HOCHSCHULE HEILBRONN

Bachelorarbeit

Software Engineering SPO4

PhotoPrism: Transformation zur Mehrbenutzer-Applikation und Integration von OpenID Connect

Timo Volkmann

199267

Prüfer
Prof. Dr.-Ing. Andreas Mayer

Betreuerin
Theresa Gresch

13. Januar 2022

Selbstständigkeitserklärung

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendete Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Stuttgart, den 13. Januar 2022 .....

Inhaltsverzeichnis

1 Einleitung

PhotoPrism ist eine aktiv entwickelte Open-Source Lösung für die Verwaltung von Bildsammlungen. Der Fokus für den Einsatz dieser Software liegt bisher bei Privatanwendern, die eine Alternative zu cloudbasierten Diensten wie zum Beispiel Google Photos oder iCloud gesucht haben. PhotoPrism bietet viele Funktionen, die auch bei den großen Anbietern zu finden sind und ermöglicht den Nutzern gleichzeitig, die Hoheit über ihre Daten zu behalten.

Mit dieser Arbeit soll das Projekt bei der Transformation in eine Mehrbenutzer-Applikation unterstützt und damit ein Beitrag zu quelloffener und freier Software geleistet werden.

1.1 Hintergrund

PhotoPrism ist eine modernes Photomanagement-System, das für den Serverbetrieb im Heimnetzwerk ausgelegt ist. Es wird von Michael Mayer und Theresa Gresch als Gründer des Projekts seit 2018 öffentlich auf GitHub entwickelt und kann mit über 18.000 Sternen und fast 1.000 Forks dort einen beachtlichen Bekanntheitsgrad vorweisen. Beide arbeiten inzwischen in Vollzeit an diesem Projekt.

Zu den Schlüsselfunktionen gehören unter anderem:

- Verwalten und Durchsuchen einer Bild- und Videosammlung
- Verschiedene Ansichten und Möglichkeiten die Sammlung zu organisieren: Alben, Personen (Gesichter), Labels (Kategorien), Standorte, Favoriten, Privat- und Archivbereich (siehe Abbildung 1)
- Automatische Verschlagwortung/Klassifizierung von Bildern
- Gesichtserkennung (Face detection and face recognition via Tensorflow)
- Unterstützung einer großen Anzahl an Bildformaten (inklusive des RAW-Formats)
- Webbasierte Benutzeroberfläche (SPA/PWA)
- Duplikaterkennung
- WebDAV-Server, um den Fernzugriff auf die Original-Daten zu erleichtern
- Bilder mit Location-Metadaten auf einer Weltkarte darstellen
- und mehr... [25]

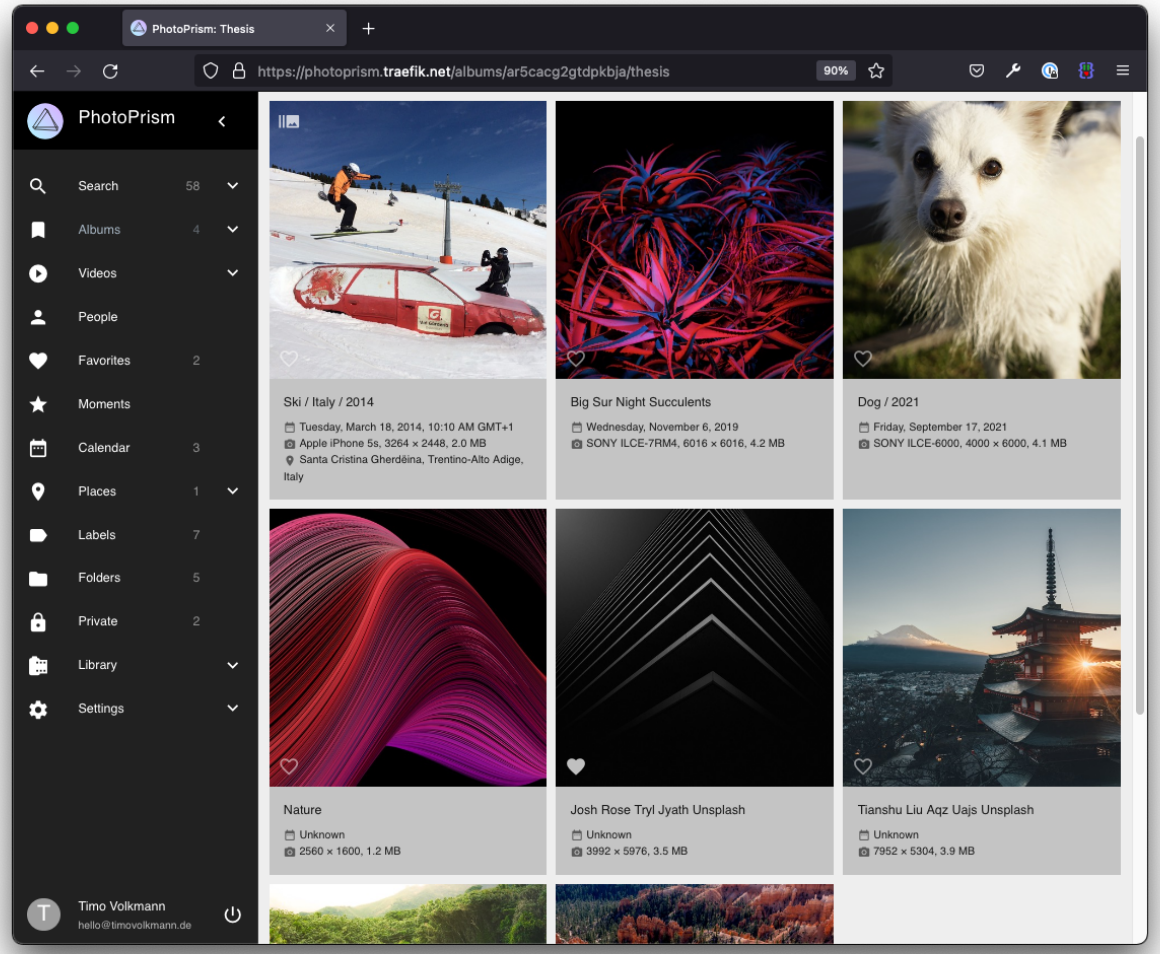

Abbildung 1: Gitter-Ansicht der Inhalte eines Albums in PhotoPrism.

Vielfach von der Community gewünscht [12] und um PhotoPrism damit für eine größere Gruppe Interessenten nutzbar zu machen, soll die Anwendung für den Mehrbenutzer-Einsatz erweitert werden. Weiter soll eine Möglichkeit geschaffen werden, das User- bzw. Identitätsmanagement und die Authentifizierung an einen externen Dienst auszulagern. Dies legt den Grundstein für eine zukünftige, engere Integration mit anderen Anwendungen und ermöglicht erhöhten Anforderungen an die Sicherheit gerecht zu werden, die von PhotoPrism selbst nicht adressiert werden können oder sollen. Hier sei zum Beispiel der Betrieb in öffentlichen Netzwerken genannt, welcher bei PhotoPrism bisher nicht im Fokus steht.

Gründe für diese Nachfrage sind vielfältig, unter anderem:

- Gemeinsame Verwendung mit Freunden und Familie

- Einschränken der Sichbarkeit von Teilen der eigenen Bildsammlung für Dritte
- Einschränken von Funktionalität wie z.B. das Deaktivieren von Bearbeiten oder Löschen für Dritte
- Gemeinsame Verwendung im Business-Umfeld, z.B. Fotografen oder Agenturen

Kurzfristig wird PhotoPrism daher zunächst um Usermanagement und Rollensystem im Kontext des privaten Einsatzes im Heimnetzwerk erweitert werden. Langfristig soll es dann für jeden Benutzer auch möglich sein, seine eigene private Sammlung an einer Instanz zu unterhalten und Inhalte mit anderen Benutzern der Instanz zusammen zu verwalten, mit der Öffentlichkeit zu teilen oder nur für sich selbst sichtbar zu machen. Darüber hinaus soll PhotoPrism in Zukunft auch in öffentlichen Netzwerken sicher eingesetzt werden können, womit dann auch ein Vernetzen von mehreren Instanzen untereinander, ein dezentrales „kollaboratives peer-to-peer sharing“ ermöglichen können, wofür ein zentralisiertes externes Usermanagement auch von Vorteil sein kann.

Weitere Informationen, Quellcode und die Dokumentation zur Anwendung sind auf der Homepage des Projektes [24] bzw. auf der GitHub-Seite des Projektes [13] zu finden.

1.2 Scope und Zielsetzung

Diese Arbeit soll eine fundierte Ausgangsbasis für den Einsatz als Mehrbenutzer-Anwendung schaffen. Zunächst werden Anforderungen aus den Vorstellungen der Maintainer und Wünschen der Benutzer formuliert, woraus dann unter Einbeziehung des aktuellen Zustands von PhotoPrism ein Konzept für eine erste Ausbaustufe erarbeitet und implementiert wird. Um einem zeitlich begrenzten Rahmen gerecht zu werden, soll der Fokus der Arbeit primär, aber nicht ausschließlich auf der Integration von OpenID Connect liegen. Dabei werden unter Berücksichtigung von Anforderungen und Sicherheitsaspekten auch Empfehlungen für weitere Schritte mit einfließen, welche nicht mehr im Rahmen der Arbeit umgesetzt werden können, aber für die Erfüllung langfristiger Ziele von hoher Bedeutung sind.

Da Authentifizierung und Autorisierung sicherheitskritische Funktionen sind, müssen Risiken hier mit besonderer Aufmerksamkeit abgewogen und enstprechend passende Maßnahmen bei der Planung und Umsetzung getroffen werden. Sicherheitsstandards, -Richtlinien und Erkenntnisse aus Sicherheitsprojekten, wie z.B. dem *OWASP* helfen hier, fundierte Entscheidungen zu treffen und einen Überblick zu erhalten was nötig ist, um eine sichere Anwendung für den zukünftigen Betrieb in öffentlichen Netzwerken anbieten zu können.

Zusammengefasst ergeben sich also für die Arbeit folgende Ziele:

1. Formulierung und Priorisierung der Anforderungen
2. Erfassung des Ist-Zustands relevanter Teile der Anwendung
3. Implementierung einer ersten Ausbaustufe für OpenID Connect
4. Empfehlungen für das Vorgehen bei der Entwicklung weiterer Ausbaustufen

Der ursprüngliche Fokus auf die Implementierung feingranularer Autorisierung wurde zugunsten der Integration von OpenID Connect aufgegeben. So kann den aufeinander aufbauenden Mehrbenutzer-Funktionalitäten, innerhalb dieser zeitlich begrenzten Arbeit, besser Rechnung getragen werden.

1.3 Methodik und Organisation

Mit dieser Arbeit soll ein Einblick in die Entwicklungsarbeit eines Open Source Projektes gegeben werden, das rein durch das Team und die Unterstützung einer freiwilligen Community vorangetrieben wird und dabei nicht durch Firmen oder Inverstoren unterstützt wird.

Die Arbeit an einem solchen Projekt erfordert zum Teil andere Herangehens- und Denkweisen wie diejenigen, die typischerweise in kommerziellen Projekten zum Einsatz kommen. Insbesondere gibt es hier weniger feste Prozesse bezüglich der Organisation des Projektes. Bei der Entwicklung wird vieles den aktuellen Umständen entsprechend geregelt. Prozesse und Methoden werden nur nach Bedarf eingeführt und es wird versucht die Entwicklung nicht durch unnötige Aufgaben zu behindern. Dies funktioniert im Open Source Kontext deutlich besser als im kommerziellen Umfeld, weil die meisten Mitwirkenden sich durch höhere intrinsische Motivation und höhere Eigenverantwortlichkeit bemerkbar machen. [33, S. 27] Auch sind bei PhotoPrism weitere Parallelen zu den Leitlinien von Raymond feststellbar, die er bei der beispielhaften Untersuchung des Open Source Projekts `fetchmail` in *Die Kathedrale und der Basar* entdeckte. So gehören zum Beispiel auch die Leitsätze *„6. Wenn du deine Benutzer als Mitprogrammierer betrachtest, ist dies der einfachste Weg zu schneller Verbesserung und effizientem Debugging.“* und *„7. Früh freigeben. Oft freigeben. Seinen Anwendern zuhören.“* [33, S. 8] bei PhotoPrism zur Philosophie des Projekts.

Bei der Entwicklung wird nach dem *Bottum-Up*-Ansatz gearbeitet, was den großen Vorteil bringt, dass bei der Programmierung noch kein endgültiges und bis ins Detail geplantes Ziel für das Endprodukt vorhanden sein muss. [31]

Die Organisation und Koordination des Projektes findet hauptsächlich auf GitHub statt, wo auch der Quellcode verwaltet wird. Hier können Entwicklungsthemen und Feedback über Issues diskutiert werden und Beiträge Dritter durch Pull Requests in den aktuellen Entwicklungsstand aufgenommen werden. Für diese Arbeit wird darüber hinaus ein eigener Bereich im dezentralen Kommunikationsnetz von *Matrix* [1] genutzt, für schnellen

und unkomplizierten Austausch während der Entwicklung an der Anwendung. *Matrix* wird auch genutzt, um einen Chat-Raum für die Community bereit zu stellen [30]. Alle Schritte, die im Rahmen der Arbeit durchgeführt werden, werden einmal pro Woche mit dem Team abgestimmt. Weiter ist zu beachten, dass die im Rahmen der Arbeit anfallenden Tests arbeitsteilig organisiert sind. Damit sind Tests die das Frontend betreffen, also auch Akzeptanztests, nicht Teil der Arbeit.

1.3.1 Struktur der Arbeit

Im folgenden Kapitel werden zunächst die Grundlagen und Technologien erörtert, die zur Umsetzung notwendig sind. Im dritten Kapitel werden die Anforderungen zusammengetragen. Das vierte Kapitel beschäftigt sich mit dem Ist-Zustand von PhotoPrism und begleitet die Umsetzung. Für das weitere Vorgehen welches über den Rahmen der Arbeit hinausgeht, werden dann im letzten Kapitel Empfehlungen gesammelt und die Ergebnisse der Arbeit nochmal kurz zusammengefasst.

Werte, die wörtlich genommen werden müssen oder Code referenzieren, werden im Text durch `Schrift mit fester Laufweite` gekennzeichnet. *Kursiv Gedrucktes* weist auf fachspezifische Formulierungen hin, die beispielsweise in Grundlagen und Spezifikationen definiert sein können oder als allgemein übliche Fachbegriffe bzw. Eigennamen gelten, die so an manchen Stellen zur besseren Verständlichkeit hervorgehoben werden. Codeausschnitte werden durch *GitHub*-Permalinks referenziert, um den Lesefluss nicht zu stören.

2 Grundlagen

2.1 Architektur und genutzte Technologien

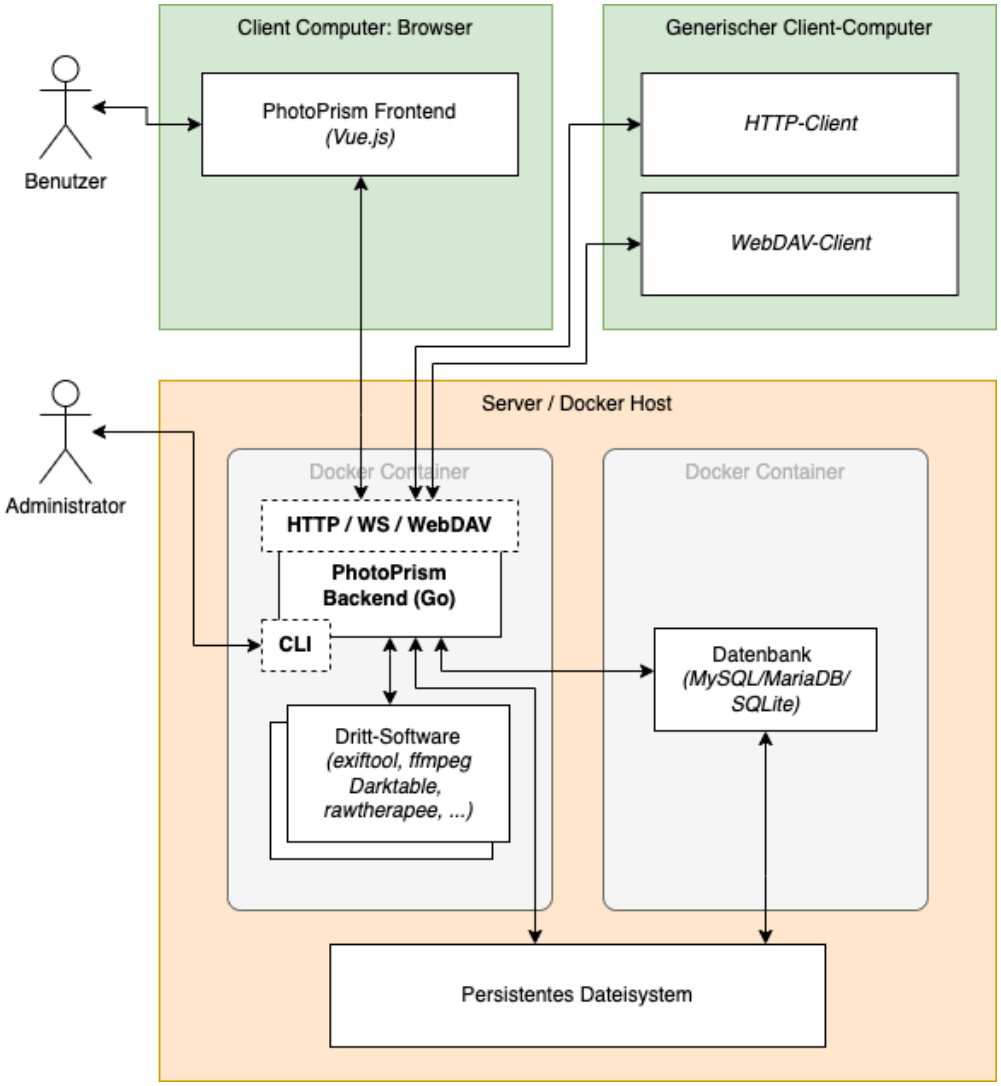

Abbildung 2: Überblick über PhotoPrisms Architektur.

Das System besteht im Wesentlichen aus der Serveranwendung (dem Backend), einem clientseitg ausgeführten Teil (dem Frontend) und einer SQL-Datenbank (MariaDB, MySQL oder SQLite) zur Persistierung strukturierter Daten und folgt damit der häufig anzutreffenden 3-Schichten-Architektur. Serverseitig gibt es noch eine ganze Liste weitere Programme, die PhotoPrism für bestimmte Aufgaben, wie beispielsweise die RAW-Konvertierung oder das Auslesen von Metadaten benötigt. Da diese Abhängigkeiten in Summe die manuelle Installation immens verkomplizieren, wird sowohl die Entwicklungsumgebung als auch die Anwendung selbst mit allen Abhängigkeiten mittels Docker

in einfach zu handhabende Images verpackt, aus denen dann Container gestartet werden können [7]. Mithilfe des Taskrunners *Make* [14] können bei der Entwicklung häufig auszuführende Abläufe durch ein einzelnes Kommando gestartet werden.

Das Backend, der Kern der Anwendung, ist in Go geschrieben. Eine schnelle, kompilierte Sprache, die sich ideal für nebenläufige Serveranwendungen eignet, die viele Aufgaben parallel ausführen sollen und eine hohe Anzahl an Clients zur selben Zeit bedienen kann. Das Frontend wird im Browser des Nutzers ausgeführt. Damit ist Javascript das Werkzeug der Wahl für den clientseitgen Teil der Anwendung. Um das Userinterface zu strukturieren bzw. zu modularisieren wird das Javascript-Framework Vue.js eingesetzt.

Während Metadaten und andere anwendungsspezifische Daten hauptsächlich in der Datenbank gespeichert werden, wird für Bilder und Videos das Dateisystem des Servers genutzt.

PhotoPrism besitzt mehrere Schnittstellen zur Interaktion, wobei die Hauptschnittstelle für alle Benutzer das browserbasierte User Interface ist, welches über eine RESTful-API mit dem Server kommuniziert. Diese Schnittstellen können auch direkt ohne die Nutzeroberfläche angesprochen werden. Von dieser Möglichkeit macht zum Beispiel eine von dritter Seite entwickelte Smartphone App Gebrauch [32]. Weiter existiert zum Zwecke der Administration eine CLI-Schnittstelle und das integrierte WebDAV-Protokoll ermöglicht Benutzern, mittels Dateiexplorer des Betriebssystems auf die Bildersammlung von PhotoPrism zuzugreifen und bietet damit eine zusätzliche komfortable Integration, um Bildersammlungen zu verwalten.

Die CLI-Schnittstelle ist dem Administrator vorbehalten. Es wird keine Zugriffskontrolle durchgeführt. Zugriff auf diese muss deshalb über das Betriebssystem eingeschränkt werden, sollten auch andere Akteure Zugang zum Server haben.

2.1.1 Relevante Module

Im Folgenden soll als Orientierungshilfe eine kurze Übersicht über die, für die Arbeit relevanten, Module von PhotoPrisms Quellcode gegeben werden.

`entity`- und `query`-Package

Diese beiden *Packages* dienen als Schnittstelle zur Datenbank. Das `entity`-Package enthält dabei die Modelle für die Datenstrukturen inklusive Validierungs- und Instanziierungslogik. Dagegen enthält das `query`-Package Operationen für komplexere Datenbankabfragen, die primär das Suchen bzw. die Abfrage von Datensätzen bezwecken.

`commands`-Package

Hier wird PhotoPrisms CLI-Schnittstelle implementiert.[1]

[1] PhotoPrism Quellcode: `internal/commands/ [feature/oidc-v2]`

`api`-Package

HTTP und Websockets Endpunkte sind in diesem Package zu finden.[2]

`service`-Package

Im `service`-Package werden die einzelnen Module von PhotoPrism initialisiert und registriert. Zugriffe auf die Funktionen der einzelnen Dienste sollten immer über dieses Package erfolgen.[3]

`acl`-Package

Das `acl`-Package enthält eine einfache Implementierung eines Modells für Zugriffskontrolle.[4]

Frontend-Komponenten

Das Frontend von PhotoPrism ist grundsätzlich über `pages`[5] organisiert. Komponenten, die an verschiedenen Stellen verwendet werden, sind im `component`-Ordner[6] zu finden. Weiterer, gemeinsam verwendeter, JavaScript-Code befindet sich im `common`-Ordner[7]. Hier ist zum Beispiel der Code zur Evaluation der Sichtbarkeit einzelner Komponenten auf Basis der Berechtigungen zu finden.

