**­­**

**T2000: Umsetzung von Multimandantenfähigkeit in der SCIM-Provisioning-Engine**

**Name der Hochschule:** Duale Hochschule Baden-Württemberg Stuttgart

**Studiengang:** Informatik

**Matrikelnummer:** 7060837

**Kurs:** TINF22F

**Abgabedatum:** 09.09.2024

**Betreuer:** Mathias Baumert

***Erklärung zur Eigenleistung***

Ich versichere hiermit, dass ich meine Projektarbeit mit dem Thema: Integration von Multimandantenfähigkeit der SCIM-Provisioning-Engine selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

--------------------------------------- Ort, Datum ----------------------------------- Unterschrift

***Sonstige Anmerkungen:*** *Als Zitierstil wurde IEEE verwendet.*

**Abstract**

The focus of this work is to extend the SCIM-Provisioning-Engine (abbreviation: SPE) application to multi-client capability. The purpose of the application is to provision user data from the identity and access management application Keycloak. Research was carried out into the technologies used and solution approaches were developed. The solution approaches were evaluated and one of three solution approaches was implemented. It was found that the SCIM provisioning engine is multi-client capable.

**Kurzfassung**

Der Schwerpunkt dieser Arbeit besteht darin, die Anwendung SCIM-Provisioning-Engine *(Abk.: SPE)* auf Multimandantenfähigkeit zu erweitern. Zweck der SPE ist es, Benutzerdaten aus der Identity -und Access Management Anwendung Keycloak zu provisionieren. Es wurde zu den angewandten Technologien recherchiert und Lösungsansätze erstellt. Die Lösungsansätze wurden bewertet und eine von drei Lösungsansätzen wurde umgesetzt. Es hat sich ergeben, dass die SPE multimandantenfähig ist.

**Abkürzungsverzeichnis**

**SPE:** Scim-Provisioning-Engine

**SSO:** Single-Sign on

**AWS:** Amazon Web Services

**IAM:** Identity – and Access Management

**HTTP:** Hypertext Transfer Protocol

**ReST:** Representational State Transfer

**API:** Application Programming Interface

**SCIM:** System for Cross-domain Identity Management

**SaaS:** Software as a Service

**IdP:** Identity Provider

**JSON:** JavaScript Object Notation

**JWT:** JSON Web Token

**OIDC:** OpenID Connect

**IoC:** Inversion of Control

**ORDMS:** objektrelationales Datenbank-Management-System

**CI:** Continuous Integration

**CD:** Continuous Delivery

**SLF4J:** Simple Logging Facade for Java

Inhalt

[1. Einleitung 1](#_Toc170488853)

[1.1 Die Scim-Provisioning-Engine 1](#_Toc170488854)

[1.2 Problemfeststellung 2](#_Toc170488855)

[1.3 Ziel dieser Arbeit 3](#_Toc170488856)

[2. Angewandte Methoden 3](#_Toc170488857)

[3. Technologien 5](#_Toc170488858)

[3.1 Keycloak 5](#_Toc170488859)

[3.2 Representational State Transfer (kurz: REST) 9](#_Toc170488860)

[3.3 System for Cross-domain Identity Management (kurz: SCIM) 10](#_Toc170488861)

[3.4 OAuth2 und OpenID Connect 12](#_Toc170488862)

[3.5 Spring und Springboot 16](#_Toc170488863)

[3.6 Springboot Security 19](#_Toc170488864)

[3.7 PostgreSQL 19](#_Toc170488865)

[3.8 Jira 20](#_Toc170488866)

[3.9 GitLab 20](#_Toc170488867)

[3.10 Kubernetes und Docker 21](#_Toc170488868)

[4. Stand der SPE vor dieser Projektarbeit 24](#_Toc170488869)

[4.1 Abhängigkeiten und Bezug zu Docker 24](#_Toc170488870)

[4.2 Authentifizierung und Autorisierung in der SPE 26](#_Toc170488871)

[4.3 Scheduler in der SPE 29](#_Toc170488872)

[4.4 ERM und Datenbankberechtigungen 32](#_Toc170488873)

[4.5 CI/CD der SPE 34](#_Toc170488874)

[5. Lösungsansätze 35](#_Toc170488875)

[6. Umsetzung der Lösung 38](#_Toc170488876)

[6.1 Die Super-Admin-Role 39](#_Toc170488877)

[6.2 Der Customer-Admin-Role 43](#_Toc170488878)

[6.3 Der Technische Admin 45](#_Toc170488879)

[7. Zusammenfassung 46](#_Toc170488880)

[7.1 Ausblick 46](#_Toc170488881)

[7.2 Bewertung der Lösung 46](#_Toc170488882)

[Literaturverzeichnis 47](#_Toc170488883)

[Abbildungsverzeichnis (muss noch genau gemacht werden) 51](#_Toc170488884)

# 1. Einleitung

## 1.1 Die Scim-Provisioning-Engine

In heutiger Zeit gibt es viele Softwarelösungen für verschiedene Zwecke. Für fast jede Tätigkeit existiert schon eine Lösung, wie bspw. ein Programm, um Termine einzutragen, ein Programm, um Software selbst zu verwalten oder ein Programm, welches Dokumente verwaltet.

Für viele Dienste, die miteinander zusammenhängen in einem Unternehmen, ist es notwendig, dieselben Benutzerinformationen in diesen Diensten auf den gleichen Informationsstand zu bringen.

Wenn bspw. der Benutzer[[1]](#footnote-2) “Chell” in der Anwendung “P” aktualisiert wird, so muss das in der Anwendung “H” auch geschehen, wenn auch dort sich derselbe Benutzer “Chell” befindet. Um dies zu ermöglichen, wurde im Unternehmen Intension die Scim-Provisioning-Engine *(kurz: SPE)* entwickelt.

Die SPE ist eine containerbasierte Anwendung, welche es ermöglicht, automatisch Benutzerinformationen an Zielanwendungen über den Standardprotokoll SCIM zu provisionieren. Die Benutzerinformationen werden dabei aus Keycloak bezogen. Zu den Benutzerinformationen gehören ebenso Informationen darüber, zu welcher Anwendung der Benutzer gehört und auch zu welcher Gruppe in der SPE.

Keycloak ist eine Open-Source-Lösung für das Identitäts- und Zugangsmanagement, die das Single-Sign on *(kurz: SSO)* ermöglicht [[1]](https://www.keycloak.org/).

IAM ist ein Konzept, dass sich über Jahrzehnte hinweg entwickelt hat, mit dem Ziel herauszufinden, wie Zugriffsrechte und Identitätsdaten sicher und geschützt verwaltet werden sollen und wie man diese Verwaltung verbessern kann.

Die Verwaltung umfasst Benutzeridentitäten, Authentifizierung, Autorisierung, Berechtigungsvergabe, sowie die Überwachung und Prüfung von Zugriffen auf Informationen und Systeme.

SSO *“ ist ein Authentifizierungsschema, das es Benutzern ermöglicht, sich einmal mit einem einzigen Satz von Anmeldedaten anzumelden und während derselben Sitzung auf mehrere Anwendungen zuzugreifen”* [[2]](https://ibm.com/de-de/topics/single-sign-on%22%3ehttps://ibm.com/de-de/topics/single-sign-on)*.*

Wenn man sich bspw. mit dem Benutzer “Freeman” anmeldet bei der Anwendung “Half-Life” und dann die Anwendung “Portal” startet, wird bei letzterer Anwendung keine Anmeldung verlangt. Der Benutzer wird automatisch durch das Single-Sign on angemeldet. Das Single-Sign out erfüllt trivialerweise dieselbe Funktion, nur dass “Freeman” abgemeldet und nicht angemeldet wird.

Alle Operationen, die mit Benutzerdaten aus Keycloak durchgeführt werden, wie z.B. zu erstellen, zu aktualisieren oder zu löschen, werden mit den jeweiligen Zielsystemen synchronisiert. Zu den Zielanwendungen gehören unter anderem Amazon Web Services *(kurz: AWS)*, Gitlab und Atlassian, die im Hauptteil genauer erläutert werden.

Die Anwendung ist größtenteils in der Programmiersprache Java programmiert.

## 1.2 Problemfeststellung

Bisher gibt es in der SPE keine Multimandantenfähigkeit.

Laut Chong [[3]](https://renatoargh.wordpress.com/wp-content/uploads/2018/01/article-multi-tenant-data-architecture-2006.pdf) sei Multimandantenfähigkeit, dass eine einzige Software mehrere Kunden bedienen kann. Damit sind auch keine getrennten Datenbanken nötig für jede neue Software.

Ohne Multimandantenfähigkeit in einer Software ergeben sich folgende Probleme:

Wenn jeder Kunde eine neue Instanz des Produktes installiert, wird die Bereitstellung der Software aufwändiger und dadurch werden auch die Kosten höher, weil dann für jede Software eine neue Datenbank oder neue Softwareabhängigkeiten benötigt werden.

Ein weiterer Nachteil ist der, dass die Unternehmensdaten in einem Einzelmandantensystem nicht getrennt von anderen Unternehmensdaten sind und diese sichtbar sind für andere Unternehmen.

Multimandantenfähigkeit verhindert diese Probleme, wodurch die Kosten reduziert werden und Unternehmensdaten getrennt werden.

## 1.3 Ziel dieser Arbeit

Das Ziel dieser Arbeit ist es zu überprüfen, ob im Backend der SPE ein multimand-antenfähiges Berechtigungskonzept entwickelbar ist.

Dabei wird geplant, wie man die Multimandantenfähigkeit im Projekt umsetzt. Dafür werden verschiedene Lösungsansätze erkundet und bewertet.

Weitere Ziele sind die Einrichtung automatisierter Test-, Versionierungs- und Deploymentprozesse für die Demoumgebung.

Es werden bewusst Methoden gewählt, welche zielführend überprüfen, ob die SPE multimandantenfähig ist.

Ein weiteres, indirektes Ziel ist es zu lernen, wie man in der Informatik wissenschaftlich arbeitet.

# 2. Angewandte Methoden

Es ist wichtig sich mit den Technologien in der SPE auszukennen, bevor man die Forschungsfrage beantworten will. Die erste eingesetzte Methode ist daher Recherchearbeit auf Basis der *Grounded Theory*.

Die *Grounded Theory*  “ist eine allgemeine Methode der vergleichenden Analyse”[[4, S. 1]](http://www.sxf.uevora.pt/wp-content/uploads/2013/03/Glaser_1967.pdf). In der *Grounded Theory* werden die Daten als *Memo* dokumentiert und man sammelt vom Anfang bis zum Ende des Forschungsprojektes qualitative Daten.

Zu den qualitativen Daten zählen in der *Grounded Theory* jegliche Daten, wie bspw. auch Videos und Diagramme [[5]](https://dl.acm.org/doi/10.1145/2884781.2884833). Ebenso zählen Interviews mit Experten, die sich mit der Thematik auskennen, die man erforschen will. Die *Grounded Theor*y ist eine qualitative Forschungsmethode, mit welcher Daten gesammelt werden.

Wenn man die *Grounded Theory* anwendet, werden von Anfang bis zum Ende der Entwicklung Informationen über die SPE und dessen angewandten Technologien gesammelt.

Die qualitative Recherche in dieser Methode wird oft als “unsystematisch” und “explorativ” bezeichnet [[4, S.223]](http://www.sxf.uevora.pt/wp-content/uploads/2013/03/Glaser_1967.pdf).

Man hat sich dennoch für diese Methode entschieden, weil sie gut anwendbar auf ein bereits bestehendes Projekt ist. Die SPE ist in Komponenten aufteilbar und durch W-Fragen wird nachgefragt, welche Komponente welche Funktion hat.

Ein weiterer Grund, dass man sich für die *Grounded Theory* entschied, war, weil bzgl. den Technologien wie bspw. Keycloak zunächst recherchiert wurde. Die Recherche nimmt einen Großteil der Projektarbeit ein.

Um zu erkunden, wie die Technologien in der SPE funktionieren werden Tutorials und technische Dokumentationen größtenteils verwendet. Es werden Entwickler befragt, welche an der Entwicklung der SPE beteiligt waren, inwiefern die Anwendung funktioniert.

Eine weitere, angewandte Methode ist es, das Verhalten der SPE zu beobachten. Unter Beobachtung versteht man hier, dass man nach der Projektrecherche herausfindet, wie einzelne Software-Komponenten in der SPE funktionieren.

Da die SPE Benutzerinformationen automatisch provisioniert, wird nach dem Starten der Anwendung im Terminal das Verhalten der Software beobachtet. Im Terminal erscheinen, während der automatischen Provisionierung Meldungen darüber, was in der SPE geschieht, bspw. Informationen darüber, welcher Benutzer im Moment provisioniert wurde. Diese Informationen helfen dabei, mehr zu erfahren, wie die SPE funktioniert.

Während der Projektarbeit wird die Methode weiterhin angewandt. Nach der Programmierung einer Software-Komponente wird beobachtet, ob sich die SPE wie gewünscht verhält.

Falls nicht, werden Veränderungen vorgenommen. Wenn die SPE sich wie gewünscht verhält, wird die nächste Software-Komponente entwickelt.

Eine weitere, verwendete Methode ist das Testen. Hier wurden nur Unittests verwendet; andere Testarten wie z.B. Systemtests und Integrationstest wurden nicht angewandt.

# 3. Technologien

## 3.1 Keycloak

Keycloak wird für das Identity –und Access Management *(kurz: IAM)* als Open-Source Lösung verwendet, um bestimmte Daten zu speichern und verfügbar zu machen.

Laut der Keycloak-Dokumentation gibt es Terminologien, damit unklare Bezeichnungen in der Anwendung definiert werden:

*Realms* verwalten *Benutzer, Berechtigungsnachweise*, Rollen und Gruppen. Die *Realms* sind voneinander getrennt und unabhängig, wodurch User aus verschiedenen Realms aufeinander keinen Zugriff auf andere User haben. Eine Ausnahme bildet der master-realm in dem auch Zugriffe auf andere Realms möglich sind.

Laut Thorgersen [[6]](https://www.google.de/books/edition/Keycloak_Identity_and_Access_Management/WBsvEAAAQBAJ?hl=de&gbpv=0) ist dies nützlich, wenn man zwei Anwendungen voneinander trennen will, um bspw. externe Anwendungen und interne Anwendungen eines Unternehmens.

Wenn man Keycloak installiert und startet, ist ein *Realm* namens *“master-realm”* schon angelegt, welchen man nicht löschen kann. Im *master-realm* ist der *Admin* angelegt. Der *Admin* ist der Benutzer, welcher andere *Realms* verwalten kann und somit Benutzer und Clients verwaltet.

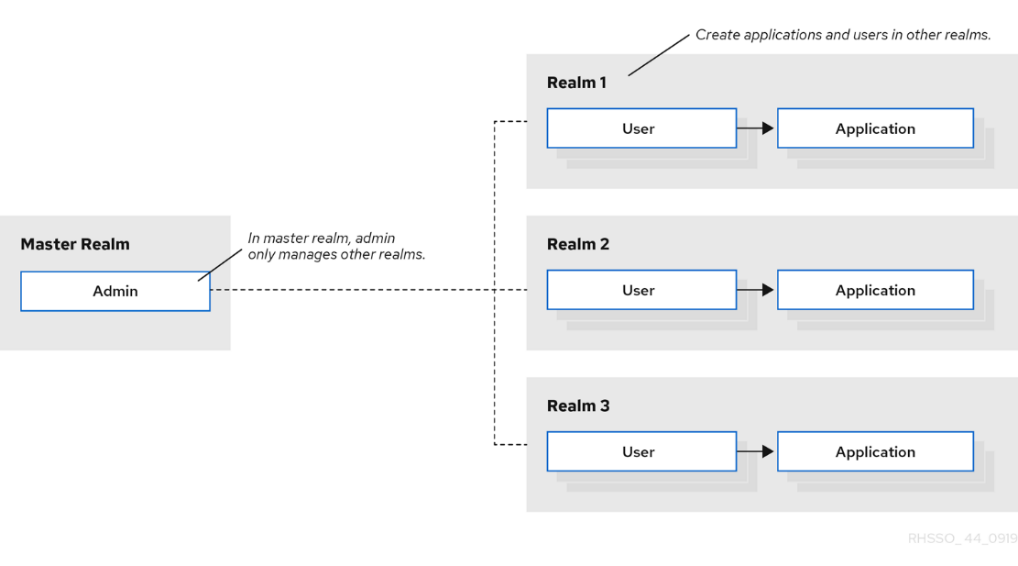


Abbildung veranschaulicht, was der Admin im Master Realm für eine Funktion hat (Quelle: „Server Administration“, [Server Administration Guide (keycloak.org)](https://www.keycloak.org/docs/latest/server_admin/index.html), Zugriffsdatum: Juni 2024)

Ein *Client* stellt eine Anwendung dar, die von Keycloak Anmeldeinformationen für Benutzer des jeweiligen Clients anfordern kann*[[2]](#footnote-3).* Ein Client ist bspw. eine Webseite, ein Software-Managementtool oder Verwaltungssysteme.

