Nutzerkonto Bund:

Modularität und Sicherheit

Für weitere Informationen kontaktieren Sie:
Christian Diemers c.diemers@tech4germany.org
Simon Zachau s.zachau@tech4germany.org

TECH 4 GERMANY

Informations Technik Zentrum Bund

Inhaltsverzeichnis

1 Einführung

1.1 Problem & Stakeholder

In Deutschland existieren über 11000 Kommunen. Durch die föderale Struktur sind bereits viele Portalverbunde auf Länder- und Kommunenebene (beispielsweise in Bayern, Baden-Württemberg, Berlin, Hamburg, etc.) entstanden. Auch einige Fachverfahren der Behörden wurden bereits digitalisiert.

Mit dem Online Zugangs Gesetz (OZG) sollen bis Ende 2022 insgesamt 575 im OZG-Umsetzungskatalog[1] beschriebenen Fachverfahren *digitalisiert* werden. Bürger, im Folgenden *Nutzer*, brauchen bisher für jedes Portal oder außerhalb von Portal gehosteten Fachverfahren einen separaten Login. Zwischen Portalen können Daten nicht ausgetauscht werden. Nicht zuletzt deshalb müssen Nutzer ihre Daten immer wieder erneut eingeben.
Ein weiteres Problem ist, dass ein Austausch von Daten mit Drittanbietern wie zum Beispiel Banken, Universitäten oder Krankenkassen, nicht möglich ist.

Im folgenden Dokument werden Drittanbieter und Anbieter von Fachverfahren als *Anbieter* zusammengefasst betrachtet.

1.2 Anforderung

Nutzer benötigen einen Zugriff auf alle Anbieter in Form eines sicheren und modularen Logins. Dieser Login eröffnet des Weiteren Zugriff auf sogenannte Mehrwertdienste. Mehrwertdienste sind beispielsweise der *Single-Sign-On*, über welchen sich der Nutzer bei Anbietern verifizieren kann. Auch der *Datensafe*, in welchem der Nutzer verschlüsselt Daten hinterlegen kann und bei jedem Anbieter, der für diese Daten autorisiert ist, transferieren kann, zählt als Mehrwertdienst. Andere Beispiele für Mehrwertdienste sind ein Postfach und Payment.
Die Architektur dieser Lösung muss drei Kriterien erfüllen. Erstens muss sie modular sein. Das bedeutet, dass durch einfache Schnittstellen Module miteinander verknüpft werden können und diese Module individuell je nach Bedarf skalierbar sind. Zweitens muss die Lösung sicher sein. Daher hält sich unsere Lösung an aktuelle Sicherheitsstandards, um gegen bekannte Cyberangriffe abgesichert zu sein. Drittens muss die Architektur Benutzerfreundlichkeit gewährleisten, damit das Nutzerkonto überhaupt verwendet wird. Mehr zur Benutzerfreundlichkeit im Nutzerkonto Bund Konzept.

1.3 Ziel

Auf dem Verwaltungsportal beta.bund.de lassen sich Anträge schnell finden. Bürgerinnen und Bürger sollen diese Anträge digital ausfüllen und absenden können ohne auf jedem Portal einen zusätzlichen Account zu erstellen oder sich erneut einloggen zu müssen. Dabei

[1] https://www.it-planungsrat.de/SharedDocs/Downloads/DE/Entscheidungen/26_Sitzung/TOP2_Anlage_OZGUmsetzungskatalog.pdf?__blob=publicationFile&v=4 (2018)

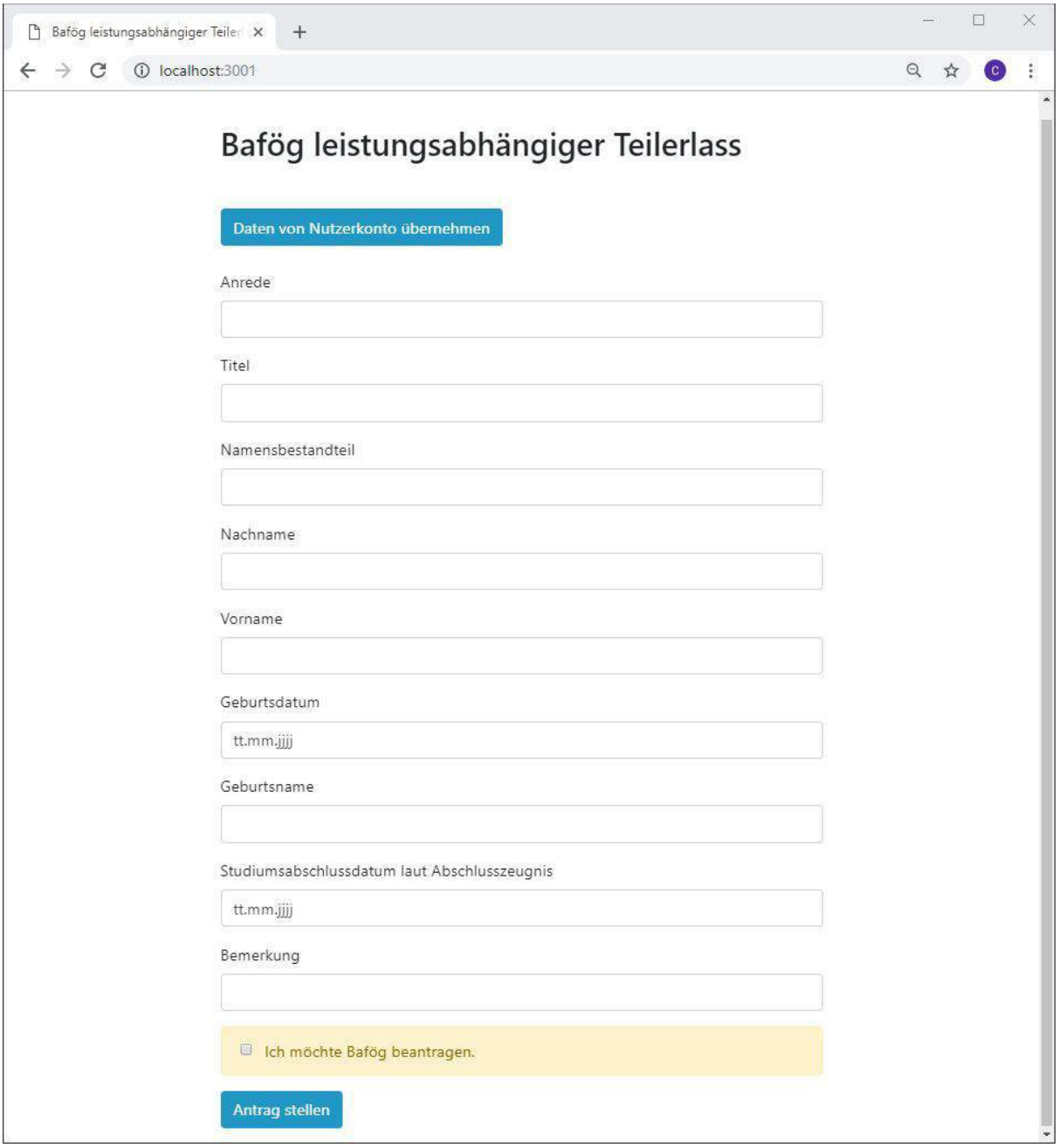
sollen auch in irgendeinem der Anträge bereits eingegebene Daten in andere Anträge übernommen werden können, sprich dieselben Daten müssen nicht mehrmals eingeben werden.