2.2 Authentifizierung und Usermanagement

Der Begriff Usermanagement bezeichnet die Menge aller Funktionen, die es ermöglichen Personen eine Identität im System zuzuordnen, diese zu identifizieren, zu authentifizieren und die Profilinformationen zu verwalten.

Die Authentifizierung beschreibt den Vorgang der Überprüfung und Verifizierung der Identität eines Benutzers oder Systems. Es gibt mehrere Wege, die Identität zu verifizieren: Den Nachweis der Kenntnis einer bestimmten Information (something you know, z.B. Passwort), den Besitz einer eindeutigen Sache (something you have, z.B. physischer Schlüssel, kryptografischer Token) oder biometrische Merkmale (something you are, z.B. Fingerabdruck). Bei Kombination mehrerer dieser Merkmale spricht man auch von Multi-Faktor-Authentifizierung. [6, S. F-91]

[2] PhotoPrism Quellcode: `internal/api/ [feature/oidc-v2]`
[3] PhotoPrism Quellcode: `internal/service/ [feature/oidc-v2]`
[4] PhotoPrism Quellcode: `internal/acl/ [feature/oidc-v2]`
[5] PhotoPrism Quellcode: `frontend/src/pages/ [feature/oidc-v2]`
[6] PhotoPrism Quellcode: `frontend/src/component/ [feature/oidc-v2]`
[7] PhotoPrism Quellcode: `frontend/src/common/ [feature/oidc-v2]`

2.3 Autorisierung und Rechteverwaltung

Alle Funktionen die zur Verwaltung von Berechtigungen gebraucht werden, werden unter dem Begriff der Rechteverwaltung zusammengefasst. Dazu gehören unter anderem Zuordnung, Auswertung und Durchsetzung von Berechtigungen.

Bei der Autorisierung wird, nachdem die Identität eines Nutzers verifiziert ist, geprüft, ob dieser bestimmte Aktionen durchführen oder auf bestimmte Ressourcen zugreifen darf. Fällt die Prüfung positiv aus, kann der Nutzer mit der Aktion fortfahren, andernfalls wird der Zugriff verweigert. Oft wird in diesem Zusammenhang auch von Access Control (AC) gesprochen.

2.3.1 Autorisierungsmodelle

Im Allgemeinen lässt sich ein Autorisierungsmodell darauf herunterbrechen, dass bei jeder Anfrage (Aktion) eines Subjekts (Benutzer oder Programm) für ein Objekt (Ressource) geprüft wird, ob eine entsprechende Berechtigung existiert.

Über die Zeit haben sich einige Modelle entwickelt, die jeweils auf spezifische Probleme im Bereich der Zugriffskontrolle zugeschnitten sind. Häufig verwendete Modelle sind: RBAC, ABAC, ACL, DAC, MAC oder Variationen davon.

Die Arbeit beschäftigt sich lediglich mit dem RBAC-Modell welches von Ferraiolo und Kuhn beschrieben wird. [8] PhotoPrism macht sich dieses Modell bereits zunutze, regelt Zugriffe aber nur auf Ebene der Ressourcen-Typen, nicht für einzelne Objekte (siehe auch Ist-Analyse des ACL-Packages).

2.3.2 Ebenen der Zugriffskontrolle

Im Rahmen dieser Arbeit ist es von Vorteil, Zugriffskontrolle über zwei Ebenen verteilt zu betrachten. Durch dieses Vorgehen lassen sich die Anforderungen an die zu wählenden AC-Modelle für Benutzerdaten getrennt von denen der Anwendungslogik betrachten und ermöglicht so zum einen weniger abstrakte Modelle zu verwenden und erleichtert zum anderen die Anforderungen schneller, iterativ und unabhängig voneinander zu implementieren und später einfacher und gezielter ändern oder erweitern zu können.

Nachteil dieses Vorgehens ist auf der anderen Seite, dass es dazu führen kann, Autorisierung an verschiedenen Stellen durch unterschiedliche Methoden zu implementieren, was spätere Änderungen oder Konsolidierung deutlich aufwendiger werden ließe (*technische Schulden*).

- **Domain Authorization** beschäftigt sich mit Zugriffen auf Bereiche und Funktionen der Anwendungslogik bzw. der generalisierten Berechtigungen für Typen von Ressourcen.

- **Data Authorization and Filtering** soll sicherstellen, dass die Erlaubnis für Zugriffe auf Benutzerdaten bzw. Ressourcen-Instanzen korrekt evaluiert und effizient gefiltert werden.

2.3.3 Domain Authorization

Damit neue Benutzer nicht direkt auf alle Bereiche und Funktionen zugreifen können, wie beispielsweise globale Einstellungen zu ändern, sich selbst Admin-Rechte zu vergeben und Daten zu modifizieren oder zu löschen, muss der Zugriff gezielt eingeschränkt werden können. Da alle Bereiche und Funktionen von PhotoPrism bekannt sind, können hier zunächst statische Zuordnungen von *Aktionen* für diese *Ressourcen* verwendet werden. Das hält den initialen Aufwand gering und bringt trotzdem Mehrwert für die Kunden.

Der Haupt-Unterschied zu *Data Authorization and Filtering* ist, dass die Zugriffskontrolle sich hier auf Bereiche der Anwendung bzw. Ressourcen-Typen statt Ressourcen-Instanzen fokussiert.

2.3.4 Data Authorization and Filtering

Im Gegensatz zu Domain Authorization zielt die Data Authorization darauf ab, Maßnahmen zur Zugriffskontrolle auf Entitäts-Ebene einzuführen, mit denen sich der Zugriff auf einzelne Objekte steuern lässt. Damit können unberechtigte Modifikationen an einzelnen Datensätzen unterbunden werden und gefilterte Ansichten auf Sammlungen umgesetzt werden. Konkrete Beispiele dafür sind die Pflege von privaten Sammlungen pro Nutzer, geteilte Alben mit Schreibzugriff für einzelne Benutzer, Zugriffskontrolle beim direkten Aufruf eines bestimmten Albums oder auch das Prüfen der Berechtigung bei direktem Zugriff auf einzelne Ressourcen (z.B. Bilder).

2.4 Single Sign-On und externes Usermanagement

Single Sign-On beschreibt eine Methodik, den Authentifizierungsprozess an eine dritte Stelle zu delegieren, die zentral für das Usermanagement verantwortlich ist. Damit lässt sich eine Identität über mehrere Systeme hinweg verwenden und es erhöht die Sicherheit, da sich ein Benutzer nur noch einmal anmelden muss und die delegierende Anwendung die Authentifizierung und das Handling von Passwörtern nicht mehr selbst durchführen muss. Dies setzt in jedem Fall voraus, dass diese dritte Stelle, oft auch als Identity Provider bezeichnet, gegenüber der Anwendung als vertrauenswürdig eingestuft werden kann.

Für Single Sign-On und das Auslagern des Usermanagements an spezialisierte Dienste kommen folgende Protokolle für PhotoPrisms Zwecke in Frage:

- **Security Assertion Markup Language (SAML) 2.0** existiert seit 2005 und unterstützt neben Authentifizierung auch Autorisierungskonzepte. Es ist ein XML-orientiertes Framework zur Übertragung von Benutzerauthentifizierung, Berechtigungen und anderen Attributen [21]. SAML ist im kommerziellen Umfeld als SSO-Lösung weit verbreitet [45]. Inzwischen wird jedoch eher OpenID Connect bevorzugt und Hersteller von IAM-Software wie beispielsweise Keycloak gehen dazu über, OIDC für die meisten Use-Cases zu empfehlen, da bei vergleichbarem Funktionsumfang geringerer Aufwand bei der Integration notwendig ist. [40]
- **OpenID Connect (OIDC)** gibt es erst seit 2014 und ist zu unterscheiden von OpenID. Trotz des geringen Alters hat dieses Protokoll jedoch schon eine beachtliche Verbreitung erreicht. OIDC basiert im Kern auf *OAuth 2.0* und erweitert dieses um eine „Identitätsebene“ und weitere optionale Funktionen, wie zum Beispiel Session Management. Weite Verbreitung, insbesondere bei anderen Open Source Projekten, vergleichsweise geringe Komplexität und überschaubarer Integrationsaufwand machen OpenID Connect zum klaren Sieger bei der Wahl eines geeigneten Protokolls für PhotoPrism.

Beide Standards weisen im Vergleich zu lokaler Authentifizierung eine höhere Komplexität auf und bringen ihrerseits wieder neue Sicherheitsrisiken ein oder können durch Fehler bei der Implementierung ernsthafte Sicherheitslücken versursachen. Werden diese Risiken jedoch angemessen adressiert, kann mit deren Hilfe Aufwand für Betrieb und Wartung von Anwendungslandschaften reduziert werden, und vor allem die Sicherheit signifikant verbessert werden, ohne alle Features in der eigenen Anwendung selbst zu implementieren.

2.4.1 OAuth 2.0

Das *OAuth 2.0 Authorization Framework* (*RFC 6749*) erlaubt einer dritten Anwendung eingeschränkten Zugriff auf eine Ressource im Namen des Eigentümers zu erhalten, ohne dessen Zugangsdaten kennen oder verarbeiten zu müssen. Stattdessen wird nach Bestätigung durch den Eigentümer ein *Access-Token* ausgestellt. Die Illustration in Abbildung 3 soll den Ablauf des Protokolls verdeutlichen.

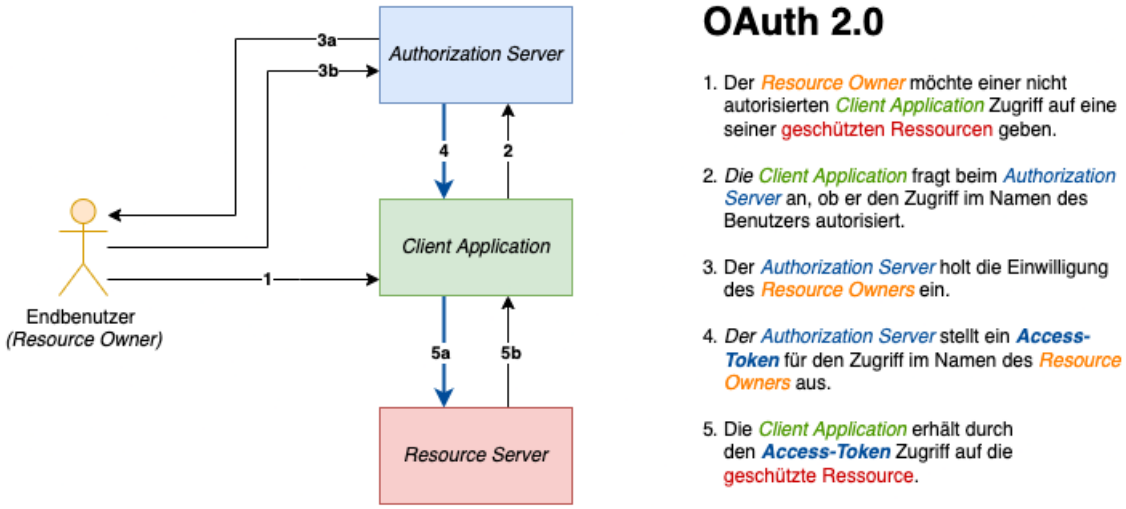

Abbildung 3: Vereinfachte Darstellung der Funktionsweise von OAuth 2.0.

In weiteren Spezifikationen wurden Best-Practices für die Verwendung mit browserbasierten Anwendungen [29] erarbeitet und Erweiterungen wie *PKCE* (*RFC 7636*) entwickelt, die für zusätzliche Sicherheit sorgen. Um die, über mehrere Dokumente verteilten, Spezifikationen in einem Dokument zu konsolidieren, wird zur Zeit mit *OAuth 2.1* [16] an einem Update gearbeitet. OAuth 2.0 bildet die Basis für OpenID Connect.

Client Typen

Da Oauth 2.0 vorsieht, dass sich die Clients selbst vor dem *Authorization Server* authentifizieren müssen, unterscheiden die Spezifikationen zwischen den beiden Clienttypen *Public* und *Confidential*. Anwendungen, die in der Lage sind das Client-Passwort (*Client Secret*) sicher zu verwalten, werden als *Confidential* betrachtet. Anwendungen, die nicht in der Lage sind die Vertraulichkeit von Client-Passwort und anderen Daten (z.B. Token) zu garantieren, werden als *Public* klassifiziert und brauchen sich daher nicht zu authentifizieren [15, Kapitel 2.1]. Aus sicherheitstechnischer Perspektive ist daher immer ein *Confidential Client* zu bevorzugen, wenn die Architektur des Systems es zulässt.

Authorization Grants

Die Spezifikation von OAuth 2.0 beschreibt eine Reihe von *Flows*, die es einem *Client* ermöglichen, *Access Token* vom *Authorization Server* anzufordern. Diese Flows bilden den Kern des Protokolls. Mit der Einführung von *PKCE* und der Entwicklung von OAuth 2.1 wurden der *Implicit Grant* und der *Resource Owner Password Credentials Grant* als veraltet markiert und sollten nicht mehr verwendet werden [16, Kapitel 12]. Damit bleibt für die Autorisierung im Namen eines Benutzers für beide Client Typen nur noch die Verwendung des *Authorization Code Grant mit PKCE* welcher in Abbildung 4 illustriert ist. Schritte, die durch PKCE im Flow ergänzt werden, sind grün hervorgehoben.

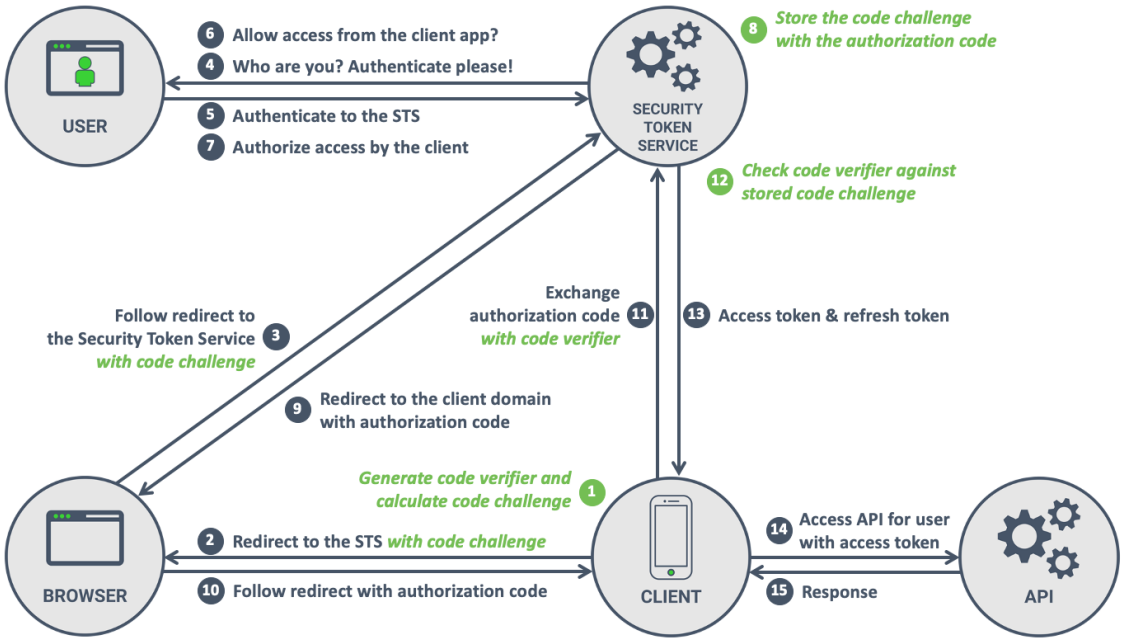

Abbildung 4: Authorization Code Flow mit PKCE. [4]

Proof Key for Code Exchange (PKCE)

PKCE, definiert in *RFC 7636*, ist eine Erweiterung für den in OAuth 2.0 beschriebenen *Authorization Code Grant*. Durch diese ist es nun auch mit vertretbarem Risiko möglich, diesen Flow bei einem *Public Client* einzusetzen. PKCE wurde dazu entwickelt Angriffe durch abgefangene *Authorization Codes* zu verhindern. Der Einsatz von PKCE wird von [16, Kapitel 7.8] daher für alle Client Typen in Verbindung mit dem *Authorization Code Grant* empfohlen.

2.4.2 OpenID Connect 1.0 (OIDC)

OpenID Connect [37] erweitert das OAuth 2.0 Protokoll um eine „Identitätsschicht“. Es ist damit möglich die Identität eines Benutzers von einer externen Stelle, dem IdP,

verifizieren zu lassen und von dieser Informationen über den Benutzer zu erhalten.

Der modulare Aufbau der *OpenID Connect Protocol Suite* ermöglicht es, sich bei der Implementierung auf einzelne Features der Spezifikationen zu beschränken, wie in Abbildung 5 zu sehen ist. Die Grafik zeigt auch die Protokolle auf die OIDC aufbaut.

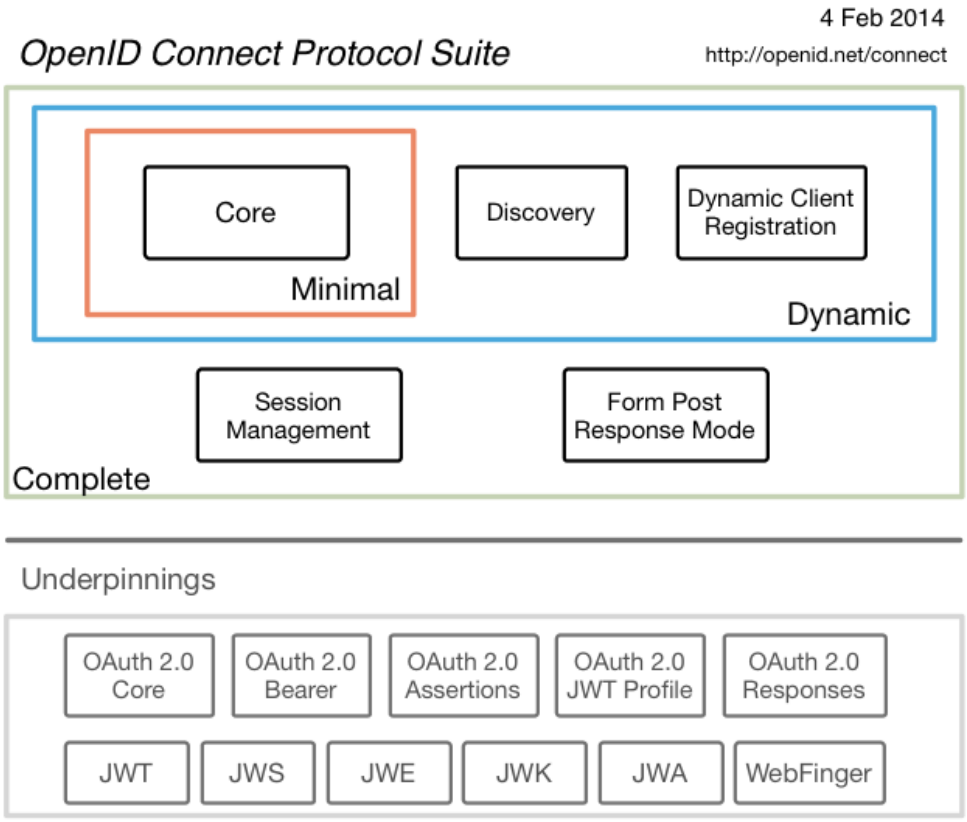

Abbildung 5: OpenID Connect Protocol Suite. [44]

Das Kern-Protokoll spezifiziert drei Akteure, deren Funktion im folgenden kurz beschrieben. Zusätzlich wird in Abbildung 6 illustriert, wie ein typischer Flow bei OIDC aussieht.

- **OpenID Provider (OP):** Der OP, im Allgemeinen auch als Identity Provider (IdP) bezeichnet, kümmert sich um die Authentifizierung des Benutzers und die Verwaltung der Profilinformationen. Die Rolle entspricht etwa der des *Authorization Servers* in OAuth 2.0. Er kann aber auch Informationen in der Rolle des *Resource Servers* bereitstellen. Zum Beispiel sei der hier der *Userinfo* Endpunkt genannt. [37, Kapitel 5.3]
- **Relying Party (RP):** Die RP entspricht der *Client Application* im OAuth 2.0 Protokoll. Sie delegiert die Authentifizierung zum OP und nutzt die Profilinformationen des authentifizierten Benutzers, ohne sich selbst um die Verwaltung der Profilinformationen kümmern zu müssen.

- **End-User:** Der Endnutzer, im weiteren Text einfach Benutzer, entspricht dem OAuth 2.0 *Resource Owner*. Er authentifiziert sich beim OP und erlaubt der RP seine Profilinformationen zu nutzen.

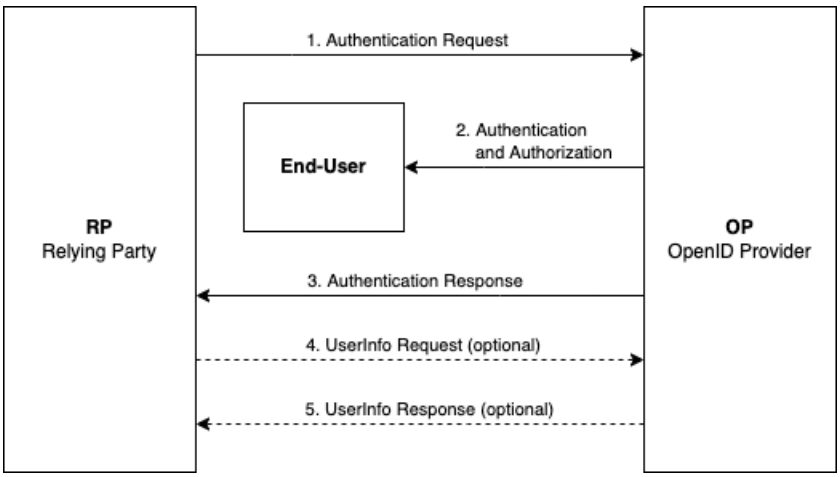

Abbildung 6: Abstrakter OpenID Connect Protokoll Flow. [37, Kapitel 1.3]

Die Profilinformationen werden durch den *Userinfo*-Endpunkt bereitgestellt oder werden direkt in Form des *ID-Tokens* an die RP übermittelt. Der *ID-Tokens* wird nach erfolgreicher Authentifizierung, zusätzlich zum *Access-Token*, ausgestellt und enthält *Claims* über den Benutzer. Im Gegensatz zum *Access-Token*, ist daher auch die Repränsetation durch einen *JWT* obligatorisch. [37, Kapitel 2]

Der für die Arbeit relevante Flow entspricht dem in Abbildung 4 gezeigten, mit dem Unterschied, dass zusätzlich noch ein *ID-Token* vom Security Token Service an PhotoPrism übermittelt wird (Schritt 13) und PhotoPrism den *Access-Token* nicht für weitere Zugriffe auf dritte APIs benötigt (Schritt 14 und 15).

