Dr. Johannes C. Schneider

Zentrale Authentifizierung mit OAuth2 in einem technologisch heterogenen App-Ökosystem

Über mich

- Fraunhofer IESE, Kaiserslautern
- Software-Entwickler
- Software-Architekt
- Abteilung Software-Architektur
  - Architektur-Schulung
  - Architektur-Bewertung

Digitale Dörfer
Kurzvorstellung

Digitale Dörfer Phase 2

Schauen sie sich unser Erklärvideo zu Phase 2 an:
https://www.youtube.com/watch?v=fHYpKpZzklo&t=1s

Digitale Dienste

DorfNews

DorffFunk

DorfDigital

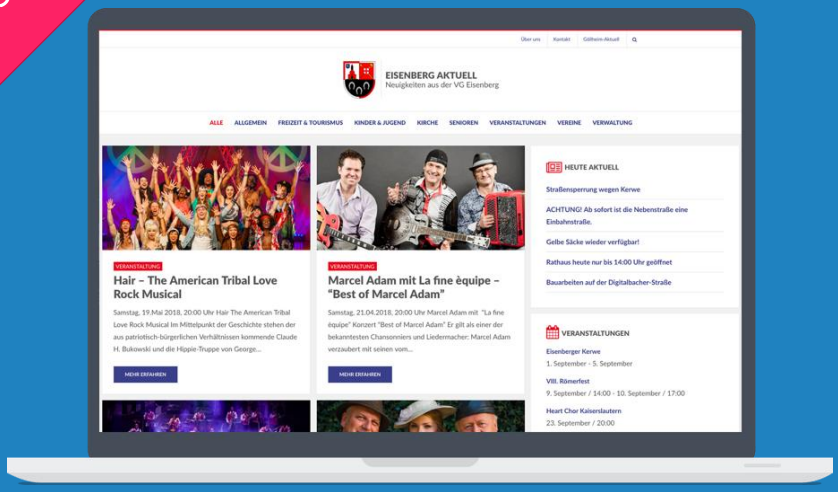
BestellBar

LieferBar

WordPress, PHP

DorfNews

Immer auf dem Laufenden!



Xamarin, iOS/Android

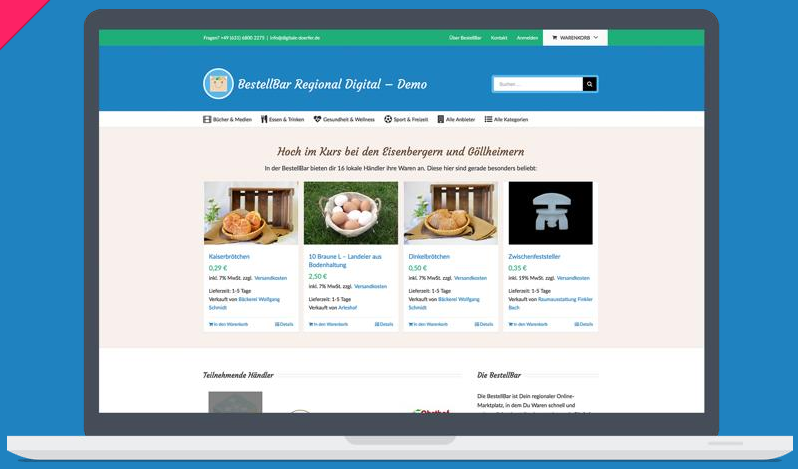
DorfFunk

Das Digitale Dorf in der Tasche!

