MSP Medien Systempartner

... für die Verlage von Morgen

MSP SSO Dokumentation

Version 2.5
Mai 2016

Auftraggeber:
MSP Medien Systempartner GmbH & Co. KG
Martinistraße 33
28195 Bremen

Ansprechpartner:
Thomas Pundt
Marco Barthel

1 Inhalt

1. Installation

1.1 Virtual-Host und Datenbank

Das gleich beschriebene Setup-Skript setzt einen funktionierenden Apache-Virtual-Host voraus. Unter Apache-2.4 kann so ein Virtual-Host-Eintrag wie folgt aussehen:

```
<VirtualHost *:80>
    ServerAdmin serveradmin@ssoserver.test
    ServerName login.ssoserver.test

    DocumentRoot /home/thomas/Dokumente/netbeans/login.ssoserver.test

    ErrorLog /var/log/apache2/login.ssoserver.test-error_log
    CustomLog /var/log/apache2/login.ssoserver.test-access_log combined

    HostnameLookups Off
    UseCanonicalName Off
    ServerSignature Off

    # Die folgenden beiden Einträge können in einer .htaccess-Datei stehen:
    # SetEnv APPLICATION_ENV development_pundt
    # SetEnv CUSTOMER vnp

    <Directory "/home/thomas/Dokumente/netbeans/login.ssoserver.test">
       Options Indexes MultiViews FollowSymLinks
       AllowOverride All
       Require all granted
    </Directory>

</VirtualHost>
```

Damit der Hostname gefunden wird, kann man ihn in der Datei /etc/hosts (Linux, Mac) oder C:\Windows\System32\drivers\etc\hosts (Windows) eintragen als:

```
127.0.0.1    login.ssoserver.test
```

Auch um das Anlegen einer Datenbank mit Datenbank-User kann sich das Setup-Skript nicht kümmern; unter Windows kann man dafür „HeidiSQL“, unter Mac OS X „Sequel Pro“ benutzen.

1.2 Das Setup-Skript

Der einfachste Weg zu einem funktionierenden SSO-Server läuft über die Benutzung des Setup-Skripts, welches unter /setup/setup.php aufgerufen werden kann. Dieses prüft zunächst, ob folgende Voraussetzungen erfüllt sind:

- PHP > 5.3.0 (wir empfehlen PHP 7)
- Alle nötigen PHP-Module vorhanden
- Schreibrechte auf einige Verzeichnisse

Weitere nötige Voraussetzungen sind:

- Webserver mit einem Virtual-Host, in dessen Document-Root die PHP-Sourcen ausgepackt sind – im Weiteren benutzen wir dazu den Namen **`login.ssoserver.test`**
- Zugriff auf eine MySQL-Datenbank mit Schreibrechten.

Konnten alle Voraussetzungen erfüllt werden, präsentiert das Setup-Skript dem User ein Formular, in dem die wichtigsten Parameter für die Konfiguration des SSO-Servers eingetragen werden können:

- **Customer-Kürzel:** Ein Zwei- oder Drei-Buchstaben-Kürzel, mit dem der Kunde identifiziert wird; dieses Kürzel dient dazu, unterschiedliche Kunden-Installationen in einem Git-Repository unterbringen zu können, ebenso wie für kundenspezifische Anpassungen am PHP-Code.
- **Application-Environment:** In einer Ini-Datei werden mehrere Abschnitte definiert, zwei davon sind fest vergeben: „production“ für die Produktiv-Umgebung beim Kunden, „testing“ für eine Test-Umgebung beim Kunden. Weitere Umgebungen für die Entwicklung können noch eingetragen werden. Hier wird der Name für die anzulegende Entwicklungs-Umgebung abgefragt.
- **Hostnamen:** Hier können schon die Hostnamen für Produktion, Testumgebung (beide optional) und Entwicklungsumgebung eingetragen werden. Als Hostname für die Entwicklungsumgebung wird der Name des aktuell benutzten Virtual Hosts eingetragen.
- **SSO-Server App-Passwort:** Für den SSO-Server hat das Setup-Skript einen Application-Key generiert. Dies ist ein zufällige Folge von 32 Zeichen Hex-Zeichen.
- **Crypt-Algorithmus:** User-Passwörter werden in der Datenbank verschlüsselt abgelegt. Für den Fall, dass bereits eine Kundendatenbank existiert, welche migriert werden soll, besteht die Möglichkeit, verschiedene Crypt-Algorithmen auszuwählen. Derzeit werden ohne weitere Anpassungen folgende Crypt-Methoden unterstützt: SHA1, MD5, doppelte MD5-Verschlüsselung, Blowfish; jeweils mit oder ohne Salt/Secret. Die Passwort-Überprüfung ist in einer Methode gekapselt (in /class/helper/msp_helper_password.php) und kann bei Bedarf einfach erweitert werden.
- **Crypt-Secret:** Für SHA1 und MD5 können die Passwörter mit einem Präfix verschlüsselt werden, damit sie nicht in öffentlichen Rainbow-Tabellen auftauchen. Dieses Präfix kann hier angegeben werden.
- **Shadow-Passwort:** Die SSO-API ist so gebaut, dass Methoden, die Daten zum eingeloggten User abfragen, neben dem Applikations-Schlüssel ein Login-Token mitgegeben werden muss. Methoden, die Daten zu nicht eingeloggten Users liefern, benötigen ein weiteres Passwort: das sogenannte Shadow-Passwort. Es wird nur an besonders vertrauenswürdige Applikationen ausgegeben.
- **Crypt-Key:** Müssen Daten symmetrisch verschlüsselt werden (z.B. das Master-Cookie), geschieht dies mit dem hier definierten Crypt-Key.
- **Datenbank:** Hier werden die Verbindungs-Parameter zu einer MySQL-Datenbank für die Entwicklungs-Umgebung abgefragt. Die Datenbank muss schon existieren und ist idealerweise leer, der Datenbank-User benötigt volle Schreibrechte für die Datenbank. Schon bestehende Tabellen werden nicht überschrieben, ebenso keine bestehenden Datenbank-Einträge.
- Durch Klick auf **„erzeugen“** wird schließlich versucht, eine Config-Datei im /config-Verzeichnis anzulegen, sowie das Datenbank-Schema zu schreiben. Für das Backend wird ein Admin-User ssoadmin/ssoadmin angelegt, der sich unter /backend einloggen kann.