Für PhotoPrism sind über das *Core*-Protokoll hinaus folgende Spezifikationen interessant:

- Discovery [38]
- Session Management [26]
- Dynamic Client Registration (Optional) [36]

Scopes und Claims

Dieses Konzept stammt aus dem zugrundeliegenden OAuth 2.0 Protokoll. Ein *Claim* ist eine Information, die mit einem Benutzer bzw. Subjekt assoziert ist. Ein *Scope* hat den Zweck, den Zugriff auf diese Claims gezielt einzuschränken. Scopes können auch als Gruppierung von Claims betrachtet werden. Beim Einleiten eines OAuth-Flows können die gewünschten Scopes via URL-Parameter beim IdP angefordert werden.

OpenID Connect nutzt diesen Mechanismus, um mittels `openid`-Scope, den für dieses Protokoll spezifischen *ID-Token* anzufragen. [37, Abschnitt 3.1.2.1] Darüber hinaus definiert OIDC weitere optionale Scopes: `profile`, `email`, `address`, `phone` und `offline_access`. [37, Abschnitt 5.4 und 11]

2.4.3 JSON Web Token (JWT)

JWT [19] ist essentieller Bestandteil von OpenID Connect. Diese Spezifikation standardisiert die Form und die Verifizierung von Informationen mittels kryptografischer Verfahren, die über den Token augetauscht werden sollen. Damit ist es möglich, Informationen in einen Token zu kodieren und vor Modifikationen Dritter zu schützen. Es entfällt so die Notwendigkeit, diese Informationen auf dem Server vorzuhalten. Ein solcher Token kann beispielsweise mit einer Eintrittskarte für ein Konzert verglichen werden. Unter `https://jwt.io/` können als interaktives Beispiel JWTs generiert, decodiert und verifiziert werden.

Der *ID-Token* aus dem OpenID Connect Protokoll setzt die Formatierung, Signierung und Validierung nach dem JWT Standard voraus. OAuth 2.0 erlaubt diese Form ebenfalls für seine *Access Token*.

Die Verfahren zur Verschlüsselung und Signierung sind durch die Standards JSON Web Encryption (JWE) [20] und JSON Web Signature (JWS) [18] standardisiert.

Die Spezifikation für JSON Web Keys (JWK) definiert darüber hinaus eine webtaugliche Repräsentation für kryptografische Schlüssel. [17]

2.5 Qualitätsmanagement

Bei PhotoPrism als Open-Source und Community-Projekt sind Anforderungen in der Regel eher von dynamischer Natur. Sie liegen anfangs oft nur als grobe Ideen ohne genaue Aktzeptanzkriterien vor, die dann während der Entwicklung und durch Nutzerfeedback konkretisiert oder weiterentwickelt werden. Außerdem sind Ressourcen zur Entwicklung aufgrund des kleinen Teams an Kernentwicklern nur sehr begrenzt planbar.

Um trotzdem die Sicherstellung eines Mindestmaß an Qualität zu gewährleisten, eignet sich ein kontinuierliches Vorgehen [22, S. 11] hier besonders gut. Dabei wird neben funktionsorientierten Testtechniken [22, S. 49] ein hoher Wert auf das Feedback durch die Community gelegt. Das Feedback fließt dann wieder in den Entwicklungszyklus ein und Features werden inkrementell verbessert. Durch einen solchen Ansatz können bei der Entwicklung, trotz begrenzter Ressourcen, schnell Ergebnisse erzielt und Probleme individuell und zielgerichtet gelöst werden.

Um dem Endbenutzer die Wahl selbst zu überlassen, ob er experimentelle bzw. Beta-Software einsetzen möchte, stellt PhotoPrism neben kontinuierlichen Builds des aktiven

Entwicklungsbranches auch *Releases* bereit, die einen stabilen Zustand der Anwendung repräsentieren. Ist ein Feature gestestet, dokumentiert und in einem sinnvollen Rahmen funktionsfähig, wird es ins Release überführt.

Konkret werden bei PhotoPrism folgende Maßnahmen eingesetzt:

- **Komponententests** prüfen die Funktionalität einzelner Funktionen oder Module als „kleinste sinnvoll unabhängig testbare Einheit eines Programms“ [22, S. 371].
- **Integrationstests** sind für die Überprüfung der korrekten Interaktion mehrerer Komponenten verantwortlich [22, S. 428]. Hierzu zählen auch Tests, die Interaktionen mit der Datenbank über den ORM prüfen.
- **Akzeptanztests** bzw. Systemtests werden eingesetzt, um die Funktionalität des gesamten Systems gegen die Anforderungen zu testen. Bei PhotoPrism beschränken sich diese hauptsächlich auf Funktionstests [22, S. 436]. Den ebenfalls von Liggesmeyer beschriebenen Regressionstests werden durch die automatisierte Durchführung aller bisher aufgezählten Testarten in der CI-Pipeline Rechnung getragen.
- **Beta-Testing** beschreibt im Kontext von PhotoPrism die freiwillige Nutzung von „Pre-Release Builds“ und das freiwillige Feedback durch diese Benutzer, z.B. in Form von Issues auf Github. Beta-Testing führt Liggesmeyer ebenfalls als Teil der Systemtests auf [22, S. 437]. Da dem bei PhotoPrism durch die Natur des Projektes ein besonderer Stellenwert zukommt, wird er hier nochmals als eigener Punkt aufgeführt.

Bei PhotoPrism wird keine scharfe Linie zwischen Komponenten- und Integrationstests gezogen. Sie werden stattdessen im selben Kontext ausgeführt und je nach Testsubjekt die passende Testoption gewählt. Durch das Vorgehen können Redundanzen und Arbeitsaufwand eingespart werden.

Tests sind keine Garantie für eine fehlerfreie Implementierung, sie können lediglich die Präsenz von Fehlern nachweisen. Sie sind darüber hinaus ein wichtiges Werkzeug, um sicherzustellen, dass auch nach Refactorings noch alles sauber funktioniert.

2.6 Sicherheit

Privatsphäre und Datenschutz sind zentrale Werte von PhotoPrism und besonders im Kontext der Verabeitung persönlicher bzw. vertraulicher Daten, dürfen sicherheitstechnische Aspekte bei der Planung und Umsetzung von Authentifizierung und Autorisierung nicht vernachlässigt werden.

Neben technischen Maßnahmen und Empfehlungen, werden außerhalb der Arbeit ausführliche How-To's in die Dokumentation aufgenommen, um unerfahrene Benutzer mit potentiellen Risiken und Schutzmaßnahmen vertraut zu machen, die von PhotoPrism nicht adressiert werden können, wie zum Beispiel der Betrieb in nicht vertrauenswürdigen Netzwerken ohne TLS-Verschlüsselung oder der Verwendung schwacher Passwörter.

Während Entwicklungsaufwand und Komplexität bei Einrichtung und Benutzung im Rahmen gehalten werden müssen, soll Benutzern gleichzeitig ein angemessener Mindestschutz vor Sicherheitsrisiken zuteil werden.

Um höheren Anforderungen an die Sicherheit in anspruchsvollen Einsatzszenarien gerecht zu werden, kann (insbesondere für die Authentifizierung) auf Protokolle zur Delegierung an spezialisierte IAM-Systeme zurückgegriffen werden.

Weiter liefert der *OWASP ASVS* einen praxisnahen Empfehlungskatalog, der für sicherheitsrelevante Entscheidungen eine gute Grundlage für die Entwicklung an PhotoPrism bereitstellt.

2.6.1 OWASP Application Security Verification Standard (ASVS)

Das OWASP ist eine Non-Profit Organisation mit Mitgliedern aus der ganzen Welt, die sich der Verbesserung der Sicherheit von Software verschrieben hat. Deren Projekt *OWASP Top 10* eignet sich bestens für Projekte mit begrenzten Ressourcen, um die größten aktuellen Sicherheitsprobleme zu identifizieren und zu priorisieren. Gleichzeitig werden Gegenmaßnahmen empfohlen und Beispiele für konkrete Angriffsszenarien gezeigt.

Um die Sicherheit einer Web-Anwendung über dieses Minimum hinweg auszubauen, hat die Organisation den *OWASP Application Security Verification Standard* [27] entwickelt, der laufend weiterentwickelt wird, um auch neusten Erkenntnissen sicherheitskritischer Themen Rechnung zu tragen. Im Gegensatz zu den Top 10 wurde der Standard so formuliert, dass er für alle Stakeholder eines Projekts eine einheitliche Definition bezüglich Sicherheit bereitstellt und die Anforderungen im einzelnen verifizierbar sind. Je nach Schutzbedarf unterscheidet der Standard hier drei Anforderungs-Stufen, wobei ASVS Level 1 (Opportunisitc) etwa den empfohlenen Maßnahmen aus den Top 10 entspricht [27, S. 9].

Außerdem strebt das OWASP ASVS an, mit anderen Normungsgremien zusammen zu arbeiten. Im Bezug auf Session-Management und Authentifizierung wurde der ASVS bereits zum Standard NIST 800-63 kompatibel gemacht. Auch der BSI Grundschutz wird wahrscheinlich in Zukunft Berücksichtigung finden[8].

[8] `https://github.com/OWASP/ASVS/issues/384`

3 Anforderungsanalyse

In diesem Kapitel werden alle Anforderungen zusammengetragen und priorisiert. Die Priorität der Anforderungen werden nach der deutschen *Autorenrichtlinie zur Erstellung eines benutzerdefinierten Bausteins* definiert (welche ihrerseits auf der englischen Spezifikation *RFC 2119* basiert):

1. **MUSS:** Essentielle Anforderung. Muss unbedingt erfüllt werden. Alle MUSS-Anforderungen eines Features müssen erfüllt sein, um es ins Release zu überführen.
2. **DARF NICHT / DARF KEIN:** Essentielle Anforderung. Darf auf keinen Fall getan werden. Alle DARF NICHT / DARF KEIN-Anforderungen eines Features müssen erfüllt sein, um es ins Release zu überführen.
3. **SOLL / SOLLTE:** Wichtige Anforderung, nicht essentiell wenn es gute Gründe gibt. Kann nachträglich implementiert werden.
4. **SOLLTE NICHT / SOLLTE KEIN:** Bedeutet dass etwas normalerweise nicht getan werden sollte, es sei denn es gibt gute Gründe. Kann nachträglich implementiert werden.
5. **KANN / DARF:** Wünschenswert, nicht essentiell. Kann bei zu hohem Aufwand ausgelassen oder auf unbestimmte Zeit verschoben werden werden.

Die Anforderungen in diesem Kapitel repräsentieren nicht den Umfang der Arbeit, sondern stellen eine Sammlung derjenigen Anforderungen dar, die während der Arbeit indentifiziert werden konnten und die im Rahmen der Mehrbenutzer-Transformation implementiert werden sollen. Es ist nicht Anspruch der Arbeit, alle Anforderungen umzusetzen. Vielmehr sollen Sie dem Leser und den Entwicklern einen Kontext verschaffen, um Entscheidungen leichter nachvollziehbar zu machen und um das Vorgehen bei der Entwicklung besser strukturieren zu können.

3.1 Funktionale Anforderungen

Die Anforderungen an die Mehrbenutzer-Funktionalität von PhotoPrism lassen sich grob in Themengruppen aufteilen:

1. Usermanagement
2. Rechteverwaltung
3. Single-Sign-On (OpenID Connect)

Die Reihenfolge der Features wurde bewusst so gewählt, da ohne Usermanagement eine Verwaltung der Rechte keinen Sinn ergibt. Ebenso ist für die Nutzung von OpenID Connect notwendig, dass PhotoPrism das Konzept unterschiedlicher Benutzer kennt.

3.1.1 Usermanagement

Um Sicherheitsaspekte und gute Benutzbarkeit out-of-the-box zu vereinen, ist es gerade zu Beginn nicht ausreichend nur auf ein externes Usermanagement zu setzen.

Durch die frühe Einführung von OpenID Connect kann ein integriertes Usermanagement zunächst mit minimalem Umfang eingeführt werden und die Integration mit OpenID Connect weiter ausgebaut werden.

Im Idealfall kann später sogar ganz auf das interne Usermanagement verzichtet werden, wenn die Integration ausgereift genug ist und leichtgewichtige und zuverlässige IAM-Lösungen mit PhotoPrism ausgeliefert werden können.

[R-U01] Neue Benutzer

Der Administrator MUSS neue Benutzer im System anlegen können. Einem Benutzer MUSS dabei mindestens ein einmaliger Benutzername und ein Passwort zugeordnet werden. Weiter SOLL eine E-Mail-Adresse (ebenfalls einmalig) und ein Anzeigename (bzw. voller Name) hinterlegt werden können. Das Passwort und der Username MUSS jeweils mindestens aus vier Zeichen bestehen. Wenn eine E-Mail-Adresse eingetragen wird MUSS diese ein gültiges Format haben. [34], [47]

[R-U02] Eigenständige Registrierung

Ein neuer Benutzer SOLL sich selbst registrieren können. In diesem Fall SOLL es die Möglichkeit geben, PhotoPrism so zu konfigurieren, dass das Benutzerkonto erst nach Verifizierung der E-Mail nutzbar ist, um potentiellem Missbrauch vorzubeugen.

[R-U03] E-Mail Whitelisting

Um den Kreis an Benutzern, die sich selbst registrieren dürfen, einzuschränken, KANN der Administrator über eine Whitelist Domains oder einzelne E-Mail-Adressen zur Selbst-Registrierung freigeben. Ein neuer Benutzer, der über kein von der Whitelist abgedecktes E-Mail-Konto verfügt, KANN sich registrieren, MUSS in dem Fall aber vom Administrator manuell freigegeben werden, um sich einloggen zu können. Alternativ kann der Administrator einen sochen selbst anlegen.

[R-U04] Benutzer entfernen/deaktivieren

Ein Administrator MUSS einen existierenden Benutzer wieder entfernen können. Da andere Daten mit einem Benutzer assoziiert sein können, MUSS der Datensatz im System verbleiben. Dabei darf der Benutzer sich nicht mehr anmelden können (MUSS). Ebenso SOLL sich ein Benutzer selbst deaktivieren oder löschen können.

[R-U05] Änderungen von Benutzerdaten durch Admin

Ein Administrator MUSS die Daten eines existierenden Benutzer nachträglich ändern können. Das gilt für folgende Eigenschaften: voller Name, Passwort, E-Mail. Ein Benutzer SOLL diese Änderungen auch selbst durchführen können. Alle anderen Eigenschaften eines Benutzers dürfen nicht geändert werden (MUSS).

Einschränkungen die in [R-U01] für Passwort und E-Mail spezifiziert sind, MÜSSEN auch bei Änderungen überprüft werden.

[R-U06] Benutzer auflisten

Ein Administrator muss den Überblick über alle Benutzer des Systems haben. Dafür MUSS das System eine Auflistung aller Benutzer samt relevanter Daten, wie Username, E-Mail und vollen Namen ausgeben können.

[R-U07] Authentifizierung: An- und Abmelden

Ein Benutzer MUSS sich mit Benutzernamen und Passwort authentifizieren und die daraufhin von PhotoPrism erstellte Session auch selbst wieder beenden können. Die Sitzung MUSS nach einer festgelegten Zeit der Inaktivität des Benutzers automatisch beendet werden.

[R-U08] 2-Faktor Authentifizierung

Ein Benutzer SOLL seinen Zugang zu PhotoPrism zusätzlich absichern, indem er Methoden zur Zwei- oder Mehr-Faktorauthentifizierung aktiviert. Der Administrator KANN festlegen, diese Form der Authentifizierung obligatorisch für alle Benutzer der Instanz vorrauszusetzen.

[R-U09] Profil-Ansicht in der Nutzeroberfläche

Ein Benutzer MUSS sich im Frontend über den Zustand seiner Authentifizierung informieren können. Wenn der Benutzer authentifiziert ist, sollen sein voller Name, seine E-Mail oder sein Username im Menü oder der Toolbar angezeigt werden. Klicken auf das Element MUSS zu einer Profil-Seite führen, bei der benutzerspezifische Einstellungen vorgenommen werden können. Diese sind zum aktuellen Zeitpunkt noch nicht näher spezifiziert.

[R-U10] E-Mail Bestätigung

Optional KANN der Administrator voraussetzen, dass der Besitz der bei [R-U01] angegeben E-Mail Adresse verifiziert wird bevor das Konto nutzbar ist, indem ein Link zur Bestätigung an die angegebene Adresse gesendet wird.

[R-U11] Passwortsicherheit

PhotoPrism SOLL kompromittierte Passwörter erkennen und Benutzer warnen können. Dazu kann entweder ein Listenabgleich oder die Nutzung einer externen API verwendet werden.

Optional KANN dem Benutzer bei Verwendung der Weboberfläche ein visuelles Feedback zur Passwortstärke angezeigt werden. (setzt R-U02 und R-U05 voraus)

3.1.2 Rechteverwaltung

[R-P01] Ressourcen und Aktionen

Es MUSS jeder Ressource ein Satz an Aktionen zugeordnet sein, der den Interaktionsmöglichkeiten mit der Ressource entspricht und über die HTTP-API ausgeführt werden soll. Zum Beispiel können auf die Ressource User die Aktionen Erstellen, Löschen, Lesen, Aktualisieren angewendet werden.

[R-P02] Rollenkonzept

PhotoPrism MUSS ein Rollenkonzept implementieren, welches in der Lage ist, einzelne Berechtigungen für verschiedene Ressourcen und Aktionen sinnvoll unter einer Rolle zu gruppieren, damit die Berechtigungen nicht jedem Benutzer einzeln zugewiesen werden müssen. Dies macht es später einfacher neue Aktionen und Ressourcen hinzuzufügen.

[R-P03] Statische Rollen

Einem Benutzer MUSS eine von zwei Rollen zugeordnet werden können:

- **Administrator:** Benutzer mit dieser Rolle dürfen alle Aktionen auf allen Ressourcen ausführen. Diese Rolle MUSS dem ersten Benutzer automatisch zugewiesen werden.
- **Member:** Dem Member ist grundsätzlich nur lesender Zugriff erlaubt, darf aber weder lesenden noch schreibenden Zugriff auf globale Einstellungen von PhotoPrism haben. Diese Rolle wird einem Benutzer standardmäßig zugeordnet. Genaue Spezifikation unter R-P04.

[R-P04] Rolle: Member

An einen Benutzer mit der Rolle Member, kurz einem Member, werden folgende Anforderungen gestellt:

- Kein Zugriff auf die Bereiche *Archiv, Überprüfen, Privat, Einstellungen, Dateien* (MUSS)

- Die zu diesen Bereichen gehörenden Menüeinträge müssen ausgeblendet werden. (SOLL)
- Als archiviert oder privat markierte Elemente dürfen nicht in den Suchergebnissen auftauchen. (MUSS)
- Als archiviert oder privat markierte Elemente dürfen nicht direkt angezeigt, heruntergeladen, gelöscht oder anderweitig manipuliert werden können. (MUSS)
- Alle anderen Daten dürfen nur lesend ausgegeben werden. Hochladen, Löschen, Modifizieren und Teilen muss unterbunden werden. (MUSS)
- UI-Elemente die Interaktionen zum Hochladen, Löschen, Modifizieren und Teilen von Daten bereitstellen, sollen deaktiviert oder, wenn keine andere Funktionalität damit assoziiert ist, ganz ausgeblendet werden. (SOLL)
- *Versteckte Gesichter* und damit assozierte UI-Elemente sollen nicht angezeigt werden. (MUSS)
- Keinen Zugriff auf die WebDAV Schnittstelle. (MUSS)

[R-P05] Dynamische Rollen

Ein Administrator KANN selbst neue Rollen erstellen, benennen und diesen erlaubte Aktionen zuordnen oder bestimmte Aktionen explizit für Inhaber dieser Rollen verbieten.

[R-P06] Eigene Mediensammlungen

Ein Benutzer SOLL eigene Sammlungen von Bildern und Videos unterhalten können, die anderen Benutzern nicht zugänglich sind.

[R-P07] Teilen von Alben mit Personen ohne Benutzerkonto

Ein Benutzer SOLL Alben mit Personen teilen können, die kein Benutzerkonto für die PhotoPrism Instanz besitzen. Alle Personen mit denen ein Album geteilt wird, können Inhalte sehen, jedoch nicht bearbeiten, löschen oder neue hinzufügen. Der Benutzer soll in der Lage sein, die geteilten Alben auch wieder zu sperren. Setzt R-P06 voraus.

[R-P08] Gemeinsame Alben mit anderen Benutzern einer Instanz

Ein Benutzer SOLL eigene private Alben mit anderen Benutzern teilen können. Dabei soll der Eigentümer steuern können, wer Zugriff darauf hat. Alle Benutzer mit denen ein Album geteilt wird, können Inhalte sehen und neue Inhalte hinzufügen und Inhalte entfernen (nur aus dem Album). Bearbeiten von Metadaten soll nur für Eigentümer der Inhalte möglich sein. Setzt R-P06 voraus.

[R-P09] Gemeinsame Alben mit Benutzern anderer Instanzen

Ein Benutzer KANN eigene private Alben mit Benutzern anderer Instanzen teilen. Alle Benutzer mit denen ein Album geteilt wird, können Inhalte sehen und neue Inhalte hinzufügen und Inhalte entfernen (nur aus dem Album). Bearbeiten von Metadaten soll nur für Eigentümer der Inhalte möglich sein. Setzt R-P06 voraus.

