Entwicklung eines Single Sign-On-Systems mit verschiedenen tokenbasierten Authentifizierungsmethoden

Bachelorarbeit Informatik

der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen

Wilhelm-Schickard-Institut für Informatik

Gutachter: Prof. Dr. Thomas Walter
Wilhelm-Schickard-Institut für Informatik, Universität Tübingen

Betreuer: Dr. Heinrich Abele
Zentrum für Datenverarbeitung, Universität Tübingen

vorgelegt von
Eric Speidel
Abgabedatum: 30.11.2017

Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich die vorgelegte Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Außerdem erkläre ich, dass die eingereichte Arbeit weder vollständig noch in wesentlichen Teilen Gegenstand eines anderen Prüfungsverfahrens gewesen ist.

Unterschrift	Ort, Datum

Danksagung

Mein Dank gilt vor Allem meinem Betreuer Heinrich Abele, welcher sich trotz eines vollen Terminkalenders immer wieder Zeit für mich nahm und zu jeder Frage und jedem Problem gute Ansätze und Ideen beisteuerte.
Außerdem bedanke ich mich bei Robert Finze für die Unterstützung bei der Einrichtung des Arbeitsplatzes und eine umfangreiche Führung durch das Zentrum für Datenverarbeitung. Ebenfalls danke ich Jörg Heitzenröther für die schnelle Bearbeitung meines Zertifikatantrags sowie Ramon Pfeiffer für seine Unterstützung und Geduld bei der Implementierung von SAML2.

Außerdem möchte ich mich bei meinem Bürokollegen und Vorgänger Tim Beckmann für viele nützliche Tipps rund um die Themen Bachelorarbeit und ZDV sowie gute Unterhaltung bedanken.
Ich bedanke mich herzlich bei Ramona Schäfer und Lucca Hellriegel für das Korrekturlesen dieser Bachelorarbeit, ohne euch wären mir viele kleine Fehler nicht aufgefallen.

Inhaltsverzeichnis

Abbildungsverzeichnis **5**

Zusammenfassung **6**

1 Einleitung **7**
1.1 Motivation . . . 7
1.2 Einordnung in bestehende Arbeiten und Dienste . . . 8

2 Hintergrund **9**
2.1 Authentifizierung . . . 9
2.1.1 OAuth2 und OpenID Connect . . . 11
2.1.2 SAML2 . . . 14
2.1.3 SAML2 und OAuth2/OpenID Connect im Vergleich . . . 18
2.2 Autorisierung . . . 19
2.2.1 Claims . . . 19
2.3 ASP.NET Core . . . 19
2.4 ASP.NET Core Identity . . . 20
2.5 Sustainsys.Saml2 . . . 20
2.6 Microsoft Internet Information Services . . . 20
2.7 Microsoft SQL Server . . . 20

3 Lösungskonzept **21**
3.1 Integratives Framework für Authentifizierung und Autorisierung . . . 21
3.2 Webserver und Datenbank . . . 21
3.3 Lokale Authentifizierung . . . 21
3.4 Externe Authentifizierung via OAuth2 + OpenID Connect . . . 21
3.5 Externe Authentifizierung via SAML2/Shibboleth . . . 22
3.6 Autorisierung . . . 22
3.7 Administration . . . 22

4 Implementierung **23**
4.1 Model-View-Controller Architektur . . . 23
4.2 Claims und Policies . . . 24
4.3 Authentifizierung . . . 24
4.3.1 Lokale Authentifizierung . . . 25
4.3.2 Lokale Authentifizierung mit Zwei-Faktor Authentifizierung . . . 25
4.3.3 OAuth2/OpenID Connect-basierte Authentifizierung . . . 26
4.3.4 SAML2-basierte Authentifizierung . . . 26
4.3.5 Automatische Auswahl der Authentifizierungsmethode . . . 27
4.3.6 Mehrere Authentifizierungsmethoden pro Nutzer . . . 28
4.4 Administration . . . 28
4.4.1 UserList . . . 28
4.4.2 Create . . . 28
4.4.3 Edit . . . 28
4.4.4 Delete . . . 28
4.5 Controller . . . 29
4.5.1 AccountController . . . 29
4.5.2 AdminController . . . 30

4.5.3 ManageController . . . 30
4.6 Sicherheit . . . 31
4.7 Layout . . . 31

5 Ergebnisse **32**
5.1 Aufbau . . . 32
5.2 Testbetrieb . . . 32
5.3 Präsentation . . . 32
5.3.1 Frontend . . . 32
5.3.2 Backend . . . 33
5.4 Kommunikation mit SAML2- und OpenID Connect-IDP . . . 33
5.5 Performance unter Last . . . 34

6 Diskussion **36**
6.1 ASP.NET Core . . . 36
6.2 OpenID Connect vs. SAML2 . . . 37
6.3 Ausblick . . . 37
6.3.1 Nutzertest . . . 37
6.3.2 Automated Deployment . . . 37

7 Anhang **38**

Literaturverzeichnis **42**

Abbildungsverzeichnis

1 Drei gängige Authentifizierungsmethoden schematisch dargestellt. 11
2 Typischer Authentifizierungsfluss mit OpenID Connect. 13
3 Vertrauenskonzept einer SAML2-Infrastruktur 14
4 Vereinfachtes Schema eines SAML2 Authentifizierungsflusses. 17
5 Schematischer MVC-Aufbau 23
6 Vom Claim zum Recht 24
7 Vertrauenskette der Plattform (links) und des Test-IDP (rechts) 27
8 Automatische Auswahl zur Authentifizierung 27
9 Frontend der Plattform für einen Loginvorgang 32
10 Backend der Plattform für die Anzeige aller Benutzer 33
11 AuthnRequest der Plattform an den Test-IDP der Universität Tübingen . . 33
12 CPU-Auslastung bei kontinuierlichen Anfragen für eine geschützte Ressource von 100 parallelen Threads 34
13 CPU-Auslastung bei kontinuierlichen Anfragen für eine geschützte Ressource von 1000 parallelen Threads 35
14 CPU-Auslastung bei kontinuierlichen Loginversuchen von 1000 parallelen Threads 35

Zusammenfassung

In dieser Arbeit wurde ein Single Sign-On (SSO)-System konzipiert, welches unterschiedliche Authentifizierungsmethoden unterstützt und darüber hinaus Autorisierungsfunktionen bereitstellt. Die verwendeten marktüblichen Authentifizierungsverfahren wurden unter den Aspekten Architektur des Verfahrens und verfahrensspezifische Sicherheitskonzepte vergleichend betrachtet. Das realisierte, auf .NET Core aufbauende, SSO-System integriert die verschiedenen Authentifizierungsverfahren und unterstützt heute relevante Webserverplattformen auf den Betriebssystemen Windows, Linux und Mac OS. Die entwickelte Softwarelösung kann nicht nur für neue .NET Core Projekte sondern auch in bestehenden Projekte verwendet werden. Unterstützt werden lokale Authentifizierung, Zwei-Faktor-Authentifizierung sowie die Standards OpenID Connect und SAML2 zum Austausch von Authentifizierungsinformation.
Mit einer Administrationsoberfläche können Benutzerdaten und -rechte verwaltet werden. Die Plattform, welche im Rahmen dieser Arbeit erstellt wurde, implementiert ein modernes Sicherheitskonzept und deckt eine Vielzahl von Anwendungsfällen ab.

1 Einleitung

1.1 Motivation

Mit der zunehmenden Digitalisierung im privaten, öffentlich-institutionellen und industriellen[1] Bereich ist es für Onlinedienste sowie deren Nutzer wichtig und zum Teil unabdingbar geworden, die Identität, Authentizität und Vertrauenswürdigkeit aller beteiligten Parteien sicherzustellen. Da dieses Problem eine stetig wachsende Bevölkerungsgruppe mit unterschiedlicher IT-Affinität betrifft, sind die Anforderungen an Prozesse zur Authentifizierung, beziehungsweise Autorisierung hoch: Der Vorgang muss einfach und transparent genug sein, um Anklang bei einer großen Zahl von Nutzern zu finden. Gleichzeitig soll er sicher vor Informationsdiebstahl, unbefugtem Zutritt und Manipulation sein. In Anbetracht wachsender Cyberkriminalität[2] besteht großes Interesse an einem sicheren Austausch sensibler Daten.

Viele Internetdienste implementieren hierfür einen Single Sign-On (SSO), der den Zugriff auf verschiedene Ressourcen und Dienste mit nur einer Anmeldung ermöglicht. So erlauben bereits viele Webseiten neben der traditionellen lokalen Anmeldung in Verbindung mit einer Registrierung mit Nutzername und Passwort auch eine Anmeldung via Drittanbieter wie Facebook [3] oder Google [4], sowie eine Anmeldung mittels eines zentralen Zugangs, der für verschiedene institutionelle Angebote gültig ist. Dieses System verfolgen unter anderem diverse Bildungseinrichtungen, darunter auch die Universität Tübingen. [5] Dies erspart dem Nutzer eine weitere Registrierung und deckt bei hohen Nutzerzahlen von etwa 2 Milliarden alleine bei Facebook [6] bereits sehr viele Benutzer ab.

Diese Arbeit soll mehrere dieser bestehenden Konzepte vergleichen und in einer Plattform vereinen, welche die entsprechenden Identitätsinformationen vieler Nutzertypen verarbeitet und Authentifizierung, Autorisierung sowie Benutzerverwaltung mit geringem Konfigurationsaufwand anbietet und von anderen Anwendungen für diese Aufgaben verwendet werden kann. Ferner soll sie plattformunabhängig, also auf allen wichtigen Betriebssystemen anwendbar sein.

1.2 Einordnung in bestehende Arbeiten und Dienste

Ein großer Anteil von Webseiten und Diensten im Consumer-Bereich bietet bereits tokenbasierte Logins via Google oder Facebook an. Dies kann, abhängig von bestehender Infrastruktur und bestehender Datenverwaltung mit erheblichem Entwicklungsaufwand einhergehen. Services sind häufig auf eine Vielzahl von Plugins angewiesen, da externe Loginanbieter oft unterschiedliche Standards verwenden, beziehungsweise denselben Standard mit anderer Konfiguration implementieren.
Zur Vereinfachung dieser Prozesse existieren Services wie OAuth.io [7] oder oneall [8], die sich größtenteils auf OAuth2-basierte Dienste beschränken. Das deckt die Mehrheit der sozialen Netzwerke ab, lässt aber weitere Optionen zum SSO wie etwa SAML2 außen vor. Für viele Programmiersprachen sind diese Services außerdem nicht verfügbar.
In bestimmten Fällen sind Logins über soziale Netzwerke auch ungeeignet, da OAuth2 im Grunde zur Autorisierung entwickelt wurde, und nur bedingt zur Authentifikation mit vielen sensiblen Daten geeignet ist. So lässt etwa Amazon [9] als Dienst, der viele sensible Daten benötigt, keine externe Authentifikation zu, bietet aber einen eigenen 'Login with Amazon' [10] Dienst an.

Diese Arbeit soll einige dieser Limitierungen überwinden und implementiert die beiden tokenbasierten Authentifizierungsverfahren OpenID Connect und SAML2 in ASP.NET Core in Konjunktion mit einer lokalen Benutzerdatenbank, die auch eine klassische Registrierung per E-Mail und Passwort zulässt. Ein Vorteil gegenüber existierenden Diensten ist hierbei die Berücksichtigung der .NET Plattform, für die häufig keine Implementation verfügbar ist, da andere Programmiersprachen und Plattformen wie PHP oder Swift bevorzugt berücksichtigt werden. Außerdem lässt sich durch die zusätzliche SAML2-Unterstützung ein größeres Spektrum an Anwendungsfällen abdecken. Die Plattform dieser Arbeit verbindet diese drei Authentifizierungsmethoden zusätzlich mit Autorisierung und Benutzerverwaltung und geht bei Verwendung innerhalb einer Applikation oder Webseite deshalb mit einer Ersparnis an Arbeitsaufwand einher.

2 Hintergrund

2.1 Authentifizierung

Als Authentifizierung wird im Allgemeinen die Überprüfung von Daten oder Aussagen einer Entität, etwa eines Servers oder einer Person, bezeichnet. Im Zuge einer erfolgreichen Authentifizierung werden die Behauptungen der Entität als wahrheitsgetreu anerkannt, bei einer fehlgeschlagenen Authentifizierung können diese nicht verifiziert werden.
Innerhalb eines Single Sign-On Systems wird die Authentifizierung dazu verwendet, um einen Benutzer, beziehungsweise dessen Inhaberschaft eines Kontos identifizieren zu können. Mit einem Benutzer verbundene Rechte und Gruppenzugehörigkeiten sind nicht Teil einer Authentifizierung. Damit ist die Authentifizierung vom Vorgang der Autorisierung 2.2 abgegrenzt. Es gibt verschiedene Möglichkeiten, eine Authentifizierung durchzuführen.

Eine Methode ist die Registrierung einer Benutzer-ID (Benutzername oder E-Mailadresse) in Kombination mit der Vergabe eines Nutzergeheimnisses, etwa einer Passphrase, eines oder mehrerer Zugangscodes. Die Kenntnis von Benutzer-ID und Geheimnis wird als hinreichender Beweis für die Inhaberschaft des Benutzerkontos angesehen. Diese Lösung ist weit verbreitet und daher auch ein häufiges Ziel von cyberkriminellen Angriffen. Dabei machen Angreifer unter anderem von folgenden Sicherheitsrisiken Gebrauch:

- Abhören einer unverschlüsselten Verbindung zwischen Nutzer und Server
- *Phishing*: Der Nutzer gibt sein Passwort bei einer Fälschung des Services an
- *Keylogging*: Schadsoftware protokolliert die Tastenanschläge bei Eingabe der Nutzerdaten und leitet diese an den Angreifer weiter
- *Bruteforcing*: Häufige, zu kurze oder rein alphanumerische Passwörter können mittels automatisierter Software in kurzer Zeit erraten oder aus Wörterbüchern extrahiert werden
- Der Angreifer verschafft sich Zutritt zur Serverdatenbank und kopiert in Klartext gespeicherte Passwörter
- Mehrfach verwendete Passwörter für unterschiedliche Services

Um die Anzahl von Angriffsvektoren zu verringern existieren beispielsweise folgende Sicherheitsvorkehrungen:

- Verschlüsselte Kommunikation zwischen Benutzer und Server mittels Transport Layer Security [11]
- Serverseitige Vorschriften für sichere Passwörter bezüglich deren Länge und Komplexität [12]
- Umfassendes Sicherheitskonzept inklusive sicherer Konfiguration aller beteiligten Server, regelmäßiger Wartung, Einsicht von Logdateien und automatischer Datensicherung, wie etwa von Scarfone et al. [13] beschrieben

Eine Alternative zu diesem Loginverfahren ist die *Zwei-Faktor-Authentifizierung* (2FA). Hier wird in einem zweiten Schritt nach Kontrolle von Benutzer-ID und Geheimnis ein weiterer Zugriffscode des Nutzers abgefragt. Dieser wird in der Regel auf einem anderen Medium, welches sich im Besitz des Nutzers befindet, generiert. Hierfür eignen sich etwa Smartphone-Applikationen, ein Bestätigungscode per SMS oder ein (automatischer) Telefonanruf. Es kommen in der Regel zeitbasierte Generatoren [14], welche aus einem symmetrischen Schlüssel von Benutzer und Server und der momentanen Uhrzeit kryptografische Hashes berechnen, zum Einsatz. Diese Schlüssel sind für eine zeitlich begrenzte Schrittdauer (zum Beispiel 30 Sekunden) gültig, anschließend wird ein neuer Hash benötigt. Generische Applikationen wie Authy[15] können solche Codes für eine Vielzahl von Webservices, welche zeitbasierte Zwei-Faktor-Authentifizierung anbieten, generieren. Mehr Sicherheit bieten hardwarebasierte Zweitfaktoren wie der YubiKey [16], die als Schlüsselträger, beziehungsweise -generator dienen. Diese implementieren einen universellen Zweitfaktor [17] (U2F), wobei die Hardware einen privaten Schlüssel verwendet, um eine Nonce des Anmeldeservers zu signieren, wodurch dieser mit dem öffentlichen Schlüssel überprüfen kann, dass wirklich der verlangte Zweitfaktor zum Einsatz kam.
Die 2FA wird vom Bundesministerium für Sicherheit in der Informationstechnik in deren Richtlinien zur Authentifizierung empfohlen [18].
Neben lokaler Service-Authentifizierung gibt es die Authentifizierung über einen Drittanbieter, den sogenannten *Identity Provider* (IDP). Hier muss sich der Nutzer mit seinen Zugangsdaten beim IDP anmelden, welcher die Authentifizierung durchführt und ggf. gegenüber dem *Service Provider* (SP) die Identität des Nutzers bestätigt. Der Benutzer muss kein weiteres Konto beim SP anlegen. Es bestehen dieselben Angriffsvektoren wie bei der passwortbasierten Authentifizierung 2.1, der IDP kann deren Anzahl analog dazu verringern.

