

Fachhochschule Bielefeld
Fachbereich Campus Minden

Studiengang M. Sc. Informatik

Evaluation von OpenID Connect und SAML

Hausarbeit

Sommersemester 2022

Vorgelegt von: Rafael Berger

Matrikelnummer: 1181289

Abgabe am: 30.06.2022

Prüfer: Prof. Dr. rer. nat. Jörg Brunsmann

Inhaltsverzeichnis

1	Unterschied Autorisierung und Authentifizierung.....	- 1 -
2	OAuth 2.0	- 2 -
2.1	Tokens	- 2 -
2.2	Rollen	- 3 -
2.3	Autorisierungsabläufe.....	- 3 -
3	OpenID Connect.....	- 4 -
3.1	JSON Web Token.....	- 4 -
3.2	ID-Token.....	- 6 -
3.3	Authentifizierungsablauf.....	- 7 -
4	SAML	- 9 -
4.1	Assertion	- 9 -
4.2	Rollen	- 9 -
4.3	Allgemeine Begriffe.....	- 10 -
4.4	Authentifizierungsablauf.....	- 11 -
4.5	Keycloak	- 12 -
5	Implementierung	- 13 -
5.1	Vorgehensweise.....	- 13 -
5.2	Metriken mit Keycloak	- 13 -
5.3	Services	- 14 -
5.4	Keycloak	- 15 -
5.5	Backend-Server	- 19 -
6	Ergebnisse	- 26 -
7	Literaturverzeichnis	- 30 -

Abbildungsverzeichnis

Abbildung 1: JSON Web Token - Header [7]	- 5 -
Abbildung 2: Authentifizierungsablauf OIDC Part 1 [8]	- 7 -
Abbildung 3: Authentifizierungsablauf OIDC Part 2 [8]	- 8 -
Abbildung 4: Authentifizierungsablauf SAML	- 11 -
Abbildung 5: Keycloak Realm Einstellungen	- 15 -
Abbildung 6: Keycloak Clients	- 16 -
Abbildung 7: Keycloak OpenIdTestClient	- 16 -
Abbildung 8: Keycloak Rollen	- 17 -
Abbildung 9: Keycloak Admin Rolle	- 17 -
Abbildung 10: Keycloak Adminuser Rollen	- 18 -
Abbildung 11: Backend Keycloak Konfiguration	- 19 -
Abbildung 12: Backend Server	- 20 -
Abbildung 13: Backend Routen	- 20 -
Abbildung 14: Backend get Access-Token	- 21 -
Abbildung 15: Ausgabe Token OIDC	- 22 -
Abbildung 16: Backend Datenabfrage	- 23 -
Abbildung 17: Response Backend	- 24 -
Abbildung 18: Response SAML	- 25 -
Abbildung 19: Hilfsskript für die Auswertung	- 26 -
Abbildung 20: Diagramm der Anmeldezeit OIDC	- 27 -
Abbildung 21: Keycloak Identitätsanbieter	- 29 -

1 Unterschied Autorisierung und Authentifizierung

Autorisierung

Bei einer Autorisierung geht es darum, dass eine bestimmte Ressource vor unerlaubten Zugriff geschützt wird. Eine Ressource kann dabei bestimmte Informationen sein oder der Zugriff auf Webseiten. Ein Nutzer oder ein Service fragt somit an, ob dieser auf die Ressource zugreifen darf und somit die Rechte dafür besitzt. [1] Eine Analogie dazu ist eine Schlüsselkarte um Türen oder Tore zu öffnen. Beim Vorlegen dieser Schlüsselkarte wird darauf geprüft, ob Zugriff zu diesen Bereich gewährt werden darf oder nicht. Falls der Zugriff erlaubt wird, öffnet sich die Tür oder das Tor und die Person kann hineingehen. In diesem Fall wird nur kontrolliert, ob die vorgelegte Schlüsselkarte die entsprechenden Befugnisse besitzt, um den abgeschlossenen Bereich zu öffnen und die Person die die Schlüsselkarte verwendet. Wenn jedoch überprüft werden soll, dass die vorzugebene Person sich auch um die echte Person handelt, kommt die Authentifizierung zu trage.

Authentifizierung

Bei der Authentifizierung spielt die Authentisierung eine wichtige Rolle. Dabei geht es um den Nachweis, dass es sich wirklich um den jeweiligen Nutzer oder Service handelt. Bei uns Menschen ist dies der Personalausweis. Dieser belegt das wir wirklich die Person sind, für die wir uns ausgeben und wird von einem zentralen Identitätsanbieter, in diesem Fall das Bürgerbüro, ausgestellt. Eine unabhängige Person kann somit kontrollieren, ob die Identität stimmt. Die Authentifizierung ist das Verfahren, dass die Identität überprüft. [1] Im Gegensatz zu der Analogie vom Beispiel der Autorisierung wird bei der Authentifizierung überprüft, ob es sich bei dem jeweiligen Nutzer der Schlüsselkarte auch um die exakte Person handelt. Dies kann zum Beispiel mittels einer Kontrolle des Personalausweises durchgeführt werden.

2 OAuth 2.0

OAuth steht für „Open Authorization“ und ist ein offenes Standardprotokoll. Dieses wird verwendet, um eine sichere API mittels einer Autorisierung zu gewährleisten. Dadurch kann die API vor unbefugten Personen oder Services abgesichert werden. Bei OAuth 2.0 handelt es sich um eine komplette Überarbeitung der Vorgängerversion, die 2012 entwickelt wurde und sich als neuer Standard von OAuth etabliert hat. Wichtig hierbei ist, dass die neuere Version nicht mehr mit der älteren Version kompatibel ist. Mit OAuth 2.0 sollen viele vorhandenen Schwachstellen behoben worden sein. Die Autorisierung durch OAuth 2.0 findet mittels eines Access-Tokens statt. [2]

2.1 Tokens

Access-Token

Ein Access-Token ist notwendig für die Autorisierung. Dabei handelt es sich um einen kodierten String, welcher nach keinen bestimmten Format aufgebaut ist. Oftmals werden Bearer-Tokens als Access-Tokens verwendet. Für ein Access-Token gibt es verschiedene Eigenschaften die erfüllt werden müssen. Zum einen darf die Interpretierung des Access-Tokens nicht durch den Client durchgeführt werden, sondern darf nur für die Anfragen an den Ressourcenserver verwendet werden. Weiterhin dürfen keine persönlichen Informationen über den Nutzer enthalten sein. [3]

Refresh-Token

In der Regel läuft das Access-Token nach einer bestimmten Zeit ab. Damit der Nutzer nicht erneut sich autorisieren muss, wird zusätzlich ein Refresh-Token erstellt. Durch dieses zusätzliche Token wird das Access-Token automatisch in regelmäßigen Abständen aktualisiert. Die Dauer für die Gültigkeit des Access-Tokens und somit die Häufigkeit der Aktualisierung hängt von der Anwendung ab. Ähnlich wie beim Access-Tokens handelt es sich hierbei um einen kodierten String, der ebenfalls nach keinen bestimmten Format aufgebaut ist. [4]

Code-Token

Bei dem Code-Token handelt es sich um einen besonderen Token, der nur bei speziellen Autorisierungsabläufe verwendet wird. Ein Beispiel dafür ist der Authorization Code Grant. Dabei wird der Code-Token verwendet, um initial ein Access-Token und ein Refresh-Token zu erhalten. [5]

2.2 Rollen

Ressourcen-Besitzer

Ein Ressourcen-Besitzer kann ein System oder ein Service sein, dass andere Nutzer oder Services den Zugriff auf die geschützte Ressource gewährt. [2]

Ressourcen-Server

Der Ressourcen-Server verwaltet die geschützte Ressource und überprüft das Access-Token auf Gültigkeit bevor dieser Zugriff auf diese ermöglicht. [2]

Client

Bei einem Client handelt es sich um eine Anwendung, die auf die geschützte Ressource vom Ressourcen-Besitzer zugreifen möchte. [2]

Autorisierungsserver

Der Autorisierungsserver ist ein wichtiger Bestandteil, da dieser das Access-Token für den Client ausstellt. Weiterhin hat der die Aufgabe den Ressourcen-Besitzer zu authentifizieren. [2]

