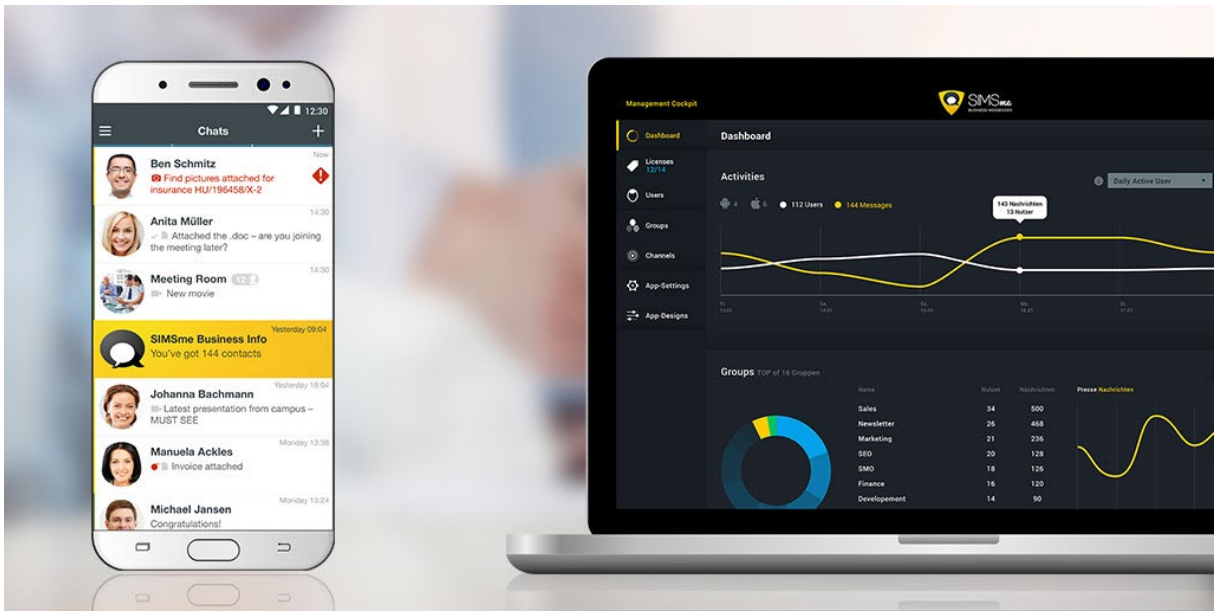
ginlo Business LDAP-Schnittstellenbeschreibung

Version 1.0 – Juni 2019



Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einleitung

Die vorliegende Schnittstellenbeschreibung richtet sich an Entwickler, um sie in die Lage zu versetzen, Nutzerdaten für ginlo automatisiert zur Verfügung zu stellen. ginlo Business stellt hierfür eine CSV-basierte Schnittstelle zum Import von Nutzerdaten zur Verfügung. Die Schnittstelle ermöglicht, Nutzer automatisiert anzulegen, zu ändern und zu löschen. Weiterhin können Nutzer auf diesem Wege ginlo-Gruppen und ginlo-Kanälen zugeordnet werden. Existieren die jeweiligen Gruppen oder Kanäle noch nicht, werden diese ebenfalls im Zuge des Imports angelegt.

In diesem Dokument wird das Format der CSV-Datenstruktur und der dazugehörigen Constraints beschrieben. Sie ist die Grundlage zur Realisierung eines Export-Adapters von Nutzerdaten. Weiterhin gibt es eine beispielhafte Implementierung eines LDAP-Adapters zum Export von Nutzerdaten aus OpenLDAP bzw. Microsoft AD LDS. Die beispielhafte Implementierung kann als Grundlage zur Realisierung eines LDAP-Adapters verwendet werden. Unabhängig von LDAP-basierten Systemen kann auch jedes andere System als Quellsystem verwendet werden, wenn die bereitgestellten Nutzerdaten der hier definierten CSV-Datenstruktur entsprechen.

Weiterhin behandelt dieses Dokument die Beschreibung eines automatischen LDAP-Imports. Mittels einer REST-Schnittstelle können die Daten des LDAP-Adapters in das Management Cockpit aufgenommen werden. Eine explizite Beschreibung der Schnittstelle wird mit Hilfe einer sogenannten Swagger-Datei realisiert. Diese Beschreibung ermöglicht es, einen Client automatisch zu generieren und für seine Bedürfnisse anzupassen.

Im Weiteren finden sich die folgenden Informationen:

- Im Kapitel 2 befindet sich die detaillierte Beschreibung der CSV-Datenstruktur. Sie ist für die Entwicklung eines Export-Adapters relevant.
- Im Kapitel 3.6.1 wird beschrieben, wie der LDAP-Server für den Adapter zu konfigurieren ist.
- Im Kapitel 3.6.2 wird das Mapping von LDAP-Feldern auf die Datenstruktur von ginlo-Nutzer definiert.
- Kapitel 4 beschreibt technisch die REST-Schnittstelle für den Import von LDAP-Daten und zeigt an einer beispielhaften Implementierung, wie diese zu verwenden ist.

Dieser Abschnitt zeigt den Aufbau der Gesamtstruktur und stellt das Zusammenspiel der einzelnen Komponenten dar. Der LDAP-Adapter wurde dazu entwickelt, Daten aus einem Microsoft Active Directory auszulesen und im CSV-Format bereitzustellen. Die explizite Struktur des Formats ist in Kapitel 2.1 beschrieben. Die durch den LDAP-Adapter erzeugte Datei bietet die Grundlage für den LDAP-Importer, welcher über die durch ginlo bereitgestellte REST-Schnittstelle mit dem Service des Management Cockpits kommunizieren und Nutzer, Gruppen, Kanäle und Keywords anlegen und aktualisieren kann sowie Zuordnungen dieser zu den Nutzern realisieren kann. Das Ergebnis dieses Zusammenspiels stellt die direkte Abbildung des Inhalts des Active Directories im ginlo Management Cockpit dar. Die nachfolgende Darstellung zeigt den Prozess visuell auf.

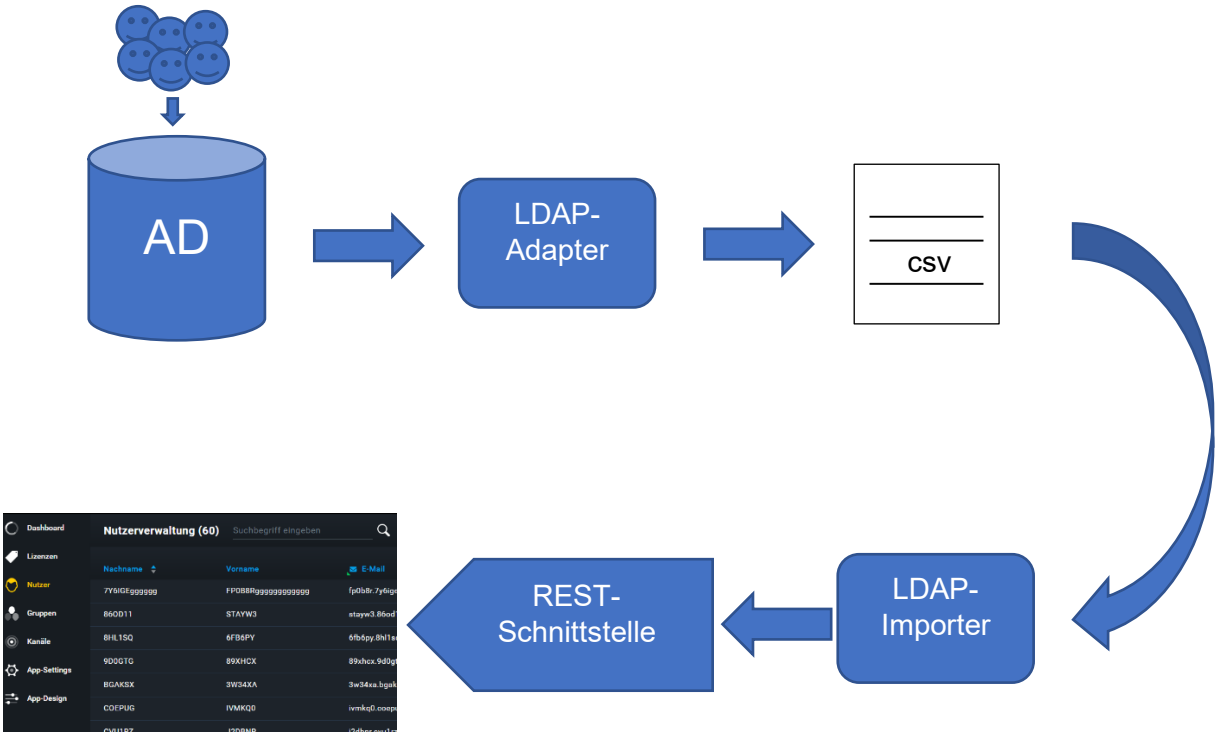


Abbildung 1: Infrastruktureller Aufbau

2 CSV-Datenstruktur

Für den Import von Nutzern in das ginlo Management Cockpit muss eine CSV-Datei mit den Nutzerinformationen erstellt werden. Dazu muss die CSV-Datei UTF8-kodiert sein und dem folgenden Format entsprechen:

"ACTION";"CREATED";"MODIFIED";"USER_ID";"DEPARTMENT";"PHONE";"EMAIL";"LAST_NAME";"FIRST_NAME";"KEYWORDS";"CHANNELS";"GROUPS"
"INSERT";"2018-02-27T03:58:21";"2018-02-27T03:58:21";"BA9170A1341038489BED9BE23C0E11F4";"IT";"+4991634000002";"max.mustermann-002@myldap.localhost";"Max Mustermann-002";"Max";"myldap GmbH, Verwaltung, IT";"myldap company channel, myldap it channel, myldap employee channel";"myldap it group"

In der ersten Zeile der CSV-Datei müssen die obigen Feldbezeichnungen vollständig vorhanden und ausschließlich mit Semikolon (ohne zusätzliche Whitespaces) getrennt sein. Weiterhin müssen die Feldbezeichnungen in Anführungszeichen gesetzt sein.

Jede nachfolgende Zeile muss die Daten eines Nutzers enthalten. Auch die Nutzerdaten müssen in Anführungszeichen gesetzt und ausschließlich durch Semikolon (ohne zusätzliche Whitespaces) getrennt sein. Für den Zeilenumbruch ist ein Linefeed (\n) zu verwenden.

In Textfeldern dürfen die folgenden Zeichen nicht verwendet werden:

; " < > { } &

Leere Zeilen werden beim Import ignoriert.

Die verwendeten Felder innerhalb der CSV-Datenstruktur sind in der untenstehenden Tabelle im Überblick dargestellt. Im nachfolgenden Kapitel werden die Felder und deren Constraints detailliert beschrieben.

Attribut	Beschreibung
ACTION	Beschreibt, ob es sich um eine Neuanlage, Modifikation oder Löschung handelt. Gültige Werte sind: INSERT, UPDATE, DELETE
CREATED	Datum der Nutzeranlage im LDAP-Provider. Format: <YYYY>-<MM>-<DD>T<HH>:<MM>:<SS>
MODIFIED	Datum der letzten Modifikation des Nutzers im LDAP-Provider Format: <YYYY>-<MM>-<DD>T<HH>:<MM>:<SS>
USER_ID	ID des Nutzers im LDAP-Provider. Das Format ist durch den jeweiligen LDAP-Provider definiert.
DEPARTMENT	Abteilung des Nutzers
PHONE	Telefonnummer des Nutzers
EMAIL	Mailadresse des Nutzers
LAST_NAME	Nachname des Nutzers
FIRST_NAME	Vorname des Nutzers
KEYWORDS	Für den Nutzer im LDAP-Provider hinterlegte ginlo Stichwörter. Format: kommaseparierte Liste
CHANNELS	Für den Nutzer im LDAP-Provider hinterlegte Kanäle. Format: kommaseparierte Liste
GROUPS	Für den Nutzer im LDAP-Provider hinterlegte Gruppen. Format: kommaseparierte Liste

Tabelle 1: CSV-Datenstruktur

2.1 Feldbeschreibung

2.1.1 ACTION

Definiert, ob ein Nutzer angelegt, geändert oder gelöscht werden soll. Zulässige Werte sind INSERT, UPDATE und DELETE. Im Falle, dass ein Nutzer mit der Aktion

- INSERT angelegt werden soll und bereits existiert, wird die Aktion als UPDATE interpretiert.
- UPDATE angelegt werden soll und noch nicht existiert, wird die Aktion als INSERT interpretiert. Sind die zu aktualisierenden Daten identisch mit den Einträgen in der Importdatei wird die Aktion ignoriert und protokolliert.
- DELETE gelöscht werden soll und nicht existiert, wird die Aktion ignoriert und protokolliert.

Ein leerer Wert ist nicht zulässig. INSERT und UPDATE verhalten sich inhaltlich gleich. Lediglich die Meldung im erzeugten Report unterscheidet sich, je nachdem, ob der Datensatz vor dem Import vorhanden war oder nicht. Dieses Verhalten kann genutzt werden, um anhand der Protokolldatei einen Sanity Check durchzuführen.

2.1.2 CREATED

Definiert den Anlagezeitpunkt des Nutzers im Quellsystem. Der Zeitstempel muss dem Format <YYYY>-<MM>-<DD>T<HH>-<MM>-<SS> entsprechen. Aktuell wird das Feld im Importprozess ignoriert.

Ein leerer Wert ist zulässig.

2.1.3 MODIFIED

Definiert den Änderungszeitpunkt des Nutzers im Quellsystem. Der Zeitstempel muss dem Format <YYYY>-<MM>-<DD>T<HH>-<MM>-<SS> entsprechen. Aktuell wird das Feld im Importprozess ignoriert.

Ein leerer Wert ist zulässig.

2.1.4 USER_ID

Ist die ID des Nutzers im Quellsystem. Die USER_ID wird als externe ID am ginlo-Nutzer gespeichert.

Der Inhalt des Feldes muss eindeutig und darf nicht mehreren Nutzern zugeordnet sein. Wird eine Mehrdeutigkeit identifiziert, werden die Daten des Nutzers nicht verarbeitet. Die maximale Länge der USER_ID beträgt 100 Zeichen. Zulässig sind alle druckbaren ASCII-Zeichen. Ausgenommen sind die Zeichen < und >.

2.1.5 DEPARTMENT

Definiert die Organisationseinheit, dem der Nutzer im Quellsystem zugeordnet ist. Die Organisationseinheit wird am ginlo Nutzer als Abteilung hinterlegt.

Ein leerer Wert ist zulässig.

2.1.6 PHONE

Definiert die Mobilfunknummer des Nutzers. Die Mobilfunknummer muss einem der beiden folgenden Formate entsprechen:

- +<Ländervorwahl><Nationale Vorwahl><Anschlusskennung>
- <Nationale Vorwahl><Anschlusskennung>

Das erste entspricht dabei dem E.164-Standard. Es muss eine gültige Mobilfunknummer übergeben werden, die maximal 15 Zeichen umfassen darf. Die Mobilfunknummer wird mit dem ginlo-Nutzer gespeichert. Wurde keine Ländervorwahl angegeben, so wird die Vorwahl +49 angenommen.

Ein leerer Wert ist zulässig. Der Inhalt des nicht-leeren Feldes muss eindeutig und darf nicht mehreren Nutzern zugeordnet sein. Wird eine Mehrdeutigkeit identifiziert, werden die Daten des Nutzers nicht verarbeitet.

2.1.7 EMAIL

Muss eine gültige E-Mail-Adresse sein. Die E-Mail-Adresse wird mit dem ginlo-Nutzer gespeichert.

Ein leerer Wert ist zulässig. Der Inhalt des nicht-leeren Feldes muss eindeutig und darf nicht mehreren Nutzern zugeordnet sein. Wird eine Mehrdeutigkeit identifiziert, werden die Daten des Nutzers nicht verarbeitet.

2.1.8 LAST_NAME

Definiert den Nachnamen des Nutzers und darf maximal eine Länge von 30 Zeichen aufweisen. Der Nachname wird am ginlo-Nutzer gespeichert.

