Token-based Authentication and Authorization in Microservices

MASTER THESIS

Christoff Schaub

Eingereicht am 18. August 2023

Friedrich-Alexander-Universität Erlangen-Nürnberg
Technische Fakultät, Department Informatik
Professur für Open Source Software

Betreuer:
Georg Schwarz, M. Sc.
Prof. Dr. Dirk Riehle, MBA

Friedrich-Alexander-Universität
Technische Fakultät

Eidesstattliche Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, 18. August 2023

Lizenz

Erlangen, 18. August 2023

Abstract

Microservices represent an architectural approach that is gaining more and more importance to realize highly scalable, reliable and maintainable systems within an agile development process. The advantages of microservice architecture are accompanied by the challenge of designing a distributed system. In such a distributed system, authorization and authentication pose a particular problem. This is because not only the calls of the users to a single service have to be secured, as is the case with a classic monolithic architecture. Instead, end-to-end authorization and authentication must be ensured for every API that can be accessed by a user, as well as for communication between microservices.

One approach to solving this problem is token-based authentication and authorization. This provides an efficient and scalable mechanism to ensure security and access control in microservice systems. In this work, we investigate how to implement a token-based authentication and authorization in a microservice architecture.

To this end, a structured literature review is first conducted to build theory. The resulting findings serve as the basis for the subsequent demonstration of token-based authentication and authorization in practice. In the demonstration, an authentication and authorization concept for an Open Data ETL platform based on an Open Source IAM solution is designed and subsequently implemented in a prototype. Finally, the lessons learned from the design and implementation are compared to the results of the literature review.

Zusammenfassung

Microservices stellen einen Architekturansatz dar, der mehr und mehr an Bedeutung gewinnt, um hoch skalierbare, verlässliche und wartbare Systeme innerhalb eines agilen Entwicklungsprozesses zu realisieren. Die Vorteile der Microservice-Architektur gehen mit der Herausforderung einher, ein verteiltes System zu entwerfen. Bei einem solchen verteilten System stellt die Autorisierung und Authentifizierung eine besondere Problematik dar. Dies liegt darin begründet, dass nicht nur die Aufrufe der User zu einem einzigen Service abgesichert werden müssen, wie dies bei einer klassischen monolithischen Architektur der Fall ist. Stattdessen muss eine durchgängige Autorisierung und Authentifizierung bei jeder durch einen User erreichbaren API sowie bei der Kommunikation der Microservices untereinander gewährleistet sein.

Ein Ansatz zur Lösung dieses Problems ist die tokenbasierte Authentifizierung und Autorisierung. Diese bietet einen effizienten und skalierbaren Mechanismus, um die Sicherheit und Zugriffskontrolle in Microservicesystemen zu gewährleisten. In dieser Arbeit wird untersucht, wie eine tokenbasierte Authentifizierung und Autorisierung in einer Microservice-Architektur implementiert werden kann.

Dafür wird zunächst eine strukturierte Literaturrecherche zur Theoriebildung durchgeführt. Die daraus resultierenden Ergebnisse dienen als Grundlage für die anschließende Demonstration der tokenbasierten Authentifizierung und Autorisierung in der Praxis. In der Demonstration wird ein Authentifizierungs- und Autorisierungskonzept für eine Open Data ETL Plattform auf Basis einer Open Source IAM-Lösung entworfen und anschließend prototypisch implementiert. Abschließend werden die aus der Konzeption und Implementierung gewonnenen Erfahrungen den Ergebnissen der Literaturrecherche gegenübergestellt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Acronyms

ABAC Attribute Based Access Control

API Application Programming Interface

DAC Discretionary Access Control

ETL Extract Transform Load

HTTPS Hypertext Transfer Protocol Secure

IAM Identity Access Management

JSON JavaScript Object Notation

JTI JWT ID

JWT Json Web Token

MAC Mandatory Access Control

OIDC OpenID Connect

OPA Open Policy Agent

PKCE Proof Key for Code Exchange

RBAC Role Based Access Control

REST Representational State Transfer

SAML Security Assertion Markup Language

SPA Single-Page-Application

SSO Single Sign-On

XACML eXtensible Access Control Markup Language

XML Extensible Markup Language

1 Einführung

Microservices sind ein Architekturansatz, der auf leichtgewichtige, wiederverwendbare, kleinteilige Services setzt, die miteinander interagieren und unabhängig voneinander bereitgestellt werden. Durch ihre lose Kopplung und funktionale Abgeschlossenheit können die Services von verschiedenen Teams getestet und entwickelt werden, was zu einer höheren Entwicklungsperformanz sowie einer leichteren Wartbarkeit führt (Abdelfattah & Cerny, 2022).

Des Weiteren bietet dieser Architekturstil den Vorteil, dass für einen Service die jeweils am besten geeignete Programmiersprache und Technologie für die zu lösende Aufgabe verwendet werden kann. Aufgrund von diesen und vielen weiteren Vorteilen, wie Skalierbarkeit und Fehlertoleranz, hat sich die Microservice-Architektur in den vergangenen Jahren als ein Standard in der Softwarebranche etabliert (Nasab et al., 2022).

Viele der eben aufgezeigten Vorzüge ergeben sich durch die starke Autonomie der Microservice-Architektur, die mit einem hohen Kommunikationsaufwand einhergeht. Dadurch entstehen aber auch Gefahren, mit welchen wiederum umgegangen werden muss. Eine dieser Herausforderungen sind Cross-Cutting Concerns. Junker (2020) beschreibt Cross-Cutting Concerns als Probleme, die mehrere oder alle Bestandteile einer Architektur betreffen. Zu diesen gehört unter anderem die Autorisierung und Authentifizierung, welche in Microservices eine besondere Problematik darstellen.

Nach Banati et al. (2018) ist dies darauf zurückzuführen, dass eine Microservice-Architektur im Vergleich zur traditionellen monolithischen Architektur zahlreiche Zugänge für Anwender und die Interaktion mit anderen Microservices bietet, was wiederum umfangreiche Authentifizierungs- und Autorisierungsprozesse erfordert. Durch diese sich ständig verändernde Anzahl von Zugangspunkten, auch zur Laufzeit, entstehen neue Herausforderungen im Bereich der Sicherheit. Um die neu entstehenden Herausforderungen der Authentifizierung und Autorisierung in Softwaresystemen zu bewältigen, gibt es verschiedene Ansätze. Einer dieser Ansätze ist die tokenbasierte Authentifizierung und Autorisierung (Banati et al., 2018). Dieser Ansatz eignet sich besonders gut für eine verteilte Architektur und

ist damit ideal für den Microservice-Architekturansatz (ShuLin & JiePing, 2020).

Das Ziel dieser Arbeit ist es, die Theorie für die Anwendung der tokenbasierten Authentifizierung und Autorisierung im Microservice Umfeld aufzubereiten und durch eine Anwendung in der Praxis zu demonstrieren. Die Demonstration findet im Umfeld des JValue-Hubs statt, eines Dienstes, der die Ausführung von Extract Transform Load (ETL) Prozessen auf Open Data Datenquellen ermöglicht. Da sich die Anwendung noch im Aufbau befindet, soll die Authentifizierungs- und Autorisierungslösung möglichst einfach erweiterbar sein.

Um dieses Ziel zu erreichen, steht folgende Forschungsfrage im Fokus der Arbeit: Wie kann man eine tokenbasierte Authentifizierung und Autorisierung in einer Microservice-Architektur implementieren?

Daraus leiten sich folgende feingranulare Forschungsfragen für die strukturierte Literaturanalyse ab, die in Kapitel 3 beantwortet werden:

- RQ1: Wie funktioniert die Token Erstellung?
- RQ2: Welche Informationen können in einem Token enthalten sein?
- RQ3: Wie wird die Authentifizierung und Autorisierung mithilfe von Token durchgeführt?
- RQ4: Wie können Access Policies mit einem tokenbasierten Ansatz realisiert werden?

2 Forschungsansatz

Um die in der Einleitung aufgeführten Forschungsfragen zu beantworten, wird eine Literaturanalyse in Anlehnung an die systematische Literaturanalyse nach Kitchenham (2004) durchgeführt. Im Anschluss erfolgt die Konzeption und prototypische Implementierung einer Authentifizierungs- und Autorisierungslösung auf Basis einer Open Source Identity Access Management (IAM) Lösung. Die Auswahl dieser IAM-Lösung geschieht anhand einer nach Nickerson et al. (2013) entwickelten Taxonomie. Im Nachfolgenden werden die verwendeten Methodiken zur Erstellung dieser Thesis näher erläutert.

2.1 Strukturierte Literaturanalyse

Die strukturierte Literaturanalyse orientiert sich an den Vorgaben von Kitchenham (2004). Da das Einhalten der gesamten vorgestellten Richtlinien der Autoren für eine Forschergruppe gedacht ist und den Zeitrahmen dieser Arbeit überschreiten würde, wird die durchgeführte Literaturanalyse nur als strukturiert und nicht als systematisch bezeichnet. Es wird an dieser Stelle darauf hingewiesen, dass die Literaturanalyse ausschließlich durch den Autor dieser Arbeit erfolgt ist.

2.1.1 Suchprozess

Zur Identifizierung der relevanten Literatur wurde zunächst ein Suchbegriff entwickelt. Dieser besteht aus der Summe der Kombination jeweils zweier Begriffe:

```
("authentication" OR "authenticate" OR "authorization" OR
"authorize" OR "access" OR "policy" OR "security")
AND
("Microservice" OR "Microservices" OR "Micro service" OR
"Micro services" OR "Micro-service" OR "Micro-services")
```

Der Suchbegriff kam in der Suchmaschine Google Scholar zum Einsatz und wurde nur auf den Titel der zu durchsuchenden Publikationen bezogen. Google Scholar ist eine Suchmaschine für wissenschaftliche Literatur, die Dokumente wie Konferenz-

Artikel, Lehrbücher oder Zeitschriftenartikel durchsuchbar macht (Universität Hamburg, 2016).

Aufgrund der Menge der Publikationen, die durch Google Scholar in Kombination mit dem Einsatz des Suchbegriffes auf Titelebene identifiziert wurden, wird keine weitere Suche in anderen Datenbanken oder mit anderen Suchbegriffen durchgeführt. Die entstehende Datenbasis wird nach eingehender Sichtung in Hinblick auf ihre Größe und Güte als ausreichend für den Umfang dieser Arbeit betrachtet.

2.1.2 Ein- und Ausschlusskriterien

Die durch den Suchbegriff identifizierten Publikationen mussten alle folgenden Einschlusskriterien erfüllen:

IC 1 : englische Sprache

IC 2 : freier Zugang

IC 3 : Publikation, die Authentifizierung und Autorisierung in Microservice-architekturen auf Basis von Token thematisiert

IC 4 : Durchlaufen eines Peer-Reviews

Dabei wurden Artikel, die mindestens einem der folgenden Ausschlusskriterien unterlagen, nicht berücksichtigt:

EC 1 : Keine Präsentation oder Evaluation einer tokenbasierten Authentifizierungs- oder Autorisierungstechnik in einer Microservice basierten Architektur

EC 2 : Literaturrecherche, die keine konkreten Handlungsempfehlungen enthält

Um aus den durch den Suchbegriff identifizierten Papern die für die Forschungsfragen relevanten herauszufiltern, wurde ein aus zwei Phasen bestehender Prozess angewandt:

1. Lesen des Abstracts, ausschließen, sofern die Einschlusskriterien nicht erfüllt werden oder eines der Ausschlusskriterien zutrifft
2. Lesen des Fazit- oder Diskussionsteils, falls nicht eindeutig

2.1.3 Qualitätsbewertung

Durch IC 4 ist sichergestellt, dass die Artikel durch ein Peer-Review gelaufen sind, wodurch weder Whitepaper, Bücher noch Blog-Artikel im Suchprozess berücksichtigt wurden.

2.1.4 Datensammlung und Analyse

Zu allen durch den Suchbegriff gefundenen Publikationen werden Autor, Titel und, sofern vorhanden, Ausschlusskriterium in Anhang A festgehalten. Insgesamt wurden mit den Suchbegriffen 137 Paper identifiziert. Die Abbildung 2.1 zeigt den Ablauf des Auswahlprozesses visuell.

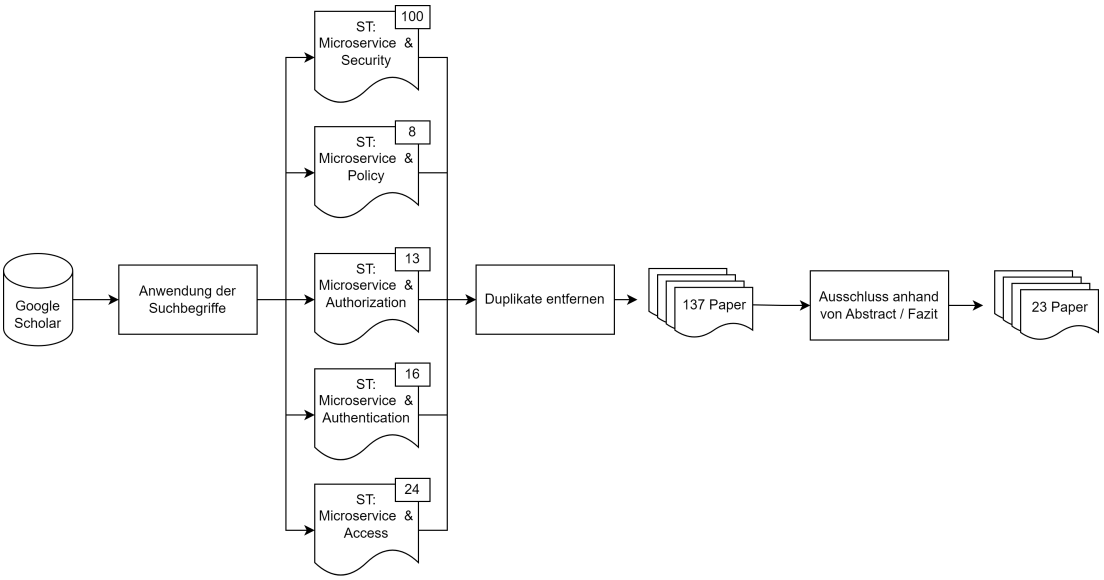

Abbildung 2.1: Literaturauswahlprozess

Nach dem Durchlaufen der ersten beiden Phasen des oben beschriebenen Filterungsprozesses bilden 23 Paper die Literaturbasis für diese Arbeit. Zur genaueren Untersuchung dieser Artikel wurde eine thematische Analyse in Anlehnung an Braun und Clarke (2006) verwendet. Diese besteht aus sechs Phasen:

1. Vertrautmachen mit den Daten
2. Generieren initialer Codes
3. Suchen nach Themen
4. Überprüfung der Themen
5. Definition und Benennung von Themen
6. Erstellung des Berichts

Die von Braun und Clarke (2006) definierten Phasen werden in dieser Arbeit wie folgt berücksichtigt:
Zunächst werden die identifizierten Paper wiederholt gelesen, um einen Überblick über die Daten zu gewinnen. Anschließend wird ein erstes initiales Coding mithilfe

von QDAcity[1] generiert. Dieses zielt darauf ab, die initiale Forschungsfrage, wie man eine tokenbasierte Authentifizierung und Autorisierung in einer Microservice-Architektur implementieren kann, in kleinere Teilbereiche zu strukturieren und durch die identifizierten relevanten Textstellen zu beantworten. Nach dem ersten initialen Coding werden in Phase 3 die Codings gereviewt und zu Themen zusammengefasst. Die Themen repräsentieren in der vorliegenden Arbeit die in Kapitel 1 aufgezeigten Forschungsfragen. In Phase 4 werden die gefundenen Forschungsfragen nochmals anhand der kodierten Textstellen überprüft. Außerdem findet in dieser Phase ebenfalls erneut ein Kodieren der Daten statt, das den Fokus auf das Beantworten der erarbeiteten Forschungsfragen legt. Die in Phase 5 erforderliche Definition der Themen und Codings erfolgt ebenfalls im Tool QDAcity, in dem zu den einzelnen Codierungen im Tool Definitionen hinterlegt werden. Durch die Codierungen ergibt sich ein Gesamtbild, das jeweils zu einer Forschungsfrage alle relevanten Textstellen der Literaturbasis abbildet. In der letzten Phase werden die aus der Datenbasis gewonnenen Erkenntnisse in Kapitel 3 präsentiert.

2.1.5 Ad hoc Literature Research

Aufgrund der Beschaffenheit der gefundenen Literatur, die sich primär mit dem Einsatz von Autorisierung und Authentifizierung im Microserviceumfeld in verschiedenen Anwendungsgebieten beschäftigt, wurden zusätzlich zu der durch die Literaturanalyse identifizierten Werke weitere Artikel durch Ad hoc Literature Research hinzugefügt. Diese Werke und Artikel dienen zur Erklärung von Grundlagen, die häufig in den Anwendungsszenarien nicht mehr explizit genannt werden, jedoch wichtig für ein Verständnis des Gesamtzusammenhangs sind. Die durch den Ad hoc Suchprozess identifizierten Werke werden allerdings nicht in die Ergebnisse der Literaturanalyse einbezogen.

2.2 Open Source IAM Analyse

Nach der Theoriebildung durch die strukturierte Literaturanalyse wird die tokenbasierte Authentifizierung und Autorisierung anschließend in der Praxis anhand der Entwicklung einer Konzeption und prototypischen Implementierung demonstriert. Diese verwenden als Basis eine Open Source IAM-Lösung. Dadurch wird die Umsetzung vieler der in 4.2 beschriebenen Anforderungen unterstützt, ohne dabei zusätzlichen eigenen Implementierungsaufwand zu erfordern. Um eine Vergleichbarkeit zwischen den Open Source IAM-Lösungen zu ermöglichen, wird eine Taxonomie in Anlehnung an das von Nickerson et al. (2013) vorgestellte Verfahren entwickelt.

Nickerson et al. (2013) beschreiben, dass eine Taxonomie im Allgemeinen der

[1] https://qdacity.com/

Bestimmung von Merkmalen einer zu untersuchenden Gruppe von Objekten dient. Um diese zu identifizieren, beginnt man mit der Definition einer Meta Charakteristik. Das Meta-Merkmal ist das umfangreichste Merkmal, welches zur Auswahl der weiteren Merkmale in der Taxonomie verwendet wird. Es beschreibt auf einer hohen Abstraktionsebene, die zu untersuchenden Merkmale der Objekte. Anschließend erfolgt die Festlegung von Endbedingungen, welche aufgrund der iterativen Beschaffenheit des Erstellungsprozesses notwendig sind. Dieser beginnt mit der Auswahl eines Ansatzes auf Basis der verfügbaren Daten und des Wissens des Forschers über die Thematik (Nickerson et al., 2013). In der vorliegenden Arbeit wird ausschließlich der empirisch konzeptionelle Ansatz zur Erstellung einer Taxonomie verwendet, der sich besonders bei einer guten Datenlage eignet. Dafür wird zunächst eine Untergruppe von Objekten bestimmt, die klassifiziert werden soll. Als Nächstes folgt die Identifizierung gemeinsamer Merkmale der Objekte, die zum Abschluss einer Iteration zu Dimensionen der Taxonomie gruppiert werden. Abschließend wird geprüft, ob die Endbedingungen erfüllt sind und eine weitere Iteration erforderlich oder die Erstellung der Taxonomie beendet ist (Nickerson et al., 2013). Die durchgeführten Schritte zur Erstellung der Taxonomie aus Sektion 4.1 sind im Anhang in Abschnitt B nachzuvollziehen.

3 Ergebnisse strukturierte Literaturanalyse

Das Kapitel stellt die Ergebnisse der Literaturanalyse dar. Die jeweiligen Sektionen beziehen sich immer auf eine Forschungsfrage und ihre jeweiligen Untergliederungen. Wie bereits in Kapitel 2 beschrieben, wurde zusätzlich zur strukturierten Literaturanalyse nach Kitchenham (2004) eine Ad hoc Literatur Research durchgeführt, um weitergehende grundlegende Erläuterungen zu ermöglichen.

3.1 Tokenerstellung und Inhalt

Zu Beginn dieses Kapitels wird die Erstellung von Token und deren Inhalt behandelt, um die Forschungsfragen RQ1 und RQ2 aus Kapitel 1 zu beantworten.

3.1.1 Tokenersteller

No.	Beschreibung	Quellen
TE1	Dedizierter Service für die Tokenerstellung	(Nasab et al., 2022) (ShuLin & JiePing, 2020) (Nehme et al., 2019) (Yu et al., 2019) (Nguyen & Baker, 2019) (Bazeniuc & Zgureanu, 2021) (Chandramouli et al., 2021) (Chatterjee & Prinz, 2022) (He & Yang, 2017) (Yarygina & Bagge, 2018) (Chandramouli et al., 2021) (Banati et al., 2018)
TE2	API Gateway als Tokenersteller	(Nasab et al., 2022)

Tabelle 3.1: Tokenersteller in Microservice-Architekturen

Zunächst wird dabei untersucht, an welcher Stelle ein Token in einer Microservice-Architektur erstellt werden soll (s. Tabelle 3.1). Die analysierten Paper liefern dazu zwei unterschiedliche Möglichkeiten. In Nasab et al. (2022) werden Experten zu verschiedenen aus der Literatur herausgearbeiteten Security Praktiken befragt. Eine der Security Praktiken ist die Unterscheidung zwischen internen und externen Microservices. Für die Kommunikation von externen mit internen Microservices werden dann zwei unterschiedliche Arten von Token verwendet. Das Application Programming Interface (API) Gateway dient an dieser Stelle als Ersteller für die Token zur Kommunikation mit internen Microservices (Nasab et al., 2022).

In allen anderen untersuchten Papern werden Token von einem dedizierten Service erstellt. Der Service kann dabei von unterschiedlicher Art sein:

- OAuth2 Server – (Bazeniuc & Zgureanu, 2021; Nasab et al., 2022; Nehme et al., 2019; Nguyen & Baker, 2019; ShuLin & JiePing, 2020; Yu et al., 2019)
- OpenID Connect Provider – (Chandramouli et al., 2021; Chatterjee & Prinz, 2022)
- SSO Server – (He & Yang, 2017)
- Benutzerdefinierter Authentifizierungsprovider – (Banati et al., 2018; Chandramouli et al., 2021; Yarygina & Bagge, 2018)

Wenn man die verschiedenen Varianten der Services betrachtet, so schließen sich diese nicht zwingend gegenseitig aus. Aus der Ad hoc Literatursuche zu Keycloak, einer IAM-Lösung, ergibt sich, dass diese sowohl die Funktionalität des OAuth2 Protokolls als auch OpenID Connect (OIDC) unterstützt und die Funktionalität eines Single Sign-On (SSO) Servers erfüllen kann (Keycloak, n. d.).

OAuth2 bietet nach Preuveneers und Joosen (2017) einen Rahmen für die Zugriffsdelegation von Nutzern. Dabei ermöglicht das Protokoll, dass Aktionen auf Geheiß eines Dritten durchgeführt werden können. Um diese Funktionalität zu realisieren, verwendet es Zugriffstoken, die nach einer erfolgten Authentifizierung ausgestellt werden. Dafür werden in Microservice-Implementierungen nach Preuveneers und Joosen (2017) OAuth-Zugangs-Token in der Regel an einem Token-Austauschpunkt gegen signierte JWT ausgetauscht. Diese werden dann mit jeder Anfrage an die Ressourcenmicroservices weitergeleitet, um dort eine Validierung und Autorisierung zu ermöglichen (Preuveneers & Joosen, 2017).

In OAuth2 wird allerdings nicht definiert, wie eine Authentifizierung durchgeführt werden soll (Abdelfattah & Cerny, 2022). Hierfür wird laut Nehme et al. (2019) in Microservice-Architekturen häufig das Authentifizierungsprotokoll OpenID Connect verwendet, das auf OAuth2 aufbaut.

Aufgrund der Einheitlichkeit in der Literatur, bezüglich des Tokenerstellers, kann die Verwendung eines dedizierten Services für die Erzeugung von Token als

Empfehlung abgeleitet werden.

3.1.2 Anzahl von Tokenarten

In der Literatur finden sich verschiedene Aussagen über die empfohlene Anzahl der Tokenarten sowie ihren jeweiligen Einsatzzweck. Die Anzahl der Tokenarten bewegt sich in den untersuchten Werken zwischen einer und drei (s. Tabelle 3.2).

No.	Beschreibung	Quellen
TA1	Eine Tokenart für externe Kommunikation	(Nasab et al., 2021)
TA2	Eine Tokenart für interne und externe Services	(Banati et al., 2018) (Safaryan et al., 2020) (Nguyen & Baker, 2019) (Yu et al., 2019)
TA3	Zwei Tokenarten - jeweils eine für interne und externe Services	(Nasab et al., 2021) (Chandramouli et al., 2021)
TA4	Zwei Tokenarten - unlesbare Token zur Kommunikation mit Außenstehenden	(He & Yang, 2017)
TA5	Zwei Tokenarten - eines für Nutzerrechte und eines für Nutzerzustimmung	(Nehme et al., 2019)
TA6	Drei Tokenarten - Referenz, Access und Refresh-Token	(Chatterjee & Prinz, 2022)

Tabelle 3.2: Tokenarten in Microservice-Architekturen

Bei den Ansätzen mit einer Tokenart wird dahingehend unterschieden, ob das Token an jeden Microservice weitergeleitet wird oder nicht:

- Eine Tokenart für externe Kommunikation, unter der Annahme, dass interne Microservices nicht erreichbar sind (Nasab et al., 2021)
- Eine Tokenart, welche im Austausch gegen externe Credentials ausgestellt und jedem Request angehangen wird (Banati et al., 2018; Nguyen & Baker, 2019; Safaryan et al., 2020; Yu et al., 2019)

Der erste Ansatz nach Nasab et al. (2021) unterscheidet sich dahingehend, dass innerhalb eines geschützten Bereiches keine Token mehr für den Aufruf von Services benötigt werden. Dies ist nur zu empfehlen, wenn sichergestellt ist, dass keine direkte Kommunikation mit den internen Microservices erfolgen kann. Falls doch eine direkte Kommunikation mit einem oder mehreren Microservices möglich ist, sind die Operationen auf den jeweiligen Ressourcen ungeschützt.

Neben den Varianten mit einem Token gibt es auch Herangehensweisen mit zwei oder drei Tokenarten:

- Zwei Tokenarten, eine für interne, eine für externe Microservices (Chandramouli et al., 2021; Nasab et al., 2021)
- Zwei Tokenarten, eine davon unlesbar für Außenstehende, welche zur Kommunikation mit dem Enduser verwendet wird. Diese wird durch das API Gateway in ein Json Web Token (JWT) für die interne Kommunikation übersetzt (He & Yang, 2017)

- Zwei Tokenarten, ein Identitätstoken für die Nutzerrechte und ein OAuth2 Token für die Nutzerzustimmung (Nehme et al., 2019)
- Drei Tokenarten, vor dem Gateway ein Referenz-/ Access-Token kombiniert mit einem Refresh-Token, hinter dem Gateway ein Wert Token (Chatterjee & Prinz, 2022; Rudrabhatla, 2020)

Die ersten beiden Herangehensweisen nach Chandramouli et al. (2021), He und Yang (2017) und Nasab et al. (2021) bieten beide den Vorteil, dass für die Kommunikation mit externen Teilnehmern separate Tokenarten verwendet werden. Sobald ein Token auf einem Endgerät gespeichert wird, ist die Gefahr des Tokendiebstahls gegeben. Durch diese beiden Varianten lässt sich besser steuern, welche Informationen in einem Token, welches zur externen Kommunikation verwendet wird, enthalten sind. Mit der Variante von He und Yang (2017) lassen sich dabei keine Informationen aus einem gestohlenen Token direkt gewinnen.

Bei Nehme et al. (2019) werden die Token für Rechteprüfungen in einem Regierungsportal genutzt. Dabei wird das ID-Token für die Berechtigungsprüfung von Endnutzern verwendet und das OAuth2 Token für die Freigabe von einzelnen Berechtigungen auf Ressourcen des Nutzers durch den Nutzer (Nehme et al., 2019).

Die letzte genannte Variante nach Chatterjee und Prinz (2022) und Rudrabhatla (2020) nutzt drei Tokenarten, was den Vorteil hat, dass hierbei der Aspekt des Ablaufs und der Erneuerung eines Tokens berücksichtigt wird. Dies kann aber bei jedem der anderen ermittelten Ansätze ebenfalls ergänzend zum Einsatz kommen.

Aus den oben genannten Anwendungsbeispielen geht hervor, dass die Frage, wie viele Tokenarten benötigt werden, vom Einsatzszenario abhängt und nicht pauschal beantwortet werden kann. Bei der Entscheidung müssen verschiedene Aspekte, wie die Kritikalität der in einem Token enthaltenen Informationen und die Sicherheit der Endgeräte, auf denen die Token gespeichert sind, betrachtet werden.

