TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für Systemarchitektur Professur Rechnernetze

MASTERARBEIT

UNTERSUCHUNG VON SSO-STRATEGIEN IM KONTEXT VOM AMCS

Alrik Geselle

Matrikel-Nr.: 3680287

Betreuer: Dr.-Ing. Tenshi Hara, Dr.-Ing. Iris Braun, Tommy Kubica, M.Sc.

verantwortlicher Hochschullehrer: Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill

Eingereicht am: 05.07.2018

SELBSTSTÄNDIGKEITSERKLÄRUNG

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form einer anderen Prüfungsbehörde vorgelegt und ist auch noch nicht veröffentlicht worden.

Dresden, 05.07.2018

……

Alrik Geselle

ABSTRACT

Diese Arbeit befasst sich mit der Untersuchung, dem Vergleich und der Wahl einer geeigneten Single Sign-on Lösung für den Einsatz mit dem Auditorium Mobile Classroom Service (AMCS). Single Sign-on ermöglicht die Verwendung mehrerer Anwendungen mit nur einem Nutzerkonto. AMCS ist eine von der Technischen Universität Dresden entwickelte Software zur Steigerung der Interaktivität zwischen den Studierenden und dem Lehrenden. Dabei steht die anonyme Nutzung an oberster Stelle. Untersucht wurden Technologien wie OAuth, OpenID, OpenID Connect, SAML, CAS und eine Eigenentwicklung. Beim Vergleich wurde die Eigenentwicklung anhand von Anforderungskriterien gewählt und prototypisch implementiert. Anhand von Funktionstests, Untersuchungen der Leistungsfähigkeit und einer Nutzerbefragung wurde der Prototyp evaluiert. Die Auswertung der Ergebnisse bestätigen die Wahl der Eigenentwicklung, zeigen aber auch Grenzen des Prototyps auf. Zum Schluss werden einige Verbesserungsvorschläge diskutiert.

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für Systemarchitektur Professur Rechnernetze

AUFGABENSTELLUNG FÜR DIE MASTER-ARBEIT

THEMA: Untersuchung von SSO-Strategien im Kontext vom AMCS

Name:	Geselle, Alrik	Studiengang:	Master Medieninformatik
Matrikel-Nummer:	3680287	Projekt/Fokus:	AMCS
verantwortlicher Hochschullehrer:	Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill		
involvierte Mitarbeiter:	Tommy Kubica, M.Sc.; Dr.-Ing. Iris Braun		
Beginn am:	2. Februar 2018	einreichen bis:	6. Juli 2018

ZIELSTELLUNG

Am Lehrstuhl Rechnernetze wurden in den vergangenen Jahren mehrere prototypische Untersuchungen zum Einsatz technischer Werkzeugen im Lehrbetrieb durchgeführt. Unter anderem wurden Werkzeuge in Vorlesungen und Übungen getestet; es wurden für den jeweiligen Einsatz valide Ergebnisse gewonnen. Am Ende der bisherigen Entwicklungen steht mit Auditorium Mobile Classroom Service (AMCS) eine komplexe Anwendungsarchitektur und -plattform mit zugehörigen mobilen Applikationen.

Zum Zwecke der einfacheren Bereitstellung von AMCS mit unterschiedlichem Funktionsumfang wird inzwischen ein Service-orientierter Einsatz angestrebt. Eine zentrale Voraussetzung zum Gelingen dieses Vorhabens ist ein Login-Service, der die Nutzer und alle zu nutzenden Dienste im Sinne eines Single-Sign-Ons (SSO) aneinanderbindet.

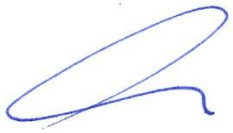
Im Rahmen dieser Master-Arbeit sollen existierende SSO-Lösungen untersucht und miteinander verglichen werden. Beim Vergleich soll besonders auf den AMCS-Anwendungsfall, nämlich die anonyme Nutzung durch Studierende und die gesicherte Nutzung für Lehrende eingegangen werden. Ggf. sind diverse Ansätze in einem technisch sinnvollen Maße für AMCS anzupassen und zu kombinieren. Anschließend soll dann untersucht werden, ob und wie die gewünschte SSO-artige Lösung der Verteilung von AMCS dienlich sein kann, insbesondere bei Einsätzen an Partnereinrichtungen, die ggf. eigene Login-Services verwenden wollen, oder gezielt zentral bereitgestellte Services einbinden wollen.

Auf Basis der Ergebnisse der Untersuchungen soll eine Proof-of-Concept-Implementierung für AMCS erfolgen und ihre Funktionalität nachgewiesen werden. Anschließend sind die üblichen Metriken zu überprüfen. Eine Evaluation insbesondere mit Rücksicht auf Zugriffszeiten und von den Nutzern wahrgenommener Performance (bspw. SUS oder NASA TLX) sind zwingend erforderlich.

Sollte sich aus den Untersuchungen eine Notwendigkeit einer Single-Sign-off-Strategie ergeben, ist diese ebenfalls zu berücksichtigen.

SCHWERPUNKTE

- Analyse und Auswahl bestehender Ansätze und Lösungen,
- Definieren von Anforderungen,
- Definieren von Bewertungskriterien für die Anforderungserfüllung,
- prototypische Umsetzung des Konzepts,
- Evaluation und Auswertung der Ergebnisse.



Dr.-Ing. Tenshi Hara
(für den Lehrstuhl)

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

ABKÜRZUNGS- UND SYMBOLVERZEICHNIS

AMCS	Auditorium Mobile Classroom Service
CAS	Central Authentication Service
CMS	Content-Management-System
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
JWT	JSON Web Token
PDO	PHP Data Objects
PHP	Rekursives Akronym für **P**HP: **H**ypertext **P**reprocessor
SAML	Security Assertion Markup Language
SSO	Single Sign-on
TLS	Transport Layer Security
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

1 EINLEITUNG

Wer kennt das Problem nicht? Mit jedem Tag mehr steigt das Angebot an digitalen Diensten. Immer häufiger wird dabei ein eigenes Nutzerkonto für die Verwendung dieser Dienste benötigt. Fast immer muss die eigene E-Mail-Adresse und ein dazugehöriges Passwort gewählt werden. Es wird empfohlen ein besonders sicheres Passwort zu wählen. Doch mit jedem weiteren Nutzerkonto wird es schwieriger sich die verschiedenen Passwörter zu merken. Oft wird aus diesem Grund das gleiche Passwort für diverse Dienste verwendet, was ein erhebliches Sicherheitsproblem darstellt.

Genau für jene Fälle gibt es eine Lösung. *Single Sign-on* nennt sich die Technologie, mit der es möglich ist, mit nur einem Nutzerkonto mehrere Dienste zu verwenden. Was zunächst wie die perfekte Lösung für diese Problematik erscheint, ist jedoch nicht ganz so einfach wie gedacht. Es existieren diverse Ansätze, die eine Single Sign-on-Funktionalität bereitstellen. Manche lassen sich beliebig verwenden, andere wiederrum sind nur für bestimmte Anwendungsszenarien gedacht. Die Einen bieten die Möglichkeit der Übertragung von weiteren Nutzerdaten, die Anderen werden nur zur Anmeldung verwendet.

1.1 MOTIVATION

Die Technische Universität Dresden entwickelt mit dem Auditorium Mobile Classroom Service (AMCS) eine Software zur Steigerung der Interaktivität zwischen den Studierenden und dem Lehrenden während einer Lehrveranstaltung. Dadurch soll der Lernerfolg gesteigert und das Feedback für den Lehrenden verbessert werden. Den Studierenden können so, während einer Lehrveranstaltung, Fragen gestellt oder Nachrichten gesendet werden.

Mit der Entwicklung von AMCS stellte sich u.a. die Frage nach einem Login-Service, der die Nutzer und alle zu nutzenden Dienste aneinanderbindet. Dabei sollen sowohl AMCS als auch zukünftige Dienste miteinbezogen werden. Das heißt, mit nur einem einzigen Nutzerkonto sollen die verschiedenen Dienste verwendet werden können. Das Besondere an AMCS ist allerdings die Möglichkeit der anonymen Nutzung, die auch weiterhin beibehalten werden soll. Ferner soll der Login-Service auch durch weitere Einrichtungen genutzt werden können. Zu diesem Zweck wird ein geeignetes Verfahren benötigt, welches den genannten Anforderungen genügt.

1.2 ZIELSTELLUNG UND VORGEHENSWEISE

Ziel dieser Arbeit ist die Untersuchung einer geeigneten Single Sign-on Lösung, welche dann prototypisch umgesetzt werden soll. Diesbezüglich wird zunächst im Kapitel 2 der Begriff des

Single Sign-on näher untersucht. Dazu zählen auch die Betrachtung des Aufbaus, sowie die Funktionsweise und Vor- und Nachteile. Danach erfolgen weitere Untersuchungen anhand von aktuellen Single Sign-on Lösungen, die im Anschluss präsentiert werden. Zusätzlich wird der momentane Stand von AMCS analysiert, um einen geeigneten Prototypen integrieren zu können. Im Kapitel 3 erfolgt die Präsentation einer Eigenentwicklung, die gleichwertig gegenüber den zuvor untersuchten Lösungen ist. Im Anschluss daran werden die genaueren Anforderungen an die gesuchte Lösung gestellt. Zur Wahl einer geeigneten Single Sign-on Lösung wird anhand der Anforderungen ein fundierter Vergleich zwischen den Lösungen statt. Im anschließenden Kapitel 4 wird die gewählte Lösung prototypisch implementiert und in AMCS integriert. Die darauffolgende Evaluation im Kapitel 5 untersucht die Funktionalität des Prototyps sowie dessen Leistungsfähigkeit. Darüber hinaus wird eine Nutzerbefragung durchgeführt, welche das Verhalten und den Funktionsumfang des Prototyps bewertet. Danach erfolgt eine Auswertung und Diskussion der Evaluationsergebnisse. Zum Schluss werden die Untersuchungen und Ergebnisse dieser Arbeit zusammengefasst und zukünftige Ideen erörtert.

2 GRUNDLAGEN

Dieses Kapitel beschreibt die Grundlagen von Single Sign-on Systemen. Dabei wird zunächst die Bedeutung des Begriffs Single Sign-on und dessen allgemeine Grundidee erklärt. Diese schließt sowohl den Aufbau, als auch die Funktionsweise ein und soll anhand von Diagrammen verdeutlicht werden. Im Anschluss daran werden eine Reihe von bereits existierenden Single Sign-on Lösungen bzw. Ansätzen vorgestellt und grob erläutert. Eine genauere technische Betrachtung der einzelnen Systeme wird dann erst im darauffolgenden Kapitel näher untersucht. Die Grundlagen dienen somit als Einstiegshilfe und sollen zudem einige interessante Fakten zur Entstehung dieser Systeme darbieten.

2.1 SINGLE SIGN-ON

Im Folgenden wird Single Sign-on allgemein beschrieben und dessen Aufbau sowie die Funktionsweise erklärt. Des Weiteren werden die grundlegenden Vor- und Nachteile dargelegt. Die verschiedenen Betrachtungen sind auf den Webbereich ausgelegt, da sich diese Arbeit auf den Kontext des Auditorium Mobile Classroom Service (AMCS) bezieht. Single Sign-on gibt es auch in anderen Bereichen wie beispielsweise für native Plattformen. Sowohl Anwendungen für Microsoft Windows oder Mac OSX als auch Apps für den mobilen Bereich, wie Google Android und Apple IOS, sind mit Single Sign-on Systemen denkbar. Auch hierfür existieren bereits diverse Lösungen, welche allerdings in den folgenden Ausführungen keine nähere Berücksichtigung finden, da es sonst über den Rahmen dieser Arbeit hinausgehen würde. Doch zunächst soll erst einmal der Begriff Single Sign-on näher betrachtet werden.

2.1.1 Begriffserklärung

Single Sign-on bedeutet so viel wie „Einmalanmeldung“. Solche Systeme ermöglichen die Nutzung verschiedener Dienste und Ressourcen mit einer einzigen Anmeldung. Typischerweise wird für jeden Dienst, der geschützte Ressourcen bereitstellt, ein eigenes Benutzerkonto benötigt. Um dann die jeweiligen Dienste nutzen zu können, ist es notwendig, sich jedes Mal erneut anzumelden. Üblicherweise authentifiziert sich ein Nutzer gegenüber dem System durch die Eingabe einer Email-Adresse bzw. eines Benutzernamens und einem Passwort. Die Authentifizierung, welche im Prinzip die Überprüfung der Identität des Nutzers darstellt, darf nicht mit der der Autorisierung verwechselt werden. Die Autorisierung beschreibt die Rechte nach der Authentifizierung eines Nutzers oder eines Systems, dass im Auftrag des Nutzers handelt (vgl. Rockford , 2004).

Mit steigender Anzahl verfügbarer Dienste wird es schnell schwierig, sich alle Zugangsdaten zu merken. Doch genau hier sollen SSO-Systeme Abhilfe schaffen. Single Sign-on soll verhindern, dass Nutzer mehrere Zugangsdaten benötigen und schneller auf Daten zugreifen können. Nach erfolgreicher Authentifizierung wird dem Nutzer eine Art Zugangsschlüssel ausgestellt. Mit diesem Zugangsschlüssel kann sich der Nutzer gegenüber einem Dienst authentifizieren, welcher ihm dann Zugriff gewährt. SSO ist vor allem in Unternehmen sinnvoll, die mehrere web- oder cloudbasierte Dienste anbieten und bei dem die Arbeitnehmer sonst viele verschiedene Zugangsdaten benötigen würden. Die diversen Zugangsdaten müssten dann bei jedem einzelnen Dienst eingegeben werden. Genau an dieser Stelle wäre es sinnvoll, die jeweiligen Anmeldeoberflächen auf eine einzige Stelle auszulagern. Ein Nutzer, der sich dann bei den Diensten anmelden möchte, meldet sich nun direkt bei ein- und derselben Anmeldeoberfläche mit immer gleichen Zugangsdaten an. Für einen Nutzer existiert also nur noch eine Identität im SSO-System, die für alle anderen Dienste gilt. Der Login-Server des SSO-Systems, welcher die Anmeldeoberfläche bereitstellt, übernimmt nun die Identifizierung des Nutzers durch ein geeignetes Authentifizierungsverfahren. In der gängigen Praxis ist das meistens eine Passworteingabe. Darüber hinaus merkt sich der Login-Server die Anmeldung des Nutzers und kann gegenüber einem Dienst authentifizieren, dass dieser ordnungsgemäß angemeldet ist. Möchte der Nutzer nun einen anderen Dienst nutzen, muss dieser sich nicht erneut anmelden und kann den Dienst sofort nutzen. Auf diese Weise kann schneller auf Dienste zugegriffen werden, wodurch sich eine Zeitersparnis ergibt.

Das genaue Gegenstück zur Single Sign-on Strategie nennt sich Single Sign-off. Ein Nutzer kann sich so mit einer einzigen Abmeldung bei allen anderen Diensten gleichzeitig abmelden. Es gibt verschiedene Umsetzungen dieses Verfahrens. Es kann automatisch oder manuell erfolgen.

2.1.2 Aufbau

Der Aufbau eines Single Sign-on Systems lässt sich einfach am Beispiel von Webanwendungen erklären: Es existiert ein Nutzer, welcher die verschiedenen Dienste im Web nutzen möchte. Die einzelnen Dienste stellen geschützte Daten bereit. Der Nutzer interagiert mit den Diensten durch einen Webbrowser, der die zum jeweiligen Dienst gehörigen Webadressen (URL) aufruft. Abbildung 2.1 zeigt das Anmeldeverfahren ohne Single Sign-on. Zu jedem der Dienste existiert eine eigenständige Anmeldeoberfläche, auf die der Nutzer geleitet wird, um sich anzumelden.

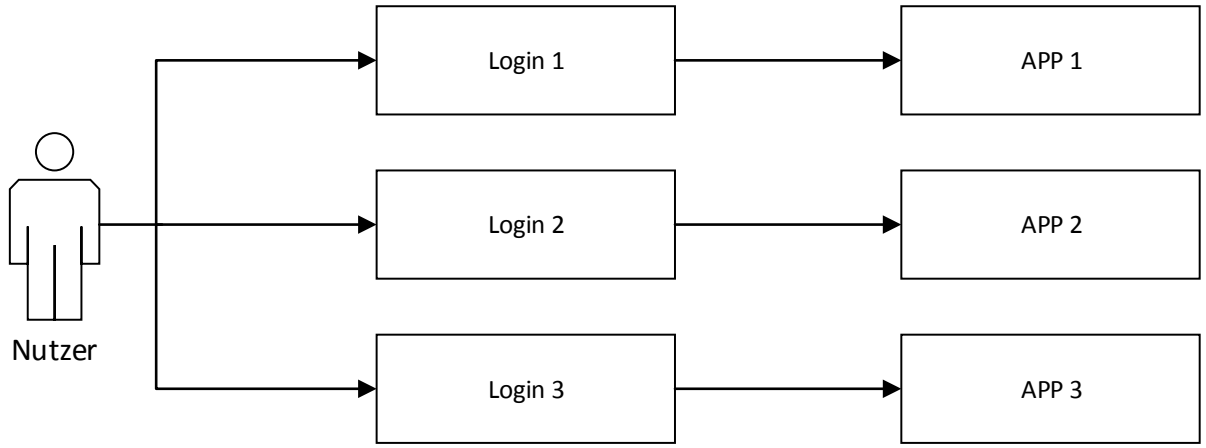


Abbildung 2.1: Beispiel eines herkömmlichen Anmeldeverfahrens, bei dem mehrere Anmeldungen notwendig sind, um verschiedene Dienste nutzen zu können.

Im Vergleich zur herkömmlichen Anmeldung zeigt Abbildung 2.2 das Anmeldeverfahren mit einem Single Sign-on System. Der Nutzer interagiert wieder mit den Diensten durch einen Webbrowser. Der Aufbau der Dienste ist bis auf die Anmeldeoberfläche unverändert. Alle Dienste sind durch eine einzige Anmeldeoberfläche miteinander verbunden.

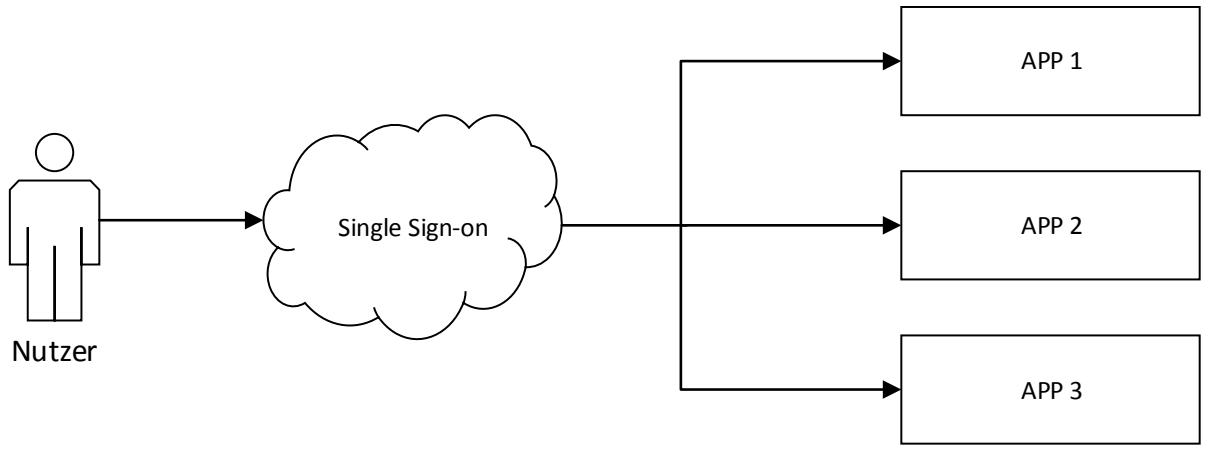


Abbildung 2.2: Beispiel eines Anmeldeverfahrens mit Single Sign-on, beim dem nur eine Anmeldung notwendig ist, um verschiedene Dienste nutzen zu können.

2.1.3 Funktionsweise

Ein Nutzer, der eine geschützte Webanwendung nutzen möchte, benutzt einen Webbrowser, um auf Webseiten der jeweiligen Dienste zu gelangen. Die Kommunikation erfolgt in der Regel über das *Hypertext Transfer Protocol (HTTP).* Wenn ein Nutzer auf einen geschützten Bereich zugreifen möchte, wird dieser zu einer Anmeldeoberfläche geleitet, sofern dieser nicht

bereits angemeldet ist. Dort findet die Authentifizierung des Nutzers statt. Die Authentifizierung erfolgt durch ein geeignetes Authentifizierungsverfahren, wie der Eingabe von einem Benutzernamen und eines Passwortes. Die Zugangsdaten werden nach der Eingabe verarbeitet, wie zum Beispiel durch einen Abgleich mit einer Datenbank. War die Authentifizierung erfolgreich, wird ihm ein sogenannter Token ausgestellt. Dieser Token ist eine Art Zugangsschlüssel, um in den geschützten Bereich zu gelangen. Typischerweise werden Tokens in Form eines HTTP-Cookie gespeichert und bestehen meist aus einer Folge zufälliger Zahlen oder Buchstaben, die keinerlei Informationen für Außenstehende enthalten. Für einen Nutzer mit einem gültigen Token wird nun eine sogenannte Session gestartet und somit ist dieser erfolgreich angemeldet. Die Webanwendung kann einen Nutzer anhand der Session wiedererkennen und direkten Zugriff auf den geschützten Bereich gewähren. Doch was genau ist eine Session und wie funktioniert sie?

Aufbau und Funktionsweise von Sessions

Eine Sitzung (engl. Session) bezeichnet in der Informatik eine Verbindung zwischen einem Server und einem Client. Sie wird durch eine Anmeldung (engl. Login) gestartet und durch eine Abmeldung (engl. Logout) wieder zerstört. Sessions sind notwendig, da HTTP-Verbindungen zustandslos sind. Das heißt, ein Nutzer lässt sich nicht ohne Weiteres identifizieren. Denn weder die IP-Adresse des Clients, noch die durch den Browser übermittelten Daten sind ausreichend genug, um einen Nutzer eindeutig zu identifizieren. Die IP-Adresse ist meist dynamisch und kann sich verändern. In der Regel ändert sie sich spätestens nach 24 Stunden, wenn der Interprovider die Zwangstrennung auslöst, um eine neue IP-Adresse zu vergeben. Aus diesem Grund ist ein Konzept notwendig, welches einem Server ermöglicht, seine Clients eindeutig zu identifizieren. Sessions ermöglichen dies, indem auf der Anwendungsschicht im *OSI-Modell*[1] ein solches Verhalten implementiert wird. Der Server weist dazu jedem Client eine eindeutige Session-ID zu und speichert weitere relevante Sitzungsdaten zu jeder ID. Eine Session-ID besteht aus einer Zeichenkette, die sich für gewöhnlich aus einer zufälligen Reihenfolge von Zahlen und Buchstaben zusammensetzt. Bekommt ein Client eine solche Session-ID zugewiesen, wird sie in der Regel in einem Cookie gespeichert. Bei jeder weiteren Kommunikation sendet der Client zusätzlich seine ihm zugewiesene Session-ID mit. Der Server weiß dann, von wem genau die HTTP-Anfrage kam.

[1] Das OSI-Modell (englisch Open Systems Interconnection Model) ist ein Modell, um die Architektur von Protokollen als Schichten darzustellen. Dadurch soll die Entwicklung und Kommunikation zwischen den Schichten sauberer und modularer aufgebaut werden, damit diese untereinander austauschbar sind.

2.1.4 Vor- und Nachteile

Single Sign-on Systeme bieten gegenüber den Mehrfachanmeldungen eine ganze Reihe an Vorteilen. Zum einen ist es für Nutzer bequemer, da sie sich weniger Zugangsdaten merken müssen. Durch den Wegfall der Mehrfachanmeldungen wird auch Zeit eingespart, die anderweitig genutzt werden kann. Zum anderen wird die Sicherheit erhöht. Denn mehrere Zugangsdaten erfordern mehrere Passwörter. Da sie dennoch einprägsam sein sollen, werden meist zu einfache und damit unsichere Passwörter gewählt. Bei einem SSO-System wird nur noch ein einziges sicheres Passwort benötigt. Somit kann nun das Passwort auch komplizierter gewählt werden als zuvor. Zusätzlich erhöht sich ebenso der Schutz vor Angriffen, da die Zugangsdaten nur einmal übertragen werden müssen. Eine immer gleiche Anmeldeoberfläche verringert zudem das Risiko einer Phishing-Attacke zum Opfer zu fallen, denn es müssen weniger Sicherheitsmerkmale überprüft werden und der Nutzer wird weniger dazu verleitet, seine Zugangsdaten bei fremden Systemen einzugeben. In den letzten Jahren ist das Angebot an Diensten wie Webseiten, Apps und Cloud-Services stetig gewachsen. Und fast überall werden Zugangsdaten benötigt. Daher ist es wenig verwunderlich, dass sich Passwortdiebstahl zu einem lukrativen Geschäft entwickelt hat. Doch auch hier bietet ein SSO-System wieder Vorteile. Die Wahrscheinlichkeit eines Passwortdiebstals wird verringert, da die Zugangsdaten nur noch bei dem SSO-System hinterlegt sind. Werden die Zugangsdaten direkt bei ein und derselben Stelle übergeben, haben die daran gekoppelten Dienste auch gar keine Möglichkeit mehr, diese sensiblen Daten falsch zu behandeln oder gar auszuspähen. Somit konzentrieren sich die Sicherheitsaspekte auf weniger Stellen als zuvor. Nicht nur die Sicherheit wird erhöht, auch bei der Account-Verwaltung eines solchen Systems werden einige Vorgänge vereinfacht. Soll zum Beispiel der Zugriff eines Nutzers verändert oder gesperrt werden, muss ein Systemadministrator dies nur noch an einer Stelle erledigen. Zusammengefasst sind demgemäß als Hauptvorteile eine erhöhte Sicherheit, ein verringerter Aufwand und Zeitersparnis zu nennen.

Auch zu beachten sind die jeweiligen Nachteile. Schafft es ein Angreifer an die Zugangsdaten zu gelangen oder die Identität des Nutzers eines SSO-Systems zu stehlen, hat er mit einem Mal Zugriff auf eine Vielzahl an Diensten anstatt auf nur einem. Dem SSO-Anbieter, also demjenigen der den Login-Server bereitstellt, sollte auf jeden Fall vertraut werden können. Er kann jederzeit mitverfolgen, über welche Dienste sich ein Nutzer anmeldet bzw. welche er gerade nutzt. Beim Abmelden von einem Dienst hat die Vereinfachung des Anmeldens durch ein SSO-System den Nachteil, dass der Nutzer bei anderen Diensten immer noch angemeldet ist, auch wenn sie nicht mehr genutzt werden. Hierfür muss speziell ein Verfahren zum Single Sign-off entwickelt werden. Diesbezüglich gibt es mehrere Möglichkeiten, welche jedoch auf den jeweiligen Anwendungsfall angepasst werden müssen. Letztlich verändert sich durch den Einsatz von Single Sign-on auch die Verfügbarkeit der Dienste. Alles steht und fällt mit der Zuverlässigkeit des SSO-Systems. Ist der Login-Server nicht verfügbar, können Nutzer keine der Dienste nutzen. Ebenso kann sich ein Nutzer nur teilweise vom System abmelden, wenn

denn eine entsprechende Single Sign-off Strategie implementiert wurde. Die Sperrung eines Benutzerkontos hat auch weitreichendere Folgen auf die Verfügbarkeit eines Dienstes als bisher. Gibt ein Nutzer mehrmals ein falsches Passwort ein und wird danach gesperrt, ist er nicht nur bei einem Dienst, sondern auch bei allen anderen Diensten gesperrt. Vielleicht ist diese Verhaltensweise aber auch erwünscht und somit als ein Vorteil anzusehen.

2.2 AKTUELLE SINGLE SIGN-ON LÖSUNGEN

In den zahlreichen online-Berichten und Veröffentlichungen zu Single Sign-on tauchen immer wieder die gleichen Technologien auf. Vor allem große Konzerne wie Google LLC und Facebook Inc. scheinen Richtungen vorzugeben. Oft sind Begriffe zu lesen wie OAuth, OpenID, SAML oder im universitären Bereich CAS und Shibboleth. Aber es gibt auch viele weitere Technologien, über die allerdings weitaus weniger berichtet wird. Zum Teil basieren die oben genannten Technologien auf früheren Entwürfen. Deshalb handelt dieses Kapitel von den gängigsten Technologien und deren vorhergehenden Entwicklungen. Der aktuelle Trend scheint in Richtung OAuth 2.0 und OpenID Connect zu gehen. Dabei basiert OpenID Connect direkt auf OAuth 2.0 und stellt deshalb eine Erweiterung dar, die für einen bestimmten Zweck, nämlich der Authentifizierung genutzt wird. OAuth hingegen ist sehr abstrakt und das Hauptaugenmerk liegt in der Autorisierung. Nichtsdestotrotz sollen auch ein paar weitere Technologien vorgestellt werden, die bei der Suche nach einer geeigneten Sing Sign-on Lösung behilflich sein können.

2.2.1 OAuth (Open Authorization)

OAuth (Open Authorization) ist ein standardisiertes und offenes Protokoll, welches eine API zur sicheren Autorisierung bereitstellt. Mit dieser lassen sich Endanwender nicht nur Autorisieren, sondern zusätzlich auch Authentifizieren. Das heißt, ein Nutzer (Resource Owner) kann einer Webanwendung (Client) erlauben, in seinem Namen auf geschützte Daten zuzugreifen, die sich auf einem anderen Server (Service Provider) befinden. Der Client erlangt dabei jedoch nie Kenntnis über die Zugangsdaten des Nutzers. (vgl. Hardt, 2012)

Das Protokoll wurde so entworfen, dass es nicht nur von Personen, wie Sicherheitsexperten implementiert werden kann. Entworfen wurde es ursprünglich von einer kleinen Gruppe von Webentwicklern, um den sogenannten delegierten Zugriff zu erlauben.

Da es immer mehr Webservices gibt, bei denen persönliche Daten verarbeitet werden können, wurde die Nachfrage nach einem geeigneten Verfahren für solch einen Zugriff immer größer. Die Notwendigkeit von OAuth lässt sich an einem einfachen Beispiel erklären: Immer mehr Menschen nutzen Online-Dienste, um ihre Fotos (Protected Resources) mit Freunden und Familie zu teilen. Nennen wir diesen Dienst (Resource Server) der Einfachheit halber

„meinefotos.de“. Manch einer möchte dann ein paar der Fotos professionell entwickeln lassen, um sie an die heimische Fotowand zu hängen. Dafür gibt es mittlerweile viele Firmen, die solche Dienste online anbieten. Um die Bequemlichkeit zu erhöhen und Zeit zu sparen, indem ein erneutes Hochladen der Fotos verhindert wird, soll dem Fotoentwickler (Client) nun die Möglichkeit gegeben werden, die liebsten Fotos direkt von meinefotos.de abzurufen. Da es sich um zwei getrennte Dienste handelt, benötigt der Fotoentwickler zuvor eine Zugriffsberechtigung. Der Fotoentwickler muss sich also bei meinefotos.de autorisieren. Dafür werden die Zugangsdaten des Benutzerkontos von meinefotos.de benötigt. Aus sicherheitstechnischer Sicht ist es jedoch nicht sinnvoll, wenn der Nutzer seine Zugangsdaten dem Fotoentwickler übergibt, damit dieser sich durch die Zugangsdaten authentifizieren kann. Denn auf diese Weise hätte der Fotoentwickler uneingeschränkten Zugriff auf das gesamte Benutzerkonto. Der Zugriff würde sich nur durch eine Passwortänderung wieder entziehen lassen. Der Nutzer möchte jedoch nur kurzzeitig Zugriff auf seine Fotos gewähren.

Genau an dieser Stelle wird ein Verfahren benötigt, welches einem anderen Client kurzeitig beschränkten Zugriff auf persönliche Daten gewährt, ohne dass dieser Kenntnis über die Zugangsdaten des Nutzers erlangt.

Entwicklung

Diese Problematik erkannten auch andere Entwickler, wie Blaine Cook und Chris Messina. Die Entwicklung von OAuth begann im November 2006. Blaine Cook und Chris Messina arbeiteten zu dieser Zeit an einer Möglichkeit, OpenID mit der Twitter API zu nutzen, um delegierte Authentifizierung zu erlauben. Also an einer Möglichkeit, wie oben im Beispiel, einem Client Zugriffsberechtigung auf Daten bei einem anderen Dienst zu gewähren. Kurz darauf kamen weitere Entwickler wie David Recordon und Larry Halff dazu, um bereits existierende Lösungen zu diskutieren. Sie kamen alle zu dem Schluss, dass es keinen offenen Standard für delegierten Zugriff auf eine API gab. Im April 2007 wurde dann eine Google-Gruppe[2] erstellt, um mit einer kleinen Gruppe von Entwicklern Vorschläge für ein offenes Protokoll zu erarbeiten. Im Juli 2007 entwarf das Team eine Spezifikation, bei der nun jeder seinen Beitrag dazu leisten konnte. 2009 wurde im Kern von OAuth ein Sicherheitsproblem entdeckt, was einem Angreifer durch die sogenannte Session Fixation ermöglichte, auf ein System zuzugreifen, nachdem der Nutzer sich erfolgreich authentifiziert hat. Um dem Sicherheitsproblem entgegen zu wirken, wurde der Kern des Protokolls noch einmal überarbeitet. (vgl. Hammer, 2007)

[2] Google Groups ist ein von Google LLC entwickelter Online-Dienst, mit dem bestimmte Internetforen durchsucht und genutzt werden können. Darüber hinaus können eigene eingerichtete Foren (Groups) über eine Webschnittstelle, per News-Feed oder auch per E-Mail genutzt werden.

Das fertige Protokoll für OAuth 1.0 wurde dann im April 2010 als RFC 5849[3] veröffentlicht. Schon zwei Jahre später wurde das OAuth 2.0 Framework als RFC 4749 veröffentlicht. Da sich mit der Veröffentlichung von OAuth 2.0 einige grundlegende Schritte geändert haben, sind im Folgenden alle weiteren Betrachtungen auf OAuth 2.0 bezogen.

Wie bereits erwähnt ist OAuth 2.0 ein Protokoll, welches sich zur Autorisierung und Authentifizierung nutzen lässt, wobei der Schwerpunkt jedoch in der Autorisierung liegt.

Aufbau und Funktionsweise

Das OAuth Protokoll definiert vier Rollen, die miteinander interagieren. Die erste Rolle ist der Resource Owner, welcher dazu fähig ist, Zugriff auf geschützte Ressourcen zu gewähren. Meistens ist das der Nutzer selbst. Der Resource Server hingegen stellt die geschützten Ressourcen bereit und liefert diese auf Anfrage aus, wenn ein gültiger Access Token vorhanden ist. Er stellt die zweite Rolle dar. Die dritte Rolle übernimmt der Client. Das kann jede Art von Anwendung sein, die im Namen des Nutzers auf die geschützten Daten zugreifen kann. Im Webbereich ist es die Webanwendung. Die vierte und letzte Rolle ist der Authorization Server. Dies ist der Login-Server, der den Access Token für den Client bereitstellt, nachdem sich der Nutzer gegenüber dem Authorization Server authentifiziert und den Zugriff auf seine Daten zugestimmt hat. Die vier Rollen werden in Abbildung 2.3 veranschaulicht.

[3] RFC (Request for Comments) besteht aus einer Sammlung von nummerierten Dokumenten, die von der IETF (Internet Engineering Task Force) veröffentlicht werden, um Standards schneller durchzusetzen.

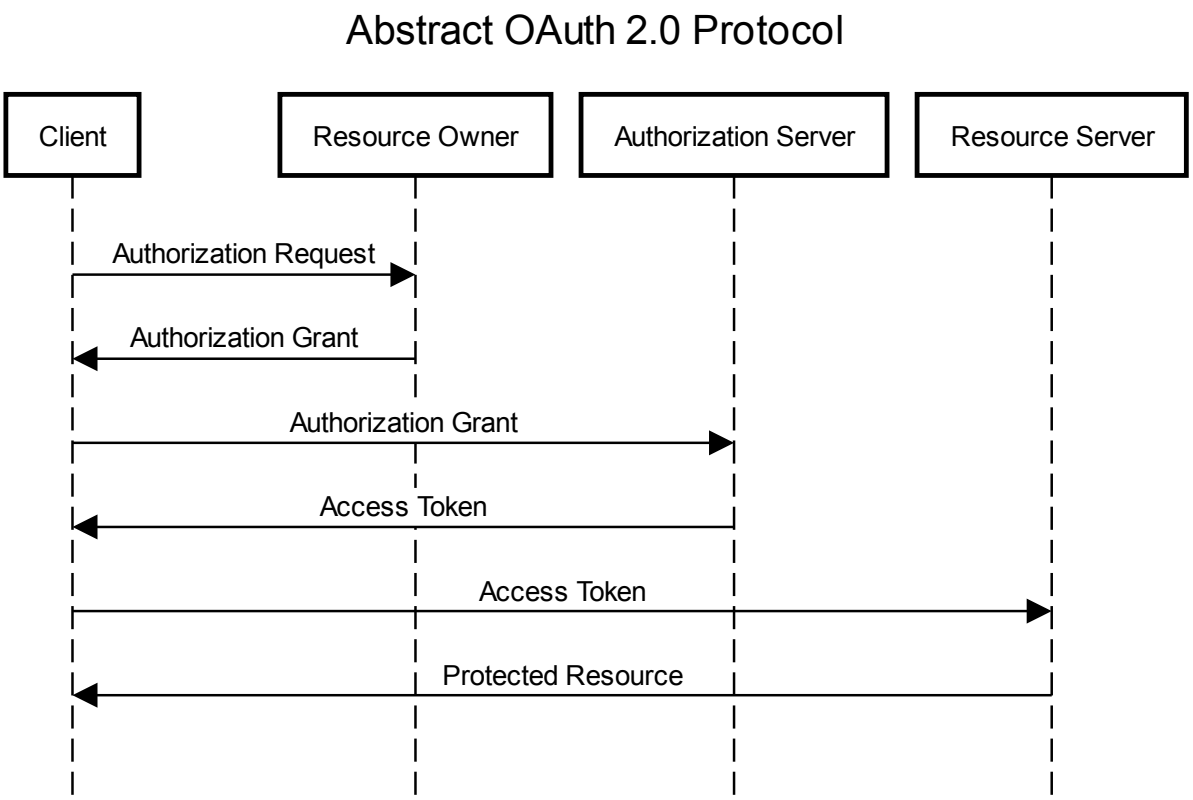


Abbildung 2.3 Kommunikationsablauf des OAuth 2.0 Protokolls in abstrakter Darstellung

Das OAuth Protokoll ist sehr abstrakt definiert. Es gibt zum Beispiel keine Vorgaben, ob der Resource Server und der Authorization Server ein gemeinsamer oder zwei voneinander getrennte Server sind.

Die Abbildung 2.3 zeigt auch den Kommunikationsverlauf zwischen den Rollen. Der Client stellt zunächst eine Autorisierungsanfrage an den Resource Owner. Das kann mit einer direkten Interaktion oder indirekt durch den Authorization Server geschehen. Nach der Zustimmung des Nutzers, erhält der Client die Genehmigung des Nutzers (Authorization Grant). Dabei gibt es verschiedene Typen der Genehmigung, je nachdem wie der Client die Anfrage stellt und welche Typen von beiden Parteien unterstützt werden. Danach sendet der Client eine Anfrage an den Authorization Server, bei dem sich der Nutzer authentifizieren muss. Daraufhin wird die Genehmigung des Nutzers überprüft und der Client erhält einen Access Token als Antwort. Mit diesem Token hat der Client nun einen Zugangsschlüssel, um auf die geschützten Ressourcen des Resource Servers zuzugreifen. Nach einer Anfrage überprüft der Resource Server zunächst die Gültigkeit des Access Tokens und nutzt diesen als Nutzerauthentifizierung.