1.3 Datenbank

Das MSP SSO verwendet zur Speicherung von Daten eine MySQL-Datenbank[1]. Um mögliche Kompatibilitätsprobleme zu vermeiden, empfehlen wir die Verwendung der Version 5.5 oder größer.

[1] Anstelle MySQL kann problemlos auch MariaDB benutzt werden; der Typ ist dann ebenfalls „mysql“. Andere Datenbanken können auf Anfrage angebunden werden. Für Unit-Tests wird eine SQLite3-Datenbank benutzt.

Sofern nicht mit dem Setup-Skript wie oben beschrieben schon geschehen, legen Sie eine Datenbank an und importieren Sie den Datenbank-Dump (die „msp_sso.sql“-Datei finden Sie im /class/database/ Verzeichnis) mit den erforderlichen Tabellen und Datensätzen. Legen Sie einen neuen Datenbank-Benutzer an und geben Sie ihm volle Lese- und Schreibrechte. Abschließend tragen Sie die entsprechenden Zugangsdaten für die Datenbank in die Konfigurationsdatei ein.

1.4 Konfigurationsdatei

Die Grund-Konfiguration wird in einer Ini-Datei wie oben beschrieben vorgenommen. Die Einträge der Ini-Datei werden in der Datei config.php den PHP-Skripten als Defines zur Verfügung gestellt. Die Ini-Datei ist in Sektionen unterteilt, sie sollte mindestens die Sektionen [production] und [testing] enthalten; eine Sektion kann alle Schlüssel einer anderen Sektion erben, indem man die vererbte Sektion angibt: [testing : production]. Das Setup-Skript legt eine Ini-Datei mit folgenden Schlüsseln an:

Schlüssel	**Beschreibung**
ssoserver.hostname	Hostname des SSO-Servers
ssoserver.key	Application-Key des SSO-Servers
ssoserver.url	URL des SSO-Servers: https://...
ssoserver.redirect_url	Eine URL, zu der weitergeleitet werden soll, wenn ein angegebener Redirect nicht validiert werden konnte
ssoserver.be_title	Überschrift im Backend
ssoserver.email.system	E-Mail-Absender, der bei der sendMail-Methode (nicht sendApplicationMail!) benutzt wird; diese Methode wird aktuell nicht benutzt.
ssoserver.cryptkey	Schlüssel für symmetrische Verschlüsselung von Daten
ssoserver.cryptsecret	Präfix für Verschlüsselung von Passwörtern in der Datenbank
ssoserver.shadowmethods_password	Zusätzliches Passwort für Zugriff auf sensible API-Methoden.
ssoserver.password_crypt_function	sha1/md5/md5_2/blowfish (letzteres nicht intensiv getestet)
ssoserver.activatedays	Zeit in Tagen, bevor ein Cleanup-Skript nicht aktivierte Accounts löscht.
ssoserver.login.type	Welche Felder kann der User zum Einloggen benutzen? userLogin oder userEmail oder “userLogin,userEmail”. Dieses wird in einer SQL-Query genauso benutzt.
ssoserver.login.type_admin	adminUserLogin oder adminUserEmail
ssoserver.login.multi_login	Legt fest, ob das Zählen von parallelen Logins aktiviert ist oder nicht. Der Wert 1 aktiviert diese Funktion.
ssoserver.login.max_login_attempts	Wie viele Loginversuche kann ein User

	machen, bevor er gesperrt wird?
ssoserver.login.max_login_duration_locked	Zeit in Sekunden, für wie lange ein User gesperrt wird, wenn er zu viele Login-Fehlversuche hatte.
ssoserver.login.use_recent_session_age	Zeit in Sekunden zwischen zwei Logins, für welche ein User dieselbe Login-Session zugewiesen bekommt. Default: 0 (heißt: jeder Login erhält seine eigene Login-Session). Dies ist ein Workaround für Systeme, bei denen sich eine Applikation über die API anmelden muss.
ssoserver.apidoc.user	Username für die /soapClient.php und /jsonClient.php Testscripten sowie das Skript /frontend/ msp_sso_api_description.php
ssoserver.apidoc.password	Passwort für den Zugriff auf selbige
ssoserver.token_valid_time	Zeit in Sekunden, wie lange eine Login-Session normalerweise gültig ist.
ssoserver.token_rememberme_time	Zeit in Sekunden, wie lange eine mit “rememberMe” erstellte Login-Session gültig ist.
ssoserver.tmp_token_valid_time	Gültigkeit in Sekunden für temporäre Token für z.B. Anfordern eines neuen Passwortes.
ssoserver.cookie.name	Name des Login-Master-Cookies. Dieses ist nur auf dem SSO-Server sichtbar.
ssoserver.cookie.valid_time	Zeit in Sekunden, wie lange dieses Cookie gültig ist.
ssoserver.cookie.loggedin	Name des Login-Cookies, welches domainweit anzeigt, dass der User eingeloggt ist.
ssoserver.cookie.loggedin_domain	Domain, für welche dieses Cookie gesetzt warden soll.
; ssoserver.addons.sap.path = addon/vnp/sap ; ssoserver.addons.hfs.path = addon/vnp/hfs	Pfade aktivierter Addons (kundenspezifische Erweiterungen)
ssoserver.max_subaccount	Anzahl Unteraccounts, die ein Kunde selber anlegen kann. Im Backend können mehr als diese Zahl angelegt warden.
ssoserver.blacklist_mail_domains	Datei in /class/helper/ mit Domainnamen, für welche keine Registrierungen akzeptiert werden.
pages.pagination	Anzahl von Listen-Einträgen im Backend pro Seite
pages.fields.salutation	Anrede für Registrierung
pages.fields.title	Titel für Registrierung
pages.fields.country	Land für Registrierung