Zulässig sind alle druckbaren ASCII-Zeichen. Ausgenommen sind die Zeichen < und > und ein leerer String.

2.1.9 FIRST_NAME

Definiert den Vornamen des Nutzers und darf maximal eine Länge von 30 Zeichen aufweisen. Der Vorname wird mit dem ginlo-Nutzer gespeichert.

Zulässig sind alle druckbaren ASCII-Zeichen. Ausgenommen sind die Zeichen < und > und ein leerer String.

2.1.10 KEYWORDS

Im Feld KEYWORDS werden ginlo-spezifische Schlagworte gespeichert, die der Organisation der Nutzer im ginlo Management Cockpit dienen. Die Schlagworte müssen im Feld als kommaseparierte Liste vorliegen. Der Feldinhalt darf maximal eine Länge von 100 Zeichen aufweisen. Die Schlagworte werden am ginlo-Nutzer gespeichert.

Zulässig sind alle druckbaren ASCII-Zeichen. Ausgenommen sind die Zeichen < und > und ein leerer String.

2.1.11 CHANNELS

Im Feld CHANNELS werden ginlo-spezifische Kanäle gespeichert, denen der Nutzer beim Import automatisch zugeordnet wird. Existiert ein Kanal zum Importzeitpunkt nicht, wird der Kanal automatisch angelegt. Ist ein dem Nutzer zugeordneter Kanal im Feld nicht länger enthalten, wird die Zuordnung des Nutzers zum Kanal im Kontext des Imports gelöscht. Die Kanäle müssen im Feld als kommaseparierte Liste vorliegen. Der Feldinhalt darf maximal eine Länge von 100 Zeichen aufweisen. Die Kanäle werden am ginlo-Nutzer gespeichert.

Ein leerer Wert ist zulässig.

2.1.12 GROUPS

Im Feld GROUPS werden ginlo-spezifische Gruppen gespeichert, denen der Nutzer beim Import automatisch zugeordnet wird. Existiert eine Gruppe zum Importzeitpunkt nicht, wird die Gruppe automatisch angelegt. Ist eine dem Nutzer zugeordnete Gruppe im Feld nicht länger enthalten, wird die Zuordnung des Nutzers zur Gruppe im Kontext des Imports gelöscht. Die Gruppen müssen im Feld als kommaseparierte Liste vorliegen. Der Feldinhalt darf maximal eine Länge von 100 Zeichen aufweisen. Die Gruppen werden am ginlo-Nutzer gespeichert.

Ein leerer Wert ist zulässig.

2.2 Fehlerübersicht

Beim Import können eine Reihe von verschiedenen Fehlern auftreten. Je nach Schwere des Fehlers werden diese in die Fehlerkategorie INFO, WARNING, ERROR und FATAL_ERROR zugeordnet.

Fehlerkategorie	Beschreibung
INFO	Es ist kein tatsächlicher Fehler aufgetreten (lediglich eine Information für den Administrator).
WARNING	Es ist ein automatisch behebbarer Fehler auf Datensatzebene aufgetreten.
ERROR	Es ist ein nicht automatisch behebbarer Fehler aufgetreten. Der Datensatz wird übersprungen, der Import geht weiter.
FATAL_ERROR	Es ist ein nicht automatisch behebbarer Fehler aufgetreten. Der Import wird abgebrochen.

Folgende Klassifizierung der Fehler ist umzusetzen:

ID	Fehler	Kategorie	Regeln zur automatischen Fehlerbehebung bei WARNING
IMP-1	Importdatei zu groß (max. 10 MB)	FATAL_ERROR	
IMP-2	Leere Zeile	ERROR	
IMP-3	Fehlendes Pflichtfeld ACTION	ERROR	
IMP-4	Fehlendes Pflichtfeld USER_ID	ERROR	
IMP-5	Fehlendes Pflichtfeld LAST_NAME	ERROR	
IMP-6	Fehlendes Pflichtfeld FIRST_NAME	ERROR	
IMP-7	Doppelte USER_ID im Import	ERROR	(Hinweis: Nur der erste Import zählt, jedes weitere Vorkommen wäre dann ein ERROR).
IMP-8	Telefonnummer bereits benutzt	INFO	Keine Sonderbehandlung, sondern bestehendes Verfahren bei einer Registrierung mit einer bereits vorhandenen Tel.Nr (Kanal aktivieren etc.)
IMP-9	E-Mail-Adresse schon benutzt	INFO	Analog zu IMP-8 - nur mit Mail
IMP-10	Freemailer – E-Mail-Adresse	ERROR	
IMP-11	Längenprüfung für LAST_NAME fehlgeschlagen	WARNING	Abschneiden auf erlaubte Länge
IMP-12	Längenprüfung für FIRST_NAME fehlgeschlagen	WARNING	Abschneiden auf erlaubte Länge

ID	Fehler	Kategorie	Regeln zur automatischen Fehlerbehebung bei WARNING
IMP-13	Längenprüfung für USER_ID fehlgeschlagen	ERROR	
IMP-14	Längenprüfung für KEYWORDS fehlgeschlagen	WARNING	Abschneiden auf erlaubte Länge
IMP-15	Längenprüfung für GROUPS fehlgeschlagen	WARNING	Abschneiden auf erlaubte Länge
IMP-16	Längenprüfung für CHANNELS fehlgeschlagen	WARNING	Abschneiden auf erlaubte Länge
IMP-17	Syntaktisch ungültige E-Mail-Adresse	ERROR	
IMP-18	Syntaktisch ungültige Telefonnummer	ERROR	
IMP-19	Fehlende E-Mail-Adresse + Telefonnummer (keiner der beiden Werte ist gesetzt)	ERROR	
IMP-20	Fehlerhaftes Datumsformat CREATED	WARNING	Feld wird ignoriert
IMP-21	Fehlerhaftes Datumsformat MODIFIED	WARNING	Feld wird ignoriert
IMP-22	Keine Lizenz zum Zuordnen vorhanden	WARNING	Dem Nutzer wird keine Lizenz zugewiesen
IMP-23	Kommunikation mit ginlo fehlgeschlagen	FATAL_ERROR	
IMP-24	Fehlerhafte ACTION Insert	WARNING	Insert wird zu Update geändert
IMP-25	Fehlerhafte ACTION Update	WARNING	Update wird zu Insert geändert
IMP-26	Fehlerhafte ACTION Delete	ERROR	
IMP-27	Ungültige Zeichen in LAST_NAME, FIRST_NAME, USER_ID	ERROR	
IMP-28	Ungültige Zeichen in FIRST_NAME	ERROR	
IMP-29	Ungültige Zeichen in USERID	ERROR	
IMP-30	Falsches Format Importdatei (bspw. falsche Anzahl Spalten, …)	FATAL_ERROR	
IMP-31	Telefonnummer doppelt in der Importdatei	ERROR	
IMP-32	E-Mail-Adresse doppelt in der Importdatei	ERROR	
IMP-33	Unbekannte ACTION	ERROR	Zeile wird nicht importiert

ID	Fehler	Kategorie	Regeln zur automatischen Fehlerbehebung bei WARNING
IMP-34	Datensatz nicht akzeptiert bzw. ein unerwarteter Fehler ist aufgetreten	FATAL_ERROR	
IMP-35	Zugeordneter Gruppenname ist mehrfach vergeben	FATAL_ERROR	
IMP-36	Zugeordneter Kanalname ist mehrfach vergeben	FATAL_ERROR	

Tabelle 2: Fehlerübersicht

2.3 CSV-Beispiele

2.3.1 Exportbeispiel für Nutzeranlage

In dem nachfolgenden Beispiel repräsentiert die CSV-Exportdatei die Anlage von drei neuen Nutzern, die in ginlo angelegt werden sollen.

```
"ACTION";"CREATED";"MODIFIED";"USER_ID";"DEPARTMENT";"PHONE";"EMAIL";"LAST_NAME";"FIRST_NAME";"KEYWORDS";"CHANNELS";"GROUPS"
"INSERT";"2018-01-27T03:58:21";"2018-02-12T03:58:21";"EF3451234";"IT";"+491627563452";"tina.schuber@mail.de";"Schuber";"Tina";"";"Sport, Urlaub";"DevOps"

"INSERT";"2018-02-27T03:58:21";"2018-02-27T03:58:21";"AFG236464";"IT";"+491717826732";"teresa.lamau@mail.de";"Lamau";"Teresa";"Team-Lead";"Team-Leads";"News"

"INSERT";"2018-02-27T03:58:21";"2018-02-27T03:58:21";"672777777";"IT";"+491516173647";"simon.terfon@mail.de";"Terfon";"Simon";"";"Sport";"News"
```