2.2.2 OpenID

OpenID ist ein Protokoll zur dezentralen Authentifikation. Es ist ein offener Standard und wurde von der OpenID Foundation gefördert. Durch OpenID bekommt der Nutzer die Möglichkeit, seinen eigenen Login-Server zu wählen, der hier OpenID-Provider genannt wird. Es besteht weiterhin die Möglichkeit einen vorhandenen oder eigens betriebenen OpenID-Provider zu nutzen. Die Nutzeridentität wird nur durch die OpenID bestimmt. Der OpenID Standard wurde jedoch bereits vollständig durch OpenID Connect abgelöst und auch von der OpenID Foundation selbst als veraltet markiert.

Entwicklung

OpenID wird von der OpenID Foundation gefördert. Es wurde 2005 durch eine Open Source Community erstellt, die ein Verfahren benötigten, welches es erlaubt, sich bei verschiedenen Webseiten mit einem einzigen Nutzerkonto anzumelden. Eine weitere Anforderung war, dass dieses Verfahren dezentralisiert und von niemanden besessen werden sollte. Jeder könnte kostenlos OpenID verwenden bzw. OpenID-Anbieter sein. Mit OpenID ist es somit nicht mehr notwendig einen Account für jede Webseite zu erstellen. Der Nutzer entscheidet lediglich, welche Informationen über den Nutzer mit der jeweiligen Webseite geteilt werden sollen. Darüber hinaus gibt es für die Webseiten keine Möglichkeit an die Zugangsdaten des Nutzers zu gelangen.

Aufbau und Funktionsweise

Wenn ein Nutzer sich auf einer Webseite (Relying Party) anmelden möchte, wird er zu einem Login-Server (OpenID Provider) geleitet, welcher OpenID anbietet. Zuvor muss der OpenID-Provider jedoch ermittelt werden. Der Nutzer gibt dazu seine gewünschte OpenID ein.

Die OpenID-Identität besteht aus einer URL-basierten Identität, welche wie eine der folgenden aussehen könnte (vgl. Dewanto, 2009):

OpenID-Identität

```
http://name.domain.de/
http://domain.de/name/
```

Abbildung 2.4 OpenID-Identität in Form einer URL

In der Regel ist der Nutzername eine Subdomain des OpenID-Anbieters. Diese URL zeigt zu oder ist der OpenID Authentifizierungsserver des Nutzers. Ein Nutzer der die URL besitzt, kann über die Metadaten seiner eigenen Webseite festlegen, auf welchen OpenID Server

verwiesen werden soll. Somit kann der Nutzer jederzeit frei entscheiden, welchen OpenID Provider er nutzen möchte. Dafür müssen lediglich folgende HTML Metadaten angegeben werden:

Variable OpenID-Identität

```
<link rel="openid2.provider" href="https://openid.anderedomain.de /" />
<link rel="openid2.local_id" href="https://name.anderedomain.de /" />
```

Abbildung 2.5 HTML-Based Discovery zur Bestimmung der OpenID-Identität mit HTML Metadaten

Dieses Verfahren wird *HTML-Based Discovery* genannt. Es gibt zwei weitere Möglichkeiten, um die für die Relying Party benötigten Informationen zu liefern: Das XRI-Auflösung[4] und das Yadis Protokoll[5]. Die beiden letzten Verfahren erzeugen am Ende ein XRDS Dokument, welches auf XML basiert. (vgl. Stepka, 2007)

Die Relying Party nutzt die URL, um den OpenID Authentifizierungsserver zu finden. Die Eingaben müssen jedoch zuerst in ein entsprechendes Format gebracht werden (Normalization). Danach werden der OpenID-Provider und die benötigten Parameter ermittelt (Discovery). Im nächsten, aber optionalen Schritt, kann die Relying Party mit dem OpenID-Provider ein geheimes Passwort aushandeln. Es wird dazu verwendet, Nachrichten zu signieren bzw. die Verbindung abzusichern und weitere kommunikationsspezifische Parameter auszutauschen. Nun wird der Nutzer zum OpenID-Provider umgeleitet. Dort muss sich der Nutzer gegebenenfalls mit seinen Zugangsdaten, also Benutzername und Passwort, anmelden, um die Identität des Nutzers gegenüber der Webseite zu bestätigen. Der Nutzer wird dann zurück zur Relying Party umgeleitet. Im Anschluss daran prüft die Relying Party die erhaltene Authentifizierungsnachricht. Der Nutzer ist nun erfolgreich angemeldet. Die folgende Abbildung verdeutlicht noch einmal die eben beschriebenen Schritte:

[4] Siehe auch https://www.oasis-open.org/committees/download.php/17293 (aufgerufen am 20.05.2018)

[5] Siehe auch http://openid.net/specs/yadis-v1.0.pdf (aufgerufen am 20.05.2018)

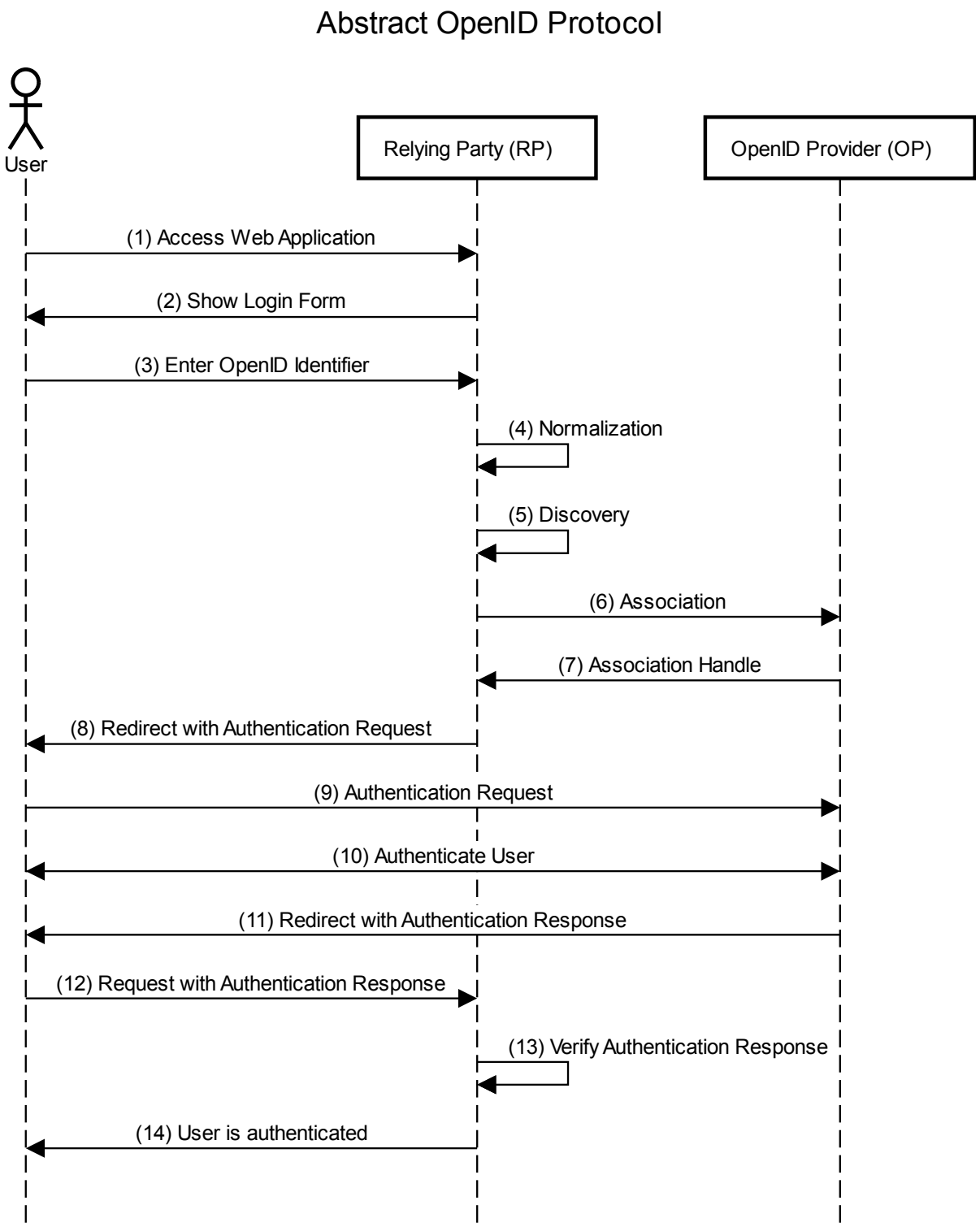


Abbildung 2.6 Kommunikationsablauf des OpenID Protokolls in abstrakter Darstellung

Wie sich sicherlich schon erahnen lässt, kann OpenID als eine Single Sign-on Lösung genutzt werden. Dadurch ist es Web-Diensten möglich, das Anmeldeverfahren durch einen fremden Server durchführen zu lassen, was *Delegation* genannt wird. Weiterhin ist es dennoch möglich ein herkömmliches Anmeldeverfahren parallel zu verwenden. (vgl. OpenID Foundation, 2007)

2.2.3 OpenID Connect (OIDC)

OpenID Connect ist eine standardisierte und auf OAuth 2.0 basierende Authentifizierungsschicht. Wie auch OpenID wird diese von der OpenID Foundation kontrolliert. Der entscheidende Unterschied zu OpenID besteht darin, dass OAuth 2.0 als Grundlage dient. Die Arbeitsweise ist somit im Vergleich zu OpenID verschieden. Der OpenID Standard wurde, wie bereits erwähnt, vollständig durch OpenID Connect abgelöst und auch von der OpenID Foundation selbst als veraltet markiert.

Anders als bei OAuth 2.0 wird OpenID Connect nur zur Authentifizierung genutzt, während OAuth 2.0 zur Autorisierung und Authentifizierung verwendet wird. Wie für Single Sign-on Systeme üblich, ist der Einsatzweck von OIDC der Zugriff auf mehrere Webanwendungen mit nur einer einzigen Anmeldung. Durch OAuth 2.0 erhält OIDC die Möglichkeit, die Authentifizierung durch externe Services, wie zum Beispiel Google Inc., Facebook Inc., Amazon Inc. usw. durchführen zu lassen.

Die Authentifizierung wird delegiert und ist somit dezentralisiert. Auf diese Weise lassen sich weitere *Authorization Server* einfach einbinden. Durch den dezentralisierten Aufbau erhöht sich die Verfügbarkeit des gesamten Systems. Fällt ein *Authorization Server* aus, kann der Nutzer einen anderen wählen, sofern er bereits ein Benutzerkonto bei diesem besitzt. Durch die erweitere Spezifikation von OAuth 2.0 lassen sich nicht nur Webanwendungen mit OpenID Connect nutzen. Auch JavaScript Clients und mobile Anwendungen lassen sich damit authentifizieren. OpenID Connect erweitert das OAuth 2.0 Framework. Funktionen wie zusätzliche Verschlüsselung, Verwalten von Sessions oder das Finden von OpenID-Providern machen OpenID Connect ideal für den Einsatz als Single Sign-on Lösung.

Entwicklung

Im Februar 2014 wurde OpenID Connect als neuer Standard von der OpenID Foundation veröffentlicht. Es ist der offizielle Nachfolger von OpenID 1.1 und OpenID 2.0. Die Entwicklung des neuen Protokolls sollte mehr Möglichkeiten bieten und auch einfacher für Entwickler zu verwenden sein. Führende Größen u.a. wie Deutsche Telekom, Facebook, Google und Microsoft arbeiteten daran mit. (vgl. Thibeau, 2014)

Aufbau und Funktionsweise

OpenID Connect besteht aus drei Parteien: der Nutzer (User), die Webanwendung bzw. der Client (Relying Party) und der Login-Server (OpenID Provider). Der Kommunikationsablauf des Protokolls lässt sich folgendermaßen abstrakt beschreiben:

Ein Nutzer greift auf die Webanwendung, der Relying Party, zu. Möchte der Nutzer auf gesicherte Daten zugreifen, muss er sich zuvor anmelden. Dazu gibt der Nutzer seine gewünschte

OpenID ein. Danach sendet die Relying Party eine Authentifizierungsanfrage an den eingegebenen OpenID Provider, indem der Nutzer zu diesem umgeleitet wird. Dann muss sich der Nutzer mit seinen Zugangsdaten beim OpenID Provider anmelden, wenn er nicht bereits angemeldet ist. Im Anschluss daran erscheint eine Meldung, bei welcher der Nutzer gefragt wird, seine Informationen mit der anfragenden Relying Party zu teilen. Nach Zustimmung wird der Nutzer zurück zur Relying Party umgeleitet. Bei allen Anfragen und Antworten werden bestimmte Parameter übertragen, welche die Parteien und die Kommunikation beschreiben. Der Nutzer hat bei der Umleitung zurück zur Relying Party einen Autorisierungscode bekommen. Die Relying Party verwendet nun diesen Code, um vom OpenID Provider einen ID und einen Access Token zu erhalten. Der ID Token, welcher die Identität des Nutzers beschreibt, wird daraufhin von der Relying Party validiert. Mit dem Access Token können die Nutzerdaten vom OpenID Provider erfragt werden. Der ID Token verwendet das sogenannte JSON Web Tokens (JWT), wodurch sich die Daten zusätzlich signieren lassen, um die Integrität gegenüber dem Kommunikationspartner zu wahren. (vgl. Sakimura, et al., 2014)

Das nachstehende Sequenzdiagramm veranschaulicht noch einmal den Kommunikationsablauf von OpenID Connect in abstrakter Darstellung.

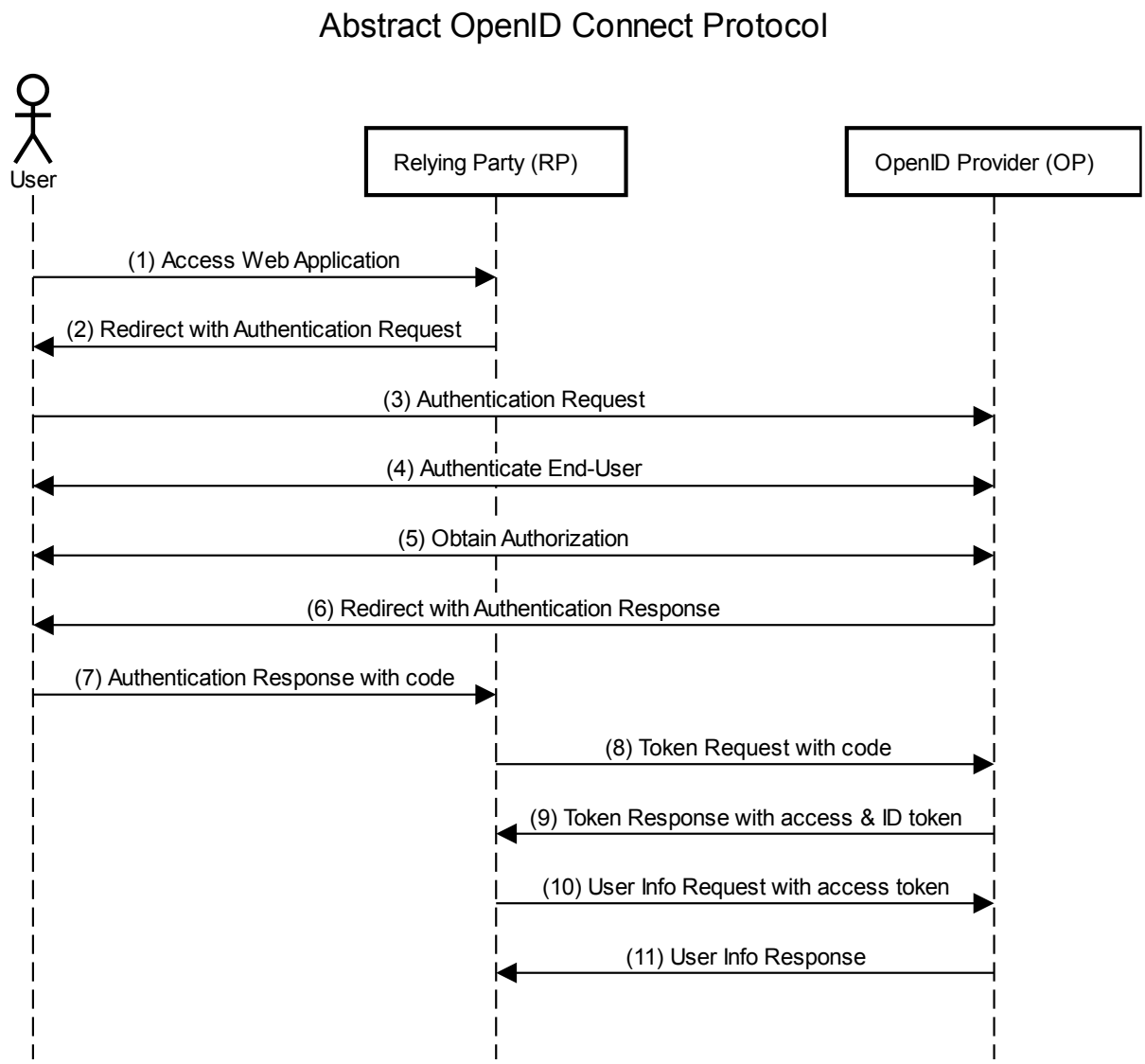


Abbildung 2.7 Kommunikationsablauf des OpenID Connect Protokolls in abstrakter Darstellung

2.2.4 Security Assertion Markup Language (SAML)

SAML ist ein XML-basiertes Framework. Es definiert einen offenen Standard, um Authentifizierungs- und Autorisierungsinformationen auszutauschen. Dabei werden auch alle sicherheitsbezogenen Informationen in einem XML-Dokument beschrieben.

Entwicklung

SAML wurde 2001 vom OASIS-Konsortium entwickelt, welches ein Zusammenschluss von Oracle, IBM, Nokia und SAP war. Im November 2002 wurde dann der OASIS Standard SAML 1.0 eingeführt. Schon ein Jahr darauf wurde im September 2003 SAML 1.1 veröffentlicht. Seit März 2005 ist SAML 2.0 der aktuelle Standard. Die letzten freigegebenen Änderungen (Errata) erfolgten im Mai 2012 (vgl. Cantor, 2012). Mit der Entwicklung zu SAML 2.0 stieg nicht nur

die Anzahl der involvierten Firmen, sondern auch der Funktionsumfang, um noch mehr Anwendungsfälle abzudecken. Die Vorteile von SAML sind vielfältig: SAML erlaubt unabhängige Entwicklung zwischen der Anwendung und dem Sicherheitssystem. Es bietet eine Reihe von standardisierten Schnittstellen und ermöglicht so die Interoperabilität zwischen verschiedenen Anwendungen unterschiedlicher Technologien. Hierdurch lassen sich komplexe Anwendungen schneller, einfacher, sicherer und zuverlässiger entwickeln. Aufgrund der Vielfalt an Anwendungsfällen, der Einfachheit, Sicherheit und den immer mehr verfügbaren Online-Diensten erfreut sich SAML wachsender Beliebtheit. Das zeigt auch eine Statistik von *Drupal*, einem Content-Management-System (CMS). (vgl. drupal.org - Dries Buytaert, 2018) Drupal bietet verschiedene Module an, um die Funktionalitäten beliebig wählen zu können. Über die Verwendung solcher Module lassen sich Statistiken erstellen. Die Verwendung des SAML-Moduls zeigt eine rasant ansteigende Beliebtheit seit Anfang 2017, bei der das Modul schon bereits seit 2 Jahren existierte. Bei derzeitigem Stand verwenden wöchentlich über 800 Projekte das SAML Modul, wohingegen es Anfang 2017 noch weniger als 50 waren. (vgl. carolgeyer, 2007)

Aufbau und Funktionsweise

SAML besteht aus drei Hauptkomponenten: der Nutzer, der Identity Provider (IdP) und der Service Provider (SP). Der Identity Provider speichert benutzerspezifische Informationen, wie zum Beispiel Benutzername, Passwort, Rollen und andere Metadaten. Er wird auch als SAML Asserting Party bezeichnet. Der Service Provider wird auch als SAML Relying Party bezeichnet. Die Bezeichnung rührt daher, dass durch SAML Inhalte bereitgestellt werden, denen der Service Provider vertraut.

SAML wurde entwickelt, um spezielle Anwendungsfälle abzudecken. Dazu gehörten Autorisierungsdienste, die den Zugriff auf Webdienste für Nutzer beschränken und vor allem Single Sign-on. Im Falle von Single Sign-on authentifiziert der Identity Provider einen Nutzer gegenüber eines Service Providers, welche z.B. die Webanwendung repräsentiert. Dabei definiert SAML nicht, wie genau die Authentifizierung beim Identity Provider zu erfolgen hat. Somit kann der Entwickler selbst bestimmen, ob dabei typischerweise Benutzername und Passwort verwendet werden oder eben ein anderes Verfahren. Sensible Daten wie Benutzername und Passwort werden auch bei SAML nicht zwischen Anwendungen ausgetauscht, sondern direkt an einer Stelle beim Identity Provider.

Das SAML-Framework besteht weiterhin aus vier Konzepten: Profile, Bindings, Protokolle und Assertions. Diese bestehen aus ineinander verschachtelten XML-Containern, die ihre jeweiligen Dateninhalte beschreiben.

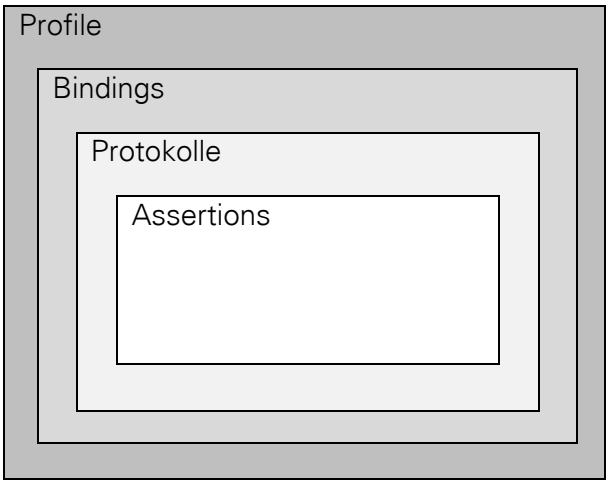


Abbildung 2.8 SAML XML-Container zur Beschreibung der Dateninhalte. Der Container ist in verschiedenen und ineinander verschachtelten Ebenen aufgeteilt.

In Abbildung 2.8 werden die ineinander verschachtelten XML-Containern bildlich dargestellt. Der innerste Container enthält die sogenannten Assertions. Sie beschreiben Eigenschaften des Nutzers und werden durch den Identity Provider erstellt. Zusätzlich wird eine digitale Signatur mitgeliefert, um die Echtheit der Informationen zu garantieren. Der Service Provider prüft dabei die digitale Signatur (DSIG), um die Authentizität und Integrität des SAML-Tokens zu verifizieren. Nach der Verifizierung wird der Inhalt des Containers analysiert, um diesbezüglich Entscheidungen treffen zu können, ob dem Nutzer Zugriff gewährt wird oder nicht. (vgl. Krafzig, et al., 2015)

Assertion Typ	Beschreibung
Attribute Statements	Enthalten spezifische Informationen, um den Benutzer zu identifizieren.
Authentication Statements	Informieren Service Provider darüber, dass der Nutzer durch Identity Provider authentifiziert wurde. Weiterhin sind zusätzliche Informationen, wie welche Methode angewandt wurde und zu welchem Zeitpunkt authentifiziert wurde, enthalten. Dazu kommen weitere mögliche Authentifizierungsinformationen.
Authorization Decision Statements	Enthalten Informationen darüber, auf welche Ressourcen der Nutzer zugreifen darf.

Tabelle 2.1 Beschreibung der verschiedenen Assertion Typen innerhalb einer SAML-Nachricht. (vgl. Krafzig, et al., 2015) und (vgl. OASIS Open - Core, 2005)

Es gibt drei Assertions-Typen, welche unterschiedliche Aufgaben übernehmen: Attribute Statements, Authentication Statements und Authorization Decision Statements. In der Tabelle 2.1 werden deren Aufgaben näher erläutert.

Protokoll Typ	Beschreibung
Artifact Resolution Protocol	Ermöglicht Referenzen auf SAML-Nachrichten anstatt die Nachrichten an sich zu versenden, um dadurch beispielsweise Datenoverhead zu minimieren. Wird verwendet, wenn z.B. SAML-Nachrichten über einen separaten sicheren Kanal übertragen werden sollen oder um Referenzen aufzulösen und die ursprüngliche komplette Nachricht anzufordern.
Assertion Query and Request Protocol	Dient der Anfrage von Assertions über Referenzen. Kann auch dafür genutzt werden, um nach Assertions zu suchen, indem Suchparameter mitgeliefert werden.
Authentication Request Protocol	Wird verwendet, um vom Identity Provider Assertions über den Nutzer zu erhalten.
Name Identifier Management Protocol	Wird verwendet, um den Identifier des zugehörigen Nutzers zu verändern. Dabei können nicht nur Werte, sondern auch das Format verändert werden. Kann sowohl vom Service Provider als auch vom Identity Provider verwendet werden.
Name Identifier Mapping Protocol	Wird vom Service Provider genutzt, um auf weitere Service Provider zuzugreifen, indem der Identifier des Nutzers vom Identity Provider abgefragt wird.
Single Log-out Protocol	SAML definiert eine Single Log-out Strategie, um alle Sessions eines Nutzers zu beenden. Der Vorgang kann nicht nur vom Nutzer selbst, sondern auch vom Service Provider und vom Identity Provider ausgelöst werden.

Tabelle 2.2 Beschreibung der Protokolltypen, die durch das SAML Protokoll spezifiziert werden. (vgl. Krafzig, et al., 2015) und (vgl. OASIS Open - Core, 2005)

Die Assertions werden im Container für Protokolle gekapselt. Die SAML-Protokolle dienen dem Informationsaustausch zwischen Service Provider und Identity Provider. Im derzeitigen SAML 2.0 Core werden sechs verschiedene SAML-Protokolle detailliert beschrieben, welche verschiedene Funktionen besitzen. Auch hier soll die Tabelle 2.2 die Funktionsweisen zusammenfassen.

Die Bindings entscheiden darüber, wie die SAML Nachrichten transportiert werden sollen. Dabei sind nur Werte gültig, die im SAML-Protokoll stehen, wie zum Beispiel die Übertragung per HTTP-POST Methode.

Der äußerste Container enthält die SAML Profile. Sie vereinen die Assertions, Protokolle und Bindings für bestimmte Einsatzwecke. Ein Profil beschreibt also, wie diese Konzepte miteinander kombiniert werden. Es gibt eine ganze Reihe verschiedener SAML-Profile. Das interessanteste für diese Arbeit und auch gleichzeitig wichtigste Profil von SAML ist das *Web Browser SSO Profile*, welches u.a. das Authentication Request Protocol verwendet.

Trotz des sehr komplexen Aufbaus des SAML Protokolls kann das Web Browser SSO Profil direkt dazu genutzt werden, um eine Single Sign-on Lösung zu implementieren. Zu beachten ist, dass das SSO Profil selbst auch wieder verschiedene Arbeitsweisen erlaubt. Der häufigste und einfachste Fall lässt sie wie folgt beschreiben. Ein Nutzer möchte auf geschützte Ressourcen zugreifen, die sich beim Service Provider befinden. Der Service Provider erhält die Anfrage des Nutzers und stellt nun eine Authentifizierungsanfrage an den Identity Provider, um die Identität des Nutzers zu erfahren.

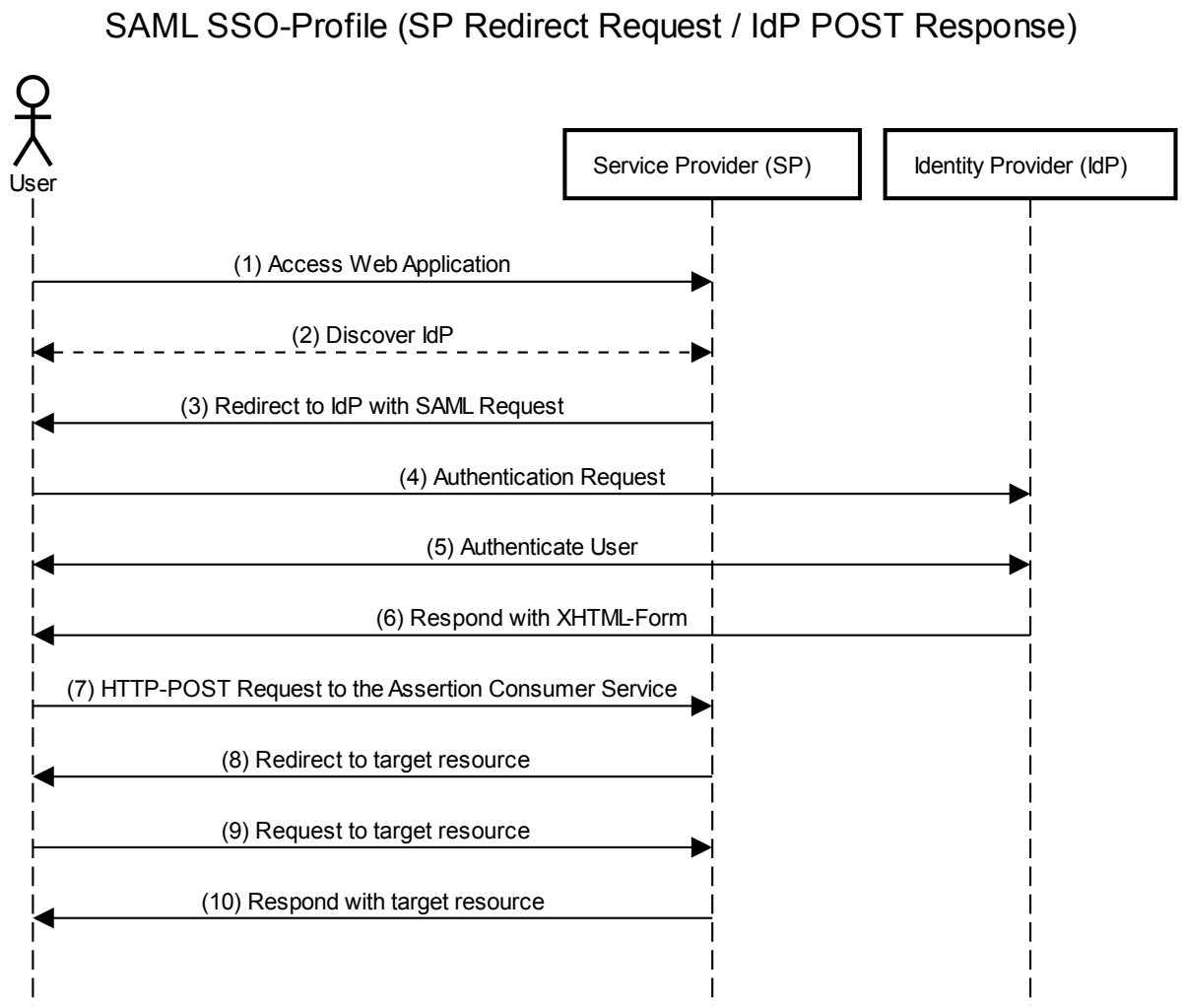


Abbildung 2.9 Kommunikationsablaufs des SAML Protokolls im SSO-Profile (SP Redirect Request / IdP POST Response) (vgl. OASIS Open - Profiles, 2005)

Wenn der Nutzer nun vom Service Provider per HTTP-Redirect zum Identity Provider umgeleitet wird und dieser wiederrum per HTTP-POST antwortet, lässt sich dieser Fall auch als *SP Redirect Request / IdP POST Response* bezeichnen. Diese Art der Kommunikation kommt dem Kommunikationsablauf von anderen Verfahren, wie zum Beispiel CAS, am nächsten.

Nach der Authentifizierung des Nutzers beim Identity Provider erhält der Service Provider die Antwort auf die zuvor gestellte Anfrage. Nach der Auswertung der Antwort erhält der Nutzer Zugriff auf die ursprünglich angeforderte Ressource, indem er auf diese umgeleitet wird. In Abbildung 2.9 wird der beschriebene Ablauf noch einmal bildlich dargestellt.

2.2.5 Central Authentication Service (CAS)

Ein weiteres und relativ simples Single Sign-on Protokoll trägt den Namen *Central Authentication Service (CAS)*. Es erlaubt einem Nutzer auf mehrere Webanwendungen zuzugreifen, wobei dieser wieder nur ein einziges Mal seine Zugangsdaten eingeben muss, um sich anzumelden. Die Zugangsdaten bestehen aus dem Benutzernamen und einem Passwort. Das Protokoll stellt auch hier sicher, dass es einer Webanwendung nicht möglich ist, an die Zugangsdaten des Nutzers zu gelangen, da dieser zum Anmelden auf einen Login-Server umgeleitet wird.

Entwicklung

Das Central Authentication Protokoll wurde von Shawn Bayern von der *Yale University* entwickelt. Es wird aus diesem Grund auch „Yale CAS" genannt. Die erste Protokollversion wurde speziell für den Einsatz als Single Sign-on System entwickelt, um universitätsintern genutzt zu werden. Später wurde das Protokoll weiter ausgebaut, um auch Webanwendungen durch einen Proxy-Server zu nutzen. Dabei erfolgt die Authentifizierung durch den Proxy. Der Nutzer kommuniziert indirekt mit der Webanwendung über den Proxy. Die neuen Funktionen wurden mit dem CAS 2.0 Protokoll veröffentlicht. Mittlerweile existieren mehrere Distributionen, welche unter anderem das CAS Protokoll unterstützen. CAS wurde im Dezember 2004 zum Projekt der *Java in Administration Special Interest Group (JASIG).* 2012 haben sich die *Sakai Foundation* und die *JASIG* unter den Namen *Apereo Foundation* zusammengeschlossen, um zusammen das CAS-Projekt fortzuführen. Die Spezifikation für die Protokollversion 3.0 wurde dann 2014 veröffentlicht. Mittlerweile implementiert Projekt *Apereo CAS* verschiedene Protokolle wie alle CAS Protokolle, SAML, OAuth, OpenID, OpenID Connect und viele weitere (vgl. Apereo Foundation - CAS , 2018). Somit ist das Projekt zu einer großen mehrsprachigem Single Sign-on Lösung geworden, welches zusätzlich zur Authentifizierung auch Autorisierung unterstützt. Apereo CAS ist Open Source und die ursprüngliche Implementierung ist in Java geschrieben. (vgl. Battaglia, 2006)

Aufbau und Funktionsweise

Zum CAS Protokoll gehören mindestens eine Webanwendung *(Applikation)*, ein Webbrowser *(Client)* und ein Login Server *(CAS Server).* Dabei ist der CAS Server im Prinzip auch eine Webanwendung, welche die Anmelde- und Authentifizierungsfunktion übernimmt. Alle Parteien kommunizieren über das zustandslose Übertragungsprotokoll HTTP.

Ein System, welches CAS implementiert, arbeitet wie folgt: Der Nutzer ruft die Webanwendung mit einem Webbrowser auf. Sie prüft zunächst, ob der Nutzer die Webanwendung vor kurzem genutzt hat und für diesen bereits eine Session besteht. Ist dies nicht der Fall, wird

der Nutzer zum CAS Server weitergeleitet. Bei der Umleitung wird die Adresse der Webanwendung an die URL gehangen, damit der Nutzer nach der Authentifizierung wieder zur richtigen Stelle zurückgeleitet werden kann. Der CAS Server prüft seinerseits ebenfalls, ob für den Nutzer bereits eine Session besteht. Wenn nicht, wird ihm ein Login Formular angezeigt. Der Nutzer gibt dann seinen Benutzernamen und sein Passwort ein und sendet das Formular ab. Zurück beim CAS Server werden die eingegebenen Zugangsdaten mit einer angebundenen Datenbank überprüft. Sind sie gültig, wird für den Nutzer eine Session erstellt und er ist damit authentifiziert. Nun erfolgt die Umleitung zurück zur Webanwendung. Dieses Mal wird ein Token dem sogenannten Service Ticket (ST) an die URL angehangen. Die Webanwendung empfängt den Token und sendet von sich aus eine Anfrage an den CAS Server, ob der Token valide ist. Der CAS Server antwortet mit ja oder nein und mindestens dem Nutzernamen. Weitere Attribute sind optional. Ist der Token gültig, eröffnet auch die Webanwendung eine Session. Der Nutzer ist nun gegenüber der Webanwendung authentifiziert und erfolgreich angemeldet. Er hat jetzt Zugriff auf die geschützten Daten. (vgl. Battaglia, 2006)

Der beschriebene Ablauf entspricht dem erstmaligen Aufruf der Webanwendung eines spezifischen Nutzers. Im Falle eines zweiten Aufrufes prüft die Webanwendung, ob noch eine gültige Session des Nutzers existiert. Denn dann ist es nicht notwendig, den CAS Server zu kontaktieren bzw. den Nutzer dahin umzuleiten.

Wird eine andere Webanwendung verwendet, welche sich ebenfalls in demselben CAS System befindet, verläuft der Anmeldevorgang schneller als zuvor. Nachdem der Nutzer zum CAS Server geleitet wurde, weil zum Beispiel keine gültige Session für den Nutzer in der Webanwendung existierte, wird dieses Mal eine gültige Session erkannt. Der Nutzer muss also seine Zugangsdaten nicht erneut eingeben, sondern wird direkt mit einem neuen Service Token zur Webanwendung zurückgeleitet. Ab diesen Punkt verhält sich der weitere Verlauf wie zuvor beschrieben und der Nutzer ist angemeldet. Dieser Vorgang kann so schnell ablaufen, dass der Nutzer davon in der Regel nichts bemerkt. Die zusätzliche Verzögerung bewegt sich im Millisekunden-Bereich.

2.2.6 Weitere Entwicklungen

Die gewählten Single Sign-on Lösungen sind nur eine kleine Auswahl. Viele weitere Technologien sind oftmals weniger bekannt, da sie zu speziell oder veraltet sind. Weitere Gründe können Lizenzbedingungen bzw. -kosten darstellen, wie es auch bei Atlassian Crowd[6] der Fall ist. Auch kann eine aufwändige Konfiguration und Installation für eine geringere Beliebtheit sorgen. Andere wiederrum, wie Shibboleth[7], basieren auf SAML und sind vor allem im Bereich der Wissenschaft beliebt.