3.1.3 Inhalt eines Tokens

Nachdem im Vorangegangenen die Anzahl der Token betrachtet wurde, soll im Nachfolgenden geklärt werden, welche Informationen ein Token enthalten kann (s. Tabelle 3.3).

No.	Beschreibung	Quellen
TI1	Referenztoken - keine sensitiven Informationen	(Nasab et al., 2022)
TI2	nutzerbezogene Informationen	(Nasab et al., 2022) (Chatterjee et al., 2022) (Banati et al., 2018) (He & Yang, 2017) (Chandramouli et al., 2021) (ShuLin & JiePing, 2020) (Preuveneers & Joosen, 2019) (Yarygina & Bagge, 2018)
TI3	Metadaten wie Erstellzeitpunkt und Ablaufzeitpunkt	(He & Yang, 2017) (Yarygina & Bagge, 2018)
TI4	Tokenersteller	(Chandramouli et al., 2021)
TI5	Berechtigungen des Tokenbesitzers	(Chandramouli et al., 2021) (Salibindla, 2018) (ShuLin & JiePing, 2020) (Preuveneers & Joosen, 2019)
TI6	Autorisierungsrichtlinien	(Preuveneers & Joosen, 2019)
TI7	Gültigkeitsbereich	(Nehme et al., 2019)

Tabelle 3.3: In Token enthaltene Informationen

In der durchgeführten Befragung von Nasab et al. (2022) schlägt ein befragter Software-Architekt eine Unterteilung nach Referenz- und Wert-Token vor. Ein Werttoken wird dabei für die interne Kommunikation von Microservice zu Microservice hinter einem Gateway benutzt und enthält nutzerbezogene Informationen. Das Referenztoken wird hingegen für die Kommunikation zwischen dem Client und einem Gateway benutzt und sollte keinerlei sensitive Informationen enthalten und nur zum Austausch gegen ein Werttoken verwendet werden (Nasab et al., 2022).

Chatterjee und Prinz (2022) verwenden Sicherheitstoken im Rahmen der Security Assertion Markup Language (SAML) 2.0, die Informationen über den Endbenutzer enthalten. Die enthaltenen Informationen können Benutzername, Adresse, E-Mail oder andere benutzerbezogene Daten sein (Chatterjee & Prinz, 2022).

Die meisten gefundenen Paper verwenden JWT, welche sich aber im nutzerbezogenen Informationsgehalt nicht von den in Chatterjee et al. (2022) verwendeten Token unterscheiden. Das JWT enthält ebenfalls Informationen, wie Benutzername, Adresse, E-Mail oder andere im Kontext der Anwendung relevante Daten (Banati et al., 2018; Chandramouli et al., 2021; He & Yang, 2017; Preuveneers & Joosen, 2019; ShuLin & JiePing, 2020; Yarygina & Bagge, 2018).

In He und Yang (2017) wird der Aufbau eines JWT noch genauer definiert. Es besteht aus Header, Payload und Signatur. Der Header enthält dabei den Typ des Tokens und den für das Erstellen verwendeten Algorithmus. Im Payload befinden

sich die Aussagen über eine Entität. Dabei sollen in einem Token alle relevanten Nutzerinformationen enthalten sein sowie ergänzende Metadaten. Die Signatur ist der letzte Bestandteil des Tokens und stellt die Integrität sicher (He & Yang, 2017).
Chandramouli et al. (2021) beschreiben, dass zur Überprüfung der Integrität neben dem Prüfen der Signatur auch noch die Überprüfung des Tokenerstellers notwendig ist, der damit ebenfalls Teil eines Tokens sein sollte. In den ergänzenden Metadaten sind häufig der Erstellzeitpunkt sowie der Ablaufzeitpunkt eines Tokens enthalten (Yarygina & Bagge, 2018).

Neben den Informationen zum Benutzer lassen sich in einem JWT auch dessen Berechtigungen im System speichern, wodurch wiederholte Berechtigungsanfragen an einen Autorisierungsserver vermieden werden können (Chandramouli et al., 2021; Preuveneers & Joosen, 2019; Salibindla, 2018; ShuLin & JiePing, 2020).

Preuveneers und Joosen (2019) codieren für ihren Anwendungszweck eine auf mehreren Parteien basierende Zugangskontrolle. Diese enthält neben den reinen Berechtigungen eines Nutzers noch Autorisierungsrichtlinien in den Token. Autorisierungsrichtlinien und deren Umsetzung mithilfe von Token werden in Sektion 3.3 noch genauer beleuchtet.

Nehme et al. (2019) haben, wie bereits im vorherigen Kapitel beschrieben, ebenfalls einen abweichenden Ansatz. In dem in ihrem Artikel konzipierten Regierungsportal werden zwei Arten von Token verwendet. Eine der beiden Tokenarten wird wie in den anderen Fällen zur Identifikation eines Nutzers und zur Berechtigungsprüfung auf Funktionen des Systems verwendet. Die zweite Tokenart, ein OAuth2-Token, dient dazu, dass ein Nutzer Berechtigungen auf seine Ressourcen vergeben kann. Es enthält eine OAuth-Client-ID, den Scope, für den das Token gültig ist sowie ein Ablaufdatum. Der Scope beschränkt dabei den Gültigkeitsbereich des Tokens, wobei dieser interaktiv durch die Zustimmung des Nutzers bestätigt wird (Nehme et al., 2019).

Token können abgesehen vom Endnutzerbezug auch für die reine Service-zu-Service-Kommunikation verwendet werden. Pontarolli et al. (2021) haben in ihrer Anwendung für jeden Service ein Token, welches zur Kommunikation mit anderen Services verwendet wird. Diese enthalten dabei den Namen eines anfragenden Services und seine zugehörigen Erlaubnisinformationen sind somit bezüglich des Informationsgehalts ähnlich zu den bereits beschriebenen Token mit Endnutzerbezug.

Zusammenfassend kann abgeleitet werden, dass der Inhalt eines Tokens ebenfalls stark vom Anwendungskontext abhängig ist. Es lässt sich jedoch festhalten, dass Token in den meisten aufgezeigten Anwendungsfällen Informationen über den Besitzer eines Tokens sowie dessen Berechtigungen enthalten.

3.1.4 Verbreitung von Informationen über das Gesamtsystem

Neben den Informationen, die einmalig in einem Token eingebettet werden, wie im Vorangegangenen beschrieben, kann es je nach Anwendungsszenario dazu kommen, dass weitere Informationen für Autorisierungsentscheidungen benötigt werden. Es kann etwa sein, dass die Autorisierungsentscheidung, ob eine Aktion erlaubt ist oder nicht, von den Daten eines oder mehrerer anderer Microservices abhängig ist. Hier stellt sich dann die Frage, wie diese benötigten Informationen vom Gesamtsystem aggregiert werden können.

Preuveneers und Joosen (2017) liefern dafür ein Anwendungsbeispiel aus dem Gesundheitswesen. Aufgrund der schützenswerten Daten, die im Medizinkontext verarbeitet werden, sollen unter anderem Patientendaten nur dann für medizinisches Personal zugänglich sein, wenn tatsächlich auch ein Termin für den Patienten vereinbart ist. Die Information, ob ein Termin für einen Patienten vereinbart ist oder nicht, wird nicht in das Token des medizinischen Personals eingebettet, da es keinen Bezug zu diesem hat. Außerdem ist diese Information aufgrund der Trennung von Belangen nicht im Patientendatenservice vorhanden, der die Informationen für Patienten bereitstellt. Dennoch wird diese Information für die Autorisierungsentscheidung, ob einem medizinischen Angestellten Zugriff auf die Daten eines Patienten gewährt werden soll, benötigt. Preuveneers und Joosen (2017) definieren deshalb für Autorisierungsentscheidungen Richtlinien, in denen jeweils spezifiziert ist, wie die benötigten Informationen erhalten werden können. Je nach Anfrage kann es sein, dass ein anderer Microservice die benötigten Informationen enthält oder die Informationen aus eigenen Daten generierbar sind (Preuveneers & Joosen, 2017).

An diesem Beispiel zeigt sich, dass ein Token nicht zwingend alle Informationen beinhaltet, die für Zugriffsentscheidungen benötigt werden. Deshalb kann es notwendig sein, weitere Informationen durch Kommunikation mit anderen Microservices für die Entscheidung einzubeziehen, um die benötigten Informationen für Autorisierungsentscheidungen über das Gesamtsystem zu verbreiten.

3.1.5 Signierung

Neben der Verbreitung von benötigten Informationen für Autorisierungsentscheidungen im Gesamtsystem, spielt auch die Signierung von Token eine wichtige Rolle. Durch die Signierung wird die Integrität und Authentizität von Token gewährleistet, um unbefugte Manipulationen zu verhindern und sicherzustellen, dass die Informationen vertrauenswürdig sind (Luber, 2022).

Grundsätzlich wird bei der Signierung eines Tokens der Header und der Payload des Tokens konkateniert und mit dem Signaturalgorithmus, der im Header spezifiziert

ist, signiert (Yan et al., 2022).

Bei asymmetrischen Signaturverfahren werden Schlüsselpaare, bestehend aus einem privaten und einem zugehörigen öffentlichen Schlüssel, verwendet, um die Signatur zu generieren (ShuLin & JiePing, 2020; Yan et al., 2022). Nach Salibindla (2018) muss dabei beachtet werden, dass niemals ein privater Key zur Erstellung einer Signatur in eine browserseitige Anwendung gelangt. Das liegt daran, dass browserseitige Anwendungen grundsätzlich als nicht sicher gelten, da sie einer Vielzahl an Angriffsvektoren ausgesetzt sind und dadurch nicht sichergestellt werden kann, dass ein im Browser gespeichertes Geheimnis vertraulich bleibt (Salibindla, 2018). Im Falle eines privaten Schlüssels für Signaturverfahren führt dies dazu, dass ein Angreifer Nachrichten erstellen kann, die als authentisch vom eigentlichen Besitzer des privaten Schlüssels gelten (Salibindla, 2018).

Es gibt verschiedene Algorithmen, die zur Erstellung einer Signatur verwendet werden können. In der analysierten Literatur wurden dabei folgende Signierungsalgorithmen genannt:

- HMAC SHA256 (He & Yang, 2017; ShuLin & JiePing, 2020)
- ECDSA SHA256 (Chatterjee et al., 2022; Chatterjee & Prinz, 2022)
- RSA (ShuLin & JiePing, 2020)

Zu betonen ist, dass durch die Signierung des Tokens der Inhalt des Tokens nicht vor dem Ausspähen durch einen Angreifer geschützt ist. Jeder, der Zugriff auf in einer Nachricht enthaltene Token hat, kann diese lesen. Um diesen Exploit zu verhindern, muss ein Token zusätzlich verschlüsselt werden, was ebenfalls möglich, aber optional ist (Salibindla, 2018).

3.1.6 Geheimer Transport

Ein weiterer Weg, um sicherzustellen, dass die Informationen, die in einem Token enthalten sind, nicht ausgelesen werden können, ist der sichere Transport. Dafür müssen Token über einen geschützten Kanal gesendet werden, was sie zusätzlich vor dem Abfangen und Wiederverwenden innerhalb ihrer Gültigkeitsdauer schützt (Yarygina & Bagge, 2018).

In der untersuchten Literatur finden sich nur wenige Angaben zum sicheren Transport von Token. In Pontarolli et al. (2021), Jander et al. (2018) und Pasomsup und Limpiyakorn (2021) wird für die Absicherung der Kommunikation Hypertext Transfer Protocol Secure (HTTPS) verwendet, was die übertragenen Daten verschlüsselt und somit für mehr Sicherheit bei der Übertragung sorgt.

Ein weiterer wichtiger Aspekt für die Sicherheit eines Tokens ist die Gültigkeitsdauer, die im nächsten Abschnitt betrachtet wird.

3.1.7 Gültigkeitsdauer

Die Gültigkeitsdauer eines Tokens wird durch eine im Token enthaltene Verfallszeit angegeben (Ethelbert et al., 2017). Dies ist notwendig, da ansonsten gestohlene Token für eine unbegrenzte Zeit als Identitätsnachweis genutzt werden können (Nehme et al., 2019).

In der analysierten Literatur wird zwischen verschiedenen Lebensdauern von Token unterschieden:

- kurzlebige Token – Gültigkeitsdauer von wenigen Minuten bis zu einer halben Stunde (Banati et al., 2018; Chatterjee et al., 2022; Chatterjee & Prinz, 2022; Rudrabhatla, 2020; Yan et al., 2022; Yarygina & Bagge, 2018)
- Token mit mittlerer Lebenszeit – zwischen 30 Tagen und mehreren Wochen gültig (Chatterjee et al., 2022; Salibindla, 2018)
- langlebige Token – gültig bis zu einem Jahr (Chatterjee et al., 2022)

Je kürzer die Gültigkeit eines Tokens ist, desto häufiger muss eine erneute Authentifizierung oder ein Refresh erfolgen, um wieder einen gültigen Zugang zu einer Anwendung zu erhalten. Das erzeugt Aufwand in der Kommunikation, bietet allerdings auch Sicherheit, da dadurch das Risiko der zu Beginn beschriebenen Problematik minimiert wird (Yarygina & Bagge, 2018).

3.1.8 Widerruf der Gültigkeit innerhalb der Gültigkeitsdauer

Wie bereits im vorherigen Kapitel beschrieben, besteht die Problematik, dass Token, die durch einen Angreifer abgefangen werden, diesem Zugriff auf geschützte Ressourcen ermöglichen. Um dieser Gefahr zu begegnen, ist es notwendig, in einem System den Widerruf von Token zu ermöglichen (s. Tabelle 3.4).

No.	Beschreibung	Quellen
TW1	Widerruf prüfen durch Autorisierungsserver	(Chandramouli et al., 2021)
TW2	Speichern von widerrufenen Token in einer DB	(Nasab et al., 2022) (Salibindla, 2018)
TW3	Speichern von gültigen Token einer DB	(He & Yang, 2017)

Tabelle 3.4: Tokenwiderruf in Microservice-Architekturen

Chandramouli et al. (2021) verweisen auf den Artikel von Lodderstedt et al. (2013) zu OAuth2 Token Introspektion[1], um diese Problematik zu lösen. Darin beschreiben die Autoren, dass ein Autorisierungsserver immer prüfen muss, ob ein

[1]https://www.rfc-editor.org/rfc/rfc7009

Widerruf stattgefunden hat (Lodderstedt et al., 2013). Es wird aber nicht genauer spezifiziert, wie dies umgesetzt werden soll.

Eine konkrete Herangehensweise ist das Speichern von widerrufenen Token in einer Datenbank. Als Datenbanktechnologie wird in Nasab et al. (2022) Redis vorgeschlagen. Redis ist eine In-Memory-Datenbank, die schnellen Zugriff auf Daten ermöglicht (Redis Ltd., 2023). Für die eindeutige Identifikation von Token empfiehlt Salibindla (2018) das Verwenden der JWT ID (JTI) Claims. Die JTI Claims von widerrufenen Token können damit bis zum Ende ihrer Gültigkeitsdauer in einer Datenbanktechnologie wie Redis gespeichert werden (Salibindla, 2018). Services haben dadurch die Möglichkeit entweder direkt durch Kommunikation mit der Datenbank oder einem vorgeschalteten Service die Token zu prüfen.

Einen ähnlichen Lösungsansatz liefern He und Yang (2017) mit dem Vorschlag, die Beziehung von externen und internen Token zu speichern. Abbildung 3.1 zeigt den Authentifizierungsvorgang mit undurchsichtigen Token. Eine zentrale Rolle bei diesem Authentifizierungsvorgang spielt dabei das API-Gateway. Der Nutzer fragt zunächst den Zugriff auf eine Ressource über das API-Gateway an. Das API-Gateway leitet die Anfrage weiter an die angefragte Ressource. Der Server, der die Ressource hält, stellt fest, dass der Request nicht authentifiziert ist und gibt dies über das API-Gateway als Antwort zurück. Daraufhin schickt der Client einen Authentifizierungsrequest an das API-Gateway, welches diesen an den Autorisierungsserver weiterleitet. Der Autorisierungsserver generiert nach erfolgreicher Authentifizierung ein JWT mit den Authentifizierungsinformationen des Nutzers und schickt dieses an das API-Gateway zurück. Im API-Gateway wird nun ein undurchsichtiges Token generiert und beide Token und deren Beziehung in einem Tokenstore gespeichert. An den Client geht nur das undurchsichtige Token zurück. Durch das Löschen des undurchsichtigen Tokens im Tokenspeicher des Gateways kann dieses nun invalidiert werden (He & Yang, 2017).

Nach He und Yang (2017) bietet das undurchsichtige Token zusätzlich den Vorteil, dass nur das API-Gateway die Beziehung zwischen diesem und dem JWT auflösen kann. Somit kann niemand am Gateway vorbei kommunizieren, selbst wenn direkte Routen zu Ressourcen bekannt sind, da ein undurchsichtiges Token von einem Ressourcenservice immer als invalide erkannt wird (He & Yang, 2017).

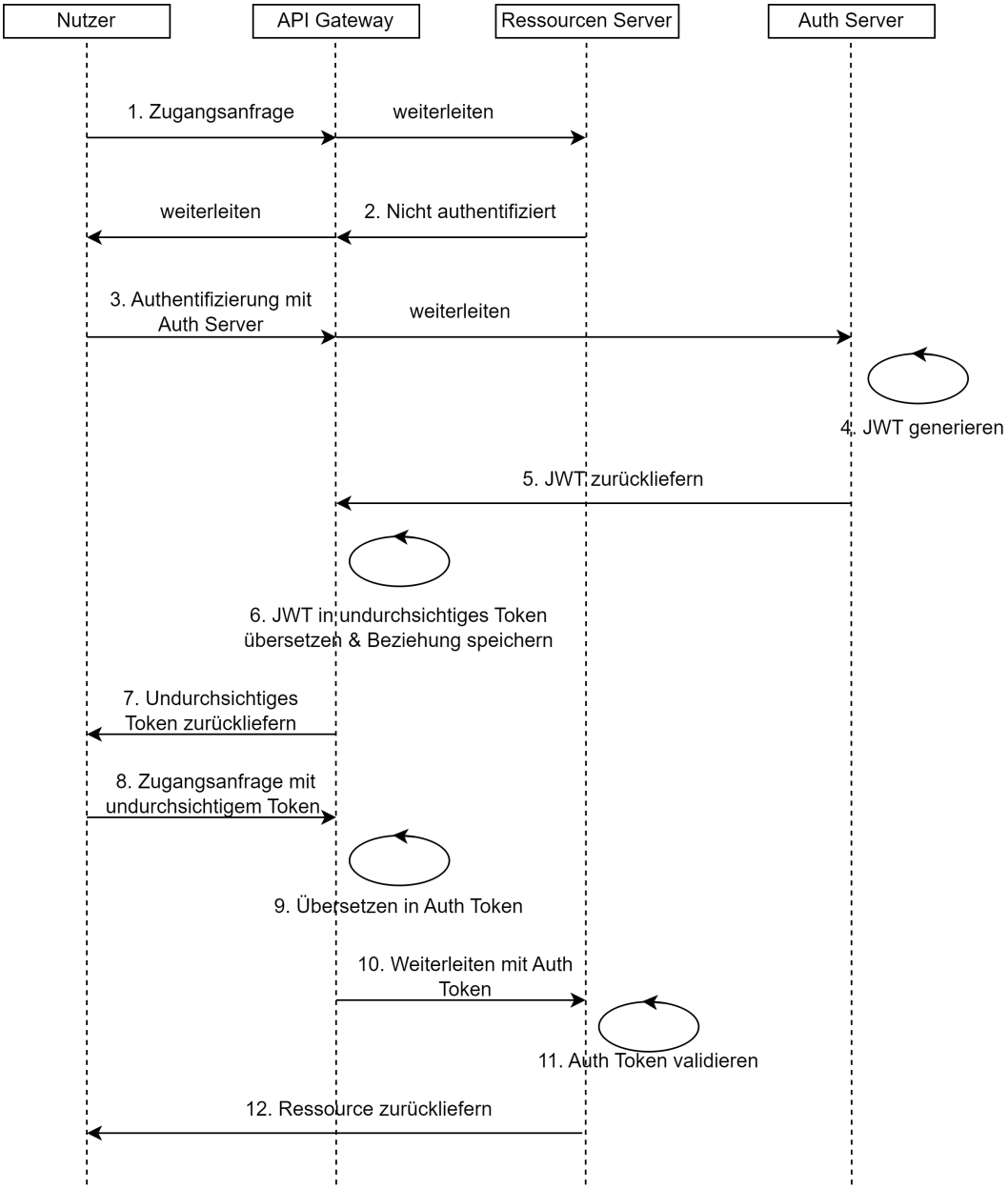

Abbildung 3.1: Authentifizierungsvorgang und Widerruf mit undurchsichtigen Token.
Quelle: In Anlehnung an He und Yang (2017, p. 8)

3.2 Authentifizierung und Autorisierung von Token

Nach Preuveneers und Joosen (2017) ist der erste Schritt in einem Autorisierungsprozess die Authentifizierung, bei welcher die Identität einer Person nachgewiesen wird. Die Authentifizierung muss zwingend vor der Autorisierung erfolgen, da ansonsten keine Garantie für die Validität eines Identitätsnachweises gegeben ist. Ein Token wird durch den Authentifizierungsschritt mit Informationen zur Identität einer Person angereichert, welche anschließend zur Autorisierung verwendet werden können (Preuveneers & Joosen, 2017).

3.2.1 Tokenauthentifizierung

Wichtig ist hierbei die Betrachtung der gesamten Architektur der tokenbasierten Authentifizierung und Autorisierung aus Anwendungssicht. Im Nachfolgenden werden verschiedene Lösungsmöglichkeiten, für eine Implementierung im Microserviceumfeld, präsentiert. Zunächst wird dabei die Architektur aus Authentifizierungsperspektive betrachtet (s. Tabelle 3.5).

No.	Beschreibung	Quellen
TAE1	Zentraler Authentifizierungsdienst	(He & Yang, 2017) (Banati et al., 2018)
TAE2	Zentraler Authentifizierungsdienst in Kombination mit JWT	(Nguyen & Baker, 2019) (Chatterjee et al., 2022) (Preuveneers & Joosen, 2017) (Pontarolli et al., 2021) (He & Yang, 2017)
TAE3	Gateway - Portalmodus	(Yang et al., 2021)
TAE4	Gateway in Kombination mit JWT	(Safaryan et al., 2020) (ShuLin & JiePing, 2020) (Nasab et al., 2022)
TAE5	Gateway in Kombinaiton mit JWT und undurchsichtigen Token	(He & Yang, 2017)

Tabelle 3.5: Tokenauthentifizierung von Endnutzern in Microservice-Architekturen

Zentraler Authentifizierungsdienst

In He und Yang (2017) werden verschiedene Lösungsmöglichkeiten für die Authentifizierung in Microservices aufgezeigt. Eine der Lösungsmöglichkeiten ist das Verwenden eines zentralen Authentifizierungsdienstes, durch den die gesamte Authentifizierung realisiert ist. Ein Anwender, der auf eine Ressource zugreifen möchte, fragt diese direkt beim entsprechenden Server an. Dieser liefert, sofern die Ressource geschützt ist, die Meldung, dass der Anfrager nicht authentifiziert

ist. Daraufhin authentifiziert sich der Anfrager über ein Authentifizierungsverfahren beim Authentifizierungsdienst. Nachdem diese erfolgreich abgeschlossen ist, erhält der Anfragende ein Token, mit welchem erneut eine Anfrage an den Ressourcendienst gestellt wird. Der Ressourcenserver validiert nun das Token durch eine Anfrage gegen den Authentifizierungsdienst und gibt daraufhin nach erfolgreicher Validierung die Ressource zurück (He & Yang, 2017). Der Ablauf ist im Abbildung 3.2 dargestellt.

Banati et al. (2018) verwenden ebenfalls einen zentralen Dienst zur Authentifizierung, jedoch wird die Validierung lokal am Ressourcenserver durchgeführt und lediglich die Prüfung, ob ein Token noch aktiv ist oder nicht, erfolgt zusätzlich über den zentralen Authentifizierungsserver. Der Test auf die Validität eines Tokens erfolgt analog zu Abbildung 3.3.

Nach He und Yang (2017) sind Nachteile dieser Lösung, dass durch den Authentifizierungsserver ein Single Point of Failure entsteht, da bei einem Ausfall die Token nicht mehr überprüft werden können. Außerdem ist durch die notwendige Kommunikation zur Überprüfung der Gültigkeit eines Tokens, im Gegensatz zu anderen Lösungen, ein hoher Kommunikationsaufwand nötig (He & Yang, 2017).

Andererseits lässt sich durch den zentralen Authentifizierungsserver ein bereits beschriebenes Problem gut lösen. Durch den zentralen Ansatz wird der Widerruf der Gültigkeit eines Tokens innerhalb seiner Gültigkeitsdauer sehr einfach, da die Token durch die wiederkehrenden Anfragen an den Server leicht zu invalidieren sind.

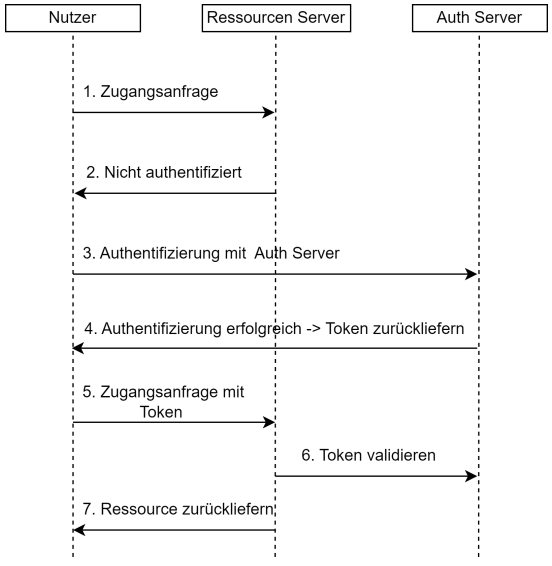

Abbildung 3.2: Authentifizierungsvorgang mit zentralem Server
Quelle: In Anlehnung an He und Yang (2017, p. 5)

Zentraler Authentifizierungsdienst in Kombination mit JWT

Ein ähnlicher Ansatz, der auch in den untersuchten Papern von Nguyen und Baker (2019), Chatterjee et al. (2022), Preuveneers und Joosen (2017) und Pontarolli et al. (2021) verwendet wird, wird ebenfalls von He und Yang (2017) genauer beleuchtet und ist in Abbildung 3.3 zu sehen. Der beschriebene Ansatz wendet ebenfalls einen zentralen Dienst zur Authentifizierung an, bei dem aber im Gegensatz zum vorherigen Ansatz keine Anfragen zur Validität eines Tokens benötigt werden. Durch die Nutzung von signierten JWT können die Ressourcenserver selbst die Signaturen der Token prüfen (He & Yang, 2017).

Diese Lösung von He und Yang (2017) bietet den Vorteil eines verringerten Kommunikationsaufwands. Dennoch bleibt die Problematik eines Single Point of Failures bezüglich des Ausstellens der Token bestehen. Bereits ausgestellte Token sind bis zum Ende ihrer Gültigkeitsdauer aber nicht davon betroffen. Aus diesem Vorteil ergibt sich aber auch der Nachteil dieser Variante. Der Widerruf der Gültigkeit ist mit dieser Lösung nicht ohne weiteres umsetzbar, da Token immer bis zum Ende ihrer Ablaufzeit valide bleiben (He & Yang, 2017).