[R-P10] Öffentlicher Bereich

PhotoPrism SOLL im Private Modus einen öffentlichen Bereich bekommen, in dem Bilder und Alben ohne Authentifizierung zugänglich gemacht werden können. Damit Bilder und Alben in diesem Bereich erscheinen, müssen Sie vom Eigentümer explizit als öffentlich markiert werden.

3.1.3 Single Sign-On (SSO)

[R-S01] Delegierte Authentifizierung und externes Usermanagement

PhotoPrism MUSS die Authentifizierung eines Benutzers an einen externen Dienst delegieren können. Über die Verwendung dieses Features MUSS allein der Administrator entscheiden können. Dabei MUSS der externe Dienst PhotoPrism alle relevanten Informationen über den Benutzer (Username, E-Mail, voller Name bzw. Anzeigename) bereit stellen.

Im Falle von aktivem Single Sign-On SOLL PhotoPrism ausschließlich externe Identitäten erlauben. Interne Benutzer, mit Ausnahme des Administrators, haben bei Verwendung der externen Authentifizierung keinen Zugriff auf die Anwendung. Damit soll sichergestellt werden, dass ausschließlich eine einzige vertrauenswürdige Quelle von Identitäten genutzt wird.

[R-S02] Automatische Weiterleitung

Ein unangemeldeter Benutzer SOLL bei aktiviertem OpenID Connect statt dem Login-Screen von PhotoPrism, direkt zum Login des IdP weitergeleitet werden. Dabei MUSS einem Administrator trotzdem die Möglichkeit bleiben sich lokal einzuloggen. Außerdem MUSS die automatische Weiterleitung deaktiviert werden, wenn der IdP nicht erreichbar ist.

[R-S03] Rechteverwaltung via Identity Provider

Voraussetzung für die externe Rechteverwaltung ist, dass PhotoPrism bereits die Anforderungen RP-01 bis RP-04 erfüllt.

PhotoPrism MUSS in der Lage sein, Informationen zur Rolle des Benutzers bei der Anmeldung am externen Identity Provider, nach festzulegenden Kriterien, zu extrahieren und dem Benutzer intern korrekt zuzuordnen.

[R-S04] Verknüpfen von Benutzern

Es SOLL ermöglicht werden, dass bereits existierende lokale Konten von Benutzern mit einer externen Identität verknüpft werden können. Dies SOLL vom Administrator durchgeführt werden können oder ein Benutzer KANN dies selbst veranlassen.

3.2 Nicht-Funktionale Anforderungen

Da PhotoPrism bereits ein produktiv eingesetztes System ist, ergeben sich viele nichtfunktionale Rahmenbedingungen schon aus der bisherigen Architektur. Für den Kontext der Arbeit sollen hier dennoch die Wichtigsten aufgeführt werden.

[R-N01] Benutzererfahrung

Da sich die Software auch an unerfahrene Benutzer richtet, SOLL der Aufwand für Konfiguration und laufenden Betrieb möglichst niedrig gehalten werden. Durch Annahmen über das Nutzungsverhalten (u.a. abgeleitet aus Community-Feedback) kann hier bei der Implementierung auf Parametrisierung verzichtet werden. Da man so nicht immer allen Ansprüchen gerecht werden kann und es in naher Zukunft auch Versionen für den professionellen Einsatz geben soll, kann durch sinnvolle Defaults der Einrichtungsaufwand niedrig gehalten und bei Bedarf trotzdem die nötige Flexibilität erreicht werden.

Zudem soll für Endanwender die Bedienung möglichst intuitiv stattfinden können und Aktionen sollen in angemessener Zeit durchgeführt werden können.

[R-N02] Datenhaltung und Rückwärts-Kompatibilität

Ist eine Persisitierung von Daten erforderlich, MUSS die bereits bestehende Datenbank-Anbindung genutzt werden. Nachträgliches Löschen oder Änderung an Feldern bestehender persistenter Datenstrukturen SOLL aus Gründen der Rückwärts-Kompatibilität nach Möglichkeit vermieden werden, da dies mangels Migrations-Logik zu unerwünschten Effekten führen kann. Neue Tabellen oder Felder können jedoch problemlos hinzugefügt werden.

[R-N03] Sicherheit

Sicherheitsaspekte aus dem *OWASP Application Security Verification Standard* SOLLEN bei der Implementierung berücksichtigt und Maßnahmen gegen potentielle Sicherheitsrisiken einer sorgfältigen Trade-Off Analyse unterzogen werden. Es ist nicht Ziel der Arbeit alle Maßnahmen zu implementieren. Jedoch SOLLTE auf Mängel und Handlungsbedarf im Hinblick auf *ASVS Stufe 1* hingewiesen werden. *ASVS Stufe 2* KANN schon berücksichtigt werden, wenn die Maßnahmen ein gutes Kosten/Nutzen-Verhältnis aufweisen.

[R-N04] Tests

Eine funktionale Anforderung MUSS durch automatisierte Tests verifiziert werden. Das Testen einer Komponente oder einer Funktion SOLL so früh wie möglich im Entwicklungsprozess stattfinden. Die Granularität der Tests soll sich an der zu testenden Funktionalität orientieren. Es sollten mindestens Tests für gewünschtes Systemverhalten, Randbedingungen und Fehlerfälle implementiert werden. [39, S. 136]

[R-N05] Drittanbieter Lizenzen

Bei Nutzung von Code aus dritter Hand ist auf eine Lizenz zu achten, die es erlaubt, den Code ohne Lizenzkosten in PhotoPrism zu verwenden.

[R-N06] Konfiguration

Parameter, die Features von PhotoPrism konfigurieren, MÜSSEN per CLI-Option oder Umgebungsvariable an das Programm übergeben werden können. Es SOLLEN idealerweise beide Optionen unterstützt werden. Parameter, die ohne Neustart der Anwendung veränderbar sein sollen, SOLLEN über die Benutzeroberfläche konfigurierbar sein.

[R-N07] Responsive Design

Das Layout des Frontend von PhotoPrism SOLL sich der Bildschirmgröße des Endgerätes entsprechend anpassen, um eine gute Benutzererfahrung für die Benutzer gewährleisten zu können.

[R-N08] Persistierung der Daten

Ein zu implementierendes Feature, das auf die Persisitierung von Daten angewiesen ist, SOLL diese in einer *SQLite* oder *MariaDB* speichern können. Dabei sollte das bereits vorhande ORM verwendet werden.

4 Analyse und Umsetzung

In diesem Kapitel wird das praktische Vorgehen der Arbeit begleitet und die Ergebnisse dokumentiert. Jeder Abschnitt beschäftigt sich mit den Details und Anforderungen der spezifischen Thematik, nach welcher bereits die Anforderungen gruppiert wurden.

4.1 Usermanagement

Eine interne Repräsentation und Verwaltung von Benutzern ist Voraussetzung für Rechteverwaltung und Nutzung externer Identitäten in PhotoPrism. Daher ist der erste Schritt zu überprüfen, welche nutzbaren Interna die Anwendung hier bereits implementiert und diese den Anforderungen entsprechend zu erweitern.

4.1.1 Ist-Analyse

Public/Private Mode

Je nach Anwendungsfall, kann PhotoPrism mit oder ohne Authentifizierung betrieben werden:

- **Public-Mode:** In diesem Modus sind alle Funktionen und Inhalte der Instanz ohne Authentifizierung und Autorisierung zugänglich. Dieser Betriebsmodus ist nur für vertrauenswürdige Umgebungen, lokale Entwicklungsumgebungen oder für Demonstrations- und Evaluationszwecke geeignet.
- **Private-Mode:** Dieser Modus wurde eingeführt, um den Betrieb auch in nicht-vertrauenswürdigen Umgebungen (z.B. vom Internet aus zugänglich) zu ermöglichen. Inhalte und Funktionen sind nur einem *Admin* mit Passwort zugänglich. Mit diesem Modus wurden bereits erste Grundlagen für den Multi-User Einsatz geschaffen, die in den folgenden Abschnitten näher beschrieben werden.

User Datenstrukturen

Bereits früh wurde über Mehrbenutzer-Funktionalitäten und Datenstrukturen, die Personen und deren Beziehungen repräsentieren, nachgedacht. Vorbereitend wurden daher umfangreiche Datenstrukturen inklusive Datenbank Repräsentation implementiert, die jedoch bisher ausschließlich für anonyme Sharing-Links und zum Bereitstellen eines Logins für einen Standard Admin-Benutzer (nur im Privat Modus) verwendet werden. Ebenso gibt es schon grundlegende CRUD-Funktionen, um die Datenstrukturen zu persistieren. Um alle Anforderungen abdecken zu können, sind hier noch Erweiterungen notwendig.

Passwörter werden nach Anwendung des *bcrypt* Hash-Algorithmus in einer separaten Tabelle gespeichert.

Da diese schon sehr umfangreich ausgestaltet wurden, bieten sie bereits eine solide Grundlage, um das Usermanagement darauf aufzubauen.

Session-Mechanismus

Um einen minimalen Zugriffsschutz auf die Inhalte einer Instanz zu gewährleisten, wurde bereits ein individueller Session-Mechanismus mit Blick auf eine künftige Mehrbenutzerfunktionalität implementiert, die in Verbindung mit dem Standard-Admin-Benutzer einen Passwortschutz für den Zugriff auf Inhalte und Funktionen bietet.

Die Sessions werden darüber hinaus dazu verwendet Informationen über den Nutzer und spezielle Tokens zu speichern, die für die Sharing-Links benötigt werden. Die Tokens werden generiert, wenn beispielsweise ein Album geteilt wird und beim Zugriff auf eine Sharing-URL verwendet, um eine Gast-Session zu erstellen, falls nicht bereits eine Session existiert.

Um Brute-Force Angriffe zu erschweren, wurde ein einfaches Rate-Limiting implementiert, der für jeden Fehlversuch 5 Sekunden Verzögerung bei der Gültigkeitsprüfung des Passworts aufaddiert. Damit würden 100 Fehlversuche bereits über 7 Stunden in Anspruch nehmen, wenn sie sequentiell ausgeführt werden. Es ist jedoch möglich, Anfragen über die API direkt auch parallel auszuführen, womit in der selben Zeit sogar weit mehr als 5000 Versuche möglich wären. Zudem zwingt der Mechanismus Benutzer schon nach der ersten falschen Passwort-Eingabe unnötig lange zu warten.

Der Mechanismus tritt außerdem nur in Kraft, wenn ein vorhandener Benutzername übermittelt wird, da der Zähler für Fehlversuche an das Benutzerkonto geknüpft ist. Dies ermöglicht es einem Angreifer herauszufinden, ob ein Benutzerkonto überhaupt existiert.

Weiter wird der Benutzer serverseitig in der Session selbst zwischengespeichert, womit eine Löschung oder Änderung des Benutzers keine Auswirkungen auf die aktive Session hat, was aufgrund der langen Gültigkeit der Session auch nicht optimal ist.

Um Sessions clientseitg zu persistieren wird die Session-ID, ein zufälliger String mit 24 Zeichen, im *HTML5 LocalStorage* des Browsers gespeichert. Diese wird bei API-Requests durch den JavaScript-Client als HTTP-Header zur Authentifizierung übermittelt.

Bei der Nutzung des LocalStorage zu diesem Zweck wird häufig damit argumentiert, dies könne im Falle eines erfolgreichen XSS-Angriffes dazu führen, dass die Session gestohlen wird. Da PhotoPrism jedoch keine externen Skripte lädt, und Vue.js bereits Maßnahmen für den Schutz gegen solche Angriffe implementiert [42], ist hier zum Einen das Risiko einer Kompromittierung durch XSS relativ gering einzustufen (ein gewisses Restrisiko kann natürlich nie zu 100% ausgeschlossen werden, z.B. durch kompromittierte NPM-Packages). Und zum Anderen ist es einem Angreifer durch einen erfolgreichen XSS-Angriff trotzdem möglich die Session zu missbrauchen, und im Namen des eingeloggten Benutzers beliebige Aktionen auszuführen. Daher muss in diesem Kontext

Priorität haben, XSS-Angriffe zu verhindern, statt die Arbeit in die Optimierung des Austausches und der Persistierung der Session auf Client-Seite zu investieren. [5]

Ein weitere Auffälligkeit ist auch beim Abruf der Ressourcen zu finden, deren URLs direkt ins HTML-Markup eingebettet werden. Diese werden vom Browser automatisch abgerufen und können nicht ohne Umwege mit dem erforderlichen Session-Header ausgestattet werden. Dies trifft zum Beispiel auf alle Bilder zu, die von PhotoPrism ausgespielt werden. PhotoPrism verzichtet daher beim Abruf der Ressourcen bisher auf das Überprüfen der Berechtigung. Ein weiterer Grund für den Verzicht auf Autorisierung dieser Zugriffe ist, dass so Nutzer-Inhalte auch ohne größeren Aufwand via CDN verteilt werden können.

Die individuelle Implementierung des Session-Managements im Front- sowie im Backend und die zusätzliche Verwendung als Token- und Datenspeichermechanismus erhöhen den Aufwand einer Änderung hier enorm. Aus diesem Grund und weil die möglichen Gefahren hier im Nutzungskontext von PhotoPrism eher gering eingeschätzt wird, ist nach Abwägung entschieden worden, den Mechanismus vorerst weiter zu verwenden, womit R-U07 als erfüllt betrachtet werden kann.

Die Verbesserung der Sicherheit bleibt jedoch weiter ein wichtiges Ziel. Daher sollen Maßnahmen für die genannten Probleme der aktuellen Implementierung in die Empfehlungen im letzten Kapitel der Arbeit aufgenommen werden (siehe 5.1.4).

4.1.2 Umsetzung

Um das Usermanagement in einen nutzbaren Zustand zu bringen und die Basis für die OIDC Implementierung zu schaffen, soll PhotoPrism nun zunächst um Schnittstellen für die Administration der Benutzer erweitert werden. Diese werden zunächst in das CLI von PhotoPrism integriert, um die Verwaltung durch einen Server-Administrator zu ermöglichen. Das CLI benötigt keine umfassenden Schutzmaßnahmen und kein komplexes GUI, wodurch die Grundfunktionalität hier zügig hergestellt werden kann.

crud-Operationen

Im ersten Schritt wird das Erstellen, Bearbeiten, Löschen und Abfragen von Benutzerkonten (R-U01, R-U04 und R-U05) vorbereitet, indem Funktionen zur Interaktion mit der Datenbank und zur Validierung implementiert werden. Diese Programm-Logik ist im `entity`-Package zu finden.

- Validierung: Das `User`-Struct erhält eine `Validate()`-Methode[9], die Username und E-Mail (sofern vorhanden) auf Syntax und Einmaligkeit überprüft (siehe R-U01). Die syntaktische Prüfung des Passworts (Länge) wird bereits separat überprüft, da dieses von PhotoPrism als eigenständige Entität behandelt wird. Die erforderliche Validierung der Eingaben findet unmittelbar vor der Datenbank Anfrage statt.
- Zum Erstellen eines *Users* wird die `CreateWithPasswort()`-Funktion eingeführt, die das Erstellen eines Benutzers samt Validierung abstrahiert.[10]
- Funktionen für das Löschen und Abrufen einzelner Benutzer sind vorhanden und bedürfen keiner Änderungen. Lediglich für das Auflisten aller aktiver Benutzer wird eine neue Funktion benötigt.[11]
- Methoden für das Speichern von Änderungen nach Abruf an einer Entität, werden durch das von PhotoPrism verwendete ORM bereitgestellt und müssen nur noch um die Validierung erweitert werden.[12]

Zum Zeitpunkt der Anforderungsanalyse wurden die Anfordeungen des OWASP ASVS bezüglich Passwortsicherheit noch nicht berücksichtigt, daher werden diese in die Empfehlungen aufgenommen (siehe 5.1.3).

CLI-Befehle

Im zweiten Schritt werden die CLI-Befehle im dafür vorgesehenen `commands`-Package implementiert: Die Funktionen hier stellen die Ansprechbarkeit von der Kommandozeile (CLI) zur Logik her und stellen die für diese Schnittstelle spezifische Formatierung der Ausgaben bereit.

Für die Abstraktion des CLI-Interfaces nutzt PhotoPrism bereits `github.com/urfave/cli`[13] und für interaktive CLI-Abfragen stellt `github.com/manifoldco/promptui`[14] einheitliche Schnittstellen und ansprechendes Nutzungserlebnis bereit.

Die Befehle umfassen `photoprism users {list|add|update|delete}`. Erklärung der Befehle und Parameter können aus dem Code[15] abgeleitet werden oder werden im Abschnitt Benutzer-Dokumentation (siehe 4.1.3) im Detail erklärt.

[9] PhotoPrism Quellcode: `internal/entity/user.go L420-L460 [feature/oidc-v2]`
[10] PhotoPrism Quellcode: `internal/entity/user.go L462-L488 [feature/oidc-v2]`
[11] PhotoPrism Quellcode: `internal/query/users.go L7-L14 [feature/oidc-v2]`
[12] PhotoPrism Quellcode: `internal/entity/user.go L146-L153 [feature/oidc-v2]`
[13] `https://github.com/urfave/cli`
[14] `https://github.com/manifoldco/promptui`
[15] PhotoPrism Quellcode: `internal/commands/users.go [feature/oidc-v2]`

Benutzeroberfläche

Das Web-Frontend stellt bis dato noch keine Informationen bezüglich eines eingeloggten Benutzers bereit. Hierfür wurden neue UI-Komponenten implementiert, welche außer einem Logout-Button noch den Namen, die E-Mail und einen Platzhalter für ein eventuelles, zukünftiges Profilbild des aktuellen Benutzers im Menü anzeigt.[16]

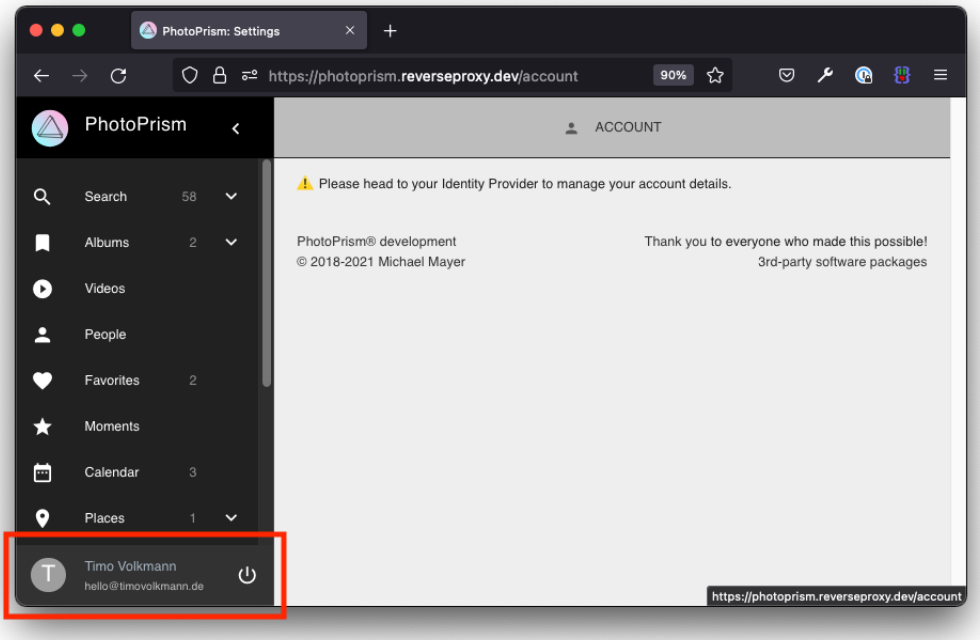

Abbildung 7: UI-Komponente für die Darstellung eines eingeloggten Benutzers.

Ein Klick auf das Element führt zu den benutzerbezogenen Einstellungen, die momentan nur aus einer Seite zum Ändern des Passwortes besteht. Der Logout-Button auf der rechten Seite des Elements beendet die aktuelle Session.

[16] PhotoPrism Quellcode: `frontend/src/component/navigation.vue L469-L486 [feature/oidc-v2]`

Testing

Automatisierte Test werden an dieser Stelle zunächst für die neuen Funktionen im `entity`-Package implementiert.[17] Da diese alle Interaktionen mit der Datenbank beinhalten, können hier direkt Integrationstests durchgeführt werden. Die dazu erforderlichen Testdaten werden entweder über Fixtures oder pro Testcase bereitgestellt.[18]

Die CLI-Befehle werden bisher noch manuell getestet. Da diese hier jedoch wenig eigene Logik beinhalten und im Grunde nur die Schnittstelle zu den `entity`-Funktionen bereitstellen, ist die Testabdeckung damit vorerst ausreichend. Die neue UI-Komponente wird durch Akzeptanztests abgedeckt und ist damit nicht Teil der Arbeit.

4.1.3 Benutzer-Dokumentation

Zur Benutzerverwaltung über die Kommandozeile stehen nun vier neue Befehle zur Verfügung. Durch CLI-Options lassen sich die Befehle parametrisieren. Das Hauptargument, um einen Benutzer eindeutig zu selektieren, ist der Username.

`$ photoprism users list`

Listet alle registrierten Benutzer tabellarisch auf. Angezeigt werden folgende Eigenschaften der Nutzer: *ID, Username, Voller Name, E-Mail, Admin-Rolle, Externer Benutzer*. Dieser Befehl besitzt keine Options oder Parameter.

`$ photoprism users add [options...]`

Erstellen eines neuen Benutzers. Dieser Befehl besitzt einen interaktiven Modus, um alle notwendigen Daten zum Anlegen eines Benutzers abzufragen. Der interaktive Modus wird automatisch deaktiviert, wenn beim Aufruf die beiden Options `--username` und `--password` übergeben werden. Gewählter Benutzername und E-Mail dürfen nicht bereits vergeben sein. Eine Liste aller möglichen Parameter kann der Tabelle 1 entnommen werden.

`$ photoprism users update [options...] [username]`

Benutzerdaten aktualisieren. Mit diesem Befehl können einzelne oder mehrere Eigenschaften des via `username` gewählten Benutzers überschrieben werden. Die Options entsprechen dem Befehl `photoprism users add`, ausgenommen des `username` (siehe Tabelle 1). Der Benutzername ist nicht änderbar.