[6] Siehe auch https://de.atlassian.com/software/crowd/pricing (aufgerufen am 25.06.2018)
[7] Siehe auch https://www.shibboleth.net/ (aufgerufen am 25.06.2018)

2.3 AMCS

Auditorium Mobile Classroom Service (AMCS) ist ein sogenanntes *Audience Response System*, welches von der TU Dresden entwickelt wird. Solche Systeme werden zum Beispiel in Vorlesungen bzw. in Lehrveranstaltungen eingesetzt, um die Interaktivität zwischen dem Dozenten und den Zuhörern zu steigern. Dabei wird AMCS von Studierenden und Lehrenden beider maßen genutzt. Durch AMCS können verschiedene Kurse bereitgestellt werden. Zu jedem Kurs können mehrere Lehrveranstaltungen existieren. Lehrveranstaltungen können hierbei Vorlesungen, Übungen oder Seminare sein. AMCS bietet eine anonyme Nutzung an. Der Nutzer meldet sich mit einem selbst gewähltem Pseudonym und einem Passwort an, wodurch ein gesichertes Nutzerkonto erstellt wird. Auf diese Weise ist es zum Beispiel möglich, Antworten von Studenten einem eindeutigen Pseudonym zuordnen zu können. Lehrende Nutzer verfügen über besondere Rechte, um Kurse und dazugehörige Lehrveranstaltungen erstellen zu können. Jeder Kurs besitzt weiterhin eine PIN, damit andere Nutzer dem Kurs beitreten können. Mithilfe der Kurse können den Studierenden verschiedenen Fragen gestellt werden. Deren Antworten werden dann automatisch ausgewertet und grafisch dargestellt. Außerdem können die Nutzer Nachrichten erhalten und der Lehrende kann direktes Feedback erhalten. Zudem ist es möglich, Kursinhalte auch zur Nachbereitung zu nutzen. (vgl. AMCS - TU Dresden, 2018)

2.3.1 Aufbau

Im Folgenden wird der Aufbau und die Funktionsweise von AMCS im Groben erklärt. Auf genauere Details wird hierbei verzichtet, da sich diese Arbeit nur auf die Nutzerauthentifizierung beschränkt. Weitere Informationen zu den verwendeten Technologien finden sich unter den weiterführenden Verlinkungen.

AMCS besteht grundlegend aus zwei Teilen, dem Front-End und dem Back-End. Das Back-End stellt das Herzstück von AMCS dar. Es beinhaltet die gesamte Datenverwaltung und -speicherung. Zur Ansteuerung wird eine REST-API implementiert. Daher ist es möglich, verschiedene Front-End Anwendungen einzusetzen. Die wohl wichtigste Front-End Anwendung ist die AMCS Webanwendung. Daher beziehen sich alle weiteren Betrachtungen zum Front-End auf die Webanwendung.

Das Front-End ist in *Angular* [8] geschrieben, einem *TypeScript* basiertem Webanwendungsframework. Mithilfe von *Node.js* [9] und dem Paketmanager *NPM* [10], wird das Framework bereitgestellt. Das Back-End hingegen ist in *Ruby on Rails* [11] geschrieben, einem *Ruby* basierten

[8] Siehe auch https://angular.io/ (aufgerufen am 21.06.2018)
[9] Siehe auch https://nodejs.org (aufgerufen am 21.06.2018)
[10] Siehe auch https://www.npmjs.com/ (aufgerufen am 21.06.2018)
[11] Siehe auch https://rubyonrails.org/ (aufgerufen am 21.06.2018)

Webframework, und übernimmt die Aufgabe der Datenverwaltung von AMCS. Zum Einsatz kommt eine vorkonfigurierte virtuelle Maschine, welche u.a. sowohl Ruby, als auch den Webserver *nginx* [12] und das Datenbankmanagementsystem *PostgreSQL* [13] bereitstellt.

Das Front-End besteht aus verschiedenen Modulen, die wiederrum Komponenten enthalten und mithilfe von HTML-Templates Webseiten erzeugen. Aufgebaut ist das Front-End ähnlich dem *Model-View-Controller-Prinzip*, wobei das Model durch das Back-End übernommen wird. Durch die Verwendung von Angular stellt das Front-End eine sogenannte *Single-Page-Anwendung* bereit. Das bedeutet, dass der Webseiteninhalt dynamisch ausgetauscht werden kann.

2.3.2 Aktueller Stand

Der aktuelle Stand betrachtet die Teile, die der Nutzerauthentifizierung zugehörig sind. Die beiden, für die Integrierung einer Single Sign-on Lösung, wichtigsten Module sind das Home Modul (Abbildung 2.10) und das Login Modul (Abbildung 2.11).

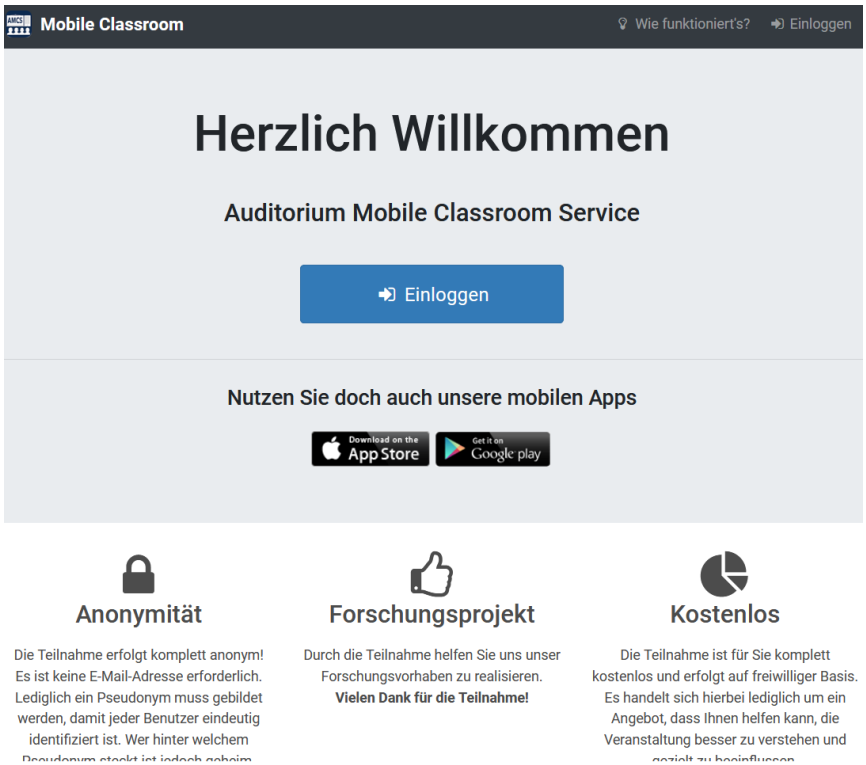


Abbildung 2.10 Darstellung der AMCS Webseite durch das Home Modul

Auf der resultierenden Seite des Home Moduls befindet sich ein blauer Button und ein Link oben rechts, welche beide zur Seite des Login Moduls führen.

[12] Siehe auch https://www.nginx.com/ (aufgerufen am 21.06.2018)
[13] Siehe auch https://www.postgresql.org/ (aufgerufen am 21.06.2018)

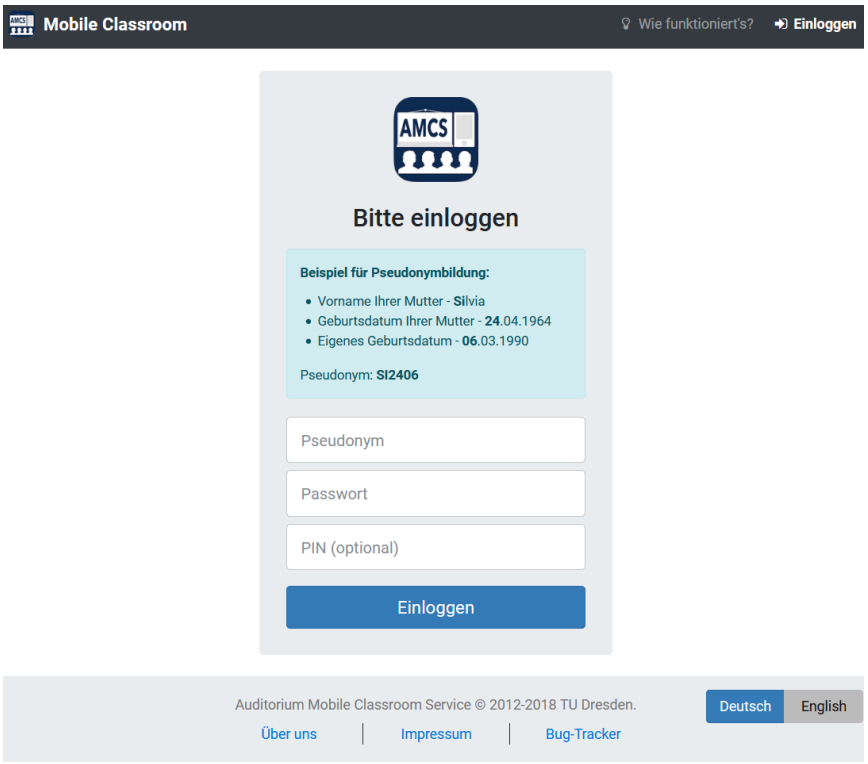


Abbildung 2.11 Darstellung der AMCS Webseite durch das Login Moduls

Auf der Seite des Login Moduls wird die Anmeldeoberfläche von AMCS dargestellt. Durch die Eingabe der Nutzerdaten und mit einem Klick auf „Einloggen“ werden die Daten zur Login Komponente gesendet. Diese sendet dann mithilfe einer HTTP-POST-Anfrage das Pseudonym und Passwort zum Back-End von AMCS und wartet auf eine Antwort. Eine Beziehung der wichtigsten Parteien ist in Abbildung 2.12 demonstriert.

Das Back-End übernimmt die Programmlogik des Front-Ends und definiert eine API für verschiedene Routen, um das Back-End von außerhalb anzusteuern. Die von der Login Komponente des Front-Ends ausgelöste Anfrage landet beim Session Controller des Back-End. Dieser empfängt und verarbeitet die Daten, indem zunächst nach vorhandenen Nutzereinträgen in der Datenbank gesucht wird. Wurde kein Eintrag gefunden, wird ein neuer Nutzer erstellt. Danach werden Hash-Werte der Passwörter, also dem Eingehenden und dem Vorhandenen, miteinander verglichen. Stimmen sie überein, wird ein JSON Web Token (JWT) erzeugt und zurück zum Front-End gesendet. Dieses Token stellt die Session des Nutzers dar und wird für alle weiteren Anfragen an das Back-End genutzt. Der Nutzer wird so durch das Token gegenüber dem Back-End authentifiziert.

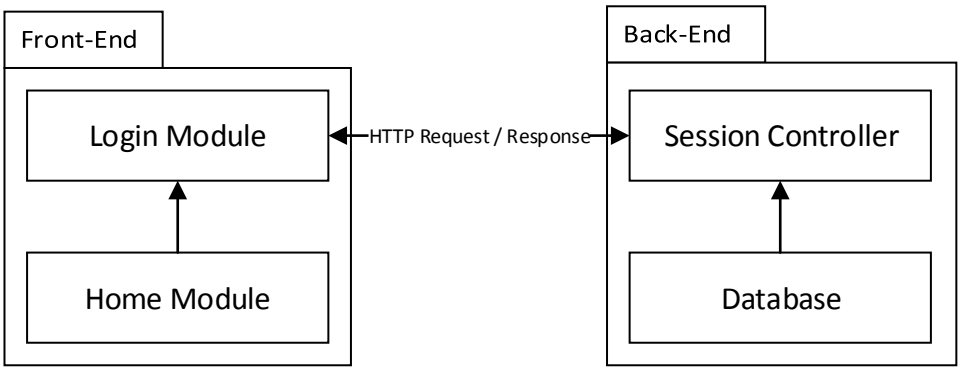


Abbildung 2.12 Schematische Darstellung der Login und Home Module des Front-End, sowie die Kommunikation zum Session Controller des Back-End während einer Anmeldeprozedur

Nachdem das Front-End die Antwort des Back-Ends erhalten hat, wird der Webseiteninhalt ausgetauscht. Der Nutzer befindet sich dann auf dem sogenannten Dashboard, einer Übersichtseite von AMCS.

2.3.3 Ziel

Das Ziel dieser Arbeit ist es, eine geeignete Single Sign-on Lösung für AMCS und andere Dienste zu finden. Wie schon bereits erwähnt, ist das wichtigste Merkmal beim Anmelden bei AMCS die anonyme Nutzung. Das heißt, es sollen keinerlei Daten des Nutzers erhoben werden, die Rückschlüsse auf die einzelne Persönlichkeit ermöglichen. Aus diesem Grund meldet sich ein Nutzer mit einem beliebigen Pseudonym und einem Passwort an. Die wichtigste Anforderung an eine geeignete Single Sign-on Lösung ist demnach die anonyme Nutzung.

Weiterhin ist es wünschenswert, die Nutzerkonten auf einem universitätsinternen Server zu verwalten. Somit wird ein zentralisierter Ansatz gefordert, bei dem sich Nutzer beispielsweise nicht über Google, Facebook und Co. anmelden. Die Begründung hierfür ist, dass dem Nutzer mehr Vertrauen gegenüber dem Login-Service geschenkt werden soll. Denn gerade Google und Facebook sind ja dafür bekannt, viele Daten über ihre Nutzer zu sammeln, wie auch wieder vor kurzem ein Bericht zeigte[14].

Die gesicherte Nutzung für Lehrende soll weiterhin gewährleistet sein, ebenso wie die Möglichkeit einer PIN-Eingabe, die für den direkten Beitritt einer Lehrveranstaltung nach erfolgreicher Anmeldung genutzt wird.

[14] Siehe auch https://www.theguardian.com/commentisfree/2018/mar/28/all-the-data-facebook-google-has-on-you-privacy (aufgerufen am 03.05.2018)

Ein weiterer Punkt wird zwar nicht explizit gefordert, jedoch sollte auch die Möglichkeit bedacht werden, bereits vorhandene Nutzerkonten in den Login-Service zu importieren, sodass mit dem bestehen System von AMCS direkt weitergearbeitet werden kann.

Zusammenfassend ergeben sich folgende Anforderungen an den Login-Service für AMCS:

(1) Universitätsinterner und zentralisierter Login-Server

(2) Anonyme Nutzung

(3) Gesicherte Nutzung für Lehrende

(4) Eingabe einer PIN von einer Lehrveranstaltung

(5) Möglichkeit des Imports bereits existierender Nutzerkonten

Tabelle 2.3 Anforderungen an den Login-Server für AMCS

2.4 VERWANDTE ARBEITEN

Wie schon im Abschnitt 2.2 angedeutet, scheinen größere Unternehmen wie Google und Facebook einen gewissen Einfluss auf den Trend zu verwendender Single Sign-on Technologien vorzugeben. Es gibt zahlreiche Anleitungen, wie zum Beispiel Google als Login für die eigene Webanwendung eingesetzt werden kann. Des Weiteren haben sich viele große Unternehmen an der Entwicklung einiger Technologien beteiligt und somit schneller vorangebracht, wie es bei OpenID der Fall ist (vgl. Schwan, 2001). Aufgrund der zahlreichen Berichte und Suchergebnisse scheint der Trend von SAML hin zu OAuth und OpenID Connect zu gehen.

Viele Unternehmen machen sich die Technologien zu Nutze und bieten fertige Single Sign-on Lösungen an. Diese sind in der Regel bzw. je nach Funktionsumfang kostenpflichtig und versprechen eine einfach zu nutzende Rundumlösung. *Auth0* [15] und *onelogin* [16] sind dabei nur ein Beispiel von vielen. Andere, wie die Webseite der *Golem Media GmbH* [17], bieten eine Hybridlösung zwischen einer klassischen Anmeldung und einer Anmeldung durch Google und Facebook an. Doch offensichtlich liegt hier eher der Fokus auf Nutzerfreundlichkeit, als auf Vereinfachung bei der Entwicklung der Webseite. Denn so existiert, zusätzlich zum Entwicklungsaufwand für eine klassische Anmeldung, auch noch der Aufwand für eine delegierte Anmeldung.

[15] Siehe auch https://auth0.com (aufgerufen am 27.06.2018)
[16] Siehe auch https://www.onelogin.com (aufgerufen am 27.06.2018)
[17] Siehe auch https://account.golem.de/login (aufgerufen am 27.06.2018)

Auf der, für Programmierer sehr bekannten, Plattform Stack Overflow finden sich oft Themen zu Single Sign-on, bei denen es immer wieder um die gleichen Technologien geht. Dabei sind OAuth, SAML, OpenID und OpenID Connect sehr oft vertreten. Eine Frage[18] vom 06.07.2009 verdeutlicht dies noch einmal. Selten finden sich auch Vergleiche von Single Sign-on Lösungen für bestimmte Kriterien, wie Björn Feustel in seinem Artikel[19] über „Vergleich von Java SSO Lösungen" schreibt.

Bei der Betrachtung anderer Audience Response Systeme wie AMCS wird schnell ersichtlich, dass die klassische Anmeldung mit einer E-Mail-Adresse und einem Passwort nach wie vor bevorzugt wird. Doch es gibt auch Ausnahmen. Bei dem Audience Response System *SMILE*[20] auf der Seite der Albert-Ludwigs-Universität Freiburg wird Shibboleth eingesetzt, wodurch auch andere wissenschaftliche Einrichtungen daran teilnehmen können. Ein weiteres Beispiel stellt *ARSnova*[21] dar. Dort ist es sogar möglich anonym teilzunehmen. Im Vergleich zu AMCS besteht der entscheidende Unterschied jedoch darin, dass die anonyme Nutzung nur durch eine Nutzer-Session beschrieben wird. Das bedeutet, dass z.B. Antworten keinem eindeutigen Namen bzw. Pseudonym zugeordnet werden können.

[18] Siehe auch https://stackoverflow.com/questions/1087031/whats-the-difference-between-openid-and-oauth (aufgerufen am 27.06.2018)
[19] Siehe auch https://www.oio.de/public/java/sso/single-sign-on-vergleich.htm (aufgerufen am 27.06.2018)
[20] Siehe auch https://smile.informatik.uni-freiburg.de/student/ (aufgerufen am 27.06.2018)
[21] Siehe auch https://arsnova.eu/mobile/# (aufgerufen am 27.06.2018)

3 KONZEPT

In diesem Kapitel wird das Konzept der Single Sign-on Lösung entwickelt, welches danach implementiert werden soll. Zunächst wird eine Eigenentwicklung präsentiert, die noch vor der Untersuchung bereits vorhandener SSO-Lösungen entworfen wurde. Dabei gab es eine Reihe von Vorüberlegungen, wie sich ein einfacher Login mit einem externen Login-Server umsetzen lässt. So entstanden verschiedene Skizzen und kleinere Prototypen von Entwürfen solcher Architekturen. Diese wurden stetig verbessert und erweitert. Letztendlich haben sie zum besseren Verständnis beigetragen und auch die grundlegende Funktion einer SSO-Lösung gezeigt. Wie sich dann in der Recherche und Analyse herausstellte, sind sich viele der bereits existierenden Architekturen sehr ähnlich und weichen von den Vorüberlegungen kaum ab.

Aus diesem Grund wird im Folgenden einer dieser Entwürfe präsentiert. Der Entwurf soll auch hier dazu beitragen, um die Ideen und Gedankengänge des Konzeptes verständlicher darzulegen. Nachdem dann die Grundidee erarbeitet wurde, soll diese mit anderen, bereits existierenden Architekturen verglichen werden. Der Vergleich soll die Wahl einer geeigneten SSO-Lösung vereinfachen, indem die verschiedenen Vor- und Nachteile aufgezeigt und bewertet werden. Da der Einsatz der gesuchten SSO-Lösung speziell für AMCS gedacht ist, werden dahingehend die Anforderungen für AMCS gegenüber anderen Merkmalen von SSO-Lösungen bevorzugt.

3.1 GRUNDIDEE DER EIGENENTWICKLUNG

Zuallererst geht es darum eine Grundidee zu entwickeln. Das ist relativ einfach möglich, auch wenn zuvor noch keine Single Sign-on Systeme genauer studiert wurden.

Zunächst müssen die betroffenen Parteien eines externen Login-Services ermittelt werden. Da sich diese Arbeit auf den Fall von AMCS bezieht, welche eine Webanwendung darstellt, ist auch die Eigenentwicklung webbasiert. Es existieren drei Parteien. Die Erste ist der Nutzer, der dazu einen Webbrowser nutzt, um Ressourcen einer Webanwendung anzufordern. Im Sinne des HTTP Kontextes werden also GET-Anfragen an eine Serveradresse gesendet. Die Serveradresse entspricht dabei der Adresse der Webanwendung, von welcher die Ressourcen angefordert werden. Die Webanwendung ist also die zweite Partei. Da sich mehrere Webanwendungen eine Anmeldeoberfläche teilen sollen, ist die dritte und damit letzte Partei der Login-Server. Die Parteien müssen nun sinnvoll verbunden werden, um miteinander kommunizieren zu können. Die erste Verbindung besteht zwischen dem Nutzer und der Webanwendung. Die Webanwendung soll den Nutzer auf die Anmeldeoberfläche weiterleiten, wenn dieser auf geschützte Ressourcen zugreifen möchte. Die Kommunikation verläuft also von der Webanwendung zurück zum Webbrowser des Nutzers und dann zum Login-Server. Dort soll sich der Nutzer authentifizieren und danach zurück zur Webanwendung geleitet werden. Die

zweite Verbindung verläuft also von der Webanwendung über den Nutzer bis zum Login-Server und zurück. Als letzter Schritt soll die Webanwendung die Möglichkeit besitzen, beim Login-Server eine Bestätigung einzuholen, dass der Nutzer tatsächlich authentifiziert wurde. Die dritte Verbindung besteht also zwischen Webanwendung und Login-Server.

Anhand dieser Informationen wird nun folgende Abbildung erstellt, welche die drei Parteien und deren Verbindungen bildlich darstellt:

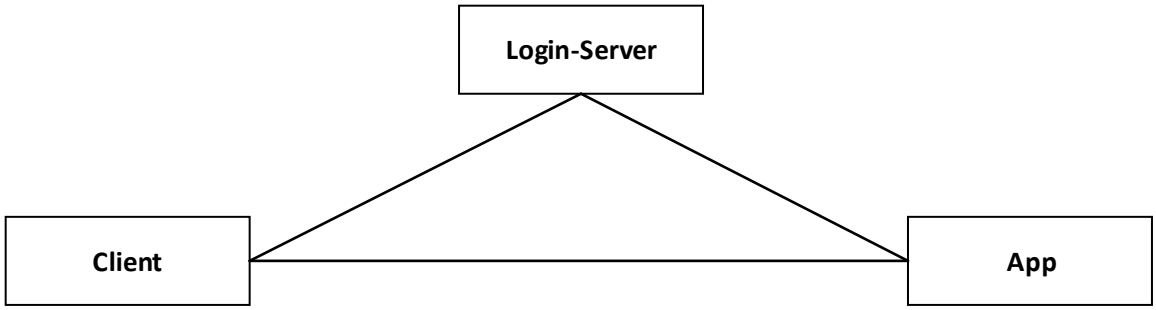


Abbildung 3.1 Single Sign-on Parteien

Der nächste Schritt besteht darin, die Details der Funktionsweise zu entwickeln. Die Abbildung 3.1 soll dann verbessert werden, um mehr Informationen vermitteln zu können. Außerdem soll ein Sequenzdiagramm erstellt werden, welches den genauen Ablauf der Kommunikation mit einigen technischen Details aufzeigt.

3.1.1 Verwendung und Sicherheit von HTTP-Cookies

Da HTTP zustandslos ist, werden Sessions eingesetzt, um Nutzer zu identifizieren. Es gibt verschiedene Wege, eine Session-ID auf der Client-Seite zu speichern, wie zum Beispiel über Cookies oder dem Local Storage. Der sinnvollste und auch sicherste Weg ist durch die Verwendung von HTTP-Cookies gegeben. Nämlich genau dann, wenn die Session-ID keine sensiblen Daten enthält und wenn es möglichst wenig Zugriffsmöglichkeiten auf das Cookie gibt. Generell sind Cookies an die jeweilige Domain gebunden. Das heißt, der Zugriff auf das entsprechende Cookie wird nur dann gewährt, wenn sich der Browser auch auf der entsprechenden Domain befindet. Das Gleiche gilt für das Local Storage Konzept. (vgl. Lv, 2018)

Es ist unbedingt darauf zu achten, dass keine sensiblen Daten in einem Cookie gespeichert werden. Einzig und allein der Server muss alle notwendigen Informationen kennen. Die Verwendung von JavaScripts kann ein gravierendes Sicherheitsproblem darstellen, denn sie sind in der Lage, auf Cookies und den Local Storage zuzugreifen. Das Problem erscheint vielleicht zunächst nicht offensichtlich. Aber in der heutigen Zeit entwickeln sich Webtechnologien rasend schnell. Es werden immer mehr Bibliotheken und Frameworks bereitgestellt, um möglichst rasch mit der Entwicklung von Webanwendungen voran zu kommen. Doch genau dort

liegt die Gefahr. Beispielsweise werden oft JavaScripts aus externen Quellen in das eigene Projekt eingebunden. Lädt der Client mit seinem Browser diese Webseite, werden auch die externen Quellen geladen und eingebunden. Nun haben jedoch plötzlich nicht nur die eigenen JavaScripts auf die Cookies und den Local Storage Zugriff, sondern auch diejenigen aus den externen Quellen. Gelingt es einem Angreifer eine dieser Quellen zu kompromittieren, kann er unentdeckt auf diese Daten zugreifen. Im Internet finden sich bereits viele Beiträge zu diesem Vorgehen, wie zum Beispiel der von Randall Degges[22]. Dieses Sicherheitsproblem betrifft nicht nur JavaScripts, sondern sogar extern eingebundene Grafiken oder Formulare, wie ein Bericht von David Gilbertson[23] zeigt.

Aus diesen Gründen setzt die Eigenentwicklung auf HTTP-Cookies, welche die Flags *HttpOnly* und *Secure* gesetzt haben. Dadurch können derartige Cookies nicht mehr von JavaScripts gelesen oder modifiziert werden. Das Secure-Flag sorgt dafür, dass diese Cookies nur noch über eine sichere HTTPS-Verbindung übertragen werden.

3.1.2 Kommunikationsablauf

Für den wohldefinierten Entwurf eines Kommunikationsablaufs sind mehrere Informationen notwendig. Die Ausgangs- und Zielbedingungen wurden bereits ermittelt. Jedoch existieren verschiedene zielführende Wege. Das System kann verschiedene Zustände annehmen, bei dem ein Nutzer versucht, auf geschützte Ressourcen zuzugreifen. Kurz gesagt müssen Vorüberlegungen über mögliche Anwendungsfälle getroffen werden.

Fall 1 – Anmelden bei erstmaligem Besuch der Webanwendung

Ein Nutzer, der die Webanwendung zum allerersten Mal nutzt, ist weder bereits angemeldet, noch existiert für ihn ein Benutzerkonto. Der Nutzer ist dem System unbekannt. Es existiert damit auch keine Session für den Nutzer. Dieser Zustand definiert den ersten Fall. Der Einfachheit halber werden für die beteiligten Parteien konkrete Beispieldaten eingesetzt. Der Nutzer agiert durch einen herkömmlichen Webbrowser. Die Webanwendung erhält den Namen „app1“ und ist über die URL „https://app1.de“ erreichbar. Der Login-Server ist über die URL „https://login.de“ zu erreichen. Als Kommunikationsprotokoll wird *Hypertext Transfer Protocol Secure (HTTPS)* verwendet. Dadurch ist eine gesicherte Verbindung garantiert, wodurch es für einen Angreifer schwieriger wird, an sensible Daten des Single Sign-on Systems zu gelangen. Das HTTP-Protokoll unterstützt verschiedene Anfragemethoden. Zunächst wird mit der GET-Methode eine Anfrage zur app1 gesendet, die wie folgt aussehen könnte:

---

[22] Randall Degges: “Please Stop Using Local Storage” - https://dev.to/rdegges/please-stop-using-local-storage-1i04 (aufgerufen am 22.06.2018)

[23] David Gilbertson: „I’m harvesting credit card numbers and passwords from your site. Here’s how.” - https://hackernoon.com/im-harvesting-credit-card-numbers-and-passwords-from-your-site-here-s-how-9a8cb347c5b5 (aufgerufen am 22.06.2018)

Client

```
GET / HTTP/1.1
Host: app1.de
Connection: keep-alive
```

Abbildung 3.2 Beispielanfrage mit der HTTP GET-Methode an die Webanwendung

Der Server der Domain „app1.de“ prüft darauf hin, ob der Client ein Session-Cookie mitgesendet hat, um eine mögliche Session wiederherzustellen. Das ist hier jedoch nicht der Fall und deshalb antwortet der Server mit dem entsprechenden Webinhalt in Form eines HTML-Dokumentes. Möchte der Client nun per GET-Methode auf einen geschützten Bereich zugreifen, antwortet der Server mit einer Umleitung zum Login-Server. Laut dem RFC2616 (vgl. Fielding, et al., 1999) muss die Antwort den Statuscode „302 Found“ enthalten. Zusätzlich wird das Headerfeld „Location“ vorgeschrieben, welches den entsprechenden URI zum Umleitungsziel enthält. Damit der Login-Server später weiß, woher der Client kam bzw. wohin er nach erfolgreicher Authentifizierung geleitet werden soll, wird der URI entsprechend mit einer Referenzadresse in „app“ kodiert:

Server

```
HTTP/1.1 302 Found
Location: https://login.de?app=app1.de/
```

Abbildung 3.3 Beispielantwort der HTTP GET-Methode vom Server mit einer Umleitung zum Login-Server

Der Browser des Clients sendet daraufhin erneut eine Anfrage mit der GET-Methode, dieses Mal jedoch an die entsprechende Adresse der vorhergehenden Nachricht:

Client

```
GET /?app=app1.de/ HTTP/1.1
Host: login.de
```

Abbildung 3.4 Beispielanfrage mit der HTTP GET-Methode an den Login-Server

Nun prüft der Login-Server ebenfalls, ob Session-Cookies vorhanden sind. Existiert keine gültige Session, wird das Login-Formular zurückgesendet. Der Client erhält nun ein Formular mit mehreren Eingabefeldern. AMCS schreibt hier drei Felder für den Login vor, die für Pseudonym, Passwort und PIN stehen, wobei Letzteres optional ist. Nach Eingabe der Daten werden

diese per HTTP POST-Methode zum Login-Server gesendet. Zu beachten ist, dass die Referenzadresse weiterhin mitgesendet werden muss, da diese sonst verloren geht. Die entsprechende POST-Anfrage könnte dann folgendermaßen aussehen:

Client

```
POST / HTTP/1.1
Host: login.de
Content-Type: application/x-www-form-urlencoded
Content-Length: 70
    pseudonym=testPseudonym
    &password=testPassword
    &app=app1.de
    &method=login
```

Abbildung 3.5 Übermittlung der Zugangsdaten des Nutzers an den Login-Server mit der HTTP POST-Methode

Die POST-Methode wurde mit Absicht gewählt, da es, je nach Situation, sinnvoll ist zwischen GET und POST zu unterscheiden. Bei der Übertragung von sensiblen Daten sollte darauf geachtet werden, dass die Daten zu keinem Zeitpunkt in der URL oder im Browserverlauf des Nutzers erscheinen. Des Weiteren sollten auch keine Lesezeichen erstellt werden können, die solche Daten enthalten. Aus diesen Gründen ist die POST-Methode das Mittel der Wahl. Weitere Vorteile sind die Möglichkeit der Übertragung binärer Daten sowie eine unbeschränkte Länge der Nutzdaten. Eine kurze Gegenüberstellung der Vor- und Nachteile zwischen den HTTP-Methoden GET und POST ist in der folgenden Tabelle 3.1 zusammengefasst.

Methode	GET	POST
Seite neu laden	Keine Besonderheiten	Daten werden erneut gesendet, wenn der Nutzer dem zustimmt
Lesezeichen erstellen	Möglich	Nicht möglich
Browserverlauf	Im Browserverlauf sichtbar	Nicht im Browserverlauf sichtbar
Cache	Zwischenspeicherung möglich	Keine Zwischenspeicherung möglich
Datentypen	Nur ASCII-Zeichen möglich	ASCII und binäre Daten möglich
Beschränkungen	Die Länge der URL ist auf mindestens 255 Bytes beschränkt.	Unbeschränkt
Sicherheit	Weniger sicher als POST, da alle Daten in der URL sichtbar sind	Sicherer als GET, da Daten wie, zum Beispiel Passwörter, nicht in der URL sichtbar sind
Sichtbarkeit	Daten sind in der URL direkt sichtbar	Daten befinden sich im Body und sind nicht direkt in der URL sichtbar

Tabelle 3.1 Vergleich von Vor- und Nachteilen zwischen den HTTP-Methoden GET und POST (vgl. HTTP Methods: GET vs. POST - w3schools, o.D.)

Der Login-Server wertet nun die Daten aus dem POST-Body der Anfrage aus, um alle Parameter mit den entsprechenden Werten zu erhalten. Wie in Abbildung 3.5 zu sehen ist, wurde der Parameter „method=login“ mitgesendet. Damit sollen Fehlinterpretationen vermieden werden, falls der Funktionsumfang des Login-Servers später erweitert werden sollte. Der Wert „login“ beschreibt genau eine Interpretationsweise der anderen Parameter und lässt sich mit einem direkten Methodenaufruf vergleichen. Nach der Auswertung werden die Zugangsdaten mit einem evtl. bereits vorhandenen Nutzerkonto abgeglichen. Das Pseudonym stellt dabei einen einzigartigen Wert dar und wird wie eine ID genutzt. Existiert zu dieser ID

bereits ein Konto, werden die jeweiligen Passwörter miteinander verglichen.[24] Bei Übereinstimmung wird für den Nutzer eine neue Session erstellt. Wie bereits in Kapitel 2.1.3 erklärt wurde, existiert zu jeder Session genau eine Session-ID. Die Session-ID wird nun in einem Session-Cookie beim Client, also dem Nutzer, hinterlegt. Bei einem weiteren Besuch des Login-Servers, kann der Nutzer wiedererkannt werden. Existiert zu dem Pseudonym noch kein Nutzerkonto, wird ein Neues angelegt. Existiert zu dem Pseudonym ein Nutzerkonto, jedoch mit einem falschen eingegebenen Passwort, muss an dieser Stelle der Nutzer aufgefordert werden, es erneut zu versuchen.

Der Nutzer wurde nun gegenüber dem Login-Server authentifiziert. Damit die Webanwendung auch weiß, ob der Nutzer sich erfolgreich authentifiziert hat, benötigt der Nutzer eine Art Zugangsschlüssel. Der Login-Server generiert hierfür einen Token sowie einen Zeitstempel. Token und Zeitstempel werden nun in einer Datenbank hinterlegt. Zusätzlich wird auch die Referenz zur Webanwendung mitgespeichert. Dies soll garantieren, dass der generierte Token nur für diese eine Referenz der Webanwendung „app1“ gültig ist. Der Zeitstempel sorgt dafür, dass der Token nur für eine bestimmte Zeitspanne lang gültig ist, danach ist dieser ungültig. Der Wert für die Zeitspanne sollte möglichst kurz gewählt werden. Ein guter Wert wäre zum Beispiel bei unter einer Minute. Denn der Token muss nur solange gültig sein, bis sich der Nutzer bei der Webanwendung angemeldet hat. Die drei Parameter müssen an einer zentralen Stelle abgespeichert werden. Es genügt nicht, sie in den Session-Daten des Nutzers zu speichern, da nicht nur der Nutzer, sondern auch die Webanwendung indirekten Zugriff darauf benötigen. Nicht nur der Token, sondern auch die Session selbst benötigt einen Zeitstempel. Ist bei einem späteren Besuch ein bestimmter Zeitpunkt überschritten, wird die Session zerstört und der Nutzer muss seine Zugangsdaten erneut eingeben. In diesem Fall sollte der Wert für die Zeitspanne größer gewählt werden. Sinnvoll wäre zum Beispiel eine Gültigkeit von einer Woche.

An dieser Stelle ist zunächst die direkte Interaktion zwischen dem Nutzer und dem Login-Server abgeschlossen. Der Nutzer muss nun zur gewünschten Webanwendung zurückgeleitet werden. Direkt im Anschluss, nachdem der Nutzer sich erfolgreich angemeldet hat, antwortet der Login-Server wieder mit dem HTTP-Code 302, um den Nutzer zur Webanwendung umzuleiten. Auch hier muss beachtet werden, dass bestimmte Parameter mitgesendet werden, in diesem Fall der zuvor generierte Token, der als Zugangsschlüssel dient.

[24] An dieser Stelle gilt es zu beachten, die Passwörter sicher im System zu hinterlegen. Nähere Details in Bezug auf die Sicherheit beim Speichern von Passwörtern, werden im Abschnitt 3.1.3 genauer erläutert.

Server

```
HTTP/1.1 302 Found
Location: https://app1.de?token=837ed61cd42c422d3b0b8be42be0b160de29145034
```

Abbildung 3.6 Antwort des Login-Servers zur Rückleitung zur Webanwendung mit angehängtem Token

Zurück bei der Webanwendung „app1“, die durch eine weitere HTTP-GET-Anfrage erreicht wird, wird nun der Token entgegengenommen. Da der Token keinerlei Information für andere als dem Login-Server darstellt, kann die Webanwendung zunächst nichts damit anfangen. Die Idee dahinter ist, dass selbst wenn der Token einem Angreifer in die Hände fällt, daraus keine weiteren Informationen gewonnen werden können. An dieser Stelle beginnt die direkte Kommunikation zwischen Webanwendung und Login-Server.

Sobald der Token der Webanwendung vorliegt, sendet diese eine Anfrage an den Login-Server per HTTP-POST-Methode. Darin enthalten sind der Token sowie der Parameter „method=validate“ und die Referenzadresse der Webanwendung. Ziel ist es, auf dem Login-Server eine andere Methode zu verwenden, als zuvor bei der Nutzeranmeldung. Der Login-Server prüft dann, ob der Token existiert und noch gültig ist, indem der Zeitstempel überprüft und die Referenzadressen miteinander verglichen werden. Der Login-Server antwortet der Webanwendung mit dem HTTP Statuscode 200 – O.K. Sind alle Bedingungen für die Gültigkeitsprüfung des Tokens erfüllt, antwortet er mit einem „true“, anderenfalls mit „false“, um der Webanwendung mitzuteilen, ob der Zugangsschlüssel des Nutzers gültig ist oder nicht. Da die Webanwendung jedoch bisher nichts weiter über den Nutzer weiß als den Token, werden vom Login-Server weitere Informationen mitgesendet. Denn der Login-Server kennt ja bereits das Pseudonym des Nutzers, welches von der Webanwendung sinnvoll verwendet werden könnte. Ein Beispiel wäre ein Begrüßungstext. Das Pseudonym könnte als eindeutige ID, in einer zur Webanwendung gehörigen Datenbank genutzt werden. Da das Format der Antwortnachricht nicht fest vorgeschrieben werden soll, wird ein kompaktes und weit verbreitetes Datenformat mit dem Namen *JavaScript Object Notation (JSON)* gewählt. Auf diese Weise lassen sich die zusätzlichen Daten in einem einheitlichen Format übertragen. Die gesamte Nachricht könnte nun folgendermaßen aussehen:

Server

```
HTTP/1.1 200 OK
Content-Type application/json
Content-Length 44
{"isValid":true,"pseudonym":"testPseudonym"}
```

Abbildung 3.7 Antwort des Login-Servers an die Webanwendung nach der Gültigkeitsprüfung des Tokens

Sobald die Anfrage vom Login-Server bearbeitet wurde, wird der Token invalidiert. Das kann zum Beispiel durch Löschen oder Markieren des Tokens geschehen. Der Token ist sozusagen ein Zugangsschlüssel zur Webanwendung mit einer einmaligen Verwendung. Auf diese Weise soll verhindert werden, dass Angreifer einen angefangenen Token wiederverwenden können.