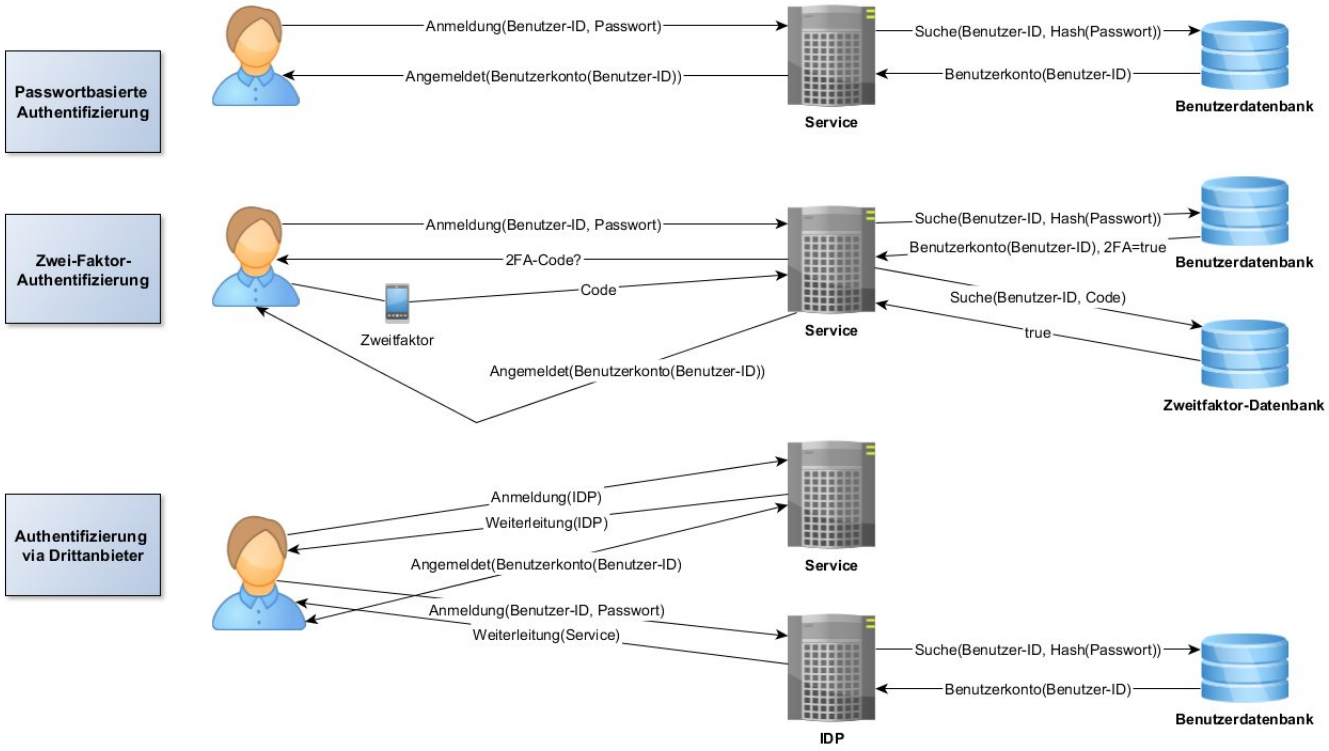

Abbildung 1: Drei gängige Authentifizierungsmethoden schematisch dargestellt.

2.1.1 OAuth2 und OpenID Connect

Open Authorization 2 (OAuth2) ist ein tokenbasiertes Autorisierungsprotokoll [19], das den Zugriff eines Services auf Nutzerdaten durch vorherige Identitätsprüfung bei einem Identity Provider ermöglicht. Dieser stellt dem SP ein *Access Token* aus, mithilfe dessen der SP Zugriff auf die Daten des Nutzers beim IDP erhält. OAuth2 ist kein Authentifizierungsprotokoll, es ermöglicht den Zugriff von Services auf im IDP hinterlegte Daten eines Nutzers.
OAuth2 kann verwendet werden, um Authentifizierungsfunktionalitäten bereitzustellen, denn eine erfolgreiche Autorisierung kann vom SP als eine erfolgreiche Authentifizierung des Nutzers beim IDP betrachtet werden; dort hinterlegte Benutzerdaten können vom SP zur Anmeldung des Nutzers im Service dienen. Dieser Anwendungsfall ist nicht Teil der OAuth2 Spezifikation, die Annahme dass einer erfolgreichen Autorisierung immer eine erfolgreiche Authentifizierung vorausgeht ist nicht allgemeingültig.
Um Authentifizierung auf Basis des OAuth2 Standards zu spezifizieren wurde der OpenID Connect Standard [20] entwickelt. Dieser baut auf OAuth2 auf und erweitert das Protokoll um eine Authentifizierungsschicht. Neben einem Access Token erhält der SP vom IDP zusätzlich das *Identity Token*, das eine erfolgreiche Authentifizierung des Benutzers beim IDP garantiert.

Beispiel
Im Folgenden wird der typische Ablauf einer Authentifizierung mit Autorisierung zum Zugriff auf eine geschützte Ressource des Service Providers schematisch dargestellt. Hierbei gibt es den Endnutzer (Browser), eine Webapplikation namens *App* als Relying Party (RP) und Google als OpenID Provider.
Bevor eine Authentifizierung oder Autorisierung möglich ist, muss App die Anmeldung via Google zur Verfügung stellen. Dafür registriert App sein Projekt in der Google Developers Console [21] und erhält dort eine ID und ein *Client Secret*. In diesem Beispiel will ein Benutzer bei App einen Beitrag verfassen. App erlaubt dies nur für angemeldete Benutzer und leitet den Nutzer zum Anmeldedialog weiter. Der Benutzer entscheidet, sich mit Google anzumelden, woraufhin App als Antwort einen *OpenID Authentication Redirect* generiert und den Nutzer damit zur Loginseite von Google weiterleitet. Dieser HTTP GET Request enthält einige OAuth-Parameter. Wichtig sind die folgenden:

- Client ID von App
- Der *scope*, mit welchem festgelegt wird, ob OpenID oder OAuth verwendet wird
- Ein *state*, um den aktuellen Status der Kommunikation zwischen App und Google festzuhalten
- Eine *redirect_uri*, wo App die Antwort von Google erwarten wird

Nachdem Google den Authentication Request vom Benutzer erhalten hat wird die übergebene Client-ID überprüft. Wenn diese nicht existiert wird die Anfrage verweigert, da App nicht bekannt ist. Nach Voraussetzung ist App bei Google registriert und der Benutzer meldet sich mit seinen Daten bei Google an. Nach erfolgter Anmeldung wird der Nutzer außerdem gefragt, ob er App wirklich Zugriff auf seine Daten gewähren möchte.
Nachdem der Benutzer dies bestätigt hat, generiert Google einen *Authorization Code*, aktualisiert den state der OAuth-Kommunikation und sendet den Benutzer mit diesen Daten mittels Redirect zurück zum *redirect_uri*-Endpunkt von App.
Dieser Code hat noch keinen Nutzen für App, er ist ein Zwischenprodukt, welches zum Erhalt des Identity Tokens dient. App muss nun direkt mit Google kommunizieren (also nicht mehr über den Browser des Nutzers) und sendet einen *Token Request*. Dieser enthält als *HTTP POST Data* den Authorization Code und in codierter Form die Client ID und das Client Secret als Authorization Header [22].
Er wird aufgrund dieser sensiblen Informationen über eine mit TLS [11] gesicherte Verbindung übermittelt. Nachdem Google den Request erhalten hat, wird der Authorization Code geprüft und bei Übereinstimmung wird App in der Response als JSON ein *Identity Token* sowie ein *Access Token* übermittelt.
Ersteres trägt und garantiert die Identität des Nutzers bei Google (etwa Benutzername und Aussteller) für App, während das Access Token von App benutzt werden kann, um Zugriff auf die Nutzerdaten zu erhalten, die ihm vom Benutzer gewährt wurden.
App gewährt dem Benutzer auf Basis dieser Informationen von Google nun Zugang.

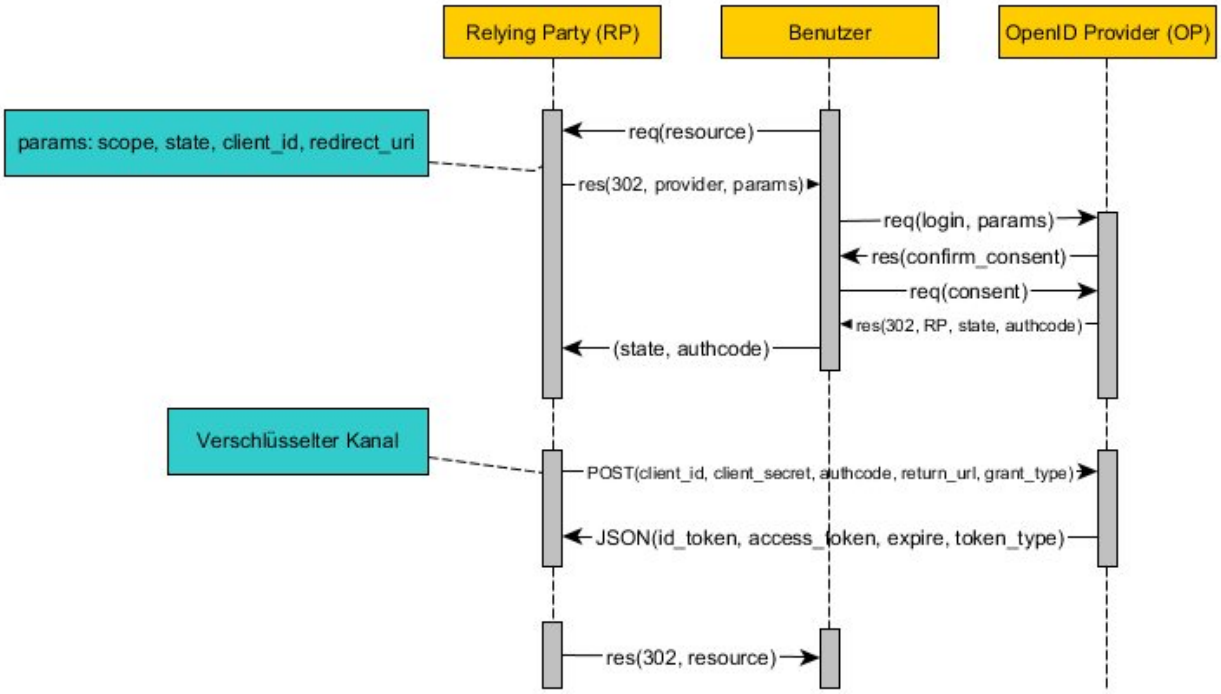

Abbildung 2: Typischer Authentifizierungsfluss mit OpenID Connect.

2.1.2 SAML2

Die *Security Assertion Markup Language 2.0* (SAML2) [23] ist ein XML-basiertes Protokoll, das den SAML Standard für den Austausch von Authentifizierungs- und Autorisierungsinformationen implementiert. SAML2 ermöglicht einen webbasierten SSO, bei welchem Daten zur Identität eines Benutzers vom Identity Provider (*SAML Authority*), bei welchem der Benutzer angemeldet ist, zu einem Service Provider (*Relying Party*) weitergegeben werden.

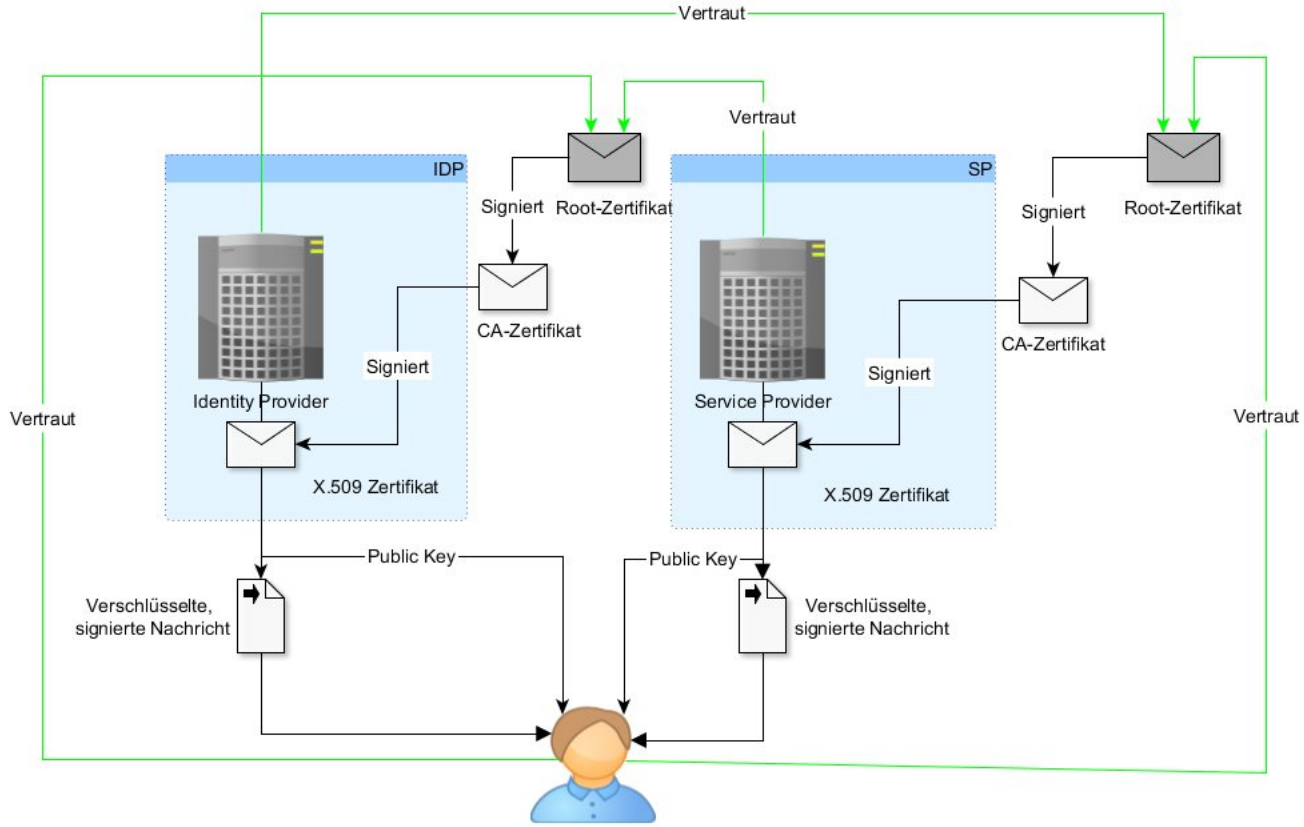

Abbildung 3: Vertrauenskonzept einer SAML2-Infrastruktur

Der SAML2 Standard lässt sich in 5 Aspekte unterteilen:

1. **SAML2 Assertions** [24]
   Assertions sind Datenpakete, welche Aussagen über eine Entität beinhalten, typischerweise einen Benutzer. Sie werden vom IDP erstellt und versandt und vom SP erhalten und benutzt, um Informationen über die Authentifizierung des Nutzers beim IDP oder seine dortigen Attribute (beliebige deskriptive Eigenschaften wie Gruppenzugehörigkeit, Wohnort et cetera) zu gewinnen und daraus Zugriffsrechte für den Benutzer im SP abzuleiten. Optional kann eine Assertion auch eine Aussage zur Autorisierung des Benutzers für die angeforderte Ressource im SP enthalten, wodurch Authentifizierung und Autorisierung komplett im IDP stattfinden.