2.3 Autorisierungsabläufe

Es gibt verschiedene Arten von Abläufe die mit OAuth 2.0 verwendet werden können. Dabei handelt es sich um den **Authorization Code Grant**, den **Implicit Grant**, den **Client Credentials Grant** und der **Resource Owner Password Credentials Grant**. Jeder dieser Autorisierungsabläufe verwendet die vorher genannten Rollen und erfüllen den gleichen Zweck – den Client zu autorisieren. [2]

3 OpenID Connect

OpenID Connect, kurz OIDC, führt neben einer Autorisierung ebenfalls eine Authentifizierung durch. Dieser Standard baut auf OAuth 2.0 auf und ist eine Abstraktionsebene über OAuth 2.0. Neben den vorhandenen Tokens von OAuth 2.0 wird ebenfalls ein ID-Token erstellt. Dieses enthält die persönlichen Informationen über den Benutzer und dient zur Authentifizierung von diesem. Für das Access-Token, das Refresh-Token und ID-Token schreibt dieser Standard vor, dass es sich dabei um JSON Web Tokens (JWT) handeln muss. [6]

3.1 JSON Web Token

Bei einem JSON Web Token handelt es sich um einen Standard (RFC 7519), der zur sicheren Übermittlung von Daten als JSON-Objekt dient. Die Besonderheit bei JSON Web Tokens liegt darin, dass diese durch eine digitale Signierung der gesendeten Daten gegenüber Veränderungen geschützt sind. Dadurch können die Daten auf Echtheit überprüft werden und es kann sichergestellt werden, dass diese während der Übertragung nicht verändert worden sind. Wichtig hierbei ist, dass die Daten innerhalb eines JSON Web Tokens standardmäßig nicht verschlüsselt werden, aber die Möglichkeit besteht dieses durchzuführen. Durch diesen hohen Maß an Sicherheit ist diese Methode geeignet für die Autorisierung von Nutzern und den Informationsaustausch im Internet. Ein Beispiel für die Verwendung von JSON Web Tokens ist die Authentifizierungsart „Single Sign On“. Dabei bekommt der Nutzer die Möglichkeit durch die einmalige Anmeldung den Zugriff auf verschiedene Services zu erhalten. Ein JSON Web Token ist in drei Komponenten aufgeteilt, die jeweils mit einem Punkt getrennt werden – zum Beispiel xxx.yyy.zzz. [7]

Header

Bei dem ersten Teil eines JSON Web Tokens handelt es sich um den Header. Dieser beinhaltet den verwendeten Signaturalgorithmus und um welchen Typ es sich handelt. [7]

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

Abbildung 1: JSON Web Token - Header [7]

In Abbildung 1 wird ein beispielhafter Header von einem JSON Web Token abgebildet, welcher den „HS 256“ als Signaturalgorithmus verwendet und vom Typ „JWT“ ist. [7]

Payload

Im Payload sind alle relevanten Informationen enthalten. Im Falle eines ID-Tokens kann der Nutzernamen und/oder die E-Mail-Adresse angegeben werden. Bei einem Access-Token können die entsprechenden Rechte des Nutzers vorhanden sein, um auf verschiedene Ressourcen zuzugreifen. Die enthaltenen Behauptungen werden Claims genannt. Dabei gibt es drei verschiedene Typen von Claims – Registrierte, Öffentliche und Private. Unter den **Registrierten Claims** sind vordefinierte Behauptungen zu verstehen, die nicht erforderlich sind, aber im allgemeinen empfohlen werden. Das Einhalten dieser Art von Claims hat den Vorteil, dass JSON Web Tokens von Drittanbietern einfacher verstanden werden. Die am Häufigsten verwendeten Registrierten Claims sind iss, exp, sub und aud. Iss sagt etwas über den Aussteller des JSON Web Tokens aus. Bei exp wird die Ablaufzeit angegeben und somit die Gültigkeitsdauer. In sub wird der Betreff und unter auf die Zielgruppe angegeben. Die nächste Art von Claims sind die **öffentlichen Claims**. Diese können frei ausgewählt werden, sollten jedoch in der IANA JSON Web Token Registry enthalten sein. Wichtig hierbei ist, dass sich bei der Verwendung einer der dort vorhandenen Claims auch an die Bedeutung gehalten wird, um mögliche Überschneidungen zu vermeiden. Zum Schluss gibt es **privaten Claims**. Für diese Claims gibt es keine Vorgaben und können frei definiert werden. [7]

Signatur

Der Header, die Payload und ein Secret Key werden zusammen gehashed und dienen somit als Signatur. Der generierte Hashwert kann aufgrund des Secret Keys nicht durch eine geänderte Payload neu generiert werden und dabei gültig sein. Eine Veränderung des Headers oder des Payloads kann durch die nicht Übereinstimmung des originalen Hashwertes erkannt werden. [7]

3.2 ID-Token

Wie zu Beginn erwähnt, handelt es sich bei dem ID-Token um eine Besonderheit im Gegensatz zu OAuth 2.0. Innerhalb dieses Tokens sind alle nutzerspezifischen Informationen enthalten, die für die Authentifizierung notwendig sind. Da es sich hierbei um ein JSON Web Token handelt, sind einige Claims seitens OpenID Connect vorgeschrieben. Bei den erforderlichen Claims handelt es sich um iss, sub, aud, exp und iat. Weitere Claims wie acr, amr und azp sind optional und können je nach Bedarf hinzugefügt werden. Abhängig der Implementierung können zwei weitere Claims, auth_time und nonce, erforderlich sein.

3.3 Authentifizierungsablauf

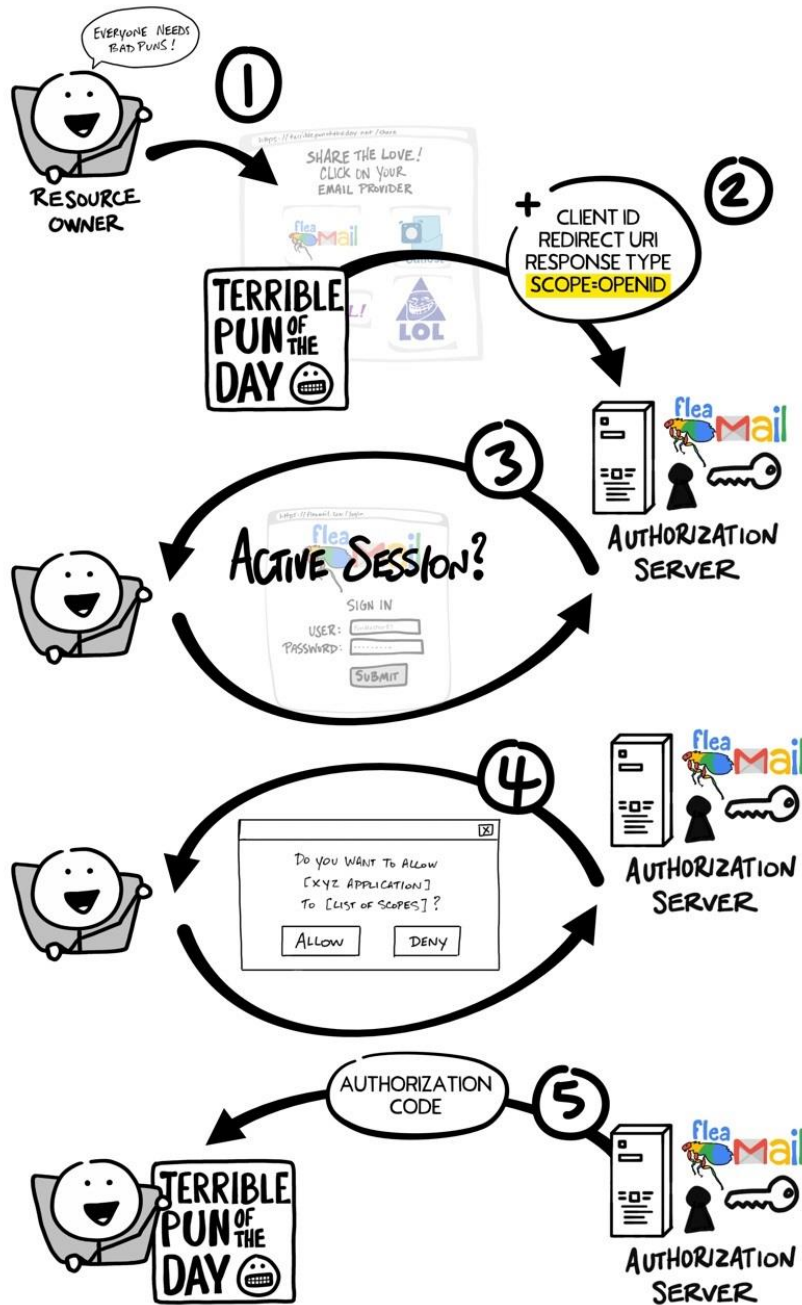


Abbildung 2: Authentifizierungsablauf OIDC Part 1 [8]