WordPress, PHP

BestellBar

Der lokale Online-Marktplatz



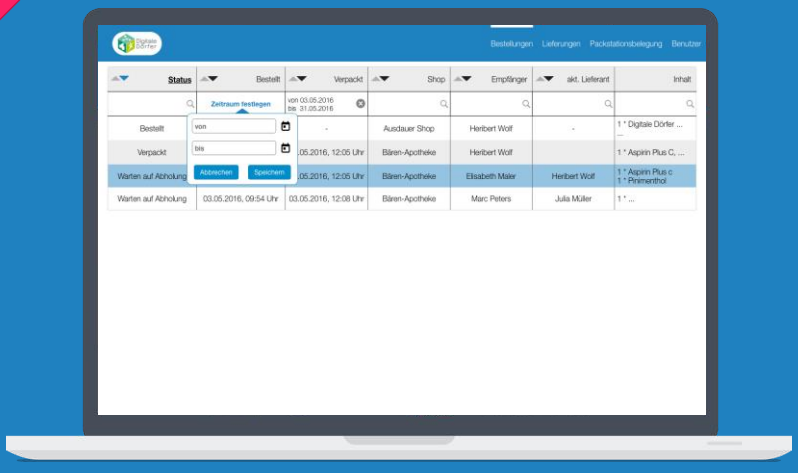
Xamarin, iOS/Android

LieferBar

Der flexibel einsetzbare Mitbringservice

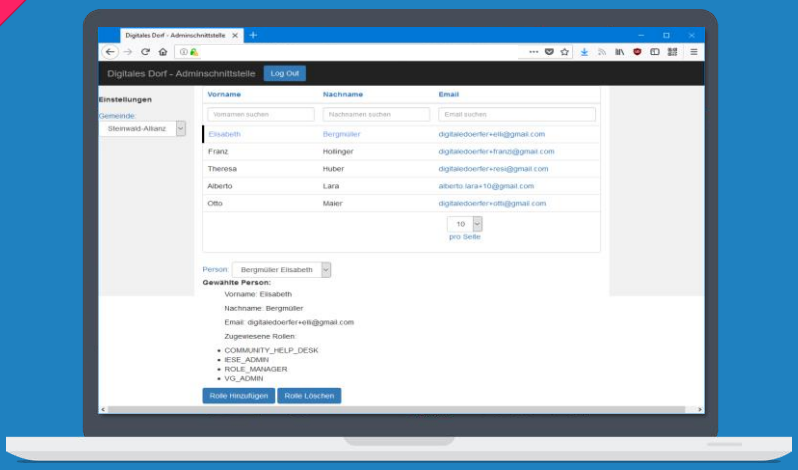
AngularJS

Admin-UI LieferBar/BestellBar



??? (Web-Technologien, OSGi Backend)

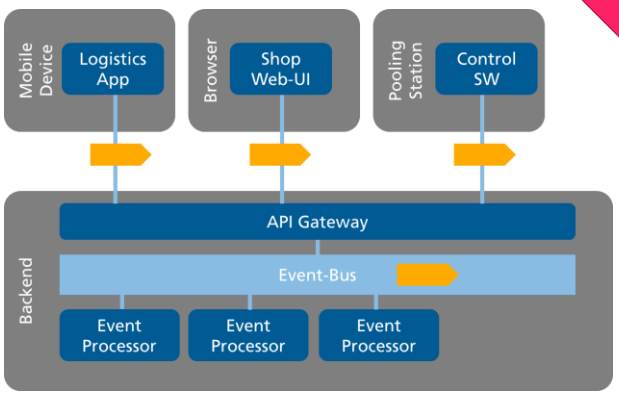
Admin-UI Rechteverwaltung



Java

Backend Basis für alle Dienste

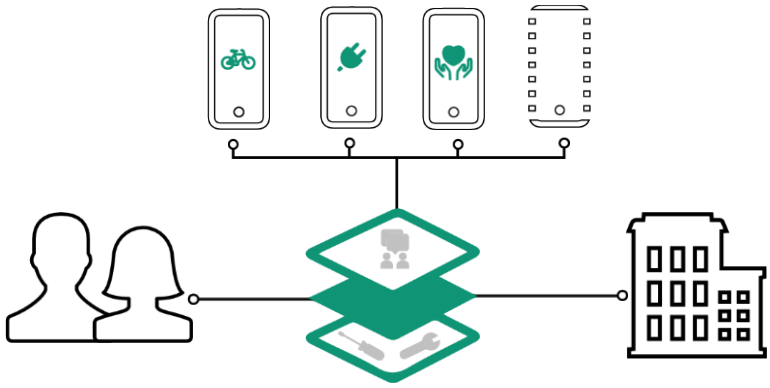
- Event-orientiert
- REST-Schnittstelle



Ausblick: Reallabor Pfaff

Vom Industrieareal zum klimaneutralen Quartier der Zukunft

Reallabor PFAFF



Feststellungen

- heterogene Technologien
- Benutzer-Login erforderlich z.B. für
  - DorfNews-Beitrag erstellen
  - DorfFunk-Plausch erstellen
  - BestellBar-Bestellung abgeben
  - Benutzerrollen verwalten
- Auch Drittanbieter-Anwendungen brauchen Zugriff auf Benutzerdaten

Kernfragen

Wie kann ein Benutzer unabhängig von der verwendeten Client-Technologie überall eine gleiche Login-Erfahrung machen?