2 Prototyp

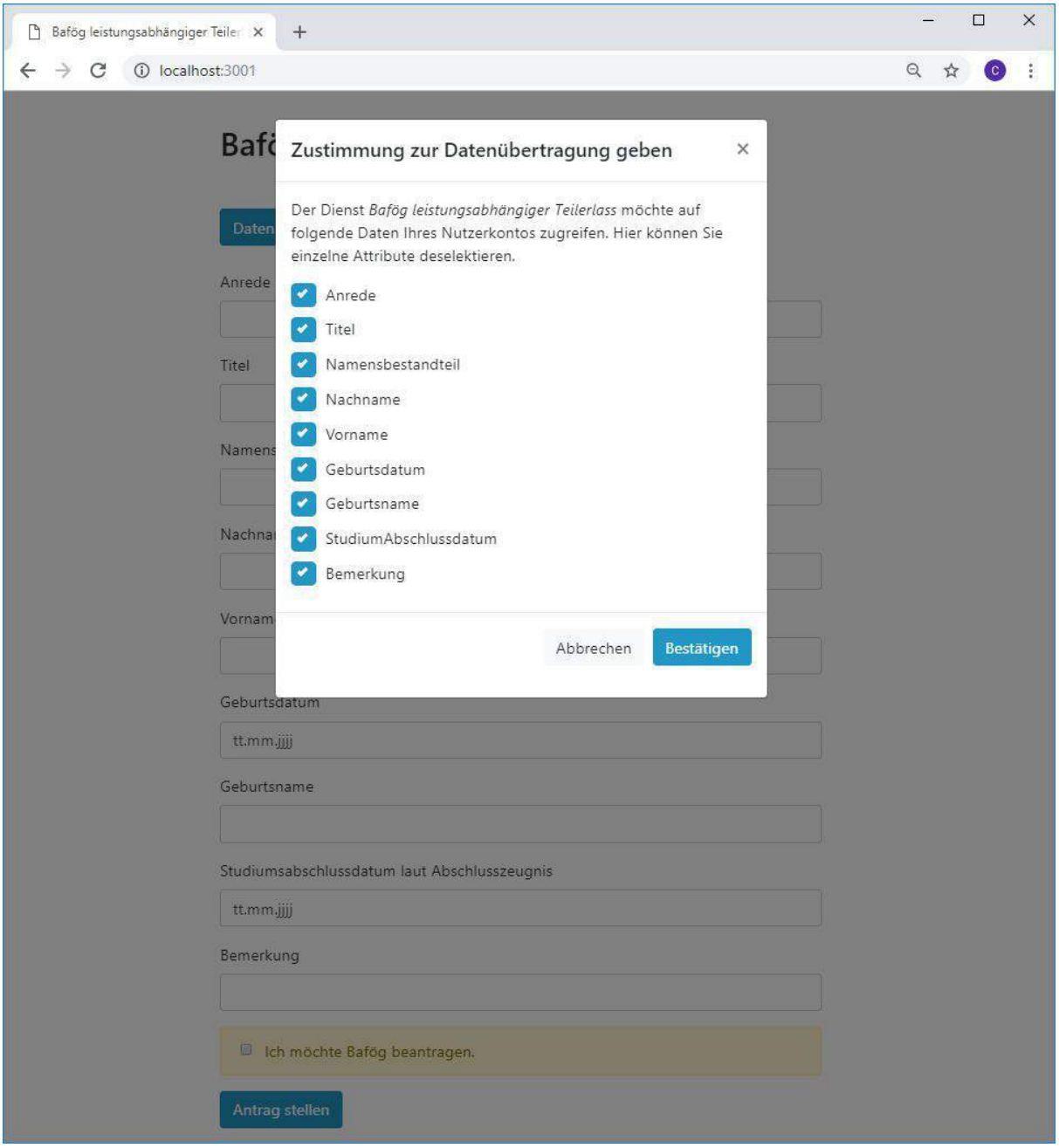
Der Spagat bei der Entwicklung großer Projekte wie dem Nutzerkonto Bund ist das Zusammenspiel der langen Planung einerseits und dem agilen Programmieren und Testen andererseits. Im Rahmen von Tech4Germany setzen wir auf die iterative Ausarbeitung modularer Bestandteile des Nutzerkonto Bund statt auf die vorherige detaillierte Planung der kompletten Architektur. Wie die komplette Architektur funktioniert, ist im Abschnitt SAD beschrieben.

2.1 Showcase

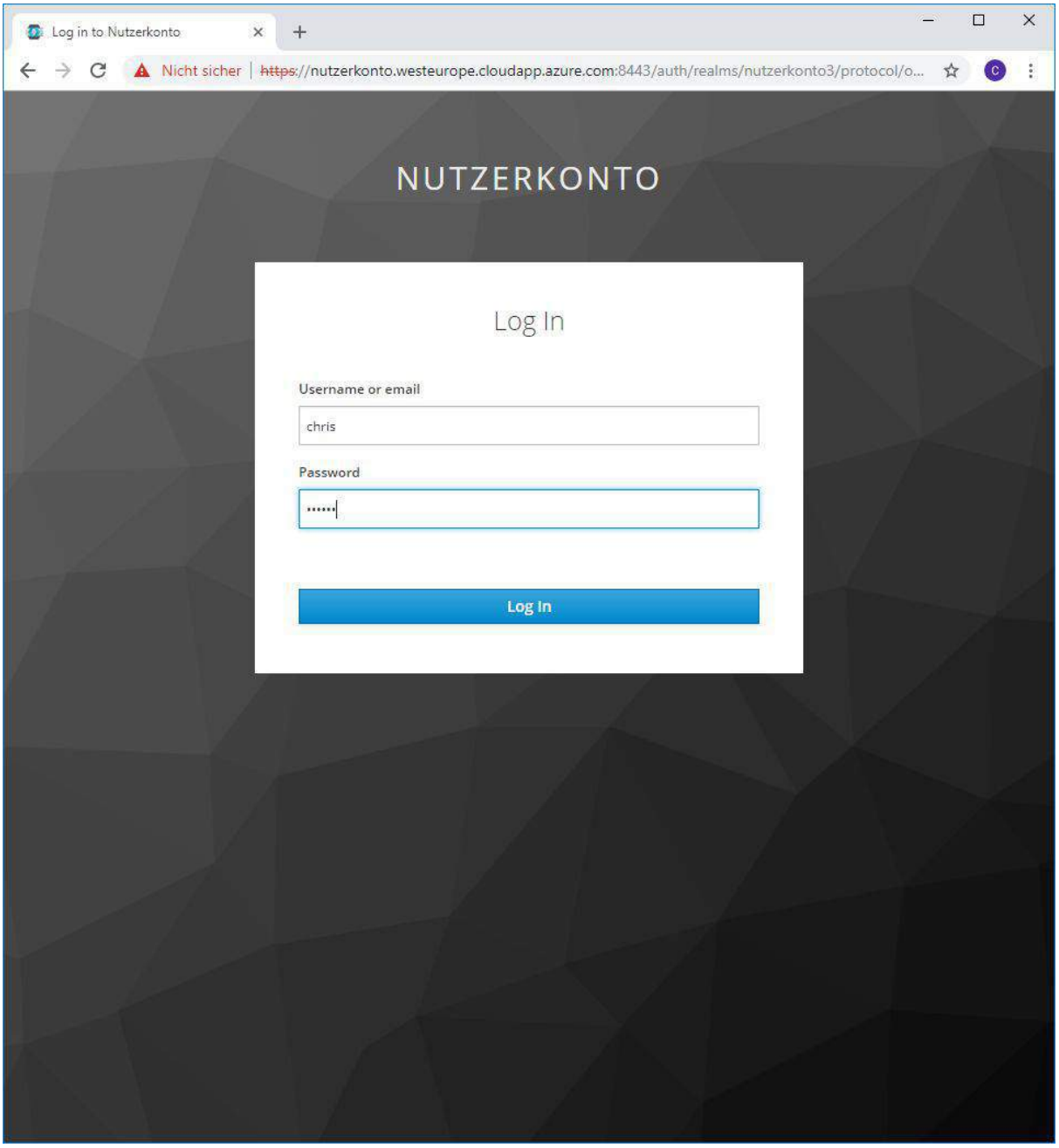
Im Folgenden beschreiben wir das Ergebnis der iterativen Entwicklung des Datentransfers, sprich das Übernehmen von Daten des Nutzerkontos in ein Fachverfahren. Der Prototyp ist das Hauptergebnis dieser Arbeit und dient gleichzeitig der Validierung von Architekturentscheidungen (siehe Abschnitt Technologieanalyse). Der verwendete Use Case, der die Datenübertragung in der Praxis zeigt, ist das Fachverfahren *BAföG leistungsabhängiger Teilerlass*.