Ein *User* stellt einen Benutzer dar, welcher sich in Keycloak bei einer Anwendung, wie bspw. GitLab, anmelden kann. Ein *User* hat Attribute, die ihn kennzeichnen, welche man verändern kann, wie z.B. der Name oder die E-Mail des Benutzers.

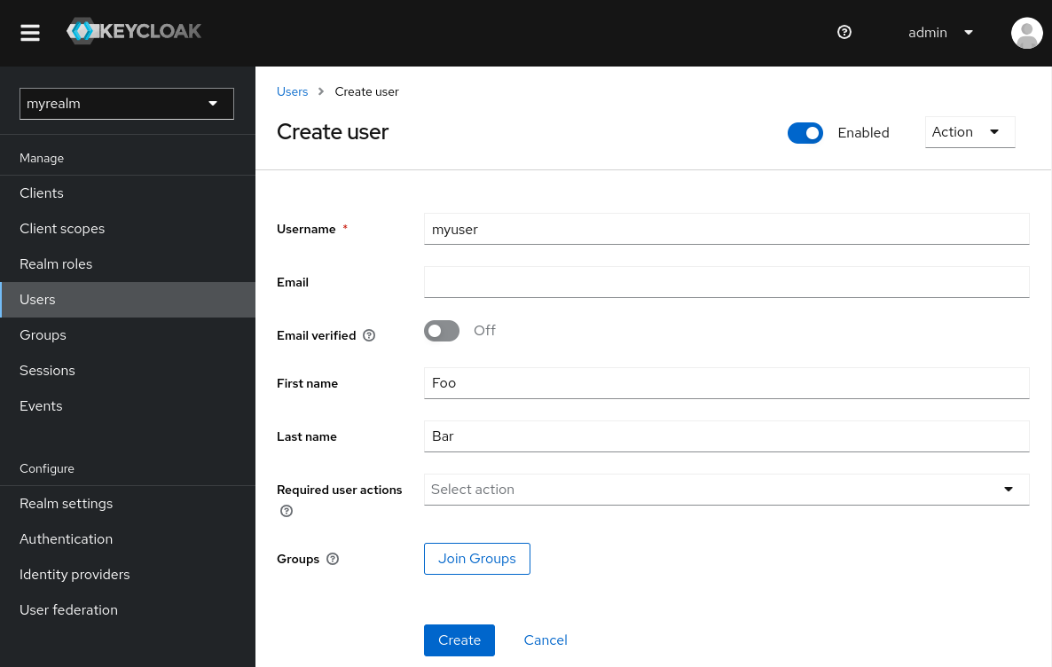


Abbildung : Neuen User in Keycloak erstellen (Quelle: „Keycloak“, [OpenJDK - Keycloak](https://www.keycloak.org/getting-started/getting-started-zip), Zugriffsdatum: Juni 2024)

Eine Rolle *(Role)* Identifiziert einen Typ oder eine Kategorie eines Users oder einer Gruppe. Rollen sind wichtig in der Autorisierung, weil nicht jeder Benutzer auf jeden Zugriff auf eine Ressource berechtigt ist. Wenn bspw. der fiktive Benutzer “Wheatley” nicht die Rolle des Entwicklungsleiters hat, aber dennoch Zugriffe auf alle Entwicklerdaten hat, wäre er auch in der Lage alle diese Entwickler aus Keycloak zu löschen, obwohl er nicht das Recht dazu haben dürfte.

Es werden dabei *Realm Roles, Client Roles* und *Composite Roles* unterschieden:

Realm Roles sind Rollen, welche an einen realm zugeordnet sind. Diese Rollen sind im ganzen Realm für alle Clients verfügbar. Wenn also der Benutzer “Ezio” im Realm “Welt” für alle Clients dieselbe Rolle hat, handelt es sich um eine Realm Role.

Client Roles sind Rollen, die einem Client zugeordnet sind. Wenn bspw. der Benutzer “L” Benutzer aus einem Client namens “Lidl” aktualisieren will, braucht er dafür die Client Role “manage-users”. Die Client roles beziehen sich darauf, welche Berechtigungen ein Benutzer in einem Client hat.

Jede Client-Role ist einem Realm zugeordnet. Dies bedeutet, dass bspw. der Benutzer “L” die Benutzer aus dem Client “Lidl” aktualisieren kann, wobei sich “Lidl” und die Benutzer im Realm “Deutschland” befinden.

Im master-realm sind in diesem Fall mehrere Client Roles verfügbar, welche zu unterschiedlichen Realms angehören. Bspw. existiert im master-realm die Client Role “manage-users” einmal für den Realm “Sega” und einmal für den Realm “nintendo”, wenn diese beiden Realms angelegt sind.

Es ist vorteilhaft, wenn die *Client Roles* einem Realm zugeordnet sind, weil dadurch eine Trennung zwischen den Realms erst entsteht. Die Wichtigkeit dieser Trennung wird in Kapitel 5 klarer.

*Composite roles* sind Rollen, die mit anderen Rollen assoziiert sind. Eine Composite Role beinhaltet damit mehrere Rollen, die Keycloak bereitstellt. Die Composite Role “chef” bspw. kann die Rollen “manage-users” und “view-users” beinhalten. So durch kann ein Benutzer nur eine Rolle haben, die multiple Rollen hat. Dies spart Zeit, weil Zuordnungen mehrerer Rollen in Keycloak Zeit bedarf.

Der Term *client scopes* steht für dasKonzept, in einen Token noch zusätzliche Attribute hinzuzufügen. Der genaue Begriff für Tokens wird on 2.2.4 erläutert. Wenn bspw. Ein Client noch ein zusätzliches Attribut haben soll, wie z.B. eine Benutzeranzahl, wird in *client\_scope* dieses Attribut mit seinem Wert, angelegt im Token.

Eine ähnliche Funktion erfüllt das *user\_mapping*, bei der es ebenso möglich ist im Token neue Bereiche, bzw. Attribute anzulegen. Der Unterschied ist, dass diese Attribute dem Benutzer zugeordnet sind und nicht dem Client. Wenn bspw. ein Benutzer namens “Rick” noch das zusätzliche Attribut “Rolle” haben soll mit dem Wert “Roll”, wird dies durchs *user mapping* konfiguriert.

Beim *service account* hat jeder *Client* ein eingebautes Dienstkonto, über das er automatisch ein Zugangstoken erhalten kann. Der Client muss sich also nicht anmelden, sondern wird durch den *service account* automatisch authentifiziert und autorisiert.

Beim *direct grant* ist es möglich für einen *User*, ein Zugriffstoken im Namen eines Benutzers über einen REST-Aufruf zu erhalten. *Direct Grant* wird in Kapitel 2.2.4 genauer erläutert.

Eine *session* ist eine Sitzung, die entsteht, wenn man sich als ein User in Keycloak anmeldet. Eine Sitzung enthält Informationen, z. B. darüber wann sich der Benutzer angemeldet hat und welche Anwendungen während dieser Sitzung am Single-Sign-On teilgenommen haben. Gespeicherte Sitzungsinformationen sind z.B. der Zeitpunkt, an dem die Sitzung begann und endete.

Berechtigungsnachweise, bzw. *Credentials* sind Daten, die Keycloak verwendet, um die Identität eines Benutzers zu überprüfen, wie z.B. Passwörter oder digitale Zertifikate. Credentials sind wichtig für Autorisierungsprozesse, welche in 2.2.4 erklärt werden.

Ein *identity token* ist ein Token, das als Teil einer HTTP-Anfrage bereitgestellt werden kann und den Zugriff auf den aufgerufenen Dienst ermöglicht. Dies ist Teil der OpenID Connect- und OAuth 2.0-Spezifikation (wird ebenso in 2.2.4 genauer erläutert).

## 3.2 Representational State Transfer (kurz: REST)

Eine REST-API (auch RESTful-API oder RESTful-Web-API genannt) ist eine Anwendungsprogrammierschnittstelle *(kurz: API),* mit der es möglich ist Ressourcen in einer Anwendung zu beeinflussen.

Der REST-Architekturstil wurde erstmals im Jahre 2000 von Roy Fielding entwickelt [[7]](https://www.ibm.com/topics/restapis#:~:text=REST%20APIs%20communicate%20through%20HTTP,request%20creates%20a%20new%20record) und wird heute in Anwendungen verwendet, weil sie den Datentransfer zwischen Client und Server erleichtert [[8]](https://aws.amazon.com/de/what-is/restful-api/#:~:text=Systems%20that%20implement%20REST%20APIs,eliminates%20some%20client%2Dserver%20interactions). Auf den Architekturstil wird hier nicht eingegangen, weil er für die Thematik in dieser Arbeit nicht relevant ist.

In anderen Worten ist REST eine API, also eine Schnittstelle für zwei oder mehreren Anwendungen, in der Operationen an Ressourcen ausgeführt werden.

Mit Hilfe der Http-Methoden wird dem Server mitgeteilt, welche Operationen er an die gelieferte Ressource ausführen, soll:

* **GET:** Für den Abruf der Ressource
* **POST: F**ür die Erstellung, Suche und Massenänderung der Ressource
* **DELETE:** Entfernt die Ressource
* **PUT:** Ersetzung von Attributen innerhalb von Ressourcen
* **PATCH (**wird in 3.3 erklärt)

Die REST-API wird in der System for Cross-domain Identity Management verwendet, welche im nächsten Kapitel erläutert wird.

## 3.3 System for Cross-domain Identity Management (kurz: SCIM)

Das SCIM-Protokoll ist ein Protokoll auf der Anwendungsebene, welche Identitätsdaten verwaltet und bereitstellt [[9]](https://www.rfc-editor.org/info/rfc7643)*.* Hauptfunktion von SCIM ist es, die Benutzerkonten automatisch zu provisionieren und zu deprovisionieren, bei der JSON für die Formatierung der Benutzerdaten und REST als Schnittstelle zur Übermittlung der Daten (s. 2.2.2), verwendet wird.

Unter Provisionierung versteht man hier es, Benutzerkonten und deren Zugriffsrechte von einem System “A” in ein System “B” bereitzustellen und zu verwalten. Die Deprovisionierung beschreibt dann die bereitgestellten Daten in “B” durch “A” zu entfernen.

Es gibt und gab außer SCIM noch andere Protokolle, die dieselben Funktionen erfüllten wie SCIM, jedoch schwierig zu implementieren waren [[9]](https://www.rfc-editor.org/info/rfc7643). Dadurch verzichteten viele darauf solche Protokolle zu implementieren, wodurch es zu höherem Entwicklungsaufwand und dadurch dann zu höheren Kosten kam.

SCIM bietet für die SPE den Vorteil, dass die Kosten für die Benutzerverwaltungsvorgänge reduziert werden.

Die genaue Struktur von SCIM ist im folgenden Diagramm vereinfacht dargestellt:



Identity Provider

SCIM-Client

HTTPS

SaaS-Applikation

SaaS-Applikation

SaaS-Applikation

Abbildung : Vereinfachte Darstellung der SCIM-Struktur

In Abbildung 5 stellen GitLab, AWS und Atlassian die SaaS-Applikationen dar.

Unter SaaS versteht man den Begriff Software as a Service. Besonders an SaaS ist, dass der Anbieter, welcher die Software bereitstellt, diese für den Kunden wartet, installiert und betreibt. AWS, Atlassian und gitLab sind damit Anwendungen, welche hier in der Abbildung als solche SaaS-Anwendungen dargestellt werden.

Der SCIM-Client in der Abbildung ist ein *identity provider (kurz: IdP).* Ein Identity Provider, bzw. Identitätsanbieter speichert und verwaltet Benutzeridentitäten. Keycloak ist bspw. ein Identitätsanbieter.

SCIMs Schema ist attributbasiert. Ein Schema an sich ist “Sammlung von Attributdefinitionen, die den Inhalt beschreiben einer ganzen oder teilweisen Ressource beschreiben” [9, S. 6].

Dies bedeutet, dass Informationen eines Objektes Attribute sind, die an den Empfänger weitergeleitet werden. Der Empfänger entscheidet dann, welche Operationen er an den Attributen vornimmt.

Ressourcen in SCIM sind Artefakte, die von einem Dienstanbieter verwaltet werden und mehrere Attribute besitzen[[9]](https://www.rfc-editor.org/info/rfc7643).Es sind kurz ausgedrückt Endpunkte im System.

Das Protokoll wendet Operationen an Ressourcen durch die Http-Methoden (in 2.2.2 erläutert) an. In SCIM gibt es die weitere HTTP-Methode **PATCH** für die teilweise Aktualisierung von Attributen. Dies ist notwendig, wenn man nur bestimmte Teilressourcen aktualisieren will.

Wenn man z.B. Ein Objekt namens “Student” über fünf Attribute besitzt, aber man will nur das Attribut “Ort” aktualisieren, geschieht dies durch die Http-Methode **PATCH**.

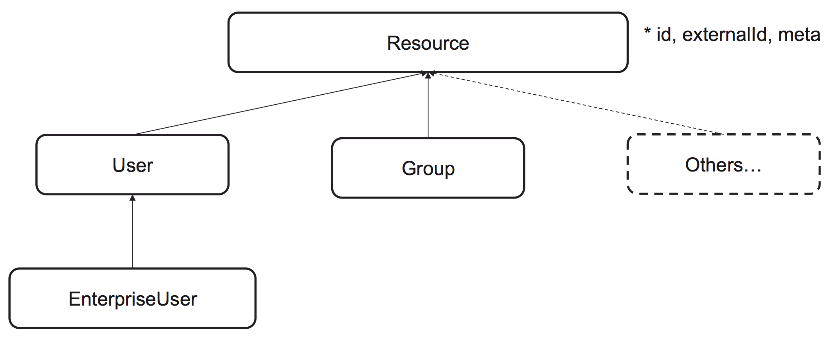


Abbildung : Ressourcenstruktur in SCIM (Quelle: "SCIM“, https://simplecloud.info/, Zugriffsdatum: Juni 2024)

Zu einer Ressource gehören bspw. die ID der jeweiligen Ressourcentypen wie User und Group, welche ihre eigenen Endpunkte haben. Die ID ist hierbei das Attribut, welches eine Ressource einzigartig und unverwechselbar macht.

Ein Ressourcentyp ist “eine Art von Ressource, die von einem Dienstanbieter verwaltet wird. Der Ressourcentyp definiert den Ressourcennamen, die Endpunkt-URL, die Schemata, und andere Metadaten, die angeben, wo eine Ressource verwaltet wird und wie sie zusammengesetzt ist, z. B. "Benutzer" oder "Gruppe” "[9, S. 5].

## 3.4 OAuth2 und OpenID Connect

OAuth entstand im Jahre 2006 durch das Projekt “Twitter OpenID” [[10]](https://oauth.net/about/introduction/), weil das Problem bestand, dass es keinen offenen Standard für die Übertragung des API-Zugangs gab.

Die Einführung eines solchen Standards war wichtig, da er es Entwicklern ermöglichte, Anwendungen sicher mit APIs zu verbinden, ohne dass sie sensible Zugangsdaten wie Benutzernamen und Passwörter speichern oder weitergeben mussten. Durch OAuth wurde es möglich, Ressourcen vor unautorisierten Zugriffen zu schützen.

OAuth2 ist ein Autorisierungsprotokoll. Das Protokoll überprüft, ob auf Ressourcen zugegriffen werden darf, wobei Zugriffstoken verwendet werden. Oft handelt es sich bei dem Format der Zugriffstokens um JSON Web Tokens *(kurz: JWT)*.

Damit der Client an die gewünschte Ressource gelangt, muss er sich bei der Quelle, wo sich diese Ressource befindet, identifizieren und authentifizieren. Dafür benötigt er vom Autorisierungsserver die c*redentials,* den *client\_secret und eine client\_id.*

Die c*redentials* beinhalten die Attribute des Clients, wie bspw. die Berechtigungen und Rollen des Clients. Das *client\_secret* ist das Passwort des Clients, welches bspw. in Keycloak automatisch generiert wird nach dem es erstellt worden ist. Die *client\_id* ist eine eindeutige Kennung, die den Client identifiziert und unverwechselbar macht. Jeder Client besitzt eine einmalige *client\_id*.

Anschließend fordert der Client beim Autorisierungsserver einen Token an.

Der Token-Ablauf funktioniert wie folgt [[11]](https://auth0.com/de/intro-to-iam/what-is-oauth-2):  
Der Client fordert beim *Autorisation Server* eine Autorisierung an (wird auch als Autorisierungsanfrage bezeichnet) und gibt dabei die Client-ID und das Geheimnis zur Identifizierung an.

Zudem stellt er einen Endpunkt-URI (Redirect-URI) bereit, an den das Zugriffstoken oder der Autorisierungscode gesendet werden soll.

Der Autorisierungsserver autorisiert den Client und prüft, ob die angeforderten Bereiche zulässig sind. Der Eigentümer der Ressource interagiert mit dem Autorisierungsserver, um den Zugriff zu gewähren.

