzqg"
  "access_token": "SlAV32hkKG",
  "token_type": "Bearer",
  "expires_in": 3600,
}
```


Single Sign On – OpenId Connect

1. domain1.com can send a session cookie to the user
2. Additionally, domain1.com can now get further «Claims» about the user at the Auth Server (additional user infos) using the id_token

Single Sign On

Benefits

- One Authentication-Procedure for N Systems
 - Time
 - Security
 - less (weak) passwords per user
 - May reduce phishing attacks

Drawbacks

- Attacker has instantly access to all services as soon as he has the credentials
- Sign off (Time-Out)?
- Availability of a SSO-Service?

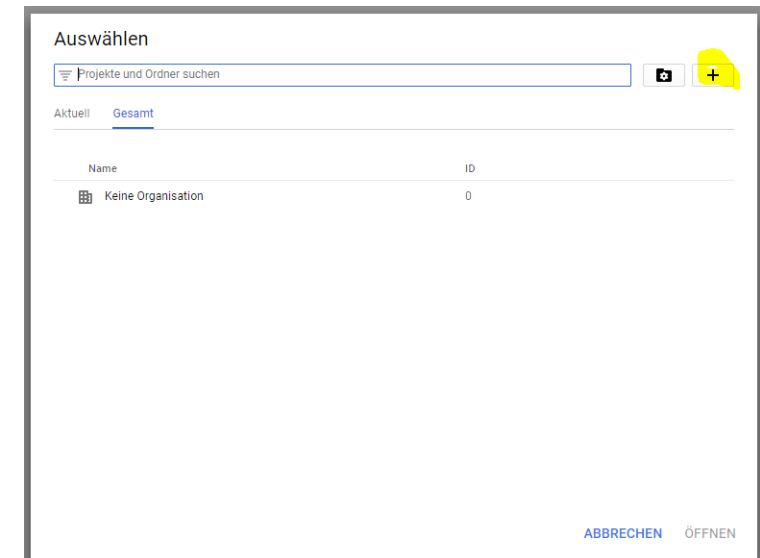
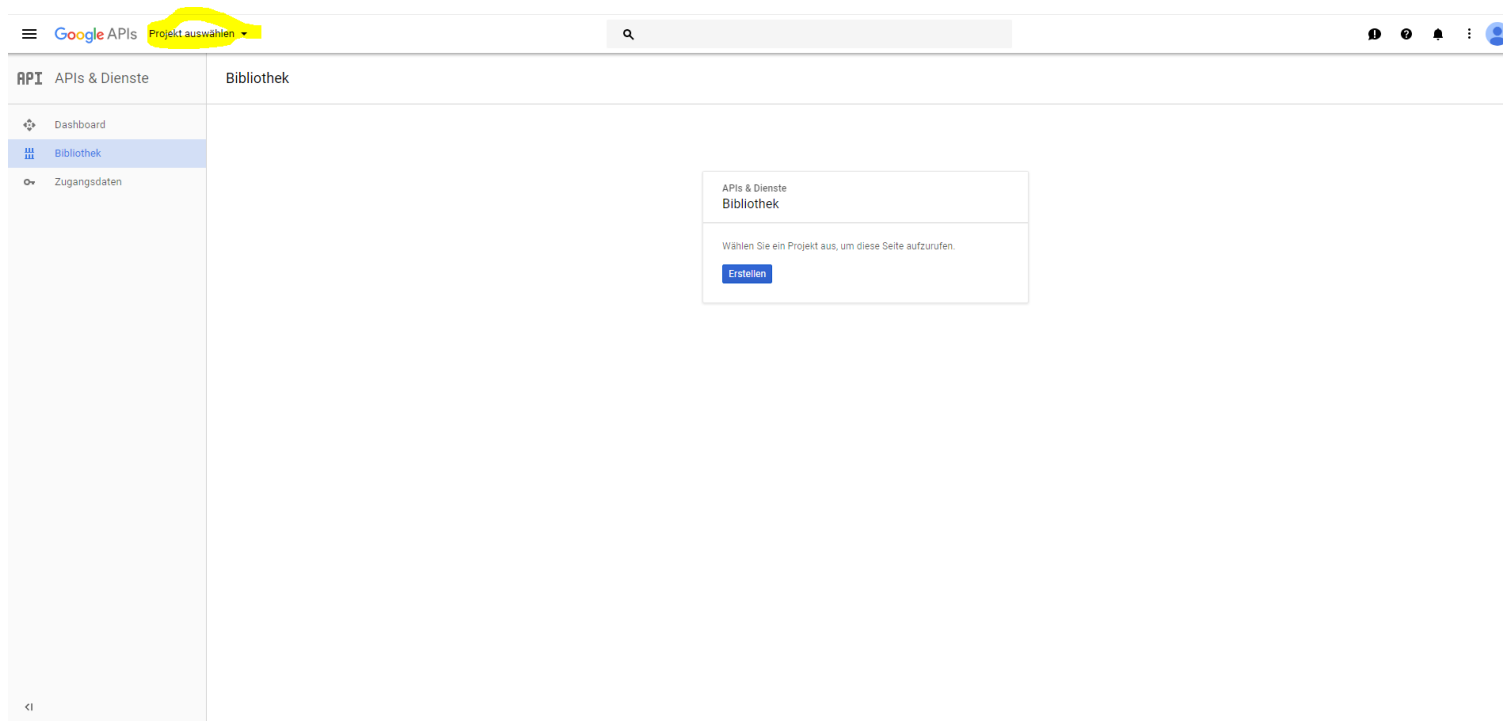
Übung – Single Sign On mit Google API

1. Setup Auth Server (Google API Account Infos) for Client / Button Integration (Google-Account wird benötigt)
<https://developers.google.com/identity/sign-in/web/devconsole-project>
2. Setup Client / Button
<https://developers.google.com/identity/sign-in/web/sign-in>
3. Get Profile Infos
<https://developers.google.com/identity/sign-in/web/people>
4. Authenticate with Backend Server (in order to get a session)
<https://developers.google.com/identity/sign-in/web/backend-auth>
5. Sign Out Button
<https://developers.google.com/identity/sign-in/web/sign-in>

Setup Auth-Server 1

Google API Console Project:

- <https://developers.google.com/identity/sign-in/web/devconsole-project>
- -> Follow the instructions (or see Screenshots):



Setup Auth-Server 2



Neues Projekt

Ihr Kontingent umfasst noch 12 Projekte. [Weitere Informationen.](#)

Projektname

M183