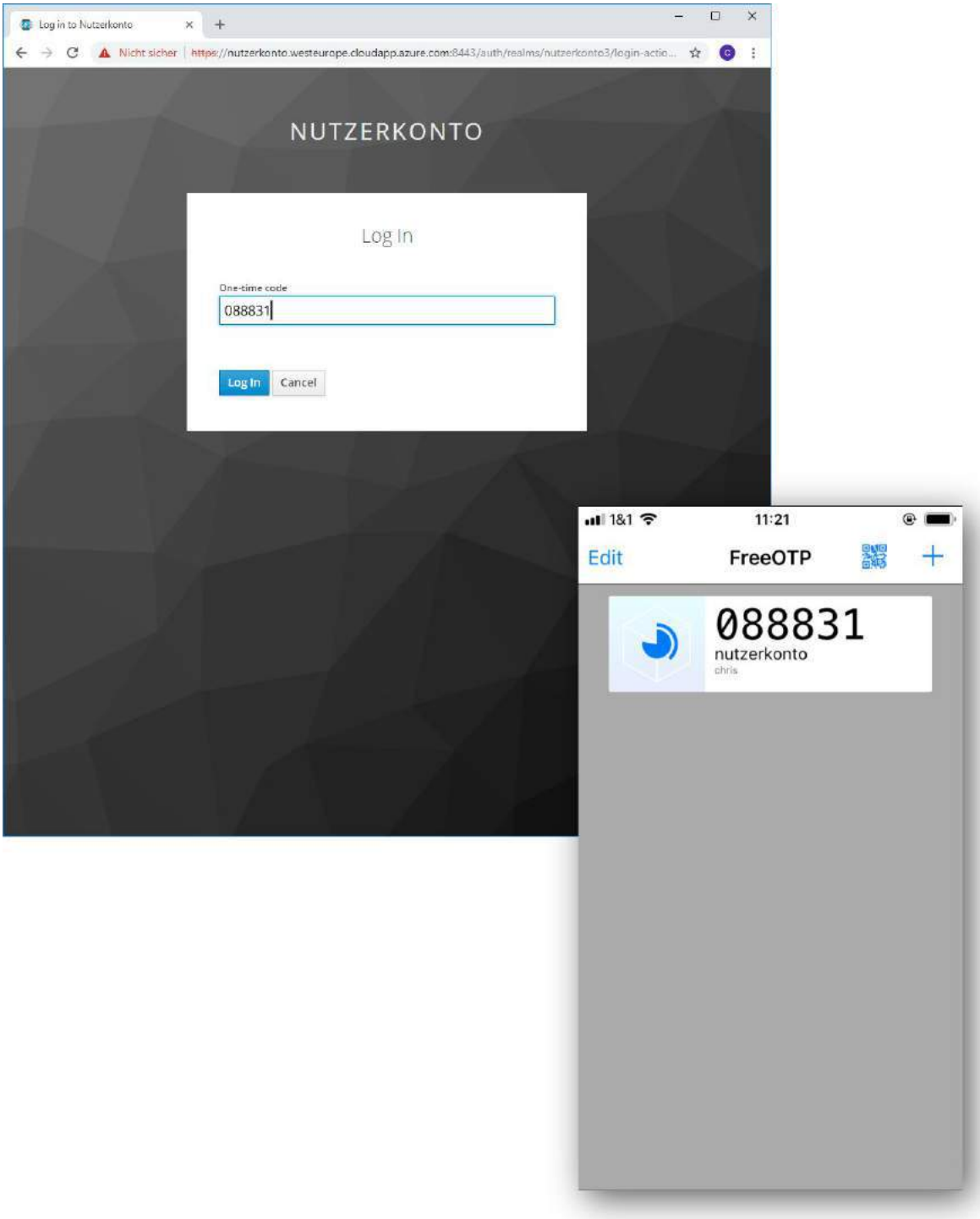
Formular des Fachverfahrens mit der Möglichkeit seine Daten aus dem Nutzerkonto zu übertragen oder die Daten manuell einzutragen.



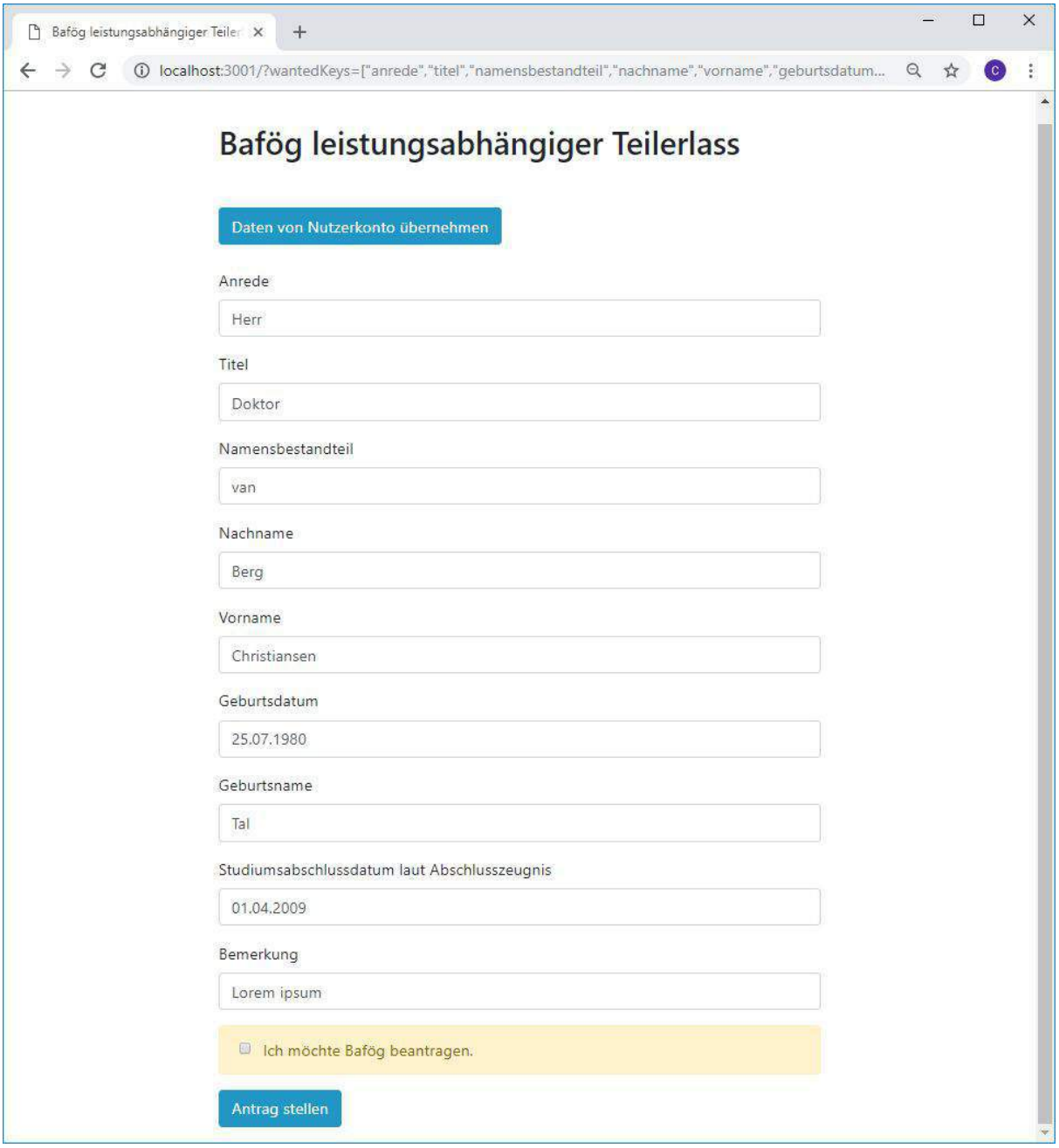
Der Nutzer wählt die Option Daten aus dem Nutzerkonto zu übertragen und sieht dabei welche Daten übertragen werden. Dadurch hat der Benutzer die volle Kontrolle und Transparenz über die Verwendung seiner Daten.