OAuth2 unterscheidet zwischen vier Rollen im Autorisierungssystem [[11]](https://auth0.com/de/intro-to-iam/what-is-oauth-2):

1. Der *Resource Owner* besitzt über die geschützte Ressource und erlaubt darauf zuzugreifen. Ein *Resource Owner* kann z.B. Keycloak sein.
2. Der Client ist die jeweilige Webseite, welche die Ressource v*om Resource Owner anfordern will*. Um auch auf die gewollte Ressource auch zuzugreifen, braucht der Client ein Zugriffstoken. Der Client ist bspw. die SPE.
3. Die Anforderung des Clients erhält zuerst vom *Authorization Server,* welcher den Zugriffstoken prüft und an den *Resource Owner* weiterleitet, wenn erfolgreich authentifiziert wurde. Es gibt zwei Endpunkte im Authorization Server:
   * **Autorisierungsendpunkt:** Wickelt die interaktive Authentifizierung und Zustimmung des Benutzers ab. Erfolgt, bevor der Token-Endpunkt aufgerufen wird.
   * **Token-Endpunkt:** Ist in einer Maschine-zu-Maschine Interaktion mit eingebunden. Durch diesen Endpunkt erhält der Benutzer einen Zugriffstoken für die Autorisierung.
4. Der *Resource Server* beschützt die Ressourcen des Benutzers und erhält vom Benutzer Zugriffsanforderungen auf eine Ressource. Es akzeptiert und validiert den Zugriffstoken des Clients.

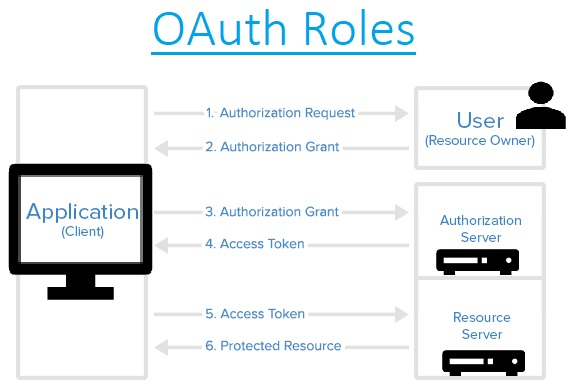


Abbildung : Rollen und Authorisationsprozess in OAuth2 (Quelle:“Medium“, [#SpringSecurity:Oauth 2.0 Roles | Medium](https://medium.com/@greekykhs/ro-acd8cb4cc0f4), Zugriffsdatum: Juni 2024)

Berechtigungen (G*rants*) in OAuth2 sind “eine Reihe von Schritten, die ein Client ausführen muss, um eine Autorisierung für den Ressourcenzugriff zu erhalten” [[11]](https://auth0.com/de/intro-to-iam/what-is-oauth-2).

Man spricht oft von dem Prozess *Authentication Flow (Authentifizierungsfluss)*, bei dem die Identität des Benutzers zuerst überprüft wird, bevor Zugriff auf die Ressource gewährt wird [[11]](https://auth0.com/de/intro-to-iam/what-is-oauth-2).

Man unterscheidet sieben *Granttypen*, jedoch werden hier nur die für das Projekt relevante *Granttypen* erläutert:

* Der *Client Credentials Grant Type* wird für automatisierte Prozesse und Microservices verwendet. Zur Authentifizierung wird das *client\_secret* und die *client\_id* benötigt.
* Authorization Code Flow

OpenID Connect *(kurz: OIDC)* ist ein Authentifizierungsprotokoll, bzw. ein Identitätsprotokoll [[12]](https://auth0.com/intro-to-iam/what-is-openid-connect-oidc), welches OAuth2 in Anwendungen ergänzt. OIDC verifiziert erst die Identität eines Clients (wird in OIDC auch als *Relying Party (kurz: RP)* bezeichnet), bevor der Autorisierungsprozess durch OAuth2 erfolgt. Die “Identitätsschicht” [[12]](https://auth0.com/intro-to-iam/what-is-openid-connect-oidc), die OIDC verfügbar macht, liegt also über der “Autorisierungsschicht” [[12]](https://auth0.com/intro-to-iam/what-is-openid-connect-oidc),, die OAuth2 bereitstellt. Um die Identitätsschicht bereitzustellen, werden Tokens verwendet. Man unterscheidet drei Tokentypen in OIDC [[13]](https://www.researchgate.net/publication/358771227_Applying_Spring_Security_Framework_with_KeyCloak-Based_OAuth2_to_Protect_Microservice_Architecture_APIs_A_Case_Study):

* *ID-Tokens sind* im JWT-Format und können Identitätsdaten bereitstellen.
* *Access Tokens* zeichnen sich durch eine kurze Lebensdauer aus. Ihre Gültigkeitsdauer läuft nach einer festgesetzten Zeit ab und wenn die Zeit abgelaufen ist, kann man sich mit den Tokens nicht mehr authentifizieren. Sie dienen dazu, dass der Benutzer auf die gewünschte Ressource Zugriff erlangt.
* *Refresh Tokens sind neue angeforderte Access Tokens.* Sie sind hingegen zu den Access Tokens langlebig. Sie haben keine festgesetzte Gültigkeitsdauer. Der Vorteil dieses Tokens ist, dass sie automatisch neu generiert werden, mit der sich der Benutzer anmelden kann.

OpenID Connect (OIDC) ergänzt OAuth2 durch zusätzliche Funktionen für die Authentifizierung und Identitätsverwaltung:

* **Authentifizierung des Benutzers**: OIDC ermöglicht es dem Client, mithilfe von ID-Tokens (JWT-Tokens) die Identität des Benutzers zu überprüfen. Diese Tokens werden oft in Verbindung mit OAuth2 verwendet, um sicherzustellen, dass der Benutzer, der Zugriff auf Ressourcen beantragt, auch tatsächlich authentifiziert ist.
* **Bereitstellung von Identitätsinformationen**: Die ID-Tokens enthalten *Claims,* die Identitätsinformationen wie den Benutzernamen oder andere verifizierte Attribute des Benutzers enthalten. Diese Informationen sind nützlich für den Client, um personalisierte Dienste anzubieten oder Berechtigungen basierend auf der Benutzeridentität zu verwalten.

Beide Protokolle, Authentifizierung - sowie Autorisierungsprotokolle, werden gemeinsam verwendet, d.h. dass das Autorisierungsprotokoll (OAuth2) das Authentifizierungsprotokoll (OIDC) beinhaltet.

OAuth2 wird hier gemeinsam mit Keycloak und Spring Security (erläutert in 2.2.6) kombiniert, weil es dadurch eine effizientere Sicherheitslösung ist.

## 3.5 Spring und Springboot

Das Spring Framework ist ein Open-Source-Framework für die Anwendungsentwicklung in der Programmiersprache Java [[14]](https://iamgodsom.wordpress.com/wp-content/uploads/2014/08/wrox-professional-java-development-with-the-spring-framework.pdf).

Springboot baut auf das Spring Framework auf und erleichtert die Entwicklung von Anwendungen. Bspw. bietet Springboot zusätzlich die Autokonfiguration [[15]](http://www.laputan.org/drc/drc.html) an. Autokonfiguration bedeutet, dass die gewünschten Abhängigkeiten im Projekt automatische konfiguriert werden in die Springboot-Anwendung.

Die angewandten Technologien, die Spring kennzeichnen, sind die *Dependency Injection* und die *Inversion of Control (kurz: IoC)***.**

Beim IoC handelt es sich um ein Prinzip, bei der ein Programmablauf einer Anwendung direkt in das Framework übertragen werden, mit dem es verbunden ist [[15]](http://www.laputan.org/drc/drc.html).Um es verständlicher zu machen, kann man sich vorstellen, dass die Java-Anwendung ein “Kunde” ist in einem Restaurant und das Framework der “Kellner”. Der “Kellner" erfüllt alle Anforderungen des “Kunden”, wobei der “Kunde” dem “Kellner” diese Anforderungen mitteilt. Somit besagt das IoC, dass das Framework die Aufgaben den Code und dessen Schnittstellen steuert und kontrolliert, statt dass das die Anwendung tut.

Die *Dependency Injection* sorgt dafür, dass einer Anwendung Abhängigkeiten und Konfigurationen bereitgestellt werden, die sich außerhalb der Objekte, die in der Anwendung definiert werden, befinden [[15]](http://www.laputan.org/drc/drc.html).

Man nimmt als Beispiel, dass es eine Java-Anwendung namens “Portal\_3” gibt, in der die Klasse “Chamber” definiert ist. Wenn die Klasse nun die Attribute “chamber\_number” und “chamber\_name” in die Datenbank gespeichert werden sollen, so geschieht dies in der Anwendung selbst. Man kann bspw. *Annotations* verwenden, welche die Klasse in die Datenbank injiziert, bzw. abbildet.

@Entity

public class Chamber {

@Id

private Long chamber\_id;

private String chamber\_number;

private String chamber\_name;

}

Abbildung : Beispiel, wie Annotation verwendet werden: @Entity kennzeichnet die Klasse als Datenbanktabelle und sorgt dafür, dass sie als solche in der Datenbank abgebildet wird

Da es durch Springboot vereinfacht wird, komplexe Projekte mit vielen Abhängigkeiten wie Datenbankanbindungen, Autorisierungsprotokollen und Zugriffskontroll-Frameworks, zu erstellen, wurde die SPE als Spring-Projekt erstellt.

Springboot-Projekte sind strukturiert aufgebaut und sind in vier Schichten aufgeteilt.

In der Controller-Schicht werden die Anfragen des Clients anhand der Endpunkte durch Protokolle wie SCIM gesendet.

In der Repository-Schicht werden durch das genannte Konzept der *Dependency -Injection* die CRUD-Operationen (Create, Read, Update, Delete) ausgeführt. Je nach http-Methode, die vom Client angefordert wird, wird die entsprechende CRUD-Methode ausgeführt. Dadurch werden Anforderungen des Clients in Sprache umgewandelt, die die Datenbank versteht.

Die Service-Schicht bestimmt, wie Ressourcen angefordert werden und unter welchen Bedingungen. Eine Bedingung ist bspw., dass nur Ressourcen gefordert werden können von Benutzern namens “Jeff”.

Die Model-Schicht beinhaltet sogenannte Models. Ein Model ist ein Objekt, welches seine Domäne abbildet [[14](https://iamgodsom.wordpress.com/wp-content/uploads/2014/08/wrox-professional-java-development-with-the-spring-framework.pdf), S. 39]. Man kann als Beispiel ein Model “Studium” nehmen. Das Model beinhaltet die Entitäten Professor, Klausur, Kurs, etc. Ein Model beinhaltet also mehrere Entitäten. Ein Model nennt man auch Domain Object.

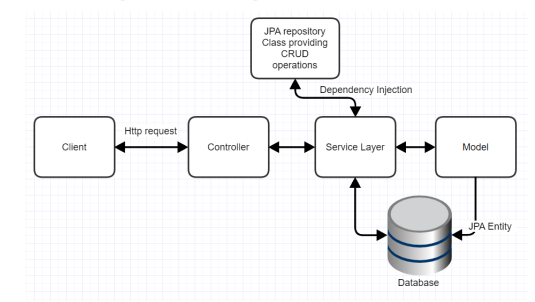


Abbildung : Vereinfachte Darstellung der Schichten in Springboot (Quelle: )

## 3.6 Springboot Security

Spring Security ist ein Authentifizierungs- und Zugriffskontroll-Framework, welches Spring-Anwendungen absichert [[16]](https://spring.io/projects/spring-security#samples). Spring Security stellt Authentifizierung- und Autorisierungsprotokolle wie OAuth2 und OpenID Connect zur Verfügung. Die aktuelle Version ist die 6.3.1.

Zu den Hauptkomponenten in Spring Security gehörender *SecurityContext* und der *SecurityContextHolder*.

Der *SecurityContextHolder* speichert Details zu dem Benutzer, welcher in der Spring-Anwendung authentifiziert ist. Sie enthält den *SecurityContext,* welcher ein Objekt zur Authentifizierung enthält. Das Objekt verfügt über folgende Attribute [[17]](https://docs.spring.io/spring-security/reference/servlet/authentication/architecture.html#servlet-authentication-authentication):

* **Credentials:** Berechtigungsanforderungen wie Passwörter
* **Principal:** Attribut, welches den Benutzer identifziert
* **Authorities:** Berechtigungen des Benutzers in der Anwendung

Die jeweiligen Klassen, welche den *SecurityContextHolder* und den *SecurityContext* enthalten, sind gleichnamig annotiert.

Spring Security wurde im Spring-Projekt integriert, weil es OAuth2 direkt verfügbar macht, wodurch keine aufwendigen Installationen erfolgen müssen [[18]](http://postgresql.de/was-ist-postgresql).

## 3.7 PostgreSQL

Bei PostgreSQL handelt es sich um ein objektrelationales Datenbank-Management-System *(kurz: ORDMS)* [[19]](https://www.researchgate.net/publication/3296158), welches von Michael Stonebraker entwickelt wurde [[19].](https://www.researchgate.net/publication/3296158)

Im Vergleich zu MySQL ist PostgreSQL nicht schneller in der Datenverarbeitung, sondern robuster und datenkonsistenter [[20]](https://www.researchgate.net/publication/296259511_A_performance_analysis_of_the_DBMS_-_MySQL_vs_PostgreSQL). Es ist wichtig, dass provisionierte Daten geschützt und in der SPE konsistent integriert sind , weil es viele identische Benutzerdaten in unterschiedlichen Systemen gibt. Bei inkonsistenter Integrität besteht die Gefahr, dass bei problematischer Datenaktualisierung wie Lost Update [[21]](https://elearning.dhbw-stuttgart.de/moodle/pluginfile.php/571565/mod_resource/content/1/Datenbanken7Transaktionen.pdf), unkorrekte Benutzerdaten in vielen Systemen provisioniert werden.

Aufgrund dessen und weil Keycloak PostgreSQL unterstützt [[22]](https://www.keycloak.org/server/db), entschied man sich PostgreSQL als Datenbank zu verwenden.

## 3.8 Jira

Jira ist ein agiles Projektmanagementtool, um Softwareentwicklung zu planen und nachzuverfolgen [[23]](https://www.atlassian.com/de/software/jira/guides/getting-started/introduction#dig-into-specific-features). Jira diente zur Erstellung der User Stories im Epic “SPE Multitenancy” für diese Projektarbeit und gewährleistete es, zeitlich zu überwachen und den Fortschritt der Entwicklung zu verfolgen. Ebenso diente es dazu die Aufgaben für diese Projektarbeit zu strukturieren (genauere Vorgehensweise wird in 2.2.5 erklärt).

## 3.9 GitLab

Code und Dokumente verändern sich im Laufe der Zeit und um diese zu verwalten, gibt es spezielle Konfigurationsverwaltungssysteme wie z.B. GitLab. GitLab basiert auf dem Konzept des *Code-Modify-Merge.* Bei diesem Konzept kopiert jeder Entwickler seine eigene Version des Codes bei sich lokal auf dem Computer. Der Code wird an einen zentralen Ort gespeichert, sobald der Entwickler den Code fertig entwickelt hat.

In GitLab unterschiedet man folgender Begriffe [[24]](https://elearning.dhbwstuttgart.de/moodle/pluginfile.php/634308/mod_resource/content/0/TINF22F_VE09_CM_CI_CD.pdf):

● Eine erstellte *Version* ersetzt eine vorangehende *Version,* also eine neue, verbesserte Form des existierenden Software-Projektes

● *Varianten* existieren zeitlich parallel, wenn bspw. zwei Entwickler am gleichen Software-Projekt arbeiten

● *Konfigurationen* sind Zusammenstellungen einzelner *Versionen* und *Varianten*

In GitLab ist es möglich Software zu deployen. Zu deployen bedeutet, Software bereitzustellen. GitLab unterstützt somit auch das Prinzip des *Continious Deployment, (kurz: CD) Continious Integration (kurz: CI)* und des *Continious Delivery (kurz: CD, nicht verwechseln mit Continious Deployment).*

Continious Deployment bedeutet, dass Software fortlaufend bereitgestellt wird an andere, fremde Benutzer.

CI bedeutet, dass der jeweilige Code der Entwickler automatisch und zentral ins Verwaltungssystem gelangen und dort geprüft wird, ob die Anwendung ohne Softwarefehler funktioniert. Es laufen unter anderem automatisierte Tests bei der Integration in GitLab [[25]](https://www.researchgate.net/publication/354720705_A_STUDY_AND_ANALYSIS_OF_CONTINUOUS_DELIVERY_CONTINUOUS_INTEGRATION_IN_SOFTWARE_DEVELOPMENT_ENVIRONMENT).

Continious Delivery umfasst eine Reihe von Praktiken, die darauf abzielen, den Prozess zu verbessern, durch den Änderungen aus der Versionskontrolle in die Produktion oder zur Freigabe des Projektes gelangen [[25]](https://www.researchgate.net/publication/354720705_A_STUDY_AND_ANALYSIS_OF_CONTINUOUS_DELIVERY_CONTINUOUS_INTEGRATION_IN_SOFTWARE_DEVELOPMENT_ENVIRONMENT).