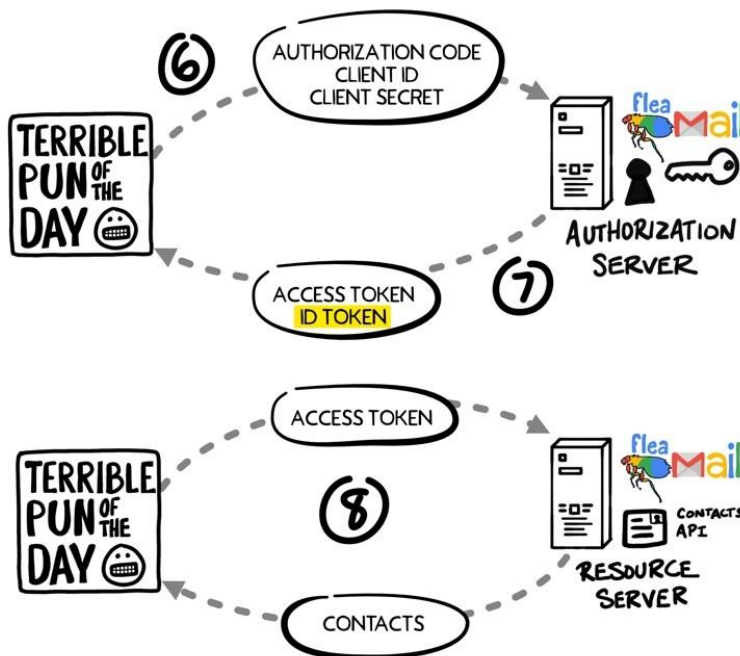


Abbildung 3: Authentifizierungsablauf OIDC Part 2 [8]

In Abbildung 2 und Abbildung 3 wird ein Beispiel für einen Authentifizierungsablauf mittels OpenID Connect dargestellt. In Schritt 1 versucht der Ressourcen-Besitzer eine Nachricht über seinen E-Mail-Anbieter zu versenden. Im nächsten Schritt leitet die Webseite die notwendigen Informationen für den Beginn des Authentifizierungsprozesses an den entsprechenden Autorisierungsserver weiter. Danach wird vom Autorisierungsserver überprüft, ob der Nutzer eine aktive Sitzung mit dem gleichen Autorisierungsserver hat und somit schon angemeldet ist. Wenn dies nicht der Fall ist, muss der Nutzer sich bei dem E-Mail-Anbieter mit seinen entsprechenden Anmeldeinformationen anmelden. In Schritt 4 wird der Nutzer dazu aufgefordert die Berechtigung für die Ausführung der gewünschten Aktivität zu erteilen. Daraufhin sendet der Autorisierungsserver den Code-Token an die Webseite. Nach Erhalt des Code-Tokens sendet die Webseite in Schritt 6 den Code-Token, die Client ID und das Client Secret zurück zu dem Autorisierungsserver. Daraufhin generiert der Autorisierungsserver das Access-Token und das ID-Token und sendet beide zu der entsprechenden Webseite zurück. Abschließend kann die Webseite mit dem erhaltenen Access-Token sich bei dem Ressourcen-Server, in diesem Fall der E-Mail-Anbieter, autorisieren und erhält die gewünschten Informationen. [8]

4 SAML

SAML steht für Security Assertion Markup Language und ist wie OpenID Connect ein offener Standard, der die Aufgabe hat, Nutzer oder Services zu authentifizieren. SAML ist deutlich älter als OpenID Connect, da es bereits 2005 von der OASIS Consortium als Version 2.0 veröffentlicht wurde. Dank SAML besteht die Möglichkeit Single Sign-On zu verwenden. Das heißt, dass ein Benutzer nur ein Account benötigt, um Zugriff auf verschiedene Dienste zu erhalten. Das hat den Vorteil, dass keine umfangreiche Nutzerverwaltung für jeden unterschiedlich Dienst benötigt wird, sondern zentral nur eine Nutzerverwaltung notwendig ist. Weiterhin muss sich der Nutzer nur einmal Anmelden und kann alle verbundenen Dienste, ohne erneute Anmeldung, nutzen. [9]

4.1 Assertion

Bei SAML wird für die Authentifizierung sogenannte „Assertions“ verwendet. Dabei handelt es sich um ein XML-Dokument, welches unterschiedliche Informationen enthalten kann und in drei unterschiedliche Arten unterteilt werden kann. Zum einen gibt es die **Authentifizierungs-Assertion**. Diese wird verwendet, um seine Anmeldung zu beweisen und diese enthält Angaben über den Zeitpunkt der Anmeldung. Zusätzlich sind Informationen darüber enthalten, welches Verfahren für die Authentifizierung durchgeführt wurde. In einer **Attributs-Assertion** sind Informationen und Daten über den jeweiligen Nutzer enthalten. Zum Schluss gibt es noch die **Autorisierungsentscheidung**. Damit werden die Zugriffsrechte des jeweiligen Nutzers angegeben. [9]

4.2 Rollen

Bei SAML gibt es drei wesentliche Rollen. Der **Benutzer** ist der Anwender des Dienstes. Als nächstes sind zwei verschiedene Arten von SAML-Provider vorhanden. Dabei handelt es sich um ein System, welches die Aufgabe hat den Benutzer dabei zu unterstützen den jeweiligen Dienst zu verwenden. Der **Service Provider** ist die Anwendung für die die Authentifizierung durchgeführt werden soll. Der **Identity Provider** wiederum ist das System, welches die Authentifizierung durchführt. Dafür enthält es die verschiedenen Anmeldeinformationen über die Benutzer. Diese Anmeldeinformationen sind abhängig von der Anwendung und der jeweiligen Implementierung. Es besteht die Möglichkeit das Informationen von dem Benutzer wie der Vorname, Nachname und Adresse enthalten sind. Aber auch die E-Mail-Adresse ist eine häufige Information, die dem Identity Provider zur Verfügung stehen. [9, 10]

4.3 Allgemeine Begriffe

Darüber hinaus kommen zusätzliche Begriffe bei SAML auf. Die SAML-Request ist eine automatische vom Service Provider generierte Anfrage, die für die Authentifizierung notwendig ist. Die SAML Response wird von dem Identity Provider generiert. Sie enthält alle wichtigen Informationen über den Benutzer. [10]