2.3.2 Exportbeispiel für Nutzeränderung

In dem nachfolgenden Beispiel werden zwei der zuvor angelegten Nutzer verändert (Nutzerin „Lamau“ wechselt in Gruppe „Telco“, Nutzer „Simon“ ändert die Rufnummer) sowie ein weiterer Nutzer hinzugefügt.

```
"ACTION";"CREATED";"MODIFIED";"USER_ID";"DEPARTMENT";"PHONE";"EMAIL";"LAST_NAME";"FIRST_NAME";"KEYWORDS";"CHANNELS";"GROUPS"
"UPDATE";"2018-02-27T03:58:21";"2018-02-27T03:58:21";"AFG236464";"IT";"+491717826732";"teresa.lamau@mail.de";"Lamau";"Teresa";"Team-Lead";"Team-Leads";"Telco"

"UPDATE";"2018-02-27T03:58:21";"2018-02-27T03:58:21";"672777777";"IT";"+491832457261";"simon.terfon@mail.de";"Terfon";"Simon";"";"Sport";"News"

"INSERT";"2018-02-27T03:58:21";"2018-02-27T03:58:21";"648728384";"Business";"+491613458725";"norbert.meier@mail.de";"Meier";"Norbert";"";"Unternehmensnachrichten, Vertrieb";"Telco, News"
```

2.3.3 Exportbeispiel für Nutzerlöschung

In dem nachfolgenden Beispiel werden zwei der zuvor angelegten Nutzer wieder gelöscht (Nutzerin „Lamau“, Nutzer „Terfon“).

ACTION";"CREATED";"MODIFIED";"USER_ID";"DEPARTMENT";"PHONE";"EMAIL";"LAST_NAME";"FIRST_NAME";"KEYWORDS";"CHANNELS";"GROUPS"
"DELETE";"2018-02-27T03:58:21";"2018-02-27T03:58:21";"AFG236464";"IT";"+491717826732";"teresa.lamau@mail.de";"Lamau";"Teresa";"";"";""

"DELETE";"2018-02-27T03:58:21";"2018-02-27T03:58:21";"672777777";"IT";"+491832457261";"simon.terfon@mail.de";"Terfon";"Simon";"";"";""

2.3.4 Exportbeispiel für fehlerhafte Nutzerdefinition

In dem nachfolgenden Beispiel wurde eine Nutzerin exportiert, die die gleiche Mobilfunknummer hat, wie der zuvor exportierte Nutzer „Norbert Meier“. Wie im Kapitel 2.1 beschrieben, ist eine mehrdeutige Zuordnung nicht zulässig und der nachfolgende Import würde den Eintrag als fehlerhaft identifizieren und nicht importieren.

ACTION";"CREATED";"MODIFIED";"USER_ID";"DEPARTMENT";"PHONE";"EMAIL";"LAST_NAME";"FIRST_NAME";"KEYWORDS";"CHANNELS";"GROUPS"
"INSERT";"2018-02-27T03:58:21";"2018-02-27T03:58:21";"1237635";"Business";"+491613458725";"sina.kundig@mail.de";"Kundig";"Sina";"";"IT News";"Logistik"

3 LDAP-Adapter

Der sogenannte LDAP-Adapter ist eine exemplarische Umsetzung eines Adapters zwischen einem LDAP-Server und der beschriebenen CSV-Struktur. Der LDAP-Adapter wird von ginlo als Vorlage bereitgestellt, erhält jedoch keinen Support durch ginlo. Der Quelltext darf jedoch im Rahmen einer Anbindung an ginlo verwendet und in Ihre Software Komponenten eingebaut und vertrieben werden („open source“). Die kanonischen Schnittstellen sind der CSV-Import über das Cockpit und über die REST-Schnittstelle.

3.1 Überblick

In diesem Kapitel wird für die Zielgruppe „Entwickler“ die bereitgestellte Implementierung eines LDAP-Adapters beschrieben. Die Implementierung ist für die Anbindung der LDAP-Provider Microsoft AD LDS und OpenLDAP vorgesehen. Hiervon abweichende LDAP-Provider können ebenfalls angebunden werden. Hier können zusätzliche Implementierungsanpassungen und ggf. zusätzliche Konfigurationselemente notwendig sein.

Aufgabe des LDAP-Adapters ist es, die Nutzerdaten innerhalb des LDAP-Providers zu identifizieren und in die definierte CSV-Struktur zu überführen. Bestandteil der Identifikation von Nutzerdaten ist es, zu bestimmen, welche Nutzer neu hinzuzufügen, welche zu ändern und welche zu löschen sind.

Der LDAP-Adapter setzt sich aus den folgenden Subkomponenten zusammen:

- H2-Datenbank zur Persistierung der exportierten Daten
- HTTP-Server zur Bereitstellung der Services
- Service-Implementierung

Der LDAP-Adapter ist in Java realisiert. Der Adapter ist eine in sich abgeschlossene Spring-Boot-Anwendung und enthält als solche die oben aufgeführten Subkomponenten. Die Klassen sind als Spring Beans realisiert und werden entsprechend konfiguriert.

3.2 Rahmenbedingungen

Für den Einsatz des LDAP-Prototyps müssen die folgenden Voraussetzungen erfüllt sein:

- Java-Laufzeitumgebung in der Version 8
- LDAP-Provider Microsoft AD LDS oder OpenLDAP
- Zusätzliche Merkmale für die Zuordnung von Gruppen, Kanälen und Schlagworten müssen im LDAP-Provider konfigurierbar sein
- Nutzerdaten müssen den Voraussetzungen des ginlo-Datenmodells für Nutzerdaten entsprechen

3.3 Verwendung

Der Export der Nutzerdaten wird durch den folgenden Aufruf gestartet:

<http-server>/secmes-ldap-simsme-exp/commands/exportLdapSimsMeUser

Unter Angabe des HTTP-Parameters exportType=ALL werden sämtliche Nutzerdaten der LDAP-Instanz exportiert. Das Delta zum vorherigen Export wird durch die Angabe des Parameters exportType=DIFF erzeugt. Die aus der LDAP-Instanz exportierten Daten werden in der H2-Datenbank persistiert. Die HTTP-Response des Service ist eine JSON-Struktur, die den konkreten Export identifiziert.

{"exportInfo":{"id":"161BDDCE52527da12d710334469bd2e4fdc17c41dee00","ldapUrl":"ldap://secmes-ldap:389","ldapBase":"dc=simsme,dc=dpag,dc=de","ldapUserRecordCount":11,"exportFile":"simsmeLdapUserExport-20180222-141345-894---161BDDCE52527da12d710334469bd2e4fdc17c41dee00---all.csv","exportInfoFile":"simsmeLdapUserExport-20180222-141345-894---161BDDCE52527da12d710334469bd2e4fdc17c41dee00---all.json","exportLogFile":"simsmeLdapUserExport-20180222-141345-894---161BDDCE52527da12d710334469bd2e4fdc17c41dee00---all.log","exportType":"ALL","exportFileLength":2926,"startTimestamp":"2018-02-22T14:13:45.893+00:00","endTimestamp":"2018-02-22T14:13:45.919+00:00","error":false,"errorStacktrace":null,"message":"OK","exportRecordCount":11,"deleteRecordCount":0,"upsertRecordCount":11}}

Exportierte Nutzerdaten werden als Snapshot in einer H2-Datenbank hinterlegt. Beim erneuten Export erfolgt über diese Datenbank der Delta-Abgleich. Hieraus ergeben sich die Aktions-Kommandos „INSERT“, „UPDATE“ und „DELETE“ für die CSV-Datenstruktur.