soapserver.wsdl	URL der WSDL-Datei für die SOAP-API
soapserver.url	URL des SOAP-Servers
jsonapi.url	URL der JSON-API
database.type	Datenbank-Typ; derzeit unterstützt werden mysql und sqlite; weitere Datenbanken, die von PDO unterstützt werden, können einfach implementiert werden, indem die Datenbank-Basis-Klassen abgeleitet und wo nötig überschrieben werden.
database.host	Zugriffs-Parameter für den Datenbank-Zugriff.
database.name	
database.user	
database.password	
; debug.oauth2 = 1	Debugging für OAuth2
; api.log = database	Logging der API-Nutzung. Parameter werden verschlüsselt abgelegt. Gültige Werte: database, file, database|file. Falls in ein File geloggt wird, wird die Datei „tmp/api.log“ benutzt.
monitor.user	Username für einen SSO-User, der für das Monitoring der Login-Funktion benutzt werden kann. Die Monitoring-Skripten sind im /monitoring-Verzeichnis abgelegt.
monitor.password	Passwort des Users.

In der Datei config.php werden weitere Konstanten definiert, die normalerweise nicht verändert werden müssen. Aus der Ini-Datei wird nach jeder Änderung automatisch eine PHP-Datei erzeugt, welche vom SSO-Server benutzt wird.

1.5 Links

Nach der erfolgreichen Installation finden Sie die einzelnen Komponenten unter folgenden URLs:

WSDL:	https://<SSO_SERVER>/wsdl/api.wsdl
JSON:	https://<SSO_SERVER>/json/api.php
	https://<SSO_SERVER>/json/authenticate.php
OAuth2:	https://<SSO_SERVER>/oauth2/authorize.php
	https://<SSO_SERVER>/oauth2/token.php
	https://<SSO_SERVER>/oauth2/ssoapi.php
Backend:	https://<SSO_SERVER>/backend/index.php
Frontend:	https://<SSO_SERVER>/frontend/index.php
Beispiele:	https://<SSO_SERVER>/frontend/overview.php
SOAP-Client:	https://<SSO_SERVER>/soapClient.php

JSON-Client: https://<SSO_SERVER>/jsonClient.php
SSO API Doku: https://<SSO_SERVER>/frontend/msp_sso_api_description.php

2. Token

Der Token (auch Authentifizierungstoken oder TokenID genannt) ist eine zentrale Komponente des MSP SSO-Systems. Der Token wird in der Datenbank abgelegt und ordnet einem Benutzer eine Login-Session zu. Er dient zur Prüfung ob ein Benutzer bereits über eine Applikation angemeldet ist. Ungültige oder abgelaufene Token (z. B. weil ein Benutzer sich nicht abgemeldet hat) sollten in regelmäßigen Intervallen automatisiert über einen Job gelöscht werden. Dieser Job ist entsprechend einzurichten (z.B. in einer crontab).

Neben dem Authentifizierungstoken existiert noch ein temporärer Token. Dieser ist ebenfalls zeitlich begrenzt gültig und wird für die Aktivierung eines Benutzerkontos oder für die Funktionen „Passwort vergessen“ und „E-Mail-Adresse ändern“ verwendet und kann nur einmal benutzt werden. Ein temporärer Token kann über die MSP SSO-API angefordert werden.

3. MSP SSO Cookie

Damit eine domainübergreifende Authentifizierung und Validierung erfolgen kann, setzt der SSO-Server nach erfolgreicher Authentifizierung eines Benutzers ein Cookie (auch Master-Cookie genannt). Dieses Cookie beinhaltet verschlüsselte Informationen, die nur der SSO-Server entschlüsseln kann, die für weitere Authentifizierung und Validierung des Benutzers notwendig ist.

Nach Beendigung einer Sitzung (Abmeldung durch den Benutzer) wird dieses Cookie gelöscht. Sollte sich der Benutzer nach einer Sitzung nicht korrekt abmelden, verliert das Cookie nach Ablauf eines zuvor definierten Zeitraumes die Gültigkeit. Bei einem Gerätewechsel (beispielsweise Desktop-PC zu Tablet) ist eine Neuanmeldung erforderlich. Dabei wird ein neues Cookie gesetzt.