4.4 Authentifizierungsablauf

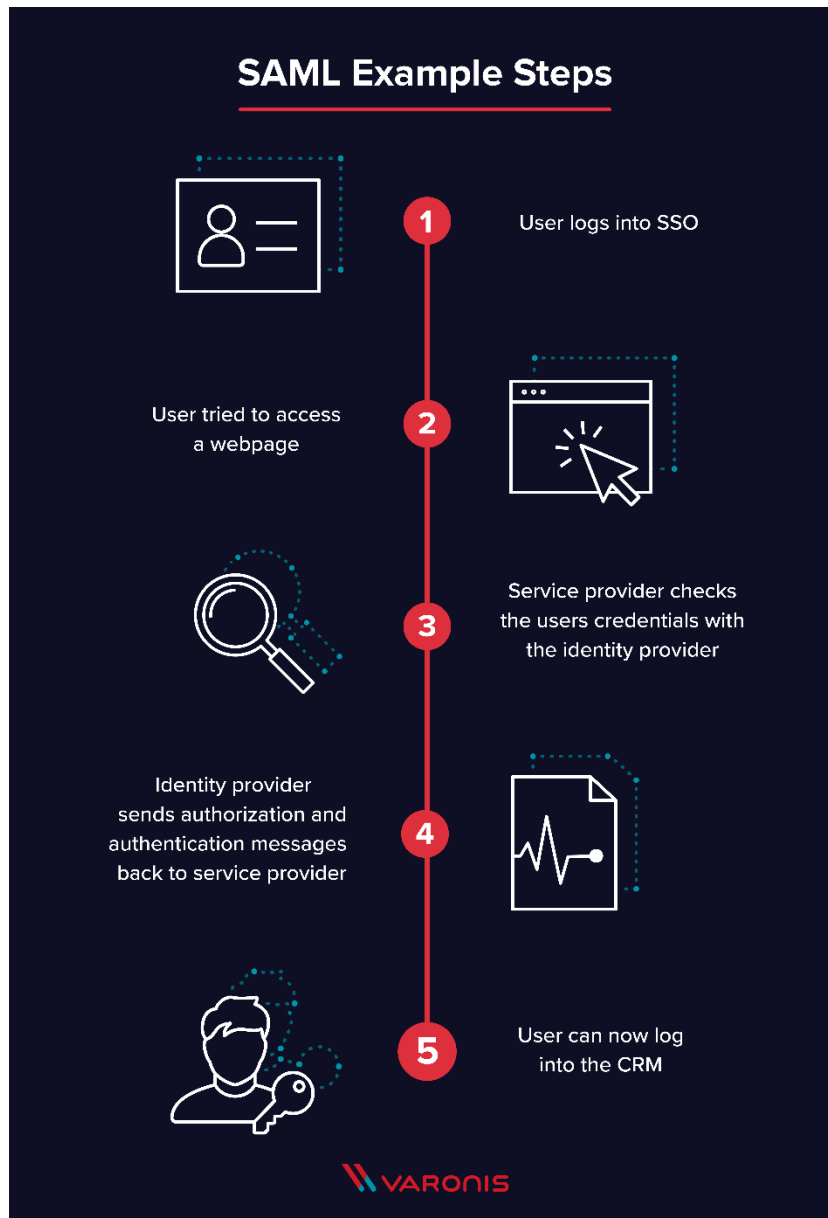


Abbildung 4: Authentifizierungsablauf SAML

In Abbildung 4 ist ein beispielhafter Authentifizierungsablauf mit SAML dargestellt. Zu Beginn meldet sich der Benutzer über SSO in das System an. Als nächstes versucht der Benutzer Zugriff auf eine geschützte Webseite zu erhalten. Der Webseitenbetreiber sendet die vom Benutzer angegebenen Anmeldedaten an den Identity Provider weiter. Dieser sendet daraufhin die

Autorisierungs- und Authentifizierungsnachrichten wieder zurück zu dem Webseitenbetreiber. Daraufhin ist der Benutzer auf der gewünschten Webseite angemeldet.

4.5 Keycloak

Keycloak ist eine Open Source Software die 2013 veröffentlicht wurde. Dabei handelt es sich um ein Identity- und Access Management, welches viele Features mitliefert, wodurch der Nutzer kaum eigene Konfigurationen benötigt. Trotzdem lassen sich bestimmte Einstellungen vornehmen, damit diese für die eigene Anwendung konfiguriert ist. Mithilfe von Keycloak lässt sich Single Sign-on implementieren. Darüber hinaus unterstützt Keycloak sowohl OAuth 2.0 für die Autorisierung und somit auch OpenID Connect für eine zusätzliche Authentifizierung als auch das Protokoll SAML. [11]

5 Implementierung

5.1 Vorgehensweise

Für die Autorisierung und Authentifizierung werden OpenID Connect und SAML genutzt und miteinander verglichen. Neben den unterschiedlichen Authentifizierungsabläufe werden die möglichen Anwendungsfälle ausgearbeitet. Für eine genauere Evaluierung ist es notwendig, neben der theoretischen Einarbeitung, diese Verfahren zu implementieren und auszuführen. Dafür wird ein bestimmter Authentifizierungsablauf ausgewählt, der sowohl mit OIDC als auch mit SAML möglich ist. Dementsprechend werden kleine Prototypen von Services implementiert, die diese Authentifizierung durchführen.

5.2 Metriken mit Keycloak

Für die Evaluierung werden verschiedene Metriken aufgestellt, die zum Schluss miteinander verglichen werden können, um ein gesamtes Fazit zu schließen. Die aufgestellten Metriken konzentrieren sich auf die Implementierung von OIDC und SAML.

Anmeldezeit

Bei dieser Metrik wird die Zeit gemessen wie lange die Authentifizierung dauert. Das bedeutet, wie schnell das Authentifizierungstoken generiert wird und eine Antwort vom Server mit den gewünschten Daten erhalten wird.

Session Parameter

Unter Session Parameter sind Einstellungsmöglichkeiten bezüglich der Konfiguration von den Tokens, sowie Angaben über die Session zu verstehen.

Developer Experience

Neben dem zeitlichen Messwert und der Konfiguration über die aktive Nutzung ist es ebenfalls wichtig zu erfassen wie hoch der Konfigurationsaufwand ist, damit sowohl OpenID Connect als auch SAML eingerichtet und funktionsfähig ist.

Signaturalgorithmus

Abhängig des Einsatzgebietes kann es wichtig sein, dass ein bestimmter Signaturalgorithmus ausgewählt wird, der den jeweiligen Richtlinien des Unternehmens entspricht.

Unterstützte Programmiersprachen

Die Authentifizierung findet überwiegend für Webservices statt, jedoch nicht ausschließlich. Daher kann es relevant sein, welche Programmiersprache verwendet werden muss für die Wahl der Authentifizierungsmöglichkeit.

Anzahl möglicher Identitätsanbieter

Für die Wahl zwischen OpenID Connect und SAML kann ein wichtiger relevanter Punkt die Auswahl der möglichen Identitätsanbieter sein.

5.3 Services

Für die Authentifizierung sowohl für OpenID Connect als auch SAML wird Keycloak als Identitätsanbieter verwendet. Nach der Authentifizierung mittels Keycloak wird auf eine geschützte Resource von einem Backend-Server der in Node.js geschrieben ist zugegriffen. Dieser ist verbunden mit einer MongoDB als Datenbank, wo die relevanten Daten gespeichert sind. MongoDB ist eine nicht relationale Datenbank, wodurch sie sich von anderen Datenbanken wie MySQL stark unterscheidet und dadurch kein Datenbankschema besitzt. Sie ist speziell geeignet für eine große Menge an Daten, da sie sowohl horizontal als auch vertikal sehr gut skalierbar ist. Zusätzlich wird häufig eine MongoDB bei Node.js Projekten verwendet, da die zwei bekannten Architekturen MEAN und MERN beide eine MongoDB beinhalten. Die Zugriffszeit ist relativ niedrig dank schnellen Abfragen, wo kaum eigener Code notwendig ist. Ein weiterer Vorteil liegt darin, dass es sich um ein Open Source Projekt handelt, wodurch keine zusätzlichen Kosten für Lizenzen aufkommen. [12] In dieser Arbeit wird kein eigener Service für den Frontend verwendet, da die Anmeldedaten des Nutzers direkt im Code hinterlegt ist. Das hat den Grund, da für die Evaluation nur die Zeit gemessen wird wie lange die Authentifizierung und die Dauert er Antwort vom Server mit den gewünschten Daten dauert. Außerdem gibt es keine Unterschied zwischen der Anfrage von Frontend mit OpenID Connect und SAML, da er nach der Anfrage die eigentliche Authentifizierung stattfindet.

5.4 Keycloak

Keycloak wird in diesem Projekt in einem Docker-Container gestartet. Das hat den Vorteil, dass Dafür wird folgender Befehl in der Konsole angegeben, womit Keycloak gestartet und zum Teil vorkonfiguriert wird:

```
docker run -p 8080:8080 -p 9990:9990 -e KEYCLOAK_USER=admin -e KEYCLOAK_PASSWORD=123456 jboss/keycloak
```

Durch diesen Befehl läuft Keycloak in einem Docker-Container auf Port 8080. Durch die Angabe von KEYCLOAK_USER und KEYCLOAK_PASSWORD wird initial ein Admin für die Administratorenkonsole registriert. Der letzte Teil des Befehl ist das verwendete Image für Docker. Nachdem Keycloak erfolgreich gestartet ist, kann die Adresse localhost:8080 aufgerufen werden. Nach der Anmeldung wird die Startseite von Keycloak angezeigt.