2. **SAML2 Protocols** [24]
   In SAML2 sind 6 Protokolle definiert, welche folgendem Schema folgen: Der *Requester* sendet ein Element, welches vom Typ *RequestAbstractType* ableitet. Der *Responder* verarbeitet es und bereitet die angeforderten Informationen in einem Element auf, welches vom Typ *StatusResponseType* ist. Dieses wird zurück an den Requester übermittelt. Das wichtigste Protokoll ist das *Authentication Request Protocol*, welches SP-initiierten Login ermöglicht. Dabei wird ein vom SP (Requester) ein *AuthnRequest*-Element erstellt und über den Benutzer zum IDP (Responder) geschickt. In diesem AuthnRequest ist der SP als *Issuer* vermerkt. Als Antwort des IDP wird eine Assertion zum Authentifizierungsstatus des Benutzers erwartet.

3. **SAML2 Bindings** [25]
   SAML2-Nachrichten können über verschiedene Kommunikationsprotokolle ausgetauscht werden. Ein Binding beschreibt, wie der Nachrichtenaustausch in einem der SAML2 Protokolle auf ein Kommunikationsprotokoll wie HTTP übertragen wird. Für das Authentication Request Protocol in einem webbasierten werden meist das HTTP Redirect Binding (SAML-Nachricht als Teil des URL, nur für kurze Nachrichten geeignet) und das HTTP POST Binding (SAML-Nachricht im Body einer HTTP-Anfrage) verwendet.

4. **SAML2 Profiles** [26]
   SAML2 Profiles definieren für diverse Anwendungsfälle Regeln, nach welchen SAML2 Assertions unabhängig des gewählten Bindings versandt, extrahiert und verarbeitet werden. Neben dem wichtigen Profil *Web Browser SSO* existieren noch weitere Anwendungsfälle, die von SAML2 unterstützt werden. So sind unter anderem Profile für *Identity Provider Discovery* (Feststellung des zu verwendenden IDP für Authentifizierung eines Nutzers) oder *Single Logout* (Abmeldung eines Nutzers in SP oder IDP resultiert in Abmeldung von allen mit diesem IDP verbundenen Sitzungen des Nutzers).

5. **SAML2 Metadata** [27]
   SAML2 Profiles erfordern einen Konsens zwischen Authority und Relying Party bezüglich verwendeter Identifier, Endpunkte, unterstützten Bindings, verwendeten Zertifikaten, Verschlüsselungs- und Signierverhalten et cetera. Dafür wurde ein Format für Metadaten spezifiziert, in welchen beiden Parteien ihre jeweiligen Einstellungen und Präferenzen im *Extensible Markup Language* (XML)-Format auflisten. Dabei sind SAML2 Metadaten als Angabe der wichtigsten Parameter und Besonderheiten einer Partei zu verstehen, nicht als umfassende Dokumentation aller Funktionen. Das Fehlen einer Option in den Metadaten bedeutet nicht zwangsläufig, dass diese nicht unterstützt wird [27].

Beispiel
Betrachtet wird ein Authentifizierungsprozess mit einem Endnutzer (Browser), einer Webapplikation *'App'* als Service Provider und einem Shibboleth Identity Provider [28] namens *'Shib'*. Shibboleth ist eine quelloffene Software für SSO, von der Universität Tübingen zur Implementation von SAML2-basierten Identitätsdiensten verwendet wird. Dieses Beispiel zeigt, welche Vorgänge durchlaufen werden, damit der Endnutzer auf eine geschützte Ressource der Webapplikation zugreifen kann. Es zeigt vereinfacht den Authentifizierungsfluss wie in der Shibboleth Dokumentation [29] beschrieben.
Bevor die Authentifizierung per SAML2 verwendet werden kann, tauschen Shib und App ihre jeweiligen Metadaten aus. Hierbei erfährt App den Namen (*EntityID*) von Shib (https://shib.auth.de), dessen öffentliche Schlüssel seiner X.509 [30] Zertifikate für Signierung mit RSA [31] und Verschlüsselung mit RSA [31] der Nachrichten sowie der URL, auf welchen der Nutzer zur Anmeldung weitergeleitet werden soll (https://shib.auth.de/SSO) und Kontaktdaten des zuständigen Administrators. Im Gegenzug erhält Shib die SP Metadaten von App, in welchen Shib erfährt dass App:

- als 'https://app.website.de/saml2' identifiziert werden möchte
- der zuständige Ansprechpartner unter 'admin@app.website.de' erreichbar ist
- seine Anfragen als XML Signature [32] mit RSA signiert, den öffentlichen Schlüssel zum X.509 Zertifikat von App, und auch von Shib derartig signierte Antworten möchte
- verschlüsselt kommuniziert und an ihn gesendete Antworten mit einem ebenfalls angegebenen öffentlichen Schlüssel zu seinem X.509 Zertifikat RSA-verschlüsselt haben möchte.

Beide Parteien müssen den Wurzelzertifikaten der Vertrauenskette beider X.509 Zertifikate vertrauen. Zu Beginn versucht der Nutzer mit seinem Browser auf eine geschützte Ressource, den Benutzerbereich von App unter 'https://app.website.de/users', zugreifen. App kennt die Identität des Benutzers zu diesem Zeitpunkt noch nicht und verweigert den Zugriff. Um seine Identität und die Zugriffsberechtigung auf die angeforderte Ressource nachzuweisen, generiert App eine verschlüsselte und signierte Anfrage für Shib, in welcher die interne ID dieses Anmeldungsvorgangs, die angeforderten Attribute des Users (E-Mail und Gruppenzugehörigkeit) und der *Assertion Consumer Service* von App, welcher die Antwort von Shib auswertet, angegeben sind.
Im nächsten Schritt wird diese Anfrage an den Nutzer übergeben und dessen Browser zur Anmeldeseite von Shib weitergeleitet. Shib erhält und entschlüsselt die Anfrage und prüft die Signatur um sicherzustellen, dass die Anfrage wirklich von App kommt. Anschließend legt Shib anhand der Anfrage fest, wie er den Nutzer authentifizieren muss und leitet diesen durch den Authentifizierungsprozess. Nachdem sich der Benutzer bei Shib angemeldet hat, sammelt Shib aus seiner Nutzerdatenbank dessen Daten und filtert alle, die nicht von App benötigt werden, heraus. Die übrig gebliebenen Attribute werden nach den Anforderungen von App als Assertion verpackt, verschlüsselt, signiert und an den Benutzer übergeben.

Im letzten Schritt wird dessen Browser zum Assertion Consumer Service von App weitergeleitet. App entschlüsselt die Antwort und prüft die Signatur. Wenn die Antwort wirklich von Shib kommt, werden die Attribute ausgewertet und der Nutzer bei Korrektheit aller Daten in App angemeldet und kann auf den Benutzerbereich zugreifen. Da App Shib vertraut, werden die übermittelten Daten als wahr angesehen und App kennt den Benutzer von nun an unter der von Shib übergebenen Identität.

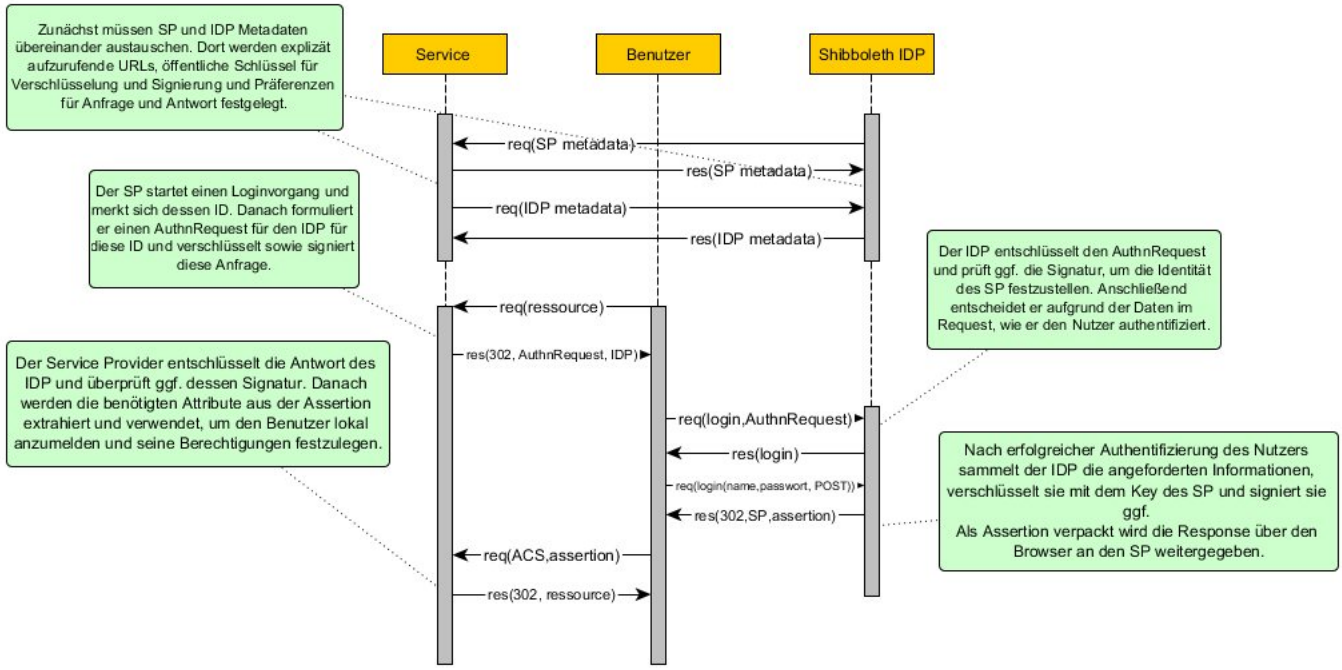

Abbildung 4: Vereinfachtes Schema eines SAML2 Authentifizierungsflusses.

2.1.3 SAML2 und OAuth2/OpenID Connect im Vergleich

SAML2 und OpenID Connect sind Lösungen zur Authentifizierung, welche beide [33][34] von großen Organisationen verwendet und unterstützt werden. OpenID Connect ist der jüngere Standard und wird von Identity Providern wie Amazon [35], Google [36] und Microsoft [37] unterstützt.
SAML2 deckt durch verschiedene Profile [26] eine Bandbreite an Anwendungsfällen ab und findet Anwendung bei Institutionen und Firmen [33]. OpenID Connect findet Adaptation vor allem bei Social Media-Diensten [34]. Die Protokolle unterscheiden sich in der Etablierung von Vertrauensbeziehungen zwischen beteiligten Parteien. In SAML2 sowie OpenID Connect ist dies zum einen bilateral möglich, indem zwischen Relying Party und Authority, bzw, OpenID Provider eine manuelle Registrierung des Gegenübers im lokalen System vorgenommen wird. In SAML2 funktioniert dies durch Integration der Metadaten des Gegenübers auf beiden Seiten. In OpenID Connect wird die Relying Party mit Client-ID und Client Secret beim OpenID Provider registriert. SAML2 unterstützt außerdem ein Föderationskonzept, hierbei existiert eine private Gruppe von Entitäten, genannt *Federation*, innerhalb welcher allen Mitgliedern der Federation vertraut wird. Vom Operator der Federation signierte Metadaten, welche alle Mitglieder der Federation auflisten, ermöglichen eine dynamische Anpassung aller Mitglieder bei Änderungen an der Federation. In OpenID Connect existiert ein derartiger Föderationsbegriff nicht, dynamische Selbstregistrierung neuer Relying Parties, welchen automatisch vertraut wird, kann alternativ erlaubt werden. In SAML2 werden Assertions digital von der Authority mit ihrem Zertifikat signiert. Dieser Signatur wird vertraut, wenn auch der *Certification Authority*, welche dieses Zertifikat ausstellt, bzw deren Wurzelzertifikat, vertraut wird. OpenID Connect unterstützt dies optional. Standardmäßig wird in OpenID Connect registrierten Relying Parties auch ohne Signatur vertraut, solange Client-ID und Secret korrekt sind.

Für SAML2 ist ein Profil für Single Logout, also die simultane Abmeldung von allen Diensten, definiert. Zum Zeitpunkt dieser Arbeit sind keine betriebsbereiten Implementierungen dieses Profils verfügbar; eine mögliche Erklärung dafür veröffentlichte das Shibboleth Consortium [38]. Für OpenID Connect ist der Single Logout noch nicht standardisiert, es existieren jedoch Entwürfe dafür. [39][40][41]

2.2 Autorisierung

Als Autorisierung werden Vorgänge bezeichnet, die anhand vordefinierter Regeln den Zugriff auf Ressourcen und Funktionen eines Systems kontrollieren. Im Kontext eines SSO-Systems sollen Benutzer nach erfolgter Authentifizierung anhand ihrer Identität oder extern zugesicherter Vertrauenswürdigkeit im Rahmen der Autorisierung Zugriff auf bestimmte Systemdaten und -funktionen erhalten. Die in 2.1 beschriebenen Protokolle OpenID Connect und SAML2 ermöglichen auch eine Autorisierung in direkter Verbindung mit der Authentifizierung über Access Tokens, beziehungsweise Assertions. Möglich ist die Autorisierung von Ressourcen entweder für einen bestimmten URL, bzw ein Verzeichnis, oder nur für einzelne Dateien.