Der Anmeldevorgang ist nun fast abgeschlossen. Nachdem die Webanwendung das O.K. vom Login-Server bekommen hat, wird auch hier eine Session erstellt. Ebenso wird, wie beim Login-Server, zu der Session ein Zeitstempel erstellt und zusammen mit dem Pseudonym in den Session-Daten abgespeichert. Die Session wird dann in einem Session-Cookie beim Client, also dem Browser des Nutzers, gespeichert. Ohne diese Session kann der Nutzer von der Webanwendung nicht wiedererkannt und müsste erst wieder zum Login-Server geleitet werden, um einen neuen Token zu generieren. Der Zeitstempel wird genutzt, um eine längere Abwesenheit des Nutzers zu erkennen. Besucht der Nutzer nach längerer Zeit die Webanwendung, erkennt diese den Zeitpunkt des letzten Besuchs. Ist er zu lange her, wird die Session zerstört und der Nutzer muss wieder zum Login-Server geleitet werden. Doch das ist Teil des zweiten Falls und wird später noch einmal näher beschrieben.

Der letzte Schritt besteht darin, den für den Nutzer sichtbaren und mittlerweile ungültigen Token aus der Adressleiste des Browsers des Nutzers zu entfernen. Denn seit der Umleitung nach der Serverantwort von Abbildung 3.6 hat sich visuell in der Adressleiste des Browsers nichts mehr verändert, da die letzte Kommunikation nur zwischen Webanwendung und Login-Server stattfand. Nachdem also die Webanwendung das O.K. vom Login-Server erhalten hat, muss sie den Browser ein weiteres Mal weiterleiten. Das Ziel ist dabei die Webanwendung selbst, jedoch ohne den Token als Adressanhang. Nach der letzten Umleitung hat sich der Nutzer erfolgreich gegenüber dem System authentifiziert und ist erfolgreich angemeldet. Er erhält nun Zugriff auf geschützte Daten, welche ihm die Webanwendung übermitteln kann. Die Webanwendung vertraut dabei ihrer eigens zuvor erstellten Session-ID, welche der Nutzer bei jeder weiteren Anfrage im Session-Cookie mitsendet. Der erste Fall ist damit abgeschlossen und wird in Abbildung 3.8 noch einmal bildlich dargestellt.

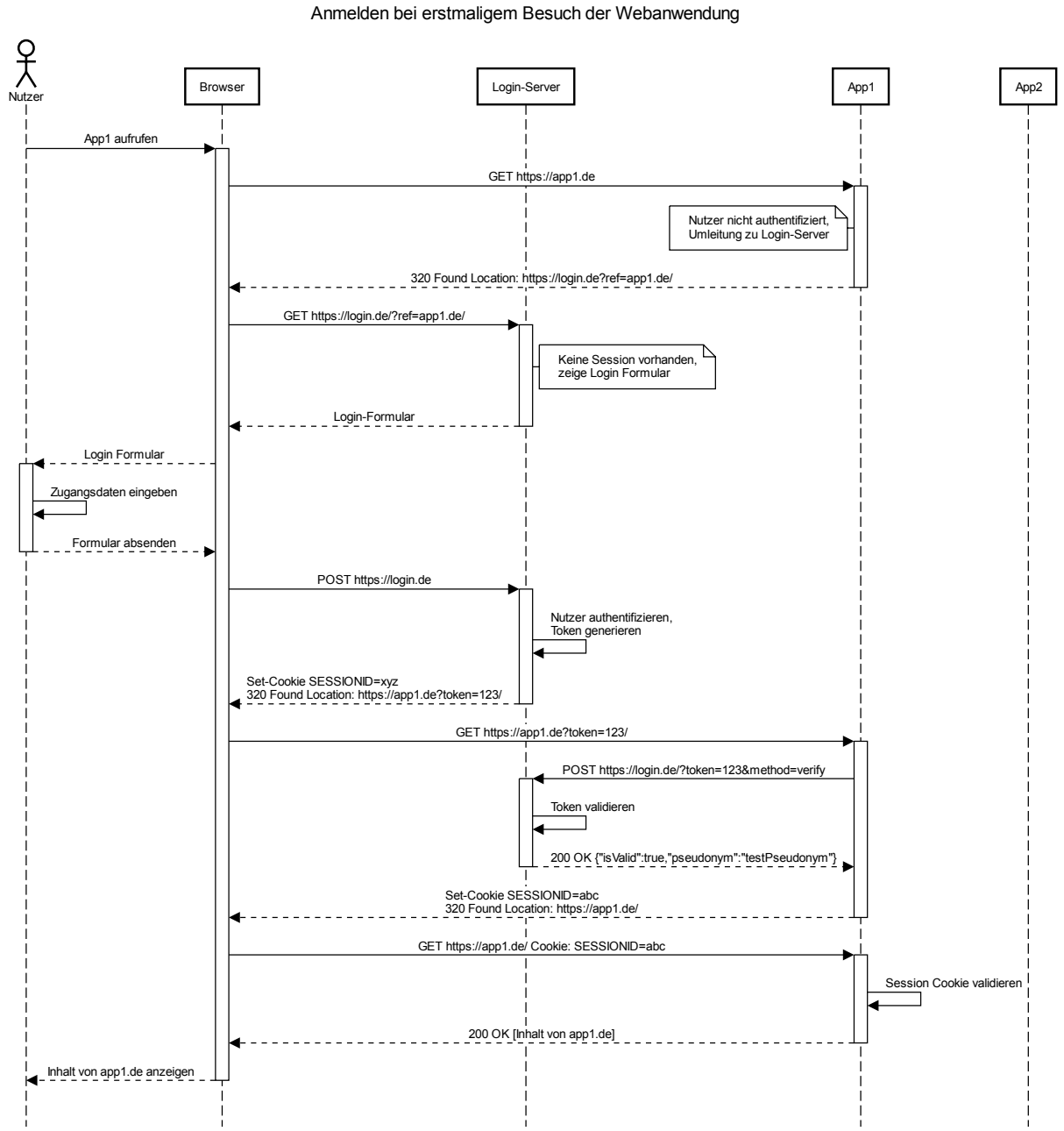


Abbildung 3.8 Kommunikationsablauf der Eigenentwicklung zur Anmeldung bei erstmaligem Besuch der Webanwendung

Fall 2 – Anmelden bei erneutem Besuch der Webanwendung

Wie sicherlich an einigen Stellen erkennbar ist, erscheint eine Unterscheidung zwischen dem ersten Besuch und den weiteren Besuchen des Nutzers als notwendig. Der zweite Fall ist also dadurch definiert, dass ein Nutzer dem System schon bekannt ist und Sessions vollständig oder zum Teil existieren. Das Hauptaugenmerk liegt also auf der Gültigkeitsprüfung bzw. Fortsetzung der Sessions. Der zweite Fall beginnt, wie der Erste, mit dem Nutzer. Er fordert Daten der Webanwendung durch eine HTTP-Anfrage mit der GET-Methode an. Dieses Mal besitzt der Nutzer jedoch schon ein Session-Cookie, welches eine Session-ID enthält. Diese

sendet er bei der Anfrage mit. Der Server der Webanwendung empfängt die Anfrage und wertet diese aus. Bei der Auswertung der Session wird die mitgesendete Session-ID entgegengenommen und geprüft, ob bereits eine Session zu dieser ID hinterlegt ist. Zunächst wird der Zustand betrachtet, dass eine Session existiert. Als erstes muss der Zeitstempel der Session auf Gültigkeit und auf Vollständigkeit überprüft werden. Das heißt, es wird geprüft, ob alle notwendigen Daten wie das Pseudonym des Nutzers existieren. Sind auch diese vorhanden, wird eine neue Session erstellt, die Daten in die neue Session kopiert und die Alte zerstört.

Die Idee dahinter ist, dass demgemäß eine neue Session-ID generiert wird und die Alte für ungültig erklärt wird. Denn falls ein Angreifer aus irgendeinem Grund an das Session-Cookie mit der Session-ID gelangt ist, kann dieser versuchen sich als der Nutzer auszugeben. Er könnte so Zugriff auf das System erhalten, ohne die Zugangsdaten des Nutzers zu kennen oder einzugeben. Unbedingt zu beachten ist, dass die Generierung einer neuen Session-ID keineswegs einen solchen Angriff verhindert. Für einen Angreifer wird es jedoch schwieriger die Session-ID des Nutzers zu stehlen[25]. (vgl. Ferrara, 2001)

Nachdem die Daten in die neue Session kopiert wurden, muss auch der Zeitstempel erneuert werden. Wie bereits im ersten Fall erklärt wurde, sollte bei der Gültigkeitsdauer des Zeitstempels eine sinnvolle Zeitspanne gewählt werden, um eine höhere Sicherheit des Systems zu gewährleisten. Anschließend wird die neue Session-ID beim Nutzer im Session-Cookie gespeichert. Der Nutzer ist nun wieder erfolgreich angemeldet und hat Zugriff auf die geschützten Daten. Existiert keine gültige Session, muss der Nutzer zum Login-Server umgeleitet werden. Genau wie im ersten Fall sendet die Webanwendung, anstatt der angeforderten Daten, einen Redirect mit dem HTTP-Statuscode 302 auf die Adresse des Login-Servers mit einem Parameter, der die Adresse der Webanwendung als Referenzadresse enthält. Dort muss sich der Nutzer wie schon zuvor mit seinen Zugangsdaten authentifizieren und wird danach zurück zur Webanwendung geleitet. Existiert jedoch auf dem Login-Server noch eine gültige Session für den Nutzer, so muss dieser seine Zugangsdaten nicht erneut eingeben. Er wird also direkt nach der Überprüfung der Session zurück zur Webanwendung umgeleitet. Natürlich wird in beiden Fällen vom Login-Server wieder ein neuer Token generiert, der für eine erneute Authentifizierung zwischen dem Nutzer und der Webanwendung notwendig ist. Der Zeitstempel der Session wird für den Login-Server jedoch nicht erneuert, da der Nutzer dazu gezwungen werden soll, sich nach einer bestimmten Zeit wieder mit der Eingabe seiner Zugangsdaten anzumelden.

[25] Um das sogenannte „Session Hijacking" weiter zu erschweren, sollten definitiv weitere Maßnahmen ergriffen werden. Denkbar wäre ein Abgleich des Browsertyps und der IP-Adresse, mit denen der Nutzer die Webanwendung besucht. Die Session-ID sollte auch schwer zu erraten sein, was durch ein geeignetes und zeitgemäßes Hashverfahren erreicht werden kann.

Für den Eigenentwurf ist zwischen der Kommunikation von Webanwendung und Login-Server keine weitere Interaktion mit dem Nutzer vorgesehen. Aus diesem Grund laufen die zwei Umleitungen und die Verifizierung des Tokens so schnell ab, dass der Nutzer in der Regel davon gar nichts mitbekommt. Für ihn könnte es nur wie eine sehr kurze Verzögerung beim Aufruf der Webanwendung wirken. Die Betrachtung des zweiten Falls ist nun auch abgeschlossen und wird in Abbildung 3.9 bildlich dargestellt.

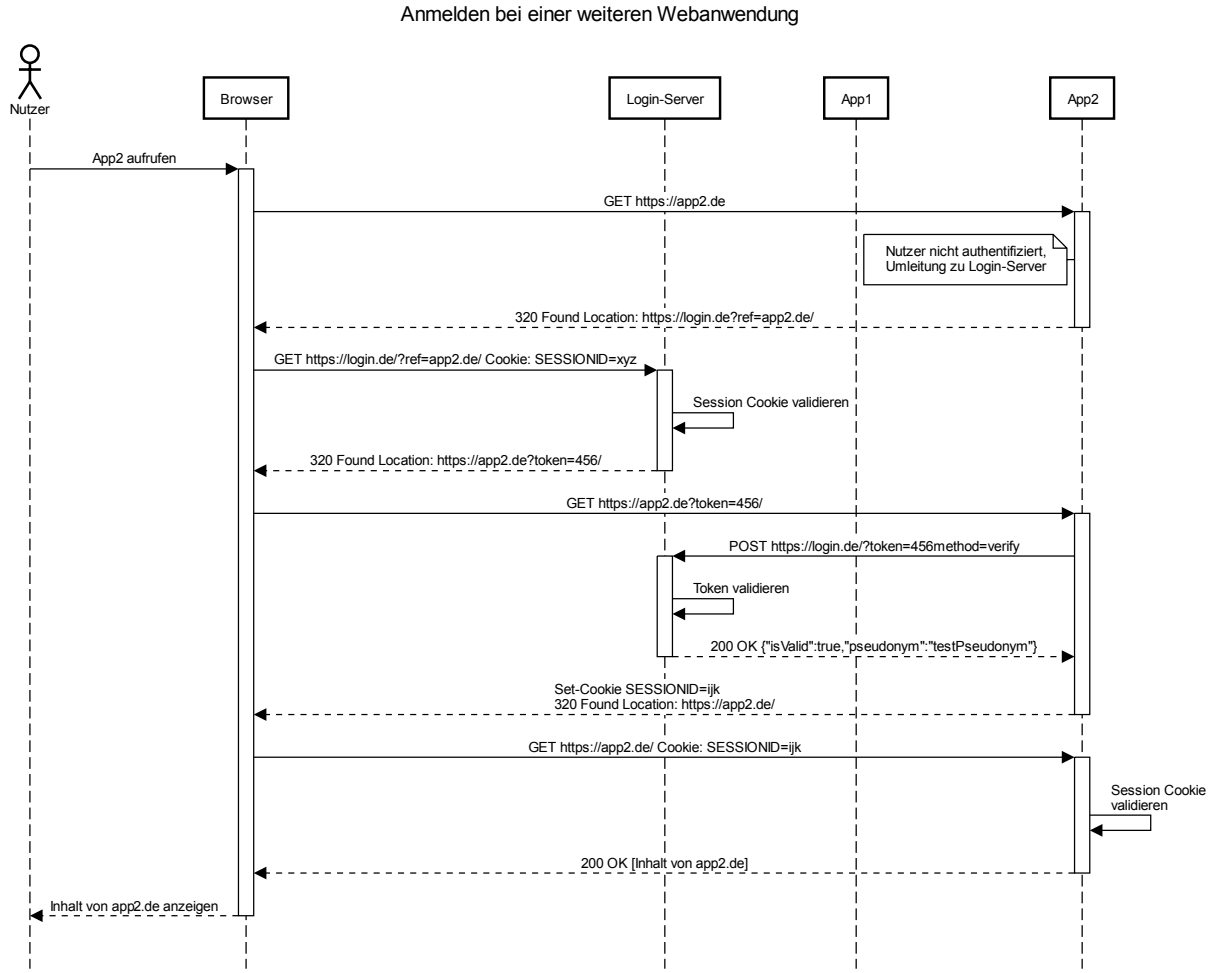


Abbildung 3.9 Kommunikationsablauf der Eigenentwicklung zur Anmeldung bei einer weiteren Webanwendung

Fall 3 – Anmelden bei einer weiteren Webanwendung

Bisher wurde nur die eine Webanwendung betrachtet. Der Sinn von Single Sign-on ist jedoch mehrere Anwendungen zu verwenden. Der dritte Fall ist also durch die Verwendung einer weiteren Webanwendung definiert. Dabei wird vorausgesetzt, dass der Nutzer sich bereits gegenüber der ersten Webanwendung authentifiziert hat. Um die Beispieldaten zu vervollständigen, erhält die zusätzliche Webanwendung den Namen „app2“ und ist über die URL „https://app2.de“ erreichbar. Der Nutzer wird zum Anmelden bei der Webanwendung „app2“

zum Login-Server umgeleitet. Der genaue Ablauf entspricht dabei dem des ersten Falles. Dort angelangt, prüft der Login-Server wieder, ob eine gültige Session existiert, wenn der Nutzer eine Session-ID in einem Session-Cookie mitgesendet hat. Existiert eine gültige Session, wird ein neuer Token generiert. Dieses Mal ist jedoch die Referenzadresse zur Webanwendung eine andere und der Token gilt nur für Webanwendung „app2". Ist die Session nicht gültig, muss sich der Nutzer wie in Fall 2 erneut mit seinen Zugangsdaten gegenüber dem Login-Server authentifizieren. Da sich dieser Fall von den anderen abgrenzen muss, existiert normalerweise eine gültige Session. Danach wird der Nutzer wieder zurück zur Webanwendung umgeleitet. Die Webanwendung überprüft wieder den erhaltenen Token beim Login-Server und erstellt bei Erfolg eine neue Session für den Nutzer. Hat der Nutzer zuvor die Webanwendung „app1" genutzt, existieren zu diesem Zeitpunkt drei Sessions. Eine existiert mit einer längeren Gültigkeitsdauer für den Login-Server und jeweils eine für die Webanwendungen „app1" und „app2". Im Regelfall bekommt auch hier der Nutzer kaum etwas vom Geschehen mit, solange gültige Sessions bereits existieren. Denn auch beim Anmelden an weiteren Webanwendungen sind keine weiteren Nutzerinteraktionen vorgesehen. Der dritte Fall ist nun auch abgeschlossen.

Fall 4 - Abmelden

Die am häufigsten vorkommenden Fälle oder Zustände, die beim Anmeldevorgang entstehen können, sollten abgedeckt sein. Der Detailgrad sollte ausreichen, um einen Prototypen implementieren zu können. Natürlich sind noch weitere Kenntnisse für die jeweils zu verwendenden Technologien notwendig. In der Regel sind Funktionen, wie Session Management, bereits implementiert und können direkt in Kombination mit HTTP Nachrichten genutzt werden. Der Entwurf der Eigenentwicklung sieht jedoch nicht nur das Anmeldeverfahren, sondern auch das Abmeldeverfahren in einem Single Sign-on System vor. Der gedachte Aufbau erlaubt es jedoch nicht, eine Single Sign-off Funktionalität ohne weiteres bereitzustellen. Damit ist die gleichzeitige Abmeldung bei allen verwendeten Diensten bzw. Webanwendungen gemeint. Daher gibt es nur zwei Möglichkeiten eines Abmeldekonzepts, die es zu unterscheiden gilt:

Varianten des Abmeldekonzepts
(1) Lokal: Abmelden bei einer einzelnen Webanwendung
(2) Global: Abmelden bei einer einzelnen Webanwendung und beim Login-Server

Tabelle 3.2 Unterscheidungen des Abmeldeverfahrens in mehrere Varianten

Die vollständige Abmeldung vom Single Sign-on System ist mit dem derzeitigen Entwurf nicht möglich, weil allen beteiligten Webanwendungen signalisiert werden müsste, dass der Nutzer sich abmelden möchte. Dabei gibt es gleich mehrere Probleme. Der Login-Server hat keine Möglichkeit, bei den Webanwendungen bestimmte Aktionen auszulösen. Zunächst müsste der Login-Server alle URLs der Webanwendungen kennen, die ein Nutzer verwendet hat, um sich mit dem Single Sign-on System anzumelden. Da der angemeldete Zustand des Nutzers aus der Existenz einer gültigen Session besteht, müsste die Webanwendung auch die zum jeweiligen Nutzer zugehörige Session-ID kennen. Momentan weiß aber nur der Nutzer, welche Session-ID er besitzt. Die Webanwendung wiederum weiß nur, welche Daten zu welcher Session-ID gehören. Es fehlt also eine Nutzerpseudonym-zu-Session-ID Beziehung. Doch angenommen eine solche Beziehung existiert und es lässt sich die zu jedem Nutzerpseudonym zugeordnete Session-ID ermitteln. Das Abmelden würde dann darin bestehen, die Sessions zu zerstören. Bei einem erneuten Besuch des Nutzers kann dann die vom Nutzer mitgesendete Session-ID keiner gültigen Session zugeordnet werden und der Nutzer muss sich erneut beim Login-Server anmelden. Eine weitere Möglichkeit wäre das Nutzerpseudonym auf eine Art *Black List* zu setzen. Das bedeutet, dass der Nutzer dann für einen bestimmten Zweck gesperrt wird. Besucht ein Nutzer erneut die Webanwendung und steht auch auf der Black List, dann wird eine entsprechend gültige Session nicht fortgesetzt, sondern zerstört. Das heißt, der Nutzer wird eigentlich erst bei erneutem Besuch abgemeldet. Die Abmeldung erfolgt sozusagen verzögert. Nach der Zerstörung der Session wird wie gewohnt fortgefahren, als ob für den Nutzer noch keine gültige Session vorhanden ist.

Soll der Nutzer nur lokal bei der Webwendung abgemeldet werden, ist die Lösung trivial. Dazu muss nur die Funktionalität der Webwendung um eine Methode erweitert werden. Löst der Nutzer diese Methode aus, wird die Session zerstört und der Nutzer muss dann umgeleitet werden, um den Vorgang abzuschließen. Die Umleitung kann dafür beispielsweise auf eine spezielle Logout-Seite führen, die dem Nutzer mitteilt, dass er nun abgemeldet wurde. Doch wie lässt sich ein Nutzer global abmelden, ohne eine Neuplanung des Konzeptes? Dazu muss nur die Sessions des Nutzers zu einem geeigneten Zeitpunkt zerstört werden. Das Konzept sieht bereits eine Zerstörung vor. Nämlich genau dann, wenn die Zeitspanne zum gespeicherten Zeitstempel einen bestimmten Wert überschritten hat. Dieses Verhalten kann dafür ausgenutzt werden, um den Nutzer immer wieder gegenüber der Webanwendung zu authentifizieren. Die Zeitspanne muss nur kurz genug sein. Immer dann, wenn die Session abgelaufen ist, wird der Nutzer erneut zum Login-Server geleitet. Der Login-Server entscheidet dann, ob der Nutzer sich mit seinen Zugangsdaten erneut anmelden muss, oder automatisch zurückgeleitet wird. Möchte sich der Nutzer nun global abmelden, müssen die Sessions in der aktuellen Webanwendung und auf dem Login-Server zerstört werden. Zuerst wird der Nutzer von der Webanwendung abgemeldet. Nachdem die Session zerstört wurde, muss der Nutzer zum Login-Server umgeleitet werden und dort sich ebenfalls abmelden. Die Umleitung ist notwendig, da wie bereits zuvor erwähnt, nur der Nutzer weiß, welche Session-ID zu ihm gehört. Direkt danach wird er zurück auf die Abmeldeseite der Webanwendung umgeleitet. Bei allen

anderen Webanwendungen ist der Nutzer dennoch weiterhin angemeldet, solange gültige Sessions existieren. Genau hier greift der Mechanismus mit der Zeitspanne der Sessions, der bereits zuvor beschrieben wurde. Der Nutzer darf ab jetzt solange die anderen Webanwendungen nicht verwenden, bis die jeweiligen Sessions abgelaufen sind. Genau aus diesem Grund sollte auch der Schwellwert für eine gültige Zeitspanne nicht zu groß gewählt werden. Umso größer die Spanne, desto länger muss der Nutzer warten.

Zusammenfassung

In diesem Unterkapitel wurde der Eigenentwurf eines einfachen Single Sign-on Systems vorgestellt. Dabei wurden der grundsätzliche Aufbau und der Kommunikationsablauf beschrieben. Danach sind die verschiedenen Zustände untersucht wurden, welche in dem System am häufigsten auftreten. Des Weiteren wurden zusätzlich zum Anmeldeverfahren auch verschiedene Möglichkeiten für ein Abmeldeverfahren untersucht. Letztendlich haben sich folgende Zustandsfälle herauskristallisiert:

Zustandsfälle
(1) Anmelden bei erstmaligem Besuch der Webanwendung
(2) Anmelden bei erneutem Besuch der Webanwendung
(3) Anmelden bei einer weiteren Webanwendung
(4) Abmelden

Tabelle 3.3 Zustandsfälle des Nutzers gegenüber dem Single Sign-on System.

Der Entwurf ist für den Webbereich gedacht, weshalb für die Kommunikation HTTP vorausgesetzt wird. Außerdem wird JSON eingesetzt, um weitere Nutzerdaten zwischen den Kommunikationspartnern zu übermitteln. Die Beziehungen der drei Parteien aus Abbildung 3.1 wird nun wie folgt erweitert:

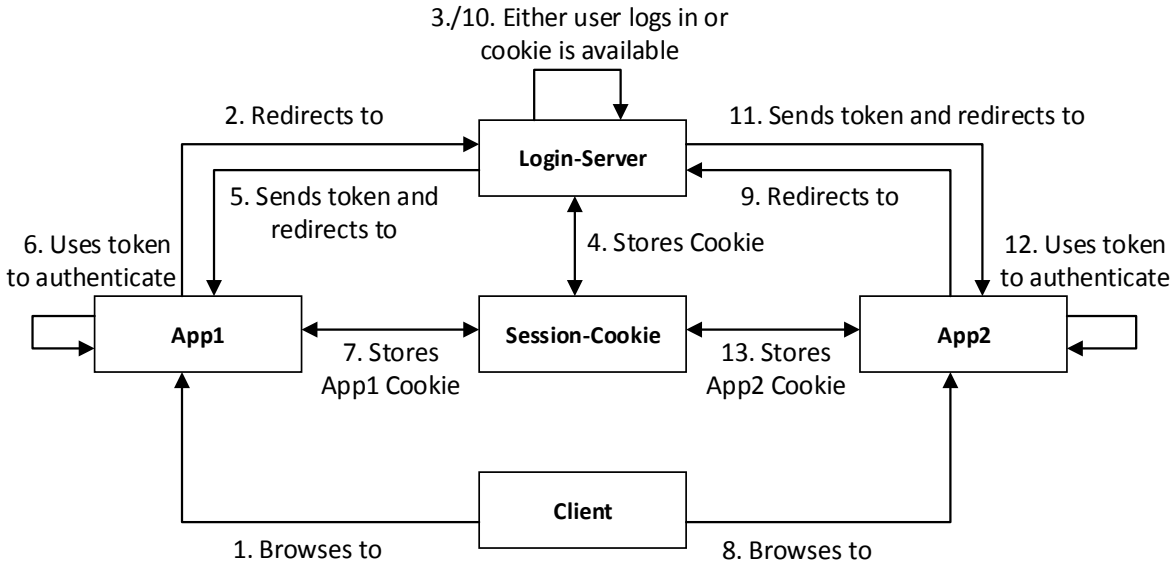


Abbildung 3.10 Erweiterte Darstellung der Single Sign-on Parteien (vgl. Peyrott, 2018)

3.1.3 Passwortsicherheit

In der heutigen Zeit wird es immer wichtiger, sensible Nutzerdaten sicher zu speichern. Aus diesem Grund werden zum Beispiel Passwörter nicht im Klartext in einer Datenbank hinterlegt, sondern es werden zuvor kryptografische Hashfunktionen darauf angewendet und nur deren Ergebnisse abgespeichert. Gelingt es einem Angreifer Zugriff auf das System und somit den Nutzerdaten zu erlangen, können die Passwörter nicht ohne Weiteres gelesen werden. Der größte Schwachpunkt an dieser Stelle ist der Mensch. Meist werden zu einfache Passwörter gewählt oder ein Passwort wird für mehrere Nutzerkonten anderer Anwendungen eingesetzt. Schwache Passwörter oder Passwörter, die in Klartext gespeichert werden, können fatale Folgen haben. Denn gerät ein Angreifer an solch ein Passwort, hat er Zugriff auf sämtliche Konten des Nutzers. Deshalb ist immer darauf zu achten, dass Passwörter und andere sensible Daten möglichst gut gesichert gespeichert werden.

Webseiten wie „haveibeenpwned.com“ verdeutlichen immer wieder, wie viele Webseiten Angriffen zum Opfer fallen. Zu den Opfern gehören jedoch nicht nur kleinere Webseiten, sondern auch solche, die von Unternehmen weltweit kommerziell genutzt werden. Ein extremes Beispiel ist das US-Amerikanische Softwareunternehmen „Adobe Systems Incorporated“. Im Dezember 2013 konnten über 150 Millionen Datensätze erbeutet werden. (vgl. Hunt, o.D.)

Umso schlimmer ist die Tatsache, dass die Passwörter nur schlecht abgesichert wurden und es möglich war, die Passwörter in Klartext wiederherzustellen. Das verdeutlicht wieder einmal, dass selbst Softwareriesen wie Adobe nicht vor Angriffen gefeit sind und wie wichtig es ist, ein kryptografisch sicheres und zeitgemäßes Verfahren zu wählen.[26]

3.1.4 Zusammenfassung

In diesem Abschnitt wurde die Funktionsweise von Single Sign-on Systemen anhand einer Eigenentwicklung näher erforscht. Durch die schrittweise Analyse traten Problemfragen auf, welche direkten Einfluss auf die Funktionsweise haben. Fragen nach der Datenübertragungstechnologie, die Art und Weise der Speicherung von Zuständen, sowie die Sicherheit von Passwörtern konnten so näher beleuchtet werden. Diese Vorgehensweise führte zu einem besseren Verständnis der Funktionsweise von Single Sign-on Systemen. Mithilfe dieser Informationen fällt der Vergleich zu anderen, bereits existierenden, Technologien einfacher.

3.2 ANFORDERUNGEN AN DAS SINGLE SIGN-ON SYSTEM

Um den Vergleich zwischen verschiedenen Single Sign-on Lösungen zu ermöglichen, müssen die Anforderungen für AMCS aus Kapitel 2.3.3 betrachtet werden. Allerdings sind diese Anforderungen nicht ausreichend, um eine sinnvolle Unterscheidung treffen zu können. Deshalb werden weitere Anforderungen in Form von Merkmalen an die gesuchte Single Sign-on Lösung gestellt:

Andere Einrichtungen, als die Technische Universität Dresden, sollen die Möglichkeit haben, die gesuchte Single Sign-on Lösung selbst zu nutzen und eigene Dienste wie AMCS einzubinden.

Die gefundene Lösung sollte möglichst wenige Codeänderungen an vorhandenen Diensten verursachen, damit die Einbindung eines solchen Login-Services möglichst einfach und schnell zu erledigen ist.

Die Kommunikation, als auch die Datenspeicherung soll in einem sinnvollen Maß gegen unbefugten Zugriff abgesichert sein.

Nicht nur die Anmeldung, sondern auch die Abmeldung eines Nutzers erweitert die Anforderungen. Der Nutzer soll weiterhin die Möglichkeit besitzen, sein Passwort zu ändern, wenn es notwendig sein sollte.

[26] Auf der Webseite https://crackstation.net/hashing-security.htm befindet sich eine gute Zusammenfassung und empfohlene Vorgehensweisen zum Thema Password Hashing, die sehr zu empfehlen sind. (aufgerufen am 20.06.2018)

Es soll auch unterschieden werden, ob und wie ein Nutzer von der Nutzung der Dienste ausgeschlossen werden kann.

Da die anonyme Nutzung der Dienste an oberster Stelle stehen, sollen die verschiedenen Lösungen auch dahingehend unterschieden werden, wie viele Daten über den Nutzer übertragen werden. Umso weniger Nutzerdaten übertragen werden, desto besser. Am besten werden nur wirklich notwendige Daten übertragen, also Daten wie ein Pseudonym, Tokens oder IDs zur Zuordnung von Anfragen und Antworten. Doch kann das auch garantiert werden?

Gibt es bereits Frameworks, welche die verschiedenen Single Sign-on Lösungen implementieren? Wenn ja, in welcher Programmiersprache?

Wie unterscheidet sich die Kommunikation zwischen den Lösungen?

Die erweiterten Anforderungen für AMCS aus Kapitel 2.3.3 lassen sich wie folgt zusammenfassen:

Anforderungen
(1) Universitätsinterner und zentralisierter Login-Server
(2) Anonyme Nutzung
(3) Gesicherte Nutzung für Lehrende
(4) Eingabe einer PIN von einer Lehrveranstaltung
(5) Möglichkeit des Imports bereits existierender Nutzerkonten
(6) Nutzung durch andere Einrichtungen
(7) Einsatz weiterer Dienste
(8) Codeänderungen an AMCS
(9) Logout
(10) Passwort ändern
(11) Nutzer Sperren
(12) Nutzerdaten-Overhead
(13) Frameworks

Tabelle 3.4 Übersicht der erweiterten Anforderungen für den Vergleich von Technologien für Single Sign-on

Diese Anforderungen bzw. Merkmale werden nun im folgenden Kapitel genutzt, um aktuelle Single Sign-on Lösungen miteinander vergleichen zu können.

3.3 VERGLEICH VON SINGLE-SIGN ON LÖSUNGEN

Um dem Vergleich zunächst eine Struktur zu geben, werden die Anforderungen bzw. Merkmale aus Kapitel 3.2 auf aktuelle Single Sign-on Lösungen angewandt. Darüber hinaus werden verfahrensspezifische Merkmale aufgezeigt, die der nachfolgenden Wahl einer geeigneten Lösung dienlich sein sollen.

Zuerst werden die jeweiligen Verfahren genannt und zu jedem dann die oben genannten Anforderungen dargelegt. Im Anschluss wird der Erfüllungsgrad der Anforderungen jeweils tabellarisch zusammengefasst.

3.3.1 OAuth

Der Vergleich zu OAuth wird in diesem Abschnitt nicht näher untersucht, weil OAuth vorrangig zur Autorisierung gedacht ist. Ein weiterer Grund ist, dass OpenID Connect direkt auf OAuth basiert und speziell für die Authentifizierung entwickelt wurde. Deshalb ist ein Vergleich zwischen beiden und auch zu anderen Technologien nicht sinnvoll. Die Betrachtung allein durch OpenID Connect ist ausreichend aussagekräftig.

3.3.2 OpenID

OpenID ist ein bereits veraltetes Verfahren, welches früher jedoch auch von größeren Firmen wie der Deutschen Telekom AG[27], Facebook Inc.[28], PayPal Inc. oder Google Inc. genutzt wurde. Für dieses Verfahren ist es sinnvoll, für jede Nachricht eine verschlüsselte TLS Verbindung zu nutzen, da es anfällig für Man-in-the-Middle und Replay Angriffe ist[29].

Universitätsinterner und zentralisierter Login-Server

OpenID ist ein dezentrales Authentifizierungsverfahren, denn der Nutzer kann selbst entscheiden, welchen *OpenID-Provider* (Login-Server) er nutzen möchte, um sich bei der jeweiligen *Relying Party* (Webanwendung) anzumelden. (vgl. OpenID Foundation, 2007)

[27] Siehe auch http://wiki.openid.net/w/file/fetch/80030087/oidc_dt_iiw_20140504.pdf (aufgerufen am 17.05.2018)

[28] Siehe auch http://www.adweek.com/digital/facebook-announces-users-will-soon-be-able-to-login-to-facebook-with-an-openid/ (aufgerufen am 17.05.2018)

[29] Siehe auch http://openid.net/specs/openid-authentication-2_0.html#security_considerations (aufgerufen am 17.05.2018)

Anonyme Nutzung

Die anonyme Nutzung mit OpenID ist möglich, weil der OpenID-Provider selbst festlegen kann, wie der Nutzer authentifiziert wird. Der Nutzer kann sich also auch durch ein Pseudonym und ein Passwort authentifizieren. Weiterhin entscheidet der OpenID-Provider und der Nutzer, welche Informationen mit der Relying Party geteilt werden. Dazu wird dem Nutzer angezeigt, welche Informationen eine Relying Party anfordert. Der Anforderung kann zugestimmt oder nicht zugestimmt werden.

Eingabe einer PIN von einer Lehrveranstaltung

Für OpenID existieren eine Reihe verschiedener offizieller Erweiterungen. Mit der Erweiterung „OpenID Attribute Exchange“ ist es möglich, weitere Nutzerdaten zwischen dem OpenID-Provider und der Relying Party auszutauschen (vgl. Hardt, et al., 2007). Durch die Erweiterung kann das Pseudonym und optional die PIN einer Lehrveranstaltung bei der Authentifizierung des Nutzers angefordert werden. Wenn der Nutzer durch die Relying Party zum OpenID-Provider umgeleitet wird, könnte die entsprechende HTTP-Nachricht wie in Abbildung 3.11 aussehen:

Relying Party

```
HTTP/1.1 302 Found
Location: https://login.de/openid
      ?openid.mode=checkid_setup
      &openid.return_to=http://app1.de/content
      &openid.app.client_id=1234
      &openid.ax.type.pseudonym=http://openid.net/schema/person/pseudonym
      &openid.ax.type.pin=http://openid.net/schema/person/pin
      &openid.ax.required=pseudonym
```

Abbildung 3.11 OpenID Beispielnachricht einer HTTP-Umleitung zum OpenID-Provider mit Anforderung weiterer Nutzerdaten.

Durch den Parameter “openid.ax.required” kann festgelegt werden, welche Nutzerdaten erforderlich und welche optional sind. Parameterwerte wie „http://openid.net/schema/person/pin“ sind keine echten Webadressen. Sie sind frei erfunden und dienen nur zur eindeutigen Unterscheidung. Die Eingabe einer PIN muss allerdings vom OpenID-Provider unterstützt werden, was nur möglich ist, wenn dieser fest vorgeschrieben wird.

Möglichkeit des Imports bereits existierender Nutzerkonten

Der Import bereits existierender Nutzerkonten ist nur möglich, solange der OpenID-Provider fest vorgeschrieben wird. Dieser OpenID-Provider muss dann entsprechend konfiguriert sein, sodass sich Nutzer mit einer Kombination aus Pseudonym und Passwort anmelden, sowie die Möglichkeit einer PIN-Eingabe existiert. Die dahinterliegende Datenbank kann dann vorhandene Nutzerkonten integrieren. Allerdings widerspricht die Festlegung eines speziellen OpenID-Provider der Grundidee von OpenID.

Nutzung durch andere Einrichtungen

Andere Einrichtungen können den speziell konfigurierten OpenID-Provider, wie oben beschrieben, mit nutzen.

Einsatz weiterer Dienste

Weitere Dienste können problemlos eingesetzt werden, da der gewünschte OpenID-Provider durch den Nutzer gewählt und in der Regel nicht durch die Relying Party festgelegt wird.

Codeänderungen an AMCS

Der Aufwand für die Codeänderungen ist etwas erhöht. Die Anmeldeoberfläche von AMCS muss nur noch ein Feld für die Eingabe einer OpenID beinhalten. Nachdem der Nutzer seine OpenID eingegeben hat, muss AMCS zunächst prüfen, ob der entsprechende OpenID-Provider gültig ist. Danach wird der Nutzer dorthin umgeleitet. Der Nutzer authentifiziert sich beim OpenID-Provider und wird danach zurück zu AMCS umgeleitet. Dort wird die Gültigkeit der Authentifizierung des Nutzers überprüft. Dafür gibt es allerdings zwei Wege. Entweder wurde vor der ersten Umleitung ein sogenanntes „client secret“ generiert, mit dem die Relying Party (AMCS) die Authentifizierungsanfrage wiedererkennen kann, oder die Relying Party muss dann selbst beim OpenID-Provider erfragen, ob die vom Nutzer übertragenen Authentifizierungsdaten gültig sind. (vgl. OpenID Foundation, 2007)

Logout

OpenID unterstützt keine globale Abmeldung. Ein Nutzer kann also nur von einer einzelnen Relying Party abgemeldet werden. (vgl. Suoranta, et al., 2013)

Passwort ändern

Da jeder Nutzer sich seinen eigenen OpenID-Provider erstellen kann bzw. einen vorhandenen OpenID-Provider selbst wählen kann, ist keine Passwortänderung im Protokoll vorgesehen. Es ist allein die Aufgabe des OpenID-Providers, sich um das Authentifizierungsverfahren zu kümmern.

Nutzer sperren

Es ist am sinnvollsten einen Nutzer in der jeweiligen Relying Party (Webanwendung) auszusperren, da der Nutzer ja selbst seinen OpenID-Provider wählen kann. Deshalb ist es auch schwierig einen bestimmten Nutzer für mehrere Relying Parties auszusperren.