Wie können wir es einem Benutzer ermöglichen, vorhandene Accounts zum Login zu verwenden?

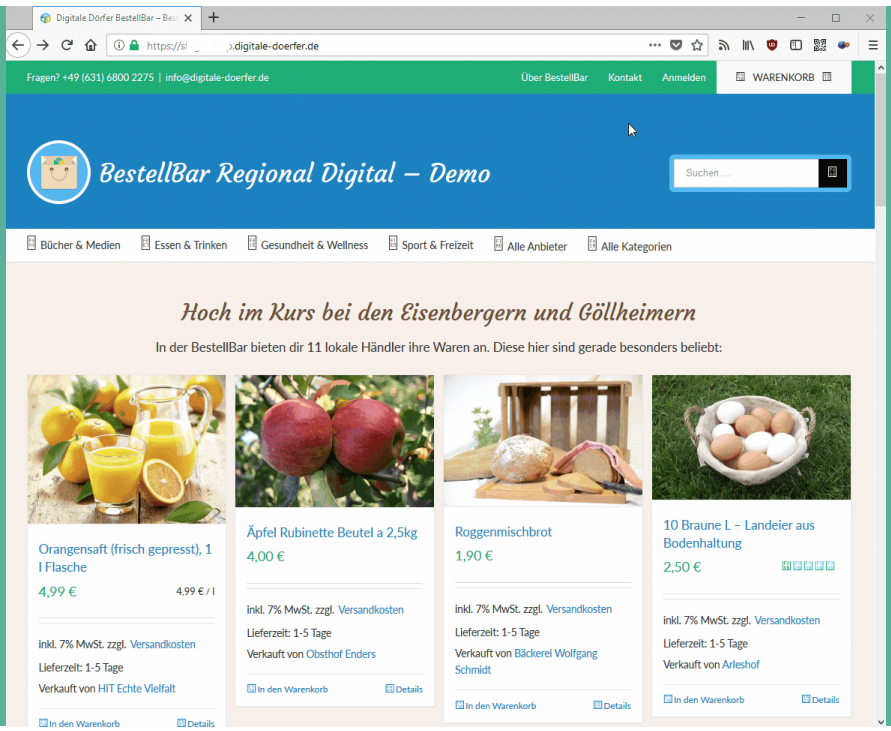
Wie kann eine Anwendung über längere Zeit Aufrufe im Namen des eingeloggten Benutzers tätigen, ohne dass sich der Benutzer neu einloggen muss oder die Anwendung das Passwort des Benutzers speichert?

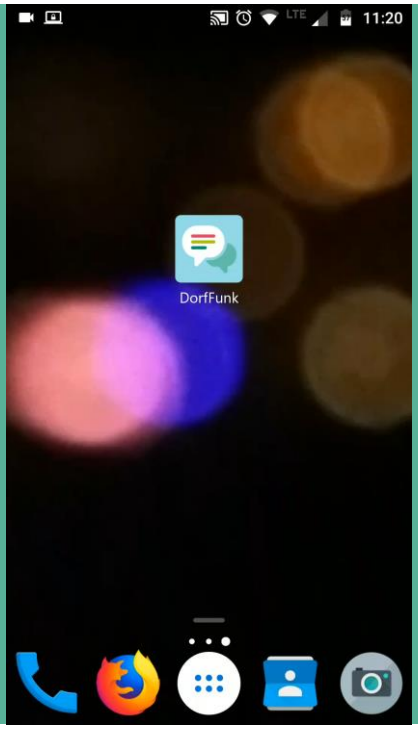
Wie kann ein Benutzer selbst entscheiden, welche Daten er einer Drittanbieter-Anwendung zur Verfügung stellt?

Wie verhindern wir, dass Drittanbieteranwendungen auf unserer Plattform Zugriff auf das Passwort des Benutzers erhalten?

Die Lösung:
Zentrale Authentifizierung mit OAuth2!