2.2.1 Claims

Claims sind Paare aus einem Schlüssel und einem oder mehreren Werten. Mithilfe von Claims lässt sich Autorisierung unabhängig[42] von vorhergehenden Authentifizierungsverfahren implementieren. Ein Claim repräsentiert eine Aussage einer Entität, die diese über sich selbst getroffen oder von einer weiteren Entität erhalten hat. Auf Basis dieser Aussagen kann eine Applikation Entscheidungen über das weitere Verfahren mit der Entität treffen. Ein Claim kann sich auf Identität, Rechte, Gruppenzugehörigkeit und weitere Eigenschaften oder Berechtigungen beziehen. [42] In einem SSO-System kann ein Benutzer Claims über sich tätigen, von der Applikation erhalten, oder von einem externen IDP verliehen bekommen. Der SP stellt dem Benutzer abhängig von dessen Claims Rechte und Funktionen innerhalb des Services zur Verfügung. Welche Berechtigungen und Richtlinien mit einem Claim einher gehen wird vom SP bestimmt. Rollenbasierte Autorisierung verwendet Claims beispielsweise um für Benutzer eine Rollenzugehörigkeit anzugeben, ein Service hat für jede Rolle entsprechende Berechtigungen hinterlegt und kann diese gemäß des Rollen-Claims des Benutzers erteilen. Claims können zum Beispiel als Cookie im Webbrowser eines Benutzers für eine Sitzung hinterlegt werden.

2.3 ASP.NET Core

Active Server Pages .NET Core (ASP.NET Core) [43] ist ein plattformübergreifendes, quelloffenes Framework zur Entwicklung von Webapplikationen. ASP.NET Core ist Teil der .NET Core Entwicklungsplattform[44] von Microsoft. Im Gegensatz zum proprietären .NET Framework, welches mit den meisten Windows-Versionen ausgeliefert wird, ist der .NET Core eine Sammlung der wichtigsten Bibliotheken des .NET Frameworks, also eine Untermenge davon. Eine mit ASP.NET Core entwickelte Webanwendung kann auch auf Linux/Unix-basierten Webservern ausgeführt werden kann.

2.4 ASP.NET Core Identity

ASP.NET Core Identity [45] ist Teil von ASP.NET Core und stellt eine umfangreiche Bibliothek zur Benutzerverwaltung zur Verfügung. Identity ermöglicht unter anderem die *Code-First* Implementation von Benutzerkontenverwaltung, lokale, Zwei-Faktor- und OpenID-basierte Authentifizierung, Claim- und rollenbasierte Autorisierung und Sitzungsverwaltung. Datenbankschemata werden automatisch anhand der im Code definierten Attribute generiert, im Gegensatz zu *Database-First* Implementationen, wo zunächst ein Datenbankschema bestimmt und anschließend der Code, welcher dieses benutzt, erstellt wird.

2.5 Sustainsys.Saml2

Sustainsys.Saml2 [46] ist eine quelloffene Bibliothek von der schwedischen Firma Sustainsys, welche SAML2 Service Provider Funktionalität für .NET Framework und seit auch kurzem auch .NET Core bereit stellt. Die Bibliothek gliedert sich als Middleware in ASP.NET Core Identity ein und implementiert die SP-Hälfte eines SAML2 Stacks mit Unterstützung für Föderationen [29, siehe 3.3], automatisches Generieren von Metadaten sowie Verschlüsselung und Signierung.

2.6 Microsoft Internet Information Services

Microsoft Internet Information Services (IIS) [47] ist eine Serverplattform von Microsoft. IIS kann unter anderem als Mail-, Web- oder Fileserver verwendet werden. Am häufigsten wird IIS zur Auslieferung von ASP.NET-basierten Webseiten oder zur Ausführung von .NET-basierten Anwendungen benutzt. IIS ist nur für Windows-Systeme verfügbar.

2.7 Microsoft SQL Server

Microsoft SQL Server (MSSQL) [48] ist ein relationales Datenbankmanagementsystem von Microsoft. Es basiert wie MySQL [49] oder PostgreSQL [50] auf der Datenbanksprache *Structured Query Language (SQL)*, deren Standards es weitgehend unterstützt. MSSQL ist für Windows und Linux in verschiedenen kostenfreien und kostenpflichtigen Editionen verfügbar.

3 Lösungskonzept

3.1 Integratives Framework für Authentifizierung und Autorisierung

Ziel dieser Arbeit ist eine adaptierbare Lösung von verschiedenen Authentifizierungs- und Autorisierungsaufgaben. Sie soll diese Funktionalitäten mit geringem Konfigurationsaufwand sowohl für bereits bestehende als auch neu entwickelte Systeme und Applikationen bereitstellen. Es kommt die aktuelle Version 2.0 von ASP.NET Core (siehe Kapitel 2.3) zum Einsatz, welche sich durch Plattformunabhängigkeit sowie und im Vergleich zum .NET Framework durch schlanke Bibliotheken unterscheidet. Es wurde darauf geachtet, mit Ausnahme der Sustainsys.Saml2 (siehe Kapitel 2.5) Bibliothek, nur feste Bestandteile des .NET Standard [51] zu verwenden, da für diese regelmäßige Pflege und Versorgung mit Updates gewährleistet sind. Dies verleiht der Plattform Langlebigkeit und kurze Wartungszeiten.

3.2 Webserver und Datenbank

Da die Arbeit auf ASP.NET Core basiert, ist der Betrieb mit diversen Server- und Datenbanksystemen möglich. Aufgrund guter Kompatibilität mit der Entwicklungsumgebung wurde IIS 10 Express als Webserver verwendet. Die Plattform kann nativ über den Multi-Plattform Server Kestrel [52] als auch mit einem zusätzlichen *Reverse Proxy Server* betrieben werden. Hierfür eignen sich gängige Server wie Apache [53] oder nginx [54].
Als Datenbank wurde eine relationale MSSQL LocalDb verwendet, welche sich wegen ihrer minimalen Größe zum Testen der Plattform eignet. Für eine Produktionsanwendung kann ein MSSQL Server, MySQL oder sonstige relationale Datenbanksysteme verwendet werden. Es ist auch ein Betrieb mit NoSQL-Lösungen möglich, dies erfordert Anpassungen des Programms.

3.3 Lokale Authentifizierung

Benutzer sollen in der Lage sein, sich mit ihrer E-Mailadresse und einem Passwort bei der Plattform zu registrieren und anzumelden. Bei einer Registrierung soll außerdem ein Benutzerprofil mit Standardrechten angelegt werden. Ferner soll es Benutzern möglich sein, ihr Passwort zu ändern und bei Verlust des Passworts automatisch zurücksetzen zu lassen. Um Zugriff auf die Plattform zu erhalten ist es notwendig, die E-Mailadresse zunächst über einen Bestätigungslink, welcher an die genannte E-Mailadresse geschickt wird, zu validieren. Dies ist nötig um sicherzustellen, dass der Benutzer für Rückfragen und Probleme auch wirklich unter den angegebenen Daten zu kontaktieren ist.

3.4 Externe Authentifizierung via OAuth2 + OpenID Connect

Neben einer lokalen Anmeldung sollen sich auch Nutzer mit Konten, die bei gewählten externen Anbietern liegen, in der Plattform einloggen können. Für Anbieter, welche den OAuth2- oder OpenID Connect-Standard zur Authentifizierung unterstützen, soll dies in Verbindung mit einem aus den Profildaten des externen Anbieters generierten lokalen Benutzerkonto möglich sein. Das System merkt sich den verwendeten Anbieter und kann das externe Profil jederzeit dem in seiner lokalen Datenhaltung zuordnen.

3.5 Externe Authentifizierung via SAML2/Shibboleth

Wie an vielen anderen Universitäten und Hochschulen kommt an der Universität Tübingen Shibboleth [29] als Software für SSO zum Einsatz. Der Shibboleth IDP der Universität Tübingen ist Teil der DFN-AAI [55] (Deutsches Forschungsnetz Authentifizierungs- und Autorisierungs-Infrastruktur) des DFN Vereins. Diese Föderation bildet die Grundlage für vertraulichen Austausch zwischen Mitgliedereinrichtungen und schafft eine Vertrauensbasis für diese. Der Shibboleth IDP fungiert als Identity Provider einer SAML2[23] Infrastruktur. Diese Plattform kann als Service Provider diesen und weitere Identity Provider der DFN-AAI nutzen und Mitgliedern einer Universität die Anmeldung an der Plattform mittels eines bereits bestehenden Zugangs am IDP der Universität ermöglichen.

3.6 Autorisierung

Nach erfolgter Authentifizierung wird das lokale Benutzerprofil geladen und eine Sitzung für den Benutzer eröffnet. Für die Dauer der Sitzung erhält der Benutzer die Berechtigungen in der Plattform, welche in der Datenbank für ihn hinterlegt sind. Bei Erstanmeldung und Registrierung wird dort für Benutzer eine Standardsammlung von Rechten hinterlegt, welche nachträglich von Administratoren geändert werden können.

3.7 Administration

Die Plattform enthält eine Oberfläche für Administratoren, mit welcher diese Benutzer einsehen, bearbeiten, löschen oder neu erstellen können. Ebenfalls können die Rechte von Benutzern geändert werden, welche diese dann ab ihrer nächsten Sitzung vom System verliehen bekommen.

4 Implementierung

4.1 Model-View-Controller Architektur

ASP.NET Core unterstützt die Model-View-Controller (MVC) Architektur für Webapplikationen [56]. Die Plattform dieser Arbeit implementiert die aktuelle Version von ASP.NET Core MVC [57]. Die Applikation wird in drei Teile separiert:

- **Controller**
  Ein Controller übernimmt Aufgaben wie Routing und Verarbeitung von Benutzereingaben und -anfragen.
- **Model**
  Models werden in ASP.NET Core vor allem zur Kapselung von Nutzereingaben und Aufbereitung von Datenbankanfragen verwendet.
- **View**
  Views werden für Präsentationslogik verwendet und bereiten die Daten aus Models grafisch (HTML, CSS) für den Webbrowser auf.

Durch die Trennung von Darstellungslogik, Geschäftslogik und Interaktionslogik können diese Komponenten der Plattform einzeln verändert werden.

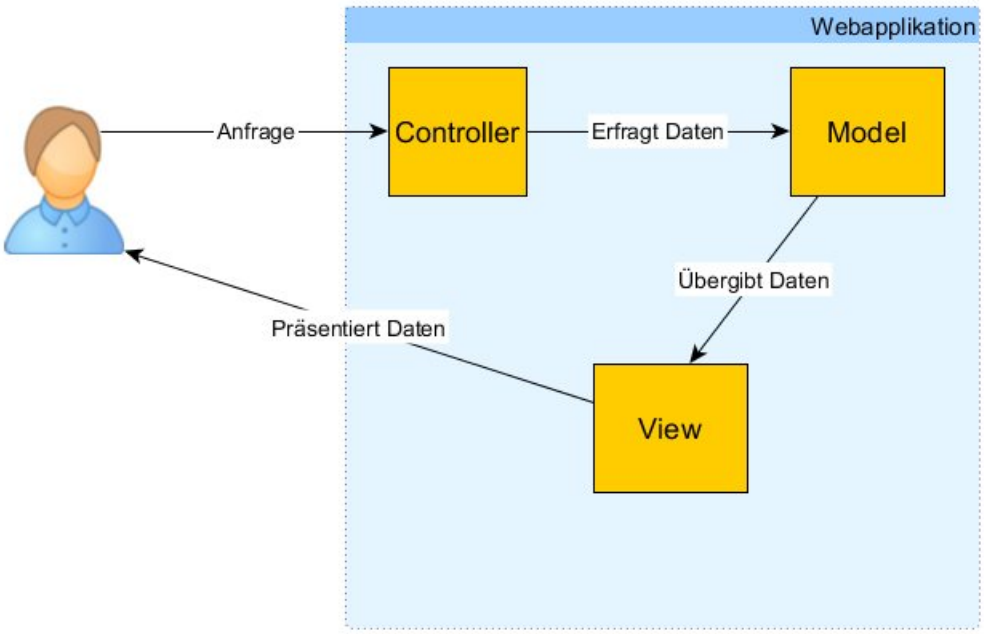

Abbildung 5: Schematischer MVC-Aufbau

4.2 Claims und Policies

Sämtliche Autorisierungsaufgaben dieser Plattform werden lokal durchgeführt. Es wird claimbasierte Autorisierung (s. 2.2.1) verwendet, um Benutzereigenschaften zu spezifizieren. Mithilfe von *Policies* werden aus den Claims eines Benutzers Zugriffsrechte abgeleitet. Bei erstmaliger Registrierung erhält jeder Benutzer den *user*-Claim, welcher ihn als registrierten Benutzer kennzeichnet. Eine dazugehörige Policy erlaubt für Benutzer mit diesem Claim den Zugriff auf Profilinformation.
Weiterhin existiert ein *admin*-Claim, welcher an Administratoren der Plattform vergeben werden kann. Die dazugehörige Policy erlaubt den Zugriff auf die Administratoroberfläche.

Weitere Claims und Policies können vor dem Start der Plattform hinzugefügt und automatisch in der Datenbank persistiert werden. So kann die Plattform an verschiedene Anwendungsszenarien angepasst werden.

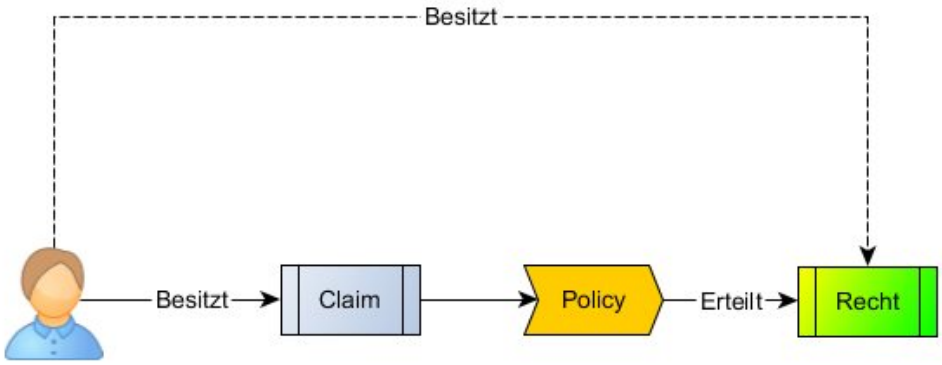

Abbildung 6: Vom Claim zum Recht

4.3 Authentifizierung

Die Plattform unterstützt folgende Authentifizierungsmodi:

- Lokale Authentifizierung
- Lokale Authentifizierung mit zusätzlicher Zwei-Faktor-Authentifizierung
- OAuth2/OpenID Connect-basierte Authentifizierung mit Facebook und Google
- SAML2-basierte Authentifizierung mit dem Test-IDP der Universität Tübingen
- Automatische Auswahl der o.g. Authentifizierungsmethoden anhand einer Benutzer-ID

4.3.1 Lokale Authentifizierung

Registrierung
Benutzer können sich mit ihrer E-Mailadresse und einem Passwort registrieren. Eingabefelder werden auf gültige E-Mailadressen und ausreichend lange Passwörter überprüft. Nach einer erfolgreichen Registrierung wird eine automatisch generierte E-Mail an die angegebene Mailadresse geschickt, welche einen Aktivierungslink enthält. Dieser muss im Browser geöffnet werden, um die E-Mailadresse zu bestätigen. Erst danach ist eine Anmeldung an der Plattform möglich.