Da CI und Continious Delivery gemeinsame Prozesse in einem Verwaltungssystem sind und beide miteinander zusammenhängen beim Deployment, spricht man verkürzt von CI/CD [26], bzw. von einer CI/CD-Pipeline. Eine Pipeline ist demnach eine Reihe automatisierter Schritte, die zusammenarbeiten, um Software von der Entwicklung bis zum Deployment zu bringen [[25]](https://www.researchgate.net/publication/354720705_A_STUDY_AND_ANALYSIS_OF_CONTINUOUS_DELIVERY_CONTINUOUS_INTEGRATION_IN_SOFTWARE_DEVELOPMENT_ENVIRONMENT).

## 3.10 Kubernetes und Docker

Container sind Softwarepakete, welche alle Elemente enthalten, die nötig sind, um als laufende Anwendung ausgeführt zu werden [[26]](https://www.docker.com/resources/what-container/). Container verfügen über ein eigenes Dateisystem, eine Bibliothek und einen Speicher, aber enthalten kein eigenes Betriebssystem.

Docker ist eine Plattform, auf der Anwendungen, bzw. Container bereitgestellt und ausgeführt werden können. In Docker unterscheidet man von Docker-images und dem Docker-Container selbst. Das Docker-images beinhaltete alle Ressourcen der Anwendung. Man kann es als einen nicht-laufenden Container definieren. Der Docker-Container ist damit der laufende Container selbst, also die gestartete und funktionierende Anwendung [[27]](https://docs.docker.com/guides/docker-overview/).

Docker bietet den Vorteil, dass es benutzerfreundlich ist, und Leistung und Sicherheit gewährt im Vergleich zu traditionellen Virtualisierungskonzepten [[28].](https://jyx.jyu.fi/bitstream/handle/123456789/48029/URN%3aNBN%3afi%3ajyu-201512093942.pdf?sequence=1&amp;isAllowed=y)

Ebenso spart es Komponenten wie Betriebssysteme, welche in Virtuellen Maschinen je Anwendung verfügbar gemacht werden.

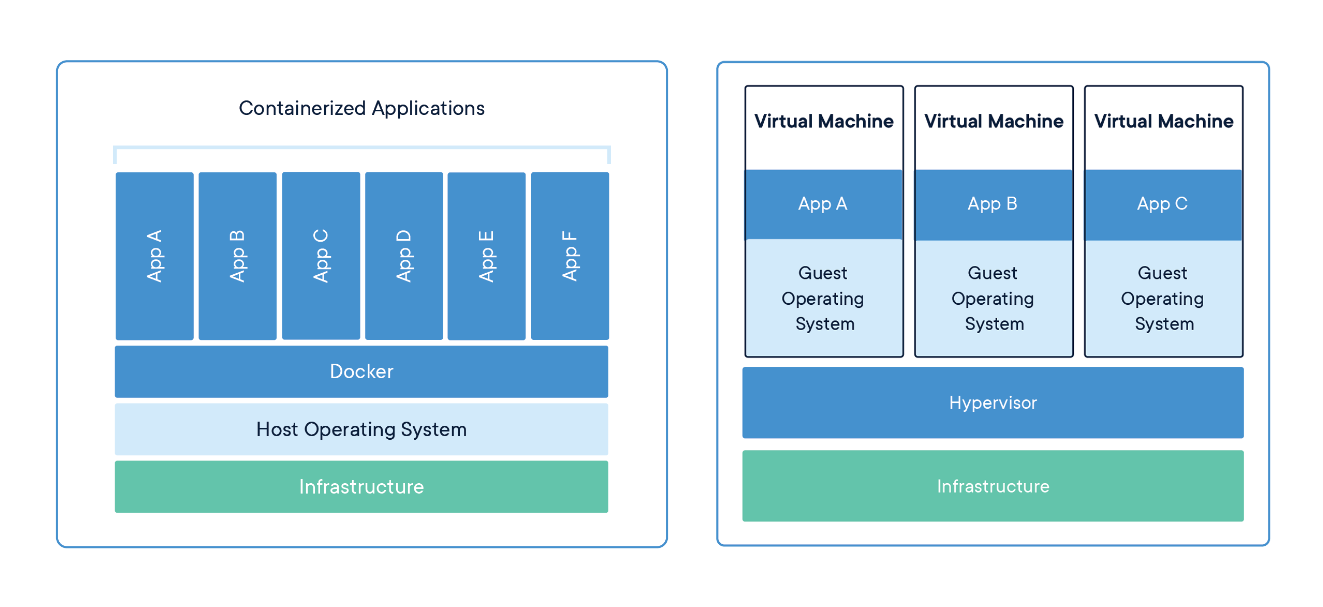


Abbildung : Vergleich von Container-basierten Anwendungen in Docker und Virtuellen Maschinen (Quelle:“Docker“, [What is a Container? | Docker](https://www.docker.com/resources/what-container/), Zugriffsdatum: Juni 2024)

Kubernetes ist eine Containerplattform, in der man containerbasierte Anwendungen verwaltet [[29]](https://kubernetes.io/docs/concepts/overview/).

Wenn sich mehrere Anwendungen den gleichen physikalischen Server teilen, kommt es zu Problemen bei der Ressourcenzuweisung. Wenn man Container verwendet, werden diese Probleme gemieden. Kubernetes verwaltet also diese Container.

In Kubernetes gibt es bestimmte Begriffe, die definiert wurden, um das Konzept von Kubernetes an sich zu beschreiben: Sogenannte *Pods* beinhalten einen oder mehrere Gruppen von Containern. Die *Nodes (Knoten)* sind Arbeitsrechner, auf denen ein oder mehrere Pod(s) läuft. Es gibt drei Knotenkomponenten [[30]](https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/):

* **Kubelet:** Ein Softwareprogramm, welcher auf jedem *Node* einzeln läuft und prüft, ob die Pods nach den Bedingungen der Container laufen.
* **Kube-proxy:** Stellt sicher, dass die Komponenten im Cluster erreichbar sind, damit bspw. Der Kubelet mit dem API-Server kommunizieren kann.
* **Container Runtime:** Verwaltet und führt den Container auf dem Node aus.

Ein *Kubernetes-Cluster* wiederrum enthält die *Nodes* und umfasst den *Kubernetes-Control-Plane*. Dieser Plane, auch Master genannt, ist eine Softwareschicht, mit der man Container anlegen, aktualisieren und löschen kann [[31]](https://kubernetes.io/docs/concepts/overview/components/).

Der Master verfügt über folgende Komponenten [[31]](https://kubernetes.io/docs/concepts/overview/components/):

* **kube-apiserver:** Das Interface der Verwaltungsoperationen. Es nimmt Anfragen von Benutzern an und aktualisiert den Stand des Clusters.
* **etcd:** Ein verteilter Speicher, welcher Daten des Clusters speichert. Der API-Server kube-apiserver liest Daten und speichert sie in etcd.
* **kube-scheduler:** Dieser Scheduler ordnet die Pods den entsprechenden Nodes im Cluster zu. Was ein Scheduler ist, wird in Kapitel 2.3.3 erläutert.
* **kube-control-manager:**  Überwacht den Zustand des Clusters über den API-Server und führt Aktionen durch, um Abweichungen vom gewünschten Zustand zu korrigieren.
* **cloud-control-manager:** Wenn Kubernetes in einer Cloud-Umgebung läuft, kommuniziert der Cloud Controller Manager mit der Cloud-API, um Cloud-Ressourcen zu verwalten.

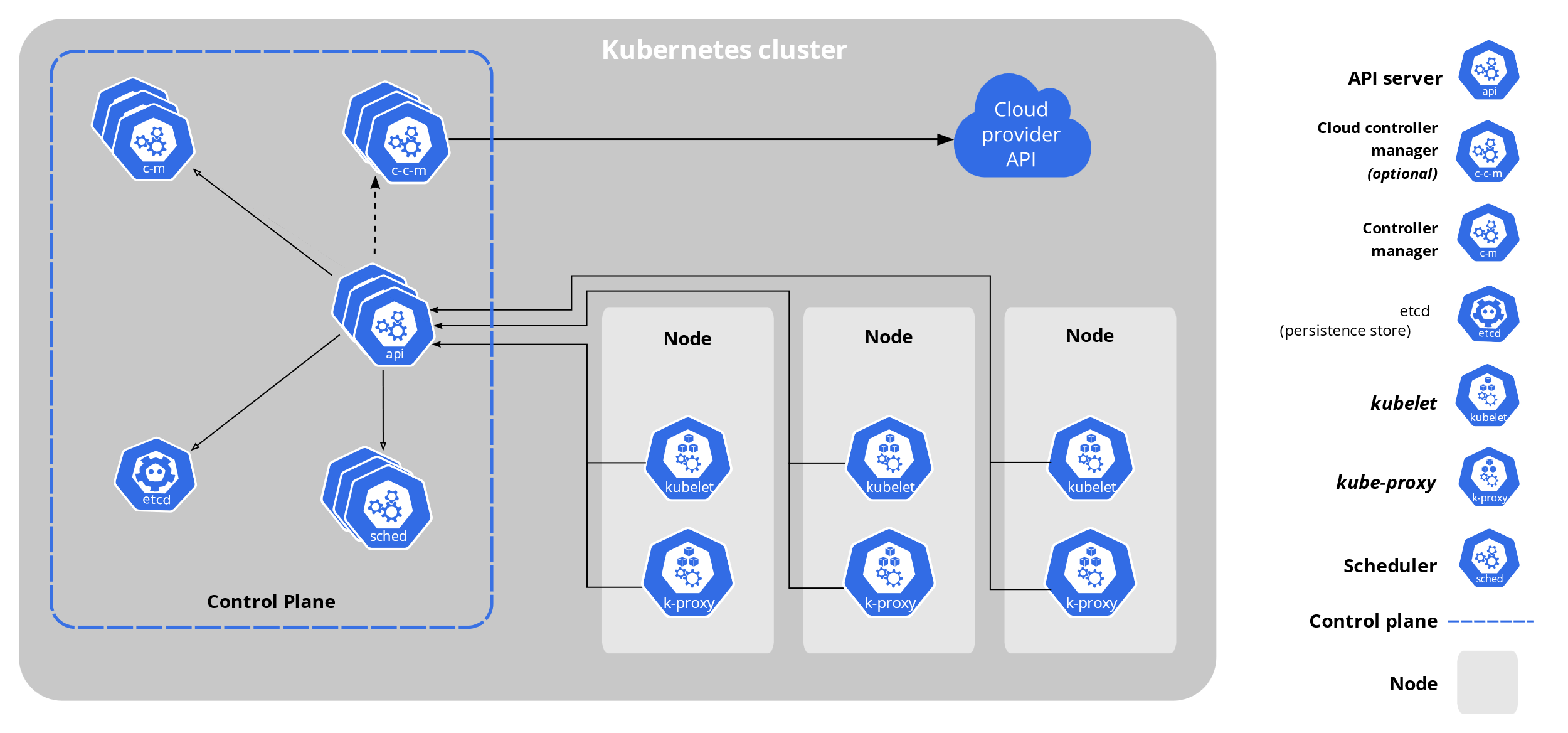


Abbildung : Vereinfachte Darstellung, wie Pods und Nodes zusammenhängen (Quelle:“Kubernetes“, [Kubernetes Components | Kubernetes](https://kubernetes.io/docs/concepts/overview/components/), Zugriffszeitpunkt: Juni 2024)

Der Unterschied zwischen Docker und Kubernetes ist, dass Docker die Anwendungen erst in Container verpackt und Kubernetes mehrere dieser Container verwaltet.

# 4. Stand der SPE vor dieser Projektarbeit

## 4.1 Abhängigkeiten und Bezug zu Docker

Da es sich bei der SPE um ein Springboot-Projekt handelt, hat sie eine Datei mit den notwendigen Anhängigkeiten, wie bspw. OAuth2 oder Springboot Security.

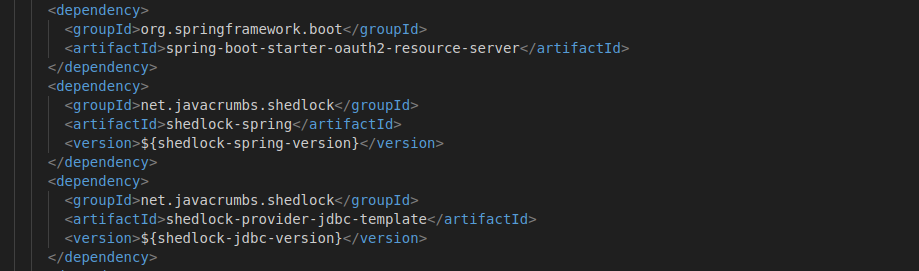


Abbildung : Ausschnitt aus der pom.xml aus der SPE

Die Abhängigkeiten werden nach dem *IoC-*Prinzip mit ins Projekt eingebunden, sobald man dies so konfiguriert hat.

Es wurden folgende Versionen der Abhängigkeiten für das Projekt verwendet:

* **Springboot:** Version 2.7.5
* **PostgreSQL:** Version 42.5.1
* **Java:** Version 11
* **Keycloak:** 23.0.0

Ansonsten werden die Versionen, wie bspw. Die Springboot Security Version automatisch aktualisiert. Die pom.xml ist so konfiguriert, dass sie immer die zuletzt veröffentlichte Version der jeweiligen Abhängigkeit verwendet.

Die SPE selbst ist als Docker-image ausführbar. In der Datei *docker-compose.yml* sind alle Anmeldeinformationen enthalten, damit sich die SPE zur Datenbank PostgreSQL und Keycloak verbinden kann. Keycloak und PostgreSQL laufen dabei in einem Docker-image.

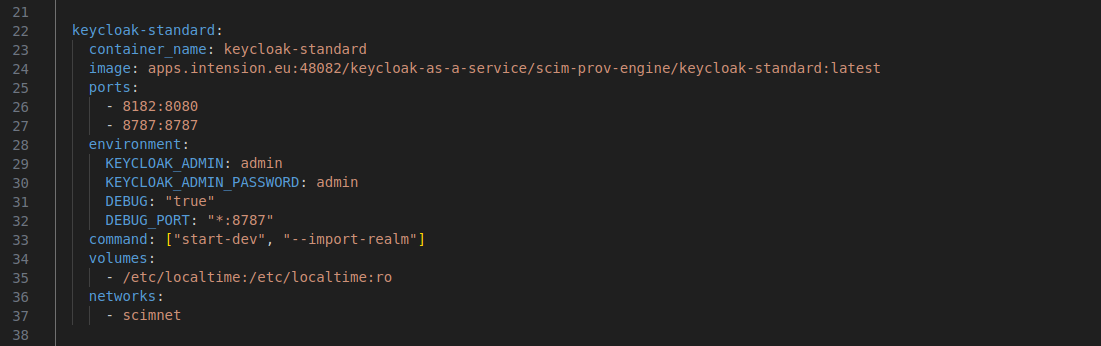


Abbildung : Ausschnitt aus der docker-compose.yml. Daten zum image, in dem sich keycloak befindet, ist angegeben.

## 4.2 Authentifizierung und Autorisierung in der SPE

In der SPE sind zwei *Clients* zur Authentifizierung angelegt:

Der *keycloak-client* ist der Client, welcher in Keycloak angelegt ist und welcher in der SPE dafür da ist, Zugriffsrechte auf Keycloak zu haben. Der *keycloak-client* muss die Rolle ***Service Account Roles*** haben. Mit dieser Rolle kann sich der *keycloak-client*, der in der SPE angelegt ist in keycloak authentifizieren. Dieser Client repräsentiert den *KeycloakWebClient,* welcher angelegt wurde, damit die SPE Zugriff auf Keycloak hat.

Über die Admin-Konsolen-URL werden jeweiligen Informationen über die User, Clients, Realms, Groups und Rollen importiert in die SPE.

Zudem muss der *keycloak-client* über die Rollen ***manage-users*** und ***view-clients*** in der Kategorie ***Service Account Roles*** besitzen.

Mit der manage-users-rolle ist man befugt anderen Benutzern Admin-Rollen zuzuteilen. Mit der Rolle view-clients ist man befugt Clients anzuzeigen.

Der *keycloak-client* ist mit dem OAuth 2.0 Authentifizierungsfluss C*lient Credentials* konfiguriert. Dieser Authentifizierungsfluss ermöglicht den Zugriff auf ein Token, das die notwendigen Berechtigungen und Daten enthält.

Im*scim-client* befindet sich der Admin, welcher die höchsten Zugriffstufe auf ein Dienstkonto hat, führt mit CRUD-Methoden Operationen auf “Target-Systemen”, also Zielapplikationen, in denen die Benutzerdaten provisioniert werden, aus.

Für den *scim-client* muss in Keycloak die ***client authentication*** und der Authentication ***Flow Direct Access Grants*** aktiviert sein.

Mit diesem Autorisierungsfluss leitet die SPE die c*lient\_id* und die c*redentials* des *scim-clients* und des *keycloak-clients aus Keycloak* direkt an den *Resource Owner*, in diesem Fall die jeweilige Zielanwendung, die in Keycloak als Client eingetragen ist.