Digitale Dörfer
Umsetzung aus Benutzersicht



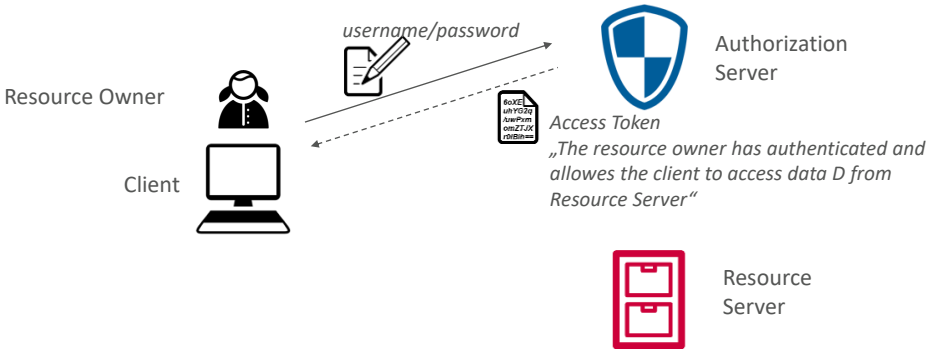


Wie OAuth2 funktioniert

Rollen

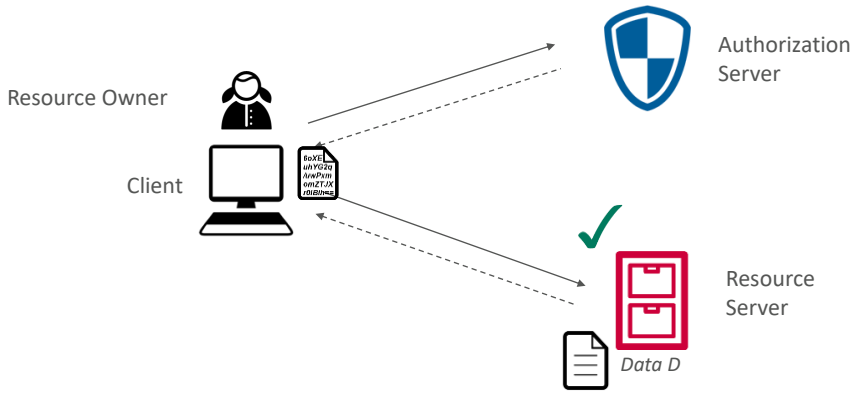
27

Genereller Ablauf



28

Genereller Ablauf



29

Scope

- frei definierbarer String, z.B. „read:profile read:location“
- OAuth2-Implementierung holt User-Consent ein
- Resource-Server muss sicherstellen, dass der übermittelte Scope zur Anforderung der Ressource berechtigt.

Hallo Johannes,

die Anwendung „DorfFunk“ möchte auf folgende Daten von dir zugreifen:

- dein Profil lesen
- deinen aktuellen Standort lesen

OK | Abbrechen

Im Detail

Access Token

- JWT-Format (JSON Web-Token)
- Base64-codiert
- Auch „Bearer Token“ genannt, weil es bei Requests im „Authentication“- Header als „Bearer <token>“ mitgeschickt wird

ey.CJhbGciOiJSUzI1NiIsImtpZCI6IlFVTkROVUZDTlRCQk1rW
kJI WTRNMEZETWpsQ05rSTBSall4UWtRNU9UVTVPQSJ9.ey
JodHRwOi8vZGlnaXRhbGUtZG9lcmZlci5kZS9lbWFpbF92ZXJpZmllZCI6ZmFsc
2UsImlzcyI6Imh0dHBzOi8vZGQtZGV2LmV1LmF1dGgwLmNvbS8iLCJzdWIiOiJ
hdXRoMHw1YWVjNDlhNGVmNzIxNTBjNjlhYTFkYWYiLCJhdWQiOlsiaHR0cH
M6Ly9kZXYuZGlnaXRhbGUtZ ZS8iLCJodHRwczovL2RkLWRldi5ldS
5hdXRoMC5jb20vdXNlcmluZn ljoxNTI1NDM0ODExLCJleHAiOjE1
MjU1MjEyMTEsImF6cCI6InhZV1VCdUtLNTRpY05ienZ3Uzg0Q05VMjh5MFVI
QWp4liwic2NvcGUiOiJvcGVuaWQgZW1haWwifQ.Gr_M8lphA2T1DTqiKDpgl7
9FfE8vEsdzRfqt1CZL3QnrFmvU-tjv63nFe2O9h_0mAMnDTMX0vmQ59XmFWt
ZZe7Bi0UzkGQUvVVLfscwAVFgLbSO81-00y7-Rcj5In-Pae6zSen9a8gkID44ERc
BbJeWfzqKs5A6Wvue5HiXCzpgLoAeVph47EERt2 T3g
d5zQSlO1iGk4upn-Dyla9QI-ZORKjOC1Ehr9FWYj
evqjVHBVP_T9s8Fjnnirp21eyJ76j6oiQs4ExhewrawdAbsKpOlYtuyCfvVNg4wcc
Tbc56YhQ