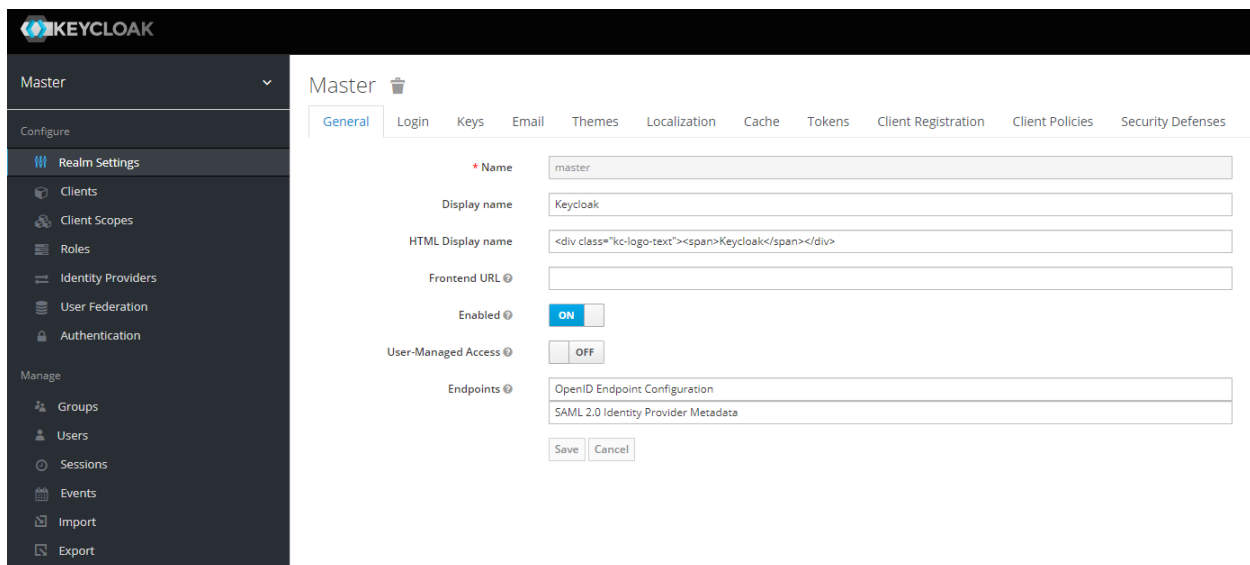



Abbildung 5: Keycloak Realm Einstellungen



Auf der dieser Seite werden Informationen zu dem verwendeten Realm, wie in Abbildung 5 zu erkennen, angezeigt. Damit OpenID Connect und SAML verwendet werden können, müssen bei den Endpoints beide angegeben werden.


Client ID	Enabled	Base URL
account	True	http://localhost:8080/auth/realms/master/account/
account-console	True	http://localhost:8080/auth/realms/master/account/
admin-cli	True	Not defined
broker	True	Not defined
master-realm	True	Not defined
openIdTestClient	True	Not defined
samlTestClient	True	http://127.0.0.1:8080/
security-admin-console	True	http://localhost:8080/auth/admin/master/console/


Abbildung 6: Keycloak Clients


Als nächsten Schritt müssen die beiden Clients für OpenID Connect und SAML erstellt werden. In Abbildung 6 sind beide Clients unter der Client ID „openIdTestClient“ und „samlTestClient“ abgebildet. Damit die jeweiligen Clients auch die unterschiedlichen Protokolle unterstützen, müssen diese konfiguriert werden.


OpenIdTestClient 


Settings Keys Roles Client Scopes  Mappers 


Client ID  openIdTestClient


Name 


Description 


Enabled  ☒ ON


Always Display in Console  ☐ OFF


Consent Required  ☐ OFF


Login Theme 

Client Protocol  openid-connect

Access Type  public

Standard Flow Enabled  ☒ ON

Implicit Flow Enabled  ☐ OFF

Direct Access Grants Enabled  ☒ ON


OAuth 2.0 Device Authorization Grant Enabled  ☐ OFF


Abbildung 7: Keycloak OpenIdTestClient

Für die Einstellung des zu verwendeten Protokolls ist die Konfiguration in Abbildung 7 dargestellt. Unter „Client Protocol“ kann das Protokoll zwischen „openid-connect“ und „saml“ für den jeweiligen Client eingestellt werden. Die restlichen Einstellungen sind Teil der Standardkonfiguration und müssen für diese Arbeit nicht weiter geändert werden.

Role Name	Composite
admin	True
create-realm	False
default-roles-master	True
offline_access	False
uma_authorization	False
user	False

Abbildung 8: Keycloak Rollen


Standardmäßig sind einige Rollen von Keycloak bereit erstellt worden, die nach Belieben konfiguriert werden können. In Abbildung 8 wurde eine zusätzliche Rolle mit dem „user“ erstellt. Die Rollen sind für die Authentifizierung gegenüber den Server nützlich. Damit können bestimmte Rechte zugewiesen werden, um auf bestimmte Daten zugreifen zu dürfen.

Admin 

Details Attributes Users in Role

Role Name: admin

Description: \${role_admin}

Composite Roles  ☒ ON

Save Cancel

▼ Composite Roles



Realm Roles	Available Roles 	Associated Roles 
	default-roles-master offline_access uma_authorization	create-realm user
	Add selected »	« Remove selected

Abbildung 9: Keycloak Admin Rolle

In Abbildung 9 wird der Admin Rolle die User Rolle zusätzlich zugewiesen, da ein Admin Nutzer ebenfalls ein User ist und somit auch mindestens die gleichen Rechte besitzen soll. Die Zuweisung von den zusätzlichen Rollen kann unter dem Punkt „Composite Roles“ durchgeführt werden, indem die entsprechende Rolle unter „Available Roles“ ausgewählt wird und durch Betätigen des entsprechenden Buttons wird der ausgewählten Rolle eine zusätzliche zugewiesen.

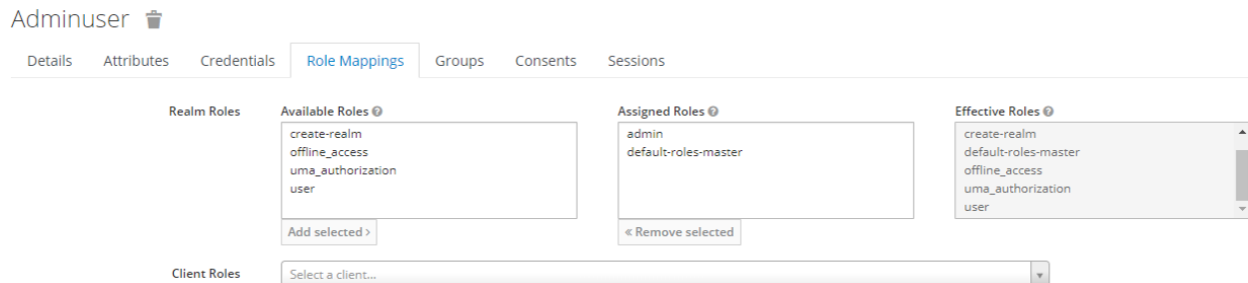


Abbildung 10: Keycloak Adminuser Rollen

Im nächsten Schritt muss ein Nutzer erstellt werden. Dies geschieht im Linken Reiter unter „Users“. Dort kann ein neuer Nutzer angelegt werden. In diesem Fall ist es der „Adminuser“. Als nächstes müssen diesem Nutzer auch die entsprechenden Rollen zugewiesen werden. Unter dem Punkt „Role Mappings“ können verschiedene Rollen ausgewählt werden. In Abbildung 10 ist der Nutzer „Adminuser“ abgebildet. Er besitzt die neu konfigurierte Rolle „admin“ sowie die Rolle „default-roles-master“, welche Standardmäßig von Keycloak erstellt worden ist. Auf der Rechten Seite unter „Effective Roles“ können alle Rollen angezeigt werden. In diesem Fall besitzt der Adminuser ebenfalls die Rolle user, da diese der Rolle admin im vorherigen Schritt zugewiesen worden ist. Nun ist Keycloak fertig konfiguriert und ist einsatzbereit.