`$ photoprism users delete [username]`

[17] PhotoPrism Quellcode: `internal/entity/user_test.go [feature/oidc-v2]`
[18] PhotoPrism Quellcode: `internal/entity/user_fixtures.go [feature/oidc-v2]`

Option	Abkürzung	Beschreibung
`--username <value>`	`-u <value>`	Obligatorisch. Setzt einmaligen Benutzernamen, der für den Login benötigt wird. Erfordert String aus mindestens vier Zeichen. Nur bei `photoprism users add`.
`--password <value>`	`-p <value>`	Obligatorisch. Setzt Login-Passwort. Erfordert String aus mindestens vier Zeichen.
`--fullname <value>`	`-n <value>`	Optional. Setzt einen Namen der statt dem Username in der Oberfläche angezeigt wird. Akzeptiert String als Wert.
`--email <value>`	`-m <value>`	Optional. Hinterlegt eine E-Mail Adresse bei dem angegeben Benutzer. Akzeptiert nur gültiges E-Mail Addressformat nach RFC5322 und RFC6532.
`--admin`	`-a`	Optional. Stattet den angegeben Benutzer mit Admin-Rechten aus.

Tabelle 1: Parameter der Befehle `photoprism users {add|update}`.

Mit diesem Befehl kann ein Benutzer anhand seines `username`s gelöscht bzw. deaktiviert werden. Eine interaktive Bestätigung verhindert ein versehentliches Löschen.

Das integrierte Usermanagement soll den Betrieb von PhotoPrism ohne die Abhängigkeit zu einem – oft komplexen – externen System ermöglichen. Insbesondere für den Einsatz mit erhöhtem Bedarf an Sicherheit, wie beispielsweise im Internet, ist der Einsatz eines IAM Systems empfohlen.

4.2 Rechteverwaltung

Dieser Abschnitt beschäftigt sich mit der Umsetzung der Rechteverwaltung in PhotoPrism. Auch hier wird der Ist-Zustand festgehalten, bevor die Implementierung geplant und durchgeführt wird. Tiefgreifende Änderungen am Autorisierungssystem sind zugunsten der OpenID Connect Integration im Rahmen der Arbeit nicht vorgesehen. Weitere

Schritte, insbesondere bezüglich des Themas *Data Authorization and Filtering*, werden im Ausblick nochmal thematisiert.

4.2.1 Ist-Analyse

Aktuelles Autorisierungsmodell

Ebenfalls wurden bereits Datenstrukturen zur Zugriffskontrolle implementiert, um die API bei Verwendung des *Private Modes* (siehe 4.1.1) vor unberechtigtem Zugriff zu schützen.

Auch die Benutzerrepräsentation von PhotoPrism sieht bereits verschiedenene vordefinierte Rollen vor[19]. Jedem Benutzer kann zu einem Zeitpunkt nur eine Rolle zugeordnet sein[20]. Für die Erfüllung von R-P03 muss daher nur noch die *Member*-Rolle definiert werden.

Das hierbei verwendete Modell basiert, entgegen der Package-Bezeichnung `acl`, auf Ideen des RBAC-Modells, wie es von [8] beschrieben wurde. In PhotoPrism kann die Berechtigung zum Durchführen einer *Aktion*, wie zum Beispiel „Lesen“ oder „Löschen“, bestimmten *Ressourcen*-Typen für Benutzer mit einer bestimmten *Rolle* erteilt werden. Dabei sind Rollen für den jeweiligen Zweck geeignete Kompositionen aus Aktionen. In Abbildung 8 werden diese Beziehungen veranschaulicht.

Die Berechtigungsprüfung findet dabei ausschließlich in den Handler-Funktionen der HTTP-API statt. Die Endpunkte der API sind so strukturiert, dass sie jeweils eine bestimmte Aktion auf einem bestimmten Ressourcen-Typen repräsentieren. Damit wird die Berechtigung für den Zugriff auf den Endpunkt selbst geregelt, auf Ebene der Daten selbst findet allerdings keine explizite Zugriffskontrolle statt. Da das vorliegende Modell also nur für *Domain Authorization* genutzt wird und momentan nur zwei Rollen unterschieden werden (Administrator und anonymer Gast), sind hier im Gegensatz zum RBAC-Modell von [8] keine Hierarchien oder Vererbungsstrukturen notwendig.

Ein konkretes Beispiel, wie dieses System bereits Anwendung findet, ist die Sharing-Funktion[21]. Hier werden einem anonymen Benutzer eingeschränkte Berechtigungen mittels `Guest`-Rolle zugewiesen, mit denen er berechtigt ist die notwendigen Endpunkte der API aufzurufen, um das mit ihm geteilte Album abrufen zu können. Dabei werden die zuvor erwähnten Tokens (siehe 4.1.1) verwendet, um eine Session zu erstellen, die den unauthentifizierten Benutzer dann zum Abruf der Daten über die API autorisiert. Damit auch nur die geteilten Bilder ensprechend des Sharing-Links abgerufen werden, wird nach der Autorisierung noch ein Such-Filter angewendet[22].

[19] PhotoPrism Quellcode: `internal/entity/user.go L60-L64 [feature/oidc-v2]`
[20] PhotoPrism Quellcode: `internal/entity/user.go L389-L416 [feature/oidc-v2]`
[21] `https://docs.photoprism.app/user-guide/share/share/`
[22] PhotoPrism Quellcode: `internal/api/search_photos.go L47-L62 [feature/oidc-v2]`

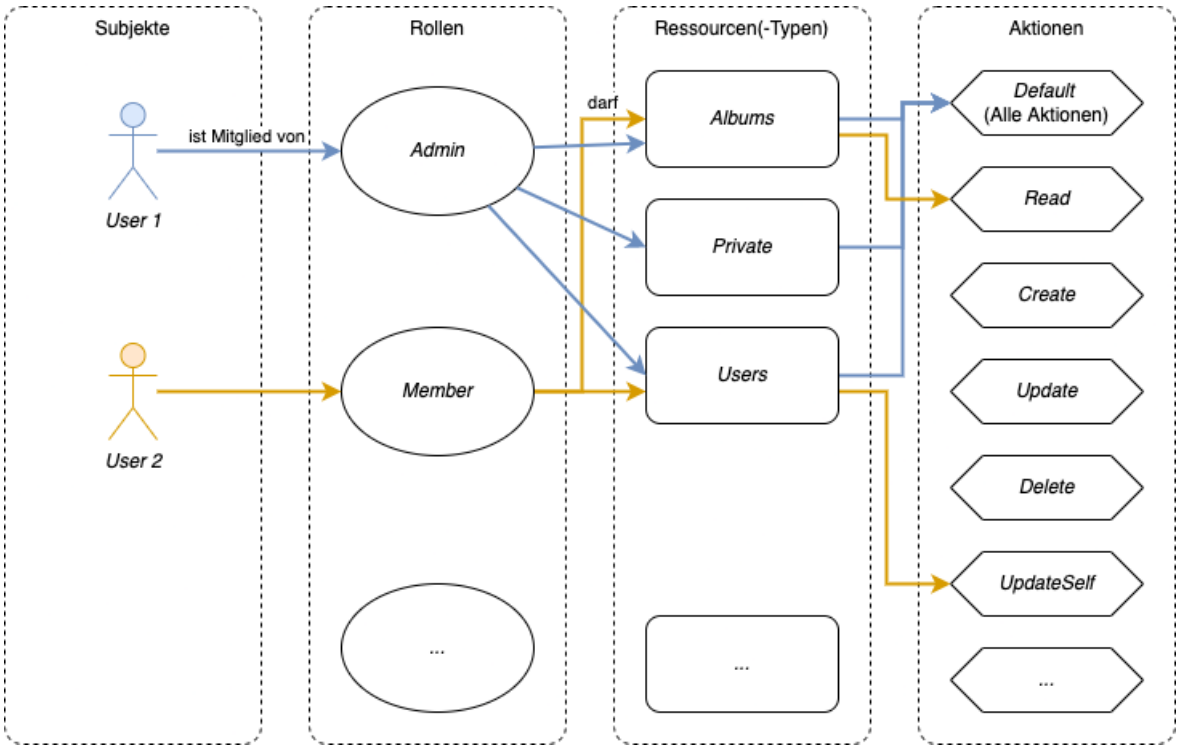

Abbildung 8: Schematische Darstellung des Rollenmodells von PhotoPrism.

Wie an diesem Beispiel zu sehen ist, werden auf Ebene der Daten (*Data Authorization and Filtering*) bisher lediglich an einigen Stellen gezielt die Suchanfragen für die `Guest`-Rolle eingeschränkt, damit auf Anfragen eines anonymen Benutzers nur die für Ihn bestimmten Daten zurückgeben werden.

Da diese Filtermechanismen nicht an jedem Endpunkt umgesetzt sind, ist es hier auch möglich, Inhalte oder Metadaten abzurufen, die nicht zum Sharing freigeben sind. Hierzu wird lediglich die `ID` der Ressource und eine gültige „anonyme“ Session benötigt, die man automatisch durch Aufrufen eines gültigen Sharing-Links ausgestellt bekommt. Bei den Endpunkten für Bilddaten gibt es überghaupt keine Zugriffskontrolle, wie schon bei der Ist-Analyse des Session-Managements aufgefallen ist (vgl. 4.1.1). Da hierdurch unter Umständen die Privatsphäre kompromittiert oder vertrauliche Informationen abfließen können, egal in welchem Maße, sollte hier über Alternativen nachgedacht werden. (siehe Empfehlung 5.1.2)

Man könnte diese Form von *Data Authorization and Filtering* hier konsequent so weiterführen, indem man bei jedem Endpunkt explizit die notwendigen Filter auf Basis von Attributen der angefragten Daten implementiert. Mit Blick auf die geplanten Erweiterungen ist dies jedoch wenig sinnvoll, da an dieser Stelle, durch Hinzufügen neuer Parameter (z.B. neue Rollen, Daten-Inhaberschaft, Hierarchien), eine signifikante Zunahme der Komplexität zu erwarten ist.

Hier sei der Vorschlag zu machen, das Rollenmodell auch auf den Bereich *Data Authorization and Filtering* auszuweiten und eine Einheitliche Schnittstelle für die Evaluation der Berechtigungen zu implementieren. Im Abschnitt 5.1.2 wird dieser Vorschlag näher erläutert. Für die Umsetzung im Rahmen der Arbeit werden die bestehenden Strukturen vorerst weiterverwendet, da eine komplette Überarbeitung hier zu viel Zeit in Anspruch nehmen würde. Für eine erste Iteration der Mehrbenutzer-Funktionalität mit Fokus auf OIDC ist das aktuelle System jedoch soweit ausreichend, dass durch kleine Modifikationen und das Ausweiten der Logik auf das Frontend ein sinnvoller Zustand erreicht werden kann.

Ungeachtet des Verbesserungspotentials, können durch die vorliegende Implementierung R-P01 und R-P02 bereits als erfüllt betrachtet werden.

Teilen von Alben mit Personen ohne Benutzerkonto

Die bereits erwähnte Sharing-Funktion versetzt Benutzer in die Lage, Alben per URL mit anderen Personen zu teilen, ohne ein Benutzerkonto bei der PhotoPrism Instanz vorauszusetzen. Dabei wird ein einzigartiger Sharing-Link generiert, der es allen Empfängern dieser URL ermöglicht, die Inhalte des geteilten Album anzusehen. Die Links können mit einer Gültigkeitsdauer versehen werden, nach deren Ablauf die Inhalte unter der URL nicht mehr abrufbar sind. Das Teilen kann auch manuell durch Löschen des Sharing-Links widerrufen werden. Dies erfüllt die Anforderung R-P07.

HTTP-Fehlercodes

Das Abweisen einer Anfrage wegen fehlender Berechtigung eines authentifizierten Benutzers wird momentan noch genauso behandelt, wie das Abweisen wegen fehlender oder abgelehnter Authentifizierung (wird mit dem HTTP Status-Code 401 quittiert) und hat im Frontend ein Beenden der Session zur Folge. Dieses Verhalten ist unerwünscht. Diese Thematik wird im Rahmen der Arbeit nicht weiter bearbeitet und wird dafür in die Empfehlungen aufgenommen (siehe 5.1.1).

4.2.2 Konzeption und Umsetzung

Zugunsten der OpenID Connect Integration und weil bisher noch Unklarheit bezüglich der technischen Umsetzung eigener Mediensammlungen für alle Benutzer einer Instanz herrscht (siehe R-P06), wird im Bereich der Rechteverwaltung nur das Notwendigste für eine sinnvolle erste Ausbaustufe implementiert. Für diese soll, ohne größere Modifikationen am Datenmodell, eine neue Rolle für Benutzer eingeführt werden, die über eingeschränkten Zugriff auf die zentrale Mediensammlung verfügt. Bestehende Strukturen sollen dabei zunächst weiterverwendet werden, um den Aufwand gering zu halten. Zusätzlich kann durch das Vorgehen in solchen kleinen Schritten, auch besser auf das Feedback und die Wünsche der Benutzer eingegangen werden.

Eine oft diskutierte Idee, um eigene Mediensammlungen (R-P06) für verschiedene Benutzer zu ermöglichen, ist der Zusammenschluss mehrerer Instanzen von PhotoPrism zu einem Cluster, bei der für jede eigene Mediensammlung eine eigene Instanz im Hintergrund läuft. Das Teilen eigener Inhalte ist hier über Replizierung bzw. einem dafür geeigneten P2P-Protokoll angedacht.
Dies mag zwar die Fähigkeit von PhotoPrism, gut mit sehr großen Sammlungen mehrerer Benutzer umgehen zu können verbessern, jedoch steigert dies die Komplexität des Systems immens. Auch lässt sich damit die Implementierung einer umfassenden Zugriffskontrollschicht nicht umgehen.

Weiter ist im Rahmen der Arbeit vorgesehen, das bestehende Rollenkonzept ins Frontend zu übernehmen, um sicherstellen zu können, dass die eingeschränkten Berechtigungen der *Member*-Rolle (R-P04) auch in der Darstellung der Benutzeroberfläche reflektiert werden. Weiter muss die Filterung der Daten für die neue Rolle angepasst werden und der Zugriff auf die WebDAV Schnittstelle eingeschränkt werden.

Propagierung von Berechtigungen ins Frontend

Um Zugriffskontrolle auch schon im Frontend anwenden zu können, ist entweder eine eigens dafür ausgelegte HTTP-Schnittstelle zu implementieren oder eine entsprechende clientseitige Logik zu implementieren und die notwendigen Informationen im Frontend bereitzustellen.

Da die Evaluation der Berechtigungen zum Rendern der Benutzeroberfläche notwendig ist, würde die Berechtigungsabfrage via HTTP-Schnittstelle durch Netzwerklatenzen bei vielen Anfragen eine spürbare Verzögerung des Seitenaufbaus verursachen und damit die Benutzererfahrung beeinträchtigen. Eine effizientere Lösung ist es daher, die Evaluation direkt im Frontend durchzuführen und bereits vorhandene Mechanismen zu nutzen, um die benötigten Informationen im Frontend vor dem Rendern schon verfügbar zu machen.

Während die Profilinformationen des Benutzers bereits mit der Session ans Frontend übertragen werden, sind die Berechtigungen bisher nicht im Frontend verfügbar. Um diese Informationen ins Frontend zu propagieren, eignet sich PhotoPrisms *WebSocket*-Schnittstelle, die bereits für Konfigurationsparameter[23] und anderen asynchronen Nachrichtenaustausch genutzt wird.

Die Berechtigungen sind bisher statisch und die Größe der Daten ist relativ gering, daher werden diese vollständig mit den restlichen Konfigurationsparametern des `ClientConfig`-Structs ans Frontend übertragen.[24] Da jede Aktion am Endpunkt der HTTP-API im Backend nochmals autorisiert werden muss, können durch böswillige Modifizierung der

[23] PhotoPrism Quellcode: `internal/api/websocket.go L67-L75 [feature/oidc-v2]`
[24] PhotoPrism Quellcode: `internal/config/client.go L70 [feature/oidc-v2]`

Daten hier höchstens versteckte UI-Bereiche sichtbar gemacht werden, jedoch keine Daten abgerufen oder verändert werden.

Da die Konfigurationsdetails im Frontend regelmäßig über diesen Kanal aktualisiert werden, kann hier später auch eine dynamische Rechteverwaltung darauf aufgebaut werden.

Bedingte Sichtbarkeit von UI-Komponenten

Die Evaluation der Berechtigungen im Frontend ist notwendig, um einem Benutzer nur die Bereiche des Userinterfaces anzeigen zu können, die seinen Berechtigungen entsprechen.

Um die Evaluation der Berechtigung direkt im Frontend ausführen zu können, wurde die Logik vom Backend[25] ins Frontend portiert.[26] Mittels *Mixin* [43] wird die zentrale Funktion zur Evaluation `hasPermission(resource, ...actions)` in allen Komponenten verfügbar gemacht und wird dort verwendet, um das Rendern von Teilen der Benutzeroberfläche zu entscheiden oder Interaktionselemente zu deaktivieren.

Da manche Komponenten durch mehrere *Aktionen* sichtbar oder aktiv sein können, wurde im Gegensatz zum Backend die Evaluationsfunktion hier dahingehend erweitert, dass gleich mehrere *Aktionen* als Parameter übergeben werden können und die Auswertung mit einer *ANY*-Semantik durchgeführt wird.

Filtern von Inhalten

Die Filterung der Inhalte entsprechend der Anforderung R-P04 auf Basis der Berechtigungen des anfragenden Nutzers ist geplant, konnte im Rahmen der zur Verfügung stehenden Zeit jedoch nicht mehr implementiert werden. Die Filterung soll zunächst durch Anpassung der Datenbankabfragen (`query`-Package) stattfinden. Hier ist jedoch zum späteren Zeitpunkt noch einmal zu prüfen, insbesondere für die Anforderungen R-P05 bis R-P10, ob hier nicht ein getrenntes und damit flexibleres Berechtigungs-System eingeführt werden muss, um die Relationen zwischen den Daten und Berechtigungen der Benutzer effizient und verständlich abbilden zu können. (siehe 5.1.2)

Einschränken des Zugriffs auf WebDAV

Die WebDAV Schnittstelle bietet zum jetzigen Zeitpunkt Lese- und Schreibzugriff auf die gesamte Mediensammlung. Da später noch eine Umstrukturierung stattfinden soll (R-P06 und R-P08) damit Benutzer auch eigene Sammlungen unterhalten können, wird bis dahin für alle Benutzer, die nicht der *Admin*-Rolle angehören, der Zugriff zu dieser

[25] PhotoPrism Quellcode: `internal/acl/acl.go [feature/oidc-v2]`
[26] PhotoPrism Quellcode: `frontend/src/common/acl.js [feature/oidc-v2]`

Schnittstelle komplett gesperrt.[27] Damit können sich übergangsweise nur Administratoren via WebDAV mit PhotoPrism verbinden.

In einer späteren Iteration soll der Zugriff über diese Schnittstelle den Berechtigungen und den Inhalten entsprechend erfolgen können.

4.2.3 Benutzer-Dokumentation

In dieser Phase des Ausbaus von PhotoPrism gibt es nun die neue *Member*-Rolle, welcher neu angelegte Benutzer standardmäßig angehören.

Da Benutzer momentan keine eigenen privaten Mediensammlungen unterhalten können und Inhalte einer Instanz damit grundsätzlich für alle Benutzer zugänglich sind, wird die *Member*-Rolle grundsätzlich nur lesende Rechte erhalten und ausgewählte Inhalte (siehe R-P04) sollen dieser Rolle verborgen bleiben.

Ausgenommen vom Schreibverbot sind die eigenen Profilinformationen und das Passwort, wobei für ersteres noch keine HTTP-Schnittstelle existiert. Diese Informationen können aber per CLI abgerufen und geändert werden (siehe 4.1.3). Das Passwort kann über das Frontend geändert werden, wie im linken Teil von Abbildung 10 zu sehen ist.

4.3 Single-Sign-On (SSO)

Um Authentifizierung und Usermanagement an einen externen Identity Provider zu delegieren, soll OpenID Connect aufgrund der breiten Unterstützung von Drittanbietern und der vergleichsweise einfachen Handhabung und Integration in PhotoPrism zum Einsatz kommen.

In diesem Abschnitt wird das Konzept für die Integration vorgestellt und die Umsetzung begleitet.

OpenID Connect ist eine umfangreiche Protokollsuite und kann auf verschiedene Weise und in unterschiedlichem Umfang integriert werden. Daher wird Teil dieses Abschnitts sein, die benötigten Komponenten zu identifizieren, um eine möglichst sichere und einfach zu handhabende Integration gewährleisten zu können.

4.3.1 Konzeption

In der Terminologie von OpenID Connect wird PhotoPrism die Rolle der *Relying Party* einnehmen und damit die Hoheit über das Usermanagement an einen externen IdP abtreten können. *OpenID Provider* Funktionalität wird in PhotoPrism selbst nicht benötigt. Die für den Betrieb als RP benötigten Mindestanforderungen leiten sich hauptsächlich

[27] PhotoPrism Quellcode: `internal/server/auth.go L63 [feature/oidc-v2]`

aus den OAuth Spezifikationen ab. Die Wahl der zu verwendenenden Features aus der OIDC Spezifikation bleibt zu großen Teilen dem Entwickler überlassen. [37, Kapitel 15.4]

Auch hier ergibt es Sinn, angesichts begrenzter Ressourcen und zugunsten einer schnellen Einführung des Features, eine stufenweise Integration anzustreben. Wegen des Risikos von Sicherheitslücken, soll nach Möglichkeit die Implementierung der komplexen Spezifikationen aus eigener Hand vermieden werden.

Um die Grundfunktionalität von OIDC herzustellen, muss dann ein geeigneter OAuth 2.0 Flow ermittelt werden, der zur Architektur von PhotoPrism passt. Da es möglich ist die RP sowohl im Frontend als auch im Backend zu implementieren, ist dies notwendig bevor geeignete Libraries ausgewählt werden können.

Grundfunktionalität

In einem ersten Schritt soll der Fokus auf der Authentifizierung liegen. Zusätzliche Features, wie die Rollenzuweisung über den IdP, können später hinzugefügt werden.