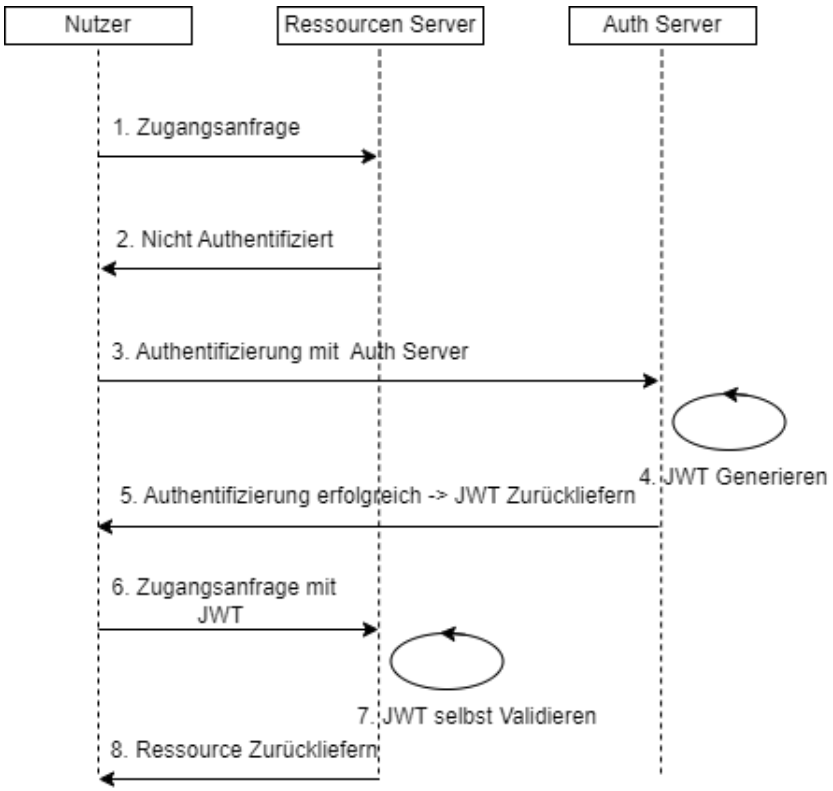

Abbildung 3.3: Authentifizierungsvorgang mit zentralem Server und JWT
Quelle: In Anlehnung an He und Yang (2017, p. 7)

Gateway

Nach Yang et al. (2021) ist ein Gateway eine weitere Möglichkeit eine Authentifizierung in einer Microservice-Architektur durchzuführen. Eine der Varianten, die durch ein Gateway realisiert werden können, ist das Verwenden des sogenannten Portalmodus. Dieser unterscheidet die Kommunikation von internen Microservices von der Kommunikation mit externen Systemen bzw. Anwendern. Letztere ist durch eine Authentifizierung am Gateway geschützt. Nach erfolgreicher Authentifizierung ist die restliche stattfindende interne Kommunikation nicht mehr durch weitere Sicherheitsmaßnahmen wie Token geschützt. Der interne Bereich wird als bereits geschützt angesehen, in den keine Anfragen von außerhalb eindringen können. Dies kann aber ein großes Sicherheitsproblem darstellen, denn falls es einem Angreifer doch gelingt in diesen geschützten Bereich mit Anfragen vorzudringen, kann er auf alle Ressourcen ohne weitere Sicherheitsprüfungen zugreifen (Yang et al., 2021).

Gateway in Kombination mit JWT

Um dieses Sicherheitsproblem zu vermeiden, kann die interne Kommunikation hinter dem Gateway ebenfalls mit JWT geschützt werden (Nasab et al., 2022; Safaryan et al., 2020; ShuLin & JiePing, 2020). Safaryan et al. (2020) beschreiben zwei Wege, wie diese Variante umsetzbar ist. In beiden Fällen wird ein Authentifizierungsdienst an das Gateway angebunden. Im ersten Fall leitet das Gateway Anfragen immer an den Authentifizierungsdienst weiter und führt keine eigene Prüfung der Token durch. Dies hat den Vorteil, dass das Gateway keinerlei Logik hinsichtlich der Tokenvalidierung enthalten muss und damit weniger komplex ist. Das Problem dabei ist aber, dass, analog zu dem im Abschnitt zentraler Authentifizierungsdienst beschriebenen Vorgehen, ein hoher Kommunikationsaufwand entsteht (Safaryan et al., 2020).

Im zweiten Fall, den Safaryan et al. (2020) aufzeigen, erfolgt die Prüfung des Tokens direkt am Gateway. Somit hat der Authentifizierungsdienst nur noch die Aufgabe, die Anmeldedaten eines Nutzers zu prüfen und darauf basierend Token zu erstellen.

ShuLin und JiePing (2020) zeigen in ihrem Artikel eine praktische Implementierung dieser Variante auf, indem Zuul[2] als Gateway-Lösung verwendet wird. Bei der initialen Anfrage eines Nutzers wird im OAuth2-Server ein JWT mit einem privaten Schlüssel signiert, generiert und an das Gateway zurückgegeben. Das Gateway kann nun eintreffende Anfragen, die bereits ein Token enthalten, direkt mit seinem öffentlichen Schlüssel auf Validität überprüfen und muss keine weitere Anfrage an den in dieser Implementierung verwendeten OAuth2-Server stellen (ShuLin & JiePing, 2020).

[2] https://github.com/Netflix/zuul

Gateway in Kombination mit JWT und undurchsichtigen Token

Eine weitere Ausprägung dieser Variante wurde bereits in Sektion 3.1 aufgezeigt. In den vorherigen Abwandlungen besteht immer noch die Problematik des Tokenwiderrufs. Diese wird hier durch die bereits beschriebene Kombination aus JWT und undurchsichtigen Token gelöst.

Ein Nachteil dieser Variante ist die durch die Lösung entstehende Komplexität im API-Gateway. In diesem muss dadurch die Verwaltung und Translation von Token vorgenommen werden. Da jeder Request durch das Gateway geht, erhöht dies potenziell die Laufzeit und Fehleranfälligkeit in Hinblick auf mögliche Ausfälle der Tokenstore Datenbank. Der große Vorteil dieser Variante ist die erhöhte Sicherheit, da die Problematik des Gateway-Bypass gelöst wird.

Service-zu-Service

Alle bisherigen Varianten haben sich mit der Authentifizierung von Endnutzern beschäftigt. Es gibt aber auch Anwendungsfälle, in denen eine reine Service-zu-Service Authentifizierung ohne Nutzerkontext notwendig ist. Beispielsweise, wenn in einer Anwendungsarchitektur nicht alle Services mit Endnutzerbezug ausgestattet sind. Zu dieser Art von Authentifizierung stellen Barabanov und Makrushin (2020) eine auf Token basierende Variante vor.

In dem Ansatz von Barabanov und Makrushin (2020) werden zur Kommunikation zwischen den Services Token ausgetauscht. Diese Token enthalten eine ID, mit der sich der aufrufende Microservice identifiziert und die zusätzlich seine Berechtigungen enthalten können. Um ein Token zu erhalten, muss ein Service mit einem vorher definierten, für jeden Service individuellen Passwort einen Aufruf an den zuständigen Service-zu-Service Authentifizierungsdienst tätigen. Im Artikel wird jedoch darauf hingewiesen, dass für die Authentifizierung der Microservicekommunikation Mutual Transport Layer Security den tokenbasierten Ansätzen vorgezogen wird (Barabanov & Makrushin, 2020).

Nach der Betrachtung von verschiedenen Ansätzen zu Authentifizierung auf Basis von Token soll nun im nächsten Abschnitt die Autorisierung genauer untersucht werden.

3.2.2 Tokenautorisierung

Nach einer erfolgreichen Authentifizierung muss vor dem Zugriff auf eine geschützte Ressource eine Autorisierung erfolgen. Durch diesen Vorgang wird festgestellt, ob ein Anwender berechtigt ist, Zugang zu einer bestimmten Ressource zu erhalten (Okta, 2023b). Bei tokenbasierter Autorisierung werden die Informationen, die in einem Token enthalten sind, zur Autorisierung genutzt (Yarygina & Bagge, 2018). Dadurch ist aber noch nicht geklärt, an welcher Stelle diese Prüfung erfolgt

und ob die im Token enthaltenen Informationen in jedem Fall ausreichend sind. Analog zur Authentifizierung gibt es ebenfalls wieder verschiedene Varianten, wie die Autorisierung in einer Microservice-Architektur auf Basis von Token realisierbar ist. Im Folgenden werden diese mit Bezug zu der vorher notwendigen Authentifizierung 3.2.1 betrachtet. (vgl. Tabelle 3.6)

No.	Beschreibung	Quellen
TAO1	Autorisierung am API-Gateway	(Safaryan et al., 2020) (Barabanov & Makrushin, 2020)
TAO2	Autorisierung am Ressourcenmicroservice	(Nasab et al., 2022) (Yarygina & Bagge, 2018) (ShuLin & JiePing, 2020) (Rudrabhatla, 2020) (ShuLin & JiePing, 2020) (Bazeniuc & Zgureanu, 2021) (Preuveneers & Joosen, 2017) (Salibindla, 2018)
TAO3	Autorisierung am API-Gateway und am Ressourcenmicroservice	(Nasab et al., 2022)
TAO4	Autorisierung am Sidecarproxy eines Service Meshes	(Chandramouli et al., 2021)

Tabelle 3.6: Tokenautorisierung in Microservice-Architekturen

Autorisierung am API-Gateway

Die erste Möglichkeit ist die Autorisierung am API-Gateway (Barabanov & Makrushin, 2020; Safaryan et al., 2020). Erfolgt diese in Kombination mit der in Sektion 3.2.1 im Abschnitt Gateway beschriebenen Authentifizierungsvariante, hat dies den Vorteil, dass die Autorisierung und Authentifizierung nur an einem zentralen Punkt erfolgen muss.

Alle anderen nachfolgenden Services müssen sich dadurch nicht mehr mit dieser Aufgabe befassen und können sich auf ihre Businessfunktionalität fokussieren. Allerdings muss bei dieser Variante darauf geachtet werden, dass keine Aufrufe das Gateway umgehen können, da diese sonst unautorisiert Zugriff auf Ressourcen erhalten. Ein weiterer Nachteil dieser Variante ist, dass die Autorisierung im Gateway in einem komplexen System mit vielen Endpunkten schnell unübersichtlich und schwer anpassbar werden kann. Ferner ist in Organisationen häufig ein dediziertes Operations Team für das API-Gateway zuständig, wodurch es Entwicklern nicht mehr ohne Weiteres möglich ist, Änderungen an der Autorisierung ihres Services durchzuführen (Barabanov & Makrushin, 2020).

Autorisierung am Ressourcenmicroservice

Neben der Autorisierung am API-Gateway gibt es auch die Möglichkeit eine dezentrale Autorisierung an den jeweils angeforderten Microservices durchzuführen. Dieses Pattern wird in der Literatur in verschiedenen Varianten vorgestellt. ShuLin

und JiePing (2020) und Bazeniuc und Zgureanu (2021) setzen dabei ein Zugangskontrollsystem ein, welches auf dem OAuth2 Protokoll basiert. Dieses verwendet JWT, um lokal am Ressourcenmicroservice die Autorisierung zu realisieren. Diese erfolgt auch hier auf Basis der im Token enthaltenen Userinformationen (Bazeniuc & Zgureanu, 2021; ShuLin & JiePing, 2020).

Analog dazu wird in Preuveneers und Joosen (2017) erläutert, dass Representational State Transfer (REST) Endpunkte auf Basis von Attributen Zugriff auf Ressourcen erlauben oder verweigern können. Dies basiert laut Preuveneers und Joosen (2017) häufig auf Domänen relevanten Attributen, die in der angefragten Ressource enthalten sind. Etwa auf dem Namen eines Patienten aus einer Krankenakte oder der Berechnung beziehungsweise Ableitung von Attributen aus dieser Krankenakte, wenn sie ressourcenintensiv sind (Preuveneers & Joosen, 2017).

Salibindla (2018) schlägt bei dieser Variante den Einsatz von Middleware in einem Microservice vor.
Ein Framework wie z. B. ExpressJS[3] bietet Komponenten wie express-jwt[4], das, sofern ein JWT vorhanden ist, dieses dekodiert und an die Request Klasse des Frameworks anfügt. Dadurch sind die im Token enthaltenen Informationen in dem entsprechenden Endpunkt verfügbar und auf Basis dieser können Autorisierungsentscheidungen getroffen werden (Okta, 2023a).

Autorisierung am API-Gateway und am Ressourcenmicroservice

In Nasab et al. (2022) wird eine Kombination der beiden Ansätze vorgeschlagen, in dem eine grob granulare Autorisierung am API-Gateway und eine fein granulare Autorisierung direkt am jeweiligen Microservice erfolgt. Die Kombination der beiden Varianten bietet den Vorteil, dass unautorisierte Anfragen bereits früh erkannt werden und scheitern, wodurch der Kommunikationsaufwand verringert wird. Dadurch werden die Ressourcenmicroservices mit weniger Anfragen belastet (Nasab et al., 2022).

Autorisierung am Sidecar Proxy eines Service Meshes

Neben den bereits genannten Möglichkeiten zeigen Chandramouli et al. (2021) eine neue Möglichkeit durch das Verwenden eines Service Meshes auf. Im vorliegenden Artikel wird die Variante anhand von Istio[5], einem Service Mesh für Kubernetes, erläutert. Die Autorisierung von Endbenutzern auf der Grundlage eines JWT ist bereits in Envoy[6], dem Sidecar Proxy des Referenznetzes Istio, integriert.

[3] https://expressjs.com/de/
[4] https://www.npmjs.com/package/express-jwt
[5] https://istio.io/
[6] https://www.envoyproxy.io/

Nach Chandramouli et al. (2021) ist Envoy durch einen Filter konfigurierbar, der Anfragen in zwei Schritten verarbeitet:

1. Verifizierung des JWT durch Extrahieren des Tokens aus dem Anfrage-Header, Überprüfung, ob Aussteller und Publikum erlaubt sind, Abrufen des öffentlichen Schlüssels inklusive Überprüfung der digitalen Signatur des Tokens
2. Abgleich der Ressourcen in der Anfrage mit den Informationen im Token, um zu bestimmen, ob dem Endbenutzer der Zugriff auf die angeforderten Ressourcen erlaubt oder verweigert werden soll

Die Architektur dieser Variante ist in der nachfolgenden Abbildung 3.4 dargestellt. Durch die Abbildung wird deutlich, dass die Verarbeitung der Anfragen ebenfalls im Container einer Microserviceanwendung, jedoch nicht im Microservice selbst, stattfindet.

Die Verarbeitung im Sidecar Proxy des Containers hat den Vorteil, dass keine Implementierungslogik im Microservice selbst vorhanden sein muss und der Service selbst dadurch weniger Komplexität beinhaltet. Die Komplexität wird dadurch in den Service Mesh verlagert. Dies führt dazu, dass für Änderungen an der Autorisierung eines Services Know-how im Bereich von Service Meshes notwendig ist (Chandramouli et al., 2021).

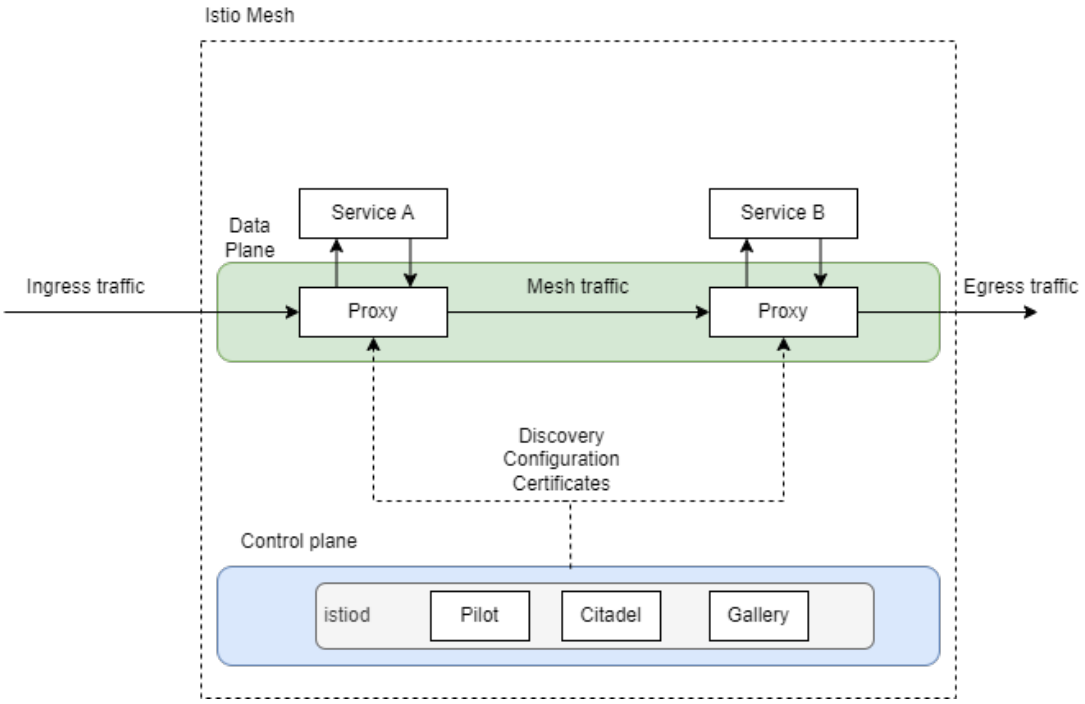

Abbildung 3.4: Istio Architektur
Quelle: In Anlehnung an Istio (n. d.)

3.3 Umsetzung von Autorisierungspolicies im Microserviceumfeld auf Basis von Token

Nachdem nun die Frage beantwortet wurde, wie mithilfe von Token ein Nutzer authentifiziert und autorisiert werden kann, ist es wichtig, auf potenzielle Herausforderungen bei der Umsetzung dieser Methode hinzuweisen. Wie bereits diskutiert, ermöglicht die direkte Implementierung von Autorisierungsentscheidungen auf Basis von Token im Code von Microservices zwar eine vollständige Kontrolle des Entwicklungsteams über den Zugriff auf ihren Service, allerdings birgt dies auch gewisse Risiken. Insbesondere setzt dies voraus, dass das jeweilige Entwicklungsteam das gesamte Sicherheitskonzept der Microservicelandschaft verstanden hat, da ansonsten inkonsistente, genauer gesagt unerwartete Berechtigungskonzepte für Anwender entstehen können (Barabanov & Makrushin, 2020).

Um dieses Problem zu umgehen und eine bessere Verwaltung der Zugriffskontrollen zu ermöglichen, bietet sich nach Barabanov und Makrushin (2020) die Auslagerung von Autorisierungspolicies aus dem Code an. Dies ermöglicht es, das Sicherheitskonzept der gesamten Microservicelandschaft separat zu verwalten und erleichtert die Wartung sowie die Konsistenz der Berechtigungen für die Benutzer. Durch die Auslagerung von Autorisierungspolicies können komplexe Zugriffskontrollen realisiert werden, die auf einer höheren Abstraktionsebene operieren und eine klar definierte Trennung zwischen Sicherheitsaspekten und Geschäftslogik schaffen. Diese Vorgehensweise führt zu einer verbesserten Wartbarkeit und Flexibilität des Systems und trägt zur Konsistenz und Effektivität der Zugriffsverwaltung in komplexen Systemen bei (Barabanov & Makrushin, 2020).

3.3.1 Definition von Autorisierungspolicies

Dabei stellt sich zunächst die Frage, was Autorisierungspolicies sind und wie diese Zugangsregeln definiert werden können. In Preuveneers und Joosen (2019) werden richtlinienbasierte Zugangskontrollmodelle als eine Lösung für die Erteilung von Berechtigungen auf der Grundlage einer Richtlinie, die außerhalb von der eigentlichen Serviceimplementierung definiert ist, beschrieben. Diese sind aus logischen Regeln zusammengesetzt, um je nach Benutzeranfrage den Zugang zum jeweiligen Dienst oder den jeweiligen Ressourcen zu erlauben oder zu verweigern (Preuveneers & Joosen, 2019).

Eine ähnliche Definition findet sich in Preuveneers und Joosen (2017), bei der ebenfalls beschrieben wird, dass bei einer richtlinienbasierten Zugriffskontrolle die Autorisierung aus einer Anwendung ausgelagert wird. Um sicherzustellen, ob eine Aktion von einem Subjekt auf einer Ressource durchgeführt werden darf, wird ein Satz von Zugriffsregeln ausgewertet. Überdies wird noch beschrieben, dass es bestimmte Anwendungsdomänen gibt, bei denen die Autorisierung eher

einem Workflow entspricht als einem Satz von Regeln, der atomar ausgewertet werden kann. Als Beispiel hierfür werden Anwendungsfälle aus dem medizinischen Bereich genannt, bei denen die Prozesse häufig als Arbeitsabläufe abgebildet sind, die verschiedene Akteure beinhalten. Dadurch entsteht eine gewisse Komplexität in der Zugriffskontrolle, die nicht einfach durch starre Regeln abgebildet werden kann. Einer dieser Anwendungsfälle betrifft die Zustimmung eines Patienten. Ein Autorisierungssystem soll beispielsweise in diesem Kontext nicht nur prüfen, ob ein Patient jemals die Einwilligung zur Einsicht von Daten gegeben hat, sondern der Patient soll diese Entscheidung mittels einer Abfrage immer wieder erneut erteilen. In Notfallsituationen kann es wiederum aber erforderlich sein, dass diese Zustimmung aufgehoben werden kann (Preuveneers & Joosen, 2017).

Der im Vorangegangenen beschriebene Anwendungsfall zeigt, dass die Definition von Zugriffskontrollen und Autorisierung im Allgemeinen stark Domänen abhängig ist. Als Möglichkeit, die Zugriffskontrollen in der eigenen Domäne zu modellieren, beschreiben Safaryan et al. (2020) die Vorgehensweise, Benutzerrechte in Form einer Tabelle darzustellen. Die Tabelle beinhaltet alle Ressourcen, auf die ein Benutzer berechtigt ist, sowie die jeweiligen Berechtigungsarten, die der Benutzer in Bezug auf ein Objekt hat.

In Chandramouli et al. (2021) wird die Konfiguration von Autorisierungsrichtlinien innerhalb von Service Meshes betrachtet. Bei diesen Richtlinien ergeben sich aus Sicht der Autoren folgende Variablen in Hinblick auf die Richtlinien:

- Zwei Autorisierungsebenen – Serviceebene und Endnutzerebene
- Access Control Model – zur Beschreibung der Autorisierungspolicies
- Speicherort der Zugangskontrolldaten – zentraler oder externer Autorisierungsserver oder als Headerdaten

Aus Sicht eines Service Meshes unterscheidet der Artikel von Chandramouli et al. (2021) anhand der Variablen dabei drei Arten von Berechtigungsrichtlinien:

- Autorisierungsrichtlinien auf Dienstebene
- Autorisierungsrichtlinien auf Endnutzerebene
- Modellbasierte Autorisierungsrichtlinien

Autorisierungsrichtlinien auf Dienstebene werden in Nasab et al. (2022) ebenfalls als notwendig beschrieben. Diese sind dafür zuständig, die Erlaubnis eines Zugriffes zwischen Services zu definieren. Die Richtlinien für die Service-zu-Service-Kommunikation sollen dabei für alle Services in einem Dienstnetz vorhanden sein. Dabei wird empfohlen, die Richtlinien mindestens auf der Ebene von Namespaces in einem Service Mesh zu definieren. Idealerweise erfolgt die Definition zwischen einzelnen Services. Dabei sollten immer nur minimale Zugriffsrechte, die für die

Umsetzung der Anwendungsfunktionalität erforderlich sind, vergeben werden (Chandramouli et al., 2021).

Bei den Autorisierungsrichtlinien auf Endnutzerebene geht es laut Chandramouli et al. (2021) darum, die Berechtigung von Endnutzern, anhand ihrer durch eine erfolgreiche Authentifizierung erworbenen Token sowie den darin enthaltenen Aussagen über den Besitzer des Tokens, zu prüfen. In den Richtlinien werden dann Regeln definiert, wer auf eine Ressource zugreifen darf. Dabei kann der Zugriff anhand von Informationen aus dem Token überprüft werden. Zum Beispiel kann eine Regel lauten, dass nur Token, die von einem bestimmten Issuer erzeugt wurden, auf eine Ressource zugreifen dürfen (Chandramouli et al., 2021).

Zusätzlich zu den Autorisierungsrichtlinien auf Dienst und Endnutzerebene werden durch Chandramouli et al. (2021) im Rahmen von Service Meshes noch die Modell-basierten Autorisierungsrichtlinien vorgestellt. Diese erfordern den Aufruf eines externen Autorisierungsservers. Der Autorisierungsserver trifft die Autorisierungsentscheidungen dann basierend auf dem jeweiligen hinterlegten Modell (Chandramouli et al., 2021).

Im Allgemeinen sollen die Richtlinien nach Chandramouli et al. (2021) folgende Komponenten beinhalten:

- Richtlinienart – Unterscheidung zwischen Genehmigungs- oder Verbotsrichtlinie
- Richtlinienziel – Zielressourcen in der Form von Diensten oder Dienstrepräsentationen
- Richtlinienquellen – Zur Anfrage berechtigte Dienste
- Richtlinienoperationen – Im Falle von REST-Ressourcen POST, GET, PUT, PATCH und DELETE
- Richtlinienbedingungen – Beschränkungen in Form von Schlüssel-Wert Paaren, die anhand der mit der Anfrage verbundenen Metadaten ausgewertet werden

3.3.2 Access Control Models

Nach der vorausgehenden Definition von Autorisierungspolicies, sollen nun im Folgenden verschiedene bekannte Access Control Modelle genauer betrachtet werden. Diese ermöglichen eine formalisierte Darstellung von Autorisierungsrichtlinien, wodurch sich Aussagen über die Sicherheit eines entworfenen Autorisierungskonzeptes treffen lassen. Damit bieten sie einen Rahmen für das Schützen von Ressourcen innerhalb eines Informationssystems (Samarati & de Vimercati, 2001). Dabei wurden die nachfolgenden Modelle in der Literatur identifiziert (s. Tabelle 3.7).

Access Control Model	Erwähnungen in der Literatur
DAC	(Preuveneers & Joosen, 2017) (Banati et al., 2018)
MAC	(Preuveneers & Joosen, 2017) ((Banati et al., 2018)
RBAC	(Preuveneers & Joosen, 2017) (Banati et al., 2018) (ShuLin & JiePing, 2020) (Pasomsup & Limpiyakorn, 2021) (Yarygina & Bagge, 2018) (Nasab et al., 2022)
SRBAC	(Banati et al., 2018)
Cloud-optimized RBAC	(Banati et al., 2018)
distributed RBAC	(Banati et al., 2018)
Hierarchical Trust RBAC	(Pasomsup & Limpiyakorn, 2021)
ABAC	(Preuveneers & Joosen, 2017) (Preuveneers & Joosen, 2019) (Banati et al., 2018) (Pasomsup & Limpiyakorn, 2021) (Yarygina & Bagge, 2018) (Chandramouli et al., 2021)

Tabelle 3.7: Identifizierte Zugriffskontrollmodelle in der Literatur

DAC und MAC

Discretionary Access Control (DAC) und Mandatory Access Control (MAC) sind zwei bekannte identitätsbasierte Zugangskontrollmodelle (Preuveneers & Joosen, 2017). Sie wurden in der analysierten Literatur jedoch wenig beachtet, sollen hier aber dennoch kurz erläutert werden.

Nach Jordan (1987) dient DAC zur Beschränkung des Zugriffs auf Objekte auf der Basis der Identität von Subjekten und/oder Gruppen, denen sie angehören. Dabei ist DAC nach Jordan (1987) diskretionär in dem Sinne, dass ein Benutzer oder Prozess, dem diskretionärer Zugang zu Informationen gewährt wird, die Freiheit hat, diese Informationen an ein anderes Subjekt weiterzugeben. Das heißt, dass die Entscheidung über den Zugang und die Weitergabe von Informationen nach eigenem Ermessen getroffen werden kann, basierend auf den Berechtigungen und Rechten, die dem Benutzer oder Prozess zugewiesen wurden (Jordan, 1987).

Die Aufgabe von MAC ist laut Dukes (2015) ebenfalls den Zugriff von Nutzern auf Ressourcen zu beschränken. Hierbei beschreibt Dukes (2015) MAC als eine Zugangskontrollpolitik, die einheitlich für alle Subjekte und Objekte innerhalb

der Grenzen eines Informationssystems durchgesetzt wird. Wenn ein Subjekt die Erlaubnis hat, auf Informationen zuzugreifen, so ist es diesem im Gegensatz zu DAC untersagt, die Informationen an nicht autorisierte Subjekte oder Objekte weiterzugeben. Außerdem kann es die Sicherheitsattribute von Subjekten, Objekten oder von Systemkomponenten des Informationssystems nicht verändern. Des Weiteren ist es nicht möglich für neu erstellte Objekte auszuwählen, welche Sicherheitsattribute diesen zugeordnet werden oder im Allgemeinen Regeln für die Zugriffskontrolle zu ändern. Dies erfolgt nur durch speziell berechtigte Nutzer der Organisation (Dukes, 2015).

RBAC, SRBAC, verteilte RBAC, Cloud-optimized RBAC und HT-RBAC

Im Folgenden wird auf das in der Literatur häufig behandelte Role Based Access Control (RBAC) eingegangen. Laut Banati et al. (2018) kombiniert RBAC DAC und MAC. Das liegt daran, dass es ermöglicht, Rollen zu definieren, die den Zugang zu Ressourcen beschränken und gleichzeitig ist durch die Rollen definierbar, welche Rollen den Zugang zu Ressourcen weitergeben dürfen. Dabei folgen die Rollen in RBAC nach Banati et al. (2018) dem Minimal-Rechteprinzip. Folgende Modifikationen von RBAC für unterschiedliche Anwendungszwecke werden im Artikel von Banati et al. (2018) erwähnt:

- distributed RBAC
- Cloud-optimized RBAC
- RBAC for Software as a Service

ShuLin und JiePing (2020) beschreiben, dass sie für ihre Autorisierungsrichtlinien hauptsächlich auf RBAC setzen. Durch die Zuweisung von Rollen und die entsprechende Konfiguration der Benutzerrechte soll eine erhebliche Steigerung der Flexibilität im System erreicht werden (ShuLin & JiePing, 2020).