HEADER

BODY

SIGNATURE

```
{
  "typ": "JWT",
  "alg": "RS256",
  "kid": "QUND...TU5OA"
} , {
  "iss": "https://auth.digitale-doerfer.de/",
  "sub": "auth0|5aec49a4ef82150c69aa1dag",
  "aud": [ "https://api.digitale-doerfer.de/" ],
  "iat": 1525434811,
  "exp": 1525521211,
  "azp": "xYWUB28",
  "scope": "read:profile read:location"
  "http://digitale-doerfer.de/email_verified": false,
}
Signatur (Binärdaten)
```

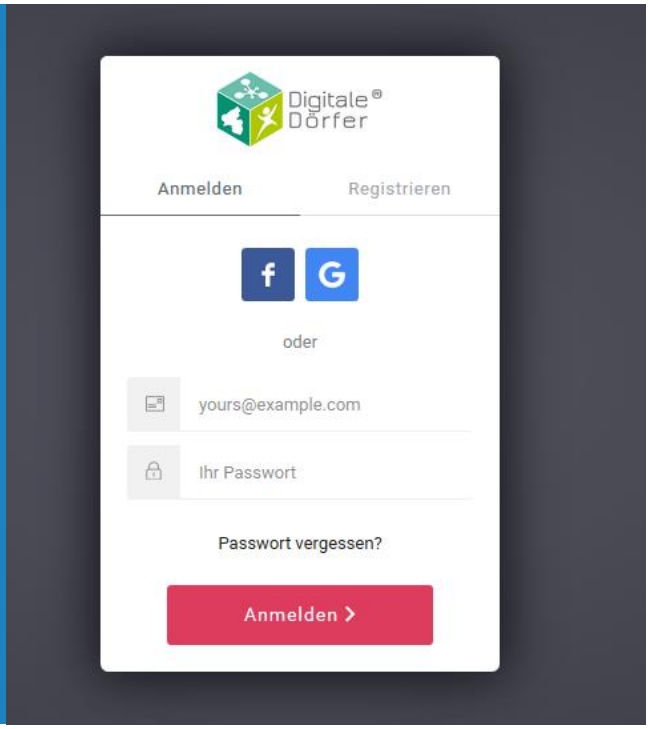
Issuer: Auth-Server
Subject: User-ID
Audience: Res.-Server API
Issued At: Timestamp
Expires: Timestamp
Authorized party: Client-ID
Scopes: erlaubte Ressourcen
Custom-Content

signiert mit privaten Schlüssel des Auth.-Servers
Res.-Server kann den öffentlichen Schlüssels speichern und offline das Token verifizieren.

Refresh-Token, Motivation

Bisheriger Flow:

Wenn access_token abgelaufen, erneuter Login (=Benutzerinteraktion) erforderlich!



Refresh-Token, Kurzfassung

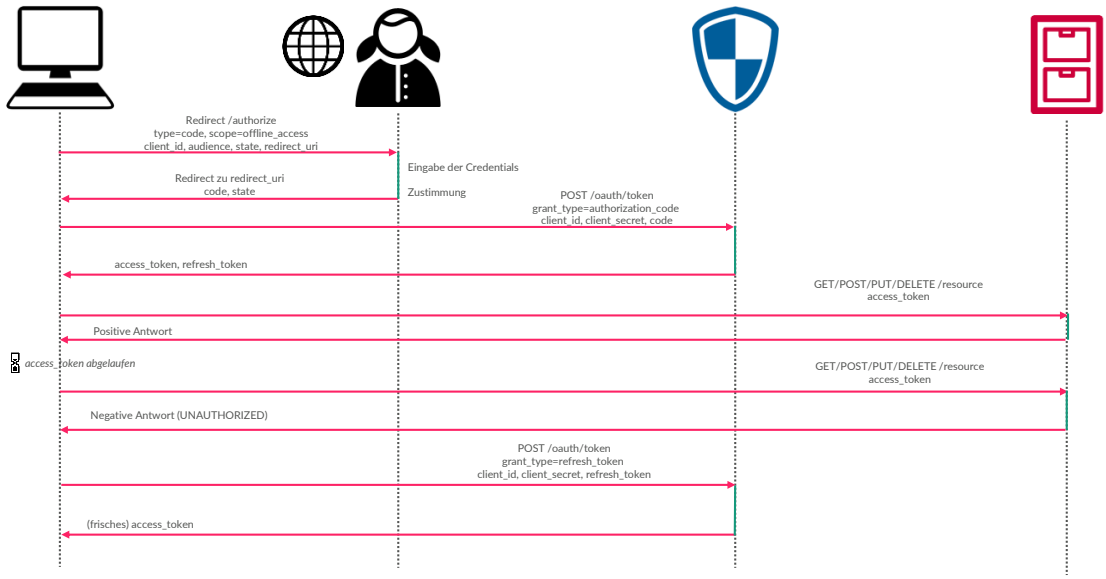
Durch
- Scope „offline_access“ und
- zusätzliche Sicherheitsmaßnahmen (client_secret)

erhält Client access_token und **refresh_token**.