4. Frontend-Authentifizierung

Die zentrale Login-Session eines eingeloggten Users wird mit dem Login-Token in der SSO-Datenbank gespeichert; der User wird anhand eines Login-Cookies identifiziert (Name im SSO konfigurierbar, normalerweise „<Kürzel>sso“), welches den Login-Token verschlüsselt speichert. Dieses Login-Cookie ist ein Host-Cookie; damit Applikationen, die unter derselben Domain wie der SSO-Server laufen, schneller entscheiden können, ob ein User eingeloggt ist, wird beim Login zusätzlich ein Domain-weites Cookie gesetzt (Name im SSO konfigurierbar, normalerweise „<Kürzel>sso_loggedin“) und beim Logout gelöscht.

Für ein Single-Sign-Login ist es deshalb *notwendig*, dass der *Browser* des Users sich am SSO-Server einloggt, und nicht die Applikation. Die Kommunikation am SSO-Server übernimmt dabei entweder das Skript https://<SSO_SERVER>/json/authenticate.php (JSONP-Aufrufe) oder das Frontend-Login-Formular https://<SSO_SERVER>/frontend/login.php?service=<Rücksprung-URL>.

4.1 JSONP-Login-Schnittstelle

Die Login-Schnittstelle für das JSONP-Protokoll bildet das Skript https://<SSO_SERVER>/json/authenticate.php. Dieses Skript kann mit folgenden Parametern aufgerufen werden – bevorzugt mit dem JSONP-Protokoll:

Einloggen eines Users
`action=authenticate&userLogin=<Benutzername>&userPass=<Benutzerpasswort>`

Login-Status überprüfen
`action=validate`

Logout
`action=logout`

Falls dieses Skript nicht über JSONP angesprochen wird, *muss* eine Rücksprung-URL als Parameter mit dem Namen `service=<URL>` übergeben werden; diese URL wird mit folgenden Parametern als Ergebnis aufgerufen:

Erfolgsfall: `mspsso_action=validate&mspsso_token=<Login-Token>`

Fehlerfall: `mspsso_action=validate&mspsso_error=<Fehler-Code>`

4.2 Login über das Frontend-Login-Formular

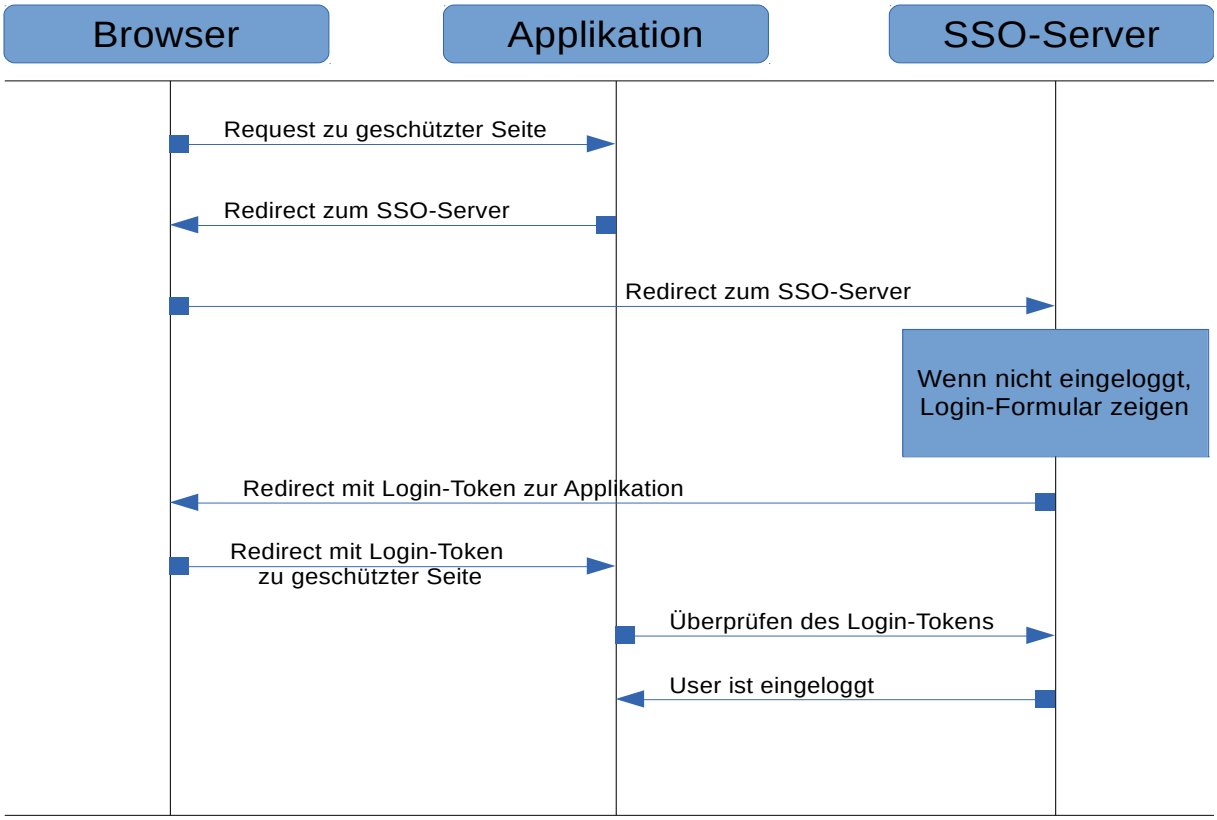
Um den Login-Status eines Users zu ermitteln, kann die JSONP-Schnittstelle mit Rücksprung-Adresse im service-Parameter benutzt werden; sie antwortet entweder mit dem aktuellen Login-Token des Users oder einem Fehlercode (505); diese beiden Parameter sollten von der Applikation abgearbeitet werden.