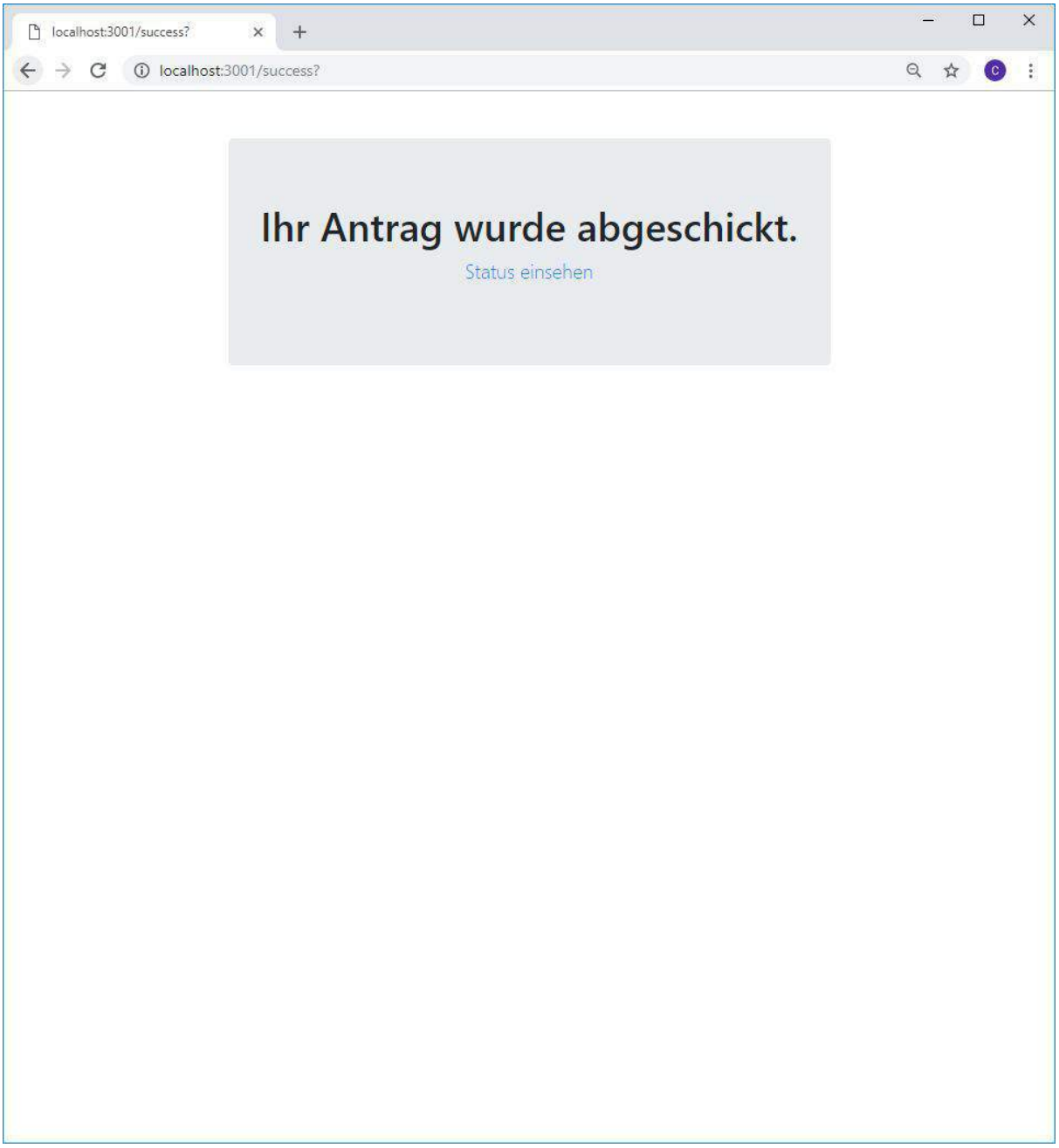
Nach dem Bestätigen wird der Nutzer auf die Nutzerkonto Anmeldemaske weitergeleitet. Hier meldet er sich mit Benutzername und Passwort an, so wie er es von anderen Plattformen gewohnt ist.



Dabei kommt der zweite Faktor bei der Authentisierung ins Spiel. Der Nutzer muss seine Anmeldung mit einem Time-Based One Time Password (TOTP) seiner Authenticator App bestätigen. Das TOTP kann auch durch einen beliebigen anderen zweiten Faktor ersetzt werden, wie zum Beispiel einen Push-Token oder den neuen Personalausweis.



Der Nutzer wird wieder zurück auf die Fachverfahren-Seite geleitet und seine Daten werden automatisch übernommen. Jetzt muss er nur noch die fehlenden Daten ausfüllen und kann den Antrag abschicken.



Der Antrag wurde erfolgreich abgeschickt. Die Session bleibt für weitere Fachverfahren innerhalb eines festgelegten Zeitfensters (z.B. 30 min) bestehen.

2.2 Installationsanleitung

Setup Nutzerkonto und exemplarischer Anbieter

- das Nutzerkonto[2] und das Anbieter[3] Repository klonen
- *npm install* in beiden Verzeichnissen ausführen
- jeweils eine *.env*-Datei im Root-Verzeichnis der Projekte mit den folgenden Attributen erstellen

Attribute	Beschreibung	Beispiel
HOST_NUTZERKONTO	URL für das Nutzerkonto Bund	http://localhost:3000
HOST_NUTZERKONTO_SP	URL für den exemplarischen Anbieter	http://localhost:3001

- nur für das Nutzerkonto: die *.config*-Datei im Root-Verzeichnis dahingehend ändern, dass die *KEYCLOAK*-Variable auf die eigene Installation von Keycloak zeigt (Installation findet im folgenden Schritt statt)
- nur für das Nutzerkonto: Ändern der *keyCloakClientID* in Zeile 31 der Datei *app.js*, sodass sie mit der ID, die in "Setup Keycloak" vermerkt wird, übereinstimmt.
- *npm start* in beiden Verzeichnissen ausführen

Setup Keycloak
Um den Prototypen zu testen, ist die Installation des Keycloak Servers eine notwendige Voraussetzung, da er die zentrale IAM-Komponente darstellt.
Als Ergänzung zur offiziellen Dokumentation ist die folgende Anleitung auf codeburst.io sehr empfehlenswert.

Grundsätzliche Schritte:

1. Herunterladen des Keycloak Server Images
2. Entpacken des Images
3. Erstellen des ersten Benutzers ist mit dem CLI- Tool *./bin/add-user-keycloak.sh*
4. Starten des Keycloak Servers mit *./bin/standalone.sh -b=0.0.0.0*
5. optional; einrichten des SMTP Servers für den Mailversand zur Benachrichtigung von Benutzern: Realm Settings → Email → SMTP Informationen eingeben (z.B. Gmail Server Daten)

Weiterführende Themen werden recht ausführlich in der offiziellen Dokumentation behandelt.

[2] https://github.com/tech4germany/nutzerkonto
[3] https://github.com/tech4germany/nutzerkonto-anbieter

3 Software Architecture Document (SAD)

3.1 Rahmenbedingungen

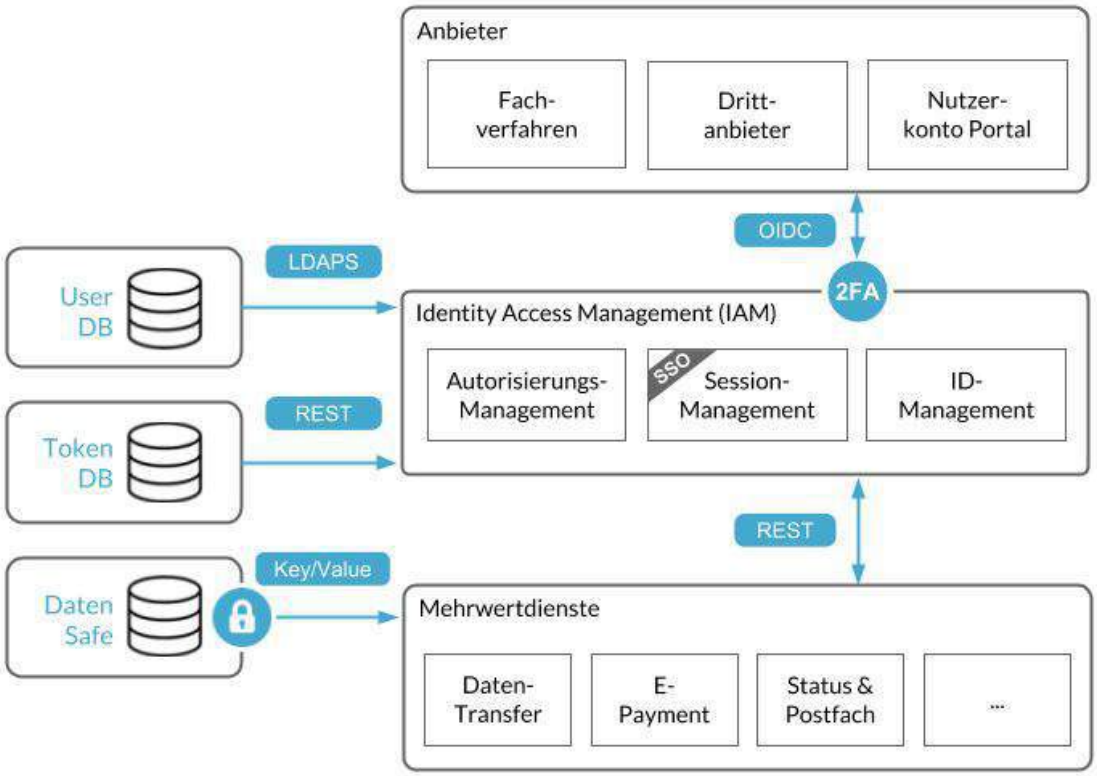
Jeder Vorgang, wie zum Beispiel das Einfüllen von Daten oder das Absenden eines Antragsformulars, benötigt ein bestimmtes Vertrauensniveau, namentlich *normal*, *substanziell*, oder *hoch*. Diese Sicherheitsvorschriften stammen vom BSI und müssen verpflichtend umgesetzt werden.
Das derzeitige Problem ist, dass eine *substanzielle* Verifizierungsmöglichkeit fehlt. Darum wird meist auf den nPA, der eIDAS-konform das höchste Vertrauensniveau unterstützt, jedoch technisch wenig flexibel und wenig nutzerfreundlich ist, zurückgegriffen. Erstens müssen weitere Verfahren, die ein substanzielles Vertrauensniveau gewährleisten, notifiziert werden. Zweitens muss die IAM-Komponente des Nutzerkontos eine einfache Integration dieser Verifizierungsverfahren bieten.
Für flexible und standardisierte Schnittstellen ist der Aufbau des Nutzerkontos in der Form einer modularen Container-Architektur wegweisend. Weiterhin ermöglicht dies, Mehrwertdienste und Anbieter dezentral zu hosten.