Weiter wird die Einrichtung und Konfiguration möglichst leicht gestaltet, damit möglichst viele Benutzer dieses Feature auch nutzen und nicht an der Hürde einer zu komplizierten Konfiguration scheitern. Dazu bietet *OpenID Connect Discovery* den passenden Mechanismus, der es erlaubt Konfigurationsdetails automatisch vom OP abzurufen. Dazu muss lediglich die *Issuer*-URL bereitgestellt werden. [38, Kapitel 4]

WebFinger, welches für *OpenID Provider Issuer Discovery* notwendig ist [38, Kapitel 2], wird hier nicht verwendet, da es bisher keine Pläne für die Unterstützung mehrerer IdPs gibt und die mit dem zusätzlichen Protokoll einhergehende Komplexität den Nutzen zum jetzigen Zeitpunkt übersteigt.

Bei erstmaliger erfolgreicher Authentifizierung beim IdP muss PhotoPrism eine eigene Repräsentation des Benutzers aus den Informationen des ID-Tokens oder des *UserInfo*-Endpunktes anlegen, damit PhotoPrism intern Inhalte zuordnen kann und die Informationen bei Bedarf lokal bereitstehen. Die Profilinformationen müssen mit jedem Login beim IdP aktualisiert werden, damit Inkonsistenzen zeitnah beseitigt werden können.

Da PhotoPrisms internes Session Management in naher Zukunft noch überarbeitet werden sollte (siehe 5.1.4), wird zunächst noch darauf verzichtet die Sessions an den Ablauf der Gültigkeit der ID-Tokens zu koppeln und durch eine stille Reauthentifizierung zu erneuern. [26] Stattdessen wird nach erfolgreicher Authentifizierung beim IdP eine lokale Session gestartet, wie sie auch bei lokaler Authentifizierung genutzt wird. Durch die regelmäßige Reauthentifizierung beim IdP wird auch eine häufigere Aktualisierung der Profilinformationen des externen Benutzers sichergestellt.

Technische und sicherheitsrelevante Details

Da nun der Umfang für die Erstimplementierung feststeht, können die technischen Details für die Integration festgelegt werden. Dies betrifft insbesondere die Zielumgebung für die RP (Implementierung im Frontend oder Backend) und die OAuth Flows, welche die Kommunikation zwischen OP und RP regeln. Diese machen den Kern der Spezifikation aus und haben einen maßgeblichen Einfluss auf die Sicherheit der Integration.

Die OAuth-Spezifikationen werden unabhängig von OpenID Connect gepflegt und stetig verbessert. Erweiterungen wie PKCE können die Sicherheit des Protokolls verbessern, obwohl diese in den OIDC Spezifikationen keine Erwähnung finden. Aus diesem Grund ist es sinnvoll zu prüfen, ob eine Berücksichtigung von Erweiterungen hier Mehrwert generiert und ob es die Kompatibilität zu externen OPs beeinträchtigt.

OAuth hat mit dem *Implicit Grant* und dem neueren *Authorization Code Grant mit PKCE* eine Möglichkeit geschaffen, den Client (bzw. RP) direkt in SPAs zu integrieren. Da PhotoPrisms Frontend als solche betrachtet werden kann und die Aussicht darauf, dass die Token-basierte Authorisierung von OAuth für den Zugriff auf PhotoPrisms APIs hier den bereits vorhandenen Session-Mechanismus ersetzen könnte, lässt die Integration im Frontend zunächst als passende Lösung erscheinen. Im Detail betrachtet, sprechen jedoch einige Punkte dagegen:

- Secrets können in clientseitigen Anwendungen weniger gut geschützt werden: Beide genannten Flows werden daher ohne *Client Secret* eingesetzt. Ohne *PKCE* ermöglicht dies dritten Akteuren, sich vor dem IdP als legitime RP auszugeben. Damit könnten bösartige Anwendungen *Authorization Codes* während des Flows abfangen und selbst gegen gültige Token tauschen oder den Token direkt abfangen, wenn er mit dem *Implicit Grant* via URL übertragen wird. Mit der *PKCE*-Erweiterung für den *Authorization Code Grant* kann dieser Flow nun auch für *Public Clients* eingesetzt werden, da der *Authorization Code* nun nicht mehr von Dritten eingelöst werden kann. [35, Kapitel 1]

  PKCE sollte jedoch keine zwingende Voraussetzung für die Benutzer sein, um die Kompatibilität zu möglichst vielen IdPs gewährleisten zu können. Damit bleibt das Abfangen von Token/Authorization Code ein mögliches Risiko für *Public Clients*.

- Die Gefahr, dass Token gestohlen oder missbraucht werden können (z.B. durch XSS oder CSRF), ist in clientseitigen Anwendungen größer als wenn diese von einer serverseitigen Anwendung verwaltet werden. Da mit diesen Token potentiell auch andere Anwendungen oder der IdP selbst kompromittiert werden könnten, ist das Risiko hier größer zu bewerten als bei Sessions die nur auf die Anwendung selbst beschränkt sind. Insbesondere Refresh-Token sind durch ihre lange Gültigkeitsdauer zusätzlich gefährdet.

- Beim *Implicit Grant* wird der statt dem *Authorization Code* direkt der Token über die Redirect-URL ausgeliefert. Damit ist diese Methode anfällig dafür, dass der Token abgefangen werden kann. Aus diesem Grund wurde dieser Flow auch

komplett aus der Neuauflage von OAuth 2.1 entfernt und soll nicht mehr verwendet werden. [16, Kapitel 10] Die bekannten Angriffsvektoren werden auch in *OAuth 2.0 Security Best Current Practice* beschrieben.

- Implementierungsaufwand bzw. späterer Wartungsaufwand ist für diese Variante deutlich größer: Entweder muss der Session-Mechanismus komplett mit einem neuen tokenbasiertem System ersetzt werden oder die APIs müssen zusätzlich zu den Sessions eine weitere Autorisierungsmethode akzeptieren.

Stattdessen wird empfohlen, die Rolle des Clients (RP) im Backend zu implementieren und die Kommunikation zwischen Front- und Backend mittels eigenem Session-Mechanismus (Cookie-basiert mit den Attributen `secure` und `HttpOnly`) abzusichern. Dadurch ist es möglich die RP als *Confidential Client* mit *Client Credentials* auszustatten und den sichereren *Authorization Code Grant* zu verwenden. Weiter wird empfohlen, PKCE für alle Szenarien einzusetzen. [29, Kapitel 6.1 und 6.2]

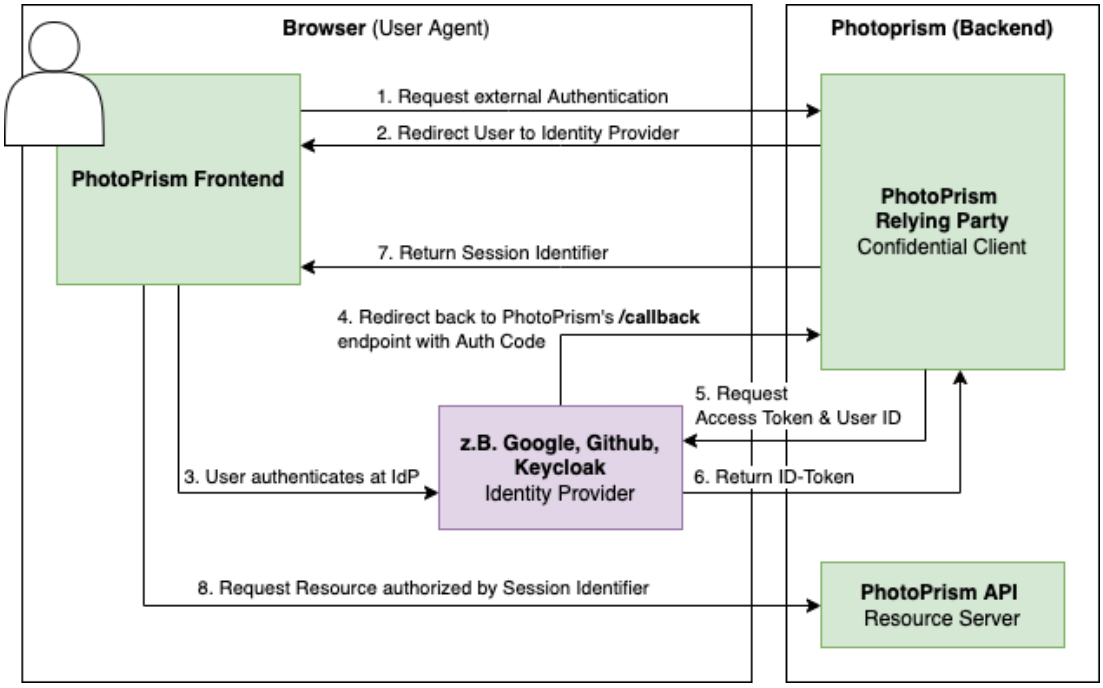

Abbildung 9: Ablauf der externen Authentifizierung in PhotoPrism via OIDC.

Auf Basis dieser Überlegungen und den technischen Anforderungen des OIDC-Protokolls ergibt sich für PhotoPrism der in Abbildung 9 gezeigte Ablauf für die Authentifizierung durch OpenID Connect.

OpenID Connect Bibliotheken

Mit dem Festlegen der Architektur kann sich nun ein Überblick über zur Verfügung stehender Bibliotheken gemacht werden. Es gibt einige umfangreiche Kandidaten für Go, die

eine integrierte Lösung mit Session-Management und Oauth2 bzw. OpenID Connect Unterstützung anbieten.[28] Diese sind jedoch nicht kompatibel mit dem vorhandenen Session Management und würden deshalb ein größeres Refactoring notwendig machen. Weiter ist bei diesen nicht ohne nähere Analyse des Quellcodes festzustellen, ob die Anforderungen damit umsetzbar sind und ob Sicherheitsaspekte ausreichend Berücksichtigung finden. Zudem setzen ausnahmslos alle der Bibliotheken auf Cookies. Glücklicherweise gibt es auch spezialisierte OAuth bzw. OpenID Connect Bibliotheken für Go, die sich im Kern auf die Implementierung der Spezifikationen beschränken.

- `golang.org/x/oauth2`

  Ist Teil der Go Standard Library und enthält eine Client Implementierung der OAuth 2.0 Spezifikationen [15]. Diese kann zum Umsetzen einer OpenID Connect RP benutzt werden, erfordert aber, OIDC-spezifische Teile des Protokolls selbst zu implementieren. PKCE muss ebenfalls selbst hinzugefügt werden. Hat von den drei Optionen die größte Verbreitung (gemessen an Sternen und Forks auf Github. Keine representative Aussage).

- `github.com/coreos/go-oidc/v3/oidc`

  Erweitert `golang.org/x/oauth2` um OpenID Connect RP Funktionalität. Bringt ebenfalls kein PKCE mit. Keine Informationen über berücksichtigte Spezifikationen.

- `github.com/caos/oidc`

  Kann für OP und RP verwendet werden. Die Implementierung der RP wurde von der OpenID Foundation zertifiziert.[29] PKCE und andere Erweiterungen werden unterstützt. Die Bibliothek bringt bereits *HTTP-Handler* für eine sicherere Integration mit und setzt im Kern ebenfalls auf die OAuth Implementierung der Standard Library. Ist in der Auswahl die jüngste Bibliothek und hat noch keine große Verbreitung, wird aber von deren Team für ihre Cloud-Identity-Plattform[30] genutzt und scheint damit praxistauglich zu sein. Die OP Implementierung ist darüber hinaus nützlich, um eine Umgebung für Integrationstests zu schaffen. Dies ist somit die vollständigste Implementierung, daher wird diese Bibliothek für die Integration in PhotoPrism eingesetzt.

Weitere Iterationen

Für zukünftige Iterationen sind zum jetzigen Zeitpunkt noch weitere Erweiterungen der OpenID Connect Integration geplant oder sollen evaluiert werden:

[28] `https://github.com/volatiletech/authboss`,
`https://github.com/abraithwaite/jeff`,
`https://github.com/markbates/goth`
[29] `https://openid.net/developers/certified/`
[30] `https://github.com/caos/zitadel`

- **Rollenmanagement via IdP:**

  Als nächstes Ziel nach dem Implementieren der Grundfunktionalität soll es möglich gemacht werden, externen Benutzern direkt über den IdP Rollen für PhotoPrism zuzuweisen. Dies kann mittels Scopes und Claims umgesetzt werden. Die Zuordnung folgt dabei einem festgelegten Muster und wird im Abschnitt Umsetzung bzw. Dokumentation im Detail erklärt. Dieses Feature bringt aus Nutzersicht zunächst den größten Mehrwert bei überschaubarem Aufwand und konnte daher noch im Rahmen der Arbeit implementiert werden (siehe 4.3.2). Weitere Iterationen in diesem Punkt sind jedoch nicht ausgeschlossen.

- **OpenID Connect Session Management:**

  Dieser Punkt setzt die Überarbeitung des Session Managements in PhotoPrism voraus und hat hohe Priorität, insbesondere für nicht-vertrauenswürdige Umgebungen. Die Erweiterungs-Spezifikation OpenID Connect Session Management beschreibt eine Methode, die Session eines Benutzers beim RP an die Session bei einem OP zu koppeln. Dies geschieht durch regelmäßiges Überprüfen der Session durch Wiederholen der Authentifizierungsanfrage im Hintergrund mit dem Parameter `prompt=none` oder durch ein unsichtbares *iFrame* welches Zugriff auf das Session Cookie des OPs hat und mittels *Polling* Änderungen am Authentifizierungsstatus des Benutzers beim IdP an den RP melden kann. [26, Kapitel 3] Das Wiederholen der Authentifizierungsanfrage kann darüber hinaus dazu genutzt werden Änderungen an den Profilinformationen des Benutzers zu synchronisieren. Durch den größeren Aufwand, insbesondere durch Verknüpfung mit PhotoPrisms Session-Mechanismus, wird dies nicht mehr im Rahmen der Arbeit implementiert werden.

- **Dynamic Client Registration:**

  Die *Dynamic Client Registration* [36] ist eine Erweiterung der OIDC Core Spezifikation und erlaubt es der RP sich automatisch bei einem OP zu registrieren. Damit lässt sich der manuelle Konfigurationsaufwand der RP reduzieren. Hier soll evaluiert werden inwiefern dies für PhotoPrism von Mehrwert sein kann. Sicherheitsaspekte sollen berücksichtigt und mit Usability abgewogen werden. Dieser Punkt fällt nicht mehr in den Rahmen der Arbeit und hat niedrige Priorität.

4.3.2 Umsetzung

Dieser Abschnitt beschreibt die Maßnahmen zur Umsetzung der Integration von OIDC. Im OWASP ASVS sind Sicherheitsaspekte zu externer Authentifizierung unter Level 3 aufgeführt, womit diese im Rahmen der Arbeit noch nicht explizit berücksicht werden. Allerdings werden protokollspezifische Sicherheitsüberlegeungen aus den Spezifikationen von OIDC und OAuth berücksichtigt, sofern diese nicht schon durch die Drittanbieter-Bibliothek abgedeckt werden. [37, Kapitel 16], [15, Kapitel 10], [29]

Konfiguration

Um OpenID Connect einrichten zu können, müssen die globalen Konfigurations-Optionen um zusätzliche Parameter erweitert werden. Diese Datenstruktur enthält alle Variablen mit denen PhotoPrisms globale Konfiguration gesteuert wird. Folgende Variablen werden für OpenID Connect hinzugefügt[31]: *Issuer*-URL, *Client ID*, *Client Secret*, *Scopes* und *Site URL* (siehe auch Tabelle 3).

Es gibt zwei Wege, die Konfigurationsparameter der Anwendung zu übergeben: Als CLI-Option und als Umgebungsvariable. Die Bibliothek `github.com/urfave/cli` stellt die hierfür notwendige Funktionalität bereit.

Das `oidc`-Package

Die Initialisierung der OpenID Connect Bibliothek `github.com/caos/oidc`, die Adaption der davon bereitgestellten API für PhotoPrism und die Implementierung spezifischer Hilfsfunktionen werden durch das neue `oidc`-Package[32] bereitgestellt.

Die Funktion `NewClient()`[33] ist dabei für die dynamische Initialisierung und Konfiguration des Clients zuständig und prüft, ob eine Verbindung zum OP hergestellt werden kann. Der genaue Ablauf wird in der folgenden Grafik dargestellt. Die OIDC-Funktionen werden über das `service`-Package aufgerufen. Die Initialisierung wird dort nach Bedarf eingeleitet.[34]

Weiter werden die von der Bibliothek bereitgestellten HTTP-Handler konfiguriert und an PhotoPrisms Web-Framework angepasst. Diese werden dann durch das `oidc`-Package exportiert:

- Der `AuthCodeUrlHandler()`[35], stellt einen Wrapper für die entsprechende Funktion der OIDC-Bibliothek für den *Authentication Request* [37, Kapitel 3.1.2.1] bereit, welcher an die Signatur der Handler-Funktion, des von PhotoPrism verwendeten Web-Frameworks `github.com/gin-gonic/gin`, angepasst ist. Aufgabe dieses Handlers ist es, den Redirect zur Initialisierung des *Authorization Code Grants* zu generieren. Um den empfohlenen *State*-Parameter [37, Kapitel 3.1.2.1] bereitstellen zu können, muss eine Funktion übergeben werden, die einen zufälligen Wert generiert. Dieser wird benötigt, um den späteren Callback dem *Authentication Request* wieder zuordnen zu können. Außerdem kann er, durch das Setzen eines Cookies mit diesem Wert im Browser, dazu verwendet werden CSRF-Angriffe zu verhindern. [15, Kapitel 10.12]

[31] PhotoPrism Quellcode: `internal/config/options.go L43-L46 [feature/oidc-v2]`
[32] PhotoPrism Quellcode: `internal/oidc/ [feature/oidc-v2]`
[33] PhotoPrism Quellcode: `internal/oidc/client.go L36-L105 [feature/oidc-v2]`
[34] PhotoPrism Quellcode: `internal/service/oidc.go L19-L24 [feature/oidc-v2]`
[35] PhotoPrism Quellcode: `internal/oidc/client.go L111-L114 [feature/oidc-v2]`

- Der `CodeExchangeUserInfo()`-Handler[36] stellt den Endpunkt für die *Authentication Response* [37, Kapitel 3.1.2.5] des OPs bereit und leitet im Falle einer erfolgreichen Authentifizierung beim OP den Austausch des *Authorization Codes* gegen Token und Profilinformationen ein. Diese Funktion stellt ebenfalls einen Wrapper für `github.com/gin-gonic/gin` dar, bei dem jedoch im Erfolgsfall direkt die Profilinformationen des Benutzers zurückgegeben werden, welche zum jetzigen Zeitpunkt die einzig benötigten Daten im weiteren Verlauf darstellen. Das gesamte Token-Handling inklusive Validierung wird von `github.com/caos/oidc` intern durchgeführt, daher werden die Token vorerst nicht benötigt und müssen nicht weiterverarbeitet werden.

Um Unregelmäßigkeiten verschiedener IdPs bei der Definition des Usernames vorzubeugen, wird vom `oidc`-Package noch eine Hilfsfunktion bereitgestellt, die aus verschieden Feldern der Profilinformationen eines Benutzers einen, den Kriterien von PhotoPrism enstprechenden, Usernamen auswählt.[37]

HTTP-Schnittstellen und Integration im Frontend

Die beiden im vorigen Abschnitt genannten Handler werden durch zwei neue HTTP-Endpunkte in den Webserver integriert und damit über HTTP ansprechbar gemacht.[38] Die Endpunkte werden nur aktiviert, wenn beim Start der Anwendung ein OP konfiguriert ist (siehe 4.3.3) und eine Verbindung zum Discovery-Endpunkt hergestellt werden kann.

Im Callback-Endpunkt werden nach erfolgreichem Abschluss des OIDC-Flows noch die Profilinformationen des Benutzers weiter verarbeitet: Der Benutzer wird erstellt oder seine Profilinformationen geupdated. Danach wird mit diesen Informationen eine Session erstellt, deren Identifizierer mit Profil- und Konfigurationsinformationen als *Response* des Handlers wieder zum Browser des Benutzers gesendet wird, wo diese dann vom Frontend entgegen genommen werden müssen.[39]

Damit das Frontend mit dem neuen redirect-basierten Login-Flow umgehen kann, ist zusätzlich die Anpassung der Login-Komponente des Frontends notwendig.

1. **Der Login-Flow im Frontend:**
   Im ersten Schritt (vgl. Abbildung 9) muss der Benutzer den Flow einleiten. Dies kann auf eine der nachfolgend beschriebenen Weisen erfolgen. Die Login-Komponente von PhotoPrism ruft dann den Endpunkt `/api/v1/auth/external` auf, woraufhin im zweiten Schritt der Redirect im Backend generiert wird und die Login-Seite des IdPs im Browser geöffnet wird.

[36] PhotoPrism Quellcode: `internal/oidc/client.go L116-L146 [feature/oidc-v2]`
[37] PhotoPrism Quellcode: `internal/oidc/helper.go L16-L29 [feature/oidc-v2]`
[38] PhotoPrism Quellcode: `internal/api/auth.go [feature/oidc-v2]`
[39] PhotoPrism Quellcode: `internal/api/auth.go L38-L78 [feature/oidc-v2]`

a) **Manuelle Initialisierung durch Klick auf den Button „Login with OpenID Connect“:**[40] Hier wird, im Gegensatz zur automatischen Weiterleitung, die externe Login-Seite in einem neuen Tab geöffnet. Dadurch kann das Neuladen des gesamten Frontends nach Abschluss der Authentifizierung vermieden werden. Diese Variante nutzt die *Storage Events* [46] des Browsers, um einen Kommunikationskanal zwischen den beiden Tabs herzustellen und auf den erfolgreichen Login oder gegebenfalls Fehler reagieren zu können. Dazu registriert das PhotoPrism-Frontend einen *Event-Handler* im Browser der bei Änderungen im *LocalStorage* ausgeführt wird und den Login-Prozess mit dem automatischen Schließen des Login-Tabs abschließen kann.

b) **Automatische Weiterleitung auf die externe Login-Seite:** Diese Variante wurde hinzugefügt, um die Benutzererfahrung zu verbessern, nachdem die Entscheidung getroffen wurde, Benutzer nur aus einer Quelle zuzulassen (siehe R-S01, Absatz 2). Die für die manuelle Initialisierung entwickelten Mechanismen wurden hier soweit möglich weiterverwendet.