Nutzerdaten-Overhead

Der Overhead ist relativ gering. Die Nutzerdaten bestehen aus einer OpenID und zusätzlichen Daten wie dem Pseudonym und einer optionalen PIN. Lediglich die benötigten Parameter für diese Informationen sind etwas lang. So kann es auch passieren, dass die maximale Länge einer HTTP-Nachricht überschritten werden kann.

Frameworks

Fast alle Frameworks sind als veraltet markiert. Viele können jedoch weiterhin bezogen werden, wie zum Beispiel *JOpenID*[30] oder *Simple OpenID PHP Class*[31], welche beide sehr leichtgewichtig sind und nur OpenID für Single Sign-on unterstützen.

[30] Siehe auch https://code.google.com/archive/p/jopenid/ (aufgerufen am 17.05.2018)
[31] Siehe auch https://www.phpclasses.org/package/3290-PHP-Authenticate-users-with-OpenID-single-sign-on.html (aufgerufen am 17.05.2018)

Anforderung	- /o/+
Universitätsinterner und zentralisierter Login-Server	-
Anonyme Nutzung	o
Eingabe einer PIN von einer Lehrveranstaltung	o
Möglichkeit des Imports bereits existierender Nutzerkonten	-
Nutzung durch andere Einrichtungen	+
Einsatz weiterer Dienste	+
Codeänderungen an AMCS	o
Logout	-
Passwort ändern	o
Nutzer sperren	-
Nutzerdaten-Overhead	o

Legende: + = unterstützt, o = teilweise unterstützt, - = nicht unterstützt

Tabelle 3.5 Übersicht zum Vergleich von OpenID

3.3.3 OpenID Connect (OIDC)

OpenID Connect setzt seine verschlüsselte TLS Verbindung zur Kommunikation voraus. Das bedeutet, dass HTTPS als Übertragungsprotokoll eingesetzt wird. Eine weitere Verschlüsselung ist nicht vorgesehen. Dadurch soll die Nutzung von OpenID Connect vereinfacht werden. Sind Signaturen notwendig, kommen *JSON Web Tokens (JWT)* zum Einsatz.

Universitätsinterner und zentralisierter Login-Server

Im Fall von OpenID Connect entspricht der Login-Server dem OpenID Provider (OP). Der OpenID Provider kann von Jedermann zur Verfügung gestellt werden. Aus diesem Grund ist der Login-Server in diesem Verfahren dezentralisiert bzw. verteilt. Dabei ist es irrelevant, wer der OpenID Provider ist. Der Client vertraut dem, was der OpenID Provider sagt.

Anonyme Nutzung

OpenID Connect basiert auf OAuth 2.0 und kann dessen Freiheiten ausnutzen, wodurch eine reine Authentifizierung neben der Autorisierung möglich ist. OpenID Connect bietet die Möglichkeit einer anonymen Nutzung. Die Funktionsweise lässt sich mit dem Sequenzdiagramm in Abbildung 2.7 besser verdeutlichen.

Wenn der Nutzer auf den Client, also der Webanwendung, zugreifen möchte, wird er zum OpenID Provider (OP) mit einem „Authentication Request“ umgeleitet. In der Abbildung entspricht das dem Schritt Nummer 3. Dabei definiert der Client, welche Informationen er vom OpenID Provider benötigt. Die Spezifikation von OpenID Connect schreibt dabei verschiedene Parameter vor, die bestimmte Werte annehmen können (vgl. Sakimura, et al., 2013). Der Parameter „scope“ wird genutzt, um zusätzliche Informationen des Nutzers zu erhalten. Das können zum Beispiel der Vor- oder Nachname sein oder auch die E-Mail-Adresse. Die Werte für den scope-Parameter sind optional bis auf eine Ausnahme. Der Wert „openid“ muss zwingend angegeben werden, damit der Authentifizierungsserver weiß, dass der Client einen OpenID Connect Request durchführt. Werden keine weiteren Werte für den Parameter scope übermittelt, so werden auch keine weiteren Informationen über den Nutzer übertragen. Es wird lediglich ein Access Token generiert, der als Zugangsschlüssel zur Webanwendung dient. Die Anfrage des Clients an den Authentifizierungsserver per HTTP-POST Methode könnte wie folgt aussehen:

Client

```
POST /authorize HTTP/1.1
Host: login.de
Content-Type: application/x-www-form-urlencoded
Content-Length: 101
   response_type=code
   &client_id=yxvlukeo436
   &redirect_uri=https://app1.de/cb
   &scope=openid
   &state=23lk5j4h6
```

Abbildung 3.12 OpenID Connect Authentication Request für die anonyme Nutzung: Wichtig ist die Zeile “scope=openid”. Durch sie wird eine anonyme Nutzung des Login-Services ermöglicht, da keine weiteren persönlichen Daten des Nutzers übertragen werden.

Im Hinblick auf das Pseudonym gibt es allerdings ein Problem: Da prinzipiell jeder OpenID Provider als Login-Server genutzt werden kann, gibt es keine Möglichkeit ein Pseudonym im Stil vom bisherigen AMCS-Login zu verwenden. Es wird also an einer geeigneten Stelle eine Art Übersetzung zwischen OpenID Connect Login und Pseudonymen benötigt. Das würde allerdings den Anmeldevorgang unnötig verlängern bzw. verkomplizieren. Eine andere Möglichkeit wäre immer denselben OpenID Provider zu verwenden, welcher ein Pseudonym und ein Passwort als Zugangsdaten nutzt. Das widerspricht allerdings der Idee von OpenID. Jeder Nutzer soll frei entscheiden können, welchen OpenID Provider er nutzen möchte. Deshalb wird die Möglichkeit der Anmeldung durch Pseudonyme als nicht unterstützt angesehen.

Eingabe einer PIN von einer Lehrveranstaltung

In der Abbildung 2.1 ist eine Beispielanfrage mit verschiedenen Parametern dargestellt. Der Parameter „state“ wird durch den Client genutzt, um die Antwort auf eine Anfrage wiederzuerkennen. Um die PIN für eine Lehrveranstaltung zwischen zu speichern, kann dieser Parameter ausgenutzt werden. Auf diese Weise ist es möglich eine eingegebene PIN ohne große Umwege bis zum Abschluss der Anmeldung bereit zu stellen, um sie dann weiter zu verarbeiten. Allerdings muss die PIN dann schon vor dem eigentlichen Anmelden in einem zusätzlichen Eingabefeld eingegeben werden. Eine weitere Möglichkeit ist die Eingabe der PIN nach Abschluss der Anmeldung.

Möglichkeit des Imports bereits existierender Nutzerkonten

Aufgrund der dezentralen Eigenschaft und da Pseudonyme von diesem Verfahren nicht direkt unterstützt werden, ist es nicht sinnvoll bereits existierende AMCS Nutzerkonten zu importieren. Es müsste immer derselbe OpenID Provider von allen Nutzern verwendet werden. Das widerspricht jedoch der OpenID Connect Idee.

Nutzung durch andere Einrichtungen

Durch den Einsatz von Rückadressen (Redirect URI), ist es ohne Probleme möglich diesen Login-Service mit dem OpenID Connect Verfahren durch andere Einrichtungen zu nutzen.

Einsatz weiterer Dienste

Es gibt mehrere Möglichkeiten weitere Dienste mit OpenID Connect zu nutzen. Sie sind abhängig davon, wie der OpenID Provider arbeitet. Das OpenID Connect Protocol spezifiziert zwei Modi. Entweder müssen alle Clients zuvor manuell beim OpenID Provider registriert werden oder es wird die dynamische Client Registrierung verwendet. Dabei kann sich ein neuer Client, also ein weiterer Dienst, selbst registrieren und alle notwendigen Informationen austauschen. Zu den Informationen gehören zum Beispiel die Berechtigungen, die der Client beim Anmeldevorgang anfordern kann. Das kann beispielsweise der Name oder die E-Mail-Adresse des Nutzers sein. Weiterhin gehören das Logo des Clients und die Client-ID zu den Informationen. Die Client-ID sendet der Client beim Authentication Request als Parameter „client_id“, wie es in Abbildung 3.12 zu sehen ist. (vgl. Sakimura, et al., 2014)

Codeänderungen an AMCS

Die benötigten Codeänderungen an AMCS sind relativ gering. Die Anmeldeoberfläche von AMCS muss dahingehend verändert werden, dass ein Nutzer seine OpenID und eine optionale PIN eingibt. Beim Absenden der Daten muss dann ein Authentication Request, also eine Authentifizierungsanfrage, mit einer Umleitung an den entsprechenden OpenID Provider durchgeführt werden. Nach erfolgreicher Authentifizierung des Nutzers, muss dieser zurück zum AMCS umgeleitet werden. Danach muss AMCS noch den Token Request zum OpenID Provider durchführen. Anschließend kann der Nutzer wie zuvor weiter behandelt werden.

Logout

OpenID Connect bietet drei verschiedene Ansätze, um einen Nutzer abzumelden. Dabei besteht die Möglichkeit den Nutzer sowohl lokal als auch global abzumelden. Das heißt, ein Nutzer kann gleichzeitig bei mehreren Diensten abgemeldet werden. (vgl. Jones, 2017)

Passwort ändern

Es existieren keine Vorgaben, wie ein Nutzer sein Passwort ändern kann, da jeder OpenID Provider selbst festlegt, wie sich ein Nutzer authentifiziert. Die gängigste Form ist jedoch die Authentifizierung per Benutzername und Passwort. Möchte ein Nutzer sein Passwort ändern, muss er das direkt bei seinem OpenID Provider erledigen.

Nutzer sperren

Aufgrund der eigenen Wahl des OpenID Providers können Nutzer nur auf Anwendungsebene ausgesperrt werden. Das heißt jede Webanwendung bzw. jeder Dienst muss selbst eine Benutzerverwaltung implementieren.

Nutzerdaten-Overhead

Der Overhead der Nachrichten ist relativ gering, da sie meist ähnlich wie in Abbildung 3.12 aufgebaut sind. Das heißt, die Nachrichten bestehen fast nur aus Nutzdaten, die sich jeweils aus einem Parameternamen und einem Parameterwert zusammensetzen.

Frameworks

Es existiert bereits eine Vielzahl an Frameworks in verschiedenen Programmiersprachen. Sie sind mal mehr, mal weniger gut ausgebaut. Manche Implementierungen unterstützen nur die

Funktion eines OpenID Providers[32] wie zum Beispiel *SimpleID*, andere wiederrum implementieren alle OpenID Connect Protokolle und mehr, wie zum Beispiel *phpOIDC*[33].

Anforderung	- /o/+
Universitätsinterner und zentralisierter Login-Server	-
Anonyme Nutzung	o
Eingabe einer PIN von einer Lehrveranstaltung	o
Möglichkeit des Imports bereits existierender Nutzerkonten	-
Nutzung durch andere Einrichtungen	+
Einsatz weiterer Dienste	+
Codeänderungen an AMCS	+
Logout	+
Passwort ändern	o
Nutzer sperren	-
Nutzerdaten-Overhead	o

Legende: + = unterstützt, o = teilweise unterstützt, - = nicht unterstützt

Tabelle 3.6 Übersicht zum Vergleich von OpenID Connect

3.3.4 Security Assertion Markup Language (SAML)

SAML bietet ein sehr breites Spektrum an bereits vordefinierten Anwendungsfällen. Sie werden durch die *Profiles* beschrieben und sind in verschiedene Aufgabenbereiche eingeteilt. Ein Bereich ist für Single Sign-on zuständig, welcher jedoch nicht nur für das Anmeldeverfahren an sich gedacht ist. Eines dieser Unterprofile wird *Web Browser SSO Profile* genannt und beschreibt verschiedene Wege einen Nutzer gegenüber dem System zu authentifizieren. Eine verschlüsselte Verbindung über HTTPS ist nicht notwendig, wird aber empfohlen.

[32] Siehe auch http://wiki.openid.net/w/page/12995226/Run%20your%20own%20identity%20server (aufgerufen am 09.05.2018)
[33] Siehe auch http://openid.net/developers/certified/ (aufgerufen am 09.05.2018)

Universitätsinterner und zentralisierter Login-Server

SAML bietet im Web Browser SSO Profile verschiedene Möglichkeiten den Nutzer zu authentifizieren. Dabei gibt es die Möglichkeit den Nutzer aus einer Liste verfügbarer Identity Provider zu wählen, welche der Identity Provider Discovery Service zur Verfügung stellen kann. Es lassen sich aber auch direkt einzelne Identity Provider vorgeben, was einen zentralisierten Ansatz ermöglicht. Denn der Identity Provider entspricht dem Login-Server. Der Identity Provider Discovery Service ist im Prinzip ein Server, der dem Nutzer eine Auswahl an verschiedenen Login-Servern bietet.

Anonyme Nutzung

Über den Identity Provider kann ein Service Provider (Client) weitere Informationen über den Nutzer anfragen. Die Nutzerinformationen werden in Assertions übertragen, welche im XML-Format definiert sind und auch im XML-Format übertragen werden. Zu beachten ist, dass der Identity Provider festlegt, welche Informationen über den Nutzer erfragt werden können. Ein Service Provider kann also nur das erfragen, was der Identity Provider bereit ist zu teilen. Verwendet der Nutzer einen eigenen Identity Provider, kann er selbst bestimmen, welche Informationen übertragen werden sollen. Mit SAML ist demnach eine anonyme Nutzung möglich, solange der Identity Provider entsprechend konfiguriert ist.

Eingabe einer PIN von einer Lehrveranstaltung

Wie bereits zur anonymen Nutzung erwähnt, ist es möglich weitere Daten in den Assertions zu übertragen. Somit wäre die Übertragung einer PIN machbar.

Möglichkeit des Imports bereits existierender Nutzerkonten

Existierende Nutzerkonten zu importieren ist nur möglich, wenn der Identity Provider fest vorgeschrieben wird und dieser auch mit einer Kombination aus Pseudonym und Passwort arbeitet. Das heißt, der Identity Provider muss speziell den in dieser Arbeit definierten Anforderungen entsprechen.

Nutzung durch andere Einrichtungen

Andere Einrichtungen können den SAML Login-Service mitnutzen, da der Identity Provider durch den Service Provider oder Identity Provider Discovery Service vorgeschrieben werden kann. Solange diese entsprechend konfiguriert sind, steht einer Nutzung Dritter nichts im Wege.

Einsatz weiterer Dienste

Weitere Dienste lassen sich nicht ohne Weiteres einsetzen. Denn in SAML werden Signaturen verwendet, die von der jeweiligen Gegenseite überprüft werden müssen. Ein neuer Dienst muss also zuvor registriert werden, wobei Zertifikate für die Signaturen ausgetauscht werden. (vgl. OASIS Open - Profiles, 2005)

Codeänderungen an AMCS

Der Aufwand ist bei SAML höher, denn es gibt viele verschiedene Arbeitsweisen, die durch die Verwendung des SSO Profils festgelegt werden. Der häufigste und einfachste Fall ist, wenn der Nutzer vom Service Provider per HTTP-Redirect zum Identity Provider geleitet wird und dieser wiederrum per HTTP-POST antwortet. Dieser Fall lässt sich auch als *SP Redirect Request / IdP POST Response* bezeichnen und kommt dem Kommunikationsablauf von anderen Verfahren, wie zum Beispiel CAS, am nächsten. Weiterhin wird aus Abbildung 2.9 deutlich, dass die Kommunikation eher zwischen dem Nutzer und dem Service Provider als dem Identity Provider stattfindet, was für einen höheren Implementierungsaufwand spricht.

Ein weiterer Punkt ist, dass Nachrichten auch im XML-Format vorliegen und deshalb entsprechend analysiert werden müssen. Das bedeutet weiteren Aufwand für die Kommunikationspartner. SAML setzt keine verschlüsselte Verbindung voraus, weil Signaturen verwendet werden können. Jedoch sollten heutzutage verschlüsselte Verbindungen immer eingesetzt werden, wenn dem nichts im Wege steht. Diese Aspekte vergrößern den Implementierungsaufwand und somit auch den Grad an Codeänderungen an AMCS.

Logout

SAML bietet speziell für die Abmeldung das Single Logout Profile an. Mit diesem ist es möglich, den Nutzer lokal und global abzumelden. Die Auslösung kann manuell durch den Nutzer oder automatisch durch den Service Provider bzw. Identity Provider erfolgen. (vgl. Browinski, 2016)

Passwort ändern

Das SAML Protokoll selbst sieht keine Möglichkeit zur Passwortänderung vor. Das eigene Passwort muss also entweder direkt beim entsprechenden Identity Provider geändert werden oder kann implementierungsabhängig vom Service Provider ausgelöst werden. Im letzten Fall ist es üblich, dass der Service Provider beim Identity Provider eine Rücksetzung des Passwortes auslöst. Der Nutzer erhält daraufhin eine E-Mail, mit welcher ein neues Passwort festgelegt werden kann. In Bezug auf diese Arbeit gibt es jedoch ein Problem. Die Nutzer sollen den Login-Service anonym nutzen können. Es ist also keine E-Mail-Adresse hinterlegt, wodurch

auch diese Art der Passwortrücksetzung nicht angewendet werden kann. Es wird also eine entsprechende Implementierung benötigt, die eine Passwortrücksetzung ohne E-Mail-Adressen ermöglicht.

Nutzer sperren

Nutzer können nur durch den Identity Provider oder in der jeweiligen Webanwendung selbst gesperrt werden. Erstes kann zum Problem werden, wenn der Nutzer selbst entscheiden kann, welchen Identity Provider er nutzen möchte. Es ist daher sinnvoll einen Nutzer durch die jeweilige Webanwendung selbst auszusperren. Zu beachten ist, dass der Nutzer alle weiteren Webanwendungen weiterhin nutzen kann und jeweils expliziert gesperrt werden muss.

Nutzerdaten-Overhead

Im Gegensatz zur Eigenentwicklung oder CAS besitzt SAML einen deutlich größeren Overhead. Zum einen gibt es die Möglichkeit viel mehr Nutzerdaten zu übertragen, die vom Identity Provider erfragt werden können. Zum anderen besitzen die Nachrichten an sich auch einen deutlich größeren Overhead, da diese meist im XML-Format vorliegen.

Frameworks

Für SAML gibt es sehr viele Frameworks bzw. Implementierungen. Erwähnenswert ist vor allem *Auth0*[34], welche ein sehr breites Spektrum an unterstützenden Protokollen und Frameworks in verschiedenen Programmiersprachen anbietet. Ein Framework, welches auch aktiv weiterentwickelt wird, jedoch weniger Protokolle unterstützt, ist *SimpleSAMLphp*[35].

[34] Siehe auch https://auth0.com (aufgerufen am 16.05.2018)
[35] Siehe auch https://simplesamlphp.org/ (aufgerufen am 16.05.2018)

Anforderung	- /o/+
Universitätsinterner und zentralisierter Login-Server	o
Anonyme Nutzung	o
Eingabe einer PIN von einer Lehrveranstaltung	+
Möglichkeit des Imports bereits existierender Nutzerkonten	o
Nutzung durch andere Einrichtungen	+
Einsatz weiterer Dienste	o
Codeänderungen an AMCS	-
Logout	+
Passwort ändern	o
Nutzer sperren	-
Nutzerdaten-Overhead	-

Legende: + = unterstützt, o = teilweise unterstützt, - = nicht unterstützt

Tabelle 3.7 Übersicht zum Vergleich von SAML

3.3.5 Central Authentication Service (CAS)

Ebenso wie bei OpenID Connect wird eine verschlüsselte Verbindung vorausgesetzt, bei der TLS über HTTPS zum Einsatz kommt. Aus diesem Grund ist keine weitere Verschlüsselung der Nachrichten vorgesehen.

Universitätsinterner und zentralisierter Login-Server

Der Login-Server wird bei CAS fest vorgeschrieben. Alle Nutzer verwenden denselben Server, um sich gegenüber dem System zu authentifizieren.

Anonyme Nutzung

CAS ist ein zentralisiertes Verfahren. Es existiert also nur ein Login-Server, bei dem die Nutzer authentifiziert werden. Der Login-Server bestimmt das Authentifizierungsverfahren und kann somit einen anonymen Login gewährleisten. Das Protokoll setzt allerdings eine Kombination aus Benutzername und Passwort voraus. Für beide existieren aber keine Regeln und so kann der Benutzername als Pseudonym behandelt werden. Nach der Validierung eines eben authentifizierten Nutzers wird der Nutzername und eine Bestätigung bzw. Abweisung der Gül-

tigkeit an den Client gesendet. Der Client ist hier wieder die Webanwendung, die eine Bestätigung der Gültigkeit vom Login-Server anfordert. Je nach verwendeter Protokollversion wird die Antwort im XML-Format bzw. reinen Text, der durch einfache Zeilenumbrüche getrennt ist, an den Client übertragen. Das Protokoll sieht also keine weitere Übertragung anderer Nutzerdaten vor, was eine anonyme Nutzung begünstigt.

Eingabe einer PIN von einer Lehrveranstaltung

Leider bietet CAS keine Möglichkeit zusätzliche Parameter während des Anmeldevorgangs zu übertragen, um sie nach Abschluss weiter zu verwenden. Die Eingabe einer PIN muss also zuvor auf Client zwischengespeichert oder nach erfolgreicher Anmeldung erfragt werden.

Möglichkeit des Imports bereits existierender Nutzerkonten

Da nur ein Login-Server existiert und der Login-Server selbst bestimmen kann, wie die Authentifizierung abläuft, ist es möglich bereits existierende Nutzer zu importieren. Natürlich müssen zuvor die vorhandenen Datensätze in ein entsprechendes Format gebracht werden, damit diese in der Datenbank importiert werden können.

Nutzung durch andere Einrichtungen

Die Nutzung durch andere Einrichtungen ist problemlos möglich, da wie auch bei OpenID Connect eine Rücksendeadresse mitgesendet wird. Der Login-Server weiß dann anhand der Adresse, wohin der Nutzer nach der Authentifizierung umgeleitet werden muss. Einzig die Wahrscheinlichkeit einer Kollision durch bereits genutzter Pseudonyme steigt mit der Anzahl anderer Einrichtungen, welche den Login-Server mitverwenden.

Einsatz weiterer Dienste

Auch der Einsatz weiterer Dienste ist problemlos möglich. Neue Dienste müssen nicht zuvor registriert und können direkt mit dem vorhandenen Login-Server genutzt werden.

Codeänderungen an AMCS

Die Codeänderungen sind beim AMCS relativ gering. Die Anmeldeoberfläche muss jedoch durch eine Umleitung zum Login-Server ersetzt werden. Nach erfolgreicher Authentifizierung des Nutzers wird dieser zurück zu AMCS umgeleitet. AMCS muss danach das erhaltene Service Ticket (ST) beim Login-Server validieren. Bei Erfolg muss der Nutzer noch einmal umgeleitet werden, um den Vorgang abzuschließen. Im Anschluss daran verhält sich AMCS wie zuvor.

Logout

Der Vorteil von CAS ist, dass sowohl ein lokales als auch ein globales Abmeldeverfahren vorgesehen ist. Eine Nutzersession wird beendet, wenn der Nutzer es selbst manuell auslöst, eine Session abgelaufen ist oder der Client vom Login-Server eine Mitteilung bekommt, dass der Nutzer abgemeldet werden soll. Somit kann ein Nutzer gleichzeitig von allen Clients abgemeldet werden.

Passwort ändern

Das Protokoll selbst sieht keine Möglichkeit zur Passwortänderung vor. Der protokollimplementierende Server muss selbst eine solche Funktionalität bereitstellen. (vgl. Apereo Foundation - CAS - Password Management, 2017)

Nutzer sperren

Ebenso ist der Ausschluss von Nutzern nicht im Protokoll definiert. Die jeweilige Implementierung muss auch hier Vorkehrungen treffen, um eine administrative Verwaltung bereit zu stellen.

Nutzerdaten-Overhead

CAS ist ein einfaches Protokoll und überträgt deshalb nur notwendige Daten. Zu diesen Daten gehören Parameter für den Anmelde- bzw. Abmeldevorgang sowie clientspezifische Parameter. Vom Nutzer selbst wird nur der Benutzername vom Login-Server übermittelt, solange der Nutzer zum Anmelden direkt auf den Login-Server umgeleitet wird.

Frameworks

Es existieren mehrere Frameworks des CAS-Protokolls. Der Entwickler *Apereo* selbst stellt eine gleichnamige Implementierung bereit, die auch eine Vielzahl weiterer Protokolle implementiert. Des Weiteren wurden nützliche Funktionen wie die Sperrung von Nutzerkonten oder die Änderung von Passwörtern eingefügt. Teilweise werden neue Funktionen über Module bereitgestellt und über eine Konfigurationsdatei konfiguriert. Es existieren auch weitere Implementierungen für andere Programmiersprachen wie z.B. *phpCAS*[36] oder *RubyCAS*[37].

[36] Siehe auch https://github.com/Jasig/phpCAS (aufgerufen am 11.05.2018)
[37] Siehe auch https://github.com/rubycas (aufgerufen am 11.05.2018)

Anforderung	- /o/+
Universitätsinterner und zentralisierter Login-Server	+
Anonyme Nutzung	+
Eingabe einer PIN von einer Lehrveranstaltung	-
Möglichkeit des Imports bereits existierender Nutzerkonten	+
Nutzung durch andere Einrichtungen	+
Einsatz weiterer Dienste	+
Codeänderungen an AMCS	+
Logout	+
Passwort ändern	-
Nutzer sperren	-
Nutzerdaten-Overhead	+

Legende: + = unterstützt, o = teilweise unterstützt, - = nicht unterstützt

Tabelle 3.8 Übersicht zum Vergleich von CAS

3.3.6 Eigenentwicklung

Sowohl die Eigenentwicklung als auch der Central Authentication Service sind sich von der Funktionsweise her sehr ähnlich. Darum besitzen beide Verfahren auch teilweise analoge Vor- und Nachteile. Es wird auch hier wieder eine verschlüsselte TLS Verbindung vorausgesetzt, um weitere Verschlüsselungsmechanismen zu vermeiden und die Verwendung zu vereinfachen.

Universitätsinterner und zentralisierter Login-Server

Entworfen wurde die Eigenentwicklung als ein zentralisiertes Verfahren. Es existiert nur ein Login-Server zu mehreren Diensten bzw. Clients, um Nutzer gegenüber dem System zu authentifizieren.

Anonyme Nutzung

Die Eigenentwicklung zielte von Anfang an auf eine anonyme Nutzung ab. Das heißt, die Nutzerkonten bestehen aus einem Pseudonym und einem Passwort. Es werden keine weiteren nutzerbezogenen Daten in der Datenbank gespeichert. Der Client bzw. die Webanwendung erhält am Ende das Pseudonym des Nutzers, um diesen im weiteren Verlauf zuordnen zu

können. Wie auch bei CAS wird der Nutzer zur Anmeldung zum Login-Server umgeleitet, um einen Token zu erhalten. Nach der Umleitung zurück zur Webanwendung, prüft diese ihrerseits die Gültigkeit des Tokens beim Login-Server und erhält dann das Nutzerpseudonym und im Idealfall auch die Bestätigung der Gültigkeit.

Eingabe einer PIN von einer Lehrveranstaltung

Die Eingabe einer PIN ist bei der Eigenentwicklung bereits vorgesehen. Zusätzlich zur Eingabe der Zugangsdaten ist die Eingabe einer PIN möglich.

Möglichkeit des Imports bereits existierender Nutzerkonten

Auch die Eigenentwicklung bietet wie CAS die Möglichkeit des Imports bereits existierender Nutzerkonten. Es existiert eine Datenbank mit Nutzerkonten, die aus einem Pseudonym und einem Passwort bestehen. Werden die Datensätze der Nutzerkonten vom AMCS in ein entsprechendes Format gebracht, ist der Import möglich. So können die Nutzer AMCS auch mit dem neuen Login-Service weiterhin wie gewohnt nutzen.

Nutzung durch andere Einrichtungen

Andere Einrichtungen können die Eigenentwicklung problemlos nutzen. Einzig das Problem der Kollisionswahrscheinlichkeit von Pseudonymen, die bereits in Verwendung sind, steigt.

Einsatz weiterer Dienste

Weitere Dienste, wie Webanwendungen, lassen sich ohne jeglicher Anpassung direkt verwenden. Der Login-Server unterscheidet die Dienste anhand ihrer Referenzadresse, die bei der Umleitung des Nutzers zum Login-Server mitgesendet wird.

Codeänderungen an AMCS

Die Codeänderungen sind praktisch gleich zu CAS. Die Anmeldeoberfläche von AMCS muss durch eine Umleitung zum Login-Server ersetzt werden. AMCS selbst muss dann noch den vom Nutzer erhaltenen Token beim Login-Server validieren. Danach wird fortgefahren wie zuvor.

Logout

Die Eigenentwicklung bietet eine lokale und teilweise globale Abmeldung. Das heißt die Nutzer-Session wird für den jeweiligen Client zerstört. Danach wird die Nutzer-Session beim Login-Server zerstört. Alle anderen Clients sind von der Abmeldung nicht betroffen. Jedoch laufen die einzelnen Sessions nach einer bestimmten Zeit ab. Der Nutzer wird dann wieder zum Login-Server umgeleitet und muss dort erneut seine Zugangsdaten eingeben, weil schließlich die Nutzer-Session durch die vorherige Abmeldung bereits zerstört wurde.

Passwort ändern

Es existiert noch keine Möglichkeit zu Änderung des Passwortes. Die Eigenentwicklung muss dahingehend noch erweitert werden. Der Vorteil ist jedoch, dass neue Funktionen direkt für den Einsatz im Single Sign-on System mit AMCS und anderen Diensten angepasst werden können.

Nutzer sperren

Auch die Aussperrung von Nutzern ist noch nicht vorgesehen. Für diesen Zweck ist eine Verwaltungsoberfläche notwendig, um einzelne Nutzer bzw. deren Pseudonyme zu verwalten. Ein anderer Weg ist die Aussperrung von Nutzers durch die Webanwendung selbst.

Nutzerdaten-Overhead

Der Overhead von Nutzerdaten ist äußerst gering. Wie auch bei CAS wird nur das Nutzerpseudonym übertragen. Alle anderen übertragenen Daten sind sogar noch geringer als bei CAS. Das liegt vor allem daran, da die Eigenentwicklung genau für einen speziellen Einsatzzweck konzipiert wurde. Das ist ein Vorteil und gleichzeitig auch ein Nachteil. Denn Entwicklungen für einen speziellen Einsatzzweck lassen sich schlechter wiederverwenden. Die Wartbarkeit ist hingegen höher.

Frameworks

Da es sich um eine Eigenentwicklung handelt, gibt es natürlich keine Frameworks bzw. andere Implementierungen.

Anforderung	- /o/+
Universitätsinterner und zentralisierter Login-Server	+
Anonyme Nutzung	+
Eingabe einer PIN von einer Lehrveranstaltung	+
Möglichkeit des Imports bereits existierender Nutzerkonten	+
Nutzung durch andere Einrichtungen	+
Einsatz weiterer Dienste	+
Codeänderungen an AMCS	+
Logout	o
Passwort ändern	-
Nutzer sperren	-
Nutzerdaten-Overhead	+

Legende: + = unterstützt, o = teilweise unterstützt, - = nicht unterstützt

Tabelle 3.9 Übersicht zum Vergleich der Eigenentwicklung

3.3.7 Gesicherte Nutzung für Lehrende

Während des Vergleichs von Single Sign-on Lösungen stellt sich heraus, dass die gesicherte Nutzung für Lehrende nicht durch die jeweilige Technologie, sondern durch jeweilige Webanwendung umgesetzt werden muss. Nur so ist es möglich, Nutzerrechte individuell für jede Webanwendung zu verwalten. Andernfalls kann es zu ungewollten Interferenzen oder schlechter Wartbarkeit kommen. Genauer gesagt, müssen die Nutzerrechte in den jeweiligen Diensten verwaltet werden, da zum Beispiel ein Nutzer unterschiedliche Berechtigungen erhalten kann. Außerdem kann ein Nutzer zusätzliche Daten erhalten, die unterschiedlich von den jeweiligen Diensten interpretiert werden. Aus diesen Gründen ist es sinnvoll, nur den reinen Anmeldevorgang durch den Login-Service umzusetzen.

3.3.8 Zusammenfassung

In diesem Abschnitt wurden verschiedene Technologien mit den zuvor definierten Anforderungen verglichen. Dabei zeigte sich, dass keine der Technologien alle Anforderungen erfüllt. Zudem gibt es auch bestimmte Merkmale, die bisher nicht betrachtet wurden. Beispielsweise weisen alle zentralisierten Verfahren eine verminderte Verfügbarkeit auf. Wird nur ein Server eingesetzt, kann das ganze System nicht mehr genutzt werden, wenn dieser ausfällt. Bei einer verteilten Struktur, können andere Server dessen Aufgabe übernehmen. Das Gleiche gilt im

Falle einer Überlastung des Servers. Fast alle Technologien gewährleisten Vertraulichkeit und Integrität der Daten durch die Verwendung einer verschlüsselten Kommunikation mit HTTPS. Auf diese Weise soll die Implementierung dieser Verfahren vereinfacht und auch Sicherheitslücken minimiert werden. Denn die Verwendung eines einzelnen, jedoch erprobten Verfahrens, wie HTTPS durch TLS, führt zu einer insgesamt höheren Sicherheit.

In Tabelle 3.10 werden noch einmal alle Technologien zusammenfassend gegenübergestellt. Die Eigenentwicklung wird aus Platzgründen in dieser Tabelle mit dem Namen *Custom* bezeichnet. Die Übersicht verdeutlicht noch einmal, dass keine Technologie alle Anforderungen gänzlich erfüllt.

Anforderung	OpenID	OIDC	SAML	CAS	Custom
Zentralisierter Login-Server	-	-	o	+	+
Anonyme Nutzung	o	o	o	+	+
Eingabe einer PIN	o	o	+	-	+
Möglichkeit des Imports	-	-	o	+	+
Nutzung durch andere Einrichtungen	+	+	+	+	+
Einsatz weiterer Dienste	+	+	o	+	+
Codeänderungen an AMCS	o	+	-	+	+
Logout	-	+	+	+	o
Passwort ändern	o	o	o	-	-
Nutzer sperren	-	-	-	-	-
Nutzerdaten-Overhead	o	o	-	+	+

Legende: + = unterstützt, o = teilweise unterstützt, - = nicht unterstützt

Tabelle 3.10 Übersicht zum Vergleich ausgewählter Technologien für Single Sign-on

3.4 WAHL EINER GEEIGNETEN SINGLE-SIGN ON LÖSUNG

Nachdem die verschiedenen Technologien im Kapitel 3.3 miteinander verglichen wurden, soll in diesem Kapitel eine der Technologien ausgewählt werden. Dazu werden zunächst die Anforderungen gewichtet und danach ausgewertet. Im Anschluss daran werden einzelne Anforderungen der übrigen Technologien diskutiert, um letztendlich ein Ergebnis zu erhalten.

Die Anforderungen aus Kapitel 3.3 lassen sich in bestimmte Muss- und Kann-Kriterien unterteilen, die sich durch das Ziel dieser Arbeit erschließen. Dadurch ergibt sich folgende Unterteilung:

Muss-Kriterien	Kann-Kriterien
Zentralisierter Login-Server	Eingabe einer PIN
Anonyme Nutzung	Möglichkeiten des Imports
Nutzung durch andere Einrichtungen	Codeänderungen an AMCS
Einsatz weiterer Dienste	Logout
	Passwort ändern
	Nutzer sperren
	Nutzerdaten-Overhead

Tabelle 3.11 Unterteilung der Anforderungen für Single Sign-on Lösungen

Die Anzahl der Muss-Kriterien ist deutlich geringer als die der Kann-Kriterien. Aus diesem Grund fällt die Entscheidung gegen einige Singe Sign-on Lösungen leicht. Die Entscheidungen werden anhand der Tabelle 3.10 getroffen.

Der zentralisierte Servereinsatz, als auch die anonyme Nutzung werden von OpenID, OpenID Connect und SAML nicht oder nur zum Teil unterstützt. Damit scheidet bereits ein Großteil der ausgewählten Lösungen aus. Übrig bleiben CAS und die Eigenentwicklung (Custom). Bei den letzten beiden Kandidaten ist eine Entscheidung schon deutlich schwieriger, denn beide erfüllen die Muss-Kriterien. Selbst die Kann-Kriterien sind nahezu identisch. Die einzigen Unterschiede ergeben sich durch die Logout-Anforderung und der Eingabe einer PIN.

Die Entscheidung zwischen beiden Lösungen sollte jedoch nicht zu voreilig getroffen werden, denn die Tabelle liefert keine genaueren Details.

Für CAS spricht, dass sich jeder bei der Weiterentwicklung beteiligen kann. Apereo stellt dazu ein Framework für CAS auf GitHub[38] bereit. Zudem spricht auch die Abmeldefunktionalität für CAS. Gegen CAS spricht, dass die Frameworks meist überladen sind. Es wird also nicht nur das CAS Protokoll implementiert, sondern eine Vielzahl weiterer Protokolle, was dazu führt, dass der Aufwand für die Installation, Verwendung und Modifikation steigt. Es werden auch mehr Speicherplatz und eventuell mehr Rechenleistung benötigt. Viele Implementierungen

[38] Siehe auch https://github.com/apereo/cas (aufgerufen am 04.06.2018)

bieten keine Unterstützung der Server-Seite. Es werden also nur clientseitige Implementierungen angeboten, die mit einem CAS-Server kommunizieren können.

Das CAS Protokoll bietet keine Möglichkeit, dem Nutzer eine Zustimmungserklärung anzuzeigen, bevor er durch den Login-Server mit Single Sign-on automatisch angemeldet wird. Des Weiteren ist auch eine Modifikation des Protokolls notwendig, um die Eingabe einer PIN für Lehrveranstaltungen in AMCS zu ermöglichen. Eine derartige Modifikation ist zwar möglich, jedoch nicht wünschenswert, da dadurch das Protokoll verändert wird und nicht mehr dem definierten Standard entspricht. Der eben genannte Punkt erlangt umso mehr an Bedeutung, weil die Arbeitsweise der Eigenentwicklung dem von CAS sehr ähnlich ist. Die Eigenentwicklung ist jedoch kein Standard und kann deshalb beliebig verändert werden.

	Central Authentication Service	Eigenentwicklung
Pro	- Open Source Community - Globale Abmeldefunktionalität	- Beliebig anpassbar, da kein offizieller Standard (Protokollfluss, PIN-Eingabe, Passwortänderung, Nutzersperrung, usw.) - Speziell für diese Arbeit entwickelt - Beeinflussbar durch Evaluation - Nur notwendige Funktionen werden implementiert
Contra	- Anpassungen weichen vom Protokollstandard ab - Frameworks sind überladen - Wenig Frameworks für Serverseite	- Nur lokale Abmeldefunktionalität - Keine Open Source Community

Tabelle 3.12 Gegenüberstellung des Für und Wider von CAS und der Eigenentwicklung

Die Nachteile von CAS sind praktisch die Vorteile der Eigenentwicklung. Sie ist sehr einfach und unterstützt nur wirklich notwendige Funktionen. Aus diesem Grund ist sie auch sehr gut an das Ziel dieser Arbeit anpassbar. Auch der Overhead bei der Kommunikation kann auf das Notwendigste reduziert werden. Fehlende Anforderungen, wie Passwortänderung oder Nutzersperrung, können nachträglich eingebaut werden.

Die Evaluation durch Testprobanden kann zur Verbesserung und somit Modifikation der Eigenentwicklung beitragen. Falls notwendig kann auch der Protokollablauf entsprechend verändert werden. Der einzige Punkt, der wirklich gegen die Eigenentwicklung spricht ist, dass es kein Open Source Projekt ist, bei dem jeder mitarbeiten kann.