Die enthaltene ID kann im Folgenden dazu genutzt werden, um die exportierten Nutzerdaten als CSV-Datei bereitzustellen. Dazu ist der nachfolgende Service unter Angabe einer der exportID aufzurufen:

<http-server>/secmes-ldap-simsme-exp/commands/exportInfo/{exportId}/download

Auf diesem Weg können alle bislang durchgeführten Exporte als CSV-Datei bereitgestellt werden.

Neben der direkten Verwendung der Service-Methoden steht zusätzlich das Swagger UI zur Verfügung, über das die Parametrisierung der Aufrufe per UI erfolgen kann. Über das Swagger UI stehen alle Services zur Verfügung, die durch den LDAP-Adapter angeboten werden. Das Swagger UI wird über die folgende URL erreicht:

<http-server>/secmes-ldap-simsme-exp/swagger-ui.html

Der Zugriff auf die LDAP-Instanz als auch das Mapping der LDAP-Attribute auf die CSV-Datenstruktur ist konfigurativ im Adapter hinterlegt. Weitere Informationen hierzu finden sich im Kapitel 3.6.2.

3.4 Statische Struktur

3.4.1 Paketstruktur

Der LDAP-Adapter ist in mehrere Pakete (Java Packages) strukturiert. Die Packages finden sich als Subpackages im Pfad **de.dpag.simsme.ldapexport**.

Die folgenden zentralen Subpackages realisieren die Funktionalität des LDAP-Adapters:

- Adapter – implementiert den Zugriff auf den LDAP-Provider und definiert das Mapping der Attribute
- Configuration – implementiert das Auslesen der Komponenten- und LDAP-Provider-Attribute aus der Konfigurationsdatei der Anwendung
- Controller – stellt die Service-Schnittstelle der Komponente bereit
- Domain – definiert die Entitäten des LDAP-Adapters
- Processor – realisiert die einzelnen Kommandos, die bei Aufruf der Service-Methoden ausgelöst werden
- Service – implementiert die Generierung der exportierten Daten als CSV-Datei

3.4.2 Klassenstruktur

3.4.2.1 Adapter

Im Package de.dpag.simsme.ldapexport.adapter.ldap sind die Klassen zur Anbindung an den LDAP-Provider realisiert. Ausgangspunkt ist die Klasse LdapAdapterServerImpl. Diese stellt die Methode zum Auslesen der Nutzerdaten zur Verfügung. Das Auslesen der Nutzerdaten erfolgt über die Klasse BaseLdapReader, die das Interface LdapReader implementiert. Basierend auf den Konfigurationsdaten zum LDAP-Provider (siehe Kapitel 3.6) wird eine Instanz des LdapReader in der Klasse LdapAdapterServerImpl erzeugt. An diese wird anschließend das Auslesen der Daten delegiert.

Der konkrete Zugriff auf einen LDAP-Provider ist unter Verwendung von Spring realisiert. Hierzu werden die Klassen LdapTemplate, LdapQueryBuilder und SearchScope verwendet.

Die Implementierung des LDAP-Adapters ist unabhängig vom LDAP-Provider und wird ausschließlich per Konfiguration an einen konkreten Provider gebunden. Auf diese Weise können LDAP-Produkte unterschiedlicher Hersteller (bspw. OpenLDAP) adressiert werden. Die zugehörigen Konfigurationsinformationen sind im Kapitel 3.6 beschrieben.

Die Zusammenhänge der Klassen finden sich in der nachfolgenden Abbildung.

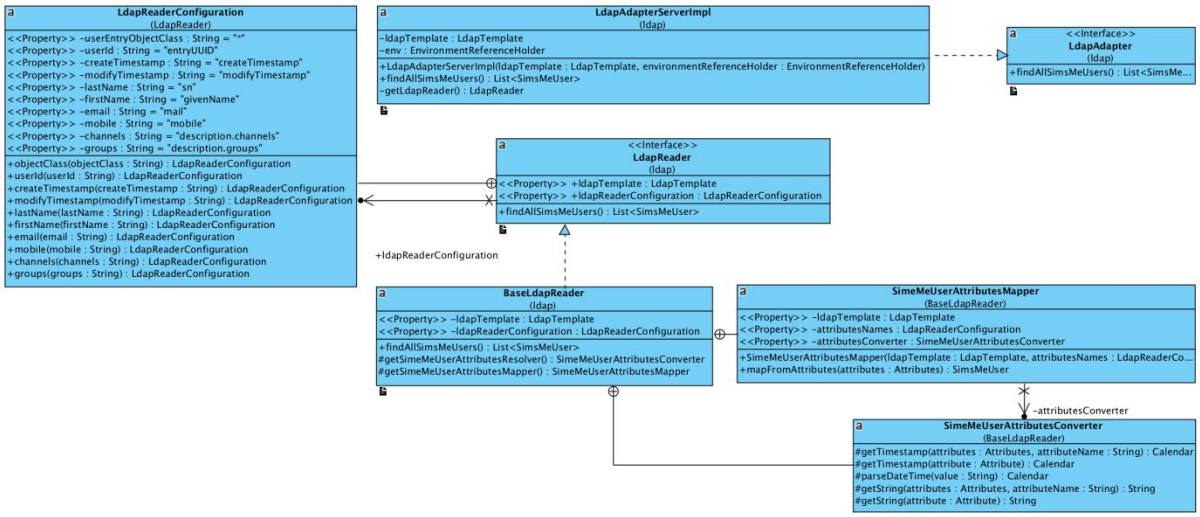


Abbildung 2: Klassendiagramm LDAP-Adapter

3.4.2.2 Controller und Commands

Die nach außen verfügbare Schnittstelle des LDAP-Adapters wird durch Klassen im Package de.dpag.simsme.ldapexport.controller realisiert. Die Klasse SecmesLdapExportController realisiert die nach außen verfügbaren Services auf Basis des Command-Patterns. Jeder Service entspricht einem Command. Für jeden nach außen verfügbaren Service gibt es eine Command-Implementierung, die die zugehörige Funktionalität realisieren. Jede Command-Klasse (bspw. ExportLdapSimsMeUser) leitet sich von der Oberklasse AbstractClientCommand ab.

Das protokollbezogene Handling der HTTP-basierenden Aufrufe ist in den Oberklassen AbstractCommandController und AbstractApplicationController abstrahiert. Auch hier wird das Spring-Framework für den technischen Umgang verwendet.

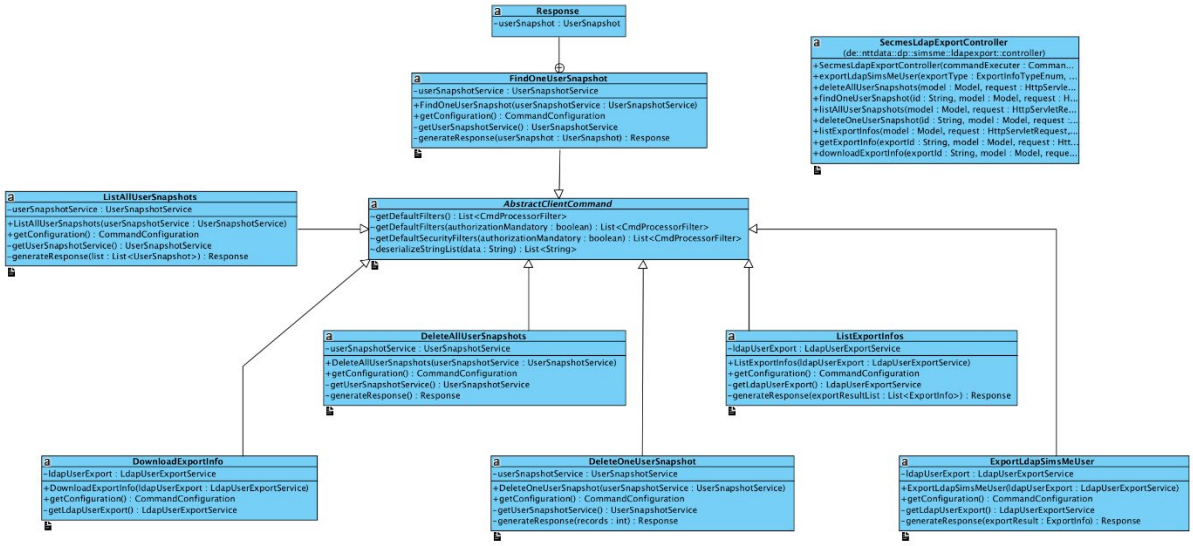


Abbildung 3: Klassendiagramm Controller und Commands

3.4.2.3 Service

Die Klassen im Package de.dpag.simsme.ldapexport.service realisieren die Generierung der Exportinformationen und -dateien als auch die Persistierung exportierter Daten in eine Datenbank (in diesem Fall H2) sowie den Zugriff darauf.