Die Projekt-ID lautet: m183-181120 [Bearbeiten](#)

Ich möchte Updates zu Funktionsankündigungen, Anwendungsvorschlägen, Feedbackumfragen und besonderen Angeboten erhalten.

☒ Ja ☐ Nein

Ich stimme zu, dass ich die [Dienste und zugehörigen APIs](#) nur unter der Voraussetzung nutzen darf, dass ich die geltenden [Nutzungsbedingungen](#) einhalte.

☒ Ja ☐ Nein

Erstellen

Abbrechen

Zugangsdaten

Anmeldedaten

OAuth-Zustimmungsbildschirm

Domainbestätigung

E-Mail-Adresse

juerg.nietlispach@gmail.com

Produktname, den Nutzer sehen

M183

Homepage-URL (optional)

https:// oder http://

Produktlogo-URL (optional)

http://www.example.com/logo.png



So sehen Endnutzer Ihr Logo.

Max. Größe: 120 x 120 px

URL der Datenschutzerklärung

Optional, bis Sie Ihre App bereitstellen

https:// oder http://

URL der Nutzungsbedingungen (optional)

https:// oder http://

Speichern

Abbrechen



Der Zustimmungsbildschirm wird Nutzern gezeigt, wenn Sie mit Ihrer Client-ID Zugriff auf deren private Daten anfordern. Er erscheint für alle Anwendungen, die in diesem Projekt registriert sind.

Sie müssen eine E-Mail-Adresse und einen Produktnamen bereitstellen, damit OAuth funktioniert.

Setup Auth-Server 3

← Client-ID erstellen

APIs Anmeldedaten

Für den Zugriff auf APIs sind Anmeldedaten erforderlich. Aktivieren Sie die APIs, die Sie verwenden möchten, und erstellen Sie dann die erforderlichen Anmeldedaten. Je nach API sind ein API-Schlüssel, ein Dienstkonto oder eine OAuth 2.0-Client-ID erforderlich. Weitere Informationen finden Sie in der API-Dokumentation.

Anmeldedaten erstellen ▾

API-Schlüssel