Alle Argumente, die für oder gegen CAS bzw. die Eigenentwicklung sprechen, werden in Tabelle 3.12 noch einmal zusammengefasst und gegenübergestellt.

3.5 SINGLE SIGN-OFF

Im bisherigen Verlauf, wurde fast ausschließlich das Verhaltensmuster des Single Sign-on betrachtet. Das genaue Gegenteil ist Single Sign-off oder auch Single Log-out (SLO). Nach einigen Recherchen wurde klar, dass es keine ultimative Lösung gibt, um Single Sign-off zu gewährleisten.

3.5.1 Problematik und mögliche Lösungsansätze

Es gibt zahlreiche Ansätze und einige Technologien, wie CAS, OpenID oder SAML, welche bereits eine derartige Funktionalität unterstützen. Der angemeldete Zustand eines Nutzers wird meist durch eine Session beschrieben. Indem die Session des Nutzers zerstört wird, wird auch der Nutzer abgemeldet. Das Problem dabei ist, dass bei Single Sign-off die Sessions aller mit dem Login-Service in Verbindung stehenden Dienste bzw. Webanwendungen zerstört werden müssen. Der Login-Service muss also seine Dienste kennen und auch erreichen können.

Zunächst muss also das Problem der Erreichbarkeit gelöst werden. Hierzu gibt es auch wieder verschiedene Ansätze. Der einfachste Ansatz bezüglich der Eigenentwicklung ist, bei der Single Sign-on Anmeldung die Rücksendeadresse der Webanwendung zu speichern. Beispielsweise könnten zu jedem Pseudonym alle Rücksendeadressen der angemeldeten Webanwendungen in einer Datenbank gespeichert werden. Wird das Abmeldeereignis ausgelöst, kann der Login-Service z.B. per HTTP-POST eine Abmeldeanfrage an die jeweiligen Rücksendeadressen senden. Die Webanwendungen müssen dann bei der Rücksendeadresse auf diese Anfrage reagieren können. Die Anfrage könnte bestimmte Parameter enthalten, die der Webanwendung mitteilen, dass ein spezifischer Nutzer abgemeldet werden soll. Genau an dieser Stelle tritt allerdings das zweite Problem auf.

Woher weiß die Webanwendung, welche Nutzersession genau zerstört werden muss? Und wie wird garantiert, dass möglichst nur der Login-Service das Recht hat, einen Nutzer per Anfrage abzumelden? Nun, die Antworten auf diese Fragen sind abermals uneindeutig. Zunächst muss an dieser Stelle bekannt sein, welche Session-ID zu dem gewünschten Nutzer gehört. Aus sicherheitstechnischer Sicht wäre es problematisch, wenn der Login-Service bereits die gesuchte Session-ID des Nutzers kennen würde. Deshalb muss diese auf einem anderen Weg identifiziert werden. Die Session-ID anhand des Nutzerpseudonyms zu ermitteln

ist auch keine gute Lösung, was die zweite der beiden oben genannten Fragen wieder aufwirft. Am sinnvollsten wäre es, den Nutzer durch einen Identifikator zu repräsentieren, den nur der Login-Service und nur die jeweiligen Webanwendungen kennen.

Aus Abbildung 3.8 geht hervor, dass die letzte Antwort des Login-Servers die Bestätigung der Verifikation und die Übermittlung des Nutzerpseudonyms ist. Die Antwortnachricht könnte um einen weiteren Parameter erweitert werden, welcher den Identifikator des Nutzers beinhaltet. Der Login-Server generiert zuvor diesen Identifikator und speichert ihn zusammen mit der Rücksendeadresse der Webanwendung ab. Die Webanwendung speichert sich den Identifikator zusammen mit der Session-ID des Nutzers ab. Sendet der Login-Service eine Abmeldeanfrage mit dem Identifikator als Parameter, kann die Webanwendung anhand des Identifikators die Session-ID des Nutzers erfahren und die Session zerstören, was wiederum zur Abmeldung des Nutzers führt.

3.5.2 Zusammenfassung eines Lösungsansatzes

Die Eigenentwicklung kann um einen Identifikator erweitert werden, der zur Findung und Zerstörung der Session-ID des Nutzers genutzt wird. Dazu speichert der Login-Server zu jeder erstmaligen Anmeldung des Nutzers bei einer Webanwendung die Rücksendeadresse und den Identifikator, der zuvor generiert wird. Nach der Verifizierungsanfrage der Webanwendung antwortet der Login-Server mit der Bestätigung, dem Nutzerpseudonym und dem Identifikator.

Die Webanwendung speichert ihrerseits den Identifikator zusammen mit der Session-ID des Nutzers ab. Am einfachsten geschieht das mit der Verwendung einer Datenbank. Erhält die Webanwendung später eine Abmeldeanfrage vom Login-Server, wird der darin enthaltene Identifikator genutzt, um die dazugehörige Session-ID aus der Datenbank abzufragen. Die entsprechende Session wird anhand der Session-ID zerstört und der Datenbankeintrag entfernt. Der Login-Server löscht nach Senden der Anfrage ebenfalls den entsprechenden Eintrag.

Mit dieser vorgeschlagenen Lösung ist es möglich, einen Nutzer bei mehreren Webanwendungen gleichzeitig abzumelden und somit Single Sign-off zu unterstützen. Auf diese Weise wird auch der zusätzliche Datenverkehr auf ein Minimum gehalten, da im Prinzip nur eine zusätzliche Nachricht pro genutzter Webanwendung gesendet werden muss. Zum Abschluss wird noch beispielhaft eine entsprechende Abmeldeanfrage über HTTP-POST dargestellt.

Login-Server

```
POST / HTTP/1.1
Host: app1.de
Content-Type: application/x-www-form-urlencoded
Content-Length: 32
    method=logout
    &identifier=1234xyz
```

Abbildung 3.13 Beispielnachricht einer Anfrage an eine Webanwendung zur Abmeldung eines spezifischen Nutzers durch den Login-Server

4 IMPLEMENTIERUNG

In diesem Kapitel geht es darum, die Eigenentwicklung, welche in Kapitel 3.1.2 vorgestellt wurde, als Login-Service zu implementieren. Des Weiteren soll im Anschluss AMCS so modifiziert werden, dass die implementierte Single Sign-on Lösung zur Anmeldung genutzt werden kann. Die Eigenentwicklung beschreibt unter anderem die Kommunikation zwischen dem Login-Server und der Webanwendung. Doch um erste Tests durchführen zu können, soll zunächst die Serverseite implementiert werden.

4.1 IMPLEMENTIERUNG DES LOGIN-SERVICES

Für eine einfachere Implementierung des Login-Servers werden relevante Funktionen zusammengetragen. Zunächst werden die öffentlichen Funktionen betrachtet:

Der Login-Server muss Funktionalitäten zur Anmeldung eines Nutzers mit Pseudonym und Passwort bereitstellen, weiterhin zur Abmeldung eines Nutzers, zur Verifizierung eines Tokens von einer Webanwendung und zur Prüfung und Wiederherstellung einer Nutzersession.

Die internen Funktionen werden von den öffentlichen genutzt und sind teilweise aus den öffentlichen Funktionen herleitbar. Nutzer verfügen über ein Pseudonym und ein Passwort. Diese müssen persistent gespeichert werden. Im Webbereich ist das am einfachsten mit einer Datenbank umsetzbar. Verschiedene Nutzer müssen in der Datenbank gespeichert, gelesen und verändert werden können.

Es existiert eine Session, die sitzungsbezogene Daten speichert. Das können der Anmeldezustand des Nutzers, sowie das Nutzerpseudonym und ein Zeitstempel sein. Besucht ein Nutzer die Anmeldeoberfläche, soll dieser anhand der Session wiedererkannt werden. Um auch weitere Informationen über den Nutzer abrufen zu können, wird das zuvor gespeicherte Pseudonym des Nutzers aus der Session genutzt. Mit dem Pseudonym können weitere Daten über die Datenbank erfragt werden. Der Login-Server muss also zusätzlich eine Session mit nutzerbezogenen Daten erstellen, speichern und zerstören können.

Zusätzlich wird eine zentrale Stelle benötigt, welche die nutzerbezogenen Datenbankabfragen regelt. Sie muss einen neuen Nutzer erstellen, sowie Nutzer anhand des Pseudonyms ausgeben können. Weiterhin wird das Token zur Anmeldung nur in Verbindung mit dem jeweiligen Nutzer erstellt, damit nur dieser sich gegenüber der Webanwendung authentifizieren kann. Nutzer müssen also zusätzlich durch das Token ausgegeben werden können. Außerdem muss es eine Möglichkeit geben, ein Token für einen Nutzer auszustellen und zu verifizieren.

Die zusammengetragenen Informationen lassen sich nun bildlich vereinfacht darstellen.

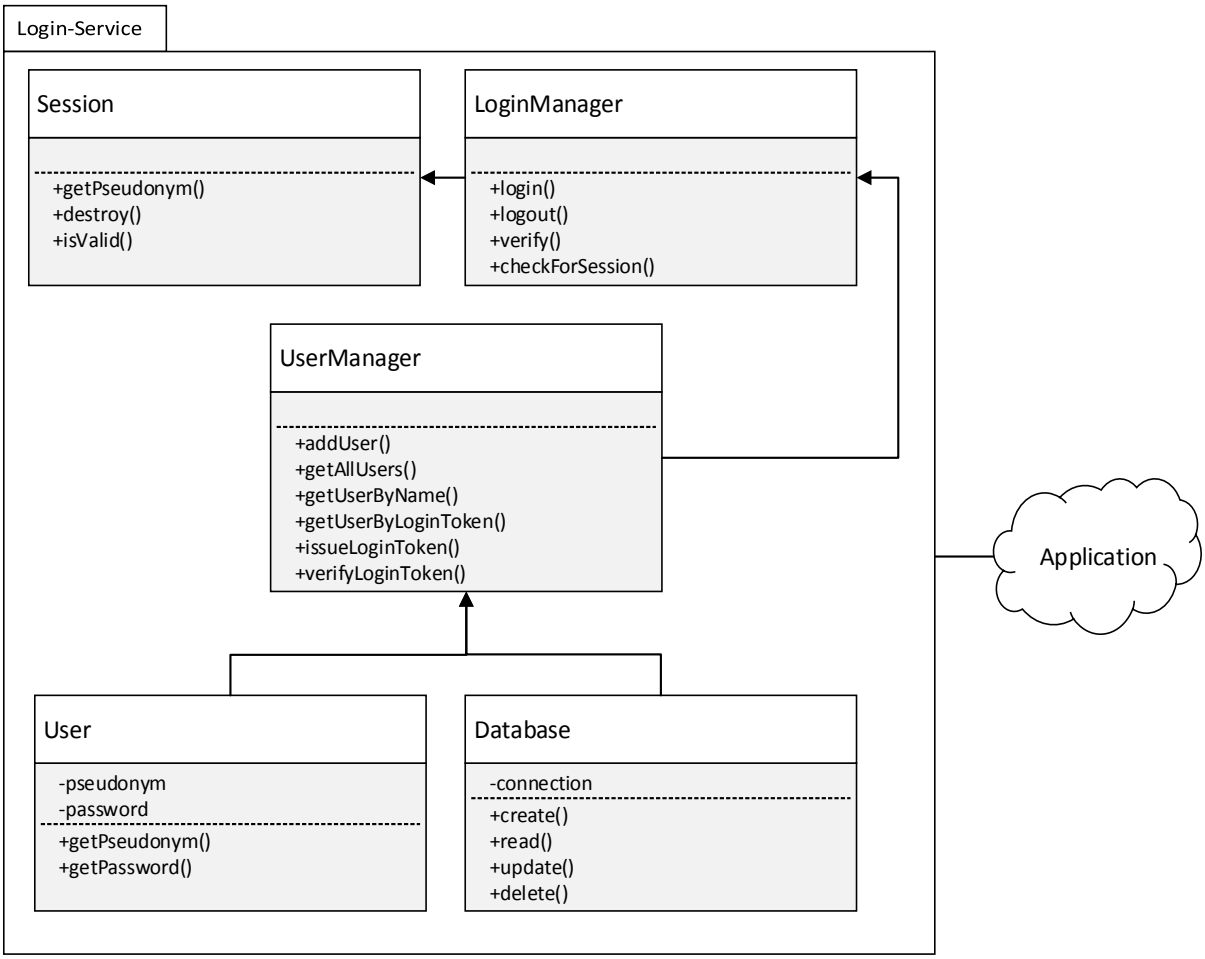


Abbildung 4.1 Vorüberlegungen zum Aufbau des Login-Servers

4.1.1 Wahl einer geeigneten Programmiersprache

Im nächsten Schritt geht es darum, eine geeignete Programmiersprache für die Implementierung zu finden. Ein passender Kandidat kann nach unterschiedlichen Kriterien gewählt werden. Ein sinnvoller Ansatz ist der Vergleich von Programmiersprachen nach der Verwendungshäufigkeit. Im Internet gibt es zahlreiche Statistiken, welche versuchen die Sprachen in einer Rangordnung einzusortieren. Ein bekanntes Beispiel ist der *TIOBE Index* [39], welcher die Sprachen anhand mehrerer Kriterien, wie zum Beispiel anhand von Suchmaschinen oder der weltweiten Anzahl von erfahrenen Entwicklern, berechnet. Demnach stehen Programmiersprachen wie Java, C und C++ an oberster Stelle. Die erste Sprache, die speziell für die Entwicklung von Webanwendungen gedacht ist, ist PHP. Für den Juni 2018 steht PHP auf dem siebten Platz. PHP ist sehr weit verbreitet und fast jeder Web Hoster bietet PHP-Server relativ

[39] Siehe auch https://www.tiobe.com/tiobe-index/ (aufgerufen am 16.06.2018)

günstig an. Aus diesen Gründen wird die Skriptsprache PHP zur Implementierung des Servers des Login-Services genutzt.

4.1.2 Wahl der geeigneten Entwicklungswerkzeuge

Zum Entwickeln von Webanwendungen mit PHP sind ein PHP Server, ein HTTP Server und eine Datenbank notwendig. Da die Entwicklung auf einem Windows-System geschieht, eignet sich zum Beispiel der Einsatz XAMPP[40] sehr gut. XAMPP wurde von Apache Friends entwickelt und soll der möglichst einfachen und schnellen Entwicklung von Webanwendungen dienen. Dabei besitzen die Buchstaben verschiedene Bedeutungen. Das X steht für den Einsatz unter verschiedenen Betriebssystemen wie Windows, Linux und Mac. Das A für den integrierten HTTP Server Apache, M für das Datenbankverwaltungssystem MariaDB (früher MySQL) und die zwei P für die Skriptsprachen Perl und PHP. (vgl. XAMPP – Wikipedia, 2018)

In der gewählten XAMPP Version enthalten ist PHP 7.2.1, Apache 2.4.29 und MariaDB 10.1.30. Zur Programmierung werden der Texteditor Notepad++ 7.5.4[41] und die integrierte Entwicklungsumgebung Eclipse Neon 4.6.3[42] eingesetzt. Als darunter liegendes Betriebssystem kommt Microsoft Windows 7 Ultimate 64 Bit SP1 zum Einsatz.

4.1.3 Implementierung

Für die lokale Entwicklung ist es hilfreich eine virtuelle Domain für den Login-Server zu erstellen. Dazu muss der Apache Server und Windows entsprechend angepasst werden. Zuerst wird Apache angepasst. Dazu kann die Konfigurationsdatei für virtuelle Hosts unter „…\xampp\apache\conf\extra\httpd-vhosts.conf“ wie folgt angepasst werden:

[40] Siehe auch https://www.apachefriends.org (aufgerufen am 18.06.2018)
[41] Siehe auch https://notepad-plus-plus.org/ (aufgerufen am 18.06.2018)
[42] Siehe auch https://www.eclipse.org/ (aufgerufen am 18.06.2018)

httpd-vhosts.conf

```
<VirtualHost *:80>
    DocumentRoot ".../xampp/htdocs/LoginService/src/login"
    ServerName login-service.de
    ErrorLog "logs/login-service.de-error.log"
    CustomLog "logs/login-service.de-access.log" common
    <Directory ".../xampp/htdocs/LoginService/src/login">
        Order allow,deny
        Allow from all
    </Directory>
</VirtualHost>

<VirtualHost *:443>
    DocumentRoot ".../xampp/htdocs/LoginService/src/login"
    ServerName login-service.de
    SSLEngine On
    SSLCertificateFile "conf/ssl.crt/server.crt"
    SSLCertificateKeyFile "conf/ssl.key/server.key"
    <Directory ".../xampp/htdocs/LoginService/src/login">
    Order allow,deny
    Allow from all
    </Directory>
</VirtualHost>
```

Abbildung 4.2 Beispielkonfiguration des Apache Servers für eine virtuelle Domain

Danach muss noch die Windows HOSTS-Datei angepasst werden, damit Windows weiß, auf welche IP-Adresse die virtuelle Domain „login-service.de“ abgebildet werden muss. Andernfalls würde Windows versuchen den Domainnamen mithilfe eines DNS-Servers im Internet in eine IP-Adresse zu übersetzen. Da die Domain jedoch nur lokal existiert, wird entweder keine oder eine falsche IP-Adresse zurückgegeben. Dazu muss die HOSTS-Datei, welche sich zum Beispiel unter „C:\Windows\System32\drivers\etc\hosts“ befindet, um nur eine Zeile erweitert werden.

hosts

```
...
127.0.0.1 login-service.de
```

Abbildung 4.3 Erweiterung der Windows HOSTS-Datei, um den Domainnamen auf die eigene IP-Adresse abzubilden

Im nächsten Schritt wird mit dem Anlegen der „index.php“ der Einstiegspunkt für den Login-Server erstellt. Dessen Aufgabe ist es, gültige einkommende Anfragen (HTTP-Requests) zu verarbeiten und entsprechend darauf zu reagieren. In allen anderen Fällen soll erst einmal

einfach nichts passieren. In diesen Fällen kann natürlich auch eine entsprechende Rückmeldung erfolgen, doch für den ersten Prototyp soll das genügen.

index.php

```
// parse all requests and save output in data
$data = parseRequests();

// try to execute requested methods using data
executeRequests();

function parseRequests() { ... }

function getParam(string $fieldName) { ... }

function executeRequests() {
    // get reference address to which the client should be redi-
rected at the end
    $application = getParam('app');

    if (isset($application)) {
        ...

        // get the requested method type
        $method = getParam('method');

        // execute requested method
        switch ($method) {
            case 'login':
                // do login ...
                break;
            case 'validate':
                // do validate ...
                break;
            default:
                // do session checks ...
                break;
        }
    } else {
        // do nothing...
    }
}
```

Abbildung 4.4 Beispielaufbau des Einstiegspunktes des Login-Server

Um eine höhere Verfügbarkeit zu gewährleisten, sollen einkommende HTTP-Anfragen auch im JSON-Format akzeptiert werden. Dazu wird die Methode „parseRequests()“ definiert, welche zuerst das Header-Feld „Content-Type“ untersucht, sofern es denn vorhanden ist. Enthält es den Wert „application/json“, wird versucht der Nachrichteninhalt mit einem JSON-Parser zu dekodieren. Anderenfalls wird der Nachrichteninhalt als Klartext betrachtet und eventuell vorhandene HTTP-Request-Variablen daraus extrahiert. Die Abbildung 3.5 und Abbildung 3.6 zeigen im Prinzip alle vorkommenden Variablennamen. In der Methode „executeRequests()“ wird dann versucht, diese Variablen zu verarbeiten und die entsprechenden Methoden mithilfe

einer switch-Anweisung aufzurufen. Wurde keine gültige Methode gefunden, soll nur versucht werden eine evtl. vorhandene Session wiederherzustellen und auf Gültigkeit zu prüfen.

Da in PHP sowohl prozedural, als auch objektorientiert programmiert werden kann, werden die oben genannten Methoden direkt aufgerufen. Die Datei „index.php“ ist nun in etwa wie in Abbildung 4.4 aufgebaut.

Im Weiteren geht es darum, die anderen Bestandteile des Servers zu implementieren. Als nächstes wird die Datenbankanbindung implementiert. PHP stellt dazu verschiedene Methoden bereit, um datenbankspezifische Operationen auszuführen. Da PHP eine Vielzahl an Datenbanksystemen unterstützt, ist die Wahl nicht einfach. Am sinnvollsten wäre eine zusätzliche Abstraktionsschicht, um einheitliche Methoden für den Zugriff auf die Datenbanken bereitzustellen. Glücklicherweise stellt PHP bereits eine solche Abstraktion zur Verfügung. Mit der PHP Data Objects Erweiterung (PDO) werden verschiedene Datenbanktypen vereint und lassen sich so leichter ansteuern. Dies bedeutet, dass bereits ein entwickelter Quellcode nicht noch einmal komplett neu geschrieben werden muss, wenn das Datenbanksystem durch ein anderes ausgetauscht wird. Die Abstraktionsschicht ist allerdings sehr dünn. Das heißt, PDO bietet nur eine einheitliche Schnittstelle für den Datenzugriff. Die SQL Befehle, welche von den verschiedenen Datenbanksystemen unterstützt werden, sind nicht für alle Typen gleich.

Deshalb wird mit der Implementierung der Datenbankklasse versucht, die SQL-Befehle weitestgehend zu kapseln, damit alle notwendigen Operationen zumindest für die Datenbanksysteme MySQL, SQLite und PostgreSQL gleich funktionieren. Der Konstruktor der objektorientierten PDOs erlaubt verschiedene Parameter zur Herstellung einer Datenbankverbindung, wobei der erste mit dem Namen *Data Source Name (DSN)* der wichtigste ist. Mit diesem Parameter werden der Datenbanktyp und weitere Datenbankspezifische Parameter angegeben. Die Abbildung 4.5 zeigt einen Teil der Implementierung der Datenbankklasse und als Beispiel die Instanziierung mit verschiedenen DSN Parametern für die entsprechenden Datenbanktypen.

Database.php

```
...

// create SQLite3 database instance
$dsn = 'sqlite:databaseTest.db';
$dbSQLite3 = new Database($dsn, $username, $password);

// create MySQL database instance
$dsn = "mysql:host=$host;dbname=$database;charset=$charset";
$dbMySQL = new Database($dsn, $username, $password);

// create PostgreSQL database instance
$dsn = "pgsql:host=$host;port=5432;dbname=$database";
$dbPostgreSQL = new Database($dsn, $username, $password);

class Database
{
    private $pdo;

    public function __construct($dsn, $usr, $pwd, $opt = null)
    {
        try
        {
            $this->pdo = new PDO($dsn, $usr, $pwd, $opt);
            ...
        }
        catch (PDOException $e)
        {
            echo "error " . $e->getMessage();
        }
    }

    public function __destruct()
    {
        // close connection
        $this->pdo = null;
    }

    ...
```

Abbildung 4.5 Beispielinstanziierungen der Datenbanklasse von verschiedenen Datenbanktypen mithilfe von PDOs

Im restlichen Verlauf der Datenbankklasse werden alle Methoden implementiert, die zum Einfügen, Abfragen, Verändern und Löschen von Datenbankdaten notwendig sind. In den Methoden werden die SQL-Befehle gekapselt, um die Verwendung der Datenbank zu vereinfachen.

Als nächstes wird die Session Klasse implementiert. PHP verfügt bereits über eine vollständige Session-Verwaltung. Dabei werden verschiedene Methoden bereitgestellt, wie zum Beispiel die Fortsetzung, Zerstörung oder Neuinitialisierung einer Session. Außerdem können beliebige Daten serverseitig in der Session abgespeichert werden. Genau diese Aufgabe übernimmt die Session Klasse. Sie definiert den Zustand der Anmeldung und speichert weitere

nützliche Daten, wie dem Nutzerpseudonym und Zeitstempel. Bei der Initialisierung einer Session können auch hier weitere Parameter übergeben werden, um das Verhalten der Session zu beeinflussen. Die Session speichert eine Session-ID in einem Cookie auf der Clientseite. Um möglichst hohe Sicherheit zu gewähren, soll die PHP Session nur noch Cookies, die nur über HTTPS und mit HTTP-POST übertragen werden, speichern und akzeptieren. Eine derartige Konfiguration wird in Abbildung 4.6 gezeigt.

Session.php

```
$options = array(
// don't include the identi-
fier in the URL, and don't read the URL for identifiers.
'use_trans_sid' => false,
// never use URLs with session identifiers
'use_cookies' => true,
// specifies whether cookies should only be sent over secure connec-
tions
'cookie_secure' => true,
// prevents attacks involved passing session ids in URLs
'cookie_httponly' => true
);

// start new or resume existing session
session_start($options);
...
```

Abbildung 4.6 Beispielkonfiguration einer PHP Session, um die Sicherheit gegen Angriffe zu erhöhen

Zusätzlich wird eine interne Methode „update()“ implementiert, welche nach jeder Änderung der Session-Daten aufgerufen wird. Ihre Aufgabe ist es, die vorhandene Session zu zerstören und die neuen Daten in eine neue Session zu übertragen. Auf diese Weise wird jedes Mal eine neue Session-ID generiert, um Angriffe zusätzlich zu erschweren. Danach wird noch eine Methode „login()“ implementiert, welche den angemeldeten Zustand eines Nutzers in der Session speichert. Weiterhin sind noch ein paar „Getter-Methoden“ notwendig, um den Zustand und das Pseudonym des Nutzers in der Session von außerhalb zu erfragen.

Die Implementierung der User Klasse ist trivial, da sie im Prinzip nur einen Datencontainer darstellt.

Der User Manager ist hingegen schon deutlich komplexer. Er benötigt Zugriff auf die User und die Datenbank Klasse. Der User Manager ist dafür zuständig, neue Nutzer in der Datenbank anzulegen oder sie über verschiedene Wege von der Datenbank abzurufen. Dabei wird immer die User Klasse instanziiert, um ein User Objekt bereit zu stellen. Weiterhin kümmert sich der User Manager um den Aufbau der Tabellen für die Nutzer in der Datenbank, wenn diese noch nicht existieren. Auch die Erstellung und Validierung von Login-Tokens wird durch den User Manager übernommen.

Zum Schluss ist der Login Manager an der Reihe. Dieser benötigt Zugriff auf den Login Manager, der User Klasse und der Session. Der Login Manager implementiert im Prinzip alle Methoden, die dann durch den Einstiegspunkt, der „index.php“, aufgerufen werden. Also diejenigen Methoden, welche durch einkommende Anfragen in der switch-Anweisung aus Abbildung 4.4 ausgelöst werden sollen. Wenn sich ein Nutzer mit seinem Pseudonym und Passwort anmeldet, wird die Methode „login()“ aufgerufen. Sie überprüft die Gültigkeit der Eingaben, indem versucht wird Nutzerinformationen anhand des eingegebenen Pseudonyms von der Datenbank abzufragen. Wurde kein entsprechender Nutzer gefunden, wird ein neuer Datensatz in der Datenbank eingetragen. Wenn jedoch ein passender Datenbankeintrag gefunden wurde, müssen im nächsten Schritt das gespeicherte und das eingegebene Passwort miteinander verglichen werden. Dabei werden jedoch nur die Hash-Werte der Passwörter auf Gleichheit überprüft. Stimmen sie überein, wird in der Session Klasse vermerkt, dass der Nutzer nun erfolgreich angemeldet ist. Danach wird das Token für die Anmeldung bei der Webanwendung generiert und der Webbrowser des Nutzers zurück zur Webanwendung umgeleitet.

Der Login Manager bietet noch zwei weitere Besonderheiten. Zum einen ist es mit PHP durch eine einzige Zeile Code möglich, den Webbrowser des Clients umzuleiten. Doch dabei ist zu beachten, dass das PHP Skript weiterläuft und evtl. negative Auswirkungen haben kann. Mit einem weiteren Befehl, wie in Abbildung 4.7 zu sehen ist, lässt sich dieses Problem jedoch einfach umgehen.

LoginManager.php

```php
// redirect browser to url
header("Location: " . $url);

// prevent further execution when redirect
exit();
```

Abbildung 4.7 Umleitung und Abbruch der Skriptausführung mit PHP

Zum anderen soll der Login Manager eine Anmeldeoberfläche zum Webbrowser des Clients senden, wenn dieser nicht beim Login-Service angemeldet ist. Um, ähnlich dem Model-View-Controller-Prinzip, die Programmlogik von der Visualisierung zu trennen, werden in PHP oft sogenannte Template-Frameworks eingesetzt. Doch PHP bietet bereits eine ähnliche Funktionalität, auch wenn so die Trennung zwischen Programmlogik und Visualisierung vielleicht nicht ganz so elegant gelöst wird. Die Idee dahinter ist, dass Ausgaben nicht direkt an den Webbrowser gesendet, sondern in einem Puffer geschrieben werden. Der Inhalt des Puffers kann dann in einer Variable gespeichert werden. Auf diese Weise ist es sogar möglich, mehrere HTML-Templates ineinander zu verschachteln und so zu einer gesamten Webseite zusammenzusetzen. Mit der PHP-Methode „extract()“ lassen sich sogar Array-Felder in einzelne Variablen umwandeln, um diese dann nur noch in den HTML-Templates auszugeben. Somit

wird die Programmlogik noch mehr abgetrennt. Die Abbildung 4.8 zeigt die Verwendung beider Ideen in der Methode „getLoginForm()“ im Login Manager, um eine komplette HTML Anmeldeoberfläche als String-Variable zurückzugeben.

LoginManager.php

```
private function getLoginForm(): string
{
    // set content type header to html
    header("Content-Type: text/html; charset=utf-8");

    // prepare data to use within the HTML code
    $data = array();
    $data['app'] = $this->application;
    $data['method'] = "login";

    // make array fields accessible as single variables
    extract($data);

    // start output buffering
    ob_start();

    // import HTML template
    include 'view/loginForm.htm';

    // output buffer content and clean it
    return ob_get_clean();
}
```

Abbildung 4.8 Beispielumsetzung eines einfachen Template Systems mit PHP

Da die Anmeldeoberfläche bisher das einzige HTML-Template ist, sind alle weiteren HTML-Daten wie der HTML-Header darin integriert. Um die visuelle Darstellung der Anmeldeoberfläche zu verbessern, werden zusätzlich *Cascading Style Sheets* und *JavaScripts* eingesetzt.

Ein erster Test mit *Postman* [43], mit einem zuvor generierten Token, lieferte das eingegebene Pseudonym. Das Ergebnis ist in Abbildung 4.9 dargestellt. Durch weitere Tests dieser Art lässt sich die Funktionalität des Login-Services einfach überprüfen.

[43] Postman ist eine Erweiterung für Google Chrome, um APIs zu testen. Siehe auch https://www.getpostman.com/ (aufgerufen am 21.06.2018)

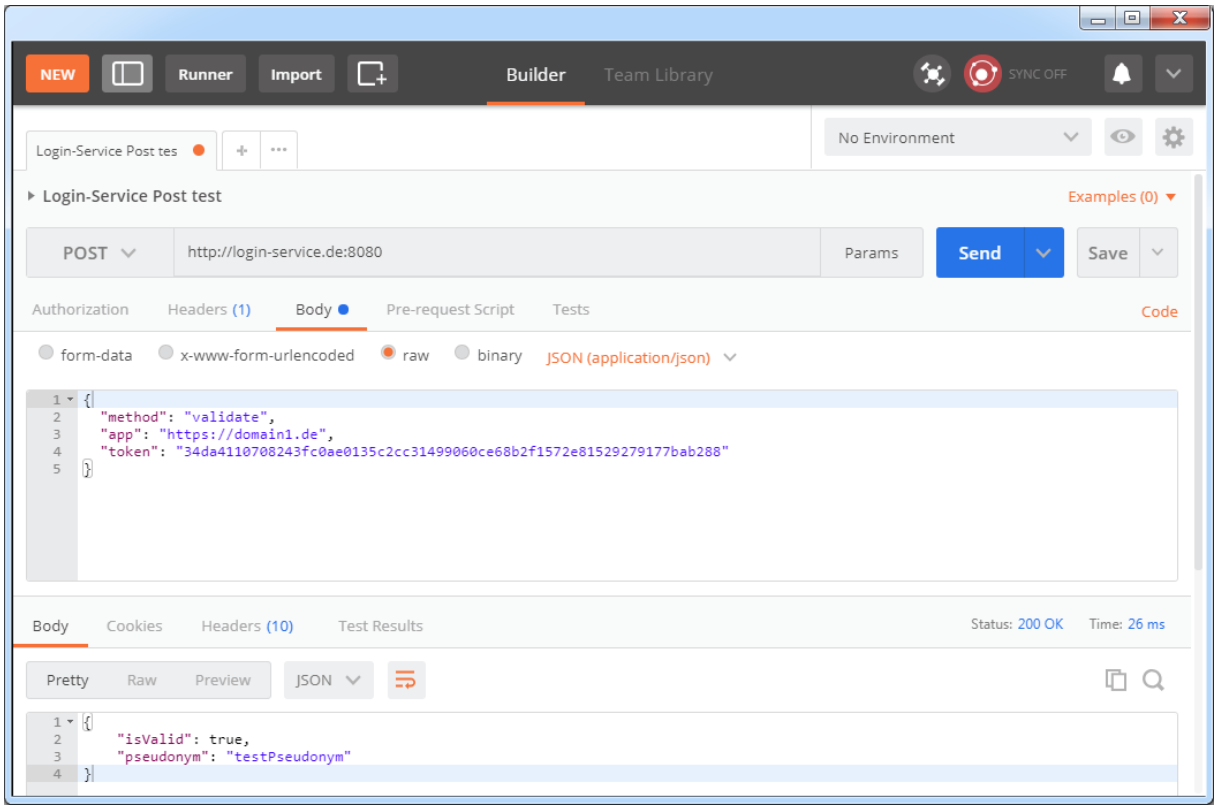


Abbildung 4.9 Ausschnitt einer Überprüfung der Funktionalität des Login-Services mit Postman

4.2 INTEGRIERUNG DES LOGIN-SERVICES IN AMCS

Wie bereits in Kapitel 2.3 erläutert, setzt sich AMCS aus einem Front-End und Back-End zusammen, die miteinander kommunizieren. Das Front-End besteht aus Modulen, die Komponenten enthalten. Das Back-End hingegen besteht aus Models und Controllern.

Da der Login-Service die zukünftige Anmeldeoberfläche bereitstellt, muss also der Anmeldeteil des Front-Ends ersetzt werden. Mit einem Klick auf „Einloggen“ soll der Nutzer nicht mehr auf das Login Modul, sondern auf den Login-Service verwiesen werden. Dadurch wird das Login Modul praktisch obsolet. Da das Home Modul die Verlinkung zum Login Modul bereitstellt, muss dieses modifiziert werden, um den Nutzer zum Login-Service umzuleiten.

Nach der Authentifizierung beim Login-Service soll der Nutzer zum Home Modul zurück umgeleitet werden. Das bedeutet, dass das Home Modul sich nun auch um die eingehenden Daten kümmern muss, die bei der Umleitung mitgesendet werden.

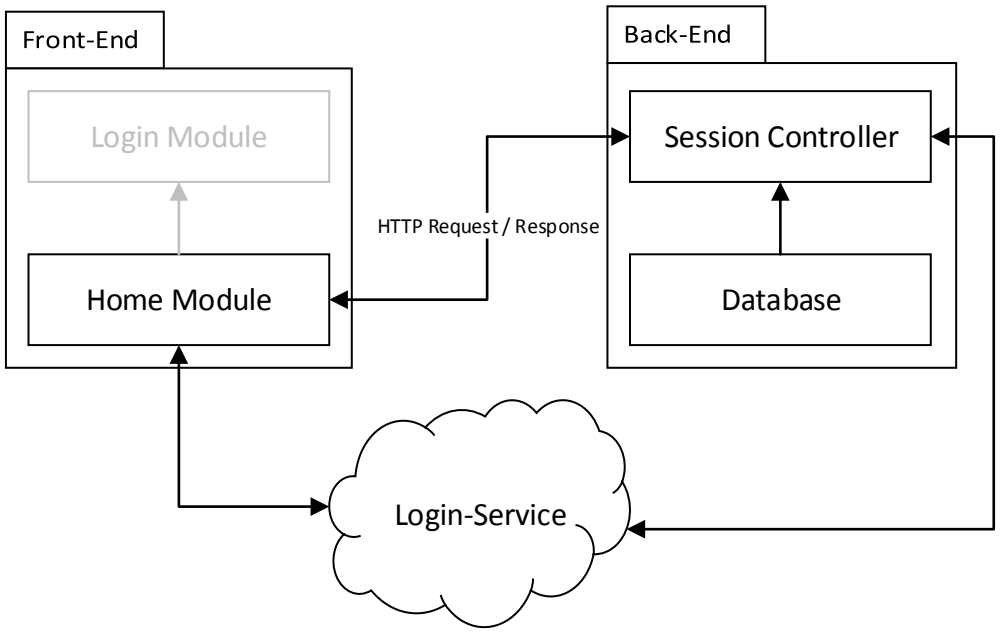


Abbildung 4.10 Schematische Darstellung der Kommunikation zwischen Front-End und Back-End in AMCS unter Nutzung des Login-Services

Das Front-End verfügt dann über das Login-Token vom Login-Service. Anstelle des Pseudonyms und des Passwortes wird dieses Token zum Back-End via HTTP-POST gesendet.

Da dieser Vorgang anders als zuvor verläuft, wird beim Back-End eine neue API Route erstellt. Die neue Route ist fast die gleiche wie zuvor. Wie in Abbildung 4.10 verdeutlicht wird, landen die Daten genauso beim Session Controller des Back-Ends, nur dass eine andere Methode als zuvor aufgerufen wird.

Beispielanfragen der verschiedenen API Routen des Front-Ends an das Back-End werden in der Abbildung 4.11 bzw. Abbildung 4.12 veranschaulicht.

Bisherige API Route

```
POST /api/auth/authenticate HTTP/1.1
HOST: amcs.website
content-type: application/json
x-amcs-api: 3
content-length: 79
{
  "auth": {
    "username": "USERNAME",
    "password": "PASSWORD"
  }
}
```

Abbildung 4.11 Beispielanfrage zur Anmeldung des Nutzers über die bisherige API Route

Neue API Route für Single Sign-on

```
POST /api/auth/authenticateSSO HTTP/1.1
HOST: amcs.website
content-type: application/json
x-amcs-api: 3
content-length: 103
{
  "auth": {
    "token":
"c54092832203d91951668e7f531687b9644caab8de42c4cfd6976ddc83579dcb"
  }
}
```

Abbildung 4.12 Beispielanfrage zur Anmeldung des Nutzers über die neue API Route für Single Sign-on

Das Back-End verfügt jetzt über das Token vom Login-Service. Es benötigt allerdings noch eine Gültigkeitsbestätigung des Tokens vom Login-Service und das Pseudonym des Nutzers. Deshalb muss als nächstes eine HTTP-POST-Anfrage vom Back-End an den Login-Service gesendet werden. Gibt der Login-Service sein O.K. und liefert er auch das benötigte Pseudonym, kann das Back-End folglich nach Datenbankeinträgen suchen. Werden keine Einträge und somit auch kein Nutzer gefunden, wird ein neuer Nutzer erstellt. Das Problem an dieser Stelle ist, dass kein Passwort des Nutzers mehr vorhanden ist. Um die Funktionalität des Back-Ends nicht weiter zu beeinflussen, wird vorerst ein festes und immer gleiches Passwort für diese neue API Route verwendet. Wie bereits in Kapitel 2.3 erwähnt, werden im Anschluss daran die Hash-Werte der Passwörter auf Gleichheit überprüft. Stimmen sie überein, wird ein JSON Web Token (JWT) erzeugt und zurück zum Front-End gesendet. Dieses Token stellt die Session des Nutzers dar und wird für alle weiteren Anfragen an das Back-End genutzt. Der Nutzer wird so durch das Token gegenüber dem Back-End authentifiziert. Die Integrierung des Anmeldeverfahrens in AMCS ist damit abgeschlossen.