In der Klasse UserSnapshotServiceDefaultImpl werden alle datenbankbezogenen Persistenzmethoden realisiert. Dies beinhaltet das Speichern von Nutzerdaten aus einem LDAP-Provider als auch das Auslesen dieser. Die Klasse LdapUserExportServiceDefaultImpl verwendet diese Methoden zum Auslesen und Erzeugen von Export-Information (ExportInfo). Wesentliche Aufgabe der Klasse ist jedoch die Generierung einer Repräsentanz von Nutzerdaten als CSV-Datei. Beide Implementierungen verwenden die Spring-Mechanismen zur Konfiguration.

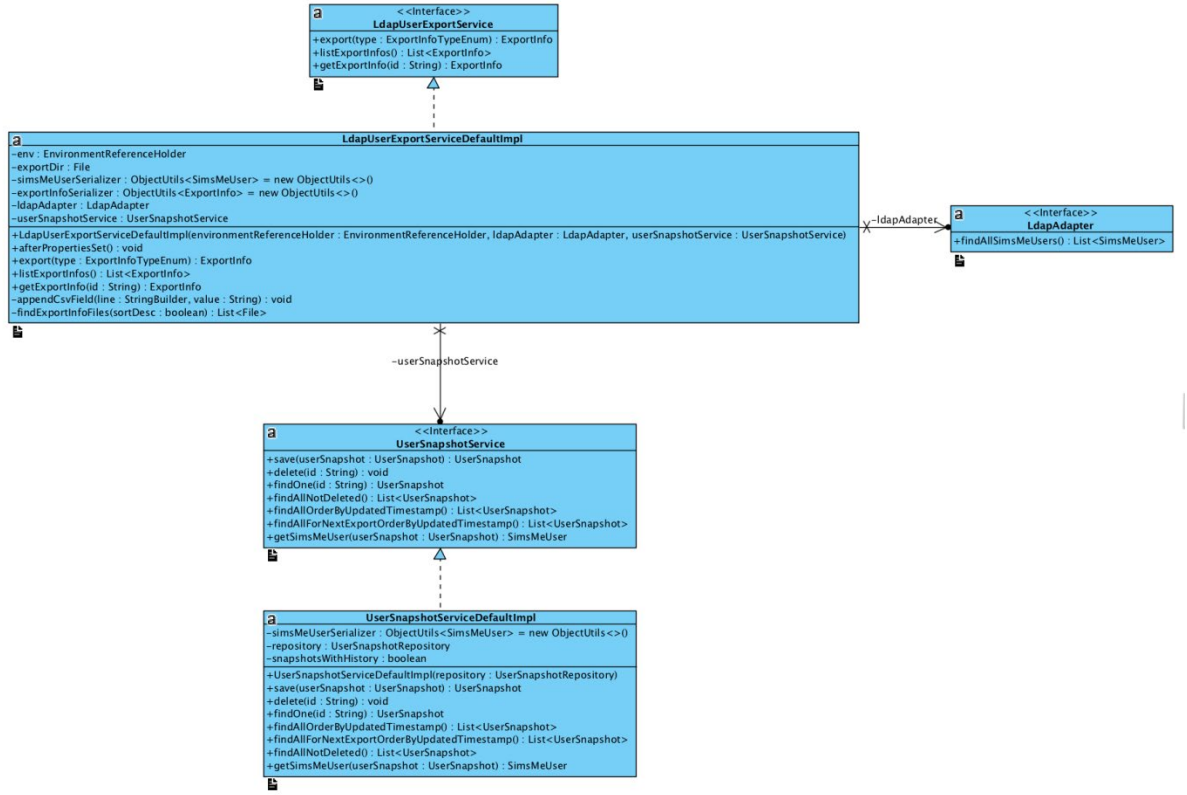


Abbildung 4: Klassendiagramm Service

3.5 Dynamische Struktur

Der nachfolgende Ablauf zeigt die Aufrufkette beim Exportieren von Nutzerdaten.

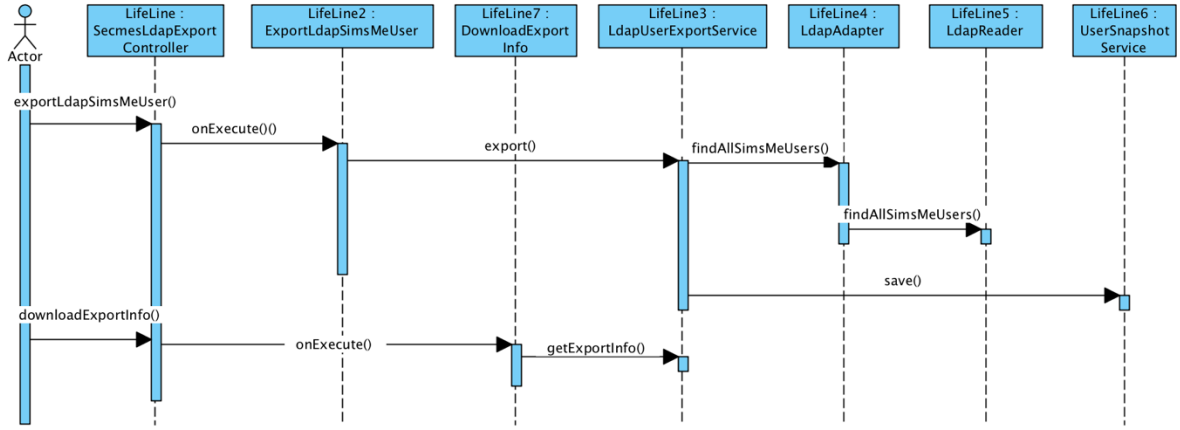


Abbildung 5: Export von Nutzerdaten

Die nachfolgende Tabelle erläutert die oben dokumentierten Schritte in kurzer Form.

Schritt	Methode	Beschreibung
1	exportLdapSimsMeUser	Entgegennahme des HTTP-Requests
2	onExecute	Start des Exports
3	export	Start LDAP-Anfrage und Erzeugung Ergebnisdateien
4	findAllSimsMeUsers	Aufbereitung der LDAP-Anfrage
5	findAllSimsMeUsers	Auslesen der Nutzerdaten aus dem LDAP-Provider
6	save	Speicherung der Daten in H2-Datenbank
7	downloadExportInfo	Entgegennahme des HTTP-Requests
8	onExecute	Start des Downloads
9	getExportInfo	Identifizierung und Download der CSV-Datei

Tabelle 3: Schritte zum Export von Nutzerdaten

3.6 Konfiguration

Der LDAP-Prototyp wird über mehrere Konfigurationsdateien an den konkreten Einsatz angepasst.