Über das refresh_token bekommt der Client **ohne Benutzerinteraktion** ein neues access_token.

Ein Refresh-Token kann vom Resource Owner oder Auth.-Server-Admin zurückgerufen werden.

Refresh-Token-Flow



Vorteile

Vorteile

- App kann Passwort des Benutzers nicht abfangen
- Resource-Server braucht keine dauernde Online-Verbindung zum Authorization-Server
- Zentraler Login (Einheitliche Login-User-Experience über alle Anwendungen)
- Gute Einbindungsmöglichkeit von Drittanbieteranwendungen
- SSO über Browser-Cookie

Nachteile

Nachteile

- ⚠ Access-Token kann nicht widerrufen werden
  - Bei uns durch kurze Dauer und keine besonders hoher Sicherheitskritikalität verschmerzbar
- Benutzer verlässt die Anwendung, wenn er sich authentifiziert (vor allem auf Android auffällig)
  - Im Browser kein Problem (einfacher Redirect)
  - iOS: integrierte WebView
- Auth-Prozess aufwändiger zu implementieren
  - Hier helfen vorgefertigte Bibliotheken

Umsetzung

Was bietet Auth0?

- ready-to-use „Authentication as a Service“
- Fertiges Frontend (Login-/Registrierungs-Seiten) und Backend (token-Endpunkte, Benutzerverwaltung)
- Übersichtliche Verwaltungsoberfläche
- Toolsupport/Libraries für alle gängigen Programmiersprachen/Frameworks
- Login und E-Mails mit wenig Aufwand individualisierbar
- Sehr gute Erweiterbarkeit durch Hooks und Rules (in JavaScript geschrieben)

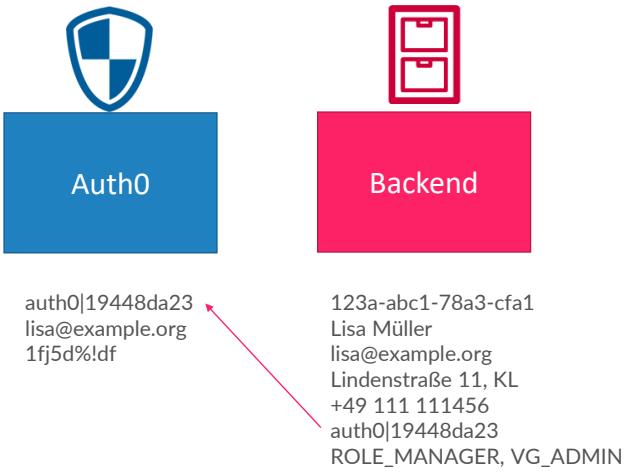
Entscheidung für Auth0?

- Bisheriges Authentifierungsverfahren (Mischung aus Session-ID und Amazon Cognito) soll schnell abgelöst werden
- Evaluierte Alternativ-Lösungen weniger ausgereift oder höherer Anpassungsaufwand