3.2 Umfang

Wir setzen die IAM-Komponente und den Datentransfer als exemplarischen Mehrwertdienst um. Sowohl die 2FA-SSO-Schnittstelle (im Prototyp mit Keycloak) als auch der Transfer der Daten sind wie beschrieben umgesetzt.
Die Datenbanken wurden im Prototyp durch statische JSON-Antworten ersetzt und lassen sich in der tatsächlichen Umsetzung gemäß Standards verwirklichen. Eine Option bei der UserDB wäre hier beispielsweise der LDAPS-Standard, der es ermöglicht, das Session- und ID-Management unabhängig voneinander zu betreiben, da beide (zum Beispiel LinOTP für das ID-Management und Keycloak für das Session-Management) somit auf dieselben Daten zugreifen können. Wenn die IAM-Komponente als eine Einheit betrachtet wird ohne Unabhängigkeit des ID- und Session-Managements, bestehen selbstverständlich alle gängigen Alternativen zu LDAPS.

Das **ID-Management** verwaltet die Schnittstelle mit der UserDB und ist primär für Authentifizierung der Benutzer zuständig, bzw für Login und somit den Aufbau der Session. Das **Session-Management** ist für die Verwaltung der Session und den SSO zuständig. Das SSO ermöglicht das Speichern einer Session, um innerhalb einer Session ein erneutes Einloggen bei Fachverfahren, Portalen und sonstigen Anbietern zu vermeiden. Das **Autorisierungs-Management** verwaltet welche Mehrwertdienste mit welchen Nutzern kommunizieren dürfen und insbesondere, auf welche Daten zugegriffen werden darf. Siehe mehr dazu im Absatz Mehrwertdienst-Prozess im Abschnitt API.

Im folgenden Diagramm sind alle Komponenten und wie sie miteinander verknüpft sind dargestellt.



Anbieter rufen die im Abschnitt API beschriebenen Schnittstellen auf, um auf Dienste via der IAM-Komponente, die als Middleware betrachtet werden kann, zuzugreifen. Dies geschieht über REST (siehe Abschnitt Technologieanalyse). Durch diese Architektur ermöglichen die einzelnen Komponenten eine einheitliche und entkoppelte Anbindung an das Nutzerkonto und sind gleichzeitig technologieunabhängig.

3.3 Modularität

Um das Nutzerkonto modular zu gestalten, stellen alle Mehrwertdienste und Services erstens APIs bereit und können zweitens unabhängig voneinander in Containern gestartet werden. Im Folgenden wird auf diese beiden Punkte genauer eingegangen.

3.3.1 API

Alle hier aufgeführten Schnittstellen werden als externe Schnittstellen kategorisiert. Dies erlaubt, die Mehrwertdienste so modular zu gestalten, dass sie problemlos von externen und nicht nur internen Anbietern verwendet werden können.

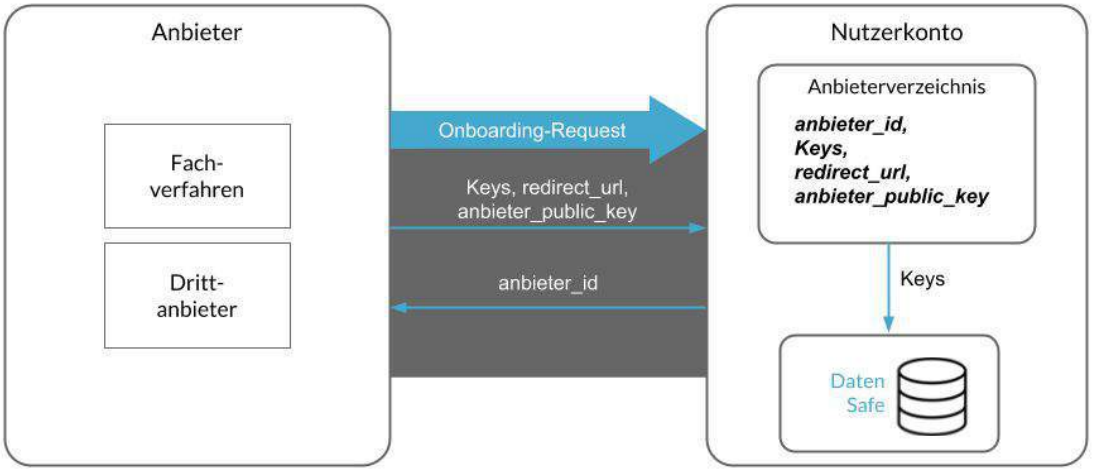
Des Weiteren kann ein Anbieter beliebig viele Anträge erstellen (siehe folgendes Diagramm). Allerdings vereinfachen wir das Modell in unserem Prototypen zugunsten übersichtlicherer API-Beschreibungen, sodass einem Anbieter nur ein Antrag zugeordnet wird.

Die API-Endpoints sind in gemäß des Diagramms im Abschnitt Umfang in drei Kategorien unterteilt.

1. Die **Onboarding**-Schnittstelle, über welche sich Anbieter registrieren können, um Nutzern Fachverfahren, andere Anträge und Mehrwertdienste zur Verfügung stellen zu können
2. Die **Authentifizierungs**-Schnittstellen, welche der Anbieter integriert, um Bürgern Zugriff zu SSO zu geben
3. Die **Mehrwertdienst**-Schnittstellen, welche der Anbieter integriert, um Bürgern Zugriff zu Mehrwerten zu geben

Die APIs aller drei Punkte und die Visualisierung deren Kontexte werden im Folgenden erläutert.

Der im folgenden Diagramm dargestellte **Onboarding**-Prozess muss nur einmal durchgeführt werden.



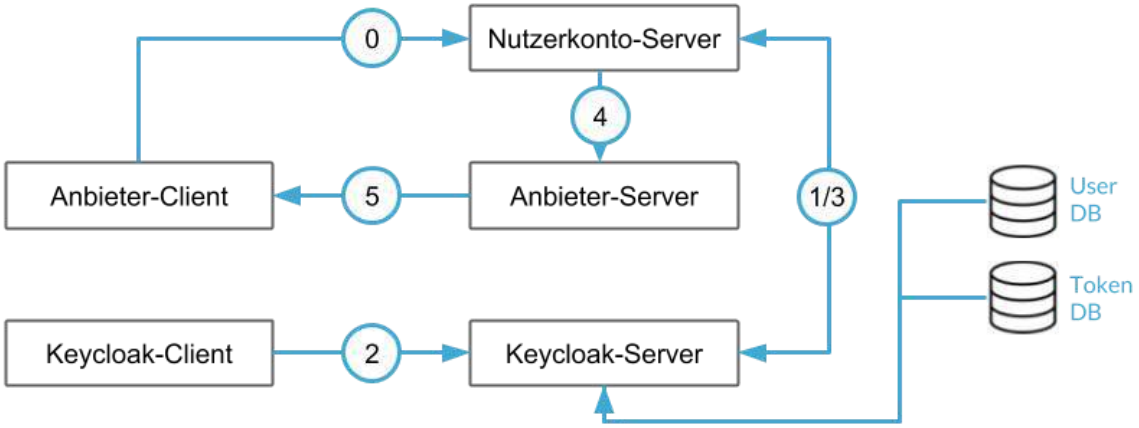
Durch den Prozess wird ein Anbietereintrag auf Seite des Nutzerkontos mit einer *anbieter_id*, mit welcher er sich gegenüber Mehrwertdiensten ausweisen kann, erstellt. Außerdem darf ein Anbieter beantragen, Datenfelder (im Folgenden *Keys*) übertragen zu wollen. Um übertragene Daten der Mehrwertdienste zu entschlüsseln, generiert er ein Key-Pair und übermittelt den public key als *anbieter_public_key* dem Nutzerkonto. In dem Feld *Keys* des Anbieterverzeichnisses wird hinterlegt, auf welche Daten des Datensafes der entsprechende Anbieter überhaupt zugreifen darf. Der Datensafe ist abstrakt betrachtet wie eine zentrale Key-Value (Schlüssel-Wert)-Tabelle. Die Keys entsprechen dem vorhandenen XÖV Standard[4], womit Daten Fachverfahren-übergreifend verwendet und gespeichert werden. In der folgenden Tabelle sind die API-Endpoints bezüglich des Onboarding-Prozesses gelistet.