Um Operationen an einen *Client*, wie bspw. Löschen (DELETE) oder erstellen (POST) durchzuführen, muss ein Token verlangt werden. Der Admin-User, der in Keycloak im Client *scim-client* eingetragen ist, muss über folgende Rollen verfügen:

* **create-realm:** Mit dieser Rolle ist man befugt *Realms* anzulegen.
* **manage-clients:** Mit dieser Rolle ist man befugt, Clients zu erstellen und zu löschen.
* **view-users:** Man ist befugt Informationen über einen Benutzer einzusehen.
* **target-system-admin-role:** Definierte Rolle im scim-client. Notwendige Rolle des Admins in keycloak, der in der SPE angelegt ist.
* **view-clients**
* **manage-users**

Jede Ausführungsfunktion einer Rolle bezieht sich auf ein Realm. Wenn bspw. ein Admin-User die Rolle manage-users für das Realm “nintendo” hat, so darf der Admin-User andere Benutzer in diesem Realm bearbeiten.

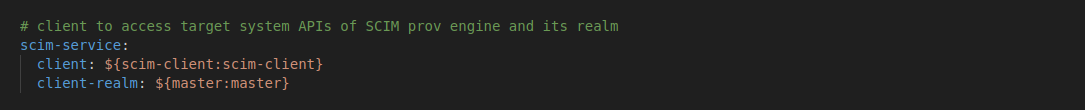


Abbildung : Abschnitt aus der application-dev.yml-Datei. Hier werden die Variablen als Platzhalter für die eigentlichen Namen der Attribute des angelegten keycloak-client aus Keycloak gesetzt.

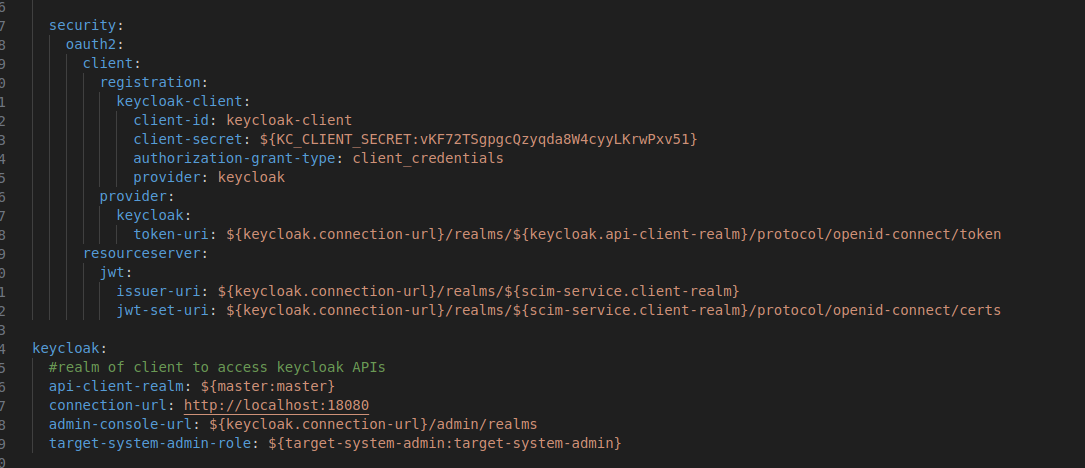


Abbildung : Wie Abb. 9, nur mit dem keycloak-client. Dieser muss im master angelegt werden.

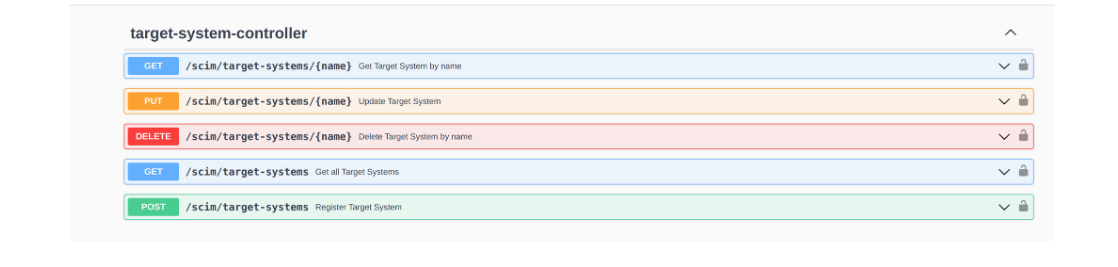


Abbildung : Endpunkte, um auf die jeweiligen JSON-Objekte Operationen auszuführen.

Folgende Klassen spielen eine wichtige Rolle für den Autorisierungsfluss:

Der *KeycloakRealmRoleConverter* ist eine Klasse in der SPE, welche die vom *scim-client* angeforderten JWT-Token in Grant-Objekte um, welche von der Spring Security in den Autorisierungsfluss gelangen.

In der Java-Klasse *OAuthRestTemplateConfig* wird OAuth2 mit OpenID Connect eingebunden und überprüft, ob der Token, der vom *scim-client* angefordert wird gültig ist. Sie verwaltete den *Client Credentials Flow*. Rollen, die geprüft werden sind bspw. die *manage-users* Rolle.

In der *SecurityConfig* werden Benutzerrechte bestimmt. In dieser Klasse werden Rollen überprüft, die den Benutzern zugeteilt sein müssen.

## 4.3 Scheduler in der SPE

Scheduling plant Abläufe, ordnet Aufgaben und optimiert Ziele bei Randbedingungen [[32]](https://www.uni-ulm.de/in/theo/lehre/archiv/ss-2017/scheduling/). Scheduling ist damit wichtig in der Informatik für automatische Prozesse.

Unter Scheduling hier fallen keine typischen Schedulingalgorithmen in Betriebssysteme wie z.B. prioritätenbasierte Algorithmen [[33]](https://dl.acm.org/doi/pdf/10.1145/321738.321743), sondern spezielle Algorithmen in der SPE.

Der Scheduler *UserImportScheduler* holt alle 2 Minuten automatisch die Daten von Keycloak. Dabei muss der *keycloak-client* in dem Realm sein, wo sich auch der admin-benutzer befindet, also im master-realm. Eine weitere Aufgabe dieses Schedulers ist es, alle Zielsysteme abzurufen. Er holt Benutzer– und Gruppenprovisionierungen.

Der *UserImportScheduler* aktualisiert, löscht und überprüft automatisch, ob neue Benutzer im System sind. Es werden sogenannte “Import-Jobs” erstellt und protokolliert, welche die Operation, die vom Scheduler ausgeführt wurde, darstellt.

Der Import-Algorithmus lässt sich wie folgt beschreiben:

1. Erstelle ein ImportJob.
2. Lösche verwaiste SCIM-Benutzer.
3. Iteriere über alle Zielsysteme.
4. Erstelle eine Benutzergruppen-Karte für jedes Zielsystem.
5. Erstelle eine Benutzerbereitstellungskarte für jedes Zielsystem.
6. Hole alle Keycloak-Benutzer mit Client-Rolle für jedes Zielsystem.
7. Iteriere über die Keycloak-Benutzer.
   1. Setze das Realm des Keycloak-Benutzers.
   2. Erstelle einen Schlüssel basierend auf Keycloak-ID und Realm.
   3. Hole den entsprechenden SCIM-Benutzer.
   4. Wenn der Benutzer in der Bereitstellungskarte vorhanden ist:
      1. Überprüfe und aktualisiere gegebenenfalls den SCIM-Benutzer.
   5. Andernfalls erstelle einen neuen SCIM-Benutzer.
   6. Verwalte Benutzergruppen-Zuweisungen.
8. Markiere Benutzer zur Deprovisionierung.

Der Scheduler *UserProvisioningScheduler* provisioniert jede Minute das Benutzerdatum und ordnet einen Benutzer in eine Gruppe zu. Auch hier muss der *keycloak-client* die entsprechenden Rollen haben, um auf den Client zugreifen zu können.

Der Provisionierung-algorithmus lässt sich wie folgt darstellen:

1. Lade Initialisierungen und Abhängigkeiten.
2. Konfiguriere den geplanten Task
3. Rufe Benutzer ab, die zur bereitgestellt sind.
4. Iteriere über die Benutzer zur Bereitstellung:
   1. Bereite Operation und die Informationen über die Ziel-Systeme vor.
   2. Versuche die Benutzer im Ziel-System zu aktualisieren.
   3. Behandle Erfolgs- und Fehlerfälle mit passenden Logging.
5. Behandle Ausnahmen während der Benutzerbereitstellung
6. Rufe Benutzergruppen ab, die zugewiesen werden sollen.
7. Iteriere über die Benutzergruppenzuweisungen:
   1. Bereite Ziel-System Informationen vor, wie den Benutzergruppen.
   2. Versuch Benutzer zu Gruppen im korrekten Ziel-system zuzuordnen.
   3. Behandle Erfolgs- und Fehlerfälle mit passenden Logging.
8. Behandle Ausnahmen während der Benutzergruppenzuweisung
9. Bereinige fehlgeschlagene Benutzergruppenzuweisungen.
10. Bereinige fehlgeschlagenen Benutzerbereitstellungen.

Bisher wurden drei Zielanwendungen definiert, in denen Provisionierung stattfindet:

* **AWS:** Eine Cloud.
* **GitLab:** Wurde in Kapitel 2.2.9 beschrieben.
* **Atlassian:** Anwendung, in der Dokumente verwaltet werden können.

Eine jede Zielanwendung muss mit einem dieser drei Identifiern gekennzeichnet sein, damit in das jeweilige Zielsystem provisioniert wird.

Der Scheduler *UserDeprovisionScheduler* deprovisioniert jede Minute Benutzer und entfernt Benutzer aus Gruppen.

Auch die Deprovisinierung lässt sich wie folgt beschreiben:

1. Hole die Benutzerbereitstellungen, die deprovisioniert werden müssen.
2. Wenn keine Benutzerbereitstellungen gefunden werden, initialisiere die Liste als leer.
3. Protokolliere die Anzahl der Benutzerbereitstellungen, die deprovisioniert werden sollen.
4. Iteriere über die Benutzerbereitstellungen.
5. Bestimme den Vorgangstyp als "Löschen".
6. Hole die Zielsystemprovision für das aktuelle Zielsystem.
7. Protokolliere die Löschung des Benutzers im Zielsystem.
8. Hole das Zielsystem und den SCIM-Benutzer aus der Benutzerbereitstellung.
9. Lösche den Benutzer im Zielsystem und behandle den Erfolg oder Fehler.
10. Entferne Benutzer aus Gruppen.

Um in der Konsole der jeweiligen Entwicklungsumgebung Provisionierungs-, Import- und Deprovisionierungsprozesse anzuzeigen, wurde das Logging-Framework Simple Logging Facade for Java *(kurz: SLF4J)* benutzt [[34]](https://www.slf4j.org/).

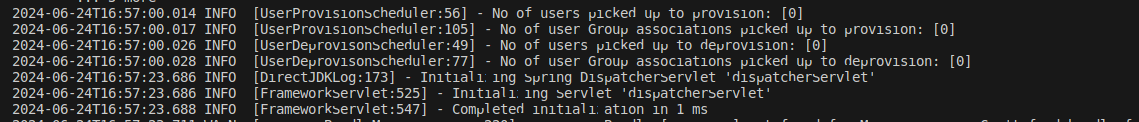


Abbildung : Ausschnitt aus der Konsole in Visual Studio Code. Provisionierungs- und Deprovisionierungsprozesse.

So durch wird der Entwickler auf Fehler in der SPE benachrichtigt und kann entsprechende Gegenmaßnahmen durchführen.

Scheduling wird durch die Konfigurationsklasse *SchedulerConfig* ermöglicht. In dieser Klasse wird ein ShedLock konfiguriert. Ein ShedLock ist eine Java-Bibliothek, welche sicherstellt, dass Aufgaben in verteilten Anwendungen nicht zugleich ausgeführt werden.

## 4.4 ERM und Datenbankberechtigungen

Im folgenden ERM werden die Beziehungen zwischen den Entitäten deutlich:

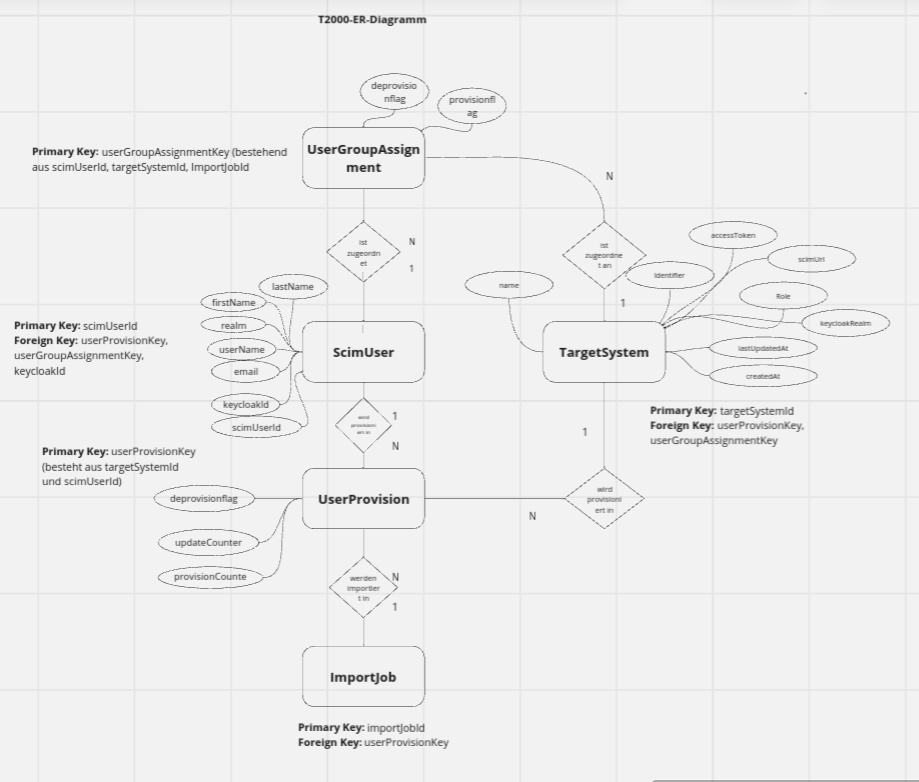


Abbildung : ERM. Veranschaulicht die Beziehungen zwischen den Entitäten der SPE.

Dies sind die Entitäten, die in der Datenbank definiert wurden. Die Entität *TargetSystem* bilden die Clients aus Keycloak ab.

Das Attribut *name* Die Tabelle *TargetSystem* ist so definiert, dass sie einzigartig ist. Dies bedeutet, dass ein Name einer Entität TargetSystem nur einmal in der Datenbank existieren darf.

Wenn es z.B. den Client mit dem Namen “Undertale” schon in der Datenbank schon gibt und man würde einen Client mit dem Namen “Undertale” nochmals in die Datenbank speichern wollen, würde die Datenbank einen Fehler werfen.

Die Tabellen werden durch XML-Dateien aktualisiert in der SPE. Ausnahmsweise werden hier keine Annotationen verwendet. XML ist ein Datenformat, das im Austausch von Daten im Internet eine wichtige Rolle spielt [[35]](https://www.w3.org/XML/).

Durch Dependency Injection werden durch die Befehle in XML, Befehle ins Datenbanksystem ausgeführt, welche die Datenbank verändern. Hierfür wurde in der SPE die Abhängigkeit Liquibase miteingebunden. Liquibase verwaltet Datenbankschemaänderungen und verarbeitet Datenbankveränderungen [[36]](https://docs.liquibase.com/concepts/introduction-to-liquibase.html).

Die veränderten Datenbankdateien werden in *Changelog-Dateien* gespeichert [[37]](https://docs.liquibase.com/concepts/changelogs/home.html). In der SPE gab es vor Umsetzung der Projektarbeit drei Changelog-Dateien.

## 4.5 CI/CD der SPE

Erneuerte Versionen der SPE wurden in GitLab bereitgestellt. In GitLab wird die SPE als ein Software-Packet verfügbar gemacht. Das Software-Packet stellt die entwickelte Anwendung als lauffähige und installierbare Datei dar. Die SPE ist in GitLab als *tgz*-Datei zu finden.

Als Paketformat wurde Helm verwendet. Helm ist ein Open-Source-Tool für Kubernetes, um Anwendungen als Software-Paket zu formatieren und zu deployen [38, S. 16].

Die SPE ist in fünf Pipeline-Phasen aufgebaut:

* **Build:** In dieser Phase wird das Projekt als Docker-Image gebildet.
* **Deploy:** Es wird überprüft, ob die SPE mit Keycloak und PostgreSQL deployed werden können.
* **Test:** In dieser Phase wird überprüft, ob die Java Unittests funktionieren (Auf Unittests wird in 2.4.2 eingegangen).
* **Security:** Es wird hier überprüft, ob die Abhängigkeiten in der Spring Security gültig und den offiziellen Standard entsprechen.
* **Release:** Die SPE wird als Helm Chart für Kubernetes vorbereitet und ist auf GitLab als herunterladbare Datei verfügbar gemacht.

In der letzten Phase, der *Release*-Phase, wird dann eine neue *Konfiguration* der SPE bereitgestellt.

# 5. Lösungsansätze

Um Multimandantenfähigkeit umzusetzen, wurde ein Konzept erdacht, bei dem man in der SPE zwei Modi integriert, welche unterschiedliche Berechtigungen und Rollen in der SPE haben.