Anmeldung
Registrierte Benutzer mit einer bestätigten E-Mailadresse können sich an der Plattform anmelden, um Zugriff auf ihr Benutzerprofil zu erhalten. Nach erfolgter Anmeldung erhält der Benutzer für die Dauer der Sitzung den Claim, welcher ihm in der Datenbank zugewiesen ist.

Abmeldung
Nach erfolgter Abmeldung wird die Sitzung des Nutzers terminiert und der Cookie, welcher den Claim des Nutzers beinhaltet, wird ungültig. Der Claim bleibt in der Datenbank bestehen und wird bei der nächsten Anmeldung erneut verliehen.

Passwortrücksetzung Der Benutzer kann ein vergessenes Passwort zurücksetzen lassen. Nach Angabe der E-Mailadresse des betroffenen Kontos erhält er eine E.Mail mit einem Link, um ein neues Passwort zu vergeben. Hierbei wird bewusst auf ein temporäres oder automatisch generiertes neues Passwort verzichtet, da dies häufig mit erneutem Verlust des Passworts oder unsicheren Aufbewahrungsmethoden einhergeht [12]

4.3.2 Lokale Authentifizierung mit Zwei-Faktor Authentifizierung

Nachdem sich ein Nutzer erfolgreich registriert und angemeldet hat, kann er in seinem Profil die Zwei-Faktor-Authentifizierung aktivieren. Die Plattform unterstützt das TOTP-Verfahren [14] zur Generierung zeitbasierter Einmalpasswörter. Zur Aktivierung benötigt der Benutzer eine Applikation wie Authy [15] oder Google Authenticator [58], welche dieses Verfahren unterstützt. Die Plattform generiert einen Aktivierungscode. Nachdem der Benutzer diesen in seiner Applikation eingegeben hat, kann diese 30 Sekunden gültige Einmalpasswörter generieren, welche ab der nächsten Sitzung des Nutzers zur Anmeldung benötigt werden.
Im Falle des Verlusts des Geräts oder der Applikation wird bei der Einrichtung ein Deaktivierungscode zur Verfügung gestellt, mit welchem die Zwei-Faktor-Authentifizierung deaktiviert werden kann. Außerdem können Administratoren die Funktion für einen Benutzer deaktivieren.

4.3.3 OAuth2/OpenID Connect-basierte Authentifizierung

Anmeldung
Benutzer können sich mit den Diensten Facebook und Google an der Plattform anmelden. Realisiert wird dies durch Authentifizierungs-Middleware in ASP.NET Core Identity. Es handelt sich um SP-initiierten Login wie in 2.1.1 beschrieben. Bei der erstmaligen Anmeldung über den gewählten Dienst muss der Nutzer zusätzlich in der Plattform eine E-Mailadresse als Benutzerkennung angeben. Mit dieser wird ein lokaler Benutzer angelegt, welchem dann die Claims des Benutzers zugeordnet werden können. Daraufhin wird eine Sitzung für den Benutzer eröffnet. Die Middleware kann um weitere Dienste wie Twitter erweitert werden.

Abmeldung Bei einer Abmeldung wird die aktuelle Sitzung des Benutzers terminiert und der Cookie ungültig. OpenID Connect besitzt noch keinen Standard für Single Logout, daher bleibt die Sitzung des Benutzers im OpenID Identity Provider (Facebook, Google) bestehen.

4.3.4 SAML2-basierte Authentifizierung

Anmeldung Benutzer können sich mit ihrem Konto des Shibboleth Test-IDP der Universität Tübingen anmelden. Aus Sicherheitsgründen ist nur die Anmeldung über diese Testumgebung möglich, die Plattform ist kompatibel mit allen IDPs der DFN-AAI Föderation [55]. Realisiert wird diese Funktion durch die Sustainsys.Saml2 [46]-Middleware. Diese ist kompatibel mit den wichtigsten Protokollen und Bindings (siehe 2.1.2) und signiert AuthnRequests mit einem X.509 Zertifikat. Nach erfolgter Authentifizierung des Benutzers beim IDP wird die Assertion entschlüsselt und die Signatur des Test-IDP validiert. Bei erstmaliger Anmeldung muss der Nutzer eine E-Mailadresse angeben, mit welcher ein lokaler Benutzer erstellt wird. Bei erfolgreicher Authentifizierung wird eine Sitzung für dem Benutzer eröffnet, welche seine Claims enthält.

Abmeldung
Bei einer Abmeldung wird die Sitzung des Benutzers terminiert und der Cookie ungültig. Da Shibboleth keinen Single Logout unterstützt, bleibt die Sitzung des Benutzers im Test-IDP bestehen, bis diese ihre Timeout-Dauer überschritten hat.

Vertrauen Requests der Plattform und Antworten des IDP werden digital signiert. So können sich beide Parteien sicher sein, dass die Nachricht wirklich vom Gegenüber stammt. Dabei vertrauen beide Parteien jeweils dem Zertifikat des anderen, da dieses von einer ebenfalls vertrauten Certification Authority ausgestellt wurde. SAML2 Metadaten für die Plattform als SP werden automatisch von der Sustainsys.Saml2-Bibliothek erstellt und können vom IDP heruntergeladen werden. Die Bibliothek automatisiert ebenfalls den Download von IDP-Metadaten.

Abbildung 7: Vertrauenskette der Plattform (links) und des Test-IDP (rechts)

4.3.5 Automatische Auswahl der Authentifizierungsmethode

Die Plattform bietet neben der manuellen Auswahl der Authentifizierungsmethode auch eine Funktion zur automatischen Erkennung einer passenden Methode. Hierbei wird der Benutzer aufgefordert, seine E-Mailadresse als Benutzerkennung einzugeben. Daraufhin wird überprüft, ob für diese bereits ein Konto existiert. Wenn dies der Fall ist, wird automatisch die für dieses Konto hinterlegte Authentifizierungsmethode ausgeführt. Falls nicht, wird mittels regulärer Ausdrücke erkannt, ob es sich um eine universitäre E-Mailadresse oder eine Gmail-Adresse [59] handelt und ggf. die entsprechende Authentifizierungsmethode gewählt. Sollte auch dies nicht zutreffen, wird der Benutzer zum Registrierungsdialog weitergeleitet, in welchem alle Methoden manuell verfügbar sind.

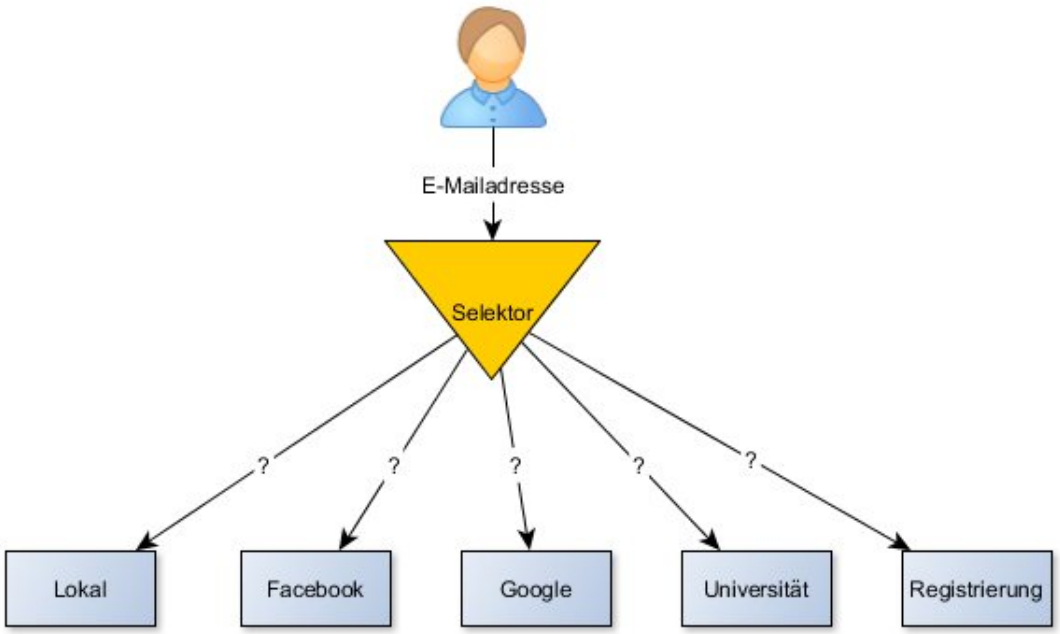

Abbildung 8: Automatische Auswahl zur Authentifizierung

4.3.6 Mehrere Authentifizierungsmethoden pro Nutzer

Sobald sich ein Nutzer das erste Mal an der Plattform angemeldet hat, besitzt er die Möglichkeit, weitere Authentifizierungsmethoden für sein Konto hinzuzufügen. So können beispielsweise ein lokales Konto, ein Facebookkonto und ein universitäres Konto mit demselben Konto in der Plattform verknüpft werden.

4.4 Administration

Benutzer, welche den *admin*-Claim besitzen, erhalten Zutritt zur Administrationsoberfläche. Diese enthält grundlegende *Create-Read-Update-Delete* (CRUD)-Funktionen [60] zur Benutzerverwaltung:

4.4.1 UserList

Zeigt eine Liste aller Benutzer an. Zeigt außerdem für jeden Nutzer dessen Profildaten, Einstellungen sowie Claims und verknüpfte Identity Provider an.

4.4.2 Create

Erlaubt es Administratoren, neue Benutzerkonten anzulegen. Benötigt werden eine E-Mailadresse und ein hinreichend sicheres [12] Passwort.

4.4.3 Edit

Erlaubt die Änderung sämtlicher Datenbankwerte eines Benutzerkontos. Dies beinhaltet auch Claims, Identity Provider und gehashte Passwörter. Änderungen sind ab der nächsten Sitzung des Benutzers aktiv.

4.4.4 Delete

Erlaubt das Löschen eines Benutzers aus der Datenbank.

4.5 Controller

Die Controller bilden das Rückgrat der Plattform. Sie dirigieren Anfragen zu den richtigen Endpunkten und rufen Funktionen verwendeter Bibliotheken wie ASP.NET Core Identity und Sustainsys.Saml2 auf. Für die meisten dieser Controllerfunktionen existiert jeweils ein View, in welchem der Benutzer die Eingaben für den Controller tätigt sowie ein Model, welches die Nutzereingaben auf syntaktische Korrektheit prüft. Für diese Plattform wurden mehrere Controller erstellt, deren Funktionen im Folgenden kurz beschrieben werden.

4.5.1 AccountController

Im AccountController wurden sämtliche Authentifizierungsfunktionen realisiert. Er erstellt jeweils eine Instanz des *UserManager* und *SignInManager*, welche Funktionen aus ASP.NET Core Identity für Benutzer-, respektive Anmeldeverwaltung zur Verfügung stellen. Die wichtigsten *Actions* des AccountController sind wie folgt:

Login
Leitet den Nutzer zum View für den manuellen Anmeldedialog für lokale und externe Konten. Nach Eingabe von lokalen Benutzerdaten werden diese vom SignInManager überprüft und bei Erfolg eine Sitzung erstellt. Leitet nach Wahl eines externen Loginproviders weiter zur Methode *ExternalLogin*.

ExternalLogin
Wird von *Login* oder *CheckForExistingUser* aufgerufen. Je nach übergebenem Provider initiiert ExternalLogin ein Authentifizierungsverfahren mit Facebook, Google oder dem Test-IDP der Universität Tübingen. Für Facebook und Google wird Middleware aus ASP.NET.Core Identity verwendet, für den universitären Login die Middleware von Sustainsys.Saml2. Die Middleware formuliert eine Anfrage an den entsprechenden IDP gemäß 2.1.1, beziehungsweise 2.1.2 und erwartet die Antwort des IDP in der Methode *ExternalLoginCallback*.

ExternalLoginCallback
Diese Methode wird vom IDP nach erfolgter Authentifizierung aufgerufen. Das resultierende Identity Token, beziehungsweise die SAML Assertion wird von der entsprechenden Middleware ausgewertet. Bei Erstanmeldung wird nach Angabe einer E-Mailadresse ein Benutzerkonto im UserManager mit Daten des IDP erstellt. Anschließend wird der Benutzer über den SignInManager in seinem lokalen Konto angemeldet.

Register
Leitet den Benutzer zum View für den Registrierungsdialog. Dort kann der Nutzer ein lokales Konto mit E-Mailadresse und Passwort erstellen. Bei einer gültigen Eingabe wird das Benutzerkonto in der Datenbank erstellt und eine Bestätigungsemail versandt. Erst nachdem das Konto bestätigt wurde, ist eine Anmeldung möglich.

LoginParser
Leitet den Benutzer zum View für die automatische Auswahl der Authentifizierungsmethode. Bei Eingabe einer gültigen E-Mailadresse wird die Methode *CheckForExistingUser* aufgerufen, welche anhand der Eingabe eine Authentifizierungsmethode auswählt.

CheckForExistingUser
Sucht zunächst in der lokalen Datenbank nach einem Eintrag für die übergebene E-Mailadresse und leitet bei Übereinstimmung zur dort vermerkten Authentifizierungsmethode weiter. Falls es keine Übereinstimmung gibt, wird die Struktur der E-Mailadresse mit regulären Ausdrücken auf mehrere Muster überprüft, welche auf ein Gmail-Konto oder eine universitäre E-Mailadresse hindeuten. Bei Übereinstimmung wird zur erkannten Authentifizierungsmethode weitergeleitet. Falls auch hier keine Übereinstimmung gefunden wird, wird der Benutzer zur Methode *Register* weitergeleitet.

Logout
Beendet bei Aufruf die Sitzung des Benutzers in der Plattform.

4.5.2 AdminController

Der AdminController stellt die Funktionen der Administrationsoberfläche bereit. Der Zugriff ist nur für Benutzer mit *admin*-Claim möglich. Er implementiert die in 4.4 beschriebenen Funktionen, indem er eine Instanz des UserManagers erstellt und mit diesem Benutzer abruft, erstellt, ändert oder löscht. Für jede der Funktionen existiert außerdem ein View, welcher vom Nutzer die geforderten Eingaben erhält.

4.5.3 ManageController

Der ManageController implementiert die Selbstverwaltung für ein Benutzerkonto. Mit den Funktionen des ManageController kann der Benutzer seine Informationen und Einstellungen einsehen und Änderungen daran vornehmen. Dies wird wieder über Instanzen des SignInManager und UserManager realisiert. Die wichtigsten *Actions* des ManageController sind wie folgt:

Index
Zeigt eine Übersicht des Benutzerkontos mit grundlegenden Profilinformationen.

ChangePassword
Lädt den View zur Änderung des Passworts. Bei korrekter Eingabe des alten Passworts sowie einem neuen Passwort wird mittels UserManager der in der Datenbank hinterlegte Passwort-Hash geändert und danach eine neue Sitzung des Benutzers geöffnet und die alte terminiert.

ExternalLogins
Lädt den View, welcher alle mit dem Konto verknüpften IDPs anzeigt. Es können weitere hinzugefügt und alte gelöscht werden. Um ein neues Konto bei einem IDP hinzuzufügen, muss der Nutzer einmal die *ExternalLogin*-Methode des AccountController durchlaufen und wird anschließend zurück zu dieser Methode geleitet. Daraufhin wird mittels UserManager in der Datenbank ein neuer Provider für den Benutzer hinzugefügt. Bei Löschung eines verknüpften Kontos entfernt der UserManager den Provider aus der Datenbank.