5.5 Backend-Server

Nachdem Keycloak konfiguriert ist, muss der Backend-Server ebenfalls konfiguriert und eingestellt werden. Zu Beginn wird ein JSON-Objekt für die Konfiguration von Keycloak erstellt.

```
1  import {RESOURCEKEYCLOAK} from "../config";
2
3  const keycloakConfig = {
4    "auth-server-url": 'http://127.0.0.1:8080/auth',
5    "bearer-only": true,
6    "public-client": true,
7    "realm": 'master',
8    "resource": RESOURCEKEYCLOAK,
9    "ssl-requried": "external"
10 }
11 export default keycloakConfig;
```

Abbildung 11: Backend Keycloak Konfiguration

In Abbildung 11 ist das JSON-Objekt für Keycloak dargestellt. Als erster wird der Endpoint von Keycloak angegeben, wodurch die Authentifizierung stattfindet. Die Einstellung „bearer-only“ und „public-client“ müssen mit den Einstellungen von Keycloak übereinstimmen. Als nächster wird der anzusprechende Realm angegeben. In diesem Fall ist es der master. Unter „resource“ muss die vorher angelegte Client ID, openIdTestClient oder samlTestClient, angegeben werden. Diese Information wird in einer Umgebungsvariable gespeichert, damit ein Wechsel zwischen OpenID Connect und SAML von außerhalb möglich ist. Im nächsten Schritt muss dem Server die Konfiguration von Keycloak als middleware hinzugefügt werden.

```
1 import bodyParser from 'body-parser';
2 import cors from "cors";
3 import express from 'express';
4 import Keycloak from 'keycloak-connect';
5 import keycloakConfig from './keyCloakConfig';
6 import session from 'express-session';
7 import routes from './routes';
8
9 const server = express();
10 server.use(cors());
11 server.use(bodyParser.json());
12 const memoryStore = new session.MemoryStore();
13 const keycloak = new Keycloak({ store: memoryStore }, keycloakConfig);
14
15 server.use(keycloak.middleware());
16 routes(server, keycloak);
17
18 module.exports = server;
```

Abbildung 12: Backend Server

In Zeile 12,13 und Zeile 15 aus Abbildung 12 wird Keycloak initialisiert und dem Server hinzugefügt. In Zeile 16 werden die verschiedenen Routen registriert.

```
1 import PostController from '../src/controllers/PostController';
2
3 export default (server, keycloak) => {
4   server.post('/api/post', keycloak.protect('realm:admin'), PostController.insert);
5   server.get('/api/post', keycloak.protect('realm:user'), PostController.getAll);
6   server.put('/api/post/:id', keycloak.protect('realm:admin'), PostController.update);
7   server.delete('/api/post/:id', keycloak.protect('realm:admin'), PostController.delete);
8 };
```

Abbildung 13: Backend Routen

Bei dem Server sind insgesamt vier verschiedene Routen möglich. Die Abbildung 13 zeigt die Konfiguration der einzelnen Routen. In diesem Fall stehen die Methoden POST, GET, PUT und DELETE zur Verfügung. Mit der Funktion „keycloak.protect“ als Parameter kann angegeben werden, welche Rolle aus welchem Realm darf welche Methode ausführen. In diesem Beispiel wird dem admin die Methoden POST, PUT und DELETE zugewiesen und der user die Methode GET. Dadurch darf jeder Nutzer mit nur der Rolle user auch nur die Methode GET aufrufen. Durch die vorherige Konfiguration, dass der admin ebenfalls die rolle user zugewiesen worden ist, darf dieser ebenfalls die Methode GET ausführen. Als nächstes müssen die beiden Funktionen für das erhalten des Access-Tokens und der Anfrage für die Daten implementiert werden.

```

1  import fetch from 'node-fetch';
2  import http from "http";
3  import https from "https";
4  import { RESOURCEKEYCLOAK } from "../../backend/config/config";
5
6  export default async function getAccessToken() {
7      const httpAgent = new http.Agent({ keepAlive: true });
8      const httpsAgent = new https.Agent({ keepAlive: true });
9
10     let requestBody = new URLSearchParams();
11     requestBody.append('grant_type', 'password');
12     requestBody.append('client_id', RESOURCEKEYCLOAK);
13     requestBody.append('realm', 'master');
14     requestBody.append('username', 'adminUser');
15     requestBody.append('password', '123');
16
17     let options = {
18         method: 'POST',
19         headers: {
20             'Content-Type': 'application/x-www-form-urlencoded',
21         },
22         agent: function (parsedURL) {
23             if (_parsedURL.protocol == 'http:') {
24                 return httpAgent;
25             }
26             else {
27                 return httpsAgent;
28             }
29         },
30         body: requestBody.toString()
31     };
32
33     const keycloakUrl = 'http://127.0.0.1:8080/auth/realms/master/protocol/openid-connect/token';
34     //const keycloakUrl = 'http://127.0.0.1:8080/auth/realms/master/protocol/saml';
35     const result = fetch(keycloakUrl, options).then(async (data) => {
36         try {
37             const response = await data.json();
38             return response;
39         }
40         catch (error) {
41             return data;
42         }
43     }).then(fetchResult => { return fetchResult });
44
45     return await result;
46 };

```

Abbildung 14: Backend get Access-Token

Die Abbildung 14 zeigt die Funktion für das Anfragen des Access-Tokens. Die wichtigsten Angaben sind in den Zeilen 10-15 zu sehen. Dort werden die verschiedenen Angaben getätigt, die für Keycloak relevant sind. Zum einen muss der „grant_type“ angegeben werden. Dabei handelt es sich um den Authentifizierungsablauf der genutzt werden soll. In diesem Fall wird der Passwort Flow für OpenID Connect verwendet. Bei der „client_id“ und dem „realm“ müssen die entsprechenden aus der Konfiguration von Keycloak angegeben werden. Da in dieser Arbeit keine Nutzerinteraktion stattfindet, wo der Nutzer aufgefordert wird sein Benutzernamen und Passwort einzugeben, werden die entsprechenden Informationen direkt angegeben. Es wird für die Authentifizierung der vorher erstellte Adminuser verwendet. In Zeile 33 wird der Endpoint von Keycloak für die Authentifizierung angegeben. Dieser unterscheidet sich abhängig davon, ob OpenID Connect oder SAML verwendet wird. Am Ende der Funktion wird das Ergebnis zurückgegeben.

[illegible]

Abbildung 15: Ausgabe Token OIDC

Das Ergebnis der Funktion wird in Abbildung 15 dargestellt. Es sind alle notwendigen Informationen enthalten. Zum einen der Access-Token, womit sich bei dem Server authentifiziert werden kann. Es ist ebenfalls der Refresh-Token vorhanden, damit der Access-Token nicht ablaufen kann. Weiterhin sind Informationen zur Gültigkeitsdauer, wann das Access-Token aktualisiert wird und um welchen Typ es sich bei den Tokens handelt in der Antwort von Keycloak vorhanden. Mit diesem Access-Token kann nun der Server nach den entsprechenden Daten angefragt werden.

```

1  import fetch from 'node-fetch';
2  import http from "http";
3  import https from "https";
4
5  export default async function getDataFromBackend(requestMethod, accessToken) {
6      const httpAgent = new http.Agent({ keepAlive: true });
7      const httpsAgent = new https.Agent({ keepAlive: true });
8
9      let options = {
10         method: requestMethod,
11         headers: {
12             'Content-Type': 'application/json',
13             'authorization': accessToken
14         },
15         agent: function (_parsedURL) {
16             if (_parsedURL.protocol == 'http:') {
17                 return httpAgent;
18             }
19             else {
20                 return httpsAgent;
21             }
22         }
23     };
24
25     const backendUrl = 'http://127.0.0.1:5000/api/post';
26     const result = fetch(backendUrl, options).then(async (data) => {
27         try {
28             const response = await data.json();
29             return response;
30         }
31         catch (error) {
32             return data;
33         }
34     }).then(fetchResult => { return fetchResult });
35
36     return await result;
37 };

```

Abbildung 16: Backend Datenabfrage

Die Funktion in Abbildung 16 ähnelt der Funktion für das Anfragen nach dem Access-Token. Der Unterschied liegt zum einen an die andere URL. In diesem Fall muss die URL für die jeweilige Methode vom dem Server angegeben werden. Wenn die Funktion mit der gewünschten Methode und das erhaltene Access-Token aufgerufen wird und der jeweilige Nutzer auch über die notwendigen Rechte verfügt, wird das Ergebnis als JSON-Objekt zurückgegeben.

```
{
  error: false,
  statusCode: 200,
  data: [
    {
      _id: '62bc411e31a0c162bc514ac7',
      title: 'Software Engineering',
      description: 'Evaluation OIDC und SAML',
      content: 'Praktikum',
      createdAt: '2022-06-29T12:10:06.943Z',
      updatedAt: '2022-06-29T12:10:06.943Z',
      slug: 'Software_Engineering',
      __v: 0
    }
  ],
  total: 1
}
```

Abbildung 17: Response Backend

Wenn alles funktioniert hat, dann sieht die Ausgabe wie in Abbildung 17 aus. Die Felder „error“ und „statusCode“ geben Information darüber, ob alles funktioniert hat oder ein Fehler aufgetreten hat. In dem Feld „data“ ist ein Array mit den ganzen Informationen von der Anfrage. In diesem Fall wurde eine GET Anfrage an das Backend geschickt, welches alle Informationen aus der Datenbank zurückgibt. Nach diesem Schritt wurde eine Authentifizierung mittels OpenID Connect durchgeführt und die Anfrage nach bestimmten Daten ausgeführt.