Anstelle der JSONP-Schnittstelle für das Login kann auch das Frontend-Formular unter http://<SSO_SERVER>/frontend/login.php?service=<URL> mit Rücksprung-Adresse aufgerufen werden, um ein Login eines Users zu erzwingen.

4.3 Beispiel-Kommunikation zwischen Client und Server

Für die Frontend-Authentifizierung ist im Verzeichnis „/client“ eine Demo-Client-Anwendung abgelegt. Diese besteht aus einer PHP-Library, die serverseitig von Applikationen eingebunden werden kann, die das MSP SSO benutzen sollen, sowie einer Javascript-Library, welche die Kommunikation zwischen Browser und SSO-Server übernimmt. (Für die Kommunikation mit dem SSO-Server ist die Javascript-Library nicht unbedingt nötig.)

In folgendem Beispiel ist der Kommunikationsfluss zwischen Browser, Applikation und SSO-Server dargestellt, für den Fall, dass die Applikation ein Login erfordert.



Eine kurze Beschreibung der Schritte:

1. Der User ruft im Browser eine durch Passwort geschützte Seite auf.
2. Die Applikation kann (in diesem Beispiel) nicht verifizieren, ob der User bereits eingeloggt ist, und erzeugt einen Redirect zum SSO-Server.
3. Der SSO-Server stellt jetzt entweder fest, dass der User noch nicht eingeloggt ist, in diesem Fall wird dem User ein Login-Formular angezeigt,
4. oder aber, dass der User bereits eingeloggt ist, in diesem Fall antwortet der SSO-Server mit einem Redirect zur Applikation; dieser Redirect beinhaltet ein Login-Token
5. Die Applikation überprüft jetzt dieses Login-Token durch eine Anfrage beim SSO-Server
6. Der SSO-Server bestätigt, dass der User eingeloggt ist.

Ab jetzt kann die Applikation beim SSO-Server die für die Applikation freigegebenen Daten abrufen.

Einige Beispiele mit realer Kommunikation finden sich im Anhang.

4.4 Die Client-Library

Die Bibliothek „mspsso/mspsso.lib.php“ definiert eine Klasse „MSP_SSO_Client“ mit folgenden Methoden; diese Bibliothek kann als Grundlage für die Integration des MSP SSO benutzt werden.

- public static function getInstance();
  liefert eine Instanz der Klasse, über welche alle Anfragen laufen sollten.
- public function configure($config) ;
  konfiguriert den SSO-Client. $config ist ein Array, in dem folgende Werte übergeben werden müssen:

host: die URL des SSO-Servers
appkey: der Applikations-Schlüssel, der der Applikation zugeteilt wurde. Der „appkey“ kann aber auch später noch übergeben werden, er ist hier optional.

- public function authenticate($user, $pass);
  authentifiziert einen User am SSO-Server. Diese Methode sollte nicht benutzt werden, da sie den Single-Sign-On-Mechanismus aushebelt – damit ist der User nur an dieser Applikation eingeloggt.
- public function isAuthenticated();
  Prüft, ob der User bereits eingeloggt ist. Liefert die Token-ID zurück, falls er eingeloggt ist, false sonst.
- public function getUserData();
  Holt User-Daten vom SSO-Server. Liefert Array mit Userdaten falls User eingeloggt ist, false sonst.
- public function validateToken();
  Validiert ein User-Token. Liefert den Loginname falls User eingeloggt ist, false sonst.
- public function checkAuthentication();
  Validiert ein User-Token. Falls dies der erste Besuch dieser Seite ist oder die (interne) Lifetime des Token abgelaufen ist, Token einmal beim SSO-Server anfordern. Liefert Loginname falls User eingeloggt ist, false sonst.
- public function setAttribute($name, $val, $status=1, $period='');
  Applikations-spezifisches User-Attribut setzen; Attribut muss im SSO bereits angelegt sein.
  $name: (string) Name des Attributs
  $val: (string) Wert des Attributs
  $status: (int) setzt Status des Attributs: 1=verfügbar, 0=gesperrt
  $period: (string) Gültig bis, Format 'YYYY-MM-DD HH:MM:SS' oder '' für unbegrenzt
  Liefert ein Array mit dem Ergebnis zurück.
- public function forceAuthentication();
  Erzwingt ein Login am SSO-Server. Die Methode generiert einen Redirect zum SSO-Server, wenn der User nicht eingeloggt ist, und überprüft ansonsten mittels checkAuthentication(), ob der User noch eingeloggt ist.

Im Verzeichnis „client/test/“ befinden sich Beispiele, wie diese Bibliothek in bestehende Applikationen eingebunden werden kann. Der Aufruf von $ssoclient->configure() klinkt sich in die Abarbeitung eines Request ein und filtert die für das SSO bestimmten Request-Parameter aus dem Request aus.

Für Java-Applikationen existiert eine entsprechende Java-Library. Informationen dazu finden Sie im Dokument MSP-SSO_Java-Client.pdf, welches wir auf Anfrage gerne zur Verfügung stellen.

5. Wissenswertes

5.1 Attribute

Das MSP SSO bietet angebundenen Applikationen die Möglichkeit, zusätzliche Benutzerinformationen abzulegen, die entweder nicht von der Applikation gespeichert werden können bzw. gespeichert werden sollen. Diese Inhalte werden in sogenannte Attribute gespeichert. Jedes Attribut kann einen alphanumerischen Wert von einer Länge von bis zu max. 250 Zeichen

speichern und kann jedem Benutzerkonto zugeordnet werden. Beispielsweise soll ein Benutzer nach Bestellung eines E-Paper-Abos eine vorübergehende Leseberechtigung erhalten, während die Überprüfung seiner Daten noch läuft. Die Auswertung der Attribute erfolgt in der Regel aufseiten der Applikation. Das MSP SSO stellt hier lediglich den Inhalt zur Verfügung.