In Pasomsup und Limpiyakorn (2021) wird eine Erweiterung des RBAC Modells für containerbasierte Anwendungen vorgeschlagen. Das Modell führt eine Vertrauenshierarchie ein, welche mehrere Microservice Domänen umfassen kann. Das Modell kann auf Vererbungsbeziehungen zwischen Microservices angewandt werden, was es weniger komplex machen soll als das im Nachgang betrachtete Attribute Based Access Control (ABAC). Außerdem unterstützt es eine zustandslose Authentifizierung (Pasomsup & Limpiyakorn, 2021).

Zum klassischen RBAC erklären Pasomsup und Limpiyakorn (2021), dass Benutzer ihre Rechte auf der Basis ihrer Zuständigkeiten erhalten. Außerdem wird beschrieben, dass das Kernmodell aus fünf Datenelementen besteht, die zueinander in Beziehung stehen:

- Benutzer
- Rollen
- Objekte
- Operationen
- Berechtigungen

Dabei wird einer Rolle eine Menge an Berechtigungen zugewiesen, die aus erlaubten Operationen auf Objekten besteht. Eine Rolle kann wiederum eine hierarchische Beziehung mit anderen Rollen eingehen und Nutzern werden immer Rollen zugewiesen (Pasomsup & Limpiyakorn, 2021).

Zu RBAC finden sich auch Vorschläge in der Literatur, wie sich dieses Modell mithilfe von Token umsetzen lässt. Yarygina und Bagge (2018) schlagen dazu vor, dass die Rollen eines Nutzers in JWT eingebettet werden können. Dadurch lassen sich in einem Microservice die durch RBAC definierten Richtlinien in Kombination mit den im Token enthaltenen Informationen prüfen und dadurch Autorisierungsentscheidungen treffen.

ABAC

Zu ABAC finden sich, wie in der oben stehenden Tabelle 3.7 dargestellt, ebenfalls einige Erwähnungen in der Literatur. Im Folgenden wird dieses Modell genauer betrachtet und die Erkenntnisse aus der Literatur zusammengetragen.

Von Preuveneers und Joosen (2017) wird beschrieben, dass ein wachsendes Interesse durch die erhöhte Flexibilität und Ausdruckskraft entstanden ist. Chandramouli et al. (2021) sehen ABAC als einen vielversprechenden Ansatz für die Unterstützung bei der Definition einer Vielzahl von Autorisierungsrichtlinien und Yarygina und Bagge (2018) empfehlen das Modell für feingranulare Autorisierung auf Ressourcen.

Preuveneers und Joosen (2017) liefern eine Definition für ABAC: Dabei wird beschrieben, dass Richtlinien und Regeln durch die Kombination verschiedener Attribute definiert werden. Dadurch entstehen Zugriffsrechte für Subjekte, die wiederum den Nutzerzugang zu Ressourcen unter bestimmten Bedingungen ermöglichen. Nach Preuveneers und Joosen (2017) gibt es in Hinblick darauf folgende Arten von Attributen:

- Ressource – auf was soll zugegriffen werden (z. B. Daten oder Microservice)
- Subjekt – wer fragt die Ressource an
- Aktion – welche Operation soll auf der Ressource durchgeführt werden (z. B. Lesen oder Schreiben)

- Umgebung – welche Attribute liegen unabhängig von Ressource, Subjekt oder Aktion noch vor

Ein Nachteil, den Banati et al. (2018) im Zusammenhang mit dem Autorisierungsframework nennen, ist, dass bei jedem Zugriff eines Benutzers innerhalb einer Cloud-basierten Anwendung eine große Anzahl von Attributen verarbeitet werden muss.

Yu et al. (2019) verwenden ABAC in Kombination mit OAuth2 und beschreiben hingegen, dass ABAC vor allem für sehr heterogene Umgebungen wie Multi-Cloud Umgebungen geeignet ist.

Chandramouli et al. (2021) erläutern die Verwendung von ABAC im Rahmen von Service Meshes. Dabei ergeben sich aus Sicht der Autoren verschiedene Vorteile. Sie beschreiben, dass aufgrund der Beschaffenheit von Cloud-Native Anwendungen, die häufig verschiedene Domänen umfassen, ein großer Satz von Variablen zur Spezifikation von Richtlinien notwendig ist. ABAC eignet sich dafür aufgrund seiner Skalierbarkeit in Bezug auf Attribut-Wert-Speicher und zugehörige Richtlinien. Dabei lassen sich die Attribute und ihre Werte für unterschiedliche Attributarten auch von unterschiedlichen Teilnehmern des Entwicklungsprozesses definieren. So beschreiben die Autoren, dass Attribute, die mit Subjekten verbunden sind, von Systemadministratoren definiert werden können. Wohingegen diejenigen, die mit Anwendungsobjekten und der Umgebung verbunden sind, von Entwicklern unabhängig zugewiesen werden können. Des Weiteren beschreiben Chandramouli et al. (2021), dass eine Richtlinie eine Regel ist, die festlegt, ob eine Aktion erlaubt oder verboten ist. Diese basiert auf bestimmten Eigenschaften desjenigen, der die Aktion ausführt, des Objekts, auf das die Aktion angewendet wird, und der Umgebung, in der die Aktion stattfindet. Diese Eigenschaften können unterschiedliche Werte haben und können sich im Laufe der Zeit verändern. Die Entscheidung, ob eine Anfrage akzeptiert oder abgelehnt wird, hängt allein von den Werten dieser Eigenschaften zum Zeitpunkt der Anfrage ab. Es besteht also keine feste Verbindung zwischen einer bestimmten Person oder Sache und einer spezifischen Aktion. Stattdessen basiert die Zugriffsentscheidung auf den aktuellen Eigenschaften der Beteiligten (Chandramouli et al., 2021).

Dadurch, dass Richtlinien durch Attribute, ohne die Anzahl von Ressourcen und Benutzer zu kennen, ausgedrückt und Benutzern unabhängig von Ressourcen Attributwerte zugewiesen werden, lassen sich Zugriffsentscheidungen laut Chandramouli et al. (2021) über Benutzeranfragen auf Basis von zentralisierten, unternehmensweiten Richtlinien treffen. Dadurch wird der DevSecOps-Ansatz unterstützt, durch den jedes Microservice-Entwicklungsteam eine starke Autonomie erhält. Entscheidungen, die ihren Microservice betreffen, können die Teams selbst vornehmen, einschließlich der Zuweisung von Attributwerten zu ihren Anwendungsobjekten (Chandramouli et al., 2021).

Chandramouli et al. (2021) kommen aufgrund der beschriebenen Vorteile zu der Schlussfolgerung, dass sich das ABAC-Autorisierungsframework optimal für Cloud-native Anwendungen, deren Design auf Microservices basiert, eignet.

Nach Betrachtung der verschiedenen in der Literatur erwähnten Zugriffskontrollmodelle kann festgehalten werden, dass hier ebenfalls entscheidend ist, wie das zu entwickelnde Anwendungssystem beschaffen ist. Je nach Anforderung und Softwarearchitektur können sich komplexere Modelle wie ABAC oder simplere Modelle wie RBAC besser für die Umsetzung eines Autorisierungskonzeptes eignen.

3.3.3 Access Control Policy Sprachen

Die vorgestellten Zugriffskontrollmodelle lassen sich unter anderem durch Zugriffskontrollrichtliniensprachen umsetzen. In diesem Abschnitt sollen zwei in der Literatur betrachtete Access Control Policy Sprachen untersucht werden.

XACML

Laut Chandramouli et al. (2021) verfügt das im vorherigen Kapitel erläuterte ABAC-Modell über zwei standardisierte Repräsentationsstrukturen, eine davon ist die eXtensible Access Control Markup Language (XACML). Diese soll im Folgenden genauer betrachtet werden. Preuveneers und Joosen (2017) beschreiben, dass XACML der Industriestandard für richtlinienbasierte Zugriffskontrolle ist. Bei XACML handelt es sich um eine domänenspezifische Sprache zur Definition von Berechtigungen auf Ressourcen (Preuveneers & Joosen, 2017). Der Architekturaufbau von XACML in Kombination mit ABAC wird in Chandramouli et al. (2021) erläutert. Abbildung 3.5 stellt den Zusammenhang dieser XACML-Architektur dar, die aus folgenden Komponenten besteht:

- Policy Decision Point (PDP) – trifft Entscheidungen über die Genehmigung oder Ablehnung von Benutzerzugriffsanfragen für die Durchführung von Aktionen auf Ressourcen.
- Policy Enforcement Point (PEP) – fängt alle Zugriffsanfragen ab und entscheidet entweder selbst oder leitet die Anfragen an den PDP weiter.
- Policy Information Point (PIP) – enthält die Attribute und die korrespondierenden Werte für alle Objekte oder Ressourcen einer Anwendung. Liefert die Attribute und Werte für die Benutzer und Ressourcen einer Anfrage zurück, um die anwendbaren Zielrichtlinien im PRP zu finden.
- Policy Retrieval Point (PRP) – enthält die Autorisierungsrichtlinien, die als logische Formeln mit Prädikaten für Attributwerte ausgedrückt werden.
- Attribute Administration Point (AAP) – bietet die Schnittstelle zur Verwaltung der in PIP gespeicherten Attribute an.

- Policy Administration Point (PAP) – bietet die Schnittstelle zur Verwaltung der in PRP gespeicherten Richtlinien.

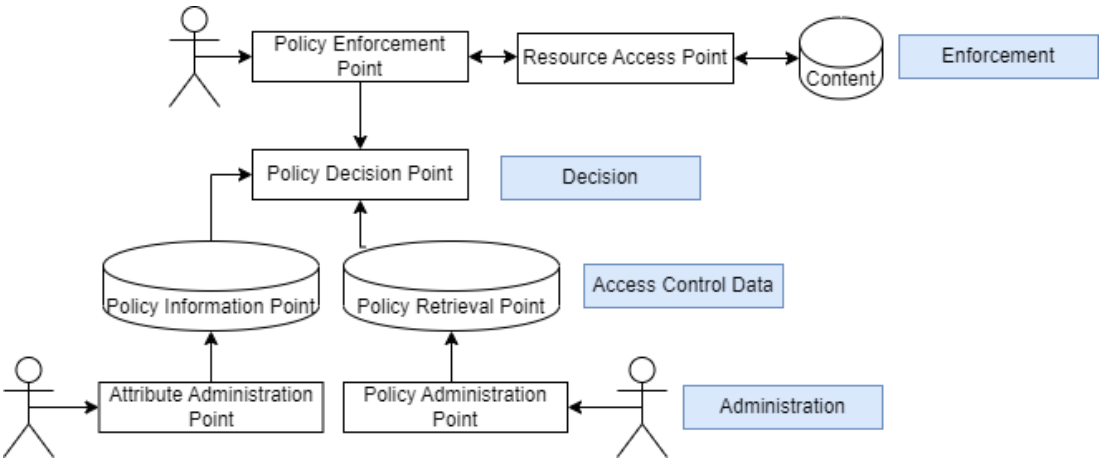

Abbildung 3.5: XACML Architektur
Quelle: In Anlehnung an Chandramouli et al. (2021, p. 11)

Auch wenn XACML nach Preuveneers und Joosen (2019) eine bekannte und bewährte attributbasierte Zugriffskontrollsprache darstellt, so ist diese aufgrund der XML-Struktur vergleichsweise umständlich und ineffizient parsebar. In Preuveneers und Joosen (2017) wird erläutert, dass XACML aufgrund der XML-Struktur komplex zu analysieren und zu ändern ist. Barabanov und Makrushin (2020) treffen sogar die Aussage, dass XACML gescheitert ist. Zum einen, weil dafür eine separate, komplizierte Syntax erlernt werden muss, was für die Softwareentwickler zu mehr Aufwand führt. Zum anderen, weil das Angebot an Open Source Integrationen unzureichend ist. Preuveneers und Joosen (2019) verwenden eine JavaScript Object Notation (JSON) basierte Sprache, die laut ihren Untersuchungsergebnissen einfacher zu verarbeiten und dabei genauso ausdrucksstark ist.

Rego und OPA

Bei der von Preuveneers und Joosen (2019) verwendeten JSON basierten Sprache handelt es sich um Rego, die Zugriffskontrollsprache des Open Policy Agent (OPA). OPA, ist eine Allzweck-Policy-Engine für die feinkörnige Steuerung in Cloud-Native-Umgebungen. Richtlinien werden in der Sprache Rego definiert. Diese bietet die Möglichkeit zur Formulierung deklarativer Regeln für die Verarbeitung von JSON-basierten Dokumenten (Preuveneers & Joosen, 2019).

Nach den Open Policy Agent contributors (n. d.) entkoppelt OPA, ähnlich zu XACML, die Entscheidungsfindung von der Durchsetzung der Richtlinie. So wird OPA analog zum PDP in XACML bei Eingang einer Anfrage abgefragt.

Als Input für die Entscheidung können strukturierte Daten wie JWT geliefert werden. Mit Rego lassen sich Domänen-agnostische Richtlinien beschreiben, wie z. B. welcher Nutzer auf welche Ressource zugreifen oder aber auch welche Subnetze eines Netzwerks miteinander kommunizieren dürfen. Der Ablauf einer Anfrageverarbeitung mit OPA ist in Abbildung 3.6 dargestellt (Open Policy Agent contributors, n. d.).

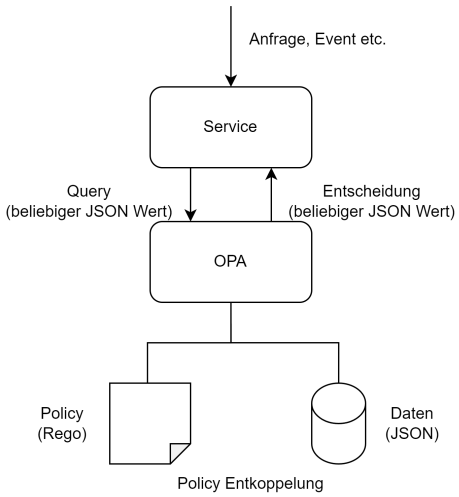

Abbildung 3.6: OPA Anfrageverarbeitung
Quelle: In Anlehnung an Open Policy Agent contributors (n. d.)

Die Nachteile, die bei XACML durch die Verwendung von XML entstehen, sind in OPA nicht gegeben. Dennoch ist zu erwähnen, dass die Sprache Rego, auch wenn sie Domänen-unabhängig und Programmiersprachen ähnlich ist, ebenfalls Einarbeitungsaufwand für das Formulieren von Richtlinien benötigt.

3.3.4 Access-Policy Architekturen

Nachdem nun Zugriffskontrollmodelle und Zugriffskontrollrichtliniensprachen in den vorangegangenen Abschnitten aus theoretischer Perspektive betrachtet wurden, wird im nächsten Abschnitt eine Architektur von Nehme et al. (2019) präsentiert, die Access Policies mithilfe von Token außerhalb des Microservice-Codes umsetzt.

Das Paper von Nehme et al. (2019) kombiniert OAuth2 Token mit der Access Policy Language XACML. Die Implementierung von Nehme et al. (2019) betrachtet ein digitales Regierungsszenario, bei dem es ermöglicht werden soll, einen Reisepass zu beantragen. Ein Anwender nutzt dazu einen bereitgestellten Passdienst. Zunächst erfolgt eine Anmeldung über das Portal, bei der Informationen von verschiedenen Ministerien erforderlich sind. Nutzer müssen der Weitergabe

dieser Informationen explizit zustimmen, wobei für jede dieser Zustimmungen ein separates OAuth2 Token erstellt wird, was explizit für eine Abteilung, also einen bestimmten Informationsbereich erstellt ist (Nehme et al., 2019).

Aus Anwendungssicht wird XACML nach Nehme et al. (2019) für administrative und OAuth2 für benutzerdefinierte Richtlinien verwendet. Die Anwendung enthält einen zentralen Autorisierungsserver, der sowohl als XACML als auch OAuth2 Server dient. Dieser wird als Policy Administration Point und Policy Decision Point verwendet und enthält einen Authentifizierungsserver, der für die Sitzungsverwaltung von ID-Token zuständig ist (Nehme et al., 2019).

In Nehme et al. (2019) werden Microservices außerdem anhand ihrer Funktion nach Consumermicroservices und Ressourcenmicroservices aufgeteilt. Jeder Microservice hat ebenfalls ein eigenes Gateway, wie in Abbildung 3.7 zu sehen ist. Nehme et al. (2019) verwenden jeweils ein Gateway pro Microservice, damit es nur einen zentralen Zugangspunkt zu jedem Microservice gibt, welcher nahezu unabhängig vom Service selbst sein soll. Außerdem ist es dadurch möglich, die Authentizität eingehender Anfragen zu überprüfen und das Gateway als Policy Enforcement Point zu verwenden. Das Gateway prüft dabei das Token und ruft den Policy Decision Point des Autorisierungsservers auf, um eine Richtlinienregelentscheidung durchzuführen. Nehme et al. (2019) beschreiben weiterhin, dass die Zugriffsregeln als eine Reihe von URLs und Aktionen definiert werden können, die einer Benutzergruppe zugeordnet sind. Komplexere Richtlinien können aber auch nach beliebigen Kriterien definiert werden (Nehme et al., 2019).

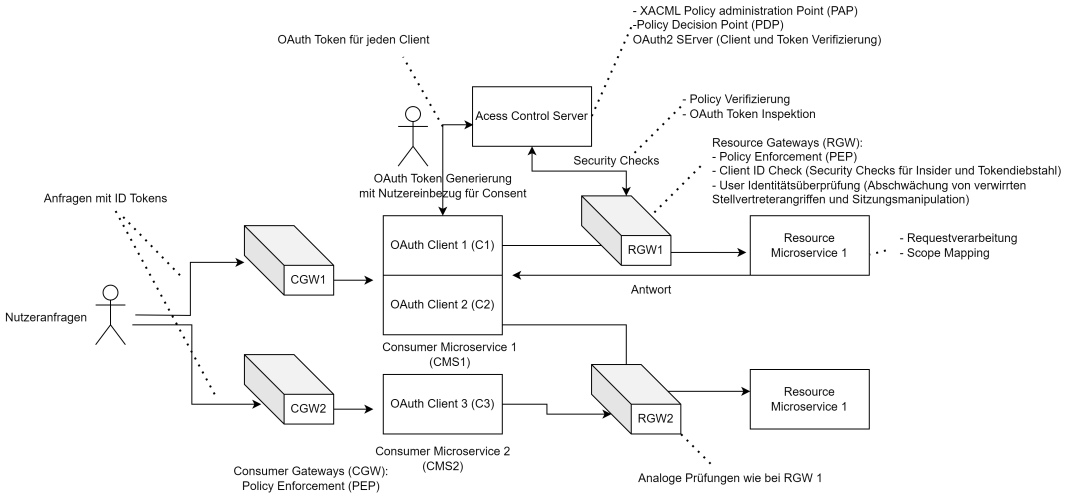

Abbildung 3.7: Überblick der Security Architektur aus Nehme et al. (2019)
Quelle: In Anlehnung an Nehme et al. (2019, p. 290)

Der Ablauf einer Anfrage in der Architektur von Nehme et al. (2019) ist in Abbildung 3.8 zu sehen und stellt sich wie folgt dar. Der Anwender erhält durch den

Log-in beim Autorisierungsserver ein ID-Token. Anschließend schickt die Anwendung eine Anfrage an das Consumer-Gateway (z. B. Reisepassservice) mit diesem ID-Token. Das Consumer-Gateway fragt beim Autorisierungsserver an, ob der Anwender autorisiert ist, den Dienst zu nutzen. Bei erfolgreicher Autorisierung wird die Anfrage an den Consumer-Microservice weitergeleitet. Daraufhin erfolgt eine Anfrage des Consumer-Microservice an den Autorisierungsserver. Dieser schickt einen Zustimmungs-Dialog für die Verwendung der benötigten Ressourcen an den Anwender. Nach Zustimmung des Anwenders erhält der Consumer-Microservice die entsprechenden OAuth-Token, mit denen in Kombination mit dem ID-Token eine Anfrage an die jeweiligen Ressourcendienste gestellt wird. Die Gateways der Ressourcenmicroservices fragen dann zunächst mit den ID-Token den Autorisierungsserver wieder für eine Policy Decision an, in der entschieden wird, ob der Anwender autorisiert ist, auf den Ressourcenmicroservice zuzugreifen. Danach wird, wenn der Nutzer autorisiert ist, das OAuth2-Token im Autorisierungsserver auf Gültigkeit geprüft und die Tokeninformationen anschließend an das Ressourcengateway zurückgeschickt. Wenn alle Bedingungen erfüllt sind, sendet das Ressourcengateway die Anfrage mit der Benutzer-ID und den Zugriffsbereichen an den korrespondierenden Microservice. Dieser Dienst schickt dann die angeforderten Daten an den Consumer-Microservice, der die Anfrage bearbeiten kann und die Antwort an den Anwender zurücksendet (Nehme et al., 2019).

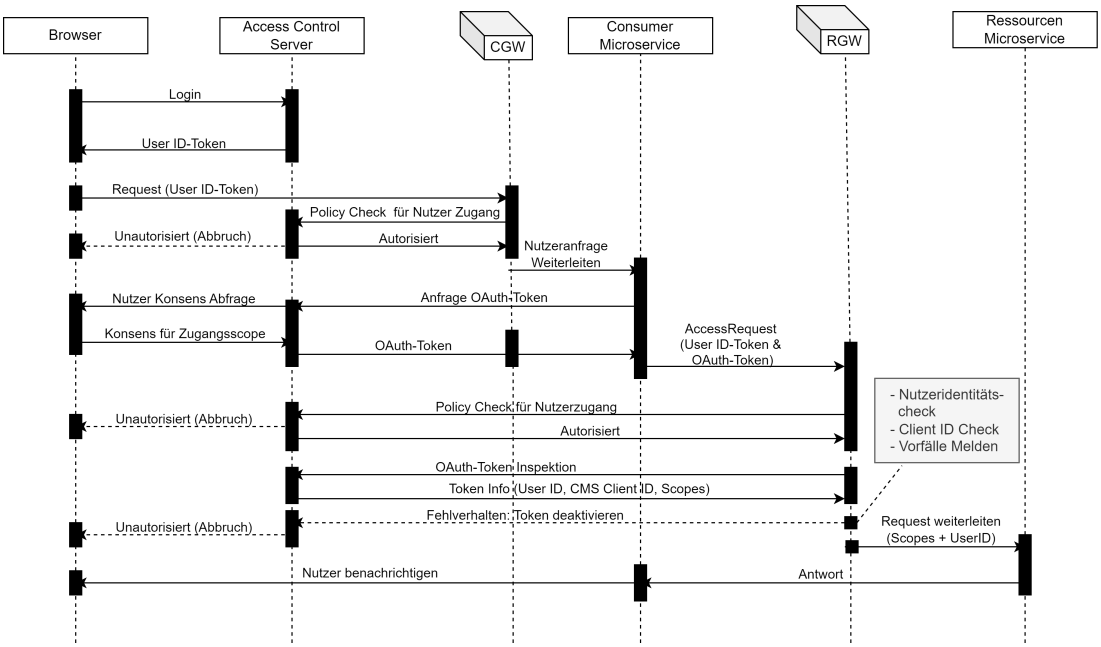

Abbildung 3.8: Sequenzdiagramm einer Service-zu-Service Interaktion nach Nehme et al. (2019)
Quelle: In Anlehnung an Nehme et al. (2019, p. 293)

Das Beispiel zeigt, dass Autorisierungsrichtlinien und die Kombination von verschiedenen Protokollen auf der Basis von Token umsetzbar sind. Im nächsten Kapitel wird nun anhand eines eigenen Anwendungsfalles die Verwendung von Token als Basis für Authentifizierung und Autorisierung in der Praxis gezeigt.

4 Demonstration

Die in diesem Kapitel erfolgende Demonstration dient zur Veranschaulichung der Konzeption und Implementierung einer tokenbasierten Authentifizierungs- und Autorisierungslösung. Damit trägt sie dazu bei, das durch die Literaturanalyse erworbene Wissen in die Praxis zu übertragen. Außerdem ermöglicht sie einen Vergleich der erworbenen Erkenntnisse aus der Theorie mit den Erfahrungen, die durch die Anwendung in der Praxis entstehen.

4.1 Kontext der Demonstration

Zu Beginn dieses Kapitels soll erläutert werden, in welchem Fachkontext sich die Demonstration befindet und welche Herausforderungen sich daraus ergeben. Bei der Anwendung, in der das Authentifizierungs- und Autorisierungskonzept entwickelt wird, handelt es sich um den JValue-Hub[1]. JValue-Hub ist eine kollaborative Data-Engineering-Plattform, welche die Ausführung und Definition von ETL Prozessen auf Open Data Datenquellen ermöglicht.

Der JValue-Hub weist eine Microservice basierte Architektur auf. Ein beispielhafter Überblick über die Interaktionen zwischen den Services ist in Abbildung 4.1 dargestellt. Die Anwendung besteht aus fünf Microservices, die via HTTP miteinander kommunizieren. Die APIs der einzelnen Microservices sind nach dem REST Paradigma konzipiert.

[1] https://github.com/jvalue/hub

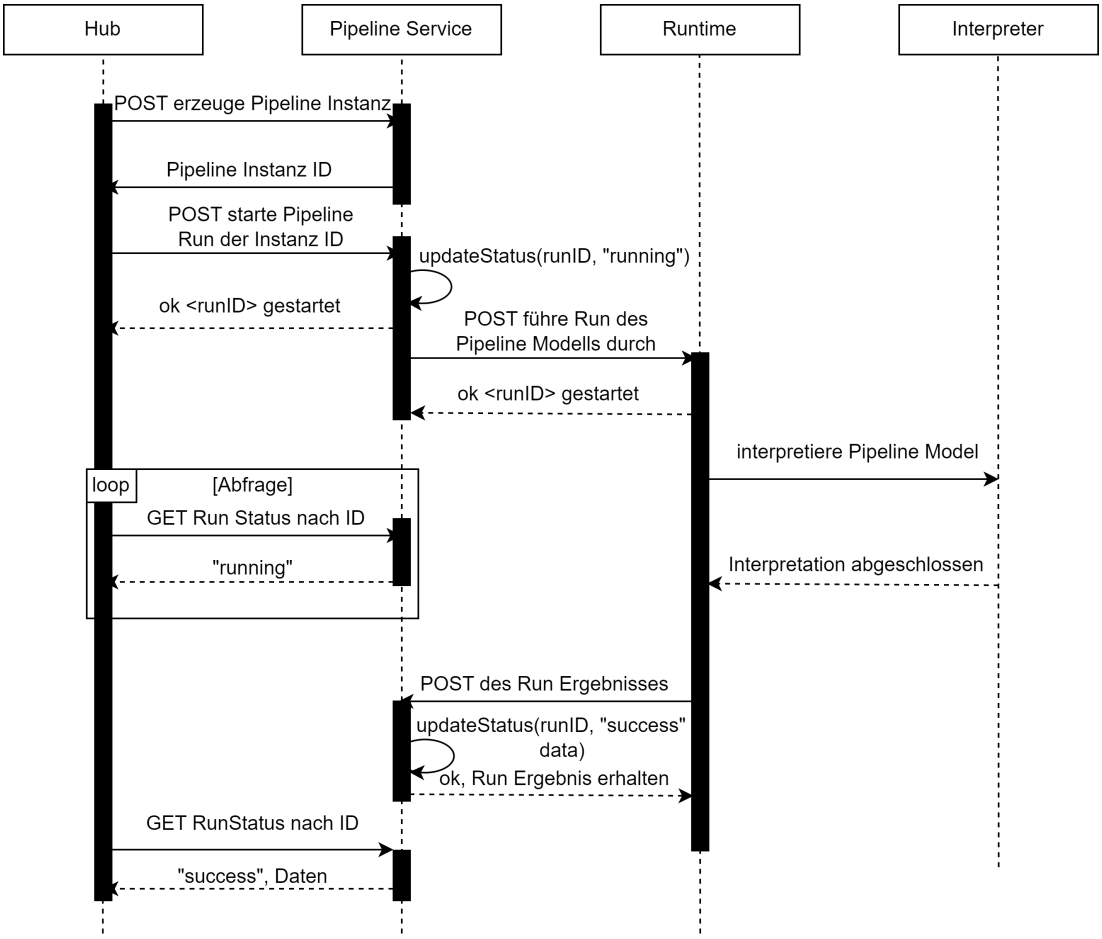

Abbildung 4.1: Überblick JValue-Hub-Service Interaktion
Quelle: In Anlehnung an JValue Core Developer (n. d.)