Für SAML werden die gleichen Schritte mit der Großteil der gleichen Konfigurationen durchgeführt. Der Unterschied liegt darin, dass die Umgebungsvariable „RESOURCEKEYCLOAK“ nicht mehr die Client ID von OpenID Connect, sondern der von SAML entspricht. Außerdem muss die URL von Keycloak für den jeweiligen Endpoint geändert werden. Dies geschieht in Abbildung 14 indem die Zeilen 33 und 34 getauscht werden. Nachdem nun die Funktion „getAccessToken“ aufgerufen wird, sollte eine XML-Assertion von SAML zurückgegeben werden. In diesem Fall wird ein Fehler geworfen und die Rückmeldung „Bad Request“ angegeben.

```
Response {
  size: 0,
  timeout: 0,
  [Symbol(Body internals)]: {
    body: PassThrough {
      _readableState: [ReadableState],
      _events: [Object: null prototype],
      _eventsCount: 4,
      _maxListeners: undefined,
      _writableState: [WritableState],
      allowHalfOpen: true,
      [Symbol(kCapture)]: false,
      [Symbol(kCallback)]: null
    },
    disturbed: true,
    error: null
  },
  [Symbol(Response internals)]: {
    url: 'http://127.0.0.1:8080/auth/realms/master/protocol/saml',
    status: 400,
    statusText: 'Bad Request',
    headers: Headers { [Symbol(map)]: [Object: null prototype] },
    counter: 0
  }
}
```

Abbildung 18: Response SAML

Die Abbildung 18 zeigt die Response nachdem nach einem Access-Token angefragt wurde. Unter „status“ und „statusText“ wird jeweils der zurückgegebene Statuscode mit der entsprechenden Nachricht angegeben. Es war im Rahmen dieser Arbeit nicht möglich diesen Fehler zu beheben und eine gültige Assertion von Keycloak mit SAML zu erhalten.

6 Ergebnisse

Anmeldezeit

Für die Auswertung der benötigten Zeit wurde ein Hilfsskript geschrieben, wodurch die Authentifizierung und das Anfragen der Daten mehrmals durchlaufen wird und in eine Excel-Datei geschrieben.

```
31   for (let i = 0; i < max_iteration; i++) {  
32       timer.start();  
33       const tokenResponse = await getAccessToken();  
34       const bearerToken = "Bearer " + tokenResponse.access_token;  
35       const response = await getDataFromBackend('GET', bearerToken);  
36       timer.stop();  
37       if (response.error) {  
38           return;  
39       }  
40       time = timer.ms();  
41       time < min ? min = time : null;  
42       time > max ? max = time : null;  
43       total += time;  
44       DATA_ROWS.push([{}], { type: Number, value: time });  
45   };
```

Abbildung 19: Hilfsskript für die Auswertung

In der Abbildung 19 ist ein Ausschnitt vom dem Hilfsskript abgebildet. Zu Beginn der Schleife wird ein Timer gestartet, um den Startzeitpunkt zu ermitteln. In den nächsten beiden Zeilen wird ein Access-Token angefragt und daraus ein Bearer-Token erstellt, welches für den Server für die Authentifizierung benötigt wird. Nachdem die Daten vom Server erhalten wurde, wird der Timer gestoppt und die Dauer für den Erhalt des Access-Tokens und den gewünschten Daten berechnet. Falls bei der response ein Fehler aufgetreten ist und somit keine Daten als Antwort erhalten worden sind, wird die Schleife unterbrochen und keine Daten in der Excel-Datei geschrieben. Die letzten Zeilen dienen lediglich zum Analysieren der minimal, maximal und durchschnittlich benötigte Zeit und das Hinzufügen einer neuen Zeile in die Excel-Datei, welches vorerst in einem Array zwischengespeichert wird. Nachdem die maximale Anzahl an Iterationen durchlaufen ist, werden die Informationen in einer Excel-Datei gespeichert.

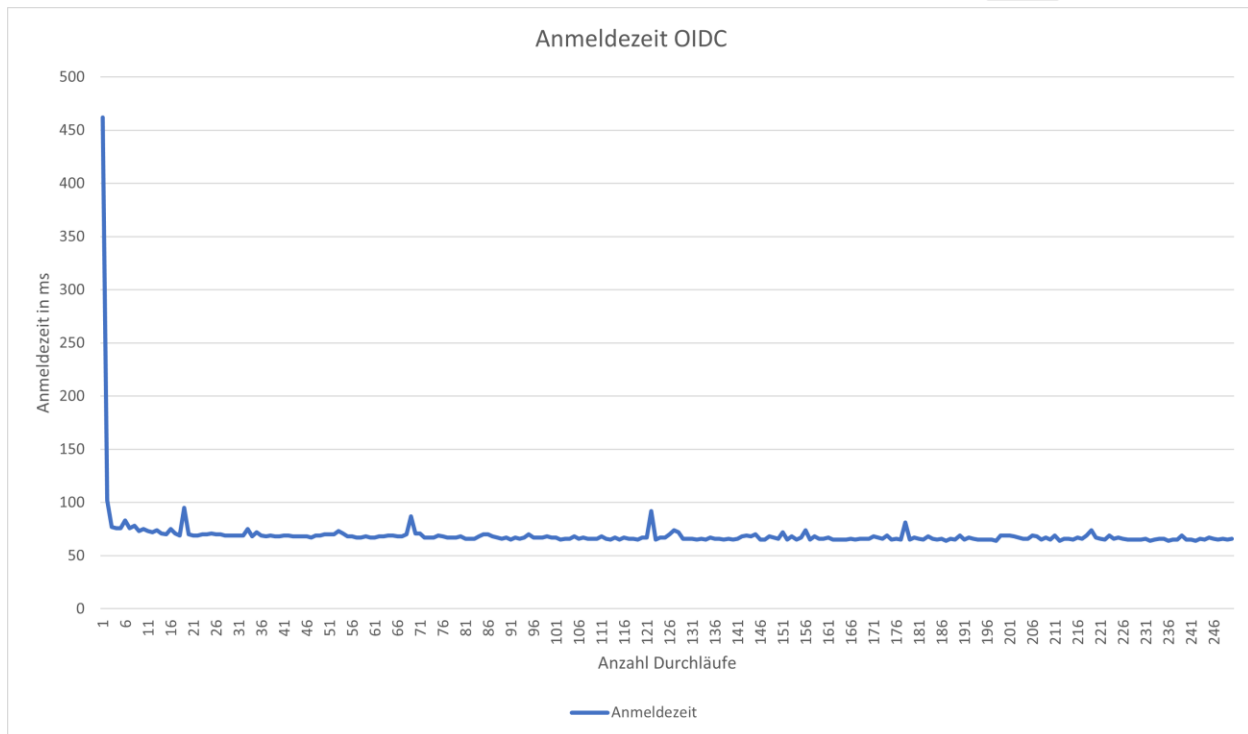


Abbildung 20: Diagramm der Anmeldezeit OIDC

In der Abbildung 20 ist die insgesamt benötigte Zeit für die Authentifizierung und der Erhalt der Daten in einem Liniendiagramm abgebildet. Es ist zu erkennen, dass zu Beginn die maximal benötigte Zeit von 462ms erreicht wird. Schon beim zweiten Durchlauf werden nur 102ms benötigt. Bis am Ende eine durchschnittliche Zeit von 63ms erreicht wird. Die schnellste Zeit liegt bei 60ms. Dadurch lässt sich erkennen, dass die Authentifizierung mittels OIDC beim erstmaligen Anmelden die längste Zeit benötigt. Bei jedem erneuten anmelden bei Keycloak wird dieser Vorgang immer schneller. Das liegt womöglich daran, dass Keycloak intern bestimmte Informationen abspeichert und cacht und somit das mehrmalige Nutzen beschleunigt wird. Insgesamt ist OIDC extrem schnell.