   Die automatische Weiterleitung wird nur ausgeführt, wenn der IdP innerhalb einer Sekunde antwortet und es im Vorfeld keine unvollendeten oder fehlgeschlagenen Loginversuche gegeben hat.[41] Der automatische Redirect kann auch durch den URL-Query-Parameter `preventAutoLogin=true` unterbunden werden. Diese Maßnahmen sind notwendig, um Benutzer nicht auszuschließen, falls es Probleme mit dem IdP gibt oder der Admin sich (als einzige Ausnahme) lokal anmelden möchte.

   Da der Redirect bei dieser Variante ohne explizite Interaktion durch den Benutzer stattfindet, wird dieser beim Öffnen in neuem Tab oder Fenster in manchen Browsern als ungewolltes Popup blockiert. Um das Blockieren der Login-Seite zu vermeiden, wird der Redirect hier im aktiven Tab ausgeführt. Dies hat jedoch den Nachteil, dass PhotoPrisms Frontend nach der Authentifizierung beim IdP erneut geladen werden muss und der Benutzer damit länger warten muss, bis der Login-Prozess abgeschlossen ist. Im Gegensatz zur ersten Variante kann hier die *storage event* Callback-Funktion nicht verwendet werden, da dafür die Login-Seite von PhotoPrism in einem zweiten Browsertab parallelen geöffnet sein muss. In diesem Fall triggert ein weiterer Redirect[42] zurück zu PhotoPrisms Login-Seite den Abschluss des Login-Flows, indem der LocalStorage beim Laden der Login-Komponente direkt auf eine vorhandene Session überprüft wird. Ist diese vorhanden, werden die Inhalte geladen und der Benutzer ist eingeloggt.[43]

2. **Handling des OAuth-Callbacks:**
   Sobald der Login beim IdP erfolgt ist, leitet dieser einen Redirect zurück zu Photo-

[40] PhotoPrism Quellcode: `frontend/src/pages/login.vue L135-L155 [feature/oidc-v2]`
[41] PhotoPrism Quellcode: `frontend/src/pages/login.vue L93-L102 [feature/oidc-v2]`
[42] PhotoPrism Quellcode: `assets/templates/callback.tmpl L51 [feature/oidc-v2]`
[43] PhotoPrism Quellcode: `frontend/src/pages/login.vue L118-L119 [feature/oidc-v2]`

Prism ein. Dieser Callback-Endpunkt ist unter dem URL-Pfad `/api/v1/auth/external` der Installation zu erreichen. Da es ineffizient ist, PhotoPrisms Frontend beim zweiten Schritt durch den Callback des IdPs neu laden zu müssen, wird für den Redirect zurück zu PhotoPrism eine leichtgewichtige HTML-Seite generiert, die nur den Zweck hat, ein kleines Skript zu transportieren, der alle relevanten Informationen für den Login des Benutzers nach erfolgreichem Abschluss des OIDC-Flows in den LocalStorage schreibt[44]. Das Frontend kann dann zu jeder Zeit auf diese zugreifen. Wenn das Frontend bereits in einem anderen Tab läuft, kann mit den bereits erwähnten *Storage Events* des Browsers der Login-Flow automatisch abgeschlossen werden.

Folgende Parameter werden in den LocalStorage geschrieben:

a) `session_id`: Enhält den Session Identifier. Entspricht dem Speicherort des aktuellen Session Managements.

b) `data`: Enthält Informationen über den eingeloggten Benutzer, die vom Frontend angezeigt werden können. Entspricht ebenfalls der aktuellen Implementierung des Session Managements im Frontend.

c) `config`: Enthält Informationen über PhotoPrisms Konfiguration, aktivierte Features und Berechtigungen aller vorhandenen Rollen.

d) `auth_error`: Wird nur im Fehlerfall gesetzt. Enthält eine spezifischere Fehlermeldung, die dem Benutzer im Frontend angezeigt wird.

Da das Passwort für externe Benutzer nicht mehr von PhotoPrism verwaltet wird, muss das Formular zum Ändern des Passworts, die bisher einzige UI-Komponente für profilbezogene Einstellungen, für externe Benutzer gesperrt werden. Wie in Abbildung 10 zu sehen, wurde es mit dem Hinweis ersetzt, Änderungen an Profilinformationen beim IdP direkt durchzuführen.

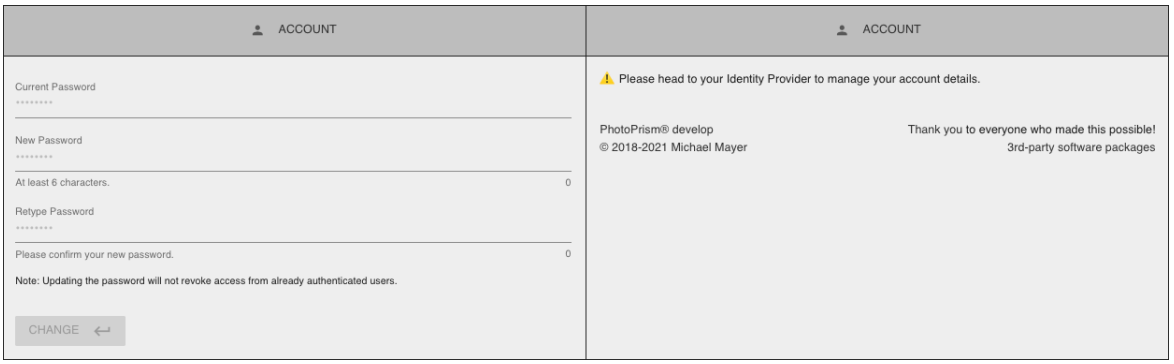

Abbildung 10: UI-Komponente für profilbezogene Einstellungen für interne (links) und externe Benutzer (rechts).

[44]PhotoPrism Quellcode: `assets/templates/callback.tmpl L45-L56 [feature/oidc-v2]`

Erweiterung um Rollenmanagement

Für die Verwaltung der Rollen über den IdP bietet sich das Konzept der *Scopes* von OAuth an (siehe 2.4.2).

Es ist möglich den benötigten Claim einem der Standard-Scopes von OIDC zuzuordnen, womit auch ohne zusätzlich konfiguriertem Scope in PhotoPrism die Rolleninformationen übermittelt werden können. Damit Administratoren die Zuordnung selbst steuern können und diese damit den Bedürfnissen Ihrer Anwendungslandschaft anpassen können, wird PhotoPrism die Möglichkeit bieten, über die globale Konfiguration zusätzliche Scopes zu definieren (vgl. Abschnitt Konfiguration), die während des Login-Prozesses dann beim IdP angefragt werden. Die Scopes werden bei der Initialisierung des `oidc`-Packages übergeben[45]. Die Nutzung benutzerdefinierter Scopes setzt die Einrichtung der entsprechenden Scopes und die Zuordnung der Claims beim IdP voraus.

Zum jetzigen Zeitpunkt werden in PhotoPrism nur die Rollen *Member* und *Admin* aktiv verwendet. Standardmäßig bekommen neue externe Benutzer nur Berechtigungen der *Member*-Rolle. Damit die Rolleninformationen eines Benutzers korrekt aus dem Scope extrahiert werden können und der Administrator des IdPs die Mappings der Claims dort richtig konfigurieren kann, wird hierfür eine neue Konvention für die Benennung des erwarteten Claims für die Rolle und dem Format des Wertes eingeführt:

1. Der Claim-Schlüssel muss folgenden Bezeichner haben: `photoprism_role`
2. Das Format des Claim Werts muss entweder ein *String* oder eine Liste aus *Strings* sein. Zu beachten ist hier, dass PhotoPrisms Benutzern momentan nur eine Rolle zugeordnet werden kann. Da beim IdP jedoch meist mehrere Rollen zugeordnet werden können, kann die Unterstützung für Listen von Werten dem Administrator hier die Einrichtung erleichtern.
3. Akzeptierte Claim-Werte sind die Rollenbezeichner mit dem Präfix `photoprism_`. Alle zum jetzigen Zeitpunkt möglichen Werte werden in folgender Tabelle dargestellt. Durch die Konvention lässt sich die Tabelle in Zukunft auch dynamisch erweitern.

Rolle	**Claim-Wert**
Admin	`photoprism_admin`

Tabelle 2: Akzeptierte Claim-Werte für Rollen.

[45] PhotoPrism Quellcode: `internal/oidc/client.go L90-L92 [feature/oidc-v2]`

Für die Extraktion der Rolleninformationen aus den Claims, wird eine neue Funktion im oidc-Package zuständig sein.[46] Diese wird dann im Handler des Callback Endpunktes eingebunden und setzt die Rolleninformation bevor die interne Repräsentation des Benutzers erstellt bzw. aktualisiert wird.[47]

Probleme mit dem `github.com/caos/oidc`-Package

Beim Einbinden der noch jungen Bibliothek wurden im Rahmen der Arbeit zwei Probleme aufgedeckt, die in Zusammenarbeit mit deren Team zügig gelöst werden konnten:

1. **Der PKCE *Code Verifier* entspricht nicht den Spezifikationen**[48]
   Der *Code Verifier*, beschrieben im PKCE Protokoll, hatte nicht die erforderliche Länge und wurde von Keycloak daher abgelehnt. Somit konnte bei aktiviertem PKCE der *Authorization Code Grant* nicht vollendet werden und die Authentifizierung ist immer fehlgeschlagen. Um diesen Fehler zu beheben, wurde ein Pull Request[49] eingereicht, der die Generierung des *Code Verifier* den Empfehlungen der Spezifikation ensprechend durchführt. Der Fix wurde in `v0.15.9` aufgenommen und das Problem damit gelöst.

2. **Validierung der JWT Signatur**[50]
   Der zur Validierung notwendige kryptografische Schlüssel (JWK, siehe 2.4.3) wird mittels des *OpenID Provider Configuration* Endpunktes vom OP bereitgestellt und kann bei der Validierung einer Token-Signatur dort abgerufen werden. Für den Fall, dass mehrere Schlüssel vom System verwaltet werden, sieht die Spezifikation hier vor, die verwendeten Schlüssel im JWT-Header eindeutig durch `kid`-Wert indentifizierbar zu machen. Bei nur einem durch den OP bereitgestellten Schlüssel ist der `kid`-Wert im JWT-Header jedoch nicht obligatorisch. [37, Kapitel 10.1], [18, Kapitel 6]

   Im vorliegenden Fall, war ein `kid`-Wert im einzigen JWK des OP vorhanden, jedoch nicht im Header des JWT. Diese Konstellation führte dazu, dass die Validierung der Signatur mit dem Fehler „invalid key“ quittiert wurde, obwohl der korrekte Schlüssel vorhanden war. Da der `kid`-Wert im JWT-Header für diesen Fall nicht voraussetzt wird, musste hier das Verhalten der Bibliothek angepasst werden, um die Konformität zum Standard zu wahren. Nach Meldung über GitHub wurde dieses Problem von den Autoren in der Version `v0.15.10` behoben.

Testing

Für das Testen dieses Features sind vor allem Integrationstests und Akzeptanztests

[46] PhotoPrism Quellcode: `internal/oidc/client.go L90-L92 [feature/oidc-v2]`
[47] PhotoPrism Quellcode: `internal/api/auth.go L52-L58 [feature/oidc-v2]`
[48] `https://github.com/caos/oidc/issues/125`
[49] `https://github.com/caos/oidc/pull/126`
[50] `https://github.com/caos/oidc/issues/124`

vorgesehen. Komponententests sind nur sporadisch für die Hilfsfunktionen notwendig, da ein Großteil der Funktionalität von externen Bibliotheken bereitgestellt wird.

Um die Funktionalität der OIDC-Integration automatisiert testen zu können, ist ein leichtgewichtiger OP notwendig, mit dem der manuelle Login eines Benutzer oder eine aktive Session beim IdP simuliert werden kann. Da `github.com/caos/oidc` auch in der Lage ist als OP zu agieren, wurde auf der Basis der Bibliothek ein IdP zu diesem Zweck implementiert und in die Docker Entwicklungsumgebung aufgenommen.[51] Mit diesem Dienst kann nun die OIDC-Integration vollautomatisch über die HTTP-API getestet werden.[52]

Dazu kommen Akzeptanztests, welche die Funktionalität nochmal aus Benutzerperspektive sicherstellen und das Frontend mit einschließen. Diese sind nicht Teil der Arbeit.

Manuelle Überprüfung der Funktionalität wurde mit Keycloak durchgeführt. Keycloak wurde ebenfalls zur Entwicklungsumgebung hinzugefügt, um zukünftig auch bei der Entwicklung primär mit dem externen Usermanagement eines IdPs arbeiten zu können.[53]

4.3.3 Benutzer-Dokumentation

PhotoPrism bietet damit nun die Option an mit externem Usermanagement betrieben zu werden. Dazu ist ein Identity Provider (IdP) mit *OpenID Connect 1.0* Unterstützung notwendig. Um das externe Usermanagement zu nutzen und die Schnittstellen zu aktivieren, müssen in der nachfolgenden Tabelle genannten Parameter beim Start von PhotoPrism übergeben werden. Diese Variablen können entweder beim Aufrufen von PhotoPrism via CLI-Option oder mittels *Umgebungsvariablen* vor dem Start übergeben werden. PhotoPrism darf nicht im Public-Mode betrieben werden, wenn OIDC genutzt werden soll.

Mit dem Setzen dieser Parameter gilt das externe Usermanagement als aktiv und interne Benutzer, mit Ausnahme des Admins, dürfen sich nicht mehr anmelden. Wenn das externe Usermanagement eingerichtet ist, gilt der externe IdP als vertrauenswürdig und externe Benutzer werden beim erstmaligen Anmelden automatisch in PhotoPrisms interner Benutzerdatenbank angelegt. Der Kreis der Benutzer, der sich bei PhotoPrism anmelden darf, kann dann nur noch über den IdP eingeschränkt werden.

Unauthentifizierte Besucher werden bei aktivem externen Usermanagement automatisch an die Login-Seite des IdPs weitergeleitet. Um die lokale Login-Seite trotzdem aufzurufen, kann entweder über den Browser zurück navigiert werden oder der Query-Parameter der Login-URL `preventAutoLogin` auf `true` gesetzt werden.
Zum Beispiel: `https://photoprism.localhost/login?preventAutoLogin=true`

[51] PhotoPrism Quellcode: `docker/dummy/oidc/app [feature/oidc-v2]`
[52] PhotoPrism Quellcode: `oidc-v2/internal/api/auth_test.go [feature/oidc-v2]`
[53] PhotoPrism Quellcode: `docker-compose.yml L154-L178 [feature/oidc-v2]`

Option/Umgebungsvariable	**Beschreibung**
`--oidc-client-id <value>` / `PHOTOPRISM_OIDC_CLIENT_ID`	Einmaliger Identifizierer der beim OP registrierten Anwendung.
`--oidc-client-secret <value>` / `PHOTOPRISM_OIDC_CLIENT_SECRET`	Client Secret der beim OP registrierten Anwendung.
`--oidc-issuer-url <value>` / `PHOTOPRISM_OIDC_ISSUER_URL`	Über die *Issuer*-URL kann die Anwendung den OP erreichen. Token werden nur von diesem *Issuer* akzeptiert.
`--oidc-scopes <value>` / `PHOTOPRISM_OIDC_CUSTOM_SCOPES`	Optional. Individueller Scope kann dazu verwendet werden, die Rolleninformationen beim OP anzufragen.
`--site-url <value>` / `PHOTOPRISM_SITE_URL`	Muss PhotoPrisms Root-URL enthalten. Wird benötigt, um den Callback-URL zu generieren.

Tabelle 3: PhotoPrisms Konfigurationsparameter für externes Usermanagement via OpenID Connect.

Wenn das externe Usermanagement aktiv ist, sollte auch das Mapping der Rollen beim IdP eingerichtet werden, da externe Benutzer sonst nicht die *Admin*-Rolle einnehmen können.

Beim IdP muss sichergestellt sein, dass:

1. Ein OIDC *Confidential Client* mit *Client ID* und *Client Secret* eingerichtet ist.
2. Der eingerichtete Client eine *gültige Redirect URL* nach folgendem Schema hinterlegt hat: `{http|https}://<photoprism_site_url>/api/v1/auth/external`
   Zum Beispiel: `https://photoprism.dev/api/v1/auth/external`
3. Der *Authorization Code Grant* unterstützt wird. Optional mit *PKCE*.

5 Ergebnisse und Ausblick

Entsprechend der Zielsetzung wurden im Rahmen der Arbeit eine erste Ausbaustufe hin zu einer Mehrbenutzer-Applikation konzipiert und implementiert. Während der Arbeit am Code sind einige Punkte aufgefallen, die insbesondere aus sicherheitstechnischer Sicht noch Verbesserungspotential haben. Diese Punkte werden im ersten Abschnitt dieses Kapitels aufgegriffen und Empfehlungen dazu ausgesprochen. Im zweiten Abschnitt wird eine kurze Übersicht der Ergebnisse bezogen auf die Anforderungen dargestellt und ein Fazit aus der Arbeit an dem Projekt gezogen. Der dritte Abschnitt wird dann einen kurzen Ausblick auf den weiteren möglichen Verlauf des Mehrbenutzer-Ausbaus von PhotoPrism gegeben.

5.1 Empfehlungen

5.1.1 API: HTTP Status-Codes 401 und 403

Problem: Die HTTP-API von PhotoPrism kann zum jetzigen Zeitpunkt unauthentifizierte Zugriffe und authentifizierte, aber unautorisierte Zugriffe nicht unterscheiden. Beide Fälle werden mit dem Status-Code 401 beantwortet, woraufhin das Frontend von fehlender bzw. ungültiger Authentifizierung ausgeht und die aktuelle Session verwirft. Dies hat zur Folge, dass der betroffene Benutzer ausgeloggt wird.

Lösung: Die HTTP-Spezifikationen der IETF sehen für die beiden genannten Fälle unterschiedliche Status-Codes vor (401 und 403). [9, Kapitel 3.1] [10, Kapitel 6.5.3] Wenn die HTTP-API diese Unterscheidung implementiert, kann im Frontend ein entsprechend anderes Verhalten, wie beispielsweise eine Fehlermeldung, als Reaktion auf eine 403 Antwort implementiert werden.

5.1.2 Data Authorization and Filtering

Es wird empfohlen für die Zugriffskontrolle auf Inhaltsebene zu evaluieren, ob die Einführung eines AC-Modells auch hier Mehrwert bieten kann. Vorteile könnten sich in der Flexibilität der Architektur bemerkbar machen, da sich mithilfe eines solchen Modells das Berechtigungssystem von den Daten entkoppeln ließe, womit es nachträglich leichter an neue Anforderungen anpassbar wäre.

Weiter sollte geprüft werden, ob eine effiziente Zugriffskontrolle an jedem Endpunkt der API umsetzbar ist, damit wirklich keine Daten an unautorisierte Benutzer ausgespielt werden.

Die Meinung der Maintainer, dass die Unkenntnis einer URL Sicherheit genug bietet, wird hier nicht geteilt. Aufgrund der begrenzten Entwickler-Ressourcen und der bisherigen Ausrichtung auf Heimnetzwerke ohne öffentliche Erreichbarkeit vom Internet, ist dieser Trade-Off zum aktuellen Zeitpunkt jedoch vertretbar.

Für die Einführung von explizitem *Data Authorization and Filtering*, könnten zum Beispiel auch Bibliotheken wie *Casbin*[54] oder *Oso*[55] eingesetzt werden. Damit könnte auch das *Data Authorization and Filtering* mit *Domain Authorization* konsolidiert werden und Zugriffskontrolle mittels einem einzelnen AC-Modul umgesetzt werden.

Für noch mehr Flexibilität und Möglichkeit des Einsatzes als separaten Dienstes, stehen darüber hinaus noch *OPA*[56] und *Keto*[57] zur Evaluation zu Verfügung. Wobei diese beiden vermutlich eher für verteilte Systeme, Microservice-Architekturen und Anwendungslandschaften in Unternehmen ausgelegt sind. Sehr interessant sind jedoch die Ambitionen des noch jungen *Keto*, welches sich zum Ziel gesetzt hat, eine Open-Source Implementierung für Google's *Zanzibar* [28] umzusetzen.

5.1.3 Passwortsicherheit

Um Daten von Benutzern bei Verwendung des internen Usermanagements besser zu schützen, wird empfohlen alle Level-1 Maßnahmen des [27, Kapitel 2.1 und 2.2] zu implementieren. Insbesondere, wenn PhotoPrisms zukünftiger Einsatzbereich auch offiziell auf öffentliche Netze ausgeweitet werden soll.

5.1.4 Session Management

In der aktuellen Implementierung des Session Managements konnten einige Schwachstellen identifiziert werden (siehe 4.1.1). Diese umfassen:

1. Rate-Limiting des Session Endpunktes
2. Änderungen von Profilinformationen eines Nutzers während einer aktiven Session wirken sich nicht auf den eingeloggten Benutzer aus.
3. Keine Zugriffskontrolle bei der API für den Abruf von Bildern und Videos

Das Rate-Limiting (Punkt Eins) kann durch Verwendung einer Middleware für Go's HTTP-Framework relativ leicht verbessert werden. `github.com/ulule/limiter` stellt beispielsweise ein fertiges Go-Package für genau diesen Zweck bereit. Eine solche Middleware kann gezielt und mit wenig Aufwand in den Session-Endpunkt eingehängt werden,

[54] `https://casbin.org/en/`
[55] `https://docs.osohq.com/index.html`
[56] `https://www.openpolicyagent.org/docs/latest/`
[57] `https://www.ory.sh/keto/`

bietet besseren Schutz und beeinträchtigt bei richtiger Konfiguration die Nutzererfahrung weniger als die aktuelle Implementierung.