Die Anwendung befindet sich derzeit noch in einer früheren Entwicklungsphase, wobei die Authentifizierung und Autorisierung bisher nicht im Fokus standen. Deshalb soll für den JValue-Hub ein Authentifizierungs- und Autorisierungskonzept auf der Basis von Token entworfen und prototypisch umgesetzt werden. Das Konzept basiert auf den in Kapitel 3 beantworteten Forschungsfragen und zeigt, wie ein Authentifizierungs- und Autorisierungskonzept auf der Basis von Token aussehen kann.

Der derzeitige Ablauf der Authentifizierung und Autorisierung basiert auf einer Kombination aus Nutzerdatenbank und JWT. Um sich einzuloggen, gibt ein Anwender Nutzername und Passwort ein. Diese Kombination wird mittels eines POST-Requests über HTTPS an die 'auth/login'-Schnittstelle des Hub-Backend-Services weitergeleitet. Im Hub-Backend-Service wird zunächst geprüft, ob ein Nutzer mit dem angegebenen Namen in der Nutzerdatenbank vorhanden ist und das eingegebene Passwort mit dem in der Datenbank gespeicherten Passwort

übereinstimmt. Der Abgleich der Passwörter erfolgt durch die bcrypt[2]- Bibliothek von Node.js[3]. Ist der Vergleich erfolgreich, wird ein JWT mit dem Nutzernamen, der ID des Anwenders und dem Ablaufdatum als Payload durch den JWT-Service[4] der NestJS JWT-Bibliothek signiert und an den Anwender gesendet. Das Token wird im lokalen Speicher des Browsers und zusätzlich im Redux [5]-Speicher der Anwendung abgelegt.

Zum initialen Speichern der Passwörter beim Anlegen eines Nutzers wird die Hashfunktion von bcrypt verwendet. Dabei wird ein Salt und das Passwort in die Hashfunktion gegeben und der resultierende Hash als Passwort in der Datenbank abgespeichert. Dieser kann dann später mit der Vergleichsfunktion von bcrypt wieder gegen ein Klartextpasswort abgeglichen werden.

Die Autorisierung erfolgt derzeit anhand der ID eines Nutzers. Am Beispiel eines Projekts innerhalb der Anwendung erfolgt diese wie folgt. Das Frontend erhält über den 'findAllByUser'-Endpunkt des Hub-Backends alle Projekte, bei denen ein Anwender entweder Mitarbeiter oder Besitzer des Projekts ist. Diese Berechtigungen sind Einträge in den Spalten 'collaborators' und 'user' der Tabelle 'project'. Über Join-Operationen auf der Tabelle werden die Projekte, auf denen der Nutzer Anzeige-Berechtigungen hat, ermittelt und diesem zurückgeliefert. Ob der Anwender beispielsweise die Beschreibung eines Projekts verändern darf, wird anhand seiner ID im Hub-Web geprüft. Dieses Konzept bildet derzeit nur die Beziehung zwischen Projekten und einzelnen Nutzern ab und hat keine Möglichkeit zur Verwaltung von übergeordneten Strukturen wie Gruppen oder ähnlichem.

4.2 Anforderungsanalyse

Um die Anforderungen an ein zukünftiges Autorisierungs- und Authentifizierungskonzept zu definieren, wird ein Workshop mit den Projektmigliedern des JValue-Hub durchgeführt. Im Workshop werden verschiedene Fragen zum aktuellen Stand der Authentifizierung und Autorisierung und zu einem künftigen Zielbild gestellt. Daraus ergeben sich folgende Anforderungen:

R1: Unterscheidung verschiedener Nutzertypen, z. B. zwischen zahlenden und nicht zahlenden Nutzern

R2: Abbildung verschiedener Projektrollen für Nutzer, z. B. Owner, Contributor, Guest

R3: Verwaltung von Teams und Organisationen

[2]https://www.npmjs.com/package/bcrypt
[3]https://nodejs.org/en
[4]https://docs.nestjs.com/security/authentication
[5]https://redux.js.org/

R4: Authentifizierte und autorisierte Maschine-zu-Maschine-Kommunikation

R5: Zugriff für Maschinen außerhalb des eigenen Netzwerks auf netzwerkinterne Ressourcen, z. B. Zugriff im Rahmen von Batch Jobs auf Datenergebnisse der Pipeline Läufe

R6: Verwaltung von Passwörtern außerhalb der eigenen Datenhaltung

R7: Zwei-Faktor-Authentifizierung im Rahmen der Anmeldung

R8: Anmeldung über Drittanbieter wie z. B. Google

R9: Single Sign-On

R10: Refresh nach Ablauf der Tokengültigkeit

R11: Sicheres Standardverfahren zur Anmeldung

R12: Berücksichtigung möglicher zukünftiger mobiler Szenarien

Zur Erfüllung dieser Anforderungen wird in der nächsten Sektion ein Konzept für eine zukünftige Authentifizierungs- und Autorisierungslösung erstellt.

4.3 Design einer Authentifizierungs- und Autorisierungslösung für den JValue-Hub

In dieser Sektion wird der konzeptionelle Entwurf einer Authentifizierungs- und Autorisierungslösung für den JValue-Hub vorgestellt. Zunächst wird dazu die Durchführung einer Open Source IAM-Analyse beschrieben, auf deren Basis das Konzept zur Authentifizierung und Autorisierung für den JValue-Hub entwickelt wird.

4.3.1 Open Source IAM Analyse

Aufgrund zusätzlicher Anforderungen, wie das Verwalten von Passwörtern außerhalb der eigenen Datenhaltung, der Unterstützung einer Zwei-Faktor-Authentifizierung, der Möglichkeit der Anmeldung über Drittanbieter sowie der Verwendung eines sicheren Standardverfahrens zur Benutzeranmeldung, wird in Absprache mit den Entwicklern des JValue-Hub beschlossen, ein Open-Source Identity und Access Management System einzusetzen. Ein solches System bietet bereits viele dieser Funktionalitäten, ohne zusätzlichen eigenen Implementierungsaufwand zu erfordern.

Um eine möglichst geeignete Lösung auszuwählen, findet zunächst die Entwicklung einer Taxonomie, angelehnt an Nickerson et al. (2013), statt. In der Taxonomie

sollen verschiedene IAM-Lösungen dargestellt und miteinander vergleichbar gemacht werden. Das Hauptaugenmerk liegt somit auf den Features, welche die Softwaresysteme anbieten. Im Anhang unter B wird die Entwicklung der Taxonomie anhand der in Nickerson et al. (2013) präsentierten Schritte beschrieben. Auf Basis der entstandenen Taxonomie und den Ausprägungen der einzelnen Tools erfolgt die Auswahl einer geeigneten Lösung. Tabelle 4.1 zeigt die entstandene Klassifikation der angebotenen Features.

Die Anforderungen aus Sektion 4.2 können mit jeder der analysierten Lösungen umgesetzt werden. R1, R2 und R3 lassen sich durch Access Control Mechanismen wie ABAC oder RBAC realisieren, indem ein Autorisierungskonzept auf Basis eines dieser Mechanismen definiert wird. R4 und R5 lassen sich durch den Einsatz von JWT Bearer mit privaten Schlüsseln nach RFC 7523[6] umsetzen, indem die Token durch das Signieren mit einem privaten Schlüssel in einem nicht interaktiven Authentifizierungsprozess gegen ein Access-Token getauscht werden. R7,R8 und R9 sind in den angebotenen Log-in-Features der IAM-Lösungen explizit aufgeführt und damit durch diese realisierbar. R6 ergibt sich automatisch durch das Verwenden einer IAM-Lösung, da diese die Speicherung der Passwörter übernimmt und dafür keine eigene Logik im Programmcode benötigt wird. Durch die Verwendung von OpenID Connect können R10, R11 und R12 ebenfalls aufwandsarm umgesetzt werden, da nach OneLogin (n. d. a) der Authorization-Code-Flow mit PKCE ein sicheres Standardverfahren für die Authentifizierung von Benutzern mobiler oder nativer Anwendungen ist.

Da, wie bereits erläutert, die Umsetzung der Anforderungen des Projektteams grundsätzlich mit allen IAM-Lösungen möglich ist, wird die Auswahl aufgrund von ersten praktischen Implementierungstests getroffen. Hierbei hat sich Zitadel als besonders intuitiv, schnell einsetzbar und benutzerfreundlich erwiesen und wird deshalb für eine prototypische Implementierung ausgewählt.

[6] https://datatracker.ietf.org/doc/html/rfc7523

Kategorie	Feature	Keycloak	OpenAM/ Forgerock	Casdoor	Zitadel	Ory
Log-In-Features	SSO	+	+	+	+	+
	CDSSO	-	+	-	-	-
	MFA	+	+	+	+	+
	Social Log-in	+	+	+	+	+
	OTP	+	+	+	+	+
	WebAuthn	+	+	+	+	+
	CTAP	-	+	-	-	+
	Client Authentication via Signed JWT RFC 7523	+	+	-	+	+
Protokolle / Standards	OpenID Connect	+	+	+	+	+
	OAuth2.0	+	+	+	+	+
	SAML	+	+	+	+	+
	FIDO2	+	+	+	+	+
Usability und Customizability	Admin Console	+	+	+	+	+
	Account Management Self-Service for Users	+	+	-	+	+
	Customize/ Bring Your Own UI	+	+	+	+	+
Access Control Mechanismen	ABAC	+	+	+	+	+
	RBAC	+	+	+	+	+
	Timebased Access control	+	-			
	Support for custom access control mechanisms	+	-	+	+	+
	Custom Permission Language	-	-	+	-	+
Deployment	Docker	+	+	+	+	+
	Kubernetes	+	+	+	+	+
	Helm-Charts	-	+	-	+	+

Tabelle 4.1: IAM Features nach Iteration 2

4.3.2 Authentifizierung

Nach der Auswahl der zu verwendenden IAM-Lösung wird in dieser Sektion eine tokenbasierte Authentifizierungslösung für den JValue-Hub beschrieben. Das Authentifizierungskonzept sowie das in der nächsten Sektion folgende Autorisierungskonzept basieren auf der ausgewählten IAM-Lösung Zitadel und nutzen deren Features als Grundlage.

Authentifizierungsprozess für Endnutzer

Für die Endnutzer-Authentifizierung ist im erarbeiteten Konzept die Verwendung des 'Authorization Code Flow with Proof Key for Code Exchange'[7] von OAuth 2.0 vorgesehen.

Laut Kölpin (2022) eignet sich dieser insbesondere für Mobile- und Webapplikationen, weil diese Anwendungen aufgrund der möglichen Angriffsvektoren nicht die Möglichkeit haben, Client Secrets sicher zu speichern. Da es sich bei dem JValue-Hub-Web um eine Single-Page-Application (SPA) und damit um eine

[7] https://datatracker.ietf.org/doc/html/rfc7636

Webapplikation handelt und aus R12 hervorgeht, dass zukünftig ebenfalls native Anwendungen auf mobilen Geräten zum Einsatz kommen sollen, eignet sich diese Art der Authentifizierung optimal für den aktuellen Stand der Anwendung sowie kommende mobile Szenarien.

Der Ablauf der Authentifizierung ist in Abbildung 4.2 in Anlehnung an Okta.Inc (n. d.) zu sehen. Der Benutzer klickt dabei zunächst innerhalb des JValue-Hub-Web auf Log-in und leitet damit den Authentifizierungsprozess ein. Anschließend wird daraufhin in der Frontendapplikation ein kryptografisch zufälliger Code-Verifier generiert und aus diesem eine Code-Challenge erstellt. Als Nächstes wird eine Anfrage an den Autorisierungsserver von Zitadel gemeinsam mit der Code-Challenge gesendet. Der Autorisierungsserver leitet den Nutzer weiter auf eine Log-in-Seite, auf der dieser sich mit Passwort und Benutzername so wie einem zweiten Faktor authentifiziert. Der Autorisierungsserver speichert die Code-Challenge und sendet nach erfolgreicher Authentifizierung einen Autorisierungscode an den JValue-Hub-Web. Dieser Autorisierungscode kann einmalig verwendet werden. Die Anwendung sendet diesen Code inklusive Code-Verifier zurück an den Autorisierungsendpunkt von Zitadel und erhält nach der Verifizierung der Code-Challenge mithilfe des Code-Verifiers durch den Autorisierungsserver ein Access- und ein ID-Token (Okta.Inc, n. d.). Damit ist die Authentifizierung eines Endnutzers erfolgreich abgeschlossen.

Das erhaltene ID-Token beinhaltet dabei Informationen über den Benutzer und das Access-Token wird wiederum zur Autorisierung verwendet (Chiarelli, 2021). Näheres zur Autorisierung findet sich in Abschnitt 4.3.3. Bei der Authentifizierung wird sich bewusst gegen den Einsatz von Refresh-Token entschieden, welche zur Erneuerung des Zugangs nach Ablauf eines Access-Tokens verwendet werden. Diese Entscheidung basiert auf der im Vorangegangenen beschriebenen Eigenschaft von Mobile- und Webapplikationen, die keine Möglichkeit zur sicheren Speicherung von Client Secrets bieten (Kölpin, 2022). In Zitadel ermöglicht ein Refresh-Token in Kombination mit der Zitadel-spezifischen Client-Id einer Anwendung das erneute Anfordern eines Access-Tokens inklusive neuem Refresh-Token (Max Peintner, 2023c). Dies hat zur Folge, dass der Diebstahl eines Refresh-Tokens und die Kenntnis über die Client-Id die Möglichkeit zur wiederholten Authentifizierung eines Angreifers bieten.

Neben der Authentifizierung von Endnutzern ist in diesem Konzept auch eine Betrachtung der Authentifizierung für die Maschine-zu-Maschine-Kommunikation notwendig, um die den Anforderungen R4 und R5 zu erfüllen. Diese wird im nächsten Abschnitt betrachtet.

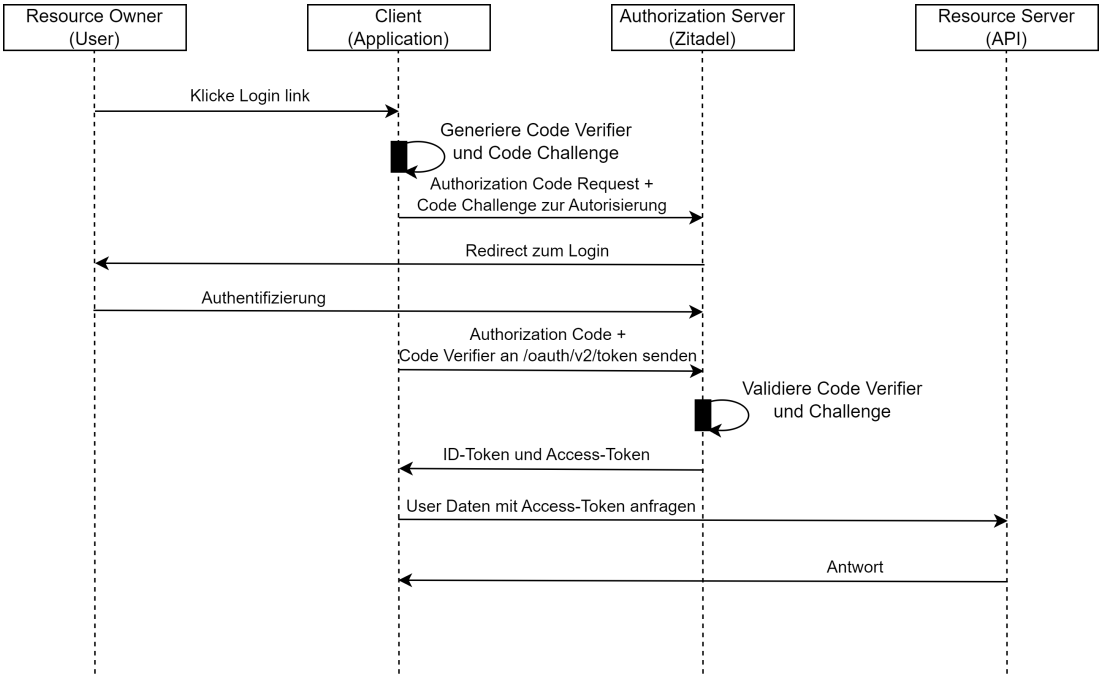

Abbildung 4.2: Authorization Code Flow mit PKCE
Quelle: In Anlehnung an (Okta.Inc, n. d.)

Authentifizierungsprozess für Maschine-zu-Maschine-Kommunikation

Um eine Authentifizierung von Maschine-zu-Maschine zu ermöglichen, ist es notwendig, eine Methode bereitzustellen, die ohne menschliche Interaktion auskommt. Für die Authentifizierung zwischen Maschinen ist in diesem Konzept die Verwendung von JWT Bearer mit privaten Schlüsseln nach RFC 7523 vorgesehen.

Für die Umsetzung dieses Prozesses muss zunächst ein Schlüsselpaar aus privatem und öffentlichem Schlüssel in Zitadel erzeugt werden. Mithilfe des privaten Schlüssels wird ein JWT erstellt, welches folgende Informationen beinhaltet:

- Header:
  - Art des Algorithmus (RS256)
  - ID des Schlüsselpaares
- Payload:
  - Besitzer (Nutzer-ID)
  - Subjekt für den das JWT ausgestellt wird (Nutzer-ID)
  - Publikum für welches das JWT bestimmt ist
  - Erstellzeitpunkt

– Ablaufzeitpunkt

Das JWT wird anschließend an den Autorisierungsserver von Zitadel gesendet, welcher das Token anhand des in Zitadel gespeicherten zugehörigen öffentlichen Schlüssels auf Validität prüft und bei Erfolg ein Access-Token analog zum Authentifizierungsprozess für Endnutzer zurückliefert (Max Peintner, 2023e). Damit bietet die Verwendung von JWT Bearer mit privaten Schlüsseln nach RFC 7523 eine Möglichkeit, Services untereinander zu authentifizieren, ohne die Notwendigkeit der Interaktion eines Endnutzers. Der Authentifizierungsprozess für die Maschine-zu-Maschine-Kommunikation kann nach einmaliger Implementierung in einem Drittsystem immer wieder erfolgen.

4.3.3 Autorisierung

Nach der Erläuterung der Authentifizierungsvorgänge für Endnutzer sowie der Maschine-zu-Maschine-Kommunikation, wird nun im Nachfolgenden das Autorisierungskonzept für den JValue-Hub betrachtet.

Autorisierungsprozess

Wie im Vorangegangenen beschrieben erhält ein Nutzer des JValue-Hub nach einer erfolgreichen Authentifizierung ein Access-Token, welches beim Zugriff auf eine Ressource zur Autorisierung verwendet wird. In Abbildung 4.3 ist dargestellt, wie der Autorisierungsprozess abläuft. Der Ressourcenserver prüft das Access-Token zunächst auf Validität. Dazu sendet er das Token an den Introspektionsendpunkt von Zitadel. Dieser liefert zurück, ob das Token noch aktiv ist oder nicht. Bei aktivem Token werden die Rollen in Form der Zitadel spezifischen Metadaten zurückgeliefert. Die Beschreibung der Rollen sowie die Berechtigungen, die sich daraus ableiten, werden in der nächsten Sektion beschrieben. Die Werte der Metadaten sind nach base64 kodiert und müssen deshalb zunächst dekodiert werden. Anschließend können ressourcenspezifische Rechteprüfungen anhand der Rollen erfolgen. So kann beispielsweise sichergestellt werden, dass das Löschen eines Projekts nur durch einen Projektbesitzer erfolgen kann. Alle für den Umgang mit den Token und für die Autorisierung spezifischen Prüfungen sollten in einer Bibliothek erfolgen, um Codedopplungen an den Endpunkten der Ressourcenserver zu minimieren. Das Rollenkonzept und die dadurch entstehenden Berechtigungen werden im Nachfolgenden noch genauer erläutert.

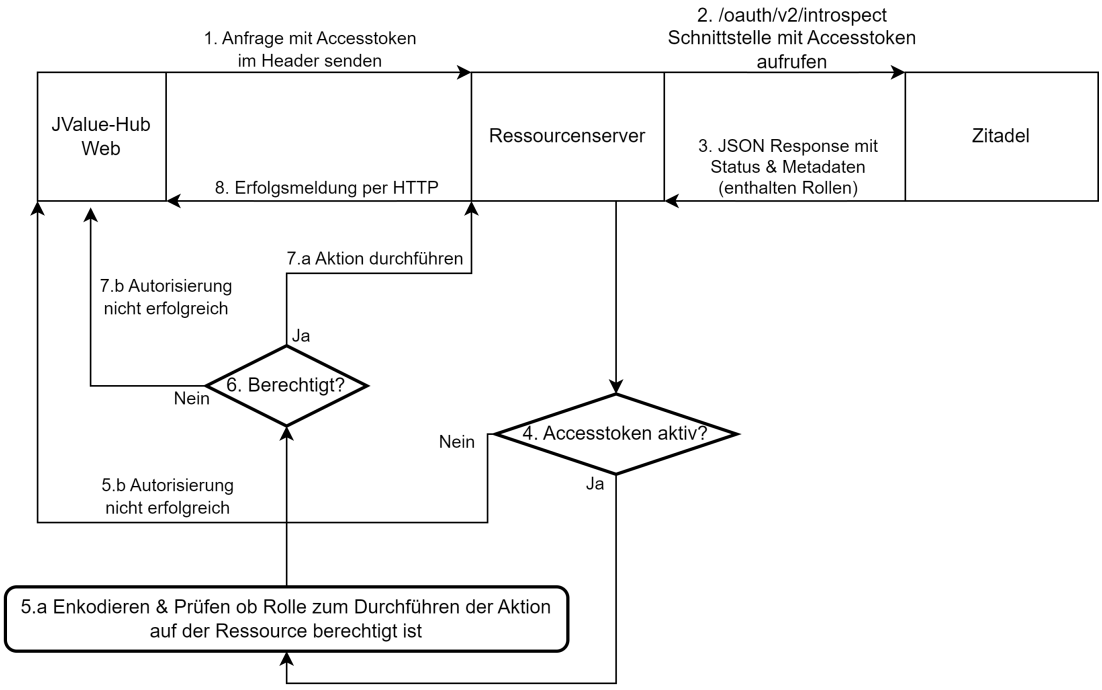

Abbildung 4.3: Autorisierungsprozess im JValue-Hub

Neben dem Konzept für Endanwender ist auch bei der Autorisierung eine der Anforderungen aus der Sektion 4.2 eine Lösung für die Autorisierung von Maschine-zu-Maschine-Kommunikation bereitzustellen. Nach der Authentifizierung eines externen Services erhält dieser, wie im Authentifizierungsprozess beschrieben, ebenfalls ein Access-Token. Dieses beinhaltet dann analog zum Authentifizierungsprozess des Endnutzers die Berechtigungen. Damit kann die Autorisierung eines Service-Nutzers entsprechend dem im Vorangegangenen dargestellten Prozess erfolgen.

Nach der Beschreibung des Autorisierungsprozesses auf Basis von Access-Token wird nun ein Rollenkonzept für die Realisierung der Anforderungen R1, R2 und R3 beschrieben.

Organisationen, Teams, Projekte, Nutzertypen und Nutzerrollen

Das Konzept zur Autorisierung von Nutzern basiert auf der Verwendung von Organisationen, Teams, Projekten sowie Nutzertypen und Rollen. Es orientiert sich dabei an bekannten Sourcecodeverwaltungslösungen wie Gitlab[8] oder GitHub[9]. Dabei soll ein Nutzer in unterschiedlichen Ebenen verwaltet werden, verschiedene

[8]https://docs.gitlab.com/ee/user/permissions.html

[9]https://docs.github.com/de/get-started/learning-about-github/access-permissions-on-github

Rollen in Organisationen, Teams und Projekten einnehmen und unterschiedliche Features durch Autorisierung nutzen können. Abbildung 4.4 zeigt die Beziehung zwischen den unterschiedlichen Verwaltungsebenen, den Nutzern, Projekten und Projektressourcen.

Da keine weiteren Eigenschaften, wie Ort oder Zeit, in der Autorisierungsentscheidung berücksichtigt werden müssen, werden die Anforderungen durch ein RBAC-Modell umgesetzt. RBAC bietet die Vorteile einer einfacheren Autorisierung, einfacher Umsetzbarkeit und leichter Verständlichkeit, jedoch wird dafür ein Trade-Off bezüglich der Flexibilität eingegangen (OneLogin, n. d. b).

Für die in Abbildung 4.4 dargestellten Entitäten Organisation (Org), Team, Projekt und Nutzer werden Rollen erstellt. Die nachfolgenden Tabellen 4.2, 4.3 und 4.4 zeigen die Aktionen, die je nach Rolle für die jeweilige Ressource erlaubt sind.

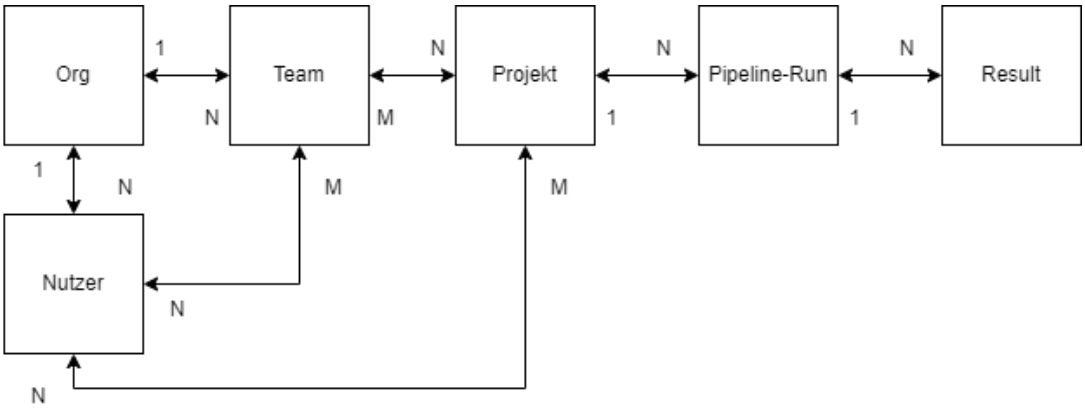

Abbildung 4.4: Ressourcenbeziehungen im JValue-Hub

Die Organisation befindet sich dabei auf oberster Ebene. Sie kann Nutzer und Teams enthalten. Ein mögliches Praxisbeispiel für eine Organisation ist ein Unternehmen oder eine Universität, die den JValue-Hub verwenden und in der Organisation die Verwaltung ihrer Nutzer vornehmen möchte.

Org Aktion	Org Owner	Org Admin	Org Member
Rollen innerhalb der Org verändern	+	+	-
Nutzer als Org Admin einladen	+	+	-
Nutzer als Org Member einladen	+	+	-
Org löschen	+	-	-
Teams in der Org erstellen	+	+	-
Teams in der Org löschen	+	+	-
Andere Nutzer der Org sehen	+	+	+

Tabelle 4.2: Rollen innerhalb einer Organisation

Teams sind immer maximal einer Organisation zugeordnet und stellen eine kleinere Strukturierungseinheit für diese dar. Hier können einzelne Nutzer, die gemeinsam an einem oder mehreren Projekten arbeiten, verwaltet werden. Ein mögliches Beispiel hierfür ist eine Gruppe von Mitarbeitern eines Unternehmens, welche sich mit einer speziellen Aufgabe befasst.

Team Aktion	Team Owner	Team Admin	Team Member	Team Guest
Rollen innerhalb des Teams verändern	+	+	-	-
Nutzer als Team Admin einladen	+	+	-	-
Nutzer als Team Member einladen	+	+	-	-
Nutzer als Team Guest einladen	+	+	-	-
Team löschen	+	-	-	-
Projekte im Team erstellen	+	+	-	-
Projekte im Team löschen	+	+	-	-
Projekte im Team bearbeiten	+	+	+	-
Projekte im Team sehen	+	+	+	+
Andere Nutzer des Teams sehen	+	+	+	+

Tabelle 4.3: Rollen innerhalb eines Teams

Projekte im JValue-Hub-Kontext sind die zentrale Stelle, an der Modelle für die Verarbeitung von Open Source Daten definiert und ausgeführt werden können. Auf Projekten können ein oder mehrere Nutzer bzw. Teams in unterschiedlicher Art und Weise berechtigt werden. Innerhalb der Projekte können Modelle definiert, Pipelines zu Modellen gestartet und Ergebnisse der Pipelines abgerufen werden.