PFAD	**PARAMETER & HEADER ATTRIBUTE**	**ANTWORT**
/registriere-anbieter	**Keys**, welche bei der Datenübertragung abgefragt und zurückübertragen werden	**anbieter_id**

[4] https://www.xoev.de/die_standards/xoev_standards_und__vorhaben-11430

	anbieter_public_key, zur Authentifizierung und Autorisierung der Datenübertragung **redirect_url** des anbieters für redirect bei der Authentifizierung	

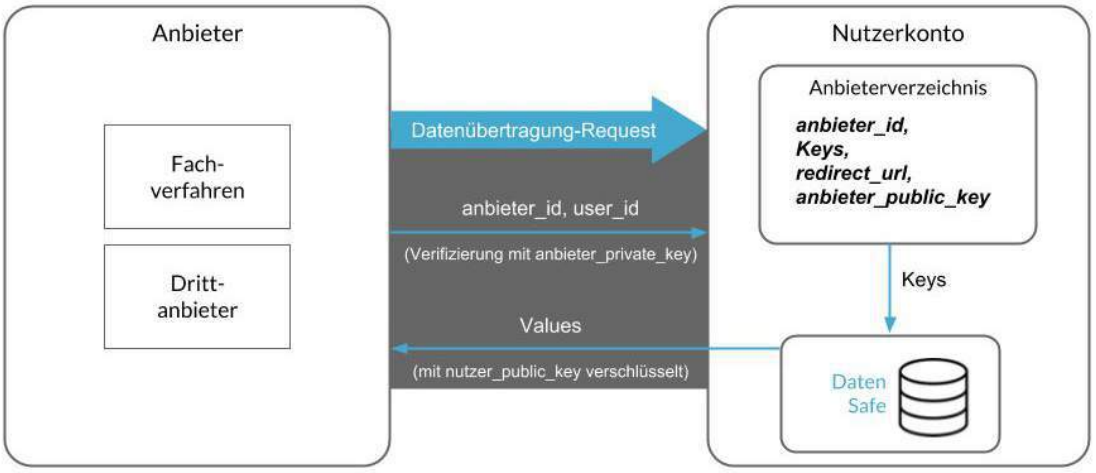
Der **Authentifizierungs**-Prozess läuft wie im folgenden Diagramm ab. Das Session-Management (prototypisch mit Keycloak, also der Keycloak-Server) speichert die Sessions zentral während beim Anbieter-Client nur die Session-ID in Form eines Cookies abgelegt wird. Der Transfer findet selbstverständlich via TLS abgesichert statt.



In der folgenden Tabelle werden die API-Endpoints des Authentifizierungs-Prozesses gezeigt. Nach dem Erstellen der Session ist SSO innerhalb des Zeitfensters möglich.

PFAD	**PARAMETER & HEADER ATTRIBUTE**	**ANTWORT**
[NUZTERKONTO_HOST]/ sso-login	**anbieter_id** für lookup auf Nutzerkonto der redirect_url **mindest_vertrauensniveau**	**erfolgreicher Login**: Session wird kreiert und redirect zum Anbieter **gescheiterter Login**: Abbruch
autom. nachgela-gerter IAM Request auf der Nutzer-konto-Seite	je nach Verifizierungsverfahren; z. B. **username**, **password**, **TOTP** oder Elster-Anbindung für ein substanzielles Vertrauensniveau oder AusweisApp2-Anbindung für höchstes Vertrauensniveau	
[NUZTERKONTO_HOST]/ sso-logout	**Session-ID Cookie** im Header	Logout erfolgreich oder nicht; Session in der IAM-Komponente wird terminiert

Als beispielhafter **Mehrwertdienst**-Prozess wird die *Datenübertragung* im folgenden Diagramm visualisiert.



Es werden die *anbieter-* und *user_id* benötigt. Die für die anbieter_id hinterlegten Keys und die user_id identifizieren die zu übermittelnden Values im Datensafe. Die Values, welche mit dem public key des Nutzers, dem *nutzer_public_key*, verschlüsselt sind, werden daraufhin dem Anbieter zurückgesendet. Mit dem *nutzer_private_key* entschlüsselt der Nutzer sie. Dies setzt voraus, dass der Nutzer ein Key-Pair erstellt und dieses mit dem Nutzerkonto verknüpft. Somit werden seine Daten im Datensafe stets verschlüsselt aufbewahrt, sodass sie ohne den Key unbrauchbar sind.
Auch anbieterspezifische Nutzerdaten werden nicht beim Anbieter gespeichert sondern im Datensafe. Dies ermöglicht die zukünftige Nutzung besagter Daten in anderen Angeboten und verhindert den Bedarf eines Nutzermappings.
In der folgenden Tabelle werden die API-Endpoints bezüglich des Datenübertragung-Prozesses genannt.

PFAD	**PARAMETER & HEADER ATTRIBUTE**	**ANTWORT**

GET **[NUZTERKONTO_HOST]/ datenuebertragen**
 	
Übertragung vom Datensafe
	anbieter_id für lookup der Keys
	
	user_id für lookup der Values der Keys
	
	Session-ID Cookie im Header
		JSON der vorliegenden vom Nutzer verschlüsselten Daten, die der Anbieter zum Ausfüllen verwenden darf, z. B.
		
		body:{
		vorname: “”,
		strasse: “dpjivloj83izerb89wiubvivnlvjkjbvojpnioopjm”
		…
		}

POST **[NUZTERKONTO_HOST]/ datenuebertragen**
 	
Rückübertragung zum Datensafe
	anbieter_id für Zuordnung der zu übertragenden vom Nutzer verschlüsselten Daten
	
	Session-ID Cookie im Header
	
	body:{
	vorname: “”,
	strasse: “dpjivloj83izerb89wiubvivnlvjkjbvojpnioopjm”
	…
	}	

weitere Mehrwertdienste		

3.3.2 Containerisierung

Die Architektur wird in Form von Microservices umgesetzt. Das bedeutet, dass es statt einem monolithischen System viele einzelne Subsysteme, die untereinander kommunizieren, gibt. Nur so wird Modularität, individuelle Skalierbarkeit und Erweiterbarkeit der einzelnen Komponenten gewährleistet. Als Infrastruktur für die Umsetzung wird Docker verwendet.

3.4 Sicherheit

Im Folgenden beschreiben wir die sicherheitsrelevanten Aspekte der vorliegenden Architektur. Zur besseren Übersicht sind diese in die Bereiche Session Management, OpenID Connect, Container, Authentisierung und Verschlüsselung unterteilt.

3.4.1 Session Management[5]

Um ein gewisse Sicherheit zu erreichen ist der erste Schritt die Erstellung einer Session-ID für einen angemeldeten Benutzer. Diese ID wird dazu verwendet, um eine auf dem Server gespeicherte Session (inklusive *Access*-und *ID*-Tokens, siehe OIDC Standard) mit dem Benutzer in Verbindung zu bringen. Die Tokens werden nach vorgegebenem JWE Standard [6](RFC 7516) signiert und verschlüsselt, damit im Falle eines erfolgreichen Angriffs auf den Server die Tokens nicht direkt ausgelesen werden können. Für die Signierung und Verschlüsselung der Tokens kann die Bibliothek für die entsprechende Sprache von der jwt.io Seite bezogen werden. Bei dem Prototypen übernimmt Keycloak diese Arbeit.