Der zweite Punkt wird dadurch verursacht, dass die Benutzerinformationen bei einer aktiven Session im Speicher der Anwendung gehalten werden. Bei Bedarf werden diese aus der Session extrahiert und nicht aus der Datenbank.[58] Wird eine Änderung der Benutzerinformationen über die Datenbank ausgeführt, hat dies daher keine Auswirkungen auf den eingeloggten Benutzer, da die Benutzerdaten in der Session dadurch nicht verändert werden. Selbst das Löschen oder ein Ändern der Rolle des Benutzers wirkt sich nicht auf eine aktive Session aus. Dies kann durch regelmäßige Invalidierung des Benutzer-Caches in kurzen Zeitintervallen vermieden werden. Dies sollte auch bei Austausch des ganzen Session Mechanismus berücksichtigt werden.

Für den dritten Punkt wird trotz bewusster Entscheidung zum Verzicht der Überprüfung der Berechtigungen empfohlen, diese Endpunkte out-of-the-box durch Zugriffskontrollen zu schützen und nicht einzig darauf zu setzen, dass die Unkenntnis der URLs Schutz genug ist („Security through obscurity“).

Für Dritt-Clients, die nicht wie Browser über ein integriertes Cookie-Management verfügen, ist zu beachten, dass ausschließlich Cookie-basierten Lösungen eher unkomfortabel sind und vermieden werden sollten.

Unter Berücksichtigung all dieser Aspekte kommen damit zwei potentielle Lösungen für Punkt Drei in Betracht:

- PhotoPrisms API akzeptiert zwei Methoden zur Autorisierung der Anfragen: Cookies und `Authorization`-Header.
- Die Nutzung eines *Web Worker* Proxies, der die Anfragen mit den erforderlichen Headern ausstattet.

Die Verwendung eines CDNs ist eher im professionelleren Umfeld zu finden und wird selten von Benutzern im Heimumfeld benötigt. Sie ist damit bezogen auf die erklärten Ziele von PhotoPrism (insbesondere Privatsphäre) als Grund für den Verzicht auf explizite Prüfung der Berechtigung, ein eher schlechter Trade-Off.

Um dies jedoch nicht auszuschließen, könnte eine durch den Administrator deaktivierbare Zugriffskontrolle der relevanten Endpunkte die Nutzung von CDNs weiter ermöglichen. Idealerweise kann mit sogenannten *signierten URLs* (z.B. durch [41] bei Amazon's *CloudFront*-CDN) sogar Zugriffskontrolle und CDN-Nutzung zu gleichen Zeit ermöglichen werden.

[58] PhotoPrism Quellcode: `internal/session/data.go L29 [feature/oidc-v2]`

5.2 Erfüllte Anforderungen

In Tabelle 4 sind alle funktionalen Anforderungen aus Kapitel 3 mit deren Status zum Ende der Arbeit aufgelistet. Einige Anforderungen können mittels IdP zu PhotoPrism hinzugefügt werden und können damit als erfüllt betrachtet werden (blau hinterlegt). Hier ist auch auf das Feedback der Nutzerschaft zu hören. Sollte sich das Hinzufügen der Funktion zu PhotoPrism selbst noch als notwendig erweisen, kann dies zu jedem Zeitpunkt noch nachgeholt werden. Die einzige teilweise erfüllte Anforderung (gelb), sollte dennoch schnellstmöglich vervollständigt werden, damit die Mehrbenutzer-Funktionen auch in einem sinnvollen Paket ins Release überführt werden können. Erfüllte Anforderungen (grün) können natürlich auch jederzeit nochmal überarbeitet bzw. erweitert werden, sollte sich im weiteren Verlauf zeigen, dass dies sinnvoll ist.

5.3 Fazit und Ausblick

Die Arbeit an PhotoPrism hat einen tiefen Einblick in die Open-Source-Welt und in komplexe Protokolle wie OpenID Connect ermöglicht. Insbesondere die Organisation und die Projektplanung werden, zumindest bei PhotoPrism, völlig anders gehandhabt als man es von Projekten dieser Größe in kommerziellem Umfeld erwarten würde. Bemerkenswert ist, was ein kleines Team hier innerhalb von drei Jahren alles geleistet hat. Hier kann nur nochmal auf die Erkenntnisse von Raymond verwiesen werden. [33]

Wichtig ist an dieser Stelle auch zu erwähnen, dass es gerade bei solch großen Dysbalancen zwischen Entwicklerstärke und Projektgröße, sehr wichtig ist, gute Kompromisse finden zu können, in kleinen Schritten zu entwickeln und Probleme und Feedback immer wieder zu reflektieren.

Mit dem Ende der Arbeit konnte eine erste nutzbare Iteration entwickelt werden, die es ermöglicht eine Mediensammlung in PhotoPrism mit mehreren Benutzern zu verwenden ohne, dass man die Instanz im *Public*-Mode betreiben muss oder jemandem das Admin-Passwort geben muss. Benutzer können nun mit eingeschränkten Rechten ausgestattet werden, damit nicht jeder Daten hinzufügen, löschen oder ändern kann. Durch OpenID Connect ist es nun möglich das Usermanagement komplett auszulagern und weitere (Sicherheits-)Funktionen hochspezialisierter Drittanbieter zu nutzen, ohne dass PhotoPrism diese selbst unterstützen muss.

Trotz dieser Fortschritte steht jedoch noch eine Menge Arbeit an, bevor die aktuelle Vision einer Mehrbenutzer-Anwendung erreicht wird, wie an der Tabelle der Anforderungen (Tabelle 4) zu sehen ist. Abbildung 11 zeigt wie der Ausbau weiter verlaufen kann. Von der erste Stufe, in der PhotoPrism im Grunde einen Benutzer eine Mediensammlung verwalten lässt, wurde die Anwendung für mehrere Benutzer und eine Mediensammlung ausgebaut. Nach Abschluss dieses Meilensteins ist die logische nächste Stufe, jeden Benutzer eine eigene Mediensammlung zu erlauben. In der vierten Stufe können dann die

ID	Bezeichnung	Status
R-U01	Neue Benutzer	erfüllt
R-U02	Eigenständige Registrierung	offen
R-U03	E-Mail Whitelisting	via IdP
R-U04	Benutzer entfernen/deaktivieren	erfüllt
R-U05	Änderungen von Benutzerdaten	erfüllt
R-U06	Benutzer auflisten	erfüllt
R-U07	Authentifizierung: An- und Abmelden	erfüllt
R-U08	2-Faktor Authentifizierung	via IdP
R-U09	Profil-Ansicht in der Nutzeroberfläche	erfüllt
R-U10	E-Mail Bestätigung	offen
R-U11	Passwortsicherheit	offen
R-P01	Ressourcen und Aktionen	erfüllt
R-P02	Rollenkonzept	erfüllt
R-P03	Statische Rollen	erfüllt
R-P04	Rolle: Member	teilweise
R-P05	Dynamische Rollen	offen
R-P06	Eigene Mediensammlungen	offen
R-P07	Teilen von Alben mit Personen ohne Benutzerkonto	erfüllt
R-P08	Gemeinsame Alben mit anderen Benutzern einer Instanz	offen
R-P09	Gemeinsame Alben mit Benutzern anderer Instanzen	offen
R-P10	Öffentlicher Bereich	offen
R-S01	Delegierte Authentifizierung und externes Usermanagement	erfüllt
R-S02	Automatische Weiterleitung	erfüllt
R-S03	Rechteverwaltung via Identity Provider	erfüllt
R-S04	Verknüpfen von Benutzern	offen

Tabelle 4: Übersicht der funktionalen Anforderungen zum Ende der Arbeit.

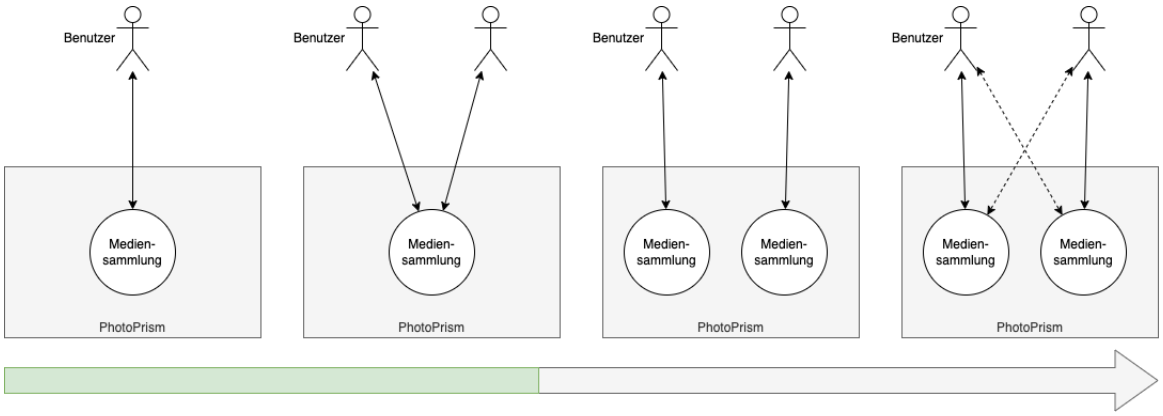

Abbildung 11: Zukünftige Meilensteine.

Berechtigungen zum gegenseitigen Freigeben von Teilen eigener Mediensammlungen und zusätzlichem öffentlichen Bereich erarbeitet werden.

Die Ergebnisse der Umsetzung in PhotoPrism können durch Aufsetzen einer Entwicklungsumgebung des `feature/oidc-v2` Branches[59] begutachtet werden. Das eigenständige Aufsetzen der Entwicklungsumgebung wird in der Dokumentation beschrieben: `https://docs.photoprism.app/developer-guide/setup/`.

[59] `https://github.com/photoprism/photoprism/tree/feature/oidc-v2`

Glossar

CDN Ein CDN, kurz für Content Delivery Network, ist ein Netzwerk aus regional verteilten Servern, die Inhalte möglichst nah bei den Konsumenten platzieren bzw. cachen, um Latenzen zu minimieren, den Datendurchsatz zu maximieren und die Last durch viele Anfragen auf mehrere Standorte zu verteilen. 35

CLI-Option Command Line Options, auch Flags oder Switches, sind Parameter, die das Verhalten eines Programms beeinflussen. Im Gegensatz zu Arguments sind diese oft optional oder werden dazu verwendet zusätzliche Parameter an das Programm via Kommandozeile (CLI) zu übergeben. 32

CSRF Bei einem Cross-site Request Forgery Angriff kann der Angreifer in einer Webanwendung Aktionen im Namen eines Benutzers ausführen, der gerade angemeldet ist. 47

IdP IdP ist die Abkürzung für Identity Provider und bezeichnet einen zentralen Dienst, der für die Authentifizierung von Benutzern verantwortlich ist und sich um die Verwaltung von Profilinformationen kümmert und diese berechtigten dritten Diensten bereitstellt. 19

P2P Peer-to-Peer. Kommunikationsnetz bestehend aus gleichberechtigten Teilnehmern. 43

RESTful-API HTTP-basierte Programmierschnittstelle, die zur Kommunikation von Client und Server verwendet wird und lose an die Konzepte von Fielding angelehnt ist [11, S. 76–106]. 13

Struct Ein Struct repräsentiert eine Sammlung von Feldern in der Programmiersprache Go. 36

Umgebungsvariable Umgebungsvariablen sind eine von Betriebssytemen bereitgestellte Möglichkeit, Konfigurationen inf Form von Zeichenketten zu definieren, die von mehreren Programmen verwendet werden können. 32

WebDAV Web-based Distributed Authoring and Versioning erweitert das HTTP-Protokoll um Funktionen zur Verwaltung von Ordnern und Dateien. In gängigen Betriebssystemen können damit relativ einfach und ohne Drittsoftware Dateisysteme über das Netzwerk lokal eingebunden werden. 7

XSS Cross-site Scripting ist ein Angriff auf Schwachstellen in Webanwendungen. Charakteristisch hierfür ist die Injektion von böswilligen Scripten in Seiten, welche auch von anderen Benutzern besucht werden. 34

Akronyme

ABAC Attribute-based Access Control. 15

AC Access Control. 15

ACL Access Control List. 15

API Application Programming Interface. 21

CLI Command Line Interface. 35

CRUD Create, Read, Update, Delete. 33

DAC Discretionary Access Control. 15

GUI Graphical User Interface. 35

HTTP Hypertext Transfer Protocol. 14

IAM Identity and Access Management. 17

IETF Internet Engineering Taskforce. 59

MAC Mandatory Access Control. 15

ORM Object Relation Mapping. 32

PWA Progressive Web Application. 7

RBAC Role-based Access Control. 15

SPA Single Page Application. 7

TLS Transport Layer Security. 23

UI User Interface. 29

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[1] *An open network for secure, decentralized communication*. The Matrix.org Foundation C.I.C. URL: https://matrix.org/ (besucht am 10. 12. 2021).

[2] *Autorenrichtlinie zur Erstellung eines benutzerdefinierten Bausteins*. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/IT-Grundschutz-Modernisierung/Benutzerdefinierte_BS/Autorenrichtlinie.pdf?__blob=publicationFile&v=3.

[3] Scott O. Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. RFC 2119. März 1997. DOI: 10.17487/RFC2119. URL: https://rfc-editor.org/rfc/rfc2119.txt.

[4] Philippe De Ryck. *From the Implicit flow to PKCE: A look at OAuth 2.0 in SPAs*. Mai 2020. URL: https://pragmaticwebsecurity.com/articles/oauthoidc/from-implicit-to-pkce.html (besucht am 10. 12. 2021).

[5] Philippe De Ryck. *Why avoiding LocalStorage for tokens is the wrong solution*. Apr. 2020. URL: https://pragmaticwebsecurity.com/articles/oauthoidc/localstorage-xss.html (besucht am 10. 12. 2021).

[6] Kelley Dempsey, Gregory Witte und Doug Rike. *Summary of NIST SP 800-53, Revision 4: Security and Privacy Controls for Federal Information Systems and Organizations*. en. 2014. DOI: https://doi.org/10.6028/NIST.CSWP.02192014.

[7] *Docker Documentation: Overview*. Docker, Inc. URL: https://docs.docker.com/get-started/overview/ (besucht am 10. 12. 2021).

[8] David F. Ferraiolo und D. Richard Kuhn. *Role-Based Access Controls*. 1992. arXiv: 0903.2171 [cs.CR]. URL: https://csrc.nist.gov/CSRC/media/Publications/conference-paper/1992/10/13/role-based-access-controls/documents/ferraiolo-kuhn-92.pdf.

[9] Roy T. Fielding und Julian Reschke. *Hypertext Transfer Protocol (HTTP/1.1): Authentication*. RFC 7235. Juni 2014. DOI: 10.17487/RFC7235. URL: https://rfc-editor.org/rfc/rfc7235.txt.

[10] Roy T. Fielding und Julian Reschke. *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*. RFC 7231. Juni 2014. DOI: 10.17487/RFC7231. URL: https://rfc-editor.org/rfc/rfc7231.txt.

[11] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. University of California, Irvine, 2000.

[12] *GitHub Issue: Multi-User Photo Gallery with private and shared photos/albums*. URL: https://github.com/photoprism/photoprism/issues/98 (besucht am 10. 12. 2021).

[13] *GitHub: Photos App powered by Go and Google TensorFlow*. URL: https://github.com/photoprism/photoprism (besucht am 10. 12. 2021).

[14] *GNU Make*. Free Software Foundation, Inc. URL: https://www.gnu.org/software/make/ (besucht am 10. 12. 2021).

[15] Dick Hardt. *The OAuth 2.0 Authorization Framework*. RFC 6749. Okt. 2012. DOI: 10.17487/RFC6749. URL: https://rfc-editor.org/rfc/rfc6749.txt.

[16] Dick Hardt, Aaron Parecki und Torsten Lodderstedt. *The OAuth 2.1 Authorization Framework*. Internet-Draft draft-ietf-oauth-v2-1-04. Work in Progress. Internet Engineering Task Force, Okt. 2021. 85 S. URL: https://datatracker.ietf.org/doc/html/draft-ietf-oauth-v2-1-04.

[17] Michael Jones. *JSON Web Key (JWK)*. RFC 7517. Mai 2015. DOI: 10.17487/RFC7517. URL: https://rfc-editor.org/rfc/rfc7517.txt.

[18] Michael Jones, John Bradley und Nat Sakimura. *JSON Web Signature (JWS)*. RFC 7515. Mai 2015. DOI: 10.17487/RFC7515. URL: https://rfc-editor.org/rfc/rfc7515.txt.

[19] Michael Jones, John Bradley und Nat Sakimura. *JSON Web Token (JWT)*. RFC 7519. Mai 2015. DOI: 10.17487/RFC7519. URL: https://rfc-editor.org/rfc/rfc7519.txt.

[20] Michael Jones und Joe Hildebrand. *JSON Web Encryption (JWE)*. RFC 7516. Mai 2015. DOI: 10.17487/RFC7516. URL: https://rfc-editor.org/rfc/rfc7516.txt.

[21] N. Klingenstein u. a. *OASIS Security Services (SAML) TC*. 2012. URL: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security (besucht am 10. 12. 2021).

[22] Peter Liggesmeyer, Hrsg. *Software-Qualität: Testen, Analysieren und Verifizieren von Software*. 2. Aufl. SpringerLink Bücher. Heidelberg: Spektrum Akademischer Verlag, 2009. URL: https://doi.org/10.1007/978-3-8274-2203-3.

[23] Torsten Lodderstedt u. a. *OAuth 2.0 Security Best Current Practice*. Internet-Draft draft-ietf-oauth-security-topics-19. Work in Progress. Internet Engineering Task Force, Dez. 2021. 52 S. URL: https://datatracker.ietf.org/doc/html/draft-ietf-oauth-security-topics-19.

[24] Michael Mayer und Theresa Gresch. *PhotoPrism Webseite*. URL: https://photoprism.app (besucht am 10. 12. 2021).

[25] Michael Mayer und Theresa Gresch. *PhotoPrism Webseite – Features*. URL: https://photoprism.app/features (besucht am 10. 12. 2021).

[26] Breno de Medeiros u. a. *OpenID Connect Session Management 1.0 - draft 30*. Aug. 2020. URL: https://openid.net/specs/openid-connect-session-1_0.html (besucht am 10. 12. 2021).

[27] *OWASP Application Security Verification Standard 4.0.2*. Okt. 2020. URL: https://github.com/OWASP/ASVS/raw/v4.0.2/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0.2-de.pdf (besucht am 10. 12. 2021).

[28] Ruoming Pang u. a. „Zanzibar: Google's Consistent, Global Authorization System". In: *2019 USENIX Annual Technical Conference (USENIX ATC '19)*. Renton, WA, 2019.

[29] Aaron Parecki und David Waite. *OAuth 2.0 for Browser-Based Apps*. Internet-Draft draft-ietf-oauth-browser-based-apps-08. Work in Progress. Internet Engineering Task Force, Mai 2021. 21 S. URL: https://datatracker.ietf.org/doc/html/draft-ietf-oauth-browser-based-apps-08.

[30] *PhotoPrism Community Chat*. Michael Mayer & Theresa Gresch GbR. URL: https://gitter.im/browseyourlife/community (besucht am 10. 12. 2021).

[31] *PhotoPrism Docs: Bottom-Up Development*. URL: https://docs.photoprism.app/developer-guide/code-quality/#bottom-up-development (besucht am 08. 01. 2022).

[32] *PhotoPrism Flutter App for iOS and Android*. URL: https://github.com/photoprism/photoprism-mobile (besucht am 10. 12. 2021).

[33] Eric S. Raymond. *Die Kathedrale und der Basar*. Aug. 1999. URL: https://www.selflinux.org/selflinux/pdf/die_kathedrale_und_der_basar.pdf (besucht am 10. 12. 2021).

[34] Pete Resnick. *Internet Message Format*. RFC 5322. Okt. 2008. DOI: 10.17487/RFC5322. URL: https://rfc-editor.org/rfc/rfc5322.txt.

[35] Nat Sakimura, John Bradley und Naveen Agarwal. *Proof Key for Code Exchange by OAuth Public Clients*. RFC 7636. Sep. 2015. DOI: 10.17487/RFC7636. URL: https://rfc-editor.org/rfc/rfc7636.txt.

[36] Nat Sakimura, John Bradley und Michael B. Jones. *OpenID Connect Dynamic Client Registration 1.0 incorporating errata set 1*. Nov. 2014. URL: https://openid.net/specs/openid-connect-registration-1_0.html (besucht am 10. 12. 2021).

[37] Nat Sakimura u. a. *OpenID Connect Core 1.0 incorporating errata set 1*. Nov. 2014. URL: https://openid.net/specs/openid-connect-core-1_0.html (besucht am 10. 12. 2021).

[38] Nat Sakimura u. a. *OpenID Connect Discovery 1.0 incorporating errata set 1*. Nov. 2014. URL: https://openid.net/specs/openid-connect-discovery-1_0.html (besucht am 10. 12. 2021).

[39] Alexander Schatten u. a. *Best Practice Software-Engineering: Eine praxiserprobte Zusammenstellung von komponentenorientierten Konzepten, Methoden und Werkzeugen*. Springer-Verlag, 2010.

[40] *Securing Applications and Services Guide*. Free Software Foundation, Inc. URL: https://www.keycloak.org/docs/latest/securing_apps/\#openid-connect-vs-saml (besucht am 10. 12. 2021).

[41] *Verwenden signierter URLs*. Google. URL: https://docs.aws.amazon.com/de_de/AmazonCloudFront/latest/DeveloperGuide/private-content-signed-urls.html (besucht am 10. 12. 2021).

[42] *Vue Mixins*. URL: https://vuejs.org/v2/guide/security.html (besucht am 10. 12. 2021).

[43] *Vue Mixins*. URL: https://vuejs.org/v2/guide/mixins.html (besucht am 10. 12. 2021).

[44] *Welcome to OpenID Connect*. URL: https://openid.net/connect/ (besucht am 10. 12. 2021).

[45] Wikipedia contributors. *SAML-based products and services — Wikipedia, The Free Encyclopedia*. [Online; accessed 7-January-2022]. 2021. URL: https://en.wikipedia.org/w/index.php?title=SAML-based_products_and_services&oldid=1061752006.

[46] *Window: storage event*. Mozilla. Sep. 2021. URL: https://developer.mozilla.org/en-US/docs/Web/API/Window/storage_event (besucht am 10. 12. 2021).

[47] Abel Yang, Shawn Steele und Ned Freed. *Internationalized Email Headers*. RFC 6532. Feb. 2012. DOI: 10.17487/RFC6532. URL: https://rfc-editor.org/rfc/rfc6532.txt.