Projekt Aktion	Projekt Owner	Projekt Admin	Projekt Member	Projekt Guest
Rollen innerhalb des Projekts verändern	+	+	-	-
Nutzer als Projekt Admin einladen	+	+	-	-
Nutzer als Projekt Member einladen	+	+	-	-
Nutzer als Projekt Guest einladen	+	+	-	-
Projekt löschen	+	-	-	-
Models im Projekt bearbeiten	+	+	+	-
Pipelines im Projekt laufen lassen	+	+	+	-
Ergebnisse der Pipelines im Projekt abrufen	+	+	+	-
Description des Projekts bearbeiten	+	+	+	-
Andere Nutzer des Projekts sehen	+	+	+	+

Tabelle 4.4: Rollen innerhalb eines Projekts

Umsetzung des Rollenkonzepts im JValue-Hub

Um das Rollenkonzept im JValue-Hub realisieren zu können, muss es in der Anwendung die Möglichkeit geben, die Rollen für Anwender festzulegen. Der Ersteller einer Organisation, eines Teams oder eines Projekts wird immer implizit zum Owner der jeweiligen Organisationseinheit und kann dadurch die Berechtigungen verwalten. Das Konzept sieht vor, dass Nutzer ihre Berechtigungen selbstständig über einen Administrationsendpunkt im JValue-Hub verwalten können. Technisch sollen die Rollen dabei im Hintergrund über die Set-User-Metadaten-Schnittstelle von Zitadel vergeben werden. Für die Realisierung des Administrationsendpunkts wird ein Zitadel-Serviceuser im JValue-Hub Web vorgesehen, der die Zitadel spezifische Rolle 'Org User Manager' innehaben muss, um die entsprechenden Metadaten bei Nutzern setzen zu können (Max Peintner, 2023b).

Die Abbildung der Rollen erfolgt in den Metadaten des Access-Tokens eines Nutzers, welches zu den verschiedenen Microservices des JValue-Hub transportiert wird, um eine Autorisierung auf Basis der Rollen zu ermöglichen. Wie bereits im Autorisierungsprozess beschrieben, erfolgt bei der Autorisierungsprüfung eines Anwenders ein Aufruf des Introspektionsendpunktes. Dabei werden die aktuellen Autorisierungsinformationen zu einem Nutzer zurückgeliefert. Denn obwohl das Access-Token bereits Autorisierungsinformationen enthält, ist es möglich, dass sich die darin enthaltenen Informationen während der Gültigkeitsdauer des Tokens ändern. Dies geschieht zum Beispiel durch das Erstellen eines neuen Projektes, auf das ein Nutzer berechtigt wird. Das vorher bereits ausgestellte Token enthält diese neuen Autorisierungsinformationen nicht. Dies hat zur Folge, dass eine Anfrage entweder scheitert oder das Token als ungültig gekennzeichnet wird, was einen erneuten Log-in erforderlich macht. Durch die Verwendung der Introspektionsschnittstelle wird also zusätzlich die Benutzerfreundlichkeit gesteigert.

Der Aufbau der Autorisierungsinformationen in den Metadaten des Access-Tokens eines Nutzers ist wie folgt: Ein Nutzer hat verschiedene Rollen in seinen Metadaten. Diesen Rollen ist jeweils ein Array aus Entitäten-IDs zugeordnet, für welche die entsprechende Rolle erfüllt ist. Das ermöglicht eine Berechtigungsprüfung für angefragte Ressourcen auf Basis der zugewiesenen Rollen zu einer Ressource. Ein Beispiel für ein Access-Token ist in Abbildung 4.5 zu sehen.

```
{
  "iss" : "http://localhost:8080",
  "sub" : "212133599430836227",
  "aud" : [
      "208814041051758595@hub",
      "210813535058460675@hub",
      "208814041051758595"
  ],
  "jti" : "221144802803056642",
  "exp" : 1688351774,
  "iat" : 1688308574,
  "nbf" : 1688308574,
  "urn:zitadel:iam:user:metadata" : {
    "OrgMember" :
        [
          "79852e4a-372b-4d9a-a359-564b46f38812",
          "59200009-c603-47ba-8eda-ba1952f4e5c6"
        ],
    "ProjektOwner" :
        [
          "052d6b90-8e8a-4b4a-af43-63548dc5d523"
        ]
    }
}
```


Abbildung 4.5: Beispiel Access-Token im JWT Format mit base64 dekodierten Werten

Nach der konzeptionellen Darstellung der Authentifizierungs- und Autorisierungslösung für den JValue-Hub wird in der nächsten Sektion die erfolgte prototypische Umsetzung dargestellt.

4.4 Prototypische Implementierung der Lösung

Um zu überprüfen, ob das zuvor beschriebene Konzept umgesetzt werden kann, ist im Rahmen dieser Arbeit eine prototypische Implementierung im JValue-Hub mithilfe von Zitadel erfolgt.

4.4.1 Authentifizierungsprozess

Für den Authentifizierungsprozess eines Nutzers bietet Zitadel verschiedene Möglichkeiten. Die Auswahl eines geeigneten Authentifizierungsprozesses wurde im Rahmen der Konzepterstellung durchgeführt. Die Konfiguration der Authentifizierung für Anwendungen erfolgt in Zitadel über eine Weboberfläche. Da es sich bei dem Frontend des JValue-Hub um eine SPA handelt, wird für die Authentifizierung die Option User Agent ausgewählt, welche nach Max Peintner (2023g) für diesen Anwendungszweck vorgesehen ist.

Bei dieser Auswahl wird implizit OIDC als Authentifizierungsprotokoll ausgewählt. Im anschließenden Konfigurationsschritt besteht die Auswahlmöglichkeit zwischen dem Code-Flow mit PKCE und dem Implicit Flow von OIDC. Aufgrund der erhöhten Sicherheit des PKCE Flows, die Christian Lüdemann (2019) ebenfalls bestätigt, wird dieser in Zitadel konfiguriert. Dafür muss zum Abschluss des Authentifizierungsprozesses eine Redirect-URI angegeben werden, an welche nach einer erfolgreichen Authentifizierung der Autorisierungscode mit dem PKCE-Code zurückgegeben wird.

Die konkrete Konfiguration erfolgt in dieser Arbeit auf Basis einer lokalen Instanz von Zitadel[10], welche zunächst mittels Docker Compose[11] gestartet wird. Bevor die Authentifizierung einer Anwendung konfiguriert werden kann, müssen zunächst eine JValue-Organisation und ein Hub-Projekt angelegt werden. In Zitadel sind Organisationen die Entität, innerhalb derer verschiedene Projekte und Nutzer, die im selben Anwendungskontext laufen und einen gemeinsamen Authentifizierungs- und Autorisierungsprozess benötigen, enthalten sind (Max Peintner, 2023d). Projekte stellen eine Untergliederung einer Organisation dar und beinhalten eng miteinander verbundene Komponenten wie Nutzer, Rollen und die dazugehörigen Anwendungen (Max Peintner, 2023f). Im Hub-Projekt wird eine Anwendung Hub-Web angelegt, die in Zitadel der Einstiegspunkt für die Authentifizierung eines Benutzers ist (Max Peintner, 2023h). Für die Anwendung Hub-Web werden, wie bereits beschrieben, der OIDC-Code-Flow mit PKCE konfiguriert und die entsprechenden Redirect und Log-out Uris für den JValue-Hub-Web festgelegt. Weitere Einstellungen, welche für die Token getroffen werden, umfassen das Inkludieren der Metadaten in den Access- und ID-Token. Außerdem wird ein Service-User zur Verifizierung der oben dargestellten Service-User-Authentifizierung und -Autorisierung erstellt sowie eine Individualisierung des User-Interfaces der Log-in Seite vorgenommen. Diese ist in Abbildung 4.6 zu sehen.

[10] https://zitadel.com/docs/self-hosting/deploy/compose
[11] https://docs.docker.com/compose/

Abbildung 4.6: Log-in-Maske des JValue-Hub-Web

Anschließend wird Zitadel in das Projekt JValue-Hub-Web integriert. Dazu wird die bisherige Log-in-Page der Anwendung umgestaltet und die Maske, in der bisher ein Nutzername und ein Passwort eingegeben werden konnten, durch einen Log-in Button ersetzt. Dieser leitet auf die, in der Abbildung 4.6, gezeigte Log-in-Maske weiter. Nach dem Log-in, welcher mithilfe des User-Manager-Objekts der 'oidc-client-ts'[12] Bibliothek erfolgt, erhält das Frontend das Access- und das ID-Token. Anschließend wird der 'Userinfo'-Endpunkt von Zitadel aufgerufen, um Informationen wie Nutzername und ID im Redux-Speicher der Anwendung zur Verfügung zu stellen. Redux stellt nach Dan Abramov (2023) ein Pattern zur Verwaltung und Aktualisierung von Anwendungszuständen mithilfe von Ereignissen zur Verfügung. Außerdem ermöglicht es das zentrale Speichern von Zuständen, was das Teilen von Zuständen in Anwendungen erleichtert (Dan Abramov, 2023).

Im Fall des JValue-Hub-Frontends dient Redux dazu, den Zustand eines Nutzers und seine zugehörigen Informationen im Frontend zu verwalten. Neben den Anpassungen beim Log-in wird das User-Repository sowie die zugehörige Datenbanktabelle gelöscht, da keine eigene Nutzerverwaltung mehr benötigt wird.

[12]https://github.com/authts/oidc-client-ts

Der Ablauf des Authentifizierungsprozesses im JValue-Hub in Anlehnung an Dakshitha Ratnayake (2023) läuft damit wie folgt ab:

1. Ein Nutzer betätigt den Log-in Button, um seine persönlichen Projekte im JValue-Hub zu sehen.
2. Der JValue-Hub-Web erkennt, dass der Nutzer noch nicht eingeloggt ist. und leitet den Nutzer zum Autorisierungsendpunkt von Zitadel um.
3. Der Nutzer authentifiziert sich über die Eingabe der Log-in Credentials auf der Log-in-Seite von Zitadel.
4. Nach erfolgreicher Prüfung der Credentials generiert Zitadel einen Autorisierungscode und gibt diesen inklusive des PKCE-Codes an die Redirect-URI des JValue-Hub-Web zurück.
5. JValue-Hub-Web verwendet den Autorisierungscode und den PKCE, um ein Access-Token über die Tokenschnittstelle von Zitadel anzufordern.
6. Zitadel verifiziert den PKCE sowie den Autorisierungscode und gibt ein Access-Token an den JValue-Hub-Web zurück.
7. JValue-Hub-Web kann nun mithilfe des Access-Tokens auf die Ressourcen des Nutzers zugreifen.

4.4.2 Autorisierungsprozess

In den Backendservices Hub-Backend und Pipeline-Service wird ein Custom-Auth-Guard[13] aus dem Framework NestJS für das Handling der Access-Token implementiert.

Ein Guard hat die Aufgabe, zu bestimmen, ob eine Anfrage von einem Routing-Handler bearbeitet wird. Im Handler können dafür Bedingungen wie Rollen zur Autorisierung verwendet werden (NestJS, n. d.).

Der implementierte Custom-Auth-Guard ruft über die 'oauth/v2/introspect'-Schnittstelle von Zitadel Informationen über einen Nutzer ab und gibt diese an den jeweiligen Controller weiter. In den abgerufenen Informationen sind unter anderem die Rollen eines Nutzers in Form von Metadaten enthalten. Auf Basis dieser Metadaten können Berechtigungsentscheidungen analog dem in Sektion 4.3.3 beschriebenen Rollenkonzept realisiert werden.

Die prototypische Implementierung zeigt die Machbarkeit der in der Sektion 4.3.2 beschriebenen Lösung auf. Im jetzigen Implementierungsstand kann eine Authentifizierung über den IAM-Provider auf Basis des OIDC-Code-Flows mit

[13] https://docs.nestjs.com/guards

PKCE und eine Autorisierung anhand der in den Token enthaltenen Metadaten erfolgen.

5 Evaluation

5.1 Überprüfung der Konformität des Konzepts mit den definierten Anforderungen

In der Tabelle 5.1 erfolgt ein Abgleich der Anforderungen aus der Anforderungsanalyse in Sektion 4.2 und deren Umsetzung durch das Konzept.

Anforderung	**Umsetzung im Konzept**
R1	Möglichkeit zur Umsetzung über die Metadaten des Access-Tokens
R2	Abbildung von Projektrollen im Access-Token über Metadaten und Abfrage über die Introspektionsschnittstelle von Zitadel zur Laufzeit
R3	Verwaltung von Teams und Organisationen erfolgt über einen Administrationsendpunkt im JValue-Hub-Web in Kombination mit der zur Laufzeit abgefragten Introspektionsschnittstelle
R4	Ermöglicht durch JWT bearer mit privaten Schlüsseln nach RFC 7523 in Kombination mit Introspektionsschnittstelle von Zitadel
R5	Ermöglicht durch JWT bearer mit privaten Schlüsseln nach RFC 7523 in Kombination mit Introspektionsschnittstelle von Zitadel
R6	Gegeben durch den Einsatz einer IAM-Lösung, die das Passwort-Handling abwickelt
R7	Zitadelfeature, welches im Endnutzer Log-in Prozess berücksichticht wird
R8	Zitadelfeature, das nicht genauer im Konzept betrachtet wird
R9	Realisiert durch die Verwendung eines Access-Token über alle JValue-Hub-Services hinweg
R10	Zugunsten erhöhter Sicherheit nicht berücksichtigt
R11	Realisiert durch OIDC-Code-Flow mit PKCE
R12	Realisiert durch OIDC-Code-Flow mit PKCE

Tabelle 5.1: Umsetzungsbeschreibung der Anforderungen

Die Möglichkeit zur Unterscheidung verschiedener Nutzertypen wird im Konzept nicht direkt betrachtet, da es derzeit noch keine bezahlten Nutzungsszenarien im JValue-Hub gibt. Allerdings lässt sich diese Anforderung leicht durch die für das Rollenkonzept verwendeten Metadaten umsetzen. Analog zum Setzen der Rollen können durch den Set-User-Metadaten-Endpunkt von Zitadel Informationen über

den Typ eines Nutzers hinterlegt werden, die dann im Token und bei Abfragen des Introspektionsendpunktes zurückgeliefert werden. Damit ist die Möglichkeit zur Umsetzung von R1 durch das Konzept gegeben.

Die Umsetzung von R2 und R3 wird ausführlich in Sektion 4.3.3 beschrieben. Rollen innerhalb eines Projekts, Teams und Organisationen lassen sich über die Metadaten eines Nutzers in Zitadel wie beschrieben abbilden. Somit sind R2 und R3 ebenfalls durch das Konzept erfüllt.

R4 und R5 beziehen sich beide auf die Authentifizierung und Autorisierung der Maschine-zu-Maschine-Kommunikation. Diese wird in den Sektionen 4.3.2 sowie 4.3.3 beschrieben. R4 und R5 sind somit durch das Konzept realisierbar.

R6 wird implizit durch das Verwenden der IAM-Lösung realisiert. R7 und R8 lassen sich ebenfalls durch Features der IAM-Lösung umsetzen, wobei R7 im Authentifizierungskonzept für Endnutzer als notwendig beschrieben ist.

Single Sign-On wird im Konzept nicht genauer betrachtet, ergibt sich aber durch die Verwendung eines Access-Tokens über alle Microservices des JValue-Hub hinweg, wodurch R9 ebenfalls im Konzept realisiert ist.

Die Möglichkeit zum Refresh eines Tokens nach Ablauf der Tokengültigkeit ist durch die verwendete IAM-Lösung gegeben. Diese wird jedoch aufgrund der sich durch die Verwendung von Refresh-Token in Zitadel ergebenden Risiken, die in Sektion 4.3.2 beschrieben sind, nicht im Konzept realisiert. Die Sicherheitsanforderungen, die sich durch die zukünftig vorgesehenen mobilen Szenarien sowie die aktuelle Verwendung einer SPA ergeben, überwiegen an dieser Stelle die sich aus der Verwendung von Refresh-Token ergebende Benutzerfreundlichkeit.

Der Einsatz des OpenID-Connect-Code-Flow mit PKCE ist ein sicheres Standardverfahren zur Anmeldung und berücksichtigt dabei die notwendige Sicherheit für mögliche zukünftige mobile Szenarien, wodurch das Konzept die Anforderungen R11 und R12 erfüllt.

Daraus folgt, dass alle Anforderungen aus Sektion 4.2, mit dem beschriebenen Konzept umsetzbar sind. Die Möglichkeit zum Refresh eines Tokens nach Ablauf der Tokengültigkeit wird jedoch aufgrund der erhöhten Sicherheitsanforderungen nicht umgesetzt. Nach der Analyse der Konformität des Konzepts mit den definierten Anforderungen wird in der nachfolgenden Sektion ein Vergleich zwischen dem entwickelten Konzept und den Ergebnissen aus der Literaturanalyse durchgeführt.

5.2 Abgleich der Ergebnisse der Literaturanalyse mit dem Konzept

5.2.1 Tokenersteller

Analog zu Kapitel 3 soll dabei zunächst die Frage betrachtet werden, an welcher Stelle die Token für die Authentifizierung und Autorisierung erzeugt werden. Im Konzept verwendet Zitadel einen zentralen Service zur Tokenerstellung. Dieser erzeugt die Access-Token, die zur Autorisierung verwendet werden. In der Literaturanalyse werden zwei Möglichkeiten für den Tokenersteller dargestellt: zum einen die Verwendung eines zentralen Services und zum anderen das API-Gateway. Somit stimmt das Konzept mit den Ergebnissen der für die Autorisierung verwendeten Token in der Literaturanalyse überein.

Neben der Erstellung der Access-Token ist es laut Konzept ebenfalls notwendig, Token für die Maschine-zu-Maschine-Kommunikation mit dem JValue-Hub zu erzeugen. In diesem Fall agieren externe Services, die nicht Teil des JValue-Hub sind, als Tokenersteller, um sich zu authentifizieren. Die dabei generiert Token können aber nicht direkt zur Autorisierung im System verwendet werden, sondern dienen lediglich zur Erstellung eines Access-Tokens. Eine solche Art der Tokenerstellung ist in der Literaturanalyse nicht identifiziert worden. Dahingehend weicht das Konzept von den Ergebnissen der Literaturanalyse ab.

Als Nächstes wird die Tokenanzahl und der Tokeninhalt des Konzepts mit den Ergebnissen der Literaturanalyse verglichen.

Tokenanzahl und Tokeninhalt

Bei Tokenanzahl und Tokeninhalt weicht das Konzept in Teilen von den in der Literaturanalyse gefundenen Ergebnissen ab. Für den Nutzerauthentifizierungsprozess werden im Konzept zwei Arten von Token verwendet: ein Access-Token und ein ID-Token. Das Access-Token ist für den Zugriff auf beschränkte Ressourcen vorgesehen und dient zur Autorisierung. Es enthält keine direkten Informationen über den Nutzer, sondern nur eine Identifikation über dessen ID, welche im Payload des JWT enthalten ist sowie dessen Berechtigungen, die im vorliegenden Konzept als Nutzermetadaten gespeichert werden. Ein Beispiel für ein Access-Token zeigt die Abbildung 4.5.
Informationen zu einem Anwender sind im ID-Token enthalten. Dieses beinhaltet alle Daten zum Tokenbesitzer, wie E-Mail-Adresse, Vor- und Nachname sowie, sofern in Zitadel konfiguriert, die entsprechenden Rollen für die Autorisierung. Bei der Maschine-zu-Maschine-Kommunikation erhält der Service nach der Authentifizierung nur ein Access-Token und kein ID-Token. Die enthaltenen Informationen entsprechen hinsichtlich der Berechtigungen aber denen des Endnutzers. Damit

gibt es im System insgesamt drei verschiedene Arten von Token: Access-Token, ID-Token sowie spezielle JWT, die im Rahmen der Authentifizierung bei Maschine-zu-Maschine-Kommunikation verwendet werden.

In der Literatur findet sich der Ansatz der Verwendung eines Access-Tokens in Kombination mit einem ID-Token ebenfalls wieder, jedoch gibt es auch abweichende Ansätze. Beispielsweise wird das ID-Token im vorliegenden Konzept nur für die Anzeige von Benutzerinformationen verwendet. Bei Nehme et al. (2019) wird das ID-Token hingegen zusätzlich für die Nutzerrechteprüfung verwendet.

Verbreitung von Informationen über das Gesamtsystem

Die Verbreitung der Informationen über das Gesamtsystem erfolgt im Konzept über das Access-Token. Dieses wird zwischen den unterschiedlichen Services im JValue-Hub zur Autorisierung weitergereicht. Das Token erlaubt den Zugriff auf beschränkte Ressourcen eines Nutzers und ermöglicht die Prüfung, ob der Besitzer des Access-Token auf bestimmte Ressourcen zugreifen darf. Dafür wird das Access-Token immer wieder zur Prüfung auf Aktualität und Validität an die Introspektionsschnittstelle von Zitadel gesendet. Diese gibt die Gültigkeit und aktuelle Autorisierungsinformationen zu einem Access-Token zurück.

In der Literatur finden sich Beispiele, in denen der Ressourcenzugriff ebenfalls von anderen Microservices innerhalb der Anwendung abhängig ist. Das ist im vorliegenden Konzept nicht der Fall, da die Informationen bezüglich Autorisierungsentscheidungen zentral in Zitadel gespeichert und bislang keine Kontextinformationen aus anderen Services notwendig sind. Der Ansatz, dass die Token zur Verbreitung von Berechtigungsinformationen über das Gesamtsystem verwendet werden, findet sich aber ebenfalls in der Literatur in Chandramouli et al. (2021), Salibindla (2018), ShuLin und JiePing (2020) sowie in Preuveneers und Joosen (2019) wieder.

Signierung

Die Signierung des Access-Tokens wird im Konzept nicht gesondert behandelt, erfolgt aber ebenfalls durch Zitadel, wofür JSON Web Keys genutzt werden. Dakshitha Ratnayake (2023) beschreibt, dass die verwendeten öffentlichen Schlüssel über den '/oauth/v2/keys'-Endpunkt von Zitadel abgerufen und somit überprüft werden können. Eine manuelle Validierung ist aufgrund der Gültigkeitsabfragen an den Introspektionsendpunkt möglich, im laufenden Betrieb aber nicht zwingend notwendig. Es erfolgt also analog zu den in der Literatur gefundenen Ergebnissen eine asymmetrische Signierung der Token.

Gültigkeitsdauer eines Tokens

Ein weiterer Aspekt, welcher nicht separat im Konzept aufgeführt ist, aber dennoch betrachtet wurde, ist die Gültigkeitsdauer. Diese kann für jedes der im Konzept verwendeten Token separat in Zitadel konfiguriert werden. Zitadel gibt als Standardwert hierbei zwölf Stunden für das Access- und ID-Token an (Max Peintner, 2023a). Hier weicht der Standardwert von Zitadel von den in der Literatur verwendeten Definitionen ab. Die Ergebnisse der Literatur in Sektion 3.1.7 unterscheiden zwischen Token mit kurzer, mittlerer und langer Gültigkeit. Der Defaultwert von Zitadel befindet sich damit am ehesten im Bereich der kurzlebigen Token, ist dafür aber eigentlich zu lange gültig. Da im Konzept die Verwendung von Refresh-Token aufgrund der Sicherheitsanforderungen nicht vorgesehen ist, entscheidet sich der Autor für den Default Wert von Zitadel. Dieser erfordert keine mehrmalige Anmeldung von JValue-Hub Anwendern innerhalb eines Arbeitstages und steigert damit die Benutzerfreundlichkeit. Im Falle eines Tokendiebstahls ist die Gültigkeitsdauer des Access-Tokens dennoch begrenzt und der Zugang zu den geschützten Ressourcen kann nicht erneuert werden.

Widerruf der Gültigkeit eines Tokens

Ein weiteres Thema, das in der Literaturanalyse betrachtet wird und in diesem Zusammenhang eine Rolle spielt, ist der Widerruf der Gültigkeit eines Tokens innerhalb seiner Gültigkeitsdauer.

Zitadel ermöglicht dies durch eine Widerruf-Schnittstelle, bei der Access- oder Refresh-Token widerrufen werden können. Die Prüfung, ob ein Token widerrufen wurde, erfolgt über den Introspektionsendpunkt. Dieser liefert als Antwort, ob ein Token noch aktiv oder bereits nicht mehr gültig ist (Max Peintner, 2023c). Analog zu der in der Literaturanalyse in Sektion 3.2.1 im Abschnitt Zentraler Authentifizierungsdienst in Kombination mit JWT betrachteten Variante ist es auch möglich, eine Client-seitige Validierung der JSON Web Token an den Ressourcenmicroservices durchzuführen. Die Option zum Widerruf der Token ist durch eine lokale Validierung allerdings nicht mehr möglich. Deshalb werden die Token bei jedem Request zur Überprüfung an den Introspektionsendpunkt gesendet. Die daraus resultierende erhöhte Requestanzahl stellt in der aktuellen Ausbaustufe des Produkts kein Problem dar. Deshalb wird im Konzept eine zentrale Überprüfung der JWT durch den Authentifizierungsservice gewählt. Für diese Wahl spricht ebenfalls, dass die Projektmitglieder des JValue-Hub die Anwendung perspektivisch auf mobilen Endgeräten einsetzen wollen. Dieses Szenario bietet zusätzliche Angriffsvektoren für einen Tokendiebstahl, weshalb die Möglichkeit zum Widerruf von Token perspektivisch an Bedeutung gewinnen wird.

In der Literatur (vgl. Sektion 3.1.8) findet sich eine Variante für den Widerruf der Gültigkeit eines Tokens, in der ein OAuth Access-Token wiederholt an den

Autorisierungsserver geschickt wird, um die Gültigkeit zu prüfen. Diese Variante entspricht dem im Konzept verwendeten Ansatz.

5.2.2 Authentifizierung und Autorisierung von Token

Die Authentifizierung von Endnutzern wird im Konzept mithilfe eines zentralen Authentifizierungsdienstes durchgeführt. Dieser Ansatz findet sich in der analysierten Literatur in Sektion 3.2.1 wieder.
Für die Authentifizierung von Service-zu-Service findet sich in der Literatur nur die Variante von Barabanov und Makrushin (2020), die in Sektion 3.2.1 im entsprechenden Abschnitt beschrieben ist und auf dem Austausch von Passwörtern basiert. Hier weicht das Konzept von den in der Literatur gefundenen Ansätzen ab. Bei der Autorisierung wird hingegen ein aus der Literatur bekannter Ansatz verwendet: die Autorisierung am Ressourcenmicroservice aus Sektion 3.2.2. Im vorliegenden Konzept findet analog zu in der Literatur vorhanden Varianten zur Autorisierung die Prüfung der Berechtigungen eines Nutzers am jeweiligen Ressourcenservice statt, der für die Ressource zuständig ist.

5.2.3 Umsetzung von Autorisierungspolicies im Microserviceumfeld auf Basis von Token

Wie im Konzept beschrieben, wird ein rollenbasierter Ansatz für die Zugriffsbeschränkung von Nutzern auf Ressourcen verwendet. Aufgrund der verhältnismäßig geringen Komplexität und der Größe des Projektteams, das am JValue-Hub arbeitet, wird eine direkte Implementierung im Code der Microservices einer Entkopplung der Accesspolicies durch Access Policy Sprachen wie XACML oder OPA vorgezogen. Dennoch ist im Konzept vorgesehen, die Accesspolicies zentral in einer Bibliothek zu verwalten, die in allen Microservices zum Einsatz kommt, um Unterschiede bei den Autorisierungsvorgängen zwischen Services zu vermeiden. Diese Variante ist ebenfalls in der Literatur zu finden. Weitere Vergleiche bezüglich der Realisierung von Autorisierungspolicies sind an dieser Stelle allerdings nicht möglich, da diese im Konzept keine Anwendung finden.

Nach dem Abgleich der Ergebnisse der Literaturanalyse mit den Ansätzen im ausgearbeiteten Konzept erfolgt im letzten Kapitel dieser Arbeit eine Diskussion der Ergebnisse. Ferner werden die Limitationen der Arbeit aufgezeigt und Schlussfolgerungen abgeleitet.

6 Diskussion und Schlussfolgerungen

6.1 Diskussion

Aus der Literaturanalyse geht hervor, dass bereits umfangreiche Forschungsergebnisse im Bereich der Authentifizierung und Autorisierung von Microservices vorliegen. Tokenbasierte Ansätze wurden innerhalb dieser Studien untersucht und dabei verschiedene Lösungsansätze vorgestellt.

Jedoch fehlen oftmals direkte Vergleiche zwischen den bereits vorhandenen Standard Frameworks. Dieser Mangel an umfassenden Vergleichsstudien erschwert es Softwareentwickler und -architekten in der Praxis, das am besten geeignete Framework für ihre spezifischen Anforderungen auszuwählen. Vergleichsstudien, welche speziell die Stärken und Schwächen der verschiedenen Frameworks herausarbeiten und klare Anwendungsszenarien empfehlen, konnten in der vorliegenden Arbeit nicht identifiziert werden.