Attribute werden zwischen applikationsabhängige und globale Attribute unterschieden und müssen vor Anwendung einmalig im Backend angelegt werden (siehe dazu Kapitel 8.2.2 und Kapitel 8.3.2). Applikationsabhängige Attribute werden für einzelne Applikationen angelegt und können auch nur von diesen gelesen und geschrieben werden. Globale Attribute dagegen stehen jeder Applikation zur Verfügung. Diese Attribute sind dem SSO-Server zugeordnet.

Attribute können nicht durch einen Benutzer angelegt oder gelöscht werden.

5.2 Unterkonten

Um das vielfach auftretende Phänomen der Mehrfachnutzung von Accounts einzuschränken, bietet das MSP SSO die Möglichkeit, zu regulären Konten sogenannte Unterkonten anzulegen. Ein Unterkonto erbt dabei alle Rechte des zugehörigen Hauptkontos. Einsetzen kann man dies z.B. als Mitleser-Konto für Familienmitglieder, Firmen, Schulklassen.

Unterkonten können vom Hauptkonto aus angelegt und gelöscht werden. In der Konfiguration des SSO-Servers wird festgelegt, wieviele Unterkonten von einem Hauptkonto aus angelegt werden können (in der Regel drei – im Backend des SSO-Servers ist die Zahl der Unterkonten zu einem Hauptkonto nicht beschränkt).

Technisch wird die Vererbung der Rechte dadurch gelöst, dass bei Login eines Unterkontos die GP-Nummer des zugehörigen Hauptkontos ausgelesen und benutzt wird. Dadurch dass es sich um ein Unterkonto handelt, lassen sich im Weiteren auch die Beschränkungen leicht umsetzen, z.B. dass ein Unterkonto keine Transaktionen tätigen darf.

Der Status, dass ein Konto ein Unterkonto ist, kann beim Aufruf der API an die Methoden createUser(), updateUser() und upgradeUser() übergeben werden. Der Status lässt sich über die Methode getUser() ermitteln.

6. MSP SSO-API

Die MSP SSO-API stellt eine Reihe von Methoden zur Kommunikation – die SSL verschlüsselt ist – mittels SOAP zwischen Applikation und SSO-Server zur Verfügung. Des Weiteren kann die Kommunikation auch über eine JSON-Schnittstelle verlaufen. Hier erfolgt die Abfrage der Benutzerdaten über einen POST-Request.

Damit eine Applikation mit der MSP SSO-API kommunizieren kann, benötigt diese einen sogenannten ApplicationKey (auch appKey genannt), der bei jedem Methoden-Aufruf zusätzlich übergeben werden muss. Der appKey wird vom SSO-Server generiert und der Applikation zur Verfügung gestellt.

6.1 Verfügbare Methoden

Im Folgenden werden die verfügbaren Methoden im Detail beschrieben. Generell gilt: Alle Schlüssel müssen übergeben werden, können aber ggf. leer bleiben, also als Leerstring. Diese Schlüssel sind mit einem * gekennzeichnet.

6.1.1 _testMethod()

Liefert alle gespeicherten Informationen über die anfragende Applikation zurück.

Folgende Schlüssel werden erwartet:

- appKey (ApplikationKey)

Rückgabecodes:

- 700 – OK
- 300 – Applikations-Schlüssel wurde nicht übergeben
- 301 – Applikations-Schlüssel ist ungültig
- 801 – unbekannter Parameter beim Aufruf benutzt

Rückgabewerte bei Code 700:

application	applicationId
	applicationName
	applicationDomain
	applicationKey
	applicationRedirect
	applicationTokenValidity
	applicationEmail
	applicationStatus
	applicationDescription

6.1.2 authenticate()

Authentifiziert einen Benutzer anhand Login und Passwort.

Diese Methode sollte nur in Ausnahmefällen benutzt werden, da sich hiermit die Applikation für den User eingeloggt. Das hat zur Folge, dass andere Applikationen nicht erkennen können, dass der User schon eingeloggt ist.

Folgende Schlüssel werden erwartet:

- appKey (ApplikationKey)
- userLogin (Login des Benutzers)
- userPass (Passwort des Benutzers)

Rückgabecodes:

- 700 – OK (der Benutzer ist damit am SSO eingeloggt)
- 201 – Benutzername oder Passwort falsch
- 202 – Benutzer ist gesperrt.

Rückgabewerte bei Code 700:

user	userId
	userGp
	userName
	userSurname

	userLogin
	userEmail
	userAlias
	userStatus
	userComment
	isSubAccount
	subAccountId
	subAccountMainId
	accountCreateOn
	accountActivateOn
	lastLogin
	tokenId

6.1.3 validateToken()

Prüft ob der Authentifizierungstoken gültig ist.

Folgende Schlüssel werden erwartet:

- appKey (ApplikationKey)
- tokenId (TokenId)

Rückgabecodes:

- 700 – OK
- 501 - Token ist ungültig.

Rückgabewerte bei Code 700:

user	userLogin

6.1.4 getLoginCount()

Liefert zurück wie oft ein Benutzer bereits eingeloggt ist. Diese Methode steht nur dann zur Verfügung, wenn das Zählen von Logins in der Konfiguration eingestellt wurde. Standardmäßig liefert diese Methode ansonsten nur den Wert 0 zurück.