Front-End

```
...
onLogout() {
    // send logout request to backend and wait for response
    this.auth.logout().subscribe(
        data => {},
        error => {
            console.log(error);
        },
        () => {
           // destroy local session
           localStorage.removeItem('token');

           // reload page
           // router.navigatebyurl("/")

           // destory login service session by redirecting the browser
           this.redirectAndLogoutAtLoginService();
        });
}

redirectAndLogoutAtLoginService()
{
    // create redirect url
    var url = 'https://login-service.de/logout.php?app=';

    // create reference address
    var returnPath = window.location.protocol
       + '//'
       + window.location.host
       + window.location.pathname;

    // redirect to
    window.location.href = url + returnPath;
}
```

Abbildung 4.13 Auszug aus dem erweiterten Front-End zur abmeldung des Nutzers

An dieser Stelle fehlt nur noch die Abmeldung des Nutzers von AMCS und dem Login-Service. Bisher sendet Front-End zuerst eine Anfrage an das Back-End, um den Nutzer ebendort abzumelden. Verlief dies fehlerfrei, meldet das Front-End den Nutzer ebenfalls lokal ab, indem die Session zerstört wird. In diesem Fall wird der entsprechende Eintrag aus dem Local Storage des Browsers entfernt und die Webseite aktualisiert. Für die Abmeldung mit dem Login-Service muss lediglich die Aktualisierung der Webseite verhindert werden. Anstelle der Aktualisierung wird der Browser zum Login-Service umgeleitet, um auch dort die Session zu zerstören. Danach wird der Browser des Nutzers wieder zurück zum Front-End umgeleitet, was sozusagen der vorhergehenden Aktualisierung der Webseite entspricht.

4.3 PROBLEME

Während der Implementierungsphase sind diverse Probleme aufgetreten. Auch wenn die Einarbeitung in Angular und Ruby nicht ganz einfach war, gelang die Integrierung bzw. Erweiterung der Programmcodes in AMCS relativ unkompliziert. Das Front-End ließ sich ohne Schwierigkeiten installieren. Die eigentlichen Probleme sind jedoch beim Testen bzw. Ausführen des Back-Ends entstanden. Für die Ausführung der virtuellen Maschine wird *Docker*[44] und *VirtualBox*[45] bereitgestellt. Die Virtualisierung über VirtualBox ließ sich immer nur einmal starten. Danach musste die gesamte virtuelle Maschine wieder gelöscht werden. Jede Neuinitialisierung dauerte relativ lang und verlief nicht immer fehlerfrei. Daher war es manchmal ein Glücksspiel, das Back-End auf Anhieb zu starten. Des Weiteren traten Fehlermeldungen ohne genauere Details auf, die später auf das Fehlen einer Schlüsseldatei zurückzuführen waren.

Die Ausführung der virtuellen Maschine mit Docker benötigte erheblich weniger Zeit. Jedoch konnte der Startvorgang nie vollständig abgeschlossen werden, da es zu Problemen mit der Pfadfindung und Windows kam.

Die Problemfindung hat insgesamt ca. eine Woche in Anspruch genommen. Ohne die Hilfe durch die Mitarbeiter des Lehrstuhls hätte es wahrscheinlich noch länger gedauert. Schlussendlich wurde weiterhin VirtualBox verwendet, um das Testen der Implementierung zu ermöglichen.

4.4 ZUSAMMENFASSUNG

In diesem Kapitel wurde der Login-Service prototypisch implementiert. Nach ersten Tests wurde im Anschluss die Verwendung des Login-Service Prototyps in AMCS integriert. Da AMCS aus einem getrennten Front-End und Back-End besteht, erfolgte die Integrierung auf verschiedene Art und Weise und in verschiedenen Programmiersprachen. Durch die gemeinsame Kommunikationstechnologie über HTTP zwischen dem Front-End, Back-End und dem Login-Service war das Senden und Empfangen von Nachrichten relativ problemlos zu handhaben.

[44] Siehe auch https://www.docker.com/ (aufgerufen am 21.06.2018)
[45] Siehe auch https://www.virtualbox.org/ (aufgerufen am 21.06.2018)

5 EVALUATION

In diesem Kapitel wird der Login-Service Prototyp evaluiert. Die Evaluation besteht aus drei Teilen. Zuerst wird die Funktionalität untersucht. Danach werden Belastungs- und Reaktionstests durchgeführt. Anschließend findet die Nutzerbefragung statt.

Es werden verschiedene Erwartungen an den Prototypen gestellt, um die Funktionsweise zu testen. Da der Login-Service von mehreren Webanwendungen verwendet werden können soll, sollen Lasten- und Reaktionszeittests durchgeführt werden. Des Weiteren sollen Testprobanden das Verhalten und die Funktionalität des Prototyps beurteilen, um deren Ergebnisse zur Verbesserung verwenden zu können. Die Ergebnisse der Evaluation werden dann ausgewertet und diskutiert.

5.1 ERWARTUNGEN

Von dem Prototyp wird vollständige Funktionalität erwartet, die der des Konzepts der Eigenentwicklung und der Implementierung entspricht.

Des Weiteren soll untersucht werden, ob der Prototyp höheren Belastungen standhält. Das ursprüngliche Anmeldeverfahren von AMCS benötigt eine Anfrage und eine Antwort, um einen Nutzer anzumelden. Mit der Single Sign-on Technologie ist es rund das Dreifache. Wird nur die reine Kommunikation zum Back-End betrachtet, also ohne Anfragen an das Front-End, müsste der Anmeldevorgang schon mindestens drei Mal so lange dauern als zuvor.

Zusätzlich sollen Testprobanden den Prototyp selbst ausprobieren. Erwartet wird, dass sie den Test selbstständig, vollständig und aufrichtig durchführen. Zur Unterstützung bei eventuell auftretende Fragen oder Problemen, sollen die Tests beobachtet werden. Durch eine adäquate Auswertung der Tests sollte sich eine Tendenz in den Ergebnissen herauskristallisieren.

5.2 DURCHFÜHRUNG DER EVALUATION

In diesem Abschnitt werden die Evaluation durchgeführt und die Ergebnisse präsentiert und ausgewertet. Für jeden Teilbereich wird kurz das genauere Vorgehen und Ziel der Durchführung erläutert. Der Übersichtlichkeit halber werden die Ergebnisse direkt kurz diskutiert.

5.2.1 Überprüfung der Funktionalität

Die Funktionalität des Prototyps wird durch eine Reihe von HTTP--Anfragen getestet, indem für vorgegebene Parameter Anfragen an den Login-Service gesendet werden. Die Antworten werden dann ausgewertet und überprüft, ob sie den Erwartungen entsprechen.

Um die Überprüfung zu vereinfachen wird *ThoughtWorks Selenium* [46] in Kombination mit *JUnit* [47] eingesetzt. Selenium ist ein Framework, um automatisierte Softwaretests von Webanwendungen durchzuführen. JUnit hingegen kann allgemein zum automatisierten Testen von Java-Software genutzt werden. Für den Test wird die Programmiersprache Java genutzt. Da Eclipse bereits als Werkzeug zur Implementierung verwendet wurde, wird es auch an dieser Stelle eingesetzt. Der Einsatz von Java hat sich auch deshalb angeboten. Mithilfe von Selenium wird eine Webbrowser-Instanz erzeugt, welche die entsprechenden Webseiten aufruft und sogar selbstständig Formulare ausfüllt und absendet.

Im Folgenden werden verschiedene Testfälle definiert, welche den Login-Service überprüfen:

Testfall
1. Anfordern der Anmeldeoberfläche ohne Referenzadresse einer Webanwendung
2. Anfordern der Anmeldeoberfläche mit Referenzadresse
3. Erstellen eines neuen Nutzers
4. Anmeldung eines Nutzers mit PIN
5. Zu kurzen Pseudonyms
6. Zu langen Pseudonyms
7. Pseudonym mit Leerzeichen
8. Kein Pseudonym
9. Kein Passwort
10. Falsches Passwort

Tabelle 5.1 Testfälle zur Überprüfung der Funktionalität des Login-Services

Für jeden Testfall werden bestimmte Parameter erzeugt und die Antwort bzw. das Ergebnis ausgewertet. Enthält die Antwort HTML-Code, kann nach bestimmten Tags gesucht und dessen Inhalt geprüft werden. Im Falle des zehnten Testfalls aus der Tabelle 5.1 ergibt sich zum Beispiel folgender Java-Code.

[46] Siehe auch https://www.seleniumhq.org/ (aufgerufen am 30.06.2018)
[47] Siehe auch https://junit.org (aufgerufen am 30.06.2018)

Front-End

```
@Test
public void wrongPassword()
{
    driver.get(url + "?app=" + app);

    Assert.assertEquals("Login Service", driver.getTitle());

    // enter pseudonym
    WebElement element = driver.findElementById("pseudonym");
    element.sendKeys(pseudonym);

    // enter password
    element = driver.findElementById("password");
    element.sendKeys("somethingstupidxxyy");

    // submit the form. WebDri-
ver will find the form for us from the element
    element.submit();

    // check error message
    Assert.assertEquals("Login credentials are inva-
lid", driver.findElementById("error").getText());
}
```

Abbildung 5.1 Beispielcode zum Testen der Anmeldung mit einem falschen Passwort

Nachdem alle Testfälle implementiert wurden, kann mithilfe von JUnit das Testen automatisiert erfolgen. Die Ausgabeübersicht eines erfolgreichen Tests ist in Abbildung 5.2 dargestellt.

Alle Testfälle konnten erfüllt werden. Bei Eingabe ungültiger Parameter wurden die erwarteten Fehlermeldungen ausgegeben. Das Abmelden musste nicht explizit getestet werden, da es nach fast jedem Test ausgeführt wurde, um anschließende Tests erfolgreich durchführen zu können. Der Login-Service ist nicht JavaScript basiert. Alle Fehlermeldungen werden also durch den Login-Service selbst ausgegeben.

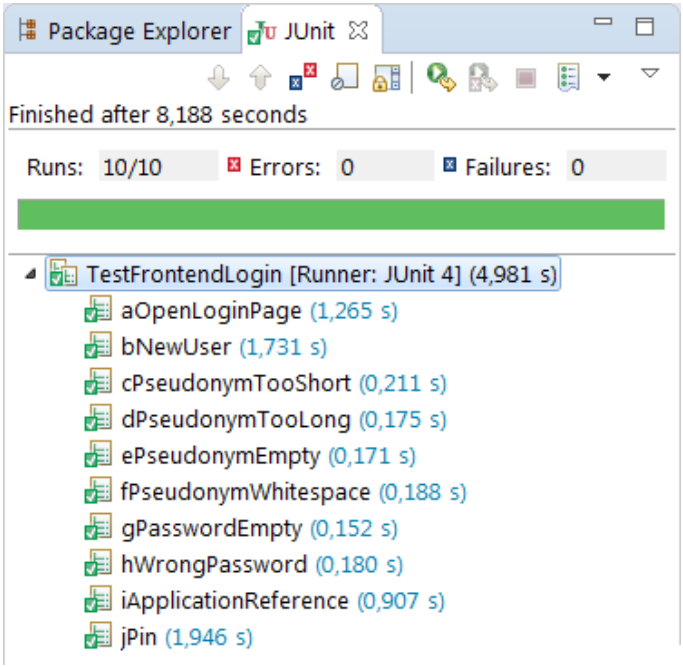


Abbildung 5.2 Erfolgreiche JUnit Web-Test des Login-Services

5.2.2 Belastungs- und Reaktionstests

Für die Belastungs- und Reaktionstests sind möglichst reale Bedingungen notwendig. Daher muss zuvor der Login-Service auf einen realen Webserver ausgelagert werden. Die Ergebnisse würden massiv beeinträchtigt werden, wenn der Login-Service wie bisher nur lokal auf einer virtuellen Domain betrieben wird.

Zu beachten ist, dass die Belastungs- und Reaktionstests nur für den Login-Service und das Back-End an sich betrachtet werden. Nach der Implementierungsphase ist aufgefallen, dass Umleitungen einen erheblichen negativen Einfluss auf die Ladezeiten des AMCS Front-End aufweisen. Das ist damit zu begründen, da das Front-End einer sogenannten Single-Page-Webanwendung entspricht. Bei jedem Aufruf, wie durch eine HTTP-Umleitung, müssen alle notwendigen Ressourcen erneut initialisiert werden. Die Dauer der Verzögerung ist somit von der Komplexität der Webanwendung abhängig. Da der Login-Service als Single Sign-on Lösung aber nicht nur für AMCS, sondern auch für weitere Webanwendungen gedacht ist, wird die Interaktion zum AMCS Front-End nicht mit in die Tests einbezogen. Die Verwendung von HTTP-Umleitungen ist aber notwendig, wie bereits im Abschnitt 3.1.1 erläutert. Nur so kann aber durch die Verwendung von Cookies die Sicherheit weiter erhöht werden. Eventuell könnten an dieser Stelle weitere Ideen, zur Zwischenspeicherung und Vermeidung zusätzlicher Kommunikation, in einer weiteren wissenschaftlichen Arbeit erforscht werden. Weiterhin werden auch alle Nutzereingaben nicht mit einbezogen.

Testablauf

Der Testablauf wird in zwei Bereiche unterteilt. Zuerst wird die Kommunikation zum bisherigen Anmeldeverfahren untersucht. Dazu werden HTTP-POST-Anfragen mit Pseudonym und Passwort zum Back-End gesendet. Das Back-End sendet, nach der Verarbeitung der Anfrage, ein JWT zurück zum Anfragenden. Die Abbildung 5.3 stellt den Ablauf bildlich dar.

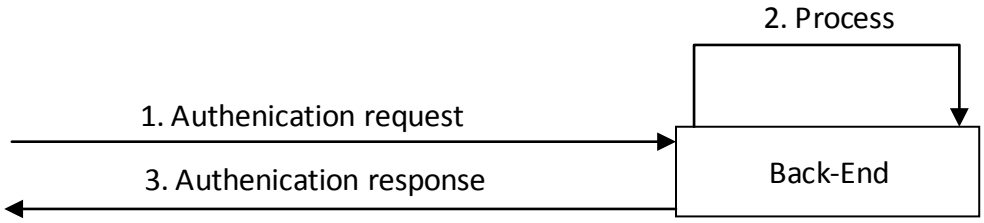


Abbildung 5.3 Testablauf des bisherigen Anmeldeverfahrens

Danach wird die Kommunikation zur Anmeldung mit Single Sing-on untersucht. Der Ablauf ist deutlich komplexer, wie in Abbildung 5.4 zu sehen ist. Zunächst muss das Anmeldeformular vom Login-Service angefordert werden. Dieser Schritt ist deutlich zeitintensiver als beim bisherigen Anmeldeverfahren und wird deshalb mit bei den Tests berücksichtigt. Im Anschluss daran, werden die entsprechenden Formulardaten zum Login-Service gesendet, um den Nutzer zu authentifizieren. Der Login-Service antwortet daraufhin mit einem Login-Token. Dieser wird dann zum Back-End gesendet, welches die Gültigkeit des Tokens überprüft und auch mit einem JWT antwortet. Für alle Schritte wird eine einzelne Zeitmessung vorgenommen, um einen direkten Vergleich zur bisherigen Anmeldung darzustellen.

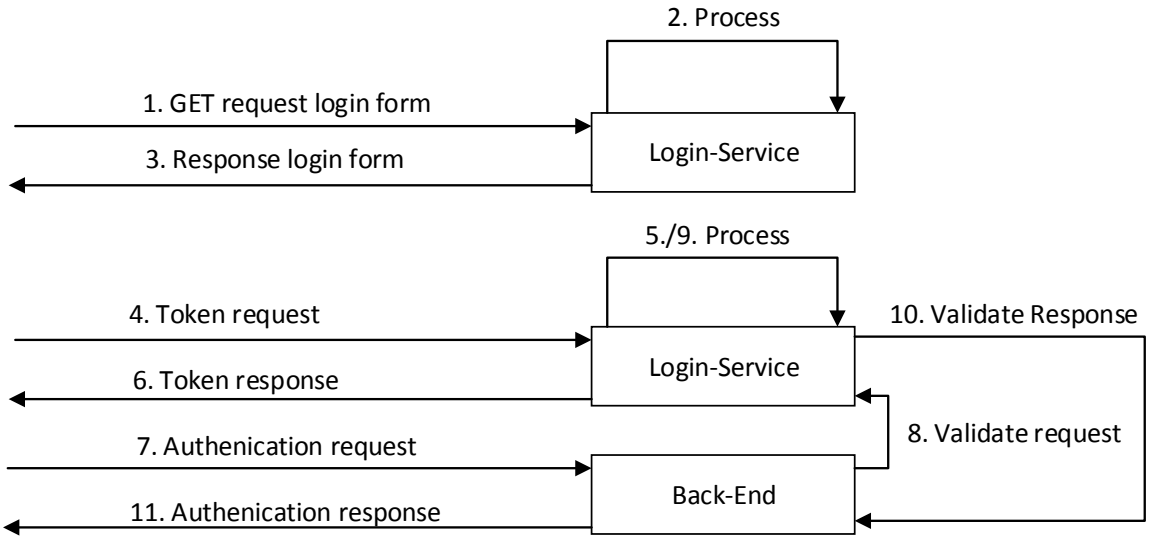


Abbildung 5.4 Testablauf des Anmeldeverfahrens mit Single Sign-on

Die Zeitmessungen erfolgen ab der Absendung einer Anfrage bis zum Empfang der Antwort und werden nachfolgend als Antwortzeit bezeichnet. Alle Tests werden mithilfe von *Apache JMeter*[48] durchgeführt, einem in Java geschriebenen Programm zur Ausführung von Lasttests in Client- und Serveranwendungen.

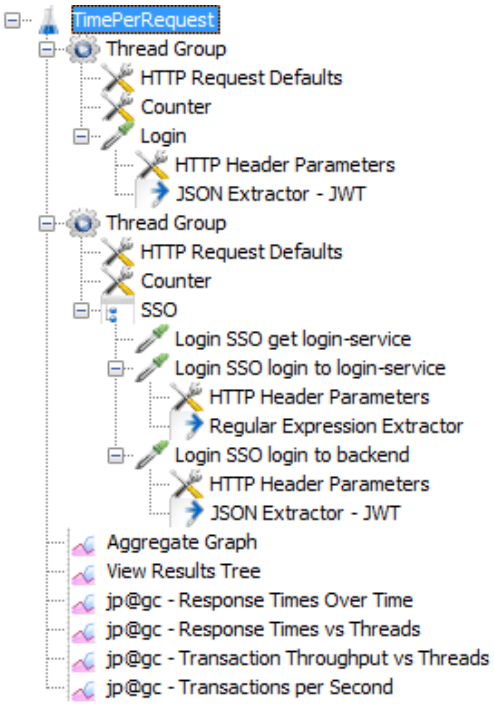


Abbildung 5.5 Testaufbau in JMeter

In JMeter werden sogenannte Thread-Groups erstellt, welche es durch die Erstellung von Threads erlauben, mehrere Anfragen parallel zu senden. Thread-Groups können dabei parallel oder sequentiell ausgeführt werden. Innerhalb der Thread-Groups werden die notwendigen Anweisungen und Parameter definiert. Listener werden dann verwendet, um die entsprechenden Zeitmessungen und Protokollierungen durchzuführen. In der zweiten Thread-Group, in der sich der Test für Single Sign-on befindet, wird ein sogenannter Transaction-Controller eingesetzt. Mithilfe des Controllers ist es möglich, mehrere Anfragen als eine zu betrachten und die verschiedenen Messwerte zu akkumulieren. Der Testaufbau in JMeter wird in Abbildung 5.5 noch einmal bildlich dargestellt.

Für möglichst realistische Testergebnisse werden der Login-Service und das Back-End auf den Staging-Server von AMCS betrieben. Dieser verfügt über vier virtuelle CPU-Kerne und 4GB RAM. Die physische CPU ist eine Intel Xeon E5-2620 v3, welche bei 2.40GHz betrieben wird. Die Messungen werden lokal ausgeführt. Die lokale Maschine ist per LAN mit 50Mbit/s an das Internet angebunden. Die durchschnittliche Paketumlaufzeit zu www.google.de wurde

[48] Siehe auch https://jmeter.apache.org/ (aufgerufen am 30.06.2018)

mit Ping ermittelt und beträgt 22ms. Eine Ermittlung durch Ping zum Staging-Server konnte nicht durchgeführt werden, da die TU Dresden scheinbar alle Ping-Anfragen ignoriert.

Belastungstest

Der Belastungstest soll ermitteln, wie viele Anmeldungen pro Sekunde durchgeführt werden können. Dazu werden die Thread-Groups in JMeter sequentiell ausgeführt. Beide versuchen dann 1000 Anmeldungen so schnell wie möglich durchzuführen. Für die Durchführung wurden zuvor 1000 neue Nutzer erzeugt, damit die Tests nur noch die reine Anmeldegeschwindigkeit messen.

Zur Durchführung werden genauso viele Threads erzeugt, welche stetig immer mehr Anmeldeanfragen zum Staging-Server senden. Innerhalb von 30 Sekunden werden so bis zu 1000 Anfragen parallel gesendet. Dabei haben sich folgende Werte ergeben:

Test	Anfragen	Fehlerrate	Durchsatz	Dauer
Login	1000	0,0 %	33,22 / s	30,10 s
Login-SSO	1000	0,0 %	25,9 / s	38,61 s

Tabelle 5.2 Ergebnisse des Belastungstests bei 1000 Anfragen

Eine Erhöhung der Anmeldeanfragen auf über 1000 hat eine Fehlerrate größer 0% ergeben, wobei der Durchsatz an Anmeldungen pro Sekunde nur sehr geringfügig gestiegen ist. Das Ergebnis von 1000 Anfragen zeigt, dass alle Anfragen verarbeitet und keine verworfen wurden.

Der Test hat auch ergeben, dass durch die Verwendung von Single Sign-on mit Leistungseinbußen bei der Verarbeitung von rund 22% zu rechnen ist.

$$Verlangsamung = 1 - \frac{Durchsatz\ SSO}{Durchsatz\ Login} = 1 - \frac{25.9}{33.22} \approx 22\%$$

Abbildung 5.6 Berechnung der Leistungseinbußen mit Single Sign-on

Reaktionstest

Der Reaktionstest soll die Antwortzeit pro Anmeldung ermitteln. Dazu werden innerhalb von 60 Sekunden sequentiell 100 Anmeldeanfragen gesendet. Es existiert also nur jeweils ein Thread, der die Tests durchführt. Bei der Durchführung haben sich folgende Werte ergeben:

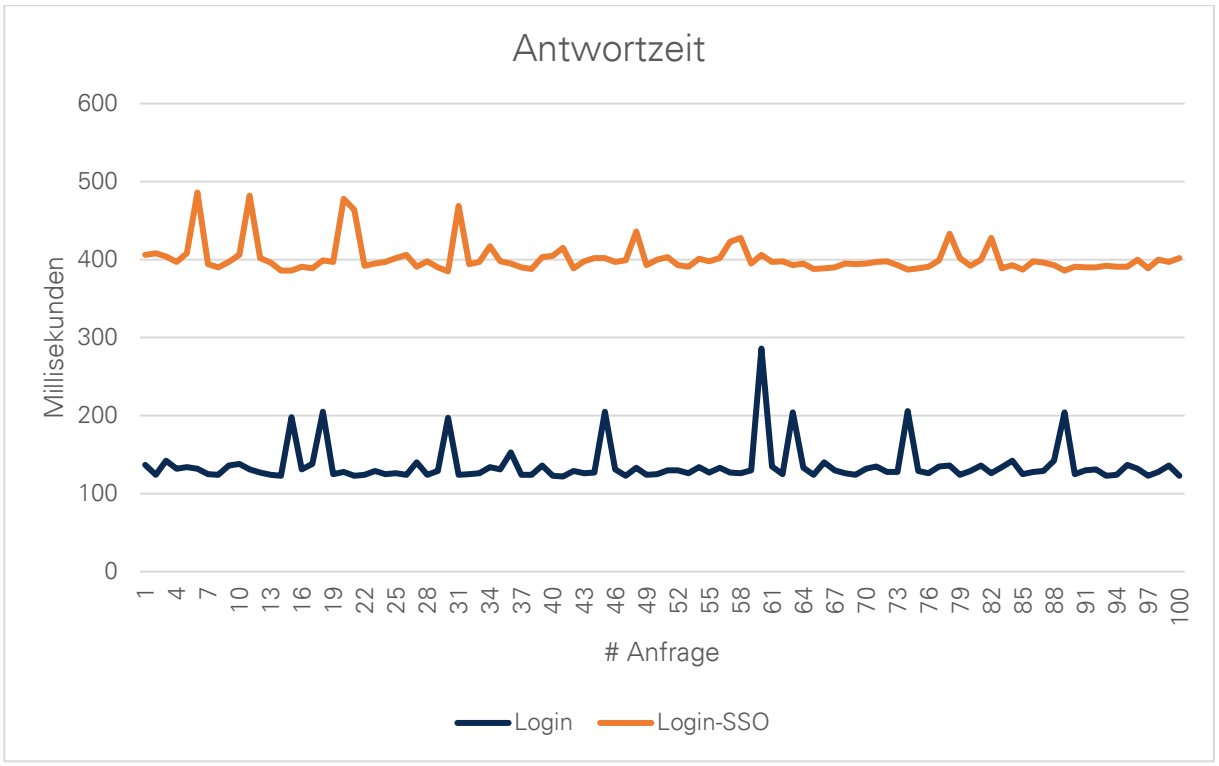


Abbildung 5.7 Ergebnisse des Reaktionstests für 100 Anmeldungen

Zu sehen ist, dass bei der Verwendung von Single Sign-on die Antwortzeit signifikant ansteigt. Die Antwortzeit für das bisherige Anmeldeverfahren bewegt sich im 130ms Bereich, die der Anmeldungen mit Single Sign-on bei 400ms. Weiterhin sind einige Abweichungen zu erkennen. Die Entstehung dieser muss aber nicht unbedingt mit der Anmeldung zusammenhängen, sondern kann auch durch Schwankungen in der Übertragungsgeschwindigkeit und -qualität hervorgerufen werden. Auch die allgemeine Serverlast trägt einen Teil dazu bei.

Aus diesen Gründen ist eine genauere Analyse der Ergebnisse notwendig. Es ist jedoch nicht ausreichend einfach nur einen Durchschnittswert zu ermitteln, denn größere Abweichungen können Einfluss auf das Ergebnis haben. Daher wird zusätzlich der Median und das empirische Quantil ermittelt. Der Median berechnet einen aussagekräftigeren Durchschnittswert, der relativ robust gegen Abweichungen ist. Das Quantil soll dann Aussagen darüber treffen, wie viel Zeit die Mehrheit der Anmeldungen benötigt. Erwartet wird, dass die Anmeldung mit Single Sign-on mindestens die dreifache Zeit benötigt. Denn aus Abbildung 5.4 geht hervor, dass die dreifache Anzahl an Anfragen und Antworten, als in Abbildung 5.3 benötigt werden. Die Kommunikation zwischen dem Back-End und dem Login-Service sollte relativ gering sein, da sich beide Parteien auf demselben Server befinden.

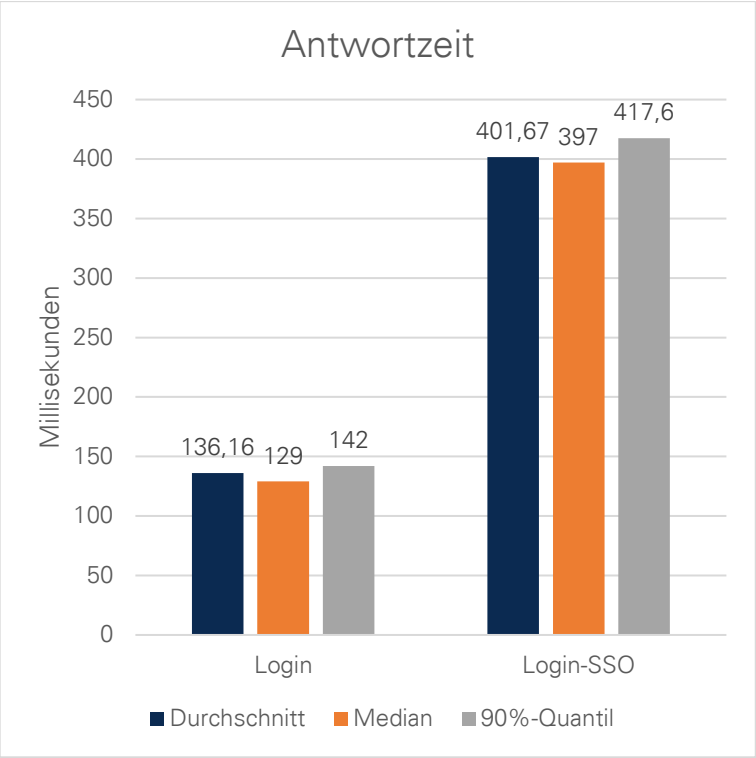


Abbildung 5.8

Die Tests haben ergeben, dass der Median der Antwortzeit für bisherige Anmeldungen bei 129ms liegt. Für den Single Sign-on Fall liegt die Antwortzeit bei 397ms. Somit benötigt eine Anmeldung mit Single Sign-on rund drei Mal so lang. Das deckt sich mit der Erwartung des Tests. Unterhalb des neunten Dezils liegen 90% der Messwerte (90%-Quantil). Die Testwerte zeigen, dass 90% der Antwortzeiten 142ms benötigen, wohingegen die Antwortzeiten mit Single Sign-on rund 417,6ms benötigen. Sie liegen über dem Median. Das heißt, dass 90% der Anmeldungen rund 142ms bzw. 417ms benötigen.

5.2.3 Nutzerbefragung

Zur Nutzerbefragung sollen Testprobanden eine feste Aufgabenliste abarbeiten. Danach sollen dazu Fragen mithilfe eines Fragebogens beantwortet werden. Für die Erstellung des Fragebogens werden verschiedene Evaluierungsmethoden eingesetzt. Verwendet werden geschlossene Fragen, bei denen die Antwortmöglichkeiten vorgegeben sind, Ja/Nein-Fragen und Ratingskalen. Die Ratingskalen verwenden eine Likert-Skala, um die Testprobanden nach ihrer persönlichen Einstellung gegenüber des Prototyps zu befragen. Für die Aufgabenstellung wird dem Testproband möglich kurz und einfach erklärt, worum es bei dieser Befragung geht. Denn das Thema Single Sign-on ist nicht jedem bekannt. Nach Erfassen der Aufgabenstellung wird der Testproband zum Verständnis dieser befragt. Folglich beginnen die eigentliche Abarbeitung der Aufgabe und die anschließende Beantwortung des Fragebogens.

Zur Verdeutlichung des Single Sign-on Systems werden für die Befragung zwei Webseiten über virtuelle Domains (https://domain1.de und https://domain2.de) eingerichtet. Beide Domains stellen beispielhaft unterschiedliche Webseiten dar, welche gemeinsam den Login-Service Prototyp nutzen. Die Verwendung dieser Beispielseiten ist notwendig. Würde der Tester nur die Implementierung in AMCS testen, würde ein Vergleich fehlen. Der Tester könnte keinen signifikanten Unterschied zu einem normalen Anmeldevorgang feststellen.

Die Testkonfiguration erfordert es, dass jeweils auf jeder Webseite auf „Anmelden" bzw. „Abmelden" geklickt werden muss. Auf diese Weise hat der Nutzer mehr Kontrolle über die beiden Webseiten und macht sich eventuell mehr Gedanken über die Abläufe. Nach einem Klick auf „Anmelden" wird der Nutzer zum Login-Service umgeleitet, um sich dort mit einem selbstgewähltem Pseudonym und Passwort zu authentifizieren. Danach wird er zurück zur ursprünglichen Webseite umgeleitet. Beim Besuch der zweiten Webseite und nach einem weiteren Klick auf „Anmelden" wird der Nutzer abermals zum Login-Service umgeleitet. Dieses Mal wird der Nutzer jedoch wiedererkannt und direkt zurückgesendet. Da keine Hinweismeldung oder ähnliches erscheint, bekommt der Nutzer von den beiden Umleitungen gar nichts mit. Zum Abmelden muss auf jeweils beiden Webseiten auf „Abmelden" geklickt werden. Dieses Verhalten soll durch die Nutzerbefragung bewertet werden, da jeder Mensch eine unterschiedliche Meinung über Kontrolle und Bequemlichkeit besitzt.

Im Folgenden werden die einzelnen Fragen beschrieben und die Ergebnisse der Befragungen durch 16 Testprobanden präsentiert und ausgewertet. Die ausgearbeitete Aufgabenstellung und der Fragebogen befinden sich im Anhang dieser Arbeit.

Fragen zum Verhalten

Bei der ersten Frage geht es darum, ob der Nutzer bei erstmaligem Besuch jeder Webseite auf „Anmelden" klicken muss oder ob der Nutzer direkt beim Aufruf automatisch angemeldet werden soll, sofern dies möglich ist. Mit dieser Frage soll herausgefunden werden, welches Vorgehen der Nutzer generell am angenehmsten empfindet, wenn er keine andere Wahl hat.

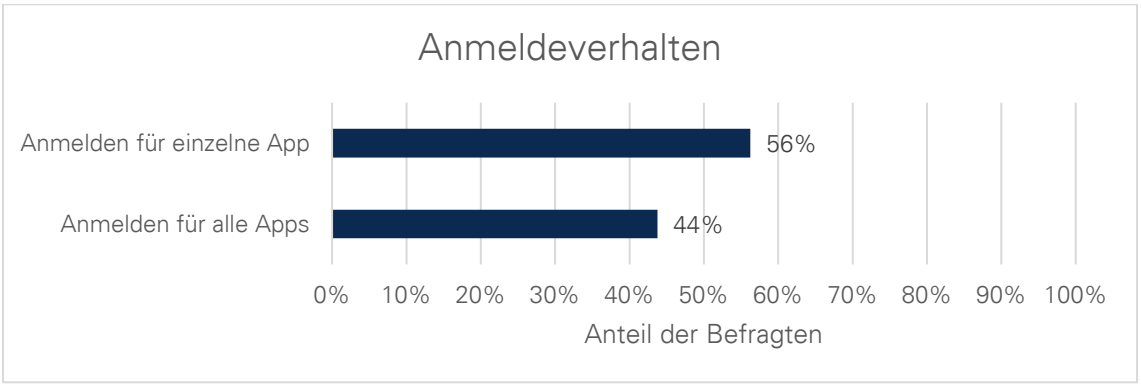


Abbildung 5.9 Ergebnis der Befragung zum Anmeldeverhalten

Die Frage wurde durch die Testprobanden relativ ausgeglichen beantwortet. Es zeigt sich eine Tendenz zum Anmelden für einzelne Anwendungen. Zum einen gaben die Testprobanden an, dass sie selbst die Kontrolle über den Vorgang behalten wollen, indem sie jeweils selbst den Anmeldevorgang auslösen. Zum anderen empfinden Sie es als bequem, wenn die Anmeldung komplett automatisiert erfolgt.

Die zweite Frage entspricht genau dem Gegenteil der ersten Frage. Der Testproband wird gefragt, ob er sich bei jeder Webseite einzeln abmelden möchte oder ob nur ein Klick auf „Abmelden" genügt, um gleichzeitig bei allen anderen in Verbindung stehenden Webseiten abgemeldet zu sein.

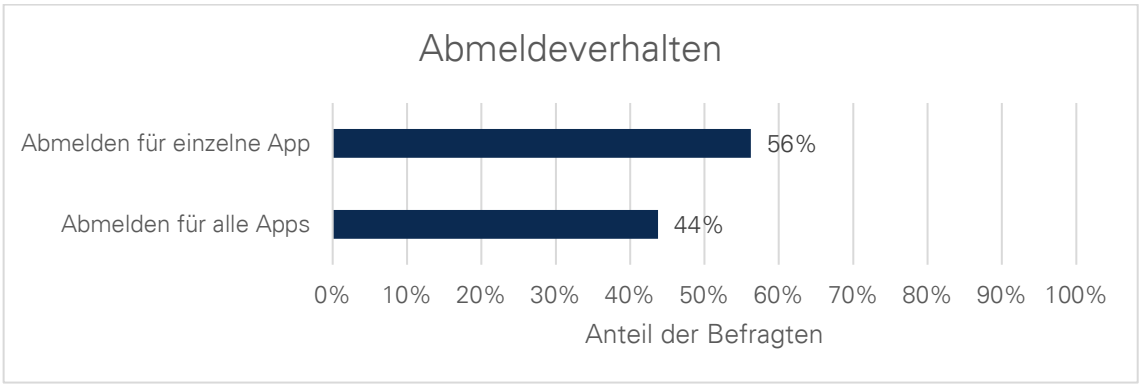


Abbildung 5.10 Ergebnis der Befragung zum Abmeldeverhalten

Das Ergebnis ist auch in diesem Fall relativ ausgeglichen. Die Angaben der Testprobanden sind die gleichen, wie bei der ersten Frage.

Fragen zu Nutzerhinweisen

Die nächsten beiden Fragen des Fragebogens sind mit „Ja" oder „Nein" zu beantworten. Sie dienen der Bewertung des Anmelde- und Abmeldeverhaltens, wenn der Nutzer vor die Wahl gestellt wird. Das heißt, die Testprobanden sollen angeben, ob sie gefragt werden möchten, automatisierte Vorgänge zu erlauben oder nicht. Auf diese Weise kann herausgefunden werden, ob sich die Nutzer mehr Kontrolle über automatisierte Vorgänge wünschen oder nicht.