Insbesondere im Bereich der praktischen Anwendung der tokenbasierten Authentifizierung und Autorisierung von Microservices herrscht immer noch eine unzureichende Studienlage vor. Zu einer ähnlichen Schlussfolgerung kommen auch andere Autoren, wie Nguyen und Baker (2019), Jander et al. (2018), de Almeida und Canedo (2022) und Pereira-Vale et al. (2019), die einen Mangel an praktischen Studien zur Implementierung von Sicherheit in Microservices im Allgemeinen feststellen. Jedoch sind praktische Fallstudien und Implementierungsbeispiele entscheidend, um die Machbarkeit und Effektivität dieser Ansätze in produktiven Anwendungen zu demonstrieren.

Die vorliegende Arbeit adressiert diese Forschungslücke, indem sie die Konzeption und Implementierung eines Authentifizierungs- und Autorisierungskonzept in der Praxis demonstriert.

Es gilt jedoch zu beachten, dass Limitierungen hinsichtlich der Demonstration in dieser Arbeit bestehen. Das aufgezeigte Konzept ist zwar prototypisch im-

plementiert, jedoch wurden keine Untersuchungen bezüglich Skalierbarkeit und Performance unternommen. Ebenfalls wurde die Anwendung nicht in eine produktive Umgebung deployt. Ferner wurden keine Penetrationstests durchgeführt, um mittels standardisierter Angriffsszenarien mögliche Schwächen in der Konzeption und Umsetzung zu identifizieren.

Die durchgeführte strukturierte Literaturanalyse unterliegt ebenfalls Limitationen, da diese nur auf den zu Beginn der Arbeit ausgearbeiteten Suchbegriffen basiert. Aufgrund des Umfangs der Arbeit wurden weitere Methoden, wie das Snowballing von Papern, nicht mehr angewandt, sodass gegebenenfalls relevante Ergebnisse nicht identifiziert und einbezogen werden konnten.

6.2 Schlussfolgerungen

In der vorliegenden Arbeit wird das Thema tokenbasierte Authentifizierung und Autorisierung in Microservices untersucht. Dabei werden zunächst durch eine umfassende Literaturanalyse die Forschungsfragen aus Sektion 1 beantwortet.

Die Untersuchung von RQ1 verdeutlicht, dass die Erstellung von Token idealerweise durch einen dedizierten Service erfolgt. Die Wahl des geeigneten Token-Typs ist abhängig von den spezifischen Anforderungen des Use Cases und den verwendeten Frameworks.

Die Ergebnisse von RQ2 zeigen, dass ein Token in den meisten analysierten Papern relevante nutzerbezogene Informationen, sowie Berechtigungen enthält. Die Gültigkeitsdauer der Token sollte möglichst kurz sein, um die Sicherheit des Systems zu erhöhen und Missbrauch zu minimieren. Dabei ist festzuhalten, dass in der untersuchten Literatur die Gültigkeitsdauer in den meisten Fällen zwischen wenigen Minuten und einer halben Stunde liegt.

Die Analyse von RQ3 hat gezeigt, dass die tokenbasierte Authentifizierung über einen zentralen Authentifizierungsdienst oder, sofern vorhanden, in Kombination mit einem Gateway erfolgen kann. Die Autorisierung wird in den meisten Papern direkt am Ressourcenmicroservice durchgeführt, kann aber ebenfalls mit einem Gateway kombiniert werden. Die Wahl ist abhängig von den spezifischen Sicherheitsanforderungen und der Architektur des Microservices-Systems. Als Alternative können Service Meshes eingesetzt werden, sofern diese bereits Teil der Anwendungsarchitektur sind.

Abschließend zeigt die Erforschung von RQ4, dass Access Policies ebenfalls mithilfe von Token realisiert werden können. Die Sektion 3.3 liefert dazu theoretische Grundlagen und die Beispielarchitektur aus Sektion 3.3.4 dient als Orientierungspunkt für eine praktische Implementierung.

Im Anschluss an die Präsentation der Ergebnisse der Literaturanalyse wird die auf-

bereitete Theorie in einer Demonstration in die Praxis überführt. Im Zuge dessen wird gezeigt, wie eine tokenbasierte Authentifizierung und Autorisierungslösung in einer Microservice-Architektur konzeptioniert und prototypisch implementiert wird. Im Rahmen der Konzeption erfolgt die Auswahl einer Open Source IAM Lösung auf Basis einer nach Nickerson et al. (2013) entwickelten Taxonomie. Die ausgearbeitete Lösung ermöglicht die Verwaltung und Autorisierung von Nutzern auf Basis von Organisationen, Teams, Projekten sowie Nutzertypen und Rollen. Die Zuweisung von Rollen und Eigenschaften erfolgt mittels Metadaten innerhalb eines Tokens. Dies ermöglicht eine besonders flexible Erweiterbarkeit der Autorisierungslösung, da die Metadaten eines Tokens beliebig anpassbar sind.

Das entwickelte Konzept erfüllt die in Sektion 4.2 definierten Anforderungen und die prototypische Implementierung in 4.4 zeigt die Machbarkeit des Konzepts in der Praxis.

Die vorliegende Arbeit liefert Erkenntnisse, die das Verständnis der Implementierung von tokenbasierten Authentifizierungs- und Autorisierungslösungen in Microservices verbessern. Es wird empfohlen, dass zukünftige Forschungsarbeiten sich auf die Entwicklung klarer Handlungsempfehlungen und Richtlinien für die Implementierung produktiver Anwendungen konzentrieren.

Appendices

A Strukturierte Literaturanalyse

Die auf den nächsten Seiten nachfolgenden Tabellen zeigen alle identifizierten Publikationen der Literaturanalyse inklusive Autor, Titel sowie sofern vorhanden das Ausschlusskriterium.

Tabelle A1: Literatur strukturierte Literaturanalyse Titel 1-10

No.	Autor(en)	Titel	Inkludiert	Ausschlussbegründung
1	Kalman, Mattias	A Distributed IoT Microservice using a Custom Ethereum Based Security Protocol		handelt von IoT security vulnerabilities ->EC1
2	Hill, Keara L	A Formal Model of Role-Based Access Control for a Microservice-Based Malware Analysis Platform		Nicht zugreifbar ->Erfüllt IC?2 nicht
3	Ranganathan, Rajagopalan	A highly-available and scalable microservice architecture for access management		Masterarbeit ->Erfüllt IC4 nicht
4	Sodhro, Ali Hassan Lakhan, Abdullah Pirbhulal, Sandeep Groenli, Tor Morten Abie, Habtamu	A Lightweight Security Scheme for Failure Detection in Microservices IoT-Edge Networks		IoT Edge Networks ->EC1
5	Barabanov, Alexander	A Method for Collecting Security-Specific Architectural Information for Microservice-Based Systems for Design Security Assessment		Ansatz zur Sammlung von Microservice-basierten Architekturinformationen zur Sicherung von Anwendungen ->EC1
6	Rojo, Javier Herrera, Juan Luis Moguel, Enrique Hernández, Juan	A Microservice Architecture for Access Control Based on Long-Distance Facial Recognition		Gesichtserkennung für die Bewältigung alltäglicher Aufgaben in Wohnheimen->EC1
7	Catalfano, Alessio Ruggeri, Armando Celesti, Antonio Fazio, Maria Villari, Massimo	A Microservices and Blockchain Based One Time Password (MBB-OTP) Protocol for Security-Enhanced Authentication		Dezentralisiertes Blockchain basierendes One Time Password (MBB-OTP)-Protokoll für eine sicherheitserhöhte Authentifizierung ->EC1
8	Damis, H. A. Shehada, D. Fachkha, C. Gawanmeh, A. Al-Karaki, J. N	A microservices architecture for ads-b data security using blockchain		Blockchain-Implementierung innerhalb eines Microservices-Frameworks für die ADS-B-Datenüberprüfung ->EC1
9	Liu, Aodi Du, Xuehui Wang, Na Zhang, Liu	A MRP-based policy conflict resolution mechanism for micro-service composition		Mechanismus zur Lösung von Richtlinienkonflikten ->EC1
10	La Franca, Marco Marino, Luca Martorana, Leonardo Leo, Marco Carcagnì, Pierluigi Distante, Cosimo Sergi, Ilaria Patrono, Luigi	A Novel Approach based on Microservices Architectures and Computer Vision to improve access to Culture Heritage		Computer-Vision-Techniken ->EC1

Tabelle A2: Literatur strukturierte Literaturanalyse Titel 11-20

No.	Autor(en)	Titel	Inkludiert	Ausschlussbegründung
11	Washington Henrique Carvalho Almeida Luciano De Aguiar Monteiro Anderson Cavalcanti De Lima Raphael Rodrigues	A Survey on Microservice Security–Trends in Architecture, Privacy and Standardization on Cloud Computing Environments		Stellt Elemente vor, die bei der Entwicklung von Lösungen auf der Grundlage von Microservices berücksichtigt werden sollten ->EC1
12	Berardi, Davide	A Survey on Microservices Security: Preliminary Findings· PDF file A Survey on Microservices Security		Nur 2 Seiten und nur vorläufige Ergebnisse ->erfüllt IC4 nicht
13	Yu, Dongjin Jin, Yike Zhang, Yuqun Zheng, Xi	A survey on security issues in services communication of Microservices enabled fog applications	x	
14	Abidi, Sarra Essafi, Mehrez Guegan, Chirine Ghedira Fakhri, Myriam Witti, Hamad Ghezala, Henda Hjjami Ben	A web service security governance approach based on dedicated micro-services		Governance-Ansatz für die Sicherheit von Webdiensten ->EC1
15	Preuveneers, Davy Joosen, Wouter	Access Control with Delegated Authorization Policy Evaluation for Data-Driven Microservice Workflows	x	
16	Cárdenas Sánchez Brian Camilo Carlos Arturo Olarte Rojas	Amazon Web Service Microservice Security Analysis		Nicht zugreifbar (Paper lädt nicht)->Erfüllt IC2 nicht
17	Nasab, Ali Rezaei Shahin, Mojtaba Raviz, Seyed Ali Hoseyni Liang, Peng Mashmool, Amir Lenarduzzi, Valentina	An Empirical Study of Security Practices for Microservices Systems	x	
18	Fu, Guo Sun, Jin Zhao, Jiantao	An optimized control access mechanism based on micro-service architecture		schlägt ein auf RBAC basierendes Optimierungsmodell für die Zugangskontrolle vor (nicht auf Basis von Token)->EC1
19	Cezary Wojciech Elert	Analysis of a microservice application that supports various types of authorization		Masterarbeit ->Erfüllt IC4 nicht
20	Jacob, Stephen Qiao, Yuansong Ye, Yuhang Lee, Brian	Anomalous distributed traffic: Detecting cyber security attacks amongst microservices using graph convolutional networks		Verwendet ein Diffusion Convolutional Recurrent Neural Network, um Vorhersagen für laufende Microservice-Aktivitäten zu treffen ->EC1

Tabelle A3: Literatur strukturierte Literaturanalyse Titel 21-30

No.	Autor(en)	Titel	Inkludiert	Ausschlussbegründung
21	Chunlei, Chang Liang, Bai Kai, Yun Kai, Li	Application of Authentication and Secret-key Distribution Mechanism of Challenge-response in Micro-service Security Supervision and Authentication Interaction		Führt Authentifizierungs- und Geheimschlüsselverteilungsmechanismus 'Challenge-Response' ein ->EC1
22	Yi, Ding Donghui, Tong Xuegang, Wang Shuliang, Zhang	Application of Authorization in Smart Grid based on the PasS Microservice architecture		Architektur auf Basis von SAML und XACML ohne Token ->EC1
23	Jacob, S. Lee, B. Qiao, Y.	Applying process mining to improve microservices cyber security situational awareness		Business Process Mining zur Verbesserung der Erkennung von Cybersicherheitsangriffen in Microservices ->EC1
24	Nguyen, Quy Baker, Oras	Applying Spring Security Framework and OAuth2 To Protect Microservice Architecture API	x	
25	Chatterjee, Ayan Prinz, Andreas	Applying spring security framework with keycloak-based OAuth2 to protect microservice architecture APIs: A case study	x	
26	Martins, Rafael Wickboldt, Juliano Araujo Granville, Lisandro Zambenedetti	Assisted Monitoring and Security Provisioning for 5g Microservices-Based Network Slices with Sweeten		SWEETEN wird als Lösung für die Automatisierung der Bereitstellung und die transparente Integration von Netzverwaltungslösungen aus verschiedenen Verwaltungsbereichen vorgeschlagen ->EC1
27	Chandranouli, Ramaswamy Butcher, Zack Chetal, Aradhna	Attribute-based Access Control for Microservices-based Applications Using a Service Mesh	x	
28	Barabanov, Alexander Makrushin, Denis	Authentication and authorization in microservice-based systems: survey of architecture patterns	x	
29	Almeida, Murilo Góes de Canedo, Edna Dias	Authentication and Authorization in Microservices Architecture: A Systematic Literature Review		Enthält interessante Erkenntnisse zum Thema auf Basis einer LR gibt aber keine konkreten Handlungsempfehlungen ->EC2
30	Xiuyu He Xudong Yang	Authentication and Authorization of End User in Microservice Architecture	x	

Tabelle A4: Literatur strukturierte Literaturanalyse Titel 31-40

No.	Autor(en)	Titel	Inkludiert	Ausschlussbegründung
31	Banati, A. Kail, E. Karoczkai, K. Kozlovszky, M.	Authentication and authorization orchestrator for microservice-based software architectures	x	
32	Rezaei Nasab, Ali Shahin, Mojtaba Liang, Peng Basiri, Mohammad Ehsan Hoseyni Raviz, Seyed Ali Khalajzadeh, Hourieh Waseem, Muhammad Naseri, Amineh	Automated identification of security discussions in microservices systems: Industrial surveys and experiments		Automatische Identifizierung von Sicherheitsdiskussionen ->EC1
33	Chondamrongkul, Nacha Sun, Jing Warren, Ian	Automated security analysis for microservice architecture		automatisierter Ansatz zur Sicherheitsanalyse für Microservice-Architekturen ->EC1
34	Xing Li Yan Chen Zhiqiang Lin Xiao Wang Jim Hao Chen	Automatic Policy Generation for {Inter-Service} Access Control of Microservices		Beitrag stellt AUTOARMOR vor zur Generierung von Zugriffskontrollrichtlinien für Microservices ->EC1
35	Sprabery, Read	Capabilities for cross-layer micro-service security		Doktorarbeit ->Erfüllt IC4 nicht
36	Kallergis, Dimitrios Garofalaki, Zacharenia Katsikogiannis, Georgios Douligeris, Christos	CAPODAZ: A containerised authorisation and policy-driven architecture using microservices		richtliniengesteuerte Berechtigungen für Machine-to-Machine in einer hybride Cloud-basierten Infrastruktur und Internet of Things (IoT)-Dienste verwendet ->EC1
37	Kothawade, Prasad Bhowmick, Partha Sarathi	Cloud Security: Penetration Testing of Application in Micro-service architecture and Vulnerability Assessment.		Masterarbeit ->Erfüllt IC4 nicht
38	Gerking, Christopher Schubert, David	Component-Based Refinement and Verification of Information-Flow Security Policies for Cyber-Physical Microservice Architectures		Verfeinerung und Verifizierung von Sicherheitsrichtlinien für Informationsflüsse in cyber-physischen Microservice-Architekturen ->EC1
39	Martin, Azanguezet Quimatio Benoit Auvarique, Ndanteng Tchakounté Fidèle, Tsognong Nkenlifack, Marcellin Julius	Continuous Single-Sign-On (CSSO) method for authentication and authorization in microservices architectures		preprint->erfüllt IC4 nicht
40	Surya Sai Venkatesh, Dasari Agarwal, Shivali	Data Access Pattern Recommendations for Microservices Architecture		Muster für Datenzugriffe auf der Basis der Trennung von Lese- und Schreibvorgängen in den vom Dienst durchgeführten Transaktionen ->EC1

Tabelle A5: Literatur strukturierte Literaturanalyse Titel 41-50

No.	Autor(en)	Titel	Inkludiert	Ausschlussbegründung
41	Walpita, P. A.	Defense In-depth security framework for Netflix OSS Micro Services		Masterarbeit ->Erfüllt IC4 nicht
42	Jander, Kai Braubach, Lars Pokahr, Alexander	Defense-in-depth and Role Authentication for Microservice Systems	x	
43	Suomalainen, Joel	Defense-in-Depth Methods in Microservices Access Control		Masterarbeit ->Erfüllt IC4 nicht
44	Dawei, Yang Yang, Gao Wei, He Kai, Li	Design and achievement of security mechanism of api gateway platform based on microservice architecture		Sicherheitsmechanismus auf Basis der API-Gateway-Plattform (keine beschreibung von token based authentifizierung oder autorisierung) ->EC1
45	Yan, Kun Pan, Yu Sui, Yongxin Ye, Song	Design and Application of Security Gateway for Transmission Line Panoramic Monitoring Platform based on Microservice Architecture		gemeinsamen Zugangstechnologien und Geschäftsszenarien des Internet der Dinge-Gateway ->EC1
46	Guo, Wei Chang, Rong Li, Bangyuan	Design of Real-time Data Access Scheme Based on Microservice Grid Operation		Zugriffsschemas auf Echtzeitdaten für den Betrieb des Stromnetzes ->EC1
47	Jacob, Stephen Qiao, Yuansong Lee, Brian	Detecting Cyber Security Attacks against a Microservices Application using Distributed Tracing.		verteiltes Tracing zur Erkennung von Cyber-Sicherheitsangriffen ->EC1
48	Chou, Joe Al-Masri, Eyhab Kanzhelev, Sergey Fattah, Hossam	Detecting Security and Privacy Risks in Microservices End-to-End Communication Using Neural Networks		Entwicklung eines Modells zur Erkennung von Datenschutz- und Sicherheitsrisiken ->EC1
49	Bahareh Sadeghian Boroujeni	Development of a shared authentication system - a microservice approach		Masterarbeit ->Erfüllt IC4 nicht
50	Tihomir Dimitrov Tenev	Development Of Hierarchical Taxonomy That Incorporates Patterns For Improving Security In Information Systems Based On Microservice Architecture		PhD ->Erfüllt IC4 nicht

Tabelle A6: Literatur strukturierte Literaturanalyse Titel 51-60

No.	Autor(en)	Titel	Inkludiert	Ausschlussbegründung
51	Díaz-Sánchez, Daniel Marín-Lopez, Andrés Almenárez Mendoza, Florina Arias Cabarcos, Patricia	DNS/DANE Collision-Based Distributed and Dynamic Authentication for Microservices in IoT		Der Artikel schlägt eine Lösung vor, die den DNSSEC/DANE-Signaturmechanismus durch die Verwendung von Chamäleon-Signaturen und die Definition eines neuen Soft-Delegation-Schemas modifiziert - >EC1
52	Sánchez, Daniel López, Andrés Mendoza, Florina Cabarcos, Patricia Arias	DNS-Based Dynamic Authentication for Microservices in IoT		Sicherheitsunterstützung für Fog Computing und die Verwendung von Microservice-Architekturen in Verbindung mit DNSSEC, DANE und Chamäleonsignaturen ->EC1
53	Baynes, K. Gilman, J. Pilone, D. Mitchell, A. E.	Doing One Thing Well: Leveraging Microservices for NASA Earth Science Discovery and Access Across Heterogenous Data Sources		Talk Transkript nicht zugreifbar ->IC2
54	Melara, Marcela S. Bowman, Mic	Enabling Security-Oriented Orchestration of Microservices		Sicherheitsrichtlinien-Framework für sicherheitsorientierte Orchestrierung von Microservices ->EC1
55	Orazi, G. Vallitu, K. Saimo, P. Virtanen S.	Enhancing and integration of security testing in the development of a microservices environment		Masterarbeit ->Erfüllt IC4 nicht
56	Nadim Habbal	Enhancing Availability of Microservice Architecture: A Case Study on Kubernetes Security Configurations		Abschlussarbeit - >Erfüllt IC4 nicht
57	Davide Berardi Saverio Giallorenzo Jacopo Mauro Andrea Melis Fabrizio Montesi Marco Prandini	Enhancing security in Microservice environments		Schwachstellenanalyse von Microservices auf Basis STRIDE aufzudecken und Empfehlungen für die entsprechenden STRIDE-Kategorien-> präsentiert aber keine konkreten Authentifizierungs oder Autorisierungstechniken ->EC1
58	Tetiana Yarygina	Exploring microservice security		PhD - >Erfüllt IC4 nicht
59	Muhammad Taqi Raza Fatima Muhammad Anwar Dongho Kim Kyu-Han Kim	FERRET: Fall-back to LTE Microservices for Low Latency Data Access		LTE 4G ->thematisch nicht passend ->EC1
60	Nehme, Antonio Jesus, Vitor Mahbub, Khaled Abdallah, Ali	Fine-Grained Access Control for Microservices	x	

Tabelle A7: Literatur strukturierte Literaturanalyse Titel 61-70

No.	Autor(en)	Titel	Inkludiert	Ausschlussbegründung
61	Pahl, Marc-Oliver Aubet, Francois-Xavier Liebald, Stefan	Graph-based IoT microservice security		graphbasierte Zugriffskontrolle für IOT ->EC1
62	Mostefaoui, Ahmed Gianini, Gabriele Damiani, Ernesto Min, Geyong	Guest Editorial: QoS and security in software for wireless and mobile micro-services		Guest Editorial ->IC4
63	Sung, Kim Bastani, F. B. I-Ling, Yen Ing-Ray, Chen	High-assurance synthesis of security services from basic microservices		Allgemeiner Prozess, um die Systemsicherheit in orthogonale Aspekte zu zerlegen, sodass es möglich ist, die Sicherheit eines Systems rigoros zu zertifizieren ->EC1
64	Shmeleva, Ekaterina	How Microservices are Changing the Security Landscape		Masterarbeit ->Erfüllt IC4 nicht
65	Pasomsup, Chittipat Limpiyakorn, Yachai	HT-RBAC: A Design of Role-based Access Control Model for Microservice Security Manager	x	
66	Márquez, Gastón Astudillo, Hernán	Identifying availability tactics to support security architectural design of microservice-based systems		Identifizieren und charakterisieren von Verfügbarkeitstaktiken mithilfe einer neu eingeführten Beschreibungsvorlage ->EC1
67	Tharun Kanmara Melody Moh	Identifying IoT-Based Botnets: 293A Microservice Architecture for IoT Management and Security		Kapitel aus einem Buch ->Erfüllt IC4 nicht
68	Gnuja, Daniel Siddiqui, Muhammad Shuaib	Identity and Access Control for micro-services based 5G NFV platforms		Präsentation einer 5G-Plattform-orientierten Lösung ->EC1
69	Riski Wahyu K	Implementasi Mutual Transport Layer Security (mTLS) Pada Arsitektur Microservices Dengan Istio Di Kubernetes		Nicht englisch ->Erfüllt IC1 nicht
70	Simon Tran Florén	Implementation and Analysis of Authentication and Authorization Methods in a Microservice Architecture:		Masterarbeit ->Erfüllt IC4 nicht

Tabelle A8: Literatur strukturierte Literaturanalyse Titel 71-80

No.	Autor(en)	Titel	Inkludiert	Ausschlussbegründung
71	Flora, Jose	Improving the security of microservice systems by detecting and tolerating intrusions		Doktorarbeit ->Erfüllt IC4 nicht
72	Leines-Vite, Lenin Pérez-Arriaga, Juan Carlos Limón, Xavier	Information And Communication Security Mechanisms For Microservices-Based Systems		Katalog und eine Diskussion von Sicherheitslösungen auf Basis einer LR aber keine konkreten Handlungsempfehlungen ->EC2
73	Kumar, Alok	Information security controls for multi-cloud and microservices		Informationssicherheitskontrollen für Multi-Cloud und Microservices enthält aber keine Informationen zu tokenbasierter Authentifizierung oder Autorisierung ->EC1
74	BAZENIUC, Ivan ZGUREANU, Aureliu	Information security in microservices architectures	x	
75	Safaryan, Olga Pinevich, Elena Roshchina, Evgenia Cherckesova, Larissa Kolennikova, Nadezhda	Information system development for restricting access to software tool built on microservice architecture	x	
76	Torkura, Kennedy A. Sukmana, Muhammad I.H. Meinel, Christoph	Integrating continuous security assessments in microservices and cloud native applications		Einführung eines neuen Sicherheitskontrollkonzepts ->EC1
77	Singh, Amandeep Raj, Vinay Ravichandra, Sadam	Integration of Attribute-Based Access Control in Microservices Architecture		Erklärt ABAC aber nicht auf Basis von Token ->EC1
78	Ahmadvand, Mohsen Pretschner, Alexander Ball, Keith Eyring, Daniel	Integrity protection against insiders in microservice-based infrastructures: From threats to a security framework		Rahmenwerk, das als Blaupause für einen insiderresistenten Integritätsschutz in Microservices dient (enthält keine tokenbasierte Authentifizierung oder Autorisierung) ->EC1
79	Zhang, Yasheng Li, Chengcheng Chen, Ning Zhang, Peiying	Intelligent Requests Orchestration for Microservice Management Based on Blockchain in Software Defined Networking: a Security Guarantee		Herausforderungen und Lösungen im Zusammenhang mit der Serviceanforderungsorchestrierung in Software-Defined Networks ->EC1
80	Muresu, Daniel	Investigating the security of a microservices architecture: A case study on microservice and Kubernetes Security		Masterarbeit ->Erfüllt IC4 nicht

Tabelle A9: Literatur strukturierte Literaturanalyse Titel 81-90

No.	Autor(en)	Titel	Inkludiert	Ausschlussbegründung
81	Kadarya, Srison	IT Infrastructure und Microservices Authentication		Masterarbeit ->Erfüllt IC4 nicht
82	Kanmaza, Tharun Theja	Management and Security of IoT systems using Microservices		Masterarbeit ->Erfüllt IC4 nicht
83	Xu, Rongxu Jin, Wenquan Kim, Dohyeun	Microservice security agent based on API gateway in edge computing		zu starker Fokus auf IOT bzw Edge Computing ->EC1
84	Ying, Fei Zhao, Shengjie Deng, Hao	Microservice Security Framework for IoT by Mimic Defense Mechanism		neuartiges System um die Sicherheit von Containern vor unbekannten Angriffen mithilfe des Mimic Defause Frameworks zu stärken ->EC1
85	Zdun, Uwe Queval, Pierre-Jean Simhandl, Georg Scandariato, Riccardo Chakravarty, Somik Jelic, Marjan Jovanovic, Aleksandar	Microservice Security Metrics for Secure Communication, Identity Management, and Observability		automatisierte Validierung von Sicherheitstaktiken in Microservice-Systemen ->EC1
86	Davide Berardi Saverio Giallorenzo Jacopo Mauro Andrea Melis Fabrizio Montesi Marco Prandini	Microservice security: a systematic literature review		behandelt keine Authentifizierung oder Autorisierung von Microservicearchitekturen auf Basis von Token ->EC1
87	Li, Na Liu, Guangyi Zhang, Huimin Zhao, Quan Zhao, Yun Tong, Zhou Wang, Yingying Sun, Junshuai	Micro-service-based radio access network		Entwicklung von Service-based Architecture (SBA) und ihrer Anwendung in den 6G Netzwerken ->EC1
88	Salihindla, Jyothi	Microservices API security	x	
89	Driss, Maha Hasan, Daniah Boulila, Wadii Ahmad, Jawad	Microservices in IoT security: current solutions, research challenges, and future directions		Überblick über Microservices-basierte Ansätze zur Sicherung von IoT-Anwendungen ->EC1
90	Christudas, Binildas	Microservices Security		Nicht zugreifbar ->Erfüllt IC2 nicht