Developer Experience

Als Entwickler spielt die vorhandene Dokumentation für die jeweiligen Protokolle eine große Rolle. Ebenfalls wichtig und hilfreich ist die Anzahl an Unterstützung die eine Onlinerecherche mit sich bringt. Bei beiden Faktoren hat OpenID Connect deutlich besser abgeschnitten als SAML. Sowohl bei der Dokumentation von Keycloak als auch während der Internetrecherche wurden deutlich mehr Tipps und Erklärungen zu OpenID Connect gefunden als für SAML. Die meisten Beispiele

sind für OpenID Connect zu finden und die wenigen für SAML erklären eine ganz andere Verwendung von SAML, die nicht zu dem Projekt gepasst haben. Aus dem Grund, dass SAML nicht erfolgreich implementiert worden ist, kann keine genaue Beurteilung bezüglich des Konfigurationsaufwandes vorgenommen werden. OpenID Connect war sehr gut verständlich und ohne große Konfigurationen sowohl seitens Keycloak als auch beim eigenen Backend-Server umsetzbar.

Für die restlichen Metriken **Session Parameter**, **Signaturalgorithmus**, **Unterstützte Programmiersprachen** und **Anzahl möglicher Identitätsanbieter** gibt es keine Unterschiede zwischen den beiden Protokollen OpenID Connect und SAML, da als gemeinsamer Identitätsanbieter Keycloak verwendet wird und dieser für beide die gleichen Einstellungen, Konfigurationen und Möglichkeiten bereitstellt. Aus diesem Grund sind diese Metriken gemeinsam zu betrachten. Für die Session Parameter bietet Keycloak eine Vielzahl an Konfigurationsmöglichkeiten. Es kann zum Beispiel für jeden einzelnen Realm Einstellungen zu den SSO Sessions, den Client Session als auch die Gültigkeitsdauer eines Access-Tokens getätigt werden. Als Signaturalgorithmus stehen insgesamt vier verschiedene Verfahren (ES, HS, PS, RS) mit jeweils drei unterschiedlichen Längen des Schlüssels (256, 384, 512), der intern verwendet wird, zur Auswahl. Es werden alle gängigen Programmiersprachen unterstützt, da Keycloak mit Plugins erweitert werden kann. [13] Als mögliche Identitätsanbieter stehen 13 verschiedene zur Auswahl.

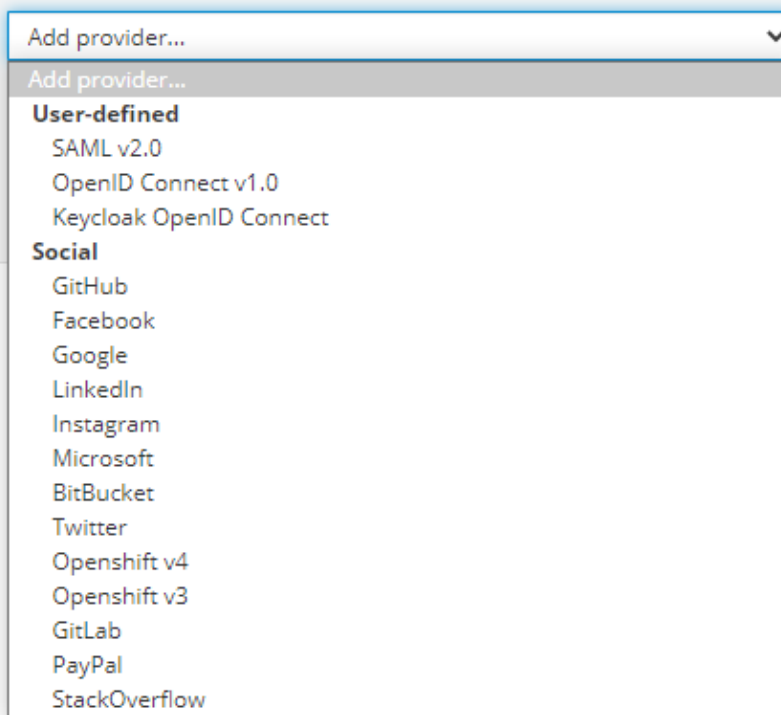


Abbildung 21: Keycloak Identitätsanbieter

Die Abbildung 21 zeigt eine Liste aller möglichen sozialen aber auch selbst definierten Identitätsanbieter. Somit werden alle großen Webseiten wie GitHub, Facebook, Google, usw. von Keycloak unterstützt.

7 Literaturverzeichnis

- [1] Perseus - Perseus, *Authentisierung, Authentifizierung und Autorisierung - Glossar - Perseus Technologies*. [Online]. Verfügbar unter: <https://www.perseus.de/2022/04/19/authentisierung-authentifizierung-und-autorisierung/> (Zugriff am: 28. Juni 2022).
- [2] IONOS Digitalguide, *OAuth (Open Authorization)*. [Online]. Verfügbar unter: <https://www.ionos.de/digitalguide/server/sicherheit/was-ist-oauth/> (Zugriff am: 28. Juni 2022).
- [3] OAuth, *Was ist ein Zugriffstoken – OAuth 2.0*. [Online]. Verfügbar unter: <https://oauth.net/2/access-tokens/> (Zugriff am: 28. Juni 2022).
- [4] OAuth 2.0 Simplified, *Refresh Tokens - OAuth 2.0 Simplified*. [Online]. Verfügbar unter: <https://www.oauth.com/oauth2-servers/making-authenticated-requests/refreshing-an-access-token/> (Zugriff am: 28. Juni 2022).
- [5] OAuth 2.0 Simplified, *Authorization Code Grant - OAuth 2.0 Simplified*. [Online]. Verfügbar unter: <https://www.oauth.com/oauth2-servers/server-side-apps/authorization-code/> (Zugriff am: 28. Juni 2022).
- [6] *Final: OpenID Connect Core 1.0 incorporating errata set 1*. [Online]. Verfügbar unter: https://openid.net/specs/openid-connect-core-1_0.html (Zugriff am: 28. Juni 2022).
- [7] Auth0.com, *JWT.IO - JSON Web Tokens Introduction*. [Online]. Verfügbar unter: <https://jwt.io/introduction> (Zugriff am: 28. Juni 2022).
- [8] Okta Developer, *An Illustrated Guide to OAuth and OpenID Connect*. [Online]. Verfügbar unter: <https://developer.okta.com/blog/2019/10/21/illustrated-guide-to-oauth-and-oidc> (Zugriff am: 28. Juni 2022).
- [9] Jeff Petters, *Was ist SAML und wie funktioniert sie?* [Online]. Verfügbar unter: <https://www.varonis.com/de/blog/was-ist-saml-und-wie-funktioniert-sie> (Zugriff am: 29. Juni 2022).
- [10] *Understanding SAML | Okta Developer*. [Online]. Verfügbar unter: <https://developer.okta.com/docs/concepts/saml/#planning-for-saml> (Zugriff am: 29. Juni 2022).
- [11] SMF, *Keycloak - Ein Überblick - SMF*. [Online]. Verfügbar unter: <https://www.smf.de/keycloak-ein-ueberblick/> (Zugriff am: 29. Juni 2022).
- [12] Ironhack, *Was ist MongoDB? Eine Anleitung zu MongoDB und wie man es auf Catalina OS installiert*. [Online]. Verfügbar unter: <https://www.ironhack.com/de/webentwicklung/was-ist-mongodb-eine-praktische-anleitung-zu-mongodb-und-wie-man-es-auf-catalina> (Zugriff am: 29. Juni 2022).
- [13] TYPO3 & Neos Agentur, *Single-Sign-On mit Keycloak*. [Online]. Verfügbar unter: <https://punkt.de/de/blog/2020/single-sign-on-mit-keycloak.html> (Zugriff am: 29. Juni 2022).