Der Super-Admin solle demnach Zugriff auf alle Realms haben dürfen, weshalb er im master-realm als Benutzer enthalten sein muss.

Damit das Konzept der Multimandantenfähigkeit umgesetzt wird, wird ein Customer-Mode definiert, bei dem der jeweilige Benutzer nur in Realm, in dem er sich befindet, Zugriffsrechte hat. Dieser darf nicht der master-realm sein, weil sonst mehrere Zugriffsrechte erlaubt sind.

Neben den beiden Modi müssen die Provisionierungen der des Customer- und der des Super-Admin-Rolle automatisch erfolgen. Dies erfolgt durch einen technischen Admin, welcher in 2.5 präsentiert wird.

Um die SPE auf Multimandantenfähigkeit zu erweitern, wurden drei Lösungskonzepte ausgearbeitet und dargelegt. Für jede Lösung ist die SPE als *Client* registriert und es wird ein technischer Admin benötigt, welcher die Zugriffsoperationen (erstellen, löschen, aktualisieren) verwaltet. In jeder der drei Lösungen werden die lokalen Admins durch einen *IdP* eingebunden.

**1. Möglichkeit:** Die SPE ist als *Client* angemeldet und befindet sich mit dem technischen Admin in einem *Realm*, welches nicht der master-realm ist.

Jedoch ist dies in Keycloak nicht möglich, weil nur im master-realm der *Admin* in der SPE andere Realms verwalten kann.

Dies wurde getestet, indem man sich als Admin in Keycloak anmeldete und die Berechtigungen verglich, die man als *Admin* im Master-realm und in einem Nicht-Master Realm hat.

Man stellte fest, dass im Master-Realm es Rollenberechtigungen für andere, erstellte Realms gab. In einem Nicht-Master-Realm fand man jedoch nur Berechtigungen, die im eigenen Realm zulässig sind.

Die erste Möglichkeit entfällt als Lösungsansatz damit also.

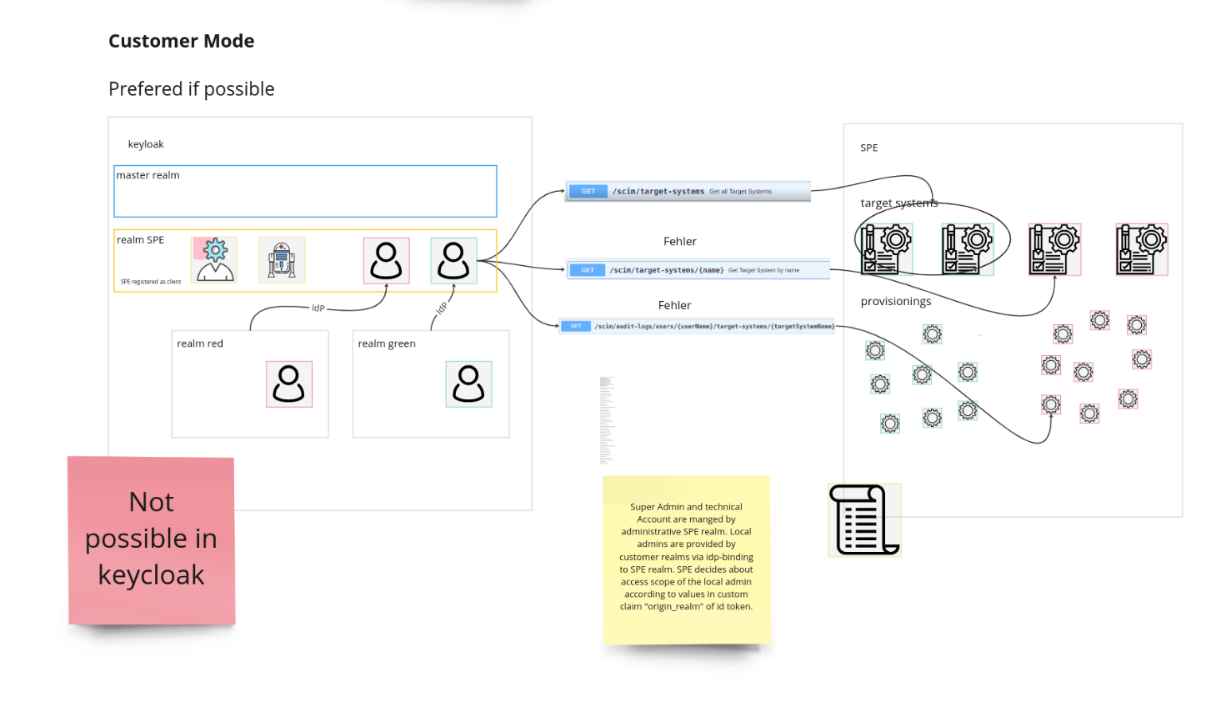


Abbildung : Erster Lösungsansatz. Diesen umzusetzen ist, ist in Keycloak nicht möglich.

**2. Möglichkeit:** Die SPE, die als Client registriert ist und der technische Admin befinden sich im master-realm, womit diese Lösung in Keycloak realisierbar ist. Diese Lösung setzt nur voraus, dass die SPE mit dem technischen Admin im master-realm sein müssen.

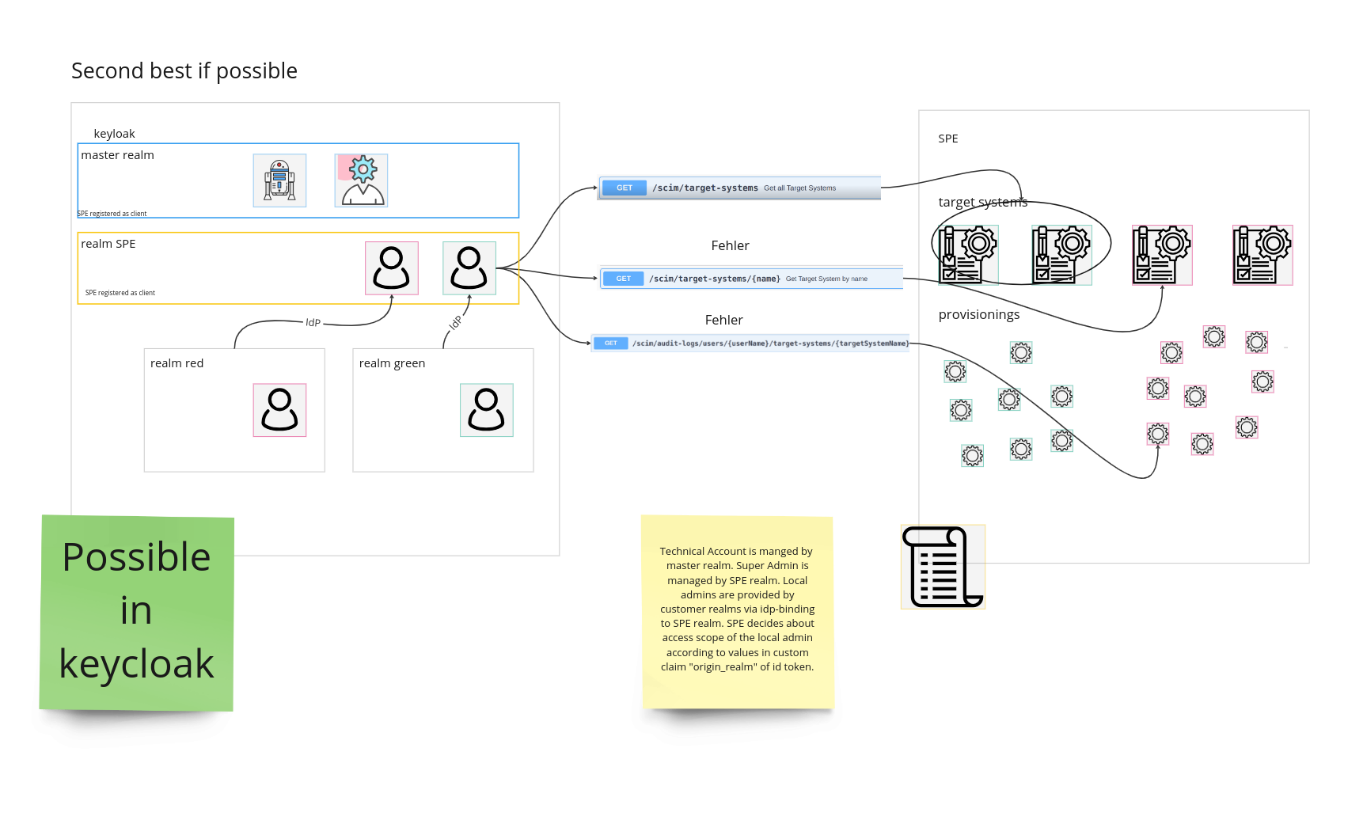


Abbildung : Zweiter Lösungsansatz. Diesen umzusetzen ist, ist in Keycloak möglich.

**3. Möglichkeit:** Der technische Admin befindet sich im master-realm, die als Client registrierte SPE in einem anderem. Diese Lösung hat den Nachteil, dass man je einem Realm eine JSON-Datei herunterladen muss. In Keycloak sind die Realms nämlich mitsamt Client und User als JSON-Datei herunterladbar, wobei alle realms dann sich in einer JSON-Datei befinden. Da in dieser jedoch mehrere JSON-Dateien erzeugt werden, wird die Anzahl an Dateien erhöht, die man mühsam verwalten muss. Es ist nicht zu empfehlen, diese Lösung umzusetzen.

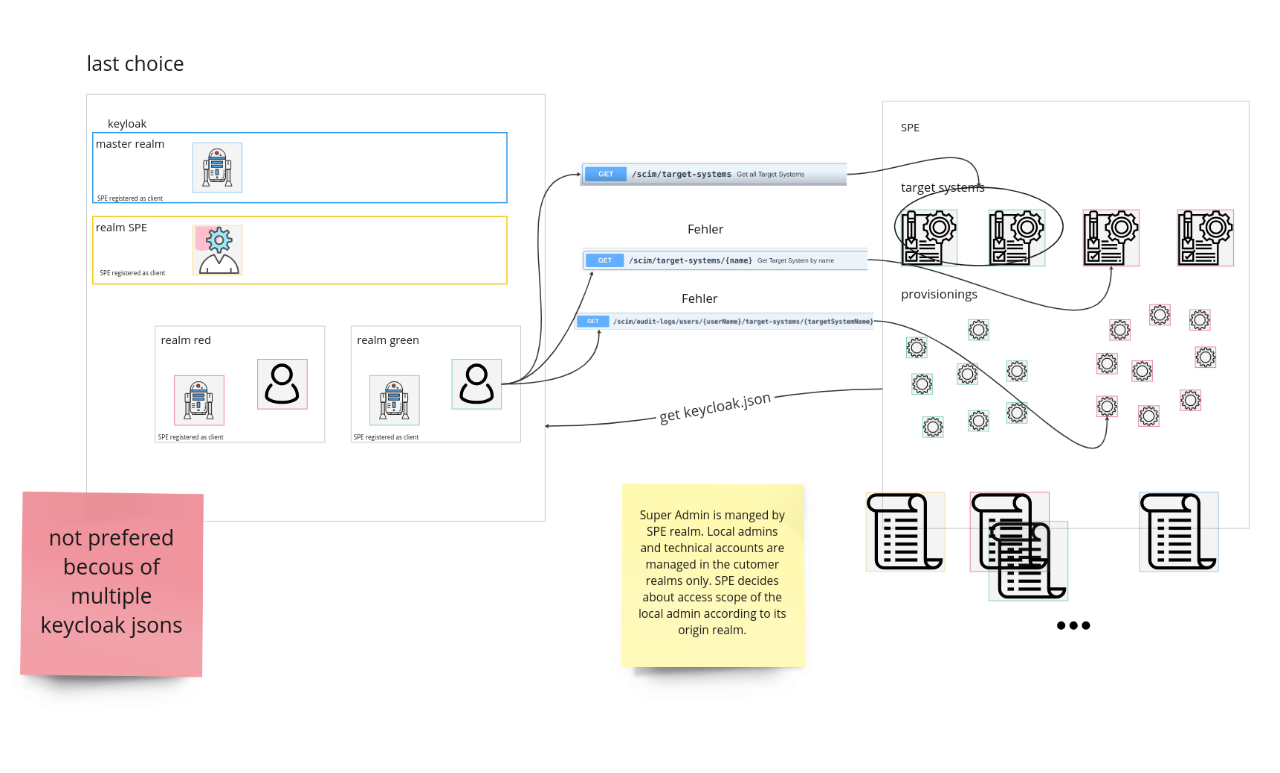


Abbildung : Dritter Lösungsansatz. Diesen umzusetzen ist, ist in Keycloak möglich, jedoch besteht der Nachteil, dass beim mehrere JSON-Dateien entstehen, wenn der Benutzer die Realms als JSON-Datei herunterladen will.

# 6. Umsetzung der Lösung

Nachdem die Lösungsansätze getestet wurden, wurde eine Reihe an User Stories in Jira angelegt:

* 1. DT-68: SuperAdminRole
  2. DT-69: CustomerAdminRole
  3. DT-70: Technischer Admin
  4. DT-71: Erkundigung zu Delivery/Deployment der SPE

Es wurde geschätzt, dass eine User Story in 2 Wochen abgeschlossen wird. Es wurde geplant, dass die 4. User Story in innerhalb einer Woche abgeschlossen sein wird.

Zu jeder User Story wird ein Unittest geschrieben. Unittests dienen in Java dazu, um Codequalität einer vollbrachten User Story, bzw. einer Aufgabe zu validieren. Tests werden in jeder User Story mitbeachtete und einprogrammiert. In der SPE gibt es für die Tests einen eigenen Testordner.

## 6.1 Die Super-Admin-Role

Bevor auf die Rolle des Super-Admins eingegangen wird, wird erläutert, wie die Einzelmandantenfähigkeit und die Mehrmandantenfähigkeit getrennt werden in der SPE.

Um beide Fähigkeiten zu trennen, wurde in der SPE in der *application-dev.yml* Datei ein Modus namens Customer-Mode angelegt.

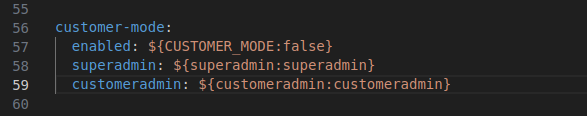


Abbildung : Ausschnitt aus der SPE; Implementierung des Customer-Modes

YAML ist wie XML eine Markup-Sprache, durch die es möglich wird, in einer YAML-Datei, wie die *application-dev.yml,* Variablen zu definieren, die in der gesamten Anwendung global verwendet werden.

In Abbildung 19 sieht man, dass ein *Boolean-Wert* gesetzt wird. Der Customer-Mode ist entweder wahr oder falsch und bestimmt, welche Operationen durch den gesetzten Wert ausgeführt werden und welche nicht. Wenn er wahr ist, geschehen alle Operationen (welche Operationen genau, wird in den nächsten Abschnitten erklärt) für den *Customer-Admin* und für den *Super-Admin* und wenn der Wert falsch ist, geschehen alle Operationen für die *Target-System-Admin-Rolle,* also für einen Benutzer im Einzelmandantensystem.

Der Super-Admin kann sich entweder im *master-realm* oder in einem anderen Realm befinden. Der Super-Admin hat die Berechtigung Clients aus verschiedenen Realms zu aktualisieren, zu erstellen oder zu löschen. Dies gelingt dadurch, dass der Super-Admin eine Rolle besitzt, die in Keycloak dies zulässt. Diese Rolle wird als *Composite Role* angelegt, wodurch der Super-Admin über mehrere *Client Roles* und *Realm Roles* verfügt.

Um zu überprüfen, wann der Benutzer die Super-Admin-Role hat, wird der Token, welcher für die Autorisierung nötig ist, im OAuth2 Autorisierungsfluss, überprüft. Dazu wurde eine neue Java-Klasse angelegt, welche den Token im Autorisierungsprozess zerlegt und im Claim *resource\_access* nach der entsprechenden Composite Role, also der Super-Admin-Rolle gesucht*.*



Abbildung : Beispiel eines dekodierten Tokens aus Keycloak.(Quelle: „RefactorFirst“, [Spring Cloud Gateway — Resource Server with Keycloak RBAC | RefactorFirst](https://refactorfirst.com/spring-cloud-gateway-keycloak-rbac-resource-server), Zugriffszeitpunkt: juni 2024)

Die Funktion, die dies überprüft ist die Funktion *hasAdminRole i*n der Klasse *AuthorizationUtil.*

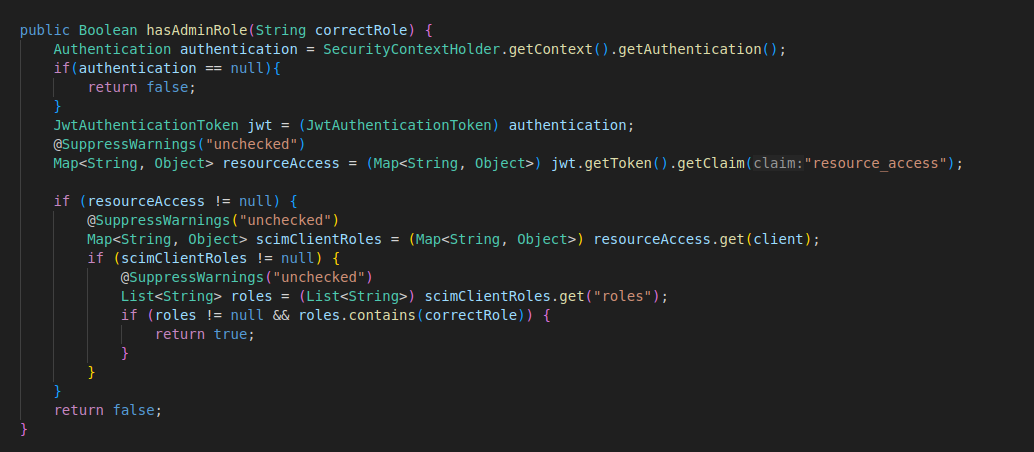


Abbildung : Die hasAdminRole - Funktion.