Die Zuordnung von Session-ID zu Tokens hat den Vorteil, dass Zugriffstokens nicht direkt beim Client gespeichert werden und Angreifer beim Client nur Zugriff auf die ID haben. Bei der Erstellung der ID oder allgemein bei Sicherheitsbibliotheken sollte immer auf bestehende, geprüfte Lösungen zurückgegriffen werden. In unserem Prototyp verwenden wir hierzu die von Keycloak bereitgestellten Funktionen. Zudem ist die Session-ID durch

[5] https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/_content/m/m04/m04394.html?nn=6604968 (2014)

[6] https://tools.ietf.org/html/rfc7516 (2018)

bestimmten Hashing- und Kodierungsverfahren unlesbar, bzw. nicht nachvollziehbar. Hier nimmt uns Keycloak ebenfalls Arbeit ab.

Die Session-ID wird bei der Kommunikation mit verschiedenen Servern in den Cookies übertragen. Dabei werden gängige Mechanismen verwendet um diese abzusichern. Diese beinhalten das Setzen des *Secure*-Flags um den Cookie Transfer nur bei https zu erlauben und das Setzen des *HttpOnly*-Flags um den Zugriff der Cookies mit JavaScript zu unterbinden.

Des Weiteren wird die Sitzungsdauer auf eine gewisse Zeit beschränkt (z.B. 30 Minuten), damit das Erraten einer Session-ID fast unmöglich wird. Nach Ablaufen der Session wird diese zentral in der IAM Komponente gelöscht. Ebenfalls gelöscht werden soll eine Session bei auffälligem Verhalten, bzw. bei einem erkannten Missbrauch der Session. Hierzu werden Verfahren/Prozesse von KeyCloak bereitgestellt

Wenn Webanwendungen einen höheren Schutzbedarf benötigen, sollte ein zusätzliches Attribut zur Identifikation einer Session verwendet werden. Dies ist beispielsweise durch die zusätzliche Speicherung der IP-Adresse möglich.

3.4.2 OpenID Connect

Wie bereits angesprochen ist das OIDC Protokoll an manchen Stellen nicht spezifisch, beziehungsweise weist Lücken in der Standardisierung auf. Allerdings gibt es viele Sicherheitsmechanismen mit denen man von Anfang an Angriffspunkte schließen kann.
Ein paar dieser Empfehlungen betreffen den *redirect_uri* Parameter der Authentifizierungsanfragen . Zum einen ist es essentiell die redirect_uri zu validieren. Dies kann auch dadurch geschehen, dass man nur aus einer kleinen Anzahl an URIs wählen kann. Zum anderen sollte immer darauf geachtet werden TLS bei der *redirect_uri* zu verwenden, um eine sichere Verbindung aufzubauen.
Zudem sollte bei Authentifizierungsanfragen immer der state Parameter mitgesendet werden um CSRF Angriffe zu verhindern[7].
Auf der Login-Seite wird kein externes JavaScript (zum Beispiel Analytics, Werbung, ...) verwendet, um die Angriffsfläche zu reduzieren.
Hinzu kommt, dass bei den Tokens die gegebenen Attribute entsprechend interpretiert werden sollten. Dies bedeutet, den *iss*-claim zu überprüfen, ob das Token von einem bekannten Server erstellt wurde, oder dass Anwendungen bei abgelaufener Gültigkeit eines Tokens entsprechend die Session terminieren.

3.4.3 Container[8]

Um Dienste (und Angebote) flexibel zu Verfügung stellen zu können, werden sie in Containern verwaltet, die durch den jeweiligen Anbieter kontrolliert werden. Dependencies werden bei diesem Vorgehen häufig in weiteren Containern verwaltet, was den Vorteil der modularen Updatebarkeit der Dependencies mit sich bringt. Hierbei ist entscheidend,

[7] https://wiki.mozilla.org/Security/Guidelines/OpenID_connect
[8] https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/IT-Grundschutz-Modernisierung/BS_Container.pdf?__blob=publicationFile&v=4 (2018)

vertrauenswürdige Docker Registries zu verwenden. Für Keycloak steht zum Beispiel https://hub.docker.com/r/jboss/keycloak/ zur Verfügung.

3.4.4 Authentisierung[9]

Zusätzlich zu Benutzername und Passwort beinhaltet die Authentifizierung aller sicherheitsrelevanten Informationen einen zweiten Faktor, wie zum Beispiel TOTP oder push Token 2FA. Hierbei authentifiziert sich der Server mit Zertifikat und die Anzahl an Versuchen ist zum Schutz gegen Brute-Force-Angriffe limitiert. Des Weiteren sind die Tokens von der IDM-Komponente (in unserem Fall Keycloak) signiert.

3.4.5 Verschlüsselung[10]

Die Übertragung jedweder Informationen zwischen Client und Server (HTTPS), zwischen Servern (HTTPS), und zu Datenbanken (LDAPS) wird über TLS (Transport Layer Security) 1.2 (Dokumentation von Keycloak zur Einrichtung von TLS) abgesichert.
Alle in den Datenbanken gespeicherten Daten werden mit asymmetrischen Verschlüsselungsverfahren (z.B. RSA) verschlüsselt. Passwörter werden nicht im Klartext gespeichert, sondern nach Verarbeitung mit einer Hash-Funktion wie z.B. SHA512.

3.5 (Technologie-)Analyse

Unsere (technologischen) Entscheidungen, beispielsweise welche Frameworks, Libraries und Protokolle wir verwenden, werden im Folgenden begründet und mit ihren Alternativen verglichen.

3.5.1 Identity Provider

Der Nukleus-Dienst, durch welchen sich der Anwender beim Anbieter identifiziert und weitere Dienste verwenden kann, ist die IAM-Komponente. Durch sie wird auch die Session erstellt und läuft die 2-Faktor-Verifizierung. Wir haben die folgenden vier Frameworks für diese Zwecke gegenübergestellt.

	LinOTP	**Keycloak**	**DUO**	**ForgeRock**
Hersteller	KeyIdentity aus Deutschland	JBoss (Abteilung von Red Hat)	Duo Security	ForgeRock
Installa-tion	Dokumentation zur Installation und Management; recht hoher Aufwand, Konfiguration der	geschrieben in Java; Dokumentation für verschiedene Use Cases (bspw	umständliches verpflichtendes 3. Portal, das das Erstellen von Anwendungen an	Dokumentation ist sehr ausführlich

9 https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/_content/m/m04/m04456.html?nn=6604968 (2014)

10 https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/_content/m/m05/m05066.html?nn=6604968 (2014)

	Module (z.B. simplesamlphp) schwierig, aber dadurch sehr modular; Skalierung nicht in Doku enthalten	horizontale/vertikale Skalierung);	Bezahlvorgang koppelt, komplexes Management mehrerer Portale	
Integration	Admin-API vorhanden Login der User über API möglich	Admin-API für Automatisierung bspw. der Erstellung neuer Accounts	Admin-API	offen APIs
2FA	Unterstützt alle gängigen Tokens	TAN-Token via bspw. FreeOTP authenticator	Unterstützt alle gängigen Tokens	Unterstützt alle gängigen Tokens
Session (SSO)	mit Erweiterungen durch z.B. simplesamlphp möglich (kompliziertes Setup) OpenID Connect Schnittstelle muss man selber schreiben	eingebaut, verhältnismäßig einfache offizielle Integration von bspw. OpenID Connect bei Anwendern durch Plugins	eingebaut, OpenID Connect nur von Drittanbietern (Google, Microsoft) oder SAML	eingebaut
Open Source	ja	ja	nein	ja
Kosten	2€/Device/Monat	$12000/Jahr	$9/Nutzer/Monat	n\a

Den IdP *duo* haben wir erst einmal außen vorgelassen, da die Installation ähnlich aufwendig ist wie die von LinOTP. Allerdings muss man bei duo seine Anwendungen bei einem extra Admin-Portal gehostet bei *duo.com* anmelden. Das bringt den Nachteil mit sich, dass die Anwendungen auf einem externen Server kontrolliert, konfiguriert und monetarisiert werden können.
LinOTP schließen wir als IDP für den Prototypen aus, da die Konfiguration des Session Managements in Verbindung mit OIDC sehr komplex war (eigene Schnittstellen erforderlich). Daher fiel unsere Wahl auf Keycloak, da dieser IDP Session Management mit OIDC out of the box unterstützt und somit sofort im Prototyp testbar ist. Forgerock schauten wir uns nur kurz an. Diese Lösung sieht Keycloak vom Umfang der Funktionen sehr ähnlich aus.