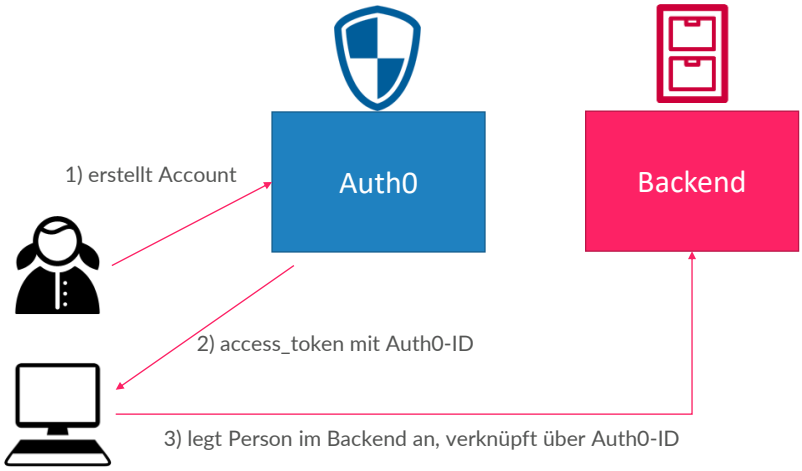
Fallstricke

Doppelte Persönlichkeit

Benutzerregistrierung



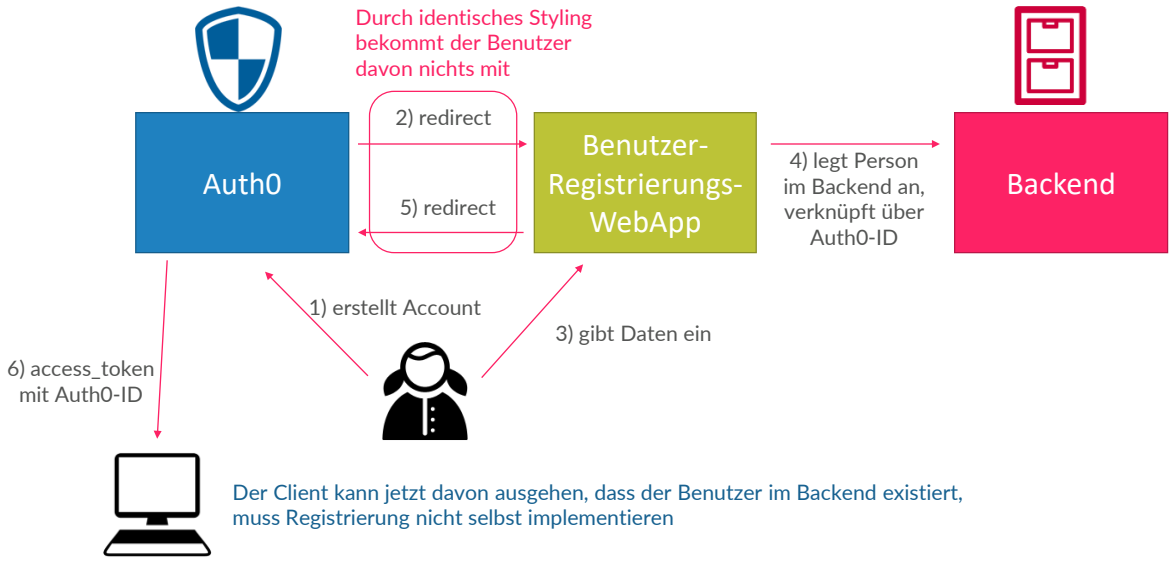
Benutzerregistrierung



Was passiert, wenn Registrierung fehlschlägt?
Mögliche Situation: Account existiert in Auth0, aber keine zugehörige Person im Backend

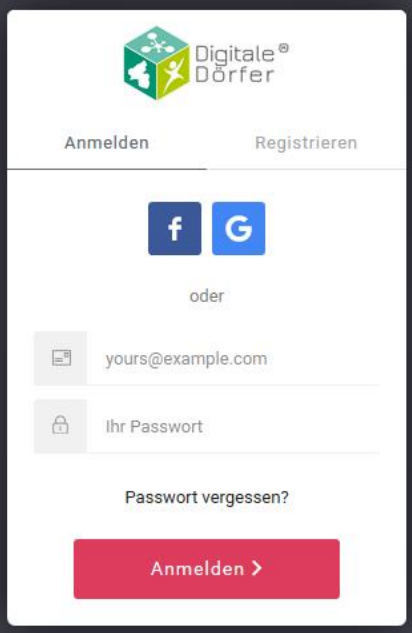
Jeder Client muss die Registrierung im Backend implementieren (Person anlegen mit Name, Adresse, Telefon, ...)

Benutzerregistrierung



User Experience
- oder -
„Wie zur Hölle hab ich mich da noch mal registriert?“

Facebook?



Google?