3.6.1 Anwendungskonfiguration

Die Konfiguration der Spring Beans erfolgt in der Datei application.yml (YAML-Datei). Für weitere Umgebungen (bspw. lokale Entwicklung) können Konfigurationsdateien angelegt werden, die der jeweiligen Umgebung gerecht werden. Die Benennung erfolgt über einen Postfix (bspw. application-local.yml). In der Konfigurationsdatei werden u. a. der Port der Anwendung, Logging-Verhalten und die Datenbank konfiguriert. Weiterhin sind hier auch die Zugriffsinformationen des LDAP-Providers hinterlegt. Eine beispielhafte Konfiguration ist im Folgenden zu sehen.

```
ldap:
  url: ${LDAP_URL:ldap://secmes-ldap:389}
  base: ${LDAP_BASE:dc=simsme}
  user: ${LDAP_USER:cn=admin,dc=simsme}
  password: ${LDAP_PASSWORD:password}
  readerImplementationClass: ${LDAP_READER_IMPLEMENTATION_CLASS:}
  readerConfigFile: ${LDAP_READER_CONFIG_FILE:}
  exportDir: ${LDAP_EXPORT_DIR:${INSTANCE_HOME:${user.dir}}/export}
```

Das Konfigurationselement readerConfigFile definiert die Konfigurationsdatei, in der das Mapping der LDAP-Attribute hinterlegt ist. Das Konfigurationselement exportDir legt das Verzeichnis fest, in dem der Adapter exportierte CSV-Dateien zwischenspeichert, um diese auf Anforderung zur Verfügung zu stellen.

3.6.2 Mapping LDAP-Attribute

Das Mapping der LDAP-Attribute auf die ginlo-Nutzerdaten wird konfigurativ definiert. Die Konfiguration erfolgt im JSON-Format und wird in der Datei LdapReaderConfiguration.json beschrieben. Neben den ginlo-Attributen ist hier auch festlegt, wo die Nutzerdaten in der Datenstruktur des LDAP-Providers zu finden sind.

Im Folgenden ist eine beispielhafte Konfiguration dargestellt. Mittels der Merkmale cd, dn und userEntryObjectClass ist festgelegt, welches Objekt auf die Nutzerdaten zu mappen ist.

```
{
  "userEntryObjectClass" : "inetOrgPerson",
  "dn" : "entryDn",
  "cn" : "cn",
  "userId" : "entryUUID",
  "createTimestamp" : "createTimestamp",
  "modifyTimestamp" : "modifyTimestamp",
  "lastName" : "sn",
  "firstName" : "givenName",
  "email" : "mail",
  "mobile" : "mobile",
  "department" : "description.department",
  "keywords" : "description.keywords",
  "channels" : "description.channels",
  "groups" : "description.groups",
  "description" : "description"
}
```

4 LDAP-Importer

Der LDAP-Importer verarbeitet die Daten des LDAP-Adapters und integriert diese automatisiert in das Management Cockpit von ginlo. Dabei kann der Prozess autark und ohne zusätzliche Administrator-Interaktion arbeiten. Insbesondere bei großen Firmen sollen manuelle Prozesse vermieden werden, um den Aufwand für die Verwaltung der eigenen Mitarbeiter möglichst gering zu halten. Um dies zu realisieren, können über die REST-Schnittstelle LDAP-Importdateien hochgeladen, deren Status abgefragt und das Ergebnisprotokoll ermittelt werden.

4.1 Voraussetzungen

Für die allgemeine Verwendung der REST-Schnittstelle ist eine Authentifizierung notwendig. Zum einen wird ein spezieller Nutzer benötigt, welcher sich mittels Basic-Authentifizierung gegen das System anmeldet und zum anderen ein Clientzertifikat. Das Anlegen eines solchen Nutzers erfolgt über das Management Cockpit durch den Administrator. Dafür ist unter den Einstellungen der Button *LDAP API aktivieren* zu finden. Eine genaue Anleitung zum Anlegen eines API-Nutzers finden Sie in der Management Cockpit Dokumentation im Kapitel „LDAP-Daten“.

Intern wird durch den Administrator ein neuer Stellvertreter vom Typ „apiuser“ angelegt. Für diesen Nutzer wird ein Passwort erzeugt und ein Clientzertifikat generiert. Das öffentliche Clientzertifikat wird analog einer Stellvertretung auf den Servern verteilt. Wenn ein API-Nutzer angelegt wurde, dann kann der Administrator über die Funktion *Zugangsdaten anzeigen* sich die Zugangsdaten (Nutzername + Passwort) anzeigen lassen und das öffentliche Clientzertifikat herunterladen, um dessen Richtigkeit zu überprüfen. Den privaten Teil des Zertifikats bekommt der angemeldete Administrator auf seine hinterlegte E-Mail-Adresse gesendet. Mit diesen Zugangsdaten kann der Kunde die Cockpit-API ansprechen. Die Authentifizierung erfolgt dabei über zwei Komponenten: Der Nutzername und das Passwort wird als Basic-Auth übergeben, und als zweiter Faktor dient das Clientzertifikat.

Sollten diese Zugangsdaten kompromittiert sein, kann der Administrator über die Cockpit-Oberfläche ein neues Passwort generieren und ggf. auch ein neues Clientzertifikat erzeugen. Die vorhandenen Zugangsdaten verlieren damit ihre Gültigkeit.

4.2 Cockpit REST-Client – Beispielimplementierung

Die genaue Spezifikation der von ginlo bereitgestellten REST-Schnittstelle ist in einer Swagger-Datei festgehalten. Diese kann unter https://admin.sims.me/adminkonsole/swagger-mgt-api.yaml heruntergeladen werden. Für eine vereinfachte Integration in die firmeneigenen Strukturen ist eine automatische Generierung eines Clients mithilfe dieser Datei über Swagger UI möglich. Die nachfolgenden Punkte beschreiben eine detaillierte Verwendung des Clients an einigen Beispielen und beinhalten nützliche Hinweise zur Anpassung des generierten Clients.

4.2.1 Abhängigkeiten (Packages)

Damit der von Swagger generierte Client lauffähig ist, sind einige Abhängigkeiten notwendig, welche dem Projekt hinzugefügt werden sollten. Dazu zählen:

- gson-2.8.1.jar
- gson-fire-1.8.0.jar
- logging-interceptor-2.7.5.jar
- okhttp-2.7.5.jar
- okio-1.6.0.jar
- swagger-annotations-1.5.15.jar
- threetenbp-1.3.5.jar

4.2.2 Verwendung eines Proxys

Einige Unternehmen sehen für die firmenübergreifende Kommunikation die Verwendung eines Proxy-Servers vor. Der generierte Swagger-Client unterstützt nativ nicht die Verwendung eines Proxys. Folgender Code zeigt beispielhaft die Integration eines eigenen Proxys:

```
final InfoApi infoApi = new InfoApi(basePath, basicUsername, basicPassword, pathCertificate, passwordCertificate);

setProxy(infoApi.getApiClient());

private static void setProxy(final ApiClient apiClient)
{
  if ((System.getProperty("http.proxyHost") != null) && !System.getProperty("http.proxyHost").isEmpty()
     && (System.getProperty("http.proxyPort") != null) && !System.getProperty("http.proxyPort").isEmpty())
  {
    apiClient.getHttpClient().setProxy(new Proxy(Proxy.Type.HTTP,
                                       new InetSocketAddress(System.getProperty("http.proxyHost"),
                                                             Integer.parseInt(System.getProperty("http.proxyPort")))));
  }
}
```

Im Speziellen kann dies über die Verwendung von Runtime-Properties gelöst werden. Diese können dann, wie im oberen Codebeispiel gezeigt, ausgelesen werden. Ein endgültiger Aufruf des Import-Clients kann dann wie folgt aussehen: **java -Dhttp.proxyHost=proxy.de -Dhttp.proxyPort=5555**.

4.2.3 Überprüfung der Verbindung

In der Swagger-Datei ist eine Funktion namens checkConnection definiert, welche es ermöglicht, die Konnektivität zu der REST-Schnittstelle zu überprüfen. Dabei steht die Identifikation der Erreichbarkeit der Schnittstelle im Vordergrund. Ist eine Verbindung über ein Zertifikat und Basic-Authentifizierung hergestellt, sind die Grundlagen für die Verwendung der weiteren Funktionen, für die eine Authentifizierung erforderlich ist, geschaffen.