Als Parameter wird die Rolle übergeben, welche aus Keycloak stammt. In der *Service-Klasse TargetSystemService* wird in den jeweiligen Funktionen

1. Überprüft, ob der customer-mode wahr ist

2. Ob es sich bei dem Benutzer um einen mit der Super-Admin-Role oder mit der Customer-Admin-Role handelt.

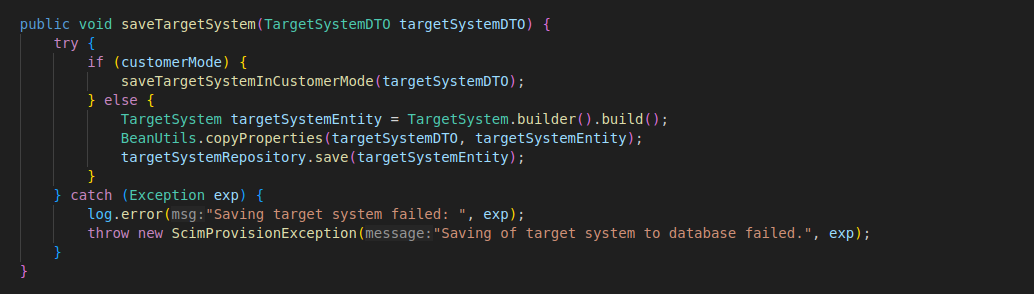


Abbildung : Als Beispiel die Funktion, welches neue Clients speichert. Hier wird zuerst geprüft, ob der Customer-Mode, die als Variable über der Klasse gesetzt ist, wahr ist.

Das Datenbankschema wurde verändert, damit die Super-Admin-Role fähig ist, auf fremde Realms zuzugreifen. In diesem Fall musste auch gewährleistet sein, dass nun Clients mit identischen Namen in verschiedenen Realms existieren dürfen und auch so in der Datenbank abgebildet werden.

Dafür wurde das Attribut *unique* in der Klasse *TargetSystem entfernt.*

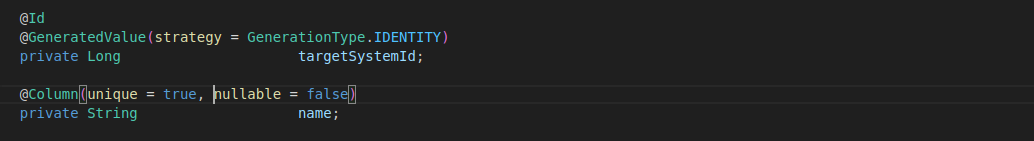


Abbildung : Vor Beginn der Projektarbeit war noch die Tabelle name unique.

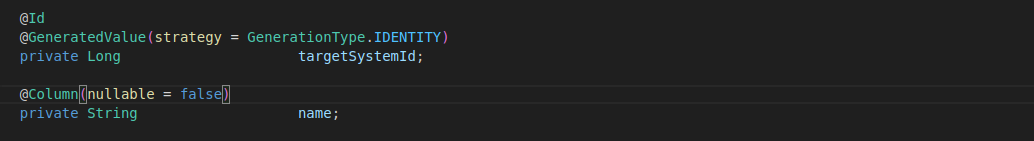


Abbildung : Die unique-kennzeichnung wird entfernt.

Zusätzlich wurde eine neue Changelog-Datei angelegt, welche die unique-kennzeichnung entfernt und das Datenbankschema anpasst. Die Endpunkte wurden dementsprechend angepasst.

Im Fall, dass es sich um einen Benutzer mit der Super-Admin-Role handelt, wird in der *Controller*-Klasse *TargetSystemController* eine andere Funktion aufgerufen, die den passenden Endpunkt enthält.

Dies war nötig umzusetzen. Wenn man sonst auf einen Client zugreift, kommt es in der Datenbank zu Fehlern. Diese Fehler teilen mit, dass es mehrere Clients mit den angeforderten Namen gibt. Es kann bspw. sein, dass es nun zwei Clients mit dem gleichen Namen gibt, z.B. *“clone\_wars”*. Mit folgenden Endpunkten wurde der Fehler gemieden:

* GET http://localhost:8080/scim/targetsystems/{name}/realms/{keycloakRealm}
* PUT http://localhost:8080/scim/targetsystems/{name}/realms/{keycloakRealm}
* POST

http://localhost:8080/scim/targetsystems/{name}/realms/{keycloakRealm}

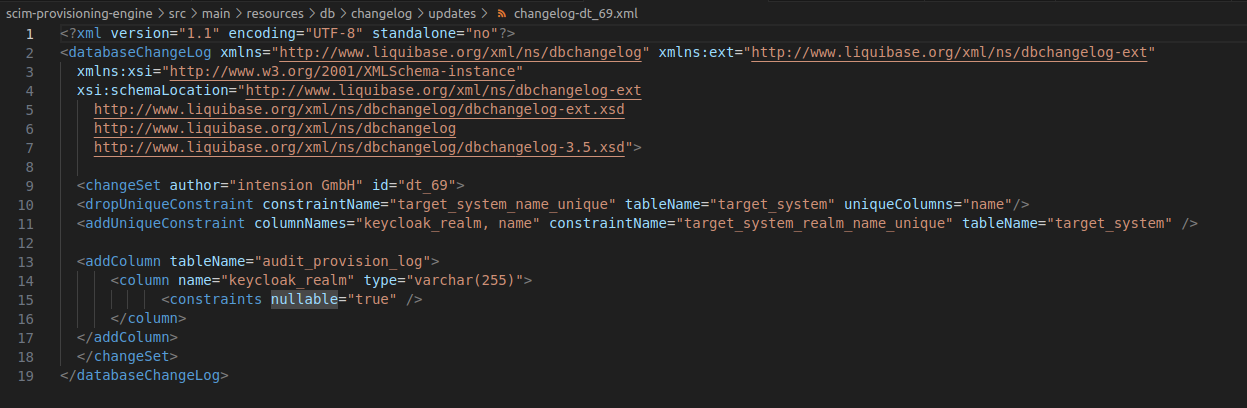


Abbildung : Die changelog-dt\_69.xml; die vierte Changelog-Datei.

## 6.2 Der Customer-Admin-Role

Die Customer-Admin-Role ist die angelegte Client Role, welche sich nicht im master-realm befinden darf. Hier hat der Benutzer mit der Customer-Admin-Role nur Zugriff auf den Client in seinem eigenem Realm.

Damit die Customer-Admin-Role angelegt wird, wird in Keycloak im Client, die der Customer-Mode-Role zugeordnet ist, im Bereich *user\_mapping* ein neues Attribut für den *Access Token* angelegt namens *realm\_scope*.

Durch den *realm\_scope* wird überprüft, ob der Benutzer mit der Customer-Admin-Role auf den Realm zugreifen will, ob er die Berechtigung dazu hat, auf den Client zuzugreifen.

Die Funktion *isClientInCorrectRealm* in der Klasse *AuthorizationUtil* überprüft, ob der Benutzer dazu berechtigt ist.

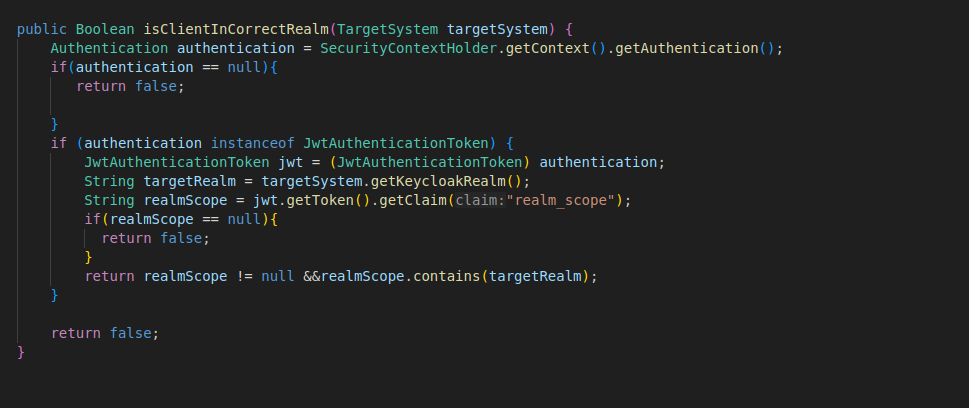


Abbildung : Die Funktion isClientInCorrectRealm.

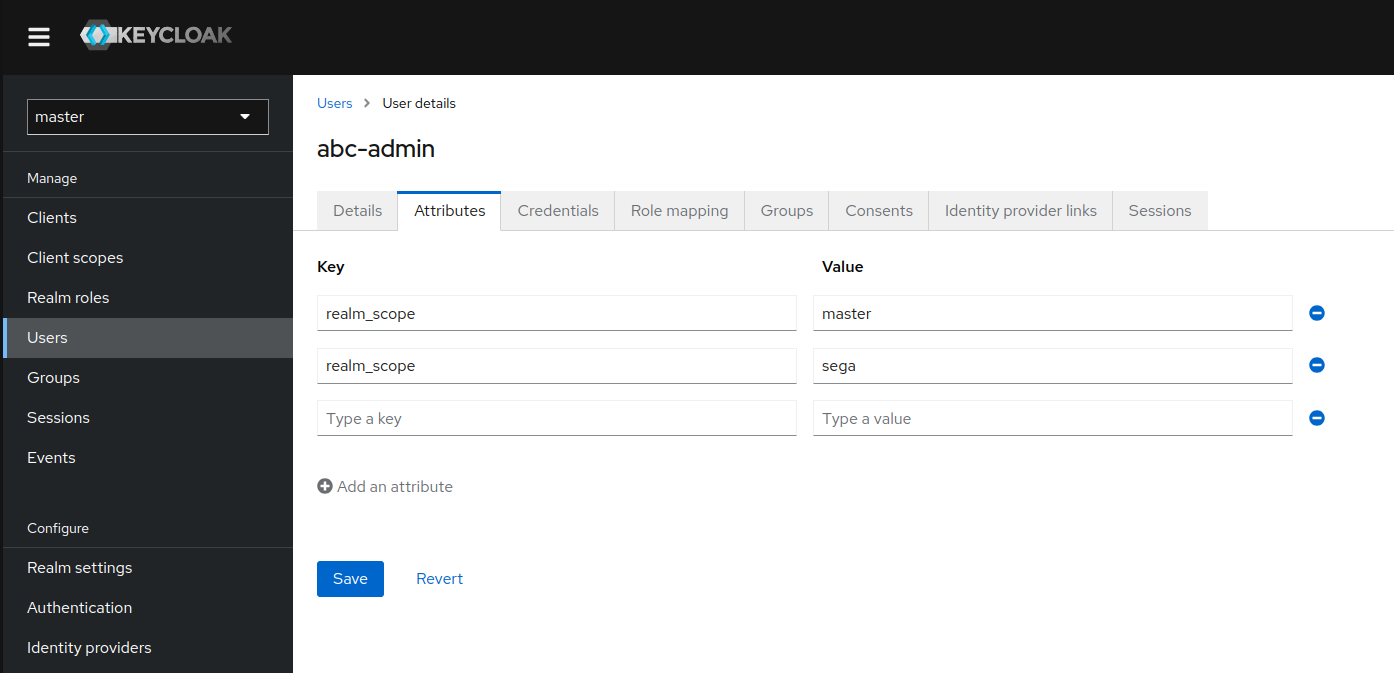


Abbildung : Anlegen eines user mappings. Ein neues Attribut namens “realm\_scope” wird eingefügt. In diesem Beispiel sind es zwei, im Falle, dass der Benutzer die Customer-Admin-Role hat, darf es nur einer sein, nämlich der Realm, in dem er sich befindet.

Dadurch, dass nun mehrere Clients mit demselben Namen in der Datenbank sind, muss im Repository nach dem Namen und nach dem Namen des Clients und nach dem Namen des Keycloak-Realms gesucht werden.

Dafür wurde im Java-Interface *TargetSystemRepository* folgende Methoden deklariert:

* TargetSystem findByNameAndKeycloakRealm(String name, String keycloakRealm);
* Long deleteByNameAndKeycloakRealm(String name, String keycloakRealm);
* TargetSystem findByKeycloakRealm(String keycloakRealm);

Der Name des Realms wird durch die Funktion *getKeycloakRealmCustomerMode* gefunden.

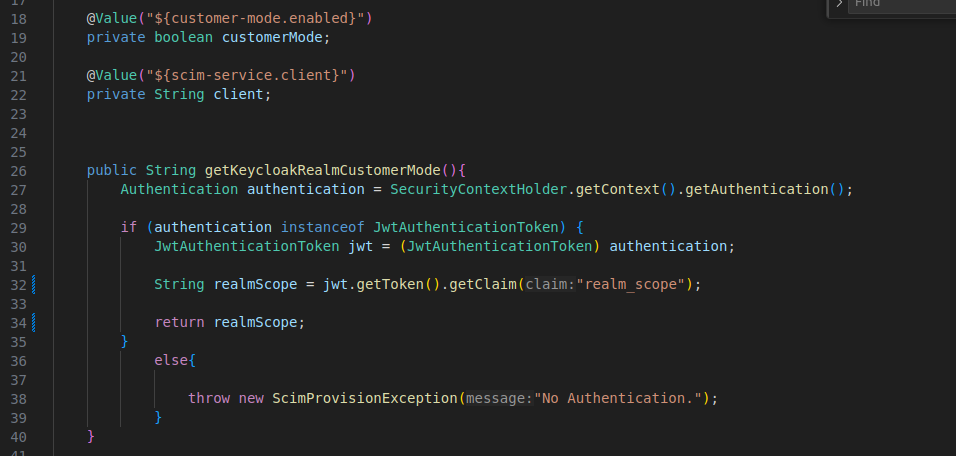


Abbildung : Die getKeycloakRealmCustomerMode findet im Access Token im Claim realm\_scope den namen des Realms, ind em sich der Client befindet, wo sich auc der Benutzer mit der Customer-Admin-Role befindet.

## 6.3 Der Technische Admin

# 7. Zusammenfassung

## 7.1 Ausblick

Die SPE wurde nur im Backend auf Multimandantenfähigkeit erweitert. Es ist wahrscheinlich, dass im Frontend ähnliche Erweiterungen im Bereich Multimandantenfähigkeit gemacht werden müssen.

Im Backend besteht noch das Potenzial, automatisierte Prozesse zu erweitern, wie bspw., dass zu jedem neu erstellten Client in der SPE automatisch ein Client in Keycloakangelegt wird.

Deploymentprozesse wurden erkundigt, aber noch nicht angewendet in der Projektarbeit, weshalb dies in späteren User Stories bearbeitet werden könnte.

## 7.2 Bewertung der Lösung

Es wollte herausgefunden werden, ob die SPE als Springboot-Projekt multimandantenfähig ist. Durch Analyse der angewandten Technologien und Testen der einzelnen Komponenten, konnte bestätigt werden, dass die SPE einzel –und multimandantenfähig ist. Dadurch, dass ein Modus integriert wurde, welches zwischen Einzel -und Multimandantensystem wechselt, ist die Anwendung in zwei Modi verfügbar. Die Lösung ist in einem Kubernetes-Cluster verfügbar und installierbar.

Die Literaturangaben bestätigen die Forschungsfrage, die durch Anwendung der *Grounded Theory* gesammelt wurden.

Dadurch das jedoch nur Unittests während der Entwicklung angewandt wurden, ist es möglich, dass nicht alle Testfälle getestet wurden. Gegebenfalls müssen noch Systemtests und Integrationstest nach Anforderungen durchgeführt werden.

Zusammengefasst lässt sich schlussfolgern, dass die SPE als Springboot-Projekt mit den Technologien SCIM und Keycloak in multimandantenfähige Systeme umwandelbar sind.