Identifiziert Ihr Projekt durch einen einfachen API-Schlüssel, um Kontingent und Zugriff zu prüfen

OAuth-Client-ID

Fordert die Zustimmung des Nutzers an, damit Ihre App auf die Daten des Nutzers zugreifen kann

Dienstkontoschlüssel

Aktiviert mithilfe von Robot-Konten Server-zu-Server-Authentifizierung auf App-Ebene

Auswahlhilfe

Beantworten Sie einige Fragen, um herauszufinden, welche Form der Anmeldung am besten geeignet ist.

Anwendungstyp

- ☒ Webanwendung
- ☐ Android [Mehr erfahren](#)
- ☐ Chrome-App [Mehr erfahren](#)
- ☐ iOS [Mehr erfahren](#)
- ☐ PlayStation 4
- ☐ Sonstige

Name

M183

Einschränkungen

Geben Sie JavaScript-Quellen oder Weiterleitungs-URLs oder beides ein.

Autorisierte JavaScript-Quellen

Zur Verwendung bei Anfragen über einen Browser. Dies ist die Ursprungs-URI der Clientanwendung. Sie darf weder einen Platzhalter (http://*.ihrebeispielurl.de) noch einen Pfad (<http://ihrebeispielurl.de/subdir>) enthalten. Wenn Sie einen nichtstandardmäßigen Port verwenden, müssen Sie ihn in der Ursprungs-URI angeben.

<http://localhost:49670>

Autorisierte Weiterleitungs-URLs

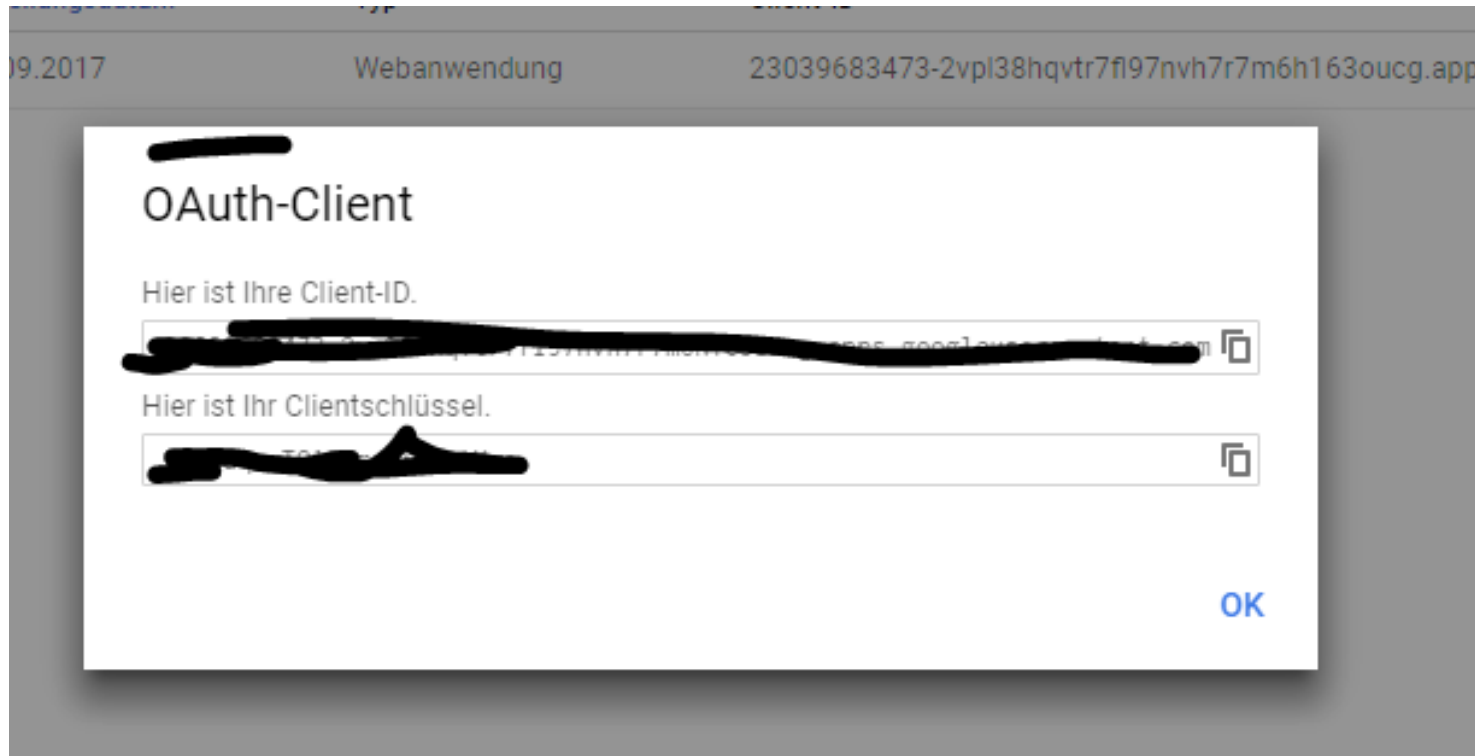
Für die Verwendung mit Anfragen über einen Webserver. Dies ist der Pfad in Ihrer Anwendung, zu dem Nutzer nach der Authentifizierung mit Google weitergeleitet werden. An den Pfad wird der Autorisierungscode für den Zugriff angehängt. Muss ein Protokoll aufweisen. Darf keine URL-Fragmente oder relativen Pfade enthalten. Öffentliche IP-Adressen sind nicht zulässig.

<http://www.example.com/oauth2callback>

Erstellen

Abbrechen

Setup Auth-Server 4



Setup Client / Button

1. Script in Header kopieren:

```
<script src="https://apis.google.com/js/platform.js" async defer></script>
```

2. Meta-Tag in Header einbinden:

```
<meta name="google-signin-client_id" content="YOUR_CLIENT_ID.apps.googleusercontent.com">
```

3. Button einbinden

```
<div class="g-signin2" data-onsuccess="onSignIn"></div>
```


Get Profile Infos

1. Profil-Informationen mittels JS eruieren:

```
function onSignIn(googleUser) {  
    var id_token = googleUser.getAuthResponse().id_token;  
    var profile = googleUser.getBasicProfile();  
    console.log(profile.getId(), profile.getName(), profile.getImageUrl());  
}
```

Pre-Setup Backend Server

1. Extend onSignIn() Javascript – Function:

```
function onSignIn(googleUser){  
  
    var id_token = googleUser.getAuthResponse().id_token;  
  
    var xhr = new XMLHttpRequest();  
    xhr.open('POST', 'http://localhost:PORT/Home/SSOTokenSignin');  
    xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');  
    xhr.onload = function() {  
        console.log('Signed in as: ' + xhr.responseText);  
    };  
    xhr.send('idtoken=' + id_token);  
  
}
```

Setup Backend Server

1. Create a Route for id_token validation (http://localhost:PORT/api/validateidtoken/{id_token})
2. Validate token (integrity) using .NET OAuth Client Library:
 1. <https://developers.google.com/api-client-library/dotnet/apis/oauth2/v2>
3. Check ID-Token at
 1. https://www.googleapis.com/oauth2/v3/tokeninfo?id_token={ID_TOKEN}
4. If token is valid -> return SESSION

Logout Button

1. Place Logout-Button

```
<a href="#" onclick="signOut();">Sign out</a>
```

2. Integrate Javascript Snippet

```
function signOut() {  
  var auth2 = gapi.auth2.getAuthInstance();  
  auth2.signOut().then(function () {  
    console.log('User signed out.');  });  
}
```