TwoFactorAuthentication
Leitet den Benutzer über mehrere *Actions* durch die Einrichtung einer Zwei-Faktor-Authentifizierung. Diese kann im eingeloggten Zustand wieder deaktiviert werden. Der

UserManager vermerkt die Benutzung der Zwei-Faktor-Authentifizierung im Datenbankeintrag des Nutzers.

4.6 Sicherheit

Authentifizierungsdaten sind per Definition sensibel und müssen vor Außenstehenden geheim gehalten werden. Diese Plattform wurde mit dem Ziel erstellt, Authentifizierung und Autorisierung auf eine für Benutzer, Betreiber und Dritte sichere Weise durchzuführen und übertragene Daten vor Diebstahl zu schützen und sicher zu hinterlegen. Dafür wurden folgende Maßnahmen getroffen:

- **Verschlüsselte Übertragung aller Daten** Die Plattform kommuniziert ausschließlich verschlüsselt. Sämtlicher Browserverkehr wird mit HTTPS [61] verschlüsselt übertragen. HTTP-Anfragen werden mittels *URL Rewriting* zu HTTPS umgeleitet.

- **Passwort-Hashing** Auch wenn sich ein Angreifer Zugriff zur Datenbank der Plattform verschafft, darf es ihm nicht möglich sein, die Passwörter zu den Benutzerkonten zu erlangen. Statt in Klartext werden Passwörter als Hashwerte in der Datenbank gespeichert. Hierfür wird in ASP.NET Core Identity als Hashfunktion die Password-based Key Derivation Function 2 (PBKDF2) [62] mit 10.000 Iterationen verwendet. Der resultierende Hashwert kann unter realen Bedingungen auch mit großer Rechenleistung erst nach Jahren via *Brute Force* in das ursprüngliche Passwort überführt werden.

- **Anti-Request-Forgery** Unter *Cross-Site-Request-Forgery* (CSRF) [63] versteht man Anfragen an eine Webseite, die nicht vom Benutzer, sondern einer schädlichen Webseite im Browser des Benutzers kommen. Bei CSRF nutzen Angreifer aus, dass ein Benutzer bereits bei der angegriffenen Webseite angemeldet ist und verwenden dessen Sitzung, um im Namen des Nutzers Aktionen auszuführen. Diese Plattform verwendet bei allen HTTP POST *Actions* ein *Anti-Forgery Token*. Alle ausfüllbaren Formulare werden mit einem zufälligen, versteckten Feld versehen, das nur dem Browser und der Plattform bekannt ist. Dieses lässt sich nicht auslesen und Anfragen, die ein falsches Token verwenden, können nicht vom Nutzer selbst stammen.

- **SQL Injection** *SQL Injection* bezeichnet das ungewollte Ausführen von SQL-Queries durch einen Angreifer, meist verursacht durch fehlende Überprüfung von Eingaben in Formularen oder GET-Parametern. Sämtliche Benutzereingaben werden in zugehörigen Models typisiert. Eingaben werden nicht direkt in eine Datenbankabfrage, sondern zunächst an Manager-Klassen übergeben, welche *Language-Integrated Queries (LINQ)* [64] verwenden und solche Eingaben automatisch parametrisieren.

4.7 Layout

Das Layout der Plattform basiert auf Bootstrap [65]. Views werden von der in ASP.NET Core enthaltenen Razor View Engine [66] gerendert.

5 Ergebnisse

Im Rahmen dieser Arbeit wurde eine Plattform erstellt, welche mehrere Authentifizierungsmethoden vereint und grundlegende Autorisierung bereitstellt. Diese Plattform ist erweiterbar und kann sowohl als Vorlage für neue Projekte als auch als Erweiterung bestehender Projekte verwendet werden. Die Plattform wurde mit verschiedenen Browsern und Endgeräten getestet.

5.1 Aufbau

Die Plattform läuft auf einem Arbeitsplatzrechner. Sie nimmt HTTPS-Anfragen auf Port 443 entgegen und leitet HTTP Anfragen auf Port 80 dorthin um. Sie wird unter Windows 10 auf einem IIS 10.0 Express-Webserver aus Visual Studio 2017 Enterprise heraus betrieben. Mit Ausnahme des Sustainsys.Saml2 Pakets werden alle Pakete zur Kompilierzeit von NuGet [67] aktualisiert oder neu heruntergeladen.

5.2 Testbetrieb

Die Plattform wurde auf einem BW-PC 5 getestet, ausgestattet mit Intel Core i7 6700 Prozessor, 16 GB RAM und 256 GB SSD- sowie 1 TB HDD-Speicher. Als Betriebssystem kommt Windows 10 zum Einsatz. Die Plattform wird von Microsoft Edge, Google Chrome und Mozilla Firefox in aktuellen Versionen korrekt dargestellt.

5.3 Präsentation

Für eine korrekte Darstellung in allen Browsern liefert die Plattform ausschließlich validen HTML5-Code aus. Dies wurde am 27. November 2017 mit dem Validator des W3C [68] überprüft.

5.3.1 Frontend

Über das Frontend sind die Registrierungs- und Loginfunktionalitäten erreichbar. Eingeloggte Benutzer können außerdem ihr Profil sowie ihre Einstellungen einsehen und Änderungen daran vornehmen. Dazu gehören Passwortänderung, Aktivierung der 2FA, Änderung der E-Mailadresse und Hinzufügen weiterer Authentifizierungsmethoden für dieses Konto.

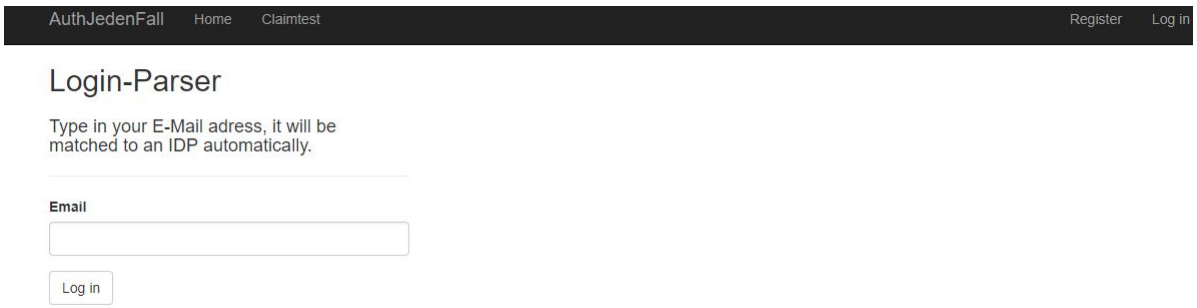

Abbildung 9: Frontend der Plattform für einen Loginvorgang

5.3.2 Backend

Benutzer mit einem *admin*-Claim können die Administrationsoberfläche verwenden. Sie erlaubt das Hinzufügen neuer Nutzer sowie Editieren, Einsehen und Löschen bestehender Nutzer.

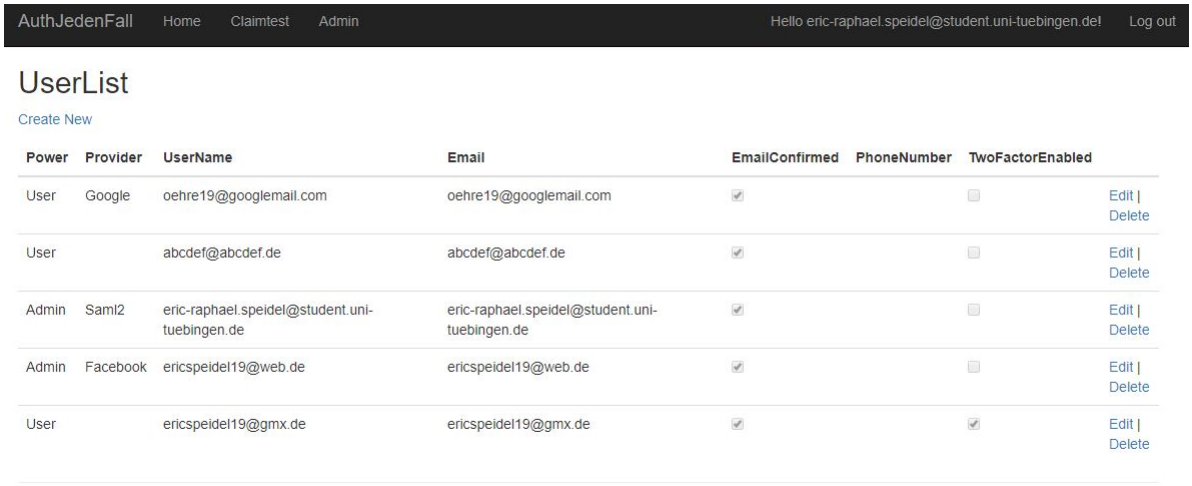

Abbildung 10: Backend der Plattform für die Anzeige aller Benutzer

5.4 Kommunikation mit SAML2- und OpenID Connect-IDP

Um Kompatibilität mit SAML2- und OpenID Connect-IDPs zu gewährleisten, muss die Plattform den Standard korrekt implementieren. Mithilfe von Browser-Logging wurden exemplarisch SAML2-SP-Metadaten, SAML2-Anfragen an den Test-IDP sowie HTTP-Requests und -Responses zwischen der Plattform und den OpenID Connect-IDP auf Fehlerfreiheit überprüft. In allen Fällen waren Anfragen der Plattform sowie Metadaten wohlgeformt und Daten aus erhaltenen Assertions, bzw Identity Tokens wurden korrekt extrahiert und in der Datenbank gespeichert. Änderungen an den Einstellungen der Middleware wurden beim erneuten Start der Plattform in Metadaten und Anfragen übernommen.

```
<saml2p:AuthnRequest AssertionConsumerServiceURL="https://dev01.wi106.uni-
tuebingen.de/Saml2/Acs"
    Destination="https://idp-test.uni-tuebingen.de/idp/profile/SAML2/Redirect/SSO"
    ID="idff0ec8d14b064399adf52f57b578af0b" IssueInstant="2017-11-27T15:50:20Z"
Version="2.0"
    xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
    xmlns:saml2p="urn:oasis:names:tc:SAML:2.0:protocol">
    <saml2:Issuer>https://dev01.wi106.uni-tuebingen.de/Saml2</saml2:Issuer>
    <saml2p:NameIDPolicy AllowCreate="false"
        Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent"/>
</saml2p:AuthnRequest>
```

Abbildung 11: AuthnRequest der Plattform an den Test-IDP der Universität Tübingen

5.5 Performance unter Last

Um die Skalierbarkeit der Plattform zu testen, wurde die CPU-Auslastung mittels Visual Studio 2017 Leistungsprofiler analysiert. Währenddessen wurde der Server mittels Apache JMeter [69] unter Last gesetzt, wobei verschiedene Zahlen an Clients und unterschiedliche Szenarien geprüft wurden. In den ersten beiden Tests wurden 100, bzw 1000 Anfragen an eine geschützte Ressource der Plattform simuliert. JMeter verwendet hierzu *Threads*. Ein Thread beginnt, wenn die HTTP-Anfrage an den Server gesendet wird und endet nach Erhalt der HTTP-Antwort des Servers. Es wurde eine *Ramp-up-period* von einer Sekunde gewählt. Das bedeutet, dass JMeter innerhalb einer Sekunde 100, bzw 1000 Threads eröffnet. Dies geschieht linear, es wird also pro 10 Millisekunden, respektive 1 Millisekunde, ein Thread eröffnet. Für die Testdauer von 30 Sekunden wurde eine kontinuierliche Belastung von 100, bzw 1000 Threads simuliert. Jeder Thread wurde nach Erhalt der HTTP-Antwort neu gestartet, sodass für die Testdauer zu jedem Zeitpunkt 100, bzw 1000 Threads parallel aktiv waren. Im letzten Test wurden fehlgeschlagene Loginversuche mit 1000 kontinuierlichen Threads simuliert, da diese eine Validierung des Anti-Forgery-Tokens beinhalten, welche zu einer höheren Auslastung führen könnte.

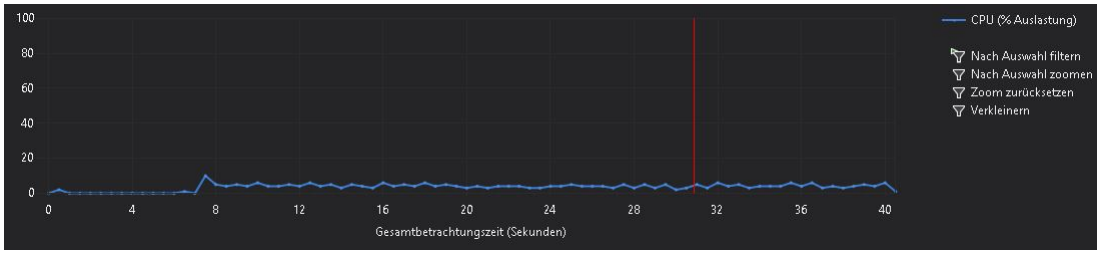

Abbildung 12: CPU-Auslastung bei kontinuierlichen Anfragen für eine geschützte Ressource von 100 parallelen Threads

Die ersten Sekunden des Tests zeigen die CPU-Auslastung des Servers im Leerlauf. Nach circa 8 Sekunden wurde die JMeter-Simulation gestartet, der Hochpunkt von circa 10% CPU-Auslastung entsteht beim nahezu parallelen Eintreffen von 100 Requests. Der Server baut diese in einer Warteschlange ab, und obwohl jeder beendete Thread neu gestartet wird, sinkt die CPU-Auslastung nach dem anfänglichen Hochpunkt auf circa 5% ab. Dies ist bekanntes Verhalten [1] von IIS.

[1] `https://sid-500.com/2017/04/16/iis-vs-apache-denial-of-service-angriff-testen/`

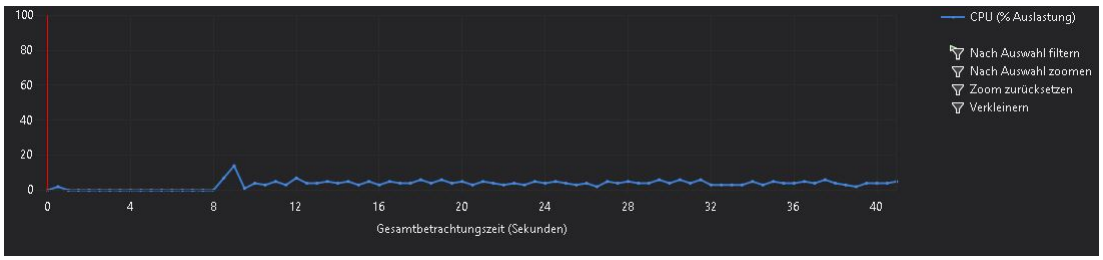

Abbildung 13: CPU-Auslastung bei kontinuierlichen Anfragen für eine geschützte Ressource von 1000 parallelen Threads

Der Test wurde mit 1000 kontinuierlichen Threads wiederholt. Es zeigte sich ähnliches Verhalten, mit einem Hochpunkt der CPU-Auslastung von circa 15% und einem anschließenden Absinken auf unter 10%. Die Plattform wird also nicht linear stärker von mehr Threads ausgelastet.