# Literaturverzeichnis

*[1] “Open-Source Identity and Access Management” [Online]. Verfügbar unter:* [*https://www.keycloak.org/*](https://www.keycloak.org/)

*[2] Scapicchio & Forrest, “Was ist Single-Sign on (SSO)?” [Online]. Verfügbar unter:* [*https://ibm.com/de-de/topics/single-sign-on">https://ibm.com/de-de/topics/single-sign-on*](https://ibm.com/de-de/topics/single-sign-on%22%3ehttps://ibm.com/de-de/topics/single-sign-on)

*[3] Chong et al., “Multi-Tenant Data Architecture”. 2006. [Online]. Verfügbar unter:* [*https://renatoargh.wordpress.com/wp-content/uploads/2018/01/article-multi-tenant-data-architecture-2006.pdf*](https://renatoargh.wordpress.com/wp-content/uploads/2018/01/article-multi-tenant-data-architecture-2006.pdf)

*[4] Glaser & Strauss. The Discovery of the Grounded Theory. [Online]. Verfügbar unter:* [*http://www.sxf.uevora.pt/wp-content/uploads/2013/03/Glaser\_1967.pdf*](http://www.sxf.uevora.pt/wp-content/uploads/2013/03/Glaser_1967.pdf)

*[5] Stol et al, “Grounded theory in software engineering research: A critical review and guidelines” [Online]. Verfügbar unter:* [*https://doi.org/10.1145/2884781.2884833*](https://doi.org/10.1145/2884781.2884833)

*[6] Thorgersen & Silva, Keycloak - Identity and Access Management for Modern Applications: Harness the power of Keycloak, OpenID Connect, and OAuth 2.0 protocols to secure applications. 2021. [Online]. Verfügbar unter:* [*https://www.google.de/books/edition/Keycloak\_Identity\_and\_Access\_Management/WBsvEAAAQBAJ?hl=de&gbpv=0*](https://www.google.de/books/edition/Keycloak_Identity_and_Access_Management/WBsvEAAAQBAJ?hl=de&gbpv=0)

*[7] “What is a REST API?” [Online]. Verfügbar unter:* [*https://www.ibm.com/topics/restapis#:~:text=REST%20APIs%20communicate%20through%20HTTP,request%20creates%20a%20new%20record*](https://www.ibm.com/topics/rest-apis#:~:text=REST%20APIs%20communicate%20through%20HTTP,request%20creates%20a%20new%20record)

*[8] “Was ist eine RESTful App?” [Online]. Verfügbar unter:* [*https://aws.amazon.com/de/what-is/restful-api/#:~:text=Systems%20that%20implement%20REST%20APIs,eliminates%20some%20client%2Dserver%20interactions*](https://aws.amazon.com/de/what-is/restful-api/#:~:text=Systems%20that%20implement%20REST%20APIs,eliminates%20some%20client%2Dserver%20interactions)

*[9] Hunt et al., “System for Cross-domain Identity Management: Core Schema. Request for Comments” [Online]. Verfügbar unter:* [*https://www.rfc-editor.org/info/rfc7643*](https://www.rfc-editor.org/info/rfc7643)

*[10] E. Hammer-Lahav, “Introduction” [Online]. Verfügbar unter:* [*https://oauth.net/about/introduction/*](https://oauth.net/about/introduction/)

*[11] “Was ist OAuth 2.0?”. [Online]. Verfügbar unter:* [*https://auth0.com/de/intro-to-iam/what-is-oauth-2*](https://auth0.com/de/intro-to-iam/what-is-oauth-2)

*[12] “What is OpenID Connect (OIDC)?” [Online]. Verfügbar unter:* [*https://auth0.com/intro-to-iam/what-is-openid-connect-oidc*](https://auth0.com/intro-to-iam/what-is-openid-connect-oidc)

*[13]* Chatterjee & Prinz, “Applying Spring Security Framework with KeyCloak-Based OAuth2 to Protect Microservice Architecture APIs: A Case Study”. 2022. [Online]. Verfügbar unter: <https://www.researchgate.net/publication/358771227_Applying_Spring_Security_Framework_with_KeyCloak-Based_OAuth2_to_Protect_Microservice_Architecture_APIs_A_Case_Study>

*[14] Johnson et al., “Professional Java Development with the Spring Framework”. [Online]. Verfügbar unter:* [*https://iamgodsom.wordpress.com/wp-content/uploads/2014/08/wrox-professional-java-development-with-the-spring-framework.pdf*](https://iamgodsom.wordpress.com/wp-content/uploads/2014/08/wrox-professional-java-development-with-the-spring-framework.pdf)

*[15] Johnson & B. Foote, “Designing Reusable Classes”. [Online]. Verfügbar unter:* [*http://www.laputan.org/drc/drc.html*](http://www.laputan.org/drc/drc.html)

*[16] “Spring Security” [Online]. Verfügbar unter:* [*https://spring.io/projects/spring-security#samples*](https://spring.io/projects/spring-security#samples)

*[17] “Servlet Authentication Architecture” [Online]. Verfügbar unter:* [*https://docs.spring.io/spring-security/reference/servlet/authentication/architecture.html#servlet-authentication-authentication*](https://docs.spring.io/spring-security/reference/servlet/authentication/architecture.html#servlet-authentication-authentication)

*[18] “Was ist PostgreSQL? Was ist PostgreSQL?” [Online]. Verfügbar unter:* [*http://postgresql.de/was-ist-postgresql*](http://postgresql.de/was-ist-postgresql)

*[19]* “The Implementation of Postgres” [Online]. Verfügbar unter: <https://www.researchgate.net/publication/3296158>

*[20] Andjelic et al., “A PERFORMANCE ANALYSIS OF THE DBMS - MySQL vs PostgreSQL” [Online]. Verfügbar unter:* [*https://www.researchgate.net/publication/296259511\_A\_performance\_analysis\_of\_the\_DBMS\_-\_MySQL\_vs\_PostgreSQL*](https://www.researchgate.net/publication/296259511_A_performance_analysis_of_the_DBMS_-_MySQL_vs_PostgreSQL)

*[21] M. Kochanowski, “*Datenbanksysteme 7. Teil – Transaktionen und ACID*”. 2023. [Online]. Verfügbar unter:* [*https://elearning.dhbw-stuttgart.de/moodle/pluginfile.php/571565/mod\_resource/content/1/Datenbanken7Transaktionen.pdf*](https://elearning.dhbw-stuttgart.de/moodle/pluginfile.php/571565/mod_resource/content/1/Datenbanken7Transaktionen.pdf)

*[22] “Configuring the Database” [Online]. Verfügbar unter:* [*https://www.keycloak.org/server/db*](https://www.keycloak.org/server/db)

*[23] “Was ist Jira?” [Online]. Verfügbar unter:* [*https://www.atlassian.com/de/software/jira/guides/getting-started/introduction#dig-into-specific-features*](https://www.atlassian.com/de/software/jira/guides/getting-started/introduction#dig-into-specific-features)

*[24] I. Bogicevic, “*Software Engineering 1 Versionsverwaltung, Tools, Programmverstehen, Code-Qualität*” [Online]. Verfügbar unter:* [*https://elearning.dhbwstuttgart.de/moodle/pluginfile.php/634308/mod\_resource/content/0/TINF22F\_VE09\_CM\_CI\_CD.pdf*](https://elearning.dhbw-stuttgart.de/moodle/pluginfile.php/634308/mod_resource/content/0/TINF22F_VE09_CM_CI_CD.pdf)

*[25] Ska, “A STUDY AND ANALYSIS OF CONTINUOUS DELIVERY, CONTINUOUS INTEGRATION IN SOFTWARE DEVELOPMENT ENVIRONMENT” . 2019. [Online]. Verfügbar unter:* [*https://www.researchgate.net/publication/354720705\_A\_STUDY\_AND\_ANALYSIS\_OF\_CONTINUOUS\_DELIVERY\_CONTINUOUS\_INTEGRATION\_IN\_SOFTWARE\_DEVELOPMENT\_ENVIRONMENT*](https://www.researchgate.net/publication/354720705_A_STUDY_AND_ANALYSIS_OF_CONTINUOUS_DELIVERY_CONTINUOUS_INTEGRATION_IN_SOFTWARE_DEVELOPMENT_ENVIRONMENT)

*[26] “Use Containers to Build, Share and Run your applications” [Online]. Verfügbar unter:* [*https://www.docker.com/resources/what-container/*](https://www.docker.com/resources/what-container/)

*[27] “Docker overview” [Online]. Verfügbar unter:* [*https://docs.docker.com/guides/docker-overview/*](https://docs.docker.com/guides/docker-overview/)

*[28] Vase, “ADVANTAGES OF DOCKER”. 2015. [Online]. Verfügbar unter:* [*https://jyx.jyu.fi/bitstream/handle/123456789/48029/URN%3aNBN%3afi%3ajyu-201512093942.pdf?sequence=1&amp;isAllowed=y*](https://jyx.jyu.fi/bitstream/handle/123456789/48029/URN%3aNBN%3afi%3ajyu-201512093942.pdf?sequence=1&amp;isAllowed=y)

*[29] “Overview” [Online]. Verfügbar unter:* [*https://kubernetes.io/docs/concepts/overview/*](https://kubernetes.io/docs/concepts/overview/)

*[30] “Viewing Pods and Nodes” [Online]. Verfügbar unter:* [*https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/*](https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/)

*[31] “Kubernetes Components” [Online]. Verfügbar unter:* [*https://kubernetes.io/docs/concepts/overview/components/*](https://kubernetes.io/docs/concepts/overview/components/)

*[32] “Scheduling”. [Online]. Verfügbar unter:* [*https://www.uni-ulm.de/in/theo/lehre/archiv/ss-2017/scheduling/*](https://www.uni-ulm.de/in/theo/lehre/archiv/ss-2017/scheduling/)

*[33] LAYLAND & LIU, “Scheduling Algorithms for Multiprogramming in a HardReal-Time Environment”. [Online]. Verfügbar unter:* [*https://dl.acm.org/doi/pdf/10.1145/321738.321743*](https://dl.acm.org/doi/pdf/10.1145/321738.321743)

*[34] “Simple Logging Facade for Java (SLF4J)”. [Online]. Verfügbar unter:* [*https://www.slf4j.org/*](https://www.slf4j.org/)

*[35] “Extensible Markup Language (XML)”. [Online]. Verfügbar unter:* [*https://www.w3.org/XML/*](https://www.w3.org/XML/)

*[36] “Introduction to Liquibase”. [Online]. Verfügbar unter:* [*https://docs.liquibase.com/concepts/introduction-to-liquibase.html*](https://docs.liquibase.com/concepts/introduction-to-liquibase.html)

*[37] “Changeglog”. [Online]. Verfügbar unter:* [*https://docs.liquibase.com/concepts/changelogs/home.html*](https://docs.liquibase.com/concepts/changelogs/home.html)

*[38] Block & Dewey, Managing Kubernetes Resources Using Helm. 2022. [Online]. Verfügbar unter:* [*https://www.google.de/books/edition/Managing\_Kubernetes\_Resources\_Using\_Helm/v02IEAAAQBAJhl=de&amp;gbpv=1&amp;dq=Managing+Kubernetes+Resources+Using+Helm&amp;printsec=frontcover*](https://www.google.de/books/edition/Managing_Kubernetes_Resources_Using_Helm/v02IEAAAQBAJ?hl=de&amp;gbpv=1&amp;dq=Managing+Kubernetes+Resources+Using+Helm&amp;printsec=frontcover)

# Abbildungsverzeichnis (muss noch genau gemacht werden)

[Abbildung 1 veranschaulicht, was der Admin im Master Realm für eine Funktion hat (Quelle: „Server Administration“, Server Administration Guide (keycloak.org), Zugriffsdatum: Juni 2024) 6](#_Toc170488529)

[Abbildung 2: Neuen User in Keycloak erstellen (Quelle: „Keycloak“, OpenJDK - Keycloak, Zugriffsdatum: Juni 2024) 7](#_Toc170488530)

[Abbildung 3: Vereinfachte Darstellung der SCIM-Struktur 11](#_Toc170488531)

[Abbildung 4: Ressourcenstruktur in SCIM (Quelle: "SCIM“, https://simplecloud.info/, Zugriffsdatum: Juni 2024) 12](#_Toc170488532)

[Abbildung 5: Rollen und Authorisationsprozess in OAuth2 (Quelle:“Medium“, #SpringSecurity:Oauth 2.0 Roles | Medium, Zugriffsdatum: Juni 2024) 14](#_Toc170488533)

[Abbildung 6: Beispiel, wie Annotation verwendet werden: @Entity kennzeichnet die Klasse als Datenbanktabelle und sorgt dafür, dass sie als solche in der Datenbank abgebildet wird 17](#_Toc170488534)

[Abbildung 7: Vereinfachte Darstellung der Schichten in Springboot (Quelle: ) 18](#_Toc170488535)

[Abbildung 8: Vergleich von Container-basierten Anwendungen in Docker und Virtuellen Maschinen (Quelle:“Docker“, What is a Container? | Docker, Zugriffsdatum: Juni 2024) 22](#_Toc170488536)

[Abbildung 9: Vereinfachte Darstellung, wie Pods und Nodes zusammenhängen (Quelle:“Kubernetes“, Viewing Pods and Nodes | Kubernetes, Zugriffszeitpunkt: Juni 2024) 24](#_Toc170488537)

[Abbildung 10: Ausschnitt aus der pom.xml aus der SPE 25](#_Toc170488538)

[Abbildung 11: Ausschnitt aus der docker-compose.yml. Daten zum image, in dem sich keycloak befindet, ist angegeben. 26](#_Toc170488539)

[Abbildung 12: Abschnitt aus der application-dev.yml-Datei. Hier werden die Variablen als Platzhalter für die eigentlichen Namen der Attribute des angelegten keycloak-client aus Keycloak gesetzt. 27](#_Toc170488540)

[Abbildung 13: Wie Abb. 9, nur mit dem keycloak-client. Dieser muss im master angelegt werden. 28](#_Toc170488541)

[Abbildung 14: Endpunkte, um auf die jeweiligen JSON-Objekte Operationen auszuführen. 28](#_Toc170488542)

[Abbildung 15: Ausschnitt aus der Konsole in Visual Studio Code. Provisionierungs- und Deprovisionierungsprozesse. 31](#_Toc170488543)

[Abbildung 16: ERM. Veranschaulicht die Beziehungen zwischen den Entitäten der SPE. 33](#_Toc170488544)

[Abbildung 17: Erster Lösungsansatz. Diesen umzusetzen ist, ist in Keycloak nicht möglich. 36](#_Toc170488545)

[Abbildung 18: Zweiter Lösungsansatz. Diesen umzusetzen ist, ist in Keycloak möglich. 37](#_Toc170488546)

[Abbildung 19: Dritter Lösungsansatz. Diesen umzusetzen ist, ist in Keycloak möglich, jedoch besteht der Nachteil, dass beim mehrere JSON-Dateien entstehen, wenn der Benutzer die Realms als JSON-Datei herunterladen will. 38](#_Toc170488547)

[Abbildung 20: Ausschnitt aus der SPE; Implementierung des Customer-Modes 39](#_Toc170488548)

[Abbildung 21: Beispiel eines dekodierten Tokens aus Keycloak.(Quelle: „RefactorFirst“, Spring Cloud Gateway — Resource Server with Keycloak RBAC | RefactorFirst, Zugriffszeitpunkt: juni 2024) 40](#_Toc170488549)

[Abbildung 22: Die hasAdminRole - Funktion. 41](#_Toc170488550)

[Abbildung 23: Als Beispiel die Funktion, welches neue Clients speichert. Hier wird zuerst geprüft, ob der Customer-Mode, die als Variable über der Klasse gesetzt ist, wahr ist. 41](#_Toc170488551)

[Abbildung 24: Vor Beginn der Projektarbeit war noch die Tabelle name unique. 42](#_Toc170488552)

[Abbildung 25: Die unique-kennzeichnung wird entfernt. 42](#_Toc170488553)

[Abbildung 26: Die changelog-dt\_69.xml; die vierte Changelog-Datei. 43](#_Toc170488554)

[Abbildung 27: Die Funktion isClientInCorrectRealm. 44](#_Toc170488555)

[Abbildung 28: Anlegen eines user mappings. Ein neues Attribut namens “realm\_scope” wird eingefügt. In diesem Beispiel sind es zwei, im Falle, dass der Benutzer die Customer-Admin-Role hat, darf es nur einer sein, nämlich der Realm, in dem er sich befindet. 44](#_Toc170488556)

[Abbildung 29: Die getKeycloakRealmCustomerMode findet im Access Token im Claim realm\_scope den namen des Realms, ind em sich der Client befindet, wo sich auc der Benutzer mit der Customer-Admin-Role befindet. 45](#_Toc170488557)

1. In dieser Arbeit wird auf das Gendern verzichtet, aufgrund des angewöhnten Schreibstils.

   Im Maskulin werden hier die weibliche, diverse sowie männliche Formen miteinbezogen [↑](#footnote-ref-2)
2. Hier wird hauptsächlich die Keycloak Dokumentation verwendet, um die Terminologien zu beschreiben:

   *”Server Administration Guide” [Online]. Verfügbar unter:* [*https://www.keycloak.org/docs/23.0.0/server\_admin/*](https://www.keycloak.org/docs/latest/server_admin/).

   Es konnten keine zielführenderen Informationen zu dieser Thematik recherchiert werden [↑](#footnote-ref-3)