3.5.2 OpenID Connect

In der Abwägung zwischen SAML und OpenID Connect bei den Schnittstellen zur IAM-Komponente entschieden wir uns für OIDC, da es ein neuerer Standard ist, der sich über die letzten Jahre in der Industrie bei großen Firmen (Microsoft, Google, etc.) etabliert hat. Ein Problem, welches bei *stateless*-Token-basierten Systemen besteht, ist das des

Single Sign Outs. Das bedeutet, ein gültiges Token kann im Falle eines Diebstahls dazu benutzt werden mit Services zu interagieren. Dieses Problem kann gelöst werden, indem man die Gültigkeit eines Tokens durch ein Ablaufdatum begrenzt oder durch das serverseitige Speichern und Sperren des Tokens. Darüber hinaus haben sowohl SAML als auch OpenID Connect den Vorteil des zentralen Logouts (Single Sign Outs).

	SAML 2.0	**OpenID Connect**
Hersteller	OASIS, 2001	OpenID Foundation, 2014
Allgemein	Offener Standard für Autorisierung und Authentifizierung	Offener Standard für Autorisierung (OAuth2 als Basis) und Authentifizierung
Datenformat	XML	JSON
BSI Akzeptanz	Standard	noch nicht geprüft

OIDC hat einige bekannte Sicherheitslücken, gegen die man sich allerdings schützen kann[11]. Des Weiteren ist OIDC einfach aufzusetzen (z. B. mit Keycloak), vielseitig einsetzbar bei allen Arten von Anwendungen und aufgrund des mittlerweile sehr weit verbreiteten Standards (umgesetzt von z.B. Google[12], Amazon[13],...) zukunftstauglicher als SAML. Deshalb empfehlen wir OIDC und bis OIDC vom BSI zugelassen wird SAML.

3.5.3 REST

Wenn nicht anders spezifiziert, verwenden wir für den Prototypen immer REST-Schnittstellen, so wie im Abschnitt API beschrieben. Eine Alternative zu REST ist SOAP. Im Folgenden werden sie gegenübergestellt.

REST	**SOAP**
● einfache Anbindung an Fremdsysteme ● einfach dekomposition von Diensten ● Skalierbar weil State mitgeschickt wird ● Ressourcen sind einzeln adressierbar ● Schrittweise Evolution möglich ● ermöglicht caching ● erlaubt viele Datenformate (JSON, XML,...)	● geeignet für Dateien ● hohe Sicherheit durch Unterstützung für WS-Security ● Unterstützung von anderen WS-*-Spezifikationen (AtomicTransaction für ACID, Reliability Messaging für Kommunikationsbestätigung) ● alle Anfragen gehen auf denselben Endpoint ● starrer Rahmen für die Entwicklung vorgegeben

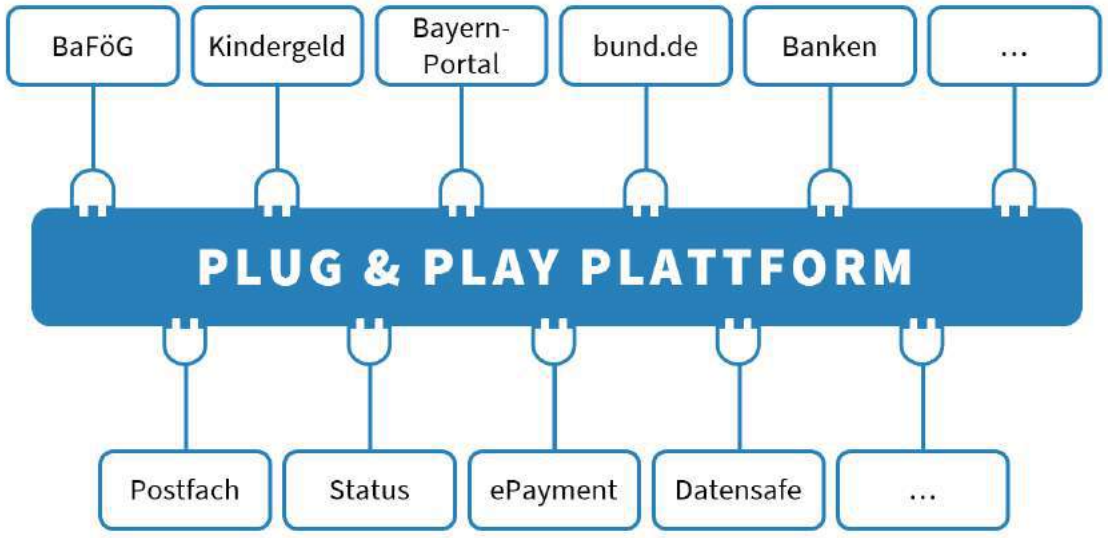
[11] http://nds.rub.de/media/ei/veroeffentlichungen/2017/01/30/oidc-security.pdf (2018)
[12] https://developers.google.com/identity/protocols/OpenIDConnect (2018)
[13] https://aws.amazon.com/de/blogs/aws/openid-connect-support/ (2014)

REST ist modularer als SOAP, wobei SOAP für den Austausch großer Daten in Erwägung gezogen werden könnte. Außerdem bietet SOAP mehr Sicherheit durch die sogenannten WS-*-Spezifikationen. Über den Prototypen hinaus legen wir deshalb für sicherheitskritische Dienste, wie zum Beispiel die Datenübertragung, nahe SOAP in Erwägung zu ziehen.

3.6 Einbindung bei Anbietern

Heutzutage wird der Föderalismus in Deutschland bei der Bereitstellung und Synchronisation von Diensten und Daten als Erschwernis empfunden. Einzelne Portalverbunde und Kommunen fangen an, zu kooperieren. Allerdings werden keine deutschlandweit gültigen Standards für Schnittstellen bereitgestellt. Dies hemmt die Bereitschaft von Kommunen und Bundesländern nachhaltig in die digitale Infrastruktur zu investieren.

Die *Plug&Play Plattform (PPP)* von Tech4Germany betrachtet den Föderalismus stattdessen als Innovationstreiber. Kommunen, Portalverbunde, Anbieter einzelner Fachverfahren sowie Drittanbieter (Banken, Universitäten, Versicherungen, etc.) verbinden sich über den Standard mit der PPP.



Hat eine Kommune ein Fachverfahren mit dem Standard digitalisiert, müssen somit andere Kommunen und Portale dieses nicht neu für sich entwickeln, sondern integrieren es bei sich ähnlich wie man Apps aus einem App Store in seinem System integriert.

Genauso werden Mehrwertdienste und weitere Funktionen mit Plug&Play beim Anbieter integriert, wenn dieser sie zur Verfügung stellen will.

Durch eine derartige Plattform werden doppelte Entwicklungskosten gespart, es entsteht eine Art Wettbewerb, der als Chance für Innovation im e-Government zu betrachten ist, und die Angebotstransparenz gibt einen Überblick über gut laufende und zu verbessernde Angebote und Dienste.

3.7 Ausblick

Nicht vollständig in aller Tiefe bearbeiteten Themen, die in Zukunft im Zusammenhang mit diesem SAD genauer betrachtet werden sollten, sind unter anderem die folgenden Fragen.

- Wie wird die Protokollierung der Zugriffe realisiert?
- Gibt es eine Historisierung der Änderungen für die Nutzer-Attribute?
- Wie könnte das Thema Provisionierung von Identitäten aussehen?

3.8 Glossar

2FA	Authentisierung durch weiteren Faktor (z.B. PIN, TAN)
API	(offene) Schnittstelle einer Software oder eines Systems
Authentifizierung	Prüfung der behaupteten Authentisierung
Authentisierung	Prüfung der Identität
Autorisierung	Prüfung der Rechte
BSI	Bundesamt für Sicherheit und Informationstechnik
CLI	command-line-interface
Container	Lösung für skalierbare Modularisierung, z. B. mit “Docker”
DB	Datenbank System
eIDAS	Verordnung über elektronische Identifizierung und Vertrauensdienste
Fachverfahren	Angebot (Formular) einer Verwaltung, wenn diese als Anbieter agiert
Identity Provider (IdP)	Anbindung zu Nutzer Datenbank/ Verzeichnis und dessen Management
nPA	Neuer Personalausweis mit Online- Funktion
OpenID Connect (OIDC)	Authentifizierungsschicht basierend auf OAuth2
REST	Architekturstil zur Beschreibung vom Austausch von Daten zwischen Systemen
SAML	XML-basiertes Datenformat zum Austausch von Authentifizierungs- und Autorisierungsinformationen
Service Provider (SP)	(Dritt)Anbieter eines Dienstes (z.B. Fachverfahren oder Portal)
Session	stehende Verbindung eines Clients mit einem Server
Single-Sign-On (SSO)	Eine Session, die ein verkürztes Einloggen in weitere mit dem Dienst verknüpfte Angebote ermöglicht
SOAP	Netzwerkprotokoll zum Austausch von Daten zwischen Systemen
TOTP	Time-based One-Time Password