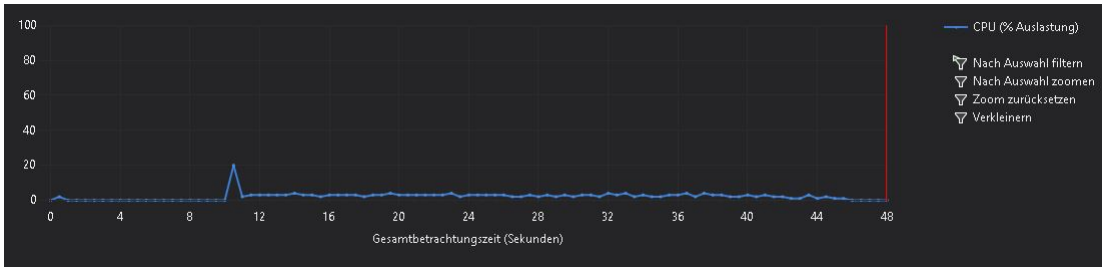

Abbildung 14: CPU-Auslastung bei kontinuierlichen Loginversuchen von 1000 parallelen Threads

Auch hier zeigt sich ein Hochpunkt der CPU-Auslastung von rund 20% beim Eintreffen der ersten 1000 Requests, danach sinkt die Auslastung stark auf weniger als 5% ab. Die stärkere initiale Auslastung ist auf die ressourcenintensive Tokenvalidierung zurückzuführen. Eine mögliche Erklärung für den starken Abfall der Auslastung ist eine Anpassung von parallel bearbeiteten Requests.
Bei einmaliger Simulation von 1000 fehlgeschlagenen Loginversuchen ohne Regeneration beendeter Threads vergingen etwa 6 Sekunden von Erstellung des ersten bis zum Ende des letzten Threads.

6 Diskussion

Zu Beginn dieser Arbeit wurde die Motivation für flexible SSO-Systeme, welche unterschiedliche Authentifizierungsmethoden anbieten, erläutert. Dem Benutzer Alternativen zur lokalen Registrierung zu bieten verkleinert Eintrittsbarrieren; rund 86% der Benutzer stören sich an einer erneuten Registrierung. [70]
Darüber hinaus wird Benutzern erlaubt, ihre Accountsicherheit durch 2FA zu erhöhen oder einen IDP zu wählen, bei welchem sie ein Konto mit einem starken Passwort oder zusätzlichen Sicherheitsmaßnahmen besitzen. Die Plattform wurde konzipiert, um von zukünftigen Webanwendungen als Vorlage benutzt oder in bestehende Arbeiten integriert zu werden. Aus diesem Grund wurde auf moderne Technologien wie ASP.NET Core 2.0 und möglichst wenige Abhängigkeiten von Drittanbieterpaketen innerhalb der Plattform gesetzt. Da Authentifizierung und Autorisierung den sicheren Umgang mit sensiblen Daten erfordern wurden sicherheitskritische Aspekte wie Passwortsicherheit, Verschlüsselung, Session Riding und SQL-Injection betrachtet und entsprechende Gegenmaßnahmen ergriffen, wie in 4.6 beschrieben. Vorläufige Tests implizieren eine gute Skalierbarkeit der Plattform, womit sie sich zur Anwendung in Services mit großen Benutzerzahlen eignet.

6.1 ASP.NET Core

Die Plattform wurde in ASP.NET Core Version 2.0, welche am 14. August 2017 erschien, erstellt. Dieses Update brachte weitere Funktionalitäten des vollen .NET Frameworks in .NET Core ein, welcher nun über 32.000 Bibliotheken umfasst und Referenzierung beliebiger .NET Framework-Bibliotheken ermöglicht. Ein zentraler Aspekt und Vorteil von .NET Core ist die Multi-Plattform-Unterstützung, weshalb die Wahl des Frameworks zur Entwicklung dieser Plattform auf ASP.NET Core fiel. Einige Services der Universität Tübingen, etwa die Livestreaming-Plattform timmscast [71] verwenden ASP.NET, was die Integration der Authentifizierungsplattform vereinfacht. Der .NET Core, welcher zuerst im Juni 2016 veröffentlicht wurde, ist ein verhältnismäßig junges Framework, was sich vor allem durch unzureichende Dokumentation im Internet und gelegentlich abweichendes Verhalten portierter Bibliotheken von deren Original im .NET Framework bemerkbar macht. So musste beispielsweise im Rahmen dieser Arbeit eine Klasse des ASP.NET Core, der *ReferencesMetadataReferenceFeatureProvider* verändert werden, da sie einen bekannten Fehler enthält, welcher in Version 2.1 des ASP.NET Core behoben werden soll.
Dies sind mögliche Erklärungen, warum der Marktanteil von ASP.NET auch nach Veröffentlichung der Multi-Plattform-Variante weiter zu sinken scheint. [72] Ebenfalls zu sehen ist, dass ASP.NET vor allem auf Webseiten mit hohen Besucherzahlen zum Einsatz kommt. Tatsächlich erzielt der .NET Core sehr gute Leistungen in verschiedenen Benchmarks [73] und es existieren Fallstudien [74], welche eine hohe Performance von ASP.NET Core sowie C# .NET Core beschreiben. Der Belastungstest 5.5 dieser Plattform indiziert ebenfalls eine gute Skalierbarkeit. Die umfangreichen ASP.NET Core-Bibliotheken ermöglichen durch viele mitgelieferte Middlewares und Funktionen eine rapide Entwicklung einer Webanwendung von Beginn an. Hohe Flexibilität und Performance von ASP.NET Core machen das Framework zu einer geeigneten Grundlage für diese Plattform.

6.2 OpenID Connect vs. SAML2

Mit der Veröffentlichung des OpenID Connect 1.0 Standards wurden einige Funktionalitäten, die OAuth2 gegenüber SAML2 fehlten, wie etwa das Identity Token als Pendant zur Assertion, eingeführt. Damit eignet sich der Standard auch über Anwendungsfälle in Social Media hinaus. Die OpenID Foundation erklärt, [75] dass OpenID Connect dieselben Anwendungsfälle wie SAML2 bedienen kann. Des Weiteren unterstützt der OpenID Connect-Standard auch die Authentifizierung in Smartphone-Applikationen, während SAML2-Implementierungen in der Regel einen Webbrowser als Client benötigen. In SAML2 sind auch andere Profile, insbesondere das *Enhanced Proxy or Client* (ECP) Profil, für die Verwendung von SAML2 außerhalb eines Webbrowsers, spezifiziert. Dieses findet jedoch laut Shibboleth Consortium [76] kaum Anwendung, vor allem im Vergleich mit OpenID Connect, welches auch in mobilen Applikationen benutzt wird.
SAML2 ist weiter verbreitet und der de facto Standard im Enterprise- sowie im öffentlichen Sektor [77], [33]. Der hohe Sicherheitsstandard von SAML2 und das Fehlen eines Föderationskonzepts in OpenID Connect sind mögliche Gründe dafür. Beide Standards finden große Anwendung, weshalb diese Plattform Unterstützung für SAML2 und OpenID Connect bietet. Zum aktuellen Zeitpunkt existieren für beide Standards keine praktikablen Implementierungen eines Single Logouts, weshalb auch diese Plattform die Funktionalität nicht anbieten kann.

6.3 Ausblick

Im Rahmen dieser Arbeit wurde erfolgreich das Konzept eines SSO-Systems mit unterschiedlichen Authentifizierungsmethoden umgesetzt. Plattformunabhängigkeit, Sicherheit und gute Performance sind Besonderheiten dieses Systems, welche eine Weiterentwicklung außerhalb dieser Arbeit interessant machen. Vorstellbar wären einige Aspekte, welche im Rahmen der Arbeit nicht möglich waren.

6.3.1 Nutzertest

Der Schwerpunkt dieser Arbeit lag auf der Bereitstellung von Funktionalitäten. Ein Benutzertest mit Personen unterschiedlicher IT-Affinität könnte Verbesserungen der Oberfläche und Navigation ermöglichen. Insbesondere die automatische Auswahl des Loginverfahrens kann als neues Konzept gegenüber einer manuellen Auswahl auf Vor- und Nachteile überprüft werden.

6.3.2 Automated Deployment

Die Plattform kann angepasst werden, um als Docker-Container ausgeliefert zu werden. Dies erleichtert die Integration in neue und bestehende Projekte. Neue Versionen der Plattform können via *Automated Deployment* in Services eingespielt werden.

7 Anhang

A: Konfiguration von Visual Studio

Das Projekt wurde mit Microsoft Visual Studio Enterprise 2017 entwickelt und nutzt einige Funktionen, die auch in der kostenfreien Express Variante, aber nicht in anderen Editoren zu finden sind. Es handelt sich um die Verwendung des integrierten IIS 10 Express, der integrierten MSSQL LocalDb und des Secret-Managers zur Verwahrung wichtiger, geheimer Daten wie SMTP Zugangsdaten und OAuth Client Secrets.

A1: Webserver

Bis auf die verwendete Sustainsys.Saml2 Bibliothek ist die Plattform vollständig mit ASP.NET Core 2.0 kompatibel. Sustainsys.Saml2 benötigt noch wenige Funktionen aus dem vollen .NET Framework, eine vollständige Kompatibilität ist aber geplant und dürfte nicht mehr lange ausstehen. Für die Plattform hat dies die Konsequenz, dass sie in der aktuellen Version mit SAML2-Funktionalität nur auf Windows Servern nativ einsetzbar ist. Es existieren Projekte wie Mono [78], welche ASP.NET Framework auch auf anderen Systemen und Webservern ermöglichen.
Ohne die SAML2-Komponente, bzw nach vollständiger Kompatibilität mit ASP.NET Core der Sustainsys.Saml2 Bibliothek ist die Plattform auf allen von .NET Core unterstützten Systemen verwendbar. Für das Hosting mit unterschiedlichen Server- und Betriebssystemen stellt Microsoft einige Anleitungen bereit [79].

A2: Datenbank

Zur Entwicklung der Plattform wurde die in Visual Studio integrierte MSSQL LocalDb verwendet, welche für Testzwecke und Development sehr gut geeignet ist. Für eine Produktionsumgebung ist diese jedoch nicht sehr tauglich. Es können verschiedene Datenbanktypen und -server verwendet werden. Ein vollwertiger Microsoft SQL Server lässt sich ohne zusätzlichen Konfigurationsaufwand verwenden, auch die Verwendung von MySQL ist möglich. Selbst nichtrelationale Datenbanken wie MongoDB können benutzt werden, dies erfordert jedoch zusätzliche Konfiguration.
In jedem Fall muss die Verbindung zur neuen Datenbank hergestellt werden. Das kann entweder in Visual Studio über den SQL Server Explorer geschehen, oder durch Bearbeiten der `ConnectionStrings` in der Datei `appsettings.json`. Für die verwendete LocalDb sieht dies beispielsweise wie folgt aus:

```
"ConnectionStrings": {
"DefaultConnection": "Server=(localdb)\\mssqllocaldb;
Database=aspnet-OAuth2Test-E7030EB0-A078-4560-8756-C33EC3BD5173;
Trusted_Connection=True;MultipleActiveResultSets=true"
}
```

A3: Secret-Manager

Der in Visual Studio 2017 enthaltene Secret-Manager wurde zum Schutz empfindlicher Zugangsdaten verwendet. Die folgenden Secrets müssen in Visual Studio eingetragen oder als Klartext in der verwendenden Klasse deklariert werden:

```
 "Authentication:Facebook:AppId": "",
"Authentication:Facebook:AppSecret": "",
"Authentication:Google:ClientId": "",
"Authentication:Google:ClientSecret": "",
"SMTPUser": "",
"SMTPPwd": ""
```

`SMTPUser` und `SMTPPwd` beziehen sich auf Zugangsdaten zum SMTP-Server der Universität Tübingen.

B: Konfiguration für SAML2

In der Datei `startup.cs` wird das Sustainsys.Saml2-Paket verwendet, um die Authentifizierung mittels SAML2 zu ermöglichen. Änderungen an der Konfiguration sind in folgendem Codeabschnitt vorzunehmen:

```
services.AddAuthentication()
.AddSaml2(options =>
{...
...
});
```

Insbesondere sollte für eine Produktionsumgebung ein neues Zertifikat verwendet werden. Bei Benutzung mehrerer IDP kann auch ein Discovery Service implementiert werden.

C: Startup-Konfiguration

In der Datei `startup.cs` werden die wichtigsten Konfigurationen der Plattform vorgenommen. Eine neue Produktionsumgebung erfordert eventuell Anpassungen der folgenden Passagen:

MVC

Das MVC-Paket wird als erster Service konfiguriert. Dies ist die Default-Konfiguration:

```
services.Configure<MvcOptions>(options =>
{
options.Filters.Add(new RequireHttpsAttribute());
});
```

Dieser Filter erzwingt die Verwendung von HTTPS für alle Verbindungen. Weitere Filter können gesetzt werden, um Headerinformationen zu verändern. Nach der Konfiguration wird das Paket weiter unten im Code geladen. Dabei muss aufgrund eines Bugs [2] eine veränderte Klasse der MVC-Bibliothek geladen werden. Dieser Workaround wird mit ASP.NET Core 2.1 voraussichtlich obsolet.

[2] Siehe `https://github.com/dotnet/core-setup/issues/2981`

```
services.AddMvc()
ConfigureApplicationPartManager(manager =>
{
var oldMetadataReferenceFeatureProvider = manager.FeatureProviders.First(f => f is
MetadataReferenceFeatureProvider);
manager.FeatureProviders.Remove(oldMetadataReferenceFeatureProvider);
manager.FeatureProviders.Add(new ReferencesMetadataReferenceFeatureProvider());
});
```

Identity
Das Identity-Paket kann konfiguriert werden, um bestimmte Vorgaben zur Registrierung neuer Benutzer zu setzen. Dies ist die Default-Konfiguration:

```
services.AddIdentity<ApplicationUser, IdentityRole>(config =>
{
config.SignIn.RequireConfirmedEmail = true;
config.Password.RequireDigit = true;
config.Password.RequiredLength = 5;
config.Password.RequireLowercase = true;
config.Password.RequireNonAlphanumeric = false;
config.Password.RequireUppercase = false;
})
```

Es werden Mindestanforderungen für Passwortsicherheit und eine zwingende Bestätigung der E-Mailadresse vor erstmaligem Login gesetzt. Weitere Optionen erlauben eine Loginsperre nach fehlgeschlagenen Loginversuchen, Änderung von Session-Cookies oder weitere Anforderungen an neue Registrierungen.

Authentication
In dieser Sektion wird der Service `Authentication` konfiguriert. Es werden die Middlewares für Facebook und Google sowie die von Sustainsys.Saml2 initialisiert.

```
services.AddAuthentication().AddFacebook(facebookOptions =>
{
facebookOptions.AppId = Configuration["Authentication:Facebook:AppId"];
facebookOptions.AppSecret = Configuration["Authentication:Facebook:AppSecret"];
});

services.AddAuthentication().AddGoogle(googleOptions =>
{
googleOptions.ClientId = Configuration["Authentication:Google:ClientId"];
googleOptions.ClientSecret = Configuration["Authentication:Google:ClientSecret"];
});


services.AddAuthentication()
.AddSaml2(options =>
{
...
});
```

App-ID und Client-Secret werden aus dem Secret-Manager geladen. Optional können sie auch als Klartext in der Konfiguration vermerkt werden.