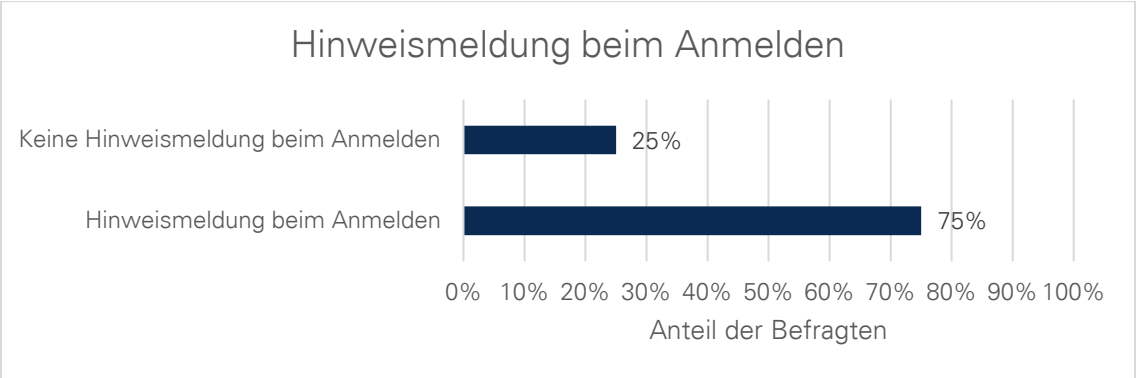


Abbildung 5.11 Ergebnis der Befragung zu Hinweismeldungen beim Anmelden

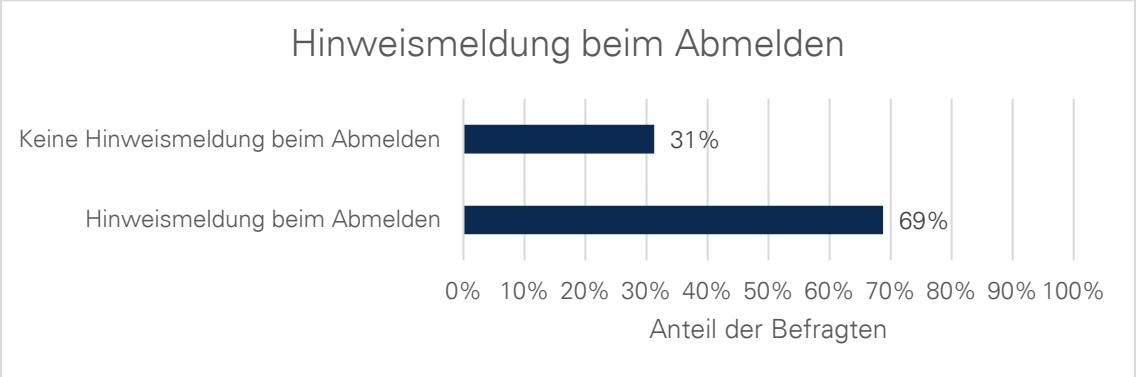


Abbildung 5.12 Ergebnis der Befragung zu Hinweismeldungen beim Abmelden

Die beiden Ergebnisse zeigen, dass rund dreiviertel der Befragten selbst entscheiden möchten, wie die Anmelde- und Abmeldevorgänge ablaufen. Das heißt, dem Nutzer ist die Kontrolle über das System wichtig. Der Prototyp könnte anhand dieser Informationen dahingehend verbessert werden, indem der Nutzer während der Authentifizierung beim Login-Service vor die Wahl gestellt wird. Mithilfe von HTML könnten Auswahlkästen bereitgestellt werden, die der Nutzer dann per Mausklick beliebig markieren kann.

Fragen zum Konzept

In diesem Teil werden dem Testprobanden acht Fragen gestellt, die mithilfe der Likert-Skala beantwortet werden sollen.

- Ich denke, der Anmeldevorgang ist sicher

Diese Frage verunsicherte zunächst die meisten Befragten. Sie gaben an, dass sie nicht genau beurteilen können, ob der Vorgang sicher sei. Zur Hilfestellung wurden ihnen die Frage gestellt, woran sie erkennen bzw. entscheiden, ob eine Webseite sicher sei oder nicht.

Nach kurzer Überlegung ergaben sich immer wieder die gleichen Antworten:

i. In der Adressleiste muss sich ein grünes Schloss befinden
ii. In der Adressleiste muss sich „https://“ befinden
iii. Die Webseite und die URL müssen seriös wirken oder bekannt sein

Dabei wurde noch einmal klargestellt, dass die beiden Beispielwebseiten zwei beliebige Webseiten im Alltag darstellen sollen. Anhand dieser Überlegungen sollte dann die eigentliche Frage beantwortet werden.

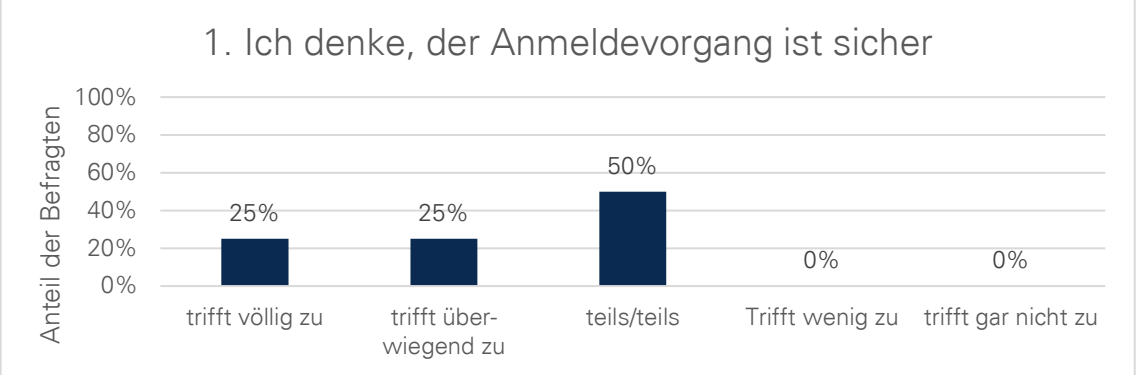


Abbildung 5.13 Ergebnis der Befragung zur Sicherheit

Das Ergebnis ist leider nicht ganz eindeutig. Die Hälfte der Befragten antwortete mit „teils/teils“. Die andere Hälfte verteilte sich gleichwertig zustimmend. Vermutlich bewertete auch ein Teil der Testprobanden die Frage nicht ganz objektiv. Jedoch ist trotzdem zu erkennen, dass die Testprobanden verunsichert sind und dem System skeptisch gegenüberstehen. Wahrscheinlich, da nicht kontrolliert werden kann, was im Hintergrund abläuft bzw. welche Daten übertragen werden. Das würde wiederrum bedeuten, dass der Nutzer sich mehr Transparenz und Kontrolle wünscht.

- Der Anmelde- bzw. Abmeldevorgand ist logisch/nachvollziehbar

Mit den nächsten beiden Fragen soll noch einmal überprüft werden, ob die Testprobanden verstanden haben, was bei der Anmeldung bzw. Abmeldung ablief.

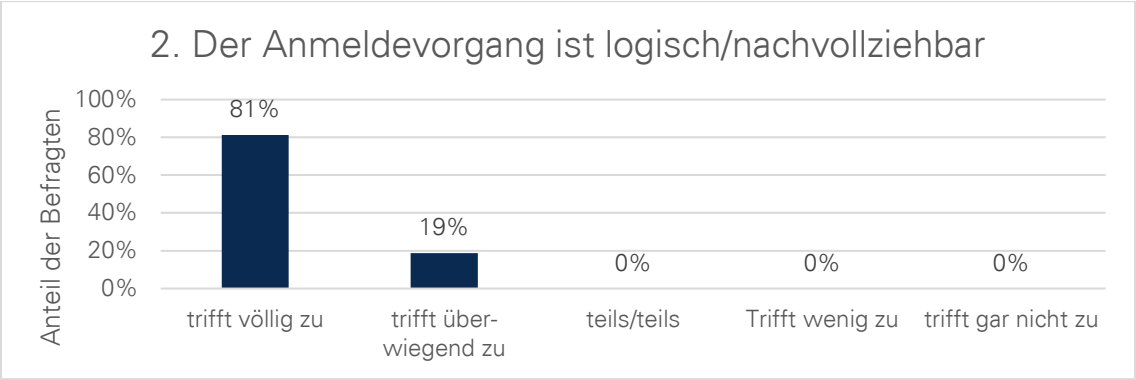


Abbildung 5.14 Ergebnis der Befragung zur Nachvollziehbarkeit der Anmeldung

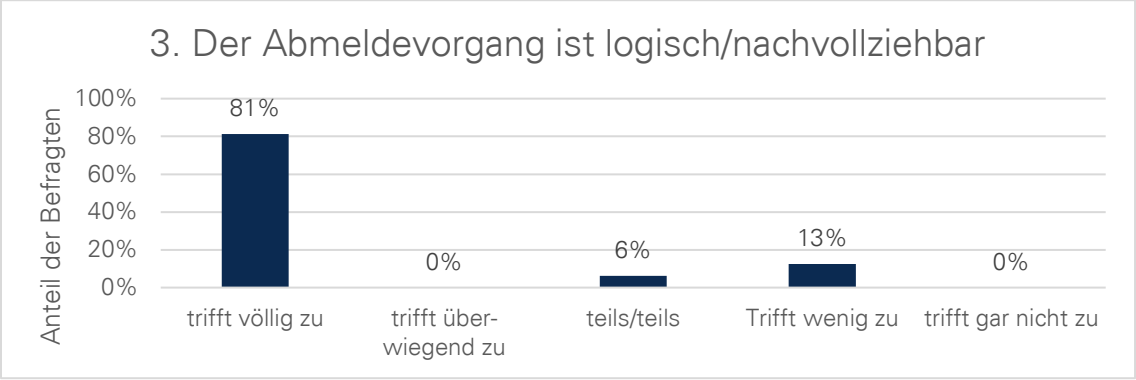


Abbildung 5.15 Ergebnis der Befragung zur Nachvollziehbarkeit der Abmeldung

Alle Befragten hatten verstanden, was nach dem Klick auf „Anmelden“ passierte und empfanden es für nachvollziehbar, dass beim Aufruf der zweiten Webseite erneut auf „Anmelden“ geklickt werden muss. Jedoch empfanden es wenige der Befragten für unlogisch, dass die Abmeldung für jede Webseite einzeln ausgelöst werden muss. In diesem Fall zeigt das Ergebnis für die Abmeldung, dass die wenigsten Testprobanden erwartet hatten, dass sie direkt bei jeder Webseite abgemeldet werden.

- Der Anmeldevorgand ist schnell
- Die automatische Anmeldung ist bequem

Diese beiden Fragen dienen der Bewertung der gefühlten Geschwindigkeit und der Bequemlichkeit bei der Nutzung des SSO Prototyps. Alle befragten stimmten zu 100% für „trifft völlig zu“. Die Ergebnisse sind wenig überraschend, denn sie sind ja praktisch vorhersehbar. Im

Unterschied zur klassischen Anmeldung fällt eine zweite Eingabe der Nutzerdaten weg. Aus diesem Grund ist die Anmeldung schnell und bequem.

- Single Sign-on ist mir wichtig

Diese Frage beschäftigt sich mit der Notwendigkeit von Single Sign-on Systemen. Wünschen sich die Probanden mehr Anmeldeoberflächen mit Single Sign-on oder ist es ihnen gleichgültig?

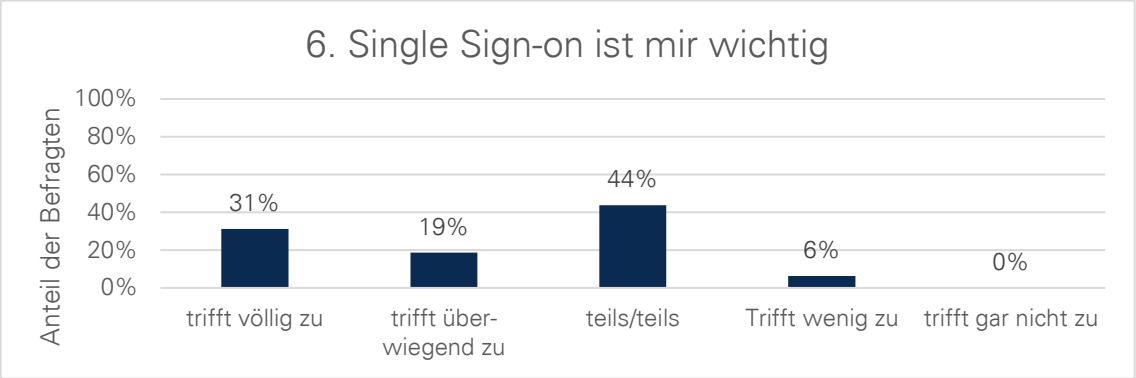


Abbildung 5.16 Ergebnis der Befragung zur Notwendigkeit von Single Sign-on

Die Ergebnisse lassen darauf schließen, dass sich die Befragten eher mehr solcher Systeme wünschen. Vermutlich, weil die Verwendung eines einzelnen Nutzerkontos bequemer ist als für jede Seite einzeln eines zu erstellen und sich verschiedene Passwörter merken zu müssen. Niemand bewertete Single Sign-on als komplett unnötig. Die meisten Stimmen sind geteilter Meinung. Daraus lässt sich ableiten, dass Single Sign-on gern genutzt wird, wenn es zu Verfügung stehen würde. Andernfalls werden weiterhin die klassischen Anmeldeverfahren genutzt.

- Die anonyme Anmeldung durch ein Pseudonym ist mir wichtig

AMCS stellt eine anonyme Nutzung bereit, indem die Anmeldung durch ein Pseudonym anstatt einer E-Mail-Adresse erfolgt. Mit dieser Frage soll ermittelt werden, wie wichtig dem Nutzer eine anonyme Nutzung ist.

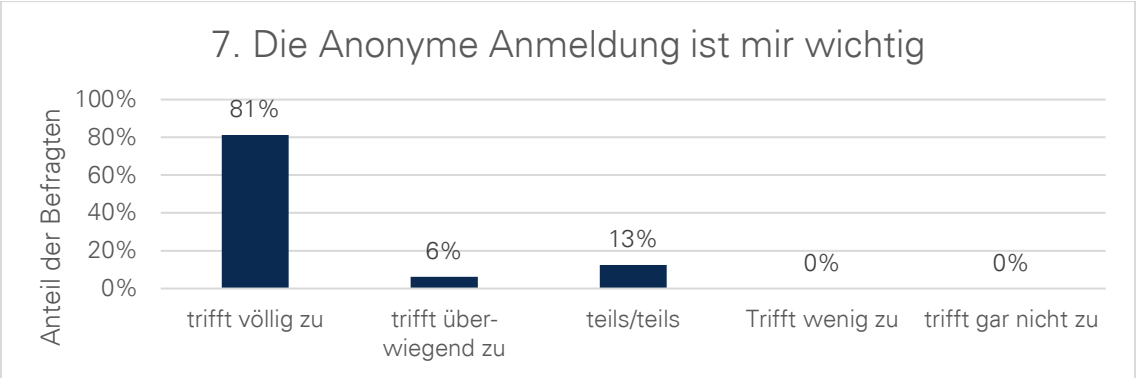


Abbildung 5.17 Ergebnis der Befragung zur anonymen Nutzung

Das Ergebnis ist relativ eindeutig. Fast allen Befragten ist die anonyme Nutzung durch Verwendung eines Pseudonyms sehr wichtig. Schon allein aus der Sicht des Datenschutzes sollten viel mehr Webanwendungen eine anonyme Nutzung bereitstellen und es dem Nutzer freistellen, ob er seine E-Mail-Adresse zusätzlich angibt oder nicht. E-Mail-Adressen sollten nur zur zusätzlichen Absicherung oder für Benachrichtigungen genutzt werden. Wenn ein Nutzer sein Passwort vergisst, kann er durch die Hinterlegung einer E-Mail-Adresse, das Passwort zurücksetzen lassen. Alternativlösungen sind aber auch durch Sicherheitsfragen denkbar.

- Ich wünsche mir zusätzlich zum Pseudonym weitere Anmeldemöglichkeiten wie per Google, Facebook und Co.

Durch die letzte Frage soll beurteilt werden, inwieweit Nutzer dazu bereit sind, Single Sign-on durch große bekannte Unternehmen zu nutzen.

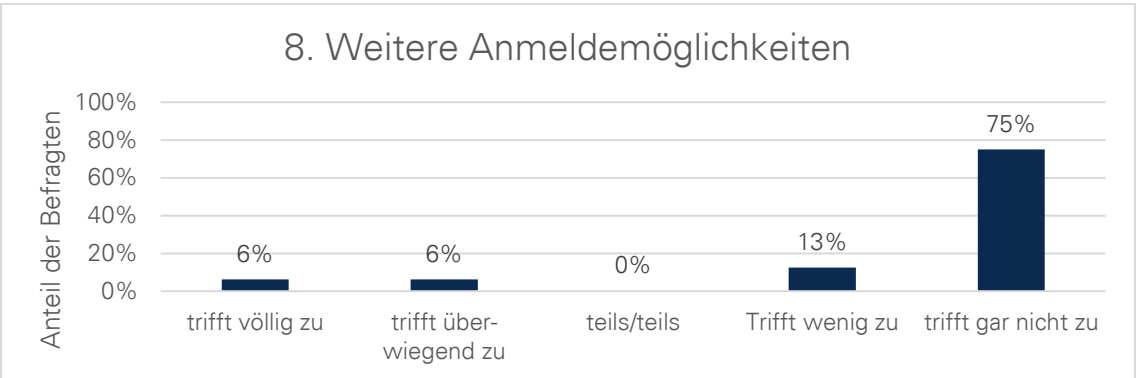


Abbildung 5.18 Ergebnis der Befragung zu weiteren Anmeldemöglichkeiten

Das Ergebnis zeigt hier deutlich, dass die Testprobanden diesen Unternehmen eher wenig vertrauen oder die weiteren Anmeldemöglichkeiten für nicht notwendig halten. Da die Anmeldung durch ein Pseudonym erfolgt, ist sie sehr schnell und es müssen keine weiteren Nutzerdaten angegeben werden.

5.3 ZUSAMMENFASSUNG DER ERGEBNISSE

Durch die Überprüfung der Funktionalität konnte gezeigt werden, dass der Prototyp die konzeptuellen Erwartungen erfüllt. Falscheingaben lieferten die entsprechenden Fehlerrückmeldungen. Mit gültigen Parametern konnten neue Nutzer angelegt und angemeldet werden.

Die Belastungs- und Reaktionstests lieferten interessante Werte. Eine Verlangsamung der Durchsatzrate von rund 22% wirkt erst einmal enorm. Doch liefert das Ergebnis mit rund 26 Anmeldungen pro Sekunde einen immer noch guten Wert. Das Ergebnis des Reaktionstests war erwartungsgemäß. Auch hier erscheint zunächst die dreifach benötigte Zeit für Single Sign-on als schlecht. Doch mit 129ms zu 397ms werden Anfragen immer noch in weniger als einer halben Sekunde verarbeitet. Der Geschwindigkeitsunterschied von 268ms sollte für den Nutzer nicht wirklich spürbar sein. Nichtsdestotrotz gilt es noch einmal daran zu erinnern, dass es bei jedem erneutem Aufruf des Front-Ends dennoch zu spürbaren Verzögerungen kommt.

Die Funktions- und Belastungstests an AMCS wurden bereits zuvor in einer anderen wissenschaftlichen Arbeit durchgeführt. Wie auch die Arbeit von Samuel Knobloch zeigt, hat sich der Einsatz von JMeter und Selenium bewährt. Die Ergebnisse dieser Arbeit weichen jedoch ab, da vor allem die Leistungsfähigkeit des Staging Servers zugenommen hat. Als Beispiel dient die Antwortzeit für das bisherige Anmeldeverfahren. In dieser Arbeit beträgt die Antwortzeit rund 129ms, wohingegen es 327ms in der Arbeit von Samuel Knobloch sind. (vgl. Knobloch, 2017)

Durch die Nutzerbefragung konnte gezeigt werden, dass die Akzeptanz eines Single Sign-on Systems mit anonymer Nutzung relativ groß ist. Die Nutzer vertrauen der anonymen Nutzung mehr als einer klassischen Anmeldung durch eine E-Mail-Adresse. Aber die Befragung hat auch gezeigt, dass der Nutzer mehr Kontrolle über das System möchte. Das kann zum Beispiel durch eine Einstellmöglichkeit während des Anmeldevorgangs geschehen.

Diese Punkte bestärken die Entscheidung für die Eigenentwicklung. Denn der Prototyp kann ohne Folgen angepasst werden, wohingegen CAS immer weiter von dessen Standard abweichen würde. Anhand der Bewertungen des Verhaltens des Prototyps, wenn der Nutzer keine Wahl hat, kann das Standardverhalten daraus abgeleitet werden. Das heißt, der Nutzer soll während der Anmeldung entscheiden können wie sich der Prototyp verhält. Nimmt er keine Einstellungsmaßnahmen vor, gilt das Standardverhalten. Laut der Befragung entspricht das Standardverhalten dem des aktuellen Prototyps. Das heißt, das An- und Abmelden muss also einzeln ausgelöst werden.

5.4 VERBESSERUNG DES PROTOTYPS

Wie sich schon durch die Nutzerbefragung ergeben hat, sollte der Prototyp dahingehend verbessert werden, dass der Nutzer über mehr Kontrolle verfügt. Die Wahl des Verhaltens des Prototyps während der Authentifizierung scheint eine geeignete Lösung zu sein. Doch es gibt noch weitere Punkte die bedacht werden müssen. Aus der Nutzerbefragung ging hervor, dass auch ein relativ großer Teil der Befragten sich eine automatische An- und Abmeldung wünscht. Der Prototyp unterstützt jedoch noch keine der Verhaltensweisen. Die automatische Anmeldung ist relativ simpel zu implementieren. Dazu muss beim Erstbesuch einer Webanwendung direkt der Nutzer zum Login-Service umgeleitet werden, also ohne vorheriges Klicken auf „Anmelden". Die automatische Abmeldung (Single Sign-off) ist hingegen deutlich schwieriger zu implementieren. Wie bereits im Abschnitt 3.5 untersucht wurde, gibt es verschiedene Lösungswege. Einer dieser Wege muss zuvor implementiert werden, wobei auch alle entsprechenden Webanwendungen angepasst werden müssen.

Anforderung	- /o/+
Universitätsinterner und zentralisierter Login-Server	+
Anonyme Nutzung	+
Eingabe einer PIN von einer Lehrveranstaltung	+
Möglichkeit des Imports bereits existierender Nutzerkonten	+
Nutzung durch andere Einrichtungen	+
Einsatz weiterer Dienste	+
Codeänderungen an AMCS	+
Logout	o
Passwort ändern	+
Nutzer sperren	-
Nutzerdaten-Overhead	+

Legende: + = unterstützt, o = teilweise unterstützt, - = nicht unterstützt

Tabelle 5.3 Neue Übersicht der unterstützten Anforderungen durch den Login-Service Prototyp

Nach der Durchführung der Evaluation wurde der Prototyp um eine weitere Anforderung gemäß der Tabelle 3.4 erweitert. Der Prototyp unterstützt nun zusätzlich die Änderung des Passwortes. Es wurden außerdem Testfälle für die Änderung des Passwortes erstellt. Somit ergibt

sich eine neue Zusammenfassung der Anforderungsunterstützung des Prototyps, die in Tabelle 5.3 dargestellt ist.

Damit fehlen nur noch 1,5 Anforderungspunkte. Der Prototyp ist aus diesem Grund für den Einsatz in AMCS soweit bereit. Anzumerken ist jedoch, dass vorhandene Nutzerkonten von AMCS noch in den Prototyp importiert werden müssten. Ein geeignetes Verfahren zum Importieren muss noch untersucht werden.

6 ZUSAMMENFASSUNG UND AUSBLICK

6.1 ZUSAMMENFASSUNG

Ziel dieser Arbeit war es, verschiedene Single Sign-on Lösungen zu untersuchen und miteinander zu vergleichen. Dabei sollte besonderes Augenmerk auf die anonyme Nutzung, wie im Anwendungsfall von AMCS, gelegt werden. Eine geeignete Lösung sollte dann prototypisch implementiert und evaluiert werden.

Für die Erfüllung dieser Ziele wurde zunächst der Begriff des Single Sign-on untersucht. Dazu gehörten die generelle Idee, der Aufbau und die Funktionsweise solcher Technologien. Des Weiteren wurden die Vor- und Nachteile von Single Sign-on näher betrachtet. Dabei stellte sich heraus, dass Single Sign-on eine Möglichkeit bietet, mehrere Anwendungen über ein und dasselbe Nutzerkonto zu nutzen. Die Vorteile von Single Sign-on überwiegen. Die Nutzer solcher Systeme müssen sich weniger Zugangsdaten merken. Die Sicherheitsfrage konzentriert sich auf weniger Punkte. Die Anmeldung erfolgt schneller.

Anschließend wurden verschiedene aktuelle Single Sign-on Lösungen untersucht. Dazu wurden der Aufbau und die Funktionsweise aufgezeigt. Danach wurde der IST-Stand von AMCS untersucht und das Ziel definiert. Zum Ziel gehören die Anforderungen, die der spätere Prototyp erfüllen soll. Zusätzlich fand die Recherche verwandter Arbeiten statt. Diese ergab, dass immer wieder die gleichen Technologien zum Einsatz kommen und dass größere Unternehmen scheinbar zum Trend beitragen. Andere Audience Response Systeme wie AMCS bieten auch eine anonyme Nutzung an. Diese arbeiten jedoch anders als bei AMCS.

Darauffolgend wurde eine Eigenentwicklung präsentiert, die zum besseren Verständnis der Funktionsweise von Single Sign-on entwickelt wurde. Im Anschluss daran konnten die Anforderungen an den Prototyp erweitert werden. Dadurch war ein sinnvoller Vergleich zwischen ausgewählten Technologien möglich. Bei der Wahl einer geeigneten Lösung stellte sich schnell heraus, dass die meisten Technologien für den Einsatz mit AMCS ungeeignet sind, da die anonyme Nutzung oder der zentralisierte Einsatz nicht gegeben waren. Die Wahl fiel letztendlich auf die Eigenentwicklung, welche im darauffolgenden Kapitel implementiert wurde. Der Prototyp wurde in PHP geschrieben und in AMCS integriert. In der Evaluation wurde die Funktionalität untersucht und Belastungstests ausgeführt. Anschließend wurde eine Nutzerbefragung mit 16 Testprobanden durchgeführt. Die Funktionalität konnte gezeigt werden, die Belastungstests entsprachen den Erwartungen und die Nutzerbefragung bestärkte die Wahl der Eigenentwicklung. Der Prototyp ist durch die zusätzliche Kommunikation rund drei Mal langsamer. Außerdem konnte das von den Nutzern erwartete Verhalten des Prototyps untersucht werden. Der Prototyp wurde daraufhin verbessert, indem die Möglichkeit der Passwortänderung implementiert wurde. Weitere Verbesserungen sind dennoch notwendig.

6.2 AUSBLICK

Der Login-Service Prototyp sollte zunächst um eine geeignete Single Sign-off Lösung erweitert werden. Im Abschnitt 3.5 wurde bereits über eine Lösung diskutiert. Die Erweiterung ist nötig, um die Möglichkeit der gleichzeitigen Abmeldung von mehreren Anwendungen zu ermöglichen. Der aktuelle Prototyp erlaubt nur die einzelne Abmeldung von Anwendungen.

Weiterhin könnte in einer fortführenden Arbeit untersucht werden, inwiefern der Prototyp ausgebaut werden kann. Zurzeit werden nur das Pseudonym und das Passwort des Nutzers gespeichert. Zusätzlich könnten weitere Datenfelder angeboten werden. Sinnvoll wäre die freiwillige Angabe einer E-Mail-Adresse, um dem Nutzer mehr Möglichkeiten der Interaktion zu bieten. Ein Nutzer könnte so Benachrichtigungen per E-Mail erhalten oder es könnte die Rücksetzung eines vergessenen Passwortes vereinfacht werden.

Das Konzept der anonymen Nutzung könnte auch weiter ausgebaut werden. Zum Beispiel untersuchte Anne Schumacher in ihrer Arbeit, Möglichkeiten zur stückweiten Deanonymisierung des Nutzers. Auf diese Weise könnte ein Nutzer, statt seinem Pseudonym, einen neuen Namen erhalten. Der Name könnte weiterhin an einen bestimmten Zweck gebunden werden. (vgl. Schumacher, 2017)

Eine Umsetzungsidee für ein Deanonymisierungskonzept wäre die Erweiterung des Login-Services. Der Nutzer bekommt zum Beispiel mehrere Anonymitätsstufen zugewiesen. Jede Stufe verfügt dann über eine unabhängige Bezeichnung. Den Stufen können danach bestimmte Bereiche in den Anwendungen zugeordnet werden. Beispiele dafür wären bestimmte Tiernamen für unterschiedliche Threads eines Forums.

A ANHANG

A.1 EVALUATION – AUFGABENSTELLUNG

Der folgende Fragebogen dient der Evaluation des Prototyps für den Login-Service. Der Login-Service soll, vereinfacht ausgedrückt, einzelne Anmeldungen mit verschiedenen Benutzerkonten für die jeweiligen Webseiten durch eine Anmeldung mit einem einzelnen Benutzerkonto ersetzen. Dieses Verhalten wird auch Single Sign-on (SSO) genannt. Dabei ist die Anmeldung zusätzlich anonym, indem ein beliebiges Pseudonym, welches nicht bereits verwendet wird, und ein Passwort vom Nutzer gewählt werden. Das Pseudonym ersetzt die Verwendung einer E-Mail-Adresse oder ähnlicher personenbezogener Daten.

Für die erfolgreiche Beantwortung des Fragebogens ist ein vorgegebener Testablauf notwendig, der in einer entsprechend konfigurierten Testumgebung stattfindet. Er dient in erster Linie der einfacheren Beantwortung der Fragen. Die Beantwortung und Auswertung der Fragen geschieht anonym. Es werden keinerlei personenbezogene Daten erhoben.

Bitte folgen Sie den Anweisungen und beantworten Sie die Fragen auf der nächsten Seite:

1. Aufrufen der ersten Webseite https://domain1.de
2. Anmelden durch Klicken auf „Anmelden“
3. Eingabe neuer oder vorhandener Zugangsdaten
4. Aufrufen der zweiten Webseite https://domain2.de
5. Anmelden durch Klicken auf „Anmelden“
6. Durch die automatische Anmeldung ist keine erneute Eingabe von Zugangsdaten notwendig
7. Abmelden von der zweiten Webseite durch Klicken auf „Abmelden“
8. Aufrufen der ersten Webseite https://domain1.de
9. Abmelden von der ersten Webseite durch Klicken auf „Abmelden“

A.2 EVALUATION – FRAGEBOGEN

Vielen Dank, dass Sie sich die Zeit nehmen, die Fragen zu beantworten. Ihre Antworten dienen ausschließlich der Bewertung und Verbesserung des Prototyps für den Login-Service.

Fragen zum Verhalten beim Anmelden

Wenn sich der Nutzer bei einer App anmeldet, dann…

☐ …soll er auch bei allen anderen Apps automatisch angemeldet werden

☐ …soll er nur für diese App angemeldet werden

Fragen zum Verhalten beim Abmelden

Wenn sich der Nutzer von einer App abmeldet, dann…

☐ …soll er auch bei allen anderen Apps automatisch abgemeldet werden

☐ …soll er nur für diese App abgemeldet werden

Fragen zu Nutzerhinweisen

Wenn sich der Nutzer von einer App abmeldet, dann soll er gefragt werden, ob er auch bei allen anderen abgemeldet werden möchte:	☐ ja	☐ nein
Der Nutzer soll bei anderen Apps gefragt werden, ob er ohne Eingabe der Zugangsdaten angemeldet werden möchte (soweit möglich):	☐ ja	☐ nein

Fragen zum Konzept

	trifft völlig zu	trifft überwiegend zu	teils/teils	Trifft wenig zu	trifft gar nicht zu
1. Ich denke, der Anmeldevorgang ist sicher:	☐	☐	☐	☐	☐
2. Der Anmeldevorgang ist logisch/nachvollziehbar:	☐	☐	☐	☐	☐
3. Der Abmeldevorgang ist logisch/nachvollziehbar:	☐	☐	☐	☐	☐
4. Der Anmeldevorgang ist schnell:	☐	☐	☐	☐	☐
5. Die automatische Anmeldung für andere Apps ist bequem:	☐	☐	☐	☐	☐
6. Single Sign-on ist mir wichtig:	☐	☐	☐	☐	☐
7. Die Anonyme Anmeldung durch ein Pseudonym ist mir wichtig:	☐	☐	☐	☐	☐
8. Ich wünsche mir zusätzlich zum Pseudonym weitere Anmeldemöglichkeiten wie per Google, Facebook und Co.:	☐	☐	☐	☐	☐

Anmerkungen/Wünsche

B LITERATURVERZEICHNIS

Hardt, Dick. 2012. RFC 6749 - The OAuth 2.0 Authorization Framework. [Online] Oktober 2012. [Zitat vom: 13. Mai 2018.] https://tools.ietf.org/html/rfc6749.

Hardt, Dick, Bufu, Johnny und Hoyt, Josh. 2007. Final: OpenID Attribute Exchange 1.0 - Final. [Online] 5. Dezember 2007. [Zitat vom: 17. Mai 2018.] http://openid.net/specs/openid-attribute-exchange-1_0.html.

AMCS - TU Dresden. 2018. AMCS - Auditorium Mobile Classroom Service. *Auditorium Mobile Classroom Service.* [Online] 12. April 2018. [Zitat vom: 27. Juni 2018.] https://amcs.website/manual.

Apereo Foundation - CAS . 2018. apereo/cas: Apereo CAS - Enterprise Single Sign On for all earthlings and beyond. *GitHub.com.* [Online] Apereo, 14. März 2018. [Zitat vom: 30. April 2018.] https://github.com/apereo/cas.

Apereo Foundation - CAS - Password Management. 2017. CAS - Password Management. [Online] 22. Februar 2017. [Zitat vom: 11. Mai 2018.] https://apereo.github.io/cas/5.1.x/installation/Password-Management.html.

Battaglia, Scott. 2006. JA-SIG Central Authentication Service. *jasig.org.* [Online] 23. August 2006. [Zitat vom: 30. April 2018.] http://developer.jasig.org/test/cas.html#history.

Browinski, Gregg. 2016. SAML Single Logout - What You Need to Know. [Online] 20. Juni 2016. [Zitat vom: 16. Mai 2018.] https://www.portalguard.com/blog/2016/06/20/saml-single-logout-need-to-know/.

Cantor, Scott. 2012. SamlV20Errata - SAML Wiki. *SAML Wiki.* [Online] Oasis, 23. Mai 2012. [Zitat vom: 30. April 2018.] https://wiki.oasis-open.org/security/SamlV20Errata.

carolgeyer. 2007. Advantages of SAML | SAML XML.org. *saml.xml.org.* [Online] OASIS, 12. Dezember 2007. [Zitat vom: 30. April 2018.] http://saml.xml.org/advantages-saml.

Dewanto, Lofi. 2009. Identity Management: Authentifizierungsdienste mit OpenID | heise Developer. *heise.de.* [Online] 21. April 2009. [Zitat vom: 21. Mai 2018.] https://www.heise.de/developer/artikel/Identity-Management-Authentifizierungsdienste-mit-OpenID-227202.html.

drupal.org - Dries Buytaert. 2018. Usage statistics for SAML Authentication. *drupal.org.* [Online] 03. April 2018. [Zitat vom: 30. April 2018.] https://www.drupal.org/project/usage/samlauth.

Ferrara, Anthony. 2001. Stackoverflow. *security - PHP Session Fixation / Hijacking - Stackoverflow.* [Online] 22. Februar 2001. [Zitat vom: 25. April 2018.] https://stackoverflow.com/a/5081453/5984852.

Fielding, Roy, et al. 1999. RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1. *The Internet Engineering Task Force.* [Online] Juni 1999. [Zitat vom: 5. Juni 2014.] http://tools.ietf.org/html/rfc2616.

Hammer, Eran. 2007. Explaining OAuth - hueniverse. [Online] 5. September 2007. [Zitat vom: 18. April 2018.] https://hueniverse.com/explaining-oauth-3735e3de27a8.

HTTP Methods: GET vs. POST - w3schools. o.D.. *HTTP Methods: GET vs. POST.* [Online] w3schools, o.D. [Zitat vom: 18. April 2018.] https://www.w3schools.com/tags/ref_httpmethods.asp.

Hunt, Troy. o.D.. Have I Been Pwned. [Online] o.D. [Zitat vom: 20. April 2018.] https://haveibeenpwned.com/PwnedWebsites.

Jones, Mike. 2017. OpenID Connect Logout Implementer's Drafts Approved | OpenID. [Online] 28. März 2017. [Zitat vom: 8. Mai 2018.] http://openid.net/2017/03/28/openid-connect-logout-implementers-drafts-approved/.

Knobloch, Samuel. 2017. *Entwicklung automatisierter Tests zur Überwachung der Integrität und Performance von Frontend und Backend für die Plattform AMCS.* Dresden : Technische Universität Dresden, 2017.

Krafzig, Dr. Dirk und Yunus , Mamoon . 2015. Anwendungssicherheit SAML – Eine Einführung - JAXenter. *jaxenter.de.* [Online] 16. April 2015. [Zitat vom: 23. Mai 2018.] https://jaxenter.de/anwendungssicherheit-saml-eine-einfuehrung-18066.

Lv, Sekito . 2018. HTTP cookies - HTTP | MDN. *developer.mozilla.org.* [Online] 6. Juni 2018. [Zitat vom: 22. Juni 2018.] https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies.

OASIS Open - Core. 2005. Assertions and Protocols for the OASIS. *oasis-open.org.* [Online] 15. März 2005. [Zitat vom: 25. Mai 2018.] http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf.

OASIS Open - Profiles. 2005. Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0. *oasis-open.org.* [Online] 15. März 2005. [Zitat vom: 16. Mai 2018.] https://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf.

OpenID Foundation. 2007. Final: OpenID Authentication 2.0 - Final. [Online] 5. Dezember 2007. [Zitat vom: 17. Mai 2018.] http://openid.net/specs/openid-authentication-2_0.html.

Peyrott, Sebastián . 2018. What is and how does Single Sign On Authentication work? *auth0.com.* [Online] 20. Juni 2018. [Zitat vom: 21. Juni 2018.] https://auth0.com/blog/what-is-and-how-does-single-sign-on-work/.

Rockford , Lhotka. 2004. Authentifizierung und Autorisierung. *msdn.microsoft.com.* [Online] 29. Juni 2004. [Zitat vom: 24. Juni 2018.] https://msdn.microsoft.com/de-de/library/bb978972.aspx.

Sakimura, Nat , et al. 2014. Final: OpenID Connect Core 1.0 incorporating errata set 1. *openid.net.* [Online] 8. November 2014. [Zitat vom: 21. Mai 2018.] http://openid.net/specs/openid-connect-core-1_0.html.

Sakimura, Nat, Bradley, John und Jones, Michael B. 2014. Final: OpenID Connect Dynamic Client Registration 1.0 incorporating errata set 1. [Online] 8. November 2014. [Zitat vom: 8. Mai 2018.] http://openid.net/specs/openid-connect-registration-1_0.html.

Sakimura, Nat, et al. 2013. Draft: OpenID Connect Basic Client Implementer's Guide 1.0 - draft 32 - Scope Values. *openid.net.* [Online] 20. Dezember 2013. [Zitat vom: 07. Mai 2018.] http://openid.net/specs/openid-connect-basic-1_0-32.html#Scopes.

Schumacher, Anne. 2017. *Mehrschritt-Deanonymisierungskonzept für initial anonyme Onlinesysteme.* Dresden : Technische Universität Dresden, 2017.

Schwan, Ben . 2001. OpenID Foundation unternimmt neuen Anlauf. [Online] Heise Medien GmbH & Co. KG, 8. September 2001. [Zitat vom: 27. Juni 2018.] https://www.heise.de/newsticker/meldung/OpenID-Foundation-unternimmt-neuen-Anlauf-1338919.html.

Stepka, Justen. 2007. Using OpenID. *TheServerSide.com.* [Online] 1. Mai 2007. [Zitat vom: 20. Mai 2018.] https://www.theserverside.com/news/1364125/Using-OpenID.

Suoranta, Sanna , et al. 2013. Logout in Single Sign-on Systems. [Online] 2013. https://link.springer.com/content/pdf/10.1007/978-3-642-37282-7_14.pdf. ISBN:978-3-642-37282-7.

Thibeau, Don. 2014. The OpenID Foundation Launches the OpenID Connect Standard | OpenID. *openid.net.* [Online] 26. Februar 2014. [Zitat vom: 21. Mai 2018.] http://openid.net/2014/02/26/the-openid-foundation-launches-the-openid-connect-standard/.

XAMPP – Wikipedia. 2018. XAMPP – Wikipedia. *wikipedia.org.* [Online] 12. Juni 2018. [Zitat vom: 18. Juni 2018.] https://de.wikipedia.org/wiki/XAMPP.