Folgende Schlüssel werden erwartet:

- appKey (ApplikationKey)
- userId (UserId)

Rückgabecodes:

- 700 – OK
- 302 – Applikation gesperrt.
- 601 – Es liegen keine Attribute für diesen Benutzer vor.

attribute	0…n	attributeName
		attributeDataId
		attributeId
		userId

		attributeDataValue
		attributeDataPeriod
		attributeDataStatus
		applicationId

6.1.5 logoutUser()

Meldet einen Benutzer vom SSO-Server ab.

Folgende Schlüssel werden erwartet:

- appKey (ApplikationKey)
- tokenId (TokenId)

Rückgabecodes:

- 700 – OK
- 206 – Logout fehlgeschlagen.

Rückgabewerte bei Code 700: Keine.

6.1.6 getUserData()

Ruft alle verfügbaren Daten des Benutzers vom SSO-Server ab. Je nach Einstellung kann der Zugriff der Daten[2] gegebenenfalls eingeschränkt sein (siehe dazu Kapitel 8.3.1).

Folgende Schlüssel werden erwartet:

- appKey (ApplikationKey)
- tokenId (TokenId)

Rückgabecodes:

- 700 – OK
- 204 – Benutzer unbekannt.
- 501 – Token ist ungültig.

Rückgabewerte bei Code 700:

Welche Benutzerdaten zurückgeliefert werden, hängt von der jeweiligen Freigabe im Backend ab. Daher besteht die Möglichkeit, dass nicht alle unten aufgeführten Felder zurückgeliefert werden.

user	userId	
	userGp	
	userName	
	userSurname	
	userLogin	
	userEmail	
	userAlias	
	userStatus	
	userComment	

[2] Im Backend kann für jede Applikation definiert werden, welche Daten das SSO zurückliefern soll.

	isSubAccount	
	subAccountId	
	subAccountMainId	
	accountCreateOn	
	accountActivateOn	
	lastLogin	
	userDataSalutation	
	userDataTitle	
	userDataStreet	
	userDataStreetnumber	
	userDataStreetaddon	
	userDataZipcode	
	userDataCity	
	userDataCountry	
	userDataPhone	
	userDataCell	
	attribute	0…n

6.1.7 checkUserLoginExist()

Prüft ob Login schon vorhanden.

- appKey (ApplicationKey)
- userLogin (Login des Benutzers)
- userId (default 0; Soll das Login nachträglich geändert werden, muss die UserID des betroffenen Benutzers übergeben werden.)

Rückgabecodes:

- 700 – OK
- 212 – Login bereits vorhanden.

Rückgabewerte bei Code 700: Keine.

6.1.8 checkUserEmailExist()

Prüft ob eine E-Mail-Adresse schon vorhanden ist. Dabei werden auch sogenannte Junk-E-Mail Adressen (wie z. B. Mailinator, Trash-Mail, usw.)[3] geprüft und ggf. blockiert.

Folgende Schlüssel werden erwartet:

- appKey = applicationKey
- userEmail = Benutzer Login
- userId = userId (default 0, Soll das Login nachträglich geändert werden, muss die UserId des betroffenen Benutzers übergeben werden.)

Rückgabecodes:

- 700 – OK
- 217 – E-Mail-Adresse ist bereits vergeben oder ungültig.

[3] Derzeit werden 414 Anbieter von Junk-E-Mail Adressen blockiert.

Rückgabewerte bei Code 700: Keine.

6.1.9 checkUserExistInSap()

Prüft anhand bestimmter Felder (z.B. Name, Vorname, Ort, usw.), ob es einen Benutzer in SAP schon gibt. Falls ja, wird die GP zurückgeliefert.

Folgende Schlüssel werden erwartet:

- appKey = applicationKey
- userSalutation = Anrede (Frau, Herr)
- userName = Vorname des Benutzers
- userSurname = Nachname des Benutzers
- userStreetnumber = Hausnummer (z. B. 15a)
- userZipcode = Postleitzahl (z. B. 26121, 01723)
- userCity = Stadt (z. B. Oldenburg)
- userCountry = Land (z. B. Deutschland, Schweiz)

Rückgabecodes:

- 700 – OK
- 220 – Geschäftspartner-Nr. konnte nicht ermittelt werden.

Rückgabewerte bei Code 700:

user	userGp

6.1.10 createUser()

Legt einen neuen Benutzer im SSO-Server an. Es wird kein neues Benutzerkonto angelegt, wenn das Login oder die E-Mail-Adresse bereits vorhanden sind. Diese Prüfungen können separat über die Methoden checkUserLoginExist() (siehe dazu Kapitel 6.1.7) und checkUserEmailExist() (siehe dazu Kapitel 6.1.8) aufgerufen werden.