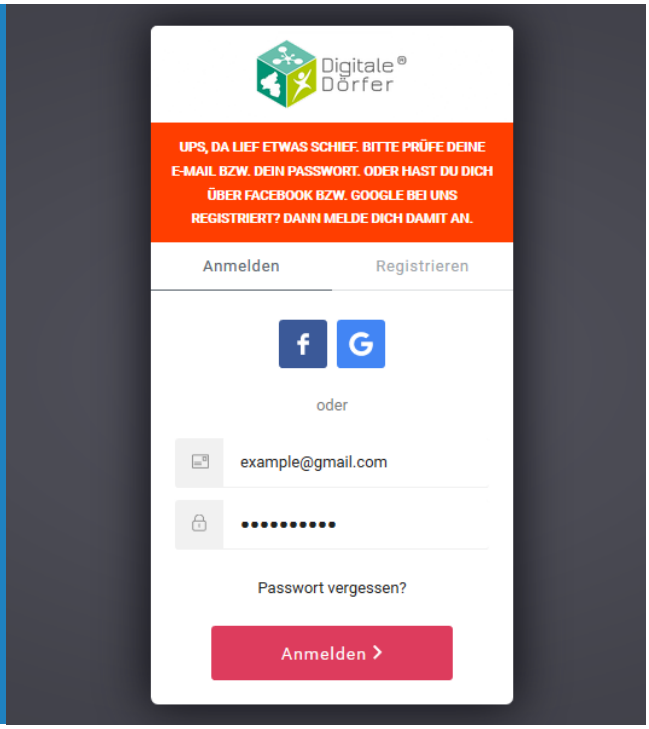
Eigener Account?

Fehlermeldungen

- … können hier helfen (wenn sie gelesen werden ☺)

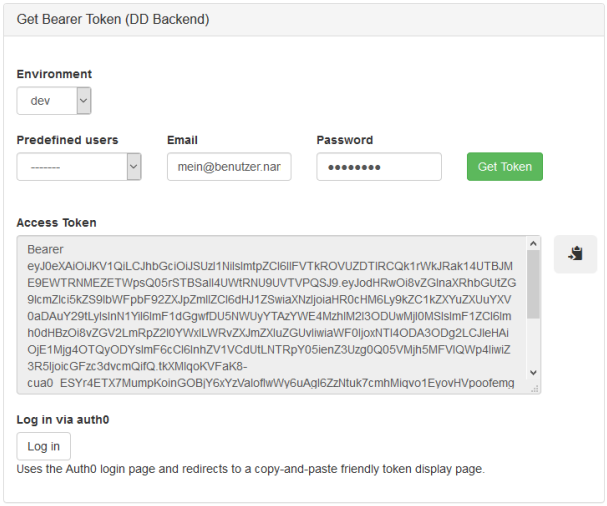
Außerdem:

- Erst eigener Account registriert, dann Social Login benutzte (gleiche E-Mail)
  → Accounts werden in Auth0 zusammengelegt

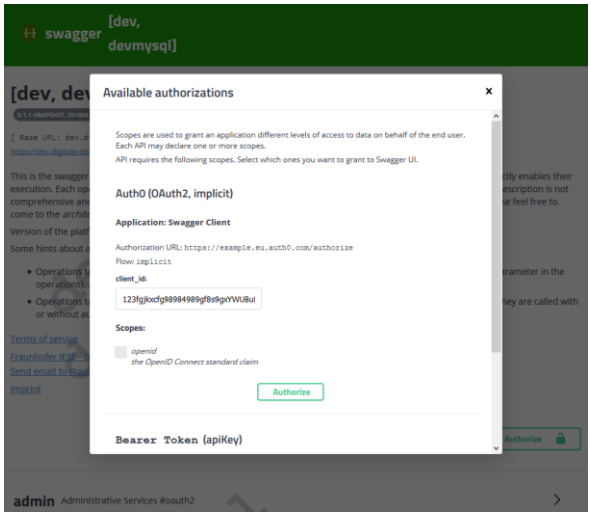


Wenn Entwickler mal schnell einen API-Aufruf machen wollen…

Wo bekomme ich das Bearer-Token her?



Mini-WebApp, die die Auth0-Management-API aufruft



OAuth-Integration von Swagger UI

Fazit

Lösung für die Eingangsfragen

- Gleicher Login unabhängig von Client-Technologie
- Social-Logins nutzen
- Über Datenfreigabe entscheiden
- Kein Zugriff auf das Passwort des Benutzers
- Dauerhafter Login ohne Speichern des Passworts

Aber:

- Der Teufel steckt im Detail, daher
  - Prototypen bauen
  - Lösungen vergleichen (no „one size fits it all“)

Dr. Johannes C. Schneider

Software-Architekt

johannes.schneider@iese.fraunhofer.de

Fraunhofer IESE, Kaiserslautern

digitale-doerfer.de

fb.com/DigitaleDoerfer

twitter.com/digitaledoerfer