Authorization
Policies, welche von Controllern zur Autorisierung benutzt werden können, werden hier initialisiert.

```
services.AddAuthorization(options =>
{
options.AddPolicy("IsRegistered", policy => policy.RequireClaim("Power"));
options.AddPolicy("IsAdmin", policy => policy.RequireClaim("Power","Admin"));
});
```

URL-Rewriter
Um HTTP-Anfragen auf HTTPS umzuleiten, wird ein URL Rewriter verwendet.

```
var options = new RewriteOptions()
.AddRedirectToHttps();
app.UseRewriter(options);
```

Literatur

[1] Promotorengruppe Kommunikation der Forschungsunion Wirtschaft. *Wirtschaft und Wissenschaft*. URL: https://www.iosb.fraunhofer.de/servlet/is/21752/Umsetzungsempfehlungen%7B%5C_%7DIndustrie%7B%5C_%7D4.0%7B%5C_%7Dfinal%7B%5C_%7D2012-10-02.pdf?command=downloadContent%7B%5C&%7Dfilename=Umsetzungsempfehlungen%7B%5C_%7DIndustrie%7B%5C_%7D4.0%7B%5C_%7Dfinal%7B%5C_%7D2012-10-02.pdf (besucht am 14. 11. 2017).

[2] Insurance Information Institute. *Facts + Statistics: Identity theft and cybercrime*. URL: https://www.iii.org/fact-statistic/facts-statistics-identity-theft-and-cybercrime (besucht am 14. 11. 2017).

[3] Facebook. *Facebook*. URL: https://www.facebook.com/ (besucht am 17. 11. 2017).

[4] Google. *Google Kontoübersicht*. URL: https://myaccount.google.com/?pli=1 (besucht am 17. 11. 2017).

[5] Zentrum für Datenverarbeitung Universität Tübingen. *Netzzugang — Zentrum für Datenverarbeitung*. URL: http://www.zdv.uni-tuebingen.de/dienstleistungen/netzzugang.html (besucht am 17. 11. 2017).

[6] Facebook. *Unternehmensdaten — Facebook Newsroom*. URL: https://de.newsroom.fb.com/company-info/ (besucht am 17. 11. 2017).

[7] OAuth.io. *OAuth Integration — OAuth.io*. URL: https://oauth.io/home (besucht am 17. 11. 2017).

[8] Oneall.com. *Social Login, Sign On & Social Network Sharing — oneall.com*. URL: https://www.oneall.com/ (besucht am 17. 11. 2017).

[9] Amazon.com Inc. *Online Versandhandel — Amazon*. URL: https://www.amazon.de/ (besucht am 17. 11. 2017).

[10] Amazon.com Inc. *Login with Amazon*. URL: https://login.amazon.com/ (besucht am 17. 11. 2017).

[11] T. Dierks und E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.2*. Techn. Ber. Aug. 2008. DOI: 10.17487/rfc5246. URL: https://www.rfc-editor.org/info/rfc5246.

[12] Robyn Hicock. *Password Guidance*. 2016. URL: https://www.microsoft.com/en-us/research/publication/password-guidance/.

[13] Karen Scarfone, Wayne Jansen und Miles Tracy. „Guide to General Server Security Recommendations of the National Institute of Standards and Technology“. In: *Special Publication* (), S. 800–123. DOI: 10.6028/NIST.SP.800-123. URL: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-123.pdf.

[14] D. M'Raihi u. a. „TOTP: Time-Based One-Time Password Algorithm“. In: (2011). ISSN: 2070-1721. URL: https://tools.ietf.org/pdf/rfc6238.pdf.

[15] Twilio Inc. *Authy – Two-Factor Authentication App*. URL: https://authy.com/ (besucht am 17. 11. 2017).

[16] Yubico. *YubiKey Two-Factor-Authentication*. URL: https://www.yubico.com/ (besucht am 17. 11. 2017).

[17] Justin Richer. „Universal Second Factor and OpenID Connect“. 2015. URL: https://www.yubico.com/wp-content/uploads/2015/08/Yubico-U2F-and-OIDC-Final.pdf.

[18] Bundesministerium für Sicherheit in der Informationstechnik. *Geeignete Auswahl von Authentikationsmechanismen*. 2013. URL: https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/%7B%5C_%7Dcontent/m/m04/m04133.html (besucht am 03. 11. 2017).

[19] D. Hardt. „The OAuth 2.0 Authorization Framework“. In: (2012). URL: https://tools.ietf.org/html/rfc6749.

[20] OpenID. *OpenID Connect*. URL: https://openid.net/connect/ (besucht am 17. 11. 2017).

[21] Google. *Creating a Google API Console project and client ID — Google Developers*. URL: https://developers.google.com/identity/sign-in/web/devconsole-project (besucht am 17. 11. 2017).

[22] „RFC 7617 - The 'Basic' HTTP Authentication Scheme“. In: (2015). ISSN: 2070-1721. URL: http://www.rfc-editor.org/info/rfc7617..

[23] Organization for the Advancement of Structured Information Standards (OASIS). *Security Assertion Markup Language (SAML) V2.0 Technical Overview*. 2008. URL: http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0.html (besucht am 02. 11. 2017).

[24] Scott Cantor u. a. „Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0“. In: *OASIS Standard (15 March 2005)* March (2005), S. 1–86. DOI: 10.1.1.154.8826. URL: https://www.oasis-open.org/committees/download.php/56776/sstc-saml-core-errata-2.0-wd-07.pdf.

[25] Scott Cantor u. a. „Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0 – Errata Composite“. In: (2015). URL: https://www.oasis-open.org/committees/download.php/56779/sstc-saml-bindings-errata-2.0-wd-06.pdf.

[26] John Hughes u. a. „Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0 – Errata Composite“. In: (). URL: https://www.oasis-open.org/committees/download.php/56782/sstc-saml-profiles-errata-2.0-wd-07.pdf.

[27] Scott Cantor u. a. „Metadata for the OASIS Security Assertion Markup Language (SAML) V2.0 – Errata Composite“. In: (). URL: https://www.oasis-open.org/committees/download.php/56785/sstc-saml-metadata-errata-2.0-wd-05.pdf.

[28] Shibboleth Consortium. *Identity Provider — Shibboleth*. URL: https://www.shibboleth.net/products/identity-provider/ (besucht am 17. 11. 2017).

[29] Shibboleth. *FlowsAndConfig - Shibboleth Concepts*. URL: https://wiki.shibboleth.net/confluence/display/CONCEPT/FlowsAndConfig (besucht am 03. 11. 2017).

[30] D Cooper u. a. „RFC 5280 - Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List Profile“. In: (2008). URL: https://tools.ietf.org/pdf/rfc5280.pdf.

[31] R L Rivest, A Shamir und L Adleman. „A Method for Obtaining Digital Signatures and Public-Key Cryptosystems“. In: (). URL: http://people.csail.mit.edu/rivest/Rsapaper.pdf.

[32] W3C. *XML Encryption Syntax and Processing Version 1.1*. URL: https://www.w3.org/TR/xmlenc-core1/ (besucht am 24. 11. 2017).

[33] Shibboleth Consortium. *Members*. URL: https://www.shibboleth.net/consortium/ (besucht am 24. 11. 2017).

[34] OpenID Foundation. *Sponsoring Members*. URL: http://openid.net/foundation/sponsoring-members/ (besucht am 24. 11. 2017).

[35] Amazon.com Inc. *OpenID Connect Support for Amazon Cognito — AWS News Blog*. URL: https://aws.amazon.com/de/blogs/aws/openid-connect-support/ (besucht am 17. 11. 2017).

[36] Google. *OpenID Connect — Google Identity Platform*. URL: https://developers.google.com/identity/protocols/OpenIDConnect (besucht am 17. 11. 2017).

[37] Microsoft. *Grundlegendes zum OpenID Connect-Authentifizierungscodefluss in Azure AD — Microsoft Docs*. URL: https://docs.microsoft.com/de-de/azure/active-directory/develop/active-directory-protocols-openid-connect-code (besucht am 17. 11. 2017).

[38] Shibboleth Consortium. *SLOIssues - Shibboleth Concepts - Shibboleth Wiki*. URL: https://wiki.shibboleth.net/confluence/display/CONCEPT/SLOIssues (besucht am 24. 11. 2017).

[39] M. Jones. *OpenID Connect Front-Channel Logout 1.0 - draft 02*. URL: http://openid.net/specs/openid-connect-frontchannel-1%7B%5C_%7D0.html (besucht am 24. 11. 2017).

[40] M. Jones und J. Bradley. *OpenID Connect Back-Channel Logout 1.0 - draft 04*. URL: http://openid.net/specs/openid-connect-backchannel-1%7B%5C_%7D0.html (besucht am 24. 11. 2017).

[41] M. Jones u. a. *OpenID Connect Session Management 1.0 - draft 28*. URL: https://openid.net/specs/openid-connect-session-1%7B%5C_%7D0.html (besucht am 24. 11. 2017).

[42] Dominick Baier u. a. „A Guide to Claims-Based Identity and Access Control: Authentication and Authorization for Services and the Web“. In: (2011), S. 374. URL: https://www.microsoft.com/en-us/download/details.aspx?id=28362.

[43] Microsoft. *Einführung in ASP.NET Core*. URL: https://docs.microsoft.com/de-de/aspnet/core/ (besucht am 06. 11. 2017).

[44] Microsoft. *.NET Core*. URL: https://www.microsoft.com/net/ (besucht am 19. 11. 2017).

[45] Microsoft und Diverse. *ASP.NET Core Identity — Github*. URL: https://github.com/aspnet/Identity (besucht am 17. 11. 2017).

[46] Sustainsys, Anders Abel und Diverse. *Sustainsys.Saml2 — GitHub*. URL: https://github.com/Sustainsys/Saml2 (besucht am 17. 11. 2017).

[47] Microsoft. *The Official Microsoft IIS Site*. URL: https://www.iis.net/ (besucht am 17. 11. 2017).

[48] Microsoft. *Microsoft-Datenplattform*. URL: https://www.microsoft.com/de-de/sql-server/ (besucht am 17. 11. 2017).

[49] Oracle Corporation. *MySQL*. URL: https://www.mysql.com/de/ (besucht am 17. 11. 2017).

[50] The PostgreSQL Global Development Group. *PostgreSQL Open source database*. URL: https://www.postgresql.org/?%7B%5C&%7D (besucht am 17. 11. 2017).

[51] Microsoft. *.NET Standard — Microsoft Docs*. URL: https://docs.microsoft.com/de-de/dotnet/standard/net-standard (besucht am 17. 11. 2017).

[52] Microsoft. *Kestrel Webserverimplementierung in ASP.NET Core — Microsoft Docs*. URL: https://docs.microsoft.com/de-de/aspnet/core/fundamentals/servers/kestrel?tabs=aspnetcore2x (besucht am 17. 11. 2017).

[53] The Apache Software Foundation. *The Apache HTTP Server Project*. URL: https://httpd.apache.org/ (besucht am 17. 11. 2017).

[54] nginx Inc. *nginx Server*. URL: https://nginx.org/en/ (besucht am 17. 11. 2017).

[55] Deutsches Forschungsnetz e.V. *DFN-AAI*. URL: https://www.aai.dfn.de/ (besucht am 17. 11. 2017).

[56] Microsoft. *Overview of ASP.NET Core MVC — Microsoft Docs*. URL: https://docs.microsoft.com/en-us/aspnet/core/mvc/overview (besucht am 24. 11. 2017).

[57] Microsoft und Diverse. *ASP.NET Core MVC*. URL: https://github.com/aspnet/Mvc (besucht am 24. 11. 2017).

[58] Google und Diverse. *Google Authenticator - GitHub*. URL: https://github.com/google/google-authenticator (besucht am 24. 11. 2017).

[59] Google. *GMail - Free E-Mail*. URL: https://mail.google.com (besucht am 24. 11. 2017).

[60] Wikipedia. *CRUD*. URL: https://de.wikipedia.org/wiki/CRUD (besucht am 26. 11. 2017).

[61] E. Rescorla. „RFC 2818 - HTTP Over TLS“. In: (2000). URL: https://tools.ietf.org/pdf/rfc2818.pdf.

[62] B Kaliski. „RFC 2898 - PKCS #5: Password-Based Cryptography Specification Version 2.0“. In: (2000). URL: https://tools.ietf.org/pdf/rfc2898.pdf.

[63] Wikipedia. *Cross-Site-Request-Forgery*. URL: https://de.wikipedia.org/wiki/Cross-Site-Request-Forgery (besucht am 24. 11. 2017).

[64] Microsoft. *Language-Integrated Query (LINQ)*. URL: https://docs.microsoft.com/de-de/dotnet/csharp/programming-guide/concepts/linq/ (besucht am 27. 11. 2017).

[65] Twitter und Diverse. *Bootstrap · The most popular HTML, CSS, and JS library in the world*. URL: https://getbootstrap.com/ (besucht am 25. 11. 2017).

[66] Microsoft und Diverse. *Razor - GitHub*. URL: https://github.com/aspnet/Razor (besucht am 25. 11. 2017).

[67] Microsoft. *NuGet Gallery — Home*. URL: https://www.nuget.org/ (besucht am 25. 11. 2017).

[68] W3C. *Nu Html Checker*. URL: https://validator.w3.org/nu/ (besucht am 27. 11. 2017).

[69] Apache. *Apache JMeter*. URL: http://jmeter.apache.org/ (besucht am 27. 11. 2017).

[70] Janrain und Blue. „The Value of Social Login“. In: (). URL: http://www1.janrain.com/rs/janrain/images/Industry-Research-Value-of-Social-Login-2013.pdf.

[71] Universität Tübingen. *timmscast Home*. URL: http://timmscast.uni-tuebingen.de/ (besucht am 28. 11. 2017).

[72] Q-Success. *Usage Statistics and Market Share of ASP.NET for Websites, November 2017*. URL: https://w3techs.com/technologies/details/pl-aspnet/all/all (besucht am 27. 11. 2017).

[73] Computer Language Benchmarks Game. *C# .NET Core vs PHP*. URL: http://benchmarksgame.alioth.debian.org/u64q/compare.php?lang=csharpcore%7B%5C&%7Dlang2=php (besucht am 28. 11. 2017).

[74] Microsoft. *App performance monitoring provider increases throughput by 2,000 percent with .NET Core*. URL: https://customers.microsoft.com/de-de/story/raygun (besucht am 28. 11. 2017).

[75] OpenID. *OpenID Connect FAQ and Q&As — OpenID*. URL: http://openid.net/connect/faq/ (besucht am 17. 11. 2017).

[76] Shibboleth Consortium. *ECP - Shibboleth Concepts - Shibboleth Wiki*. URL: https://wiki.shibboleth.net/confluence/display/CONCEPT/ECP (besucht am 29. 11. 2017).

[77] Onelogin. *97% of SaaS Vendors Backing SAML-based Single Sign-on*. URL: https://www.onelogin.com/press-center/press-releases/97-percent-of-saas-vendors-backing-saml-based-single-sign-on (besucht am 29. 11. 2017).

[78] Mono Project. *ASP.NET — Mono*. URL: http://www.mono-project.com/docs/web/aspnet/ (besucht am 17. 11. 2017).

[79] Microsoft. *Übersicht über Hosting und Bereitstellung – ASP.NET Core — Microsoft Docs*. URL: https://docs.microsoft.com/de-de/aspnet/core/publishing/?tabs=aspnetcore2x (besucht am 17. 11. 2017).

Hier CD einkleben...