Folgender Beispielcode zeigt den Aufruf der Funktion:

```
//Prüfen der Verbindung
Company          company = infoApi.checkConnection();
```

4.2.4 Verwendung des Client-Zertifikats

Die Unterstützung eines Client-Zertifikats wird nicht automatisch durch Swagger UI generiert und muss nachgepflegt werden. Der folgende Codeausschnitt zeigt beispielhaft, wie ein übergebenes Zertifikat mit dem entsprechenden Zertifikat-Passwort an den HTTP-Client übergeben werden kann.

```
KeyStore        keyStore                = KeyStore.getInstance("PKCS12");
FileInputStream clientCertificateContent = new FileInputStream(certificate);

keyStore.load(clientCertificateContent, passwordCertificate.toCharArray());

KeyManagerFactory keyManagerFactory =
KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm());

keyManagerFactory.init(keyStore, passwordCertificate.toCharArray());

SSLContext sslContext = SSLContext.getInstance("TLS");

sslContext.init(keyManagerFactory.getKeyManagers(), null, null);

httpClient = new OkHttpClient();
httpClient.setSslSocketFactory(sslContext.getSocketFactory());

verifyingSsl = true;
```

Dies kann beispielsweise im Konstruktor des API-Clients realisiert werden. Da es sich um ein PKCS12-Zertifikat handelt, wird zunächst eine Keystore-Instanz erzeugt, welche das übergebene Client-Zertifikat lädt. Mithilfe einer KeyManagerFactory wird das Zertifikat im Anschluss entschlüsselt und an den HTTP-Client übergeben.

4.2.5 Verwendung der Basic-Authentifizierung

Der zweite Authentifizierungskanal wird über eine Basic-Authentifizierung realisiert. Der folgende Codeausschnitt zeigt, wie diese in das Projekt integriert werden kann. Auch hier ist es möglich, jene in den Konstruktor des API-Clients einzubinden.

```
// Setup authentications (key: authentication name, value: authentication).
authentications = new HashMap<String, Authentication>();

HttpBasicAuth basic = new HttpBasicAuth();

basic.setUsername(basicUsername);
basic.setPassword(basicPassword);

authentications.put("api_auth", basic);

// Prevent the authentications from being modified.
authentications = Collections.unmodifiableMap(authentications);
```

Wichtig ist noch, dass diese Authentifizierung als Headerparameter an den entsprechenden Request innerhalb der UserApi oder InfoApi gesetzt wird (siehe nächsten Ausschnitt).

```
apiClient.getAuthentications().get("api_auth").applyToParams(null, localVarHeaderParams);
```

4.2.6 Auslesen einer Importdatei

Die zu importierenden, aus dem LDAP-Adapter stammenden Daten können an einer bestimmten Stelle im System abgelegt werden. Der Pfad zur Importdatei kann ebenso als Parameter an den Swagger-Client übergeben werden und der Inhalt der Datei in folgender Form ausgelesen werden.

```
java.nio.file.Path csvFile = Paths.get(pathCSV);
```

4.2.7 Beispieldurchführung eines Imports

Nachdem die Daten erfolgreich an den Swagger-Client übergeben wurden, kann der Import über die REST-Schnittstelle gestartet werden. Im folgenden Codebeispiel wird gezeigt, wie eine Instanz der UserApi mit den entsprechenden Credentials erzeugt, ein Proxy registriert und der Import gestartet wird.

```
final UserApi userApi = new UserApi(basePath, basicUsername, basicPassword, pathCertificate, passwordCertificate);

setProxy(userApi.getApiClient());

CsvImportResult importResult = userApi.startCsvImport(csvFile);
```

Das Resultat wird durch ein CsvImportResult repräsentiert, wie es im YAML-File beschrieben ist.

```
String importId = URLDecoder.decode(getImportId(importResult), "UTF-8");
```

Aus diesem Resultat kann nun die ImportId extrahiert werden, was durch die Funktion getImportId realisiert wird. Die ImportId ist ein eindeutiges Identifikationsmerkmal des Imports und kann auch nur von der entsprechenden Company ausgelesen werden.

```
private static String getImportId(final CsvImportResult importResult)
{
  String[] url = importResult.getCsvImportResult().getStatusUrl().split("/");

  return url[url.length - 1];
}
//Warte bis Import abgeschlossen
waitForImport(userApi, importId);
```

Die Abfrage des ImportStates ermöglicht das Erkennen des Status des Imports. Durch regelmäßiges Polling gegen die REST-Schnittstelle wird die Fertigstellung interpretiert. Wechselt der Status auf den Zustand „done“, ist der Import abgeschlossen. Neben dem Status liefert der ImportState auch Informationen über aufgetretene Fehlerlevel in den Stufen Info, Warnung, Error und Fatal Error.
Die Methode „waitForImport“ zeigt beispielhaft die Verwendung des Abfragens des ImportStates.

```
private static void waitForImport(final UserApi userApi,
                                  final String  importId)
  throws io.swagger.client.ApiException, java.lang.InterruptedException
{
  ImportState importState = userApi.getCsvImportStateByGuid(importId);

  if (!importState.getImportState().getError().getERROR().equals("0"))
  {
    LOGGER.log(Level.WARNING, "Es gab einen Fehler beim Status holen");
  }

  if (!importState.getImportState().getError().getFATALERROR().equals("0"))
  {
    LOGGER.log(Level.WARNING, "Es gab einen Fehler beim Status holen");
  }

  int countTries = 0;

  while (!importState.getImportState().getMode().getValue().equals("done"))
  {
    if (countTries == 100)
    {
      break;
    }

    importState = userApi.getCsvImportStateByGuid(importId);

    if (!importState.getImportState().getError().getERROR().equals("0"))
    {
      LOGGER.log(Level.WARNING, "Es gab einen Fehler beim Status holen");
    }

    if (!importState.getImportState().getError().getFATALERROR().equals("0"))
    {
      LOGGER.log(Level.WARNING, "Es gab einen Fehler beim Status holen");
    }

    countTries++;

    Thread.sleep(1000);
  }
}
```

4.2.8 Speichern des Ergebnisprotokolls

Nach Abschluss des Imports kann das vollständige Protokoll heruntergeladen werden. Während der Durchführung wird ein Prüfbericht geliefert. Der komplette Ergebnisbericht wird erst angeboten, wenn der Import den Status „done“ angenommen hat.

Folgendes Codebeispiel zeigt das Speichern des Ergebnisses in einer csv-Datei.

```
private static void downloadProtocol(final UserApi,
                                     final String  importId)
   throws io.swagger.client.ApiException
{
   String report = userApi.downloadReportByGuid(importId);

   try(
      BufferedWriter bw = Files.newBufferedWriter(Paths.get(importId.substring(6) + ".csv"),
                                          Charset.forName("UTF-8"))
      )
   {
      bw.write(report);
   }
   catch (java.io.IOException ioex)
   {
      LOGGER.log(Level.SEVERE, "Fehler beim Speichern des Protokolls");
   }
}
```

4.2.9 Speichern der Importübersicht

Die Speicherung der Übersicht eines Imports ist ebenfalls möglich. Das Beispiel zeigt die Abfrage der Übersicht des Imports über die REST-API und der darauffolgenden Persistierung.

```
private static void downloadImportOverview(final InfoApi api,
                                           final String  token)
   throws io.swagger.client.ApiException
{
   String overview = api.importOverview(token);

   if (overview.isEmpty())
   {
      LOGGER.log(Level.SEVERE, "Import-Übersicht ist leer");
   }

   try(
      BufferedWriter bw = Files.newBufferedWriter(Paths.get("uebersicht" + token + ".JSON"),
                                          Charset.forName("UTF-8"))
      )
   {
      bw.write(overview);
   }
   catch (IOException ioex)
   {
      LOGGER.log(Level.SEVERE.WARNING, "Fehler beim Speichern der Import-Übersicht");
   }
}
```

4.2.10 Automatisierungsmöglichkeit des LDAP-Importers

Mit Hilfe eines Batch-Skripts kann der LDAP-Importer automatisiert aufgerufen werden. Beispielsweise kann der Inhalt des Skriptes wie folgt aussehen:

SET PATH=C:\Program Files\Java\jre1.8.0_191\bin\;%PATH%;

java -Dhttp.proxyHost=unknown.proxy.com -Dhttp.proxyPort=1234 -jar CockpitClient.jar [Basis Url der Management Cockpit Api] [apiUserName] [apiUserPasswort] [Zertifikat] [Zertifikatspasswort] [Importdatei] [Authentifizierungstoken]

Für die vollständige Funktionalität des Swagger-Clients ist mindestens Java 8 notwendig.