Tabelle A10: Literatur strukturierte Literaturanalyse Titel 91-100

No.	Autor(en)	Titel	Inkludiert	Ausschlussbegründung
91	Abdelfattah, Amr S. Cerny, Tomas	Microservices Security Challenges and Approaches	x	
92	Indrasiri, Kasun Siriwardena, Prabath	Microservices Security Fundamentals		Buch ->erfüllt IC4 nicht
93	Dias, Wajjakkara Kankanamge Anthony Nuwan Siriwardena, Prabath	Microservices security in action		Buch ->erfüllt IC4 nicht
94	Kalubowila, D. C. Athukorala, S. M. Tharaka, B. A. S. Samarasekara, H. W. Y. R Samaratunge Arachchilage, Udara Srimath S. Kasthurirathna, Dharshana	Optimization of Microservices Security		Lösung zur Verringerung der Latenzzeit auf fehlerhafte Anfragen durch die Implementierung eines externen Validierungsmodells ->EC1
95	Yarygina, Tetiana Bagge, Anya Helene	Overcoming Security Challenges in Microservice Architectures	x	
96	Castillo Rivas, Diego Antonio Guamán, Daniel	Performance and Security Evaluation in Microservices Architecture Using Open Source Containers		Definieren einer DevOps-Pipeline, um die Leistung und Sicherheit in der Vorproduktionsumgebung zu verbessern ->EC1
97	Asik, Tugrul Selcuk, Yunus Emre	Policy enforcement upon software based on microservice architecture		Metriken und Richtlinien zur Messung der Qualität einer Anwendung, die auf einer Microservice-Architektur basiert ->EC1
98	Paschke, Adrian	Provalets: Component-Based Mobile Agents as Microservices for Rule-Based Data Access, Processing and Analytics		Provalets - mobile Regelagenten für regelbasierten Datenzugriff, semantische Verarbeitung und Inferenzanalyse, die als Microservices bereitgestellt werden können ->EC1
99	Tenev, Tihomir Tsvetanov, Simeon	Recommendations for enhancing security in microservice environment altered in an intelligent way		Schwachstellenanalyse verteilter Anwendungen mit STRIDE, Bereitstellung von Empfehlungen, Umwandlung der Empfehlungen mit Hilfe des CIM-Modells (aber keine tokenbasierten Authentifizierungs- oder Autorisierungstechniken) ->EC1
100	Ordóñez-Camacho, Diego	Reducing the IoT security breach with a microservice architecture based on TLS and OAuth2		stark auf IOT fokussiert, weicht dadurch zu weit vom eigentlichen fokus ab ->EC1

Tabelle A11: Literatur strukturierte Literaturanalyse Titel 101-110

No.	Autor(en)	Titel	Inkludiert	Ausschlussbegründung
101	Zhao, Jiantao Sun, Jin	Research on Access Control Model Based on RBAC Model in Microservice Environment		Erforschung der traditionellen Zugangskontrolltechnologie werden die Einschränkungen und Mängel der Zugangskontrolle in der Mikro-Service-Umgebung analysiert und ein Optimierungsmodell der Zugangskontrolle auf der Grundlage von RBAC vorgeschlagen ->EC1
102	ShuLin, Yang JiePing, Hu	Research on Unified Authentication and Authorization in Microservice Architecture	x	
103	Natalino, Carlos Manso, Carlos Vilalta, Ricard Monti, Paolo Munoz, Raul Furdek, Marija	Scalable Physical Layer Security Components for Microservice-Based Optical SDN Controllers		Sicherheitskomponenten für die Sicherheit der physikalischen Schicht in optischen Netzwerken ->EC1
104	Spring Security	Secure your web applications, RESTful services, and microservice architectures		Paywall + Buch ->erfüllt IC2, IC4 nicht
105	Dannberg, Daniel	Security Analysis of Microservice Choices		Abschlussarbeit ->Erfüllt IC4 nicht
106	Esposito, Christian Castiglione, Aniello Tudorica, Constantin-Alexandru Pop, Florin	Security and privacy for cloud-based data management in the health network service chain: a microservice approach		Architektur eines sicheren Managers für die cloudbasierte Verwaltung und den Austausch von Daten im Gesundheitswesen, aber ohne tokenbasierte Authentifizierung und Autorisierung ->EC1
107	Barabanov, Alexander Makrushin, Denis	Security Audit Logging in Microservice-Based Systems: Survey of Architecture Patterns		Thread Model für logging systeme ->EC1
108	Jalonen, Samppa	Security Challenges of Microservices		Bachelorarbeit ->Erfüllt IC4 nicht
109	Epstein, Aaron	Security Concerns for Microservices on the Cloud		Veröffentlichung über Tufts Universität ->Erfüllt IC4 nicht
110	Rudrabhatla, Chaitanya K.	Security Design Patterns in Distributed Microservice Architecture	x	

Tabelle A12: Literatur strukturierte Literaturanalyse Titel 111-120

No.	Autor(en)	Titel	Inkludiert	Ausschlussbegründung
111	Pereira-Vale, Anelis Fernandez, Eduardo B. Monge, Raúl Astudillo, Hernán Márquez, Gastón	Security in microservice-based systems: A multivocal literature review		keine detailierten Informationen zu Authentifizierung oder Autorisierung auf Basis von Token ->EC1, EC2
112	Mateus-Coelho, Nuno Cruz-Cunha, Manuela Ferreira, Luis Gonzaga	Security in microservices architectures		Aufdeckung der wichtigsten Sicherheitsaspekte von Microservices (keine Informationen zu tokenbasierter Authentifizierung oder Autorisierung) ->EC1
113	Pereira-Vale, Anelis; Marquez, Gaston Astudillo, Hernan Fernandez, Eduardo B.	Security mechanisms used in microservices-based systems: A systematic mapping		stellt fest, dass OAuth2 am häufigsten in den analysierten Arbeiten referenziert wird, enthält aber keine Handlungsempfehlungen ->EC2
114	eszko, Pawel	Security of enterprise systems built in microservices architecture		Masterarbeit ->Erfüllt IC4 nicht
115	T. Tenev	Security Patterns for Microservice Data Management		Klassifizierung von Sicherheitsmustern in Microservices aber erwähnt keine tokenbasierten Authentifizierungs- oder Autorisierungstechniken ->EC1
116	Tihomir, Tenev Dimitar, Birov	Security Patterns for Microservices located on different Vendors		Schwachstelenanalyse mit STRIDE von Anwendungen die bei verschiedenen PaaS Anbietern angesiedelt sind + Bereitstellung von Empfehlungen (keine tokenbasierten Authentifizierungs- oder Autorisierungsmethoden) ->EC1
117	Wu, Xiaochun Shen, Yuling Zhang, Junnan	Security Technologies and Research Challenges on Microservice-Based NFV.		NFV-Sicherheit auf der Grundlage von Microservices ->EC1
118	Sun, Yuqiong Nanda, Susanta Jaeger, Trent	Security-as-a-service for microservices-based cloud applications		Design für Security-as-a-Service für Microservices-basierte Cloud-Anwendungen (Überwachung und Durchsetzung von Richtlinien für den Netzwerkverkehr auf, um Cloud-Anwendungen zu sichern) ->EC1
119	Rossi, Fabiana Cardellini, Valeria Lo Presti, Francesco	Self-adaptive Threshold-based Policy for Microservices Elasticity		Skalierung von Microservice-basierten Anwendungen mithilfe von dynamischen, schwellenwertbasierten Richtlinien und Reinforcement-Learning-Ansätzen ->EC1
120	Chatterjee, Ayan Gerdes, Martin W. Khatiwada, Pankaj Prinz, Andreas	SFTSDH: Applying Spring Security Framework With TSD-Based OAuth2 to Protect Microservice Architecture APIs	x	

Tabelle A13: Literatur strukturierte Literaturanalyse Titel 121-130

No.	Autor(en)	Titel	Inkludiert	Ausschlussbegründung
121	Ponce, Francisco Soldani, Jacopo Astudillo, Hernán Brogi, Antonio	Should Microservice Security Smells Stay or be Refactored? Towards a Trade-off Analysis		Trade-Off-Analyse für das Beibehalten oder Refactoren von Securitysmells ->EC1
122	Ponce, Francisco Soldani, Jacopo Astudillo, Hernán Brogi, Antonio	Smells and refactorings for microservices security: a multivocal literature review		Beschäftigt sich mit Smells in Microservices und den korrespondierenden Refactorings ->EC1
123	Billawa, Priyanka Tukaram, Anusha Bamblhore Ferreyra, Nicolás E. Díaz Steghöfer, Jan-Philipp Scandariato, Riccardo Simhandl, Georg	SoK: Security of Microservice Applications: A Practitioners' Perspective on Challenges and Best Practices		LR zum Thema bietet aber keine konkreten Handlungsempfehlungen ->EC2
124	Nautiyal Cantos, Erick Andre	Stateless authentication microservice		Spanisch,Bachelor Thesis ->erfüllt IC1, IC4 nicht
125	Jiang, Yu Li, Chengkai Li, Ying	SUAM: A Service Unified Access Model for Microservice Management		Service Unified Access Model (SUAM) für Standartisierung des einheitlichen Zugriffs auf Microservices ->EC1
126	Luigi Lavazza Roy Oberhauser Radek Koci Stephen Clyde	Survey on microservice architecture-security, privacy and standardization on cloud computing environment		bibliographische Übersicht über die Microservice-Architektur, für Sicherheits-, Datenschutz- und Standartisierungsaspekte in Cloud-Computing-Umgebungen (keine tokenbasierten Authentifizierungs oder Autorisierungstechniken) ->EC1
127	Sauwens, Marijn Heydari Beni, Emad Jaunes, Kristof Lagaisse, Bert Joosen, Wouter	ThunQ: A Distributed and Deep Authorization Middleware for Early and Lazy Policy Enforcement in Microservice Applications		ThunQ vor. eine verteilte Autorisierungs-Middleware, die Autorisierungsrichtlinien sowohl früh an der Fassaden-API durchsetzt, als auch langsam, indem sie Autorisierungsentscheidungen auf den entsprechenden Datenkontext verschiebt ->verwendet aber keine tokenbasierten Mechanismen ->EC1
128	Pancholi, Meghna Kellas, Andreas D. Kemerlis, Vasileios P. Sethumadhavan, Simha	Timeloops: Automatic System Call Policy Learning for Containerized Microservices		Timeloops - eine neuartige Technik zum automatischen Erlernen von Richtlinien zur Filterung von Systemaufrufen für containerisierte Microservices-Anwendungen ->EC1
129	Rangta, Monika	Tools for Security Auditing and Hardening in Microservices Architecture		Masterarbeit ->Erfüllt IC4 nicht
130	Bamblhore Tukaram, Anusha; Schneider, Simon Díaz Ferreyra, Nicolás E. Simhandl, Georg Zdun, Uwe Scandariato, Riccardo	Towards a Security Benchmark for the Architectural Design of Microservice Applications		Erstellung eines Benchmarks für das Design der Microservice-Anwendungen -> zu wenig zu tokenbasierter Authentifizierung und Autorisierung ->EC1

Tabelle A14: Literatur strukturierte Literaturanalyse Titel 131-137

No.	Autor(en)	Titel	Inkludiert	Ausschlussbegründung
131	Li, Xing Chen, Yan Lin, Zhiqiang	Towards Automated Inter-Service Authorization for Microservice Applications		Jarvis als erster automatisierter, serviceübergreifender Autorisierungsmechanismus für Microservice-Anwendungen (kein tokenbasierter Authentifizierungs- und Autorisierungsmechanismus) ->EC1
132	Preuveneers, Davy Joosen, Wouter	Towards Multi-party Policy-based Access Control in Federations of Cloud and Edge Microservices	x	
133	Ponce, Francisco	Towards Resolving Security Smells in Microservice-Based Applications		Erkennung von architektonischen Smells, die mit der Sicherheit in Microservices-basierten Anwendungen verbunden sind ->EC1
134	Pontarolli, Ricardo P. Bigheti, Jeferson A. Sa, Lucas Borges Rodrigues de Godoy, Eduardo P.	Towards Security Mechanisms for an Industrial Microservice-Oriented Architecture	x	
135	Yang, Jiangbing Hou, Haibo Li, Hui Zhu, Qinrong	User Fast Authentication Method Based on Microservices	x	
136	Moskvichev, Anton Dolgachev, Mihail	Using microkernel virtualization means to ensure the security of systems with microservice architecture		kyrillisch ->Erfüllt IC1 nicht
137	Md Shazibul Islam Shamim	Youtube for software security?: Youtube Videos Provide Pointers for Microservice Security		Kein Paper (nur Posterabstract) und keine Informationen zu tokenbasierter Authentifizierung und Autorisierung ->EC1

B Open Source IAM Analyse

Diese Sektion zeigt die durchgeführten Schritte für die Open Source IAM Analyse nach Nickerson et al. (2013).

Schritt 1: Meta-Charakteristik: Features von Open Source basierten IAM-Lösungen.

Schritt 2: Bedingungen für die Beendigung: Die Methode endet, sobald die in 4.2 identifizierten Anforderungen durch die IAM-Lösungen erfüllbar und mindestens fünf verschiedene IAM-Lösungen analysiert sind.

Iteration 1:

Schritt 3: An dieser Stelle wird sich zunächst für den empirisch konzeptuellen Ansatz aus Nickerson et al. (2013) entschieden. Dieser definiert sich dadurch, dass er Objekte beinhaltet, zu denen ein Forscher bereits erste Erkenntnisse gesammelt hat, z. B. durch das Lesen von Fachliteratur (Nickerson et al., 2013). Durch das Lesen der Fachliteratur in Kapitel 3, sind bereits erste IAM-Lösungen identifiziert worden.

Schritt 4e: In der berücksichtigten Literatur finden sich folgende IAM-Lösungen:

- Keycloak (Chatterjee et al., 2022; Chatterjee & Prinz, 2022; Preuveneers & Joosen, 2019)
- OpenAM/Forgerock (Preuveneers & Joosen, 2017)

Schritt 5e: Dabei werden die folgenden Features nach Analyse der IAM-Lösungen identifiziert:

- Single Sign-On
- Cross Domain Single Sign-On
- Multi Faktor Authentifizierung
- Social Log-in
- OpenID Connect
- OAuth2.0
- SAML
- Admin Console
- Account Management Selfservice für Nutzer

- ABAC
- RBAC
- Policy Based Access Control
- Timebased Access Control
- Support for custom access control mechanisms
- Docker

Schritt 6e: Aus den Features entstehen nach der Gruppierung folgende Kategorien:

- Loginfeatures
- unterstütze Protokolle zur Authentifizierung / Autorisierung
- Account Management Features
- Zugriffskontrollmechanismen
- Deploymentmöglichkeiten

Tabelle B1 zeigt die entstandene Klassifikation der angebotenen Features nach dieser Iteration.

Schritt 7: Prüfen der Endbedingungen: Da durch die erste Iteration noch keine fünf IAM-Lösungen analysiert wurden, sind die Endbedingungen noch nicht erfüllt.

Iteration 2:

Schritt 3: An dieser Stelle wird sich erneut für den empirisch konzeptuellen Ansatz aus Nickerson et al. (2013) entschieden. Durch die Recherche im Rahmen dieser Arbeit und den Austausch mit den Entwicklern von JValue werden weitere IAM-Lösungen gefunden.

Schritt 4e: Folgende weitere Lösungen wurden identifiziert:

- Casdoor
- Zitadel
- Ory

Schritt 5e: Dabei werden folgende zusätzliche Features durch die Analyse der IAM-Lösungen identifiziert:

- One Time Passwords
- WebAuthn

- CTAP
- FIDO2
- Client Authentication via Signed JWT RFC 7523
- Eigene Berechtigungssprache
- Kubernetes
- Helm-Charts

Schritt 6e: Aus den neu hinzugekommen Features ergeben sich nach erneuter Gruppierung folgende Kategorien:

- Loginfeatures
- unterstütze Protokolle zur Authentifizierung / Autorisierung
- Account Management Features
- Zugriffskontrollmechanismen
- Deploymentmöglichkeiten

Tabelle 4.1 zeigt die entstandene Klassifikation der angebotenen Features nach dieser Iteration.

Schritt 7: Prüfen der Endbedingungen: Die Bedingungen aus Schritt 2 sind erfüllt. Die Anforderungen aus Sektion 4.2 können mit den analysierten Lösungen umgesetzt werden. Zudem ist die zweite Bedingung, dass fünf IAM-Lösungen analysiert werden sollen, ebenfalls erfüllt.

Kategorie	**Feature**	**Keycloak**	**OpenAM / Forgerock**
Loginfeatures	SSO	+	+
	CDSSO	-	+
	MFA	+	+
	Social Login	+	+
Protokolle / Standards	OpenIDConnect	+	+
	OAuth2.0	+	+
	SAML	+	+
Usability und Customizability	Admin Console	+	+
	Account Management SelfService for Users	+	+
Access Control Mechanismen	ABAC	+	+
	RBAC	+	+
	Timebased Access control	+	-
	Support for custom access control mechanisms	+	-
Deploymentmöglichkeiten	Docker	+	+

Tabelle B1: IAM Features nach Iteration 1

Literaturverzeichnis

Abdelfattah, A. S., & Cerny, T. (2022). *Microservices Security Challenges and Approaches*. https://aisel.aisnet.org/isd2014/proceedings2022/currenttopics/7/

Banati, A., Kail, E., Karoczkai, K., & Kozlovszky, M. (2018). Authentication and authorization orchestrator for microservice-based software architectures. *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 1180–1184. https://doi.org/10.23919/MIPRO.2018.8400214

Barabanov, A., & Makrushin, D. (2020). Authentication and authorization in microservice-based systems: survey of architecture patterns. https://doi.org/10.48550/arXiv.2009.02114

Bazeniuc, I., & Zgureanu, A. (2021). Information security in microservices architectures. *Electronics, Communications and Computing*, 206–209.

Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative Research in Psychology*, *3*(2), 77–101. https://doi.org/10.1191/1478088706qp063oa

Chandramouli, R., Butcher, Z., & Chetal, A. (2021). *Attribute-based Access Control for Microservices-based Applications Using a Service Mesh*. National Institute of Standards and Technology. https://doi.org/10.6028/nist.sp.800-204b

Chatterjee, A., Gerdes, M. W., Khatiwada, P., & Prinz, A. (2022). SFTSDH: Applying Spring Security Framework With TSD-Based OAuth2 to Protect Microservice Architecture APIs. *IEEE Access*, *10*, 41914–41934. https://doi.org/10.1109/ACCESS.2022.3165548

Chatterjee, A., & Prinz, A. (2022). Applying Spring Security Framework with KeyCloak-Based OAuth2 to Protect Microservice Architecture APIs: A Case Study. *Sensors*, *22*(5). https://doi.org/10.3390/s22051703

Chiarelli, A. (2021). ID Token and Access Token: What's the Difference? Learn what ID and access tokens are and how to correctly use them in the OpenID Connect and OAuth context. Verfügbar 6. August 2023 unter https://auth0.com/blog/id-token-access-token-what-is-the-difference/

Christian Lüdemann. (2019). Implicit Flow Vs. Code Flow With PKCE. Verfügbar 8. Juli 2023 unter https://christianlydemann.com/implicit-flow-vs-code-flow-with-pkce

Dakshitha Ratnayake. (2023). Secure Logins and Resource Access with ZITADEL and OpenID Connect - Part 2 (CAOS Ltd., Hrsg.). Verfügbar 8. Juli 2023 unter https://zitadel.com/blog/secure-logins-with-zitadel-part-2

Dan Abramov. (2023). Redux Essentials, Part 1: Redux Overview and Concepts. Verfügbar 8. Juli 2023 unter https://redux.js.org/tutorials/essentials/part-1-overview-concepts

de Almeida, M. G., & Canedo, E. D. (2022). Authentication and authorization in microservices architecture: A systematic literature review. *Applied Sciences*, *12*(6), 3023.

Dukes, C. (2015). Committee on National Security Systems (CNSS) Glossary (Committee on National Security Systems, Hrsg.). Verfügbar 16. Juni 2023 unter https://rmf.org/wp-content/uploads/2017/10/CNSSI-4009.pdf

Ethelbert, O., Moghaddam, F. F., Wieder, P., & Yahyapour, R. (2017). A JSON Token-Based Authentication and Access Management Schema for Cloud SaaS Applications. *2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*, 47–53. https://doi.org/10.1109/FiCloud.2017.29

He, X., & Yang, X. (2017). Authentication and Authorization of End User in Microservice Architecture. *Journal of Physics: Conference Series*, *910*, 012060. https://doi.org/10.1088/1742-6596/910/1/012060

Istio. (n. d.). Verfügbar 29. Juli 2023 unter https://istio.io/latest/docs/ops/deployment/architecture/

Jander, K., Braubach, L., & Pokahr, A. (2018). Defense-in-depth and role authentication for microservice systems. *Procedia computer science*, *130*, 456–463.

Jordan, C. (1987). *A Guide to Understanding Discretionary Access Control in Trusted Systems* (Bd. 3). National Computer Security Center. https://apps.dtic.mil/sti/pdfs/ADA392813.pdf

Junker, A. (2020). Lösungsmuster für Cross-Cutting Concerns in Microservices (Alkmene Verlag GmbH, Hrsg.). Verfügbar 2. Juni 2023 unter https://www.informatik-aktuell.de/entwicklung/methoden/loesungsmuster-fuer-cross-cutting-concerns-in-microservices.html

JValue Core Developer. (n. d.). Hub. Verfügbar 11. August 2023 unter https://github.com/jvalue/hub

Keycloak. (n. d.). Securing Applications and Services Guide. Verfügbar 7. Juni 2023 unter https://www.keycloak.org/docs/latest/securing_apps/

Kitchenham, B. (2004). Procedures for performing systematic reviews. *Keele, UK, Keele University*, *33*(2004), 1–26.

Kölpin, L. (2022). Der OAuth2 PKCE-Flow. Verfügbar 6. August 2023 unter https://larskoelpin.de/2022-09-11-oauth-pkce/

Lodderstedt, T., Dronia, S., & Scurtescu, M. (2013). OAuth 2.0 Token Revocation. Verfügbar 16. Mai 2023 unter https://www.rfc-editor.org/rfc/rfc7009

Luber, S. (2022). Was ist JSON Web Token (JWT)? Verfügbar 8. Juni 2023 unter https://www.security-insider.de/was-ist-json-web-token-jwt-a-1094265/

Max Peintner. (2023a). Instance Settings (CAOS Ltd., Hrsg.). Verfügbar 8. Juli 2023 unter https://zitadel.com/docs/guides/manage/console/instance-settings#oidc-token-lifetimes-and-expiration

Max Peintner. (2023b). Managers (CAOS Ltd., Hrsg.). Verfügbar 8. Juli 2023 unter https://zitadel.com/docs/guides/manage/console/managers

Max Peintner. (2023c). OpenID Connect Endpoints (CAOS Ltd., Hrsg.). Verfügbar 8. Juli 2023 unter https://zitadel.com/docs/apis/openidoauth/endpoints

Max Peintner. (2023d). Organizations (CAOS Ltd., Hrsg.). Verfügbar 8. Juli 2023 unter https://zitadel.com/docs/guides/manage/console/organizations

Max Peintner. (2023e). Private Key JWT (CAOS Ltd., Hrsg.). Verfügbar 8. Juli 2023 unter https://zitadel.com/docs/guides/integrate/private-key-jwt

Max Peintner. (2023f). Projects (CAOS Ltd., Hrsg.). Verfügbar 8. Juli 2023 unter https://zitadel.com/docs/guides/manage/console/projects

Max Peintner. (2023g). Quick start guide (CAOS Ltd., Hrsg.). Verfügbar 8. Juli 2023 unter https://zitadel.com/docs/guides/start/quickstart

Max Peintner. (2023h). What is an application? (CAOS Ltd., Hrsg.). Verfügbar 8. Juli 2023 unter https://zitadel.com/docs/guides/manage/console/applications

Nasab, A. R., Shahin, M., Raviz, S. A. H., Liang, P., Mashmool, A., & Lenarduzzi, V. (2021). An Empirical Study of Security Practices for Microservices Systems. *arXiv e-prints*, arXiv:2112.14927.

Nasab, A. R., Shahin, M., Raviz, S. A. H., Liang, P., Mashmool, A., & Lenarduzzi, V. (2022). *An Empirical Study of Security Practices for Microservices Systems*. arXiv. https://doi.org/10.48550/arXiv.2112.14927

Nehme, A., Jesus, V., Mahbub, K., & Abdallah, A. (2019). Fine-Grained Access Control for Microservices. In N. Zincir-Heywood, G. Bonfante, M. Debbabi & J. Garcia-Alfaro (Hrsg.), *Foundations and Practice of Security* (S. 285–300). Springer International Publishing.

NestJS (Hrsg.). (n. d.). Verfügbar 8. Juli 2023 unter https://docs.nestjs.com/guards

Nguyen, Q., & Baker, O. F. (2019). Applying Spring Security Framework and OAuth2 To Protect Microservice Architecture API. *J. Softw.*, *14*(6), 257–264.

Nickerson, R. C., Varshney, U., & Muntermann, J. (2013). A method for taxonomy development and its application in information systems. *European Journal of Information Systems*, *22*(3), 336–359. https://doi.org/10.1057/ejis.2012.26

Okta. (2023a). express-jwt. Verfügbar 15. Mai 2023 unter https://github.com/auth0/express-jwt

Okta. (2023b). Vergleich von Authentifizierung und Autorisierung. Verfügbar 7. Mai 2023 unter https://www.okta.com/de/identity-101/authentication-vs-authorization/

Okta.Inc. (n. d.). https://auth0.com/docs/get-started/authentication-and-authorization-flow/authorization-code-flow-with-proof-key-for-code-exchange-pkce

OneLogin. (n. d. a). https://developers.onelogin.com/openid-connect/guides/auth-flow-pkce

OneLogin. (n. d. b). RBAC vs ABAC: Make the Right Call. Verfügbar 8. Juli 2023 unter https://www.onelogin.com/learn/rbac-vs-abac

Open Policy Agent contributors. (n. d.). Open Policy Agent - Introduction. Verfügbar 17. Juni 2023 unter https://www.openpolicyagent.org/docs/latest/

Pasomsup, C., & Limpiyakorn, Y. (2021). HT-RBAC: A Design of Role-based Access Control Model for Microservice Security Manager. *2021 International Conference on Big Data Engineering and Education (BDEE)*. https://doi.org/10.1109/bdee52938.2021.00038

Pereira-Vale, A., Márquez, G., Astudillo, H., & Fernandez, E. B. (2019). Security mechanisms used in microservices-based systems: A systematic mapping. *2019 XLV Latin American Computing Conference (CLEI)*, 01–10.

Pontarolli, R. P., Bigheti, J. A., de Sá, L. B. R., & Godoy, E. P. (2021). Towards Security Mechanisms for an Industrial Microservice-Oriented Architecture. *2021 14th IEEE International Conference on Industry Applications (INDUSCON)*, 679–685.

Preuveneers, D., & Joosen, W. (2017). Access Control with Delegated Authorization Policy Evaluation for Data-Driven Microservice Workflows. *Future Internet*, *9*(4), 58. https://doi.org/10.3390/fi9040058

Preuveneers, D., & Joosen, W. (2019). Towards Multi-party Policy-based Access Control in Federations of Cloud and Edge Microservices. *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. https://doi.org/10.1109/eurospw.2019.00010

Redis Ltd. (2023). Redis. Verfügbar 16. Mai 2023 unter https://redis.io/

Rudrabhatla, C. K. (2020). Security Design Patterns in Distributed Microservice Architecture. https://doi.org/10.48550/arXiv.2008.03395

Safaryan, O., Pinevich, E., Roshchina, E., Cherckesova, L., & Kolennikova, N. (2020). Information system development for restricting access to software tool built on microservice architecture. *E3S Web of Conferences*, *224*, 01041. https://doi.org/10.1051/e3sconf/202022401041

Salibindla, J. (2018). Microservices API Security. *International Journal of Engineering and Technical Research*, *V7*(01). https://doi.org/10.17577/IJERTV7IS010137

Samarati, P., & de Vimercati, S. C. (2001). Access Control: Policies, Models, and Mechanisms (R. Focardi & R. Gorrieri, Hrsg.), 137–196.

ShuLin, Y., & JiePing, H. (2020). Research on Unified Authentication and Authorization in Microservice Architecture. *2020 IEEE 20th International Conference on Communication Technology (ICCT)*, 1169–1173. https://doi.org/10.1109/ICCT50939.2020.9295931

Universität Hamburg. (2016). Recherchetipp: 3 Tipps für Google Scholar: Datenbanken und Fachportale. Verfügbar 5. August 2023 unter https://www.wiso.uni-hamburg.de/bibliothek/ueber-die-bibliothek/neues-aus-der-bibliothek/recherchetipps/nachricht16-039-recherchetipp09.html

Yan, K., Pan, Y., Sui, Y., & Ye, S. (2022). Design and Application of Security Gateway for Transmission Line Panoramic Monitoring Platform based on Microservice Architecture. *2022 IEEE 6th Information Technology and Mechatronics Engineering Conference (ITOEC)*. https://doi.org/10.1109/itoec53115.2022.9734463

Yang, J., Hou, H., Li, H., & Zhu, Q. (2021). User Fast Authentication Method Based on Microservices. *2021 IEEE International Conference on Power Electronics, Computer Applications (ICPECA)*, 93–98. https://doi.org/10.1109/ICPECA51329.2021.9362656

Yarygina, T., & Bagge, A. H. (2018). Overcoming Security Challenges in Microservice Architectures. *2018 IEEE Symposium on Service-Oriented System Engineering (SOSE)*. https://doi.org/10.1109/sose.2018.00011

Yu, D., Jin, Y., Zhang, Y., & Zheng, X. (2019). A survey on security issues in services communication of Microservices-enabled fog applications. *Concurrency and Computation: Practice and Experience*, *31*(22), e4436.