Folgende Schlüssel werden erwartet:

- appKey (ApplikationKey)
- userGp (Geschäftspartner-Nr. wenn vorhanden, default 0)
- userName (Name des Benutzers)
- userSurname (Nachname des Benutzers)
- userLogin (Login des Benutzers)
- userPass (Passwort des Benutzers)
- userEmail (E-Mail des Benutzers)
- userAlias * (Alias, optional)
- userStatus (1 = nicht gesperrt, 0 = gesperrt)
- userComment * (Anmerkung zum Benutzer, optional)
- isSubAccount (1 = Unterkonto, 0 = Hauptkonto)
- subAccountId (0...n, default 0)
- subAccountMainId (userId des Hauptkonto-Eigentümer, default 0)
- userSalutation (Anrede Frau oder Herr)
- userTitle (Titel, z. B. Dr., Prof., Prof. Dr., default null)

- userStreet (Straße, z.B. Max-Planck-Str.)
- userStreetnumber (Hausnummer, z. B. 15a)
- userStreetaddon * (Adresszusatz, z. B. linke Seite, default null)
- userZipcode (Postleitzahl, z. B. 26121, 01723)
- userCity (Stadt, z. B. Oldenburg)
- userCountry (Land, z. B. Deutschland, Schweiz)
- userPhone * (Festnetz, z. B. 0441/223344, +49 441 223344, default null)
- userCell * (Mobil, z. B. 0172/223344, +49 1722 23344, default null)
- userCompany (Firma, default null)
- sendActivateLink * (Aktivierungslink versenden, optional; Parameter enthält dann den Email-Key des Email-Templates, welches benutzt werden soll)
- activateLink * (diesen Aktivierungslink benutzen, optional; default wird /frontend/password.php benutzt, hierüber kann der User sich ein Passwort setzen)

Rückgabecodes:

- 700 – OK
- 208 – Fehler beim Anlegen eines Benutzers.
- 212 – Login bereits vorhanden.
- 217 – E-Mail-Adresse ist bereits vergeben oder ungültig.

Rückgabewerte bei Code 700:

user	userGp

6.1.11 sendActiveLink()

Verschickt eine E-Mail mit einem Aktivierungslink zur Aktivierung eines Benutzerkontos.

Folgende Schlüssel werden erwartet:

- appKey = applicationKey
- userEmail = Benutzer E-Mail Adresse
- activateUrl = Link zur Aktivierungsseite
- mailContentId = ID des E-Mail Textes das im SSO-Server angelegt wurde

Rückgabecodes:

- 700 – OK
- 216 - E-Mail-Adresse ist unbekannt.
- 219 - Benutzer wurde bereits aktiviert.
- 802 - Aktivierungs-E-Mail konnte nicht verschickt werden.

Rückgabewerte bei Code 700: Keine.

6.1.12 activateUser()

Aktiviert einen Benutzer nach der Registrierung im SSO-Server.

Folgende Schlüssel werden erwartet:

- appKey (ApplikationKey)
- userId (UserId des Benutzers)

Rückgabecodes:

- 700 – OK
- 210 – Benutzer konnte nicht aktiviert werden.

Rückgabewerte bei Code 700: Keine.

6.1.13 activateUserByTmpToken()

Aktiviert einen Benutzer nach der Registrierung im SSO-Server.

Folgende Schlüssel werden erwartet:

- appKey (ApplikationKey)
- tmpTokenValue (temporäres Token)

Rückgabecodes:

- 700 – OK
- 210 – Benutzer konnte nicht aktiviert werden.

Rückgabewerte bei Code 700: Keine.

6.1.14 setUserStatus()

Setzt den Status des Benutzers.

Folgende Schlüssel werden erwartet:

- appKey (ApplikationKey)
- userId (UserId des Benutzers)
- userStatus (1 = nicht gesperrt, 0 = gesperrt)

Rückgabecodes:

- 700 – OK
- 211 - Benutzerstatus konnte nicht aktualisiert werden.

Rückgabewerte bei Code 700: Keine.

6.1.15 updateUser()

Aktualisiert die Daten des Benutzers.

Folgende Schlüssel werden erwartet:

- appKey (ApplikationKey)
- userId (UserId des Benutzers)
- userGp (Geschäftspartner-Nr. sofern vorhanden, default 0)
- userName (Name des Benutzers)
- userSurname (Nachname des Benutzers)
- userAlias * (Alias, optional)

- userComment (Anmerkung zum Benutzer, optional)
- isSubAccount (1 = Unterkonto, 0 = Hauptkonto)
- subAccountId (0...n, default 0)
- subAccountMainId (userId des Hauptkonto-Eigentümer, default 0)
- userSalutation (Anrede Frau oder Herr)
- userTitle (Titel, z. B. Dr., Prof., Prof. Dr., default null)
- userStreet (Straße, z.B. Max-Planck-Str.)
- userStreetnumber (Hausnummer, z. B. 15a)
- userStreetaddon * (Adresszusatz, z. B. linke Seite, default null)
- userZipcode (Postleitzahl, z. B. 26121, 01723)
- userCity (Stadt, z. B. Oldenburg)
- userCountry (Land, z. B. Deutschland, Schweiz)
- userPhone * (Festnetz, z. B. 0441/223344, +49 441 223344, default null)
- userCell * (Mobil, z. B. 0172/223344, +49 1722 23344, default null)

Rückgabecodes:

- 700 – OK
- 209 - Benutzerdaten wurden nicht aktualisiert.

Rückgabewerte bei Code 700: Keine.

6.1.16 upgradeUser()

Wandelt ein Unterkonto zu einem Hauptkonto um.

Folgende Schlüssel werden erwartet:

- appKey (ApplicationKey)
- userId (userId des Benuters)
- subAccountMainId (userId des Hauptkontoinhabers)

Rückgabecodes:

- 700 – OK
- 218 - Das Unterkonto konnte nicht hochgestuft werden.

Rückgabewerte bei Code 700: Keine.

6.1.17 updateUserGp()

Aktualisiert die Geschäftspartner-Nr. eines Benutzers.

Folgende Schlüssel werden erwartet:

- appKey (ApplicationKey)
- userId (userId des Benuters)
- userGp (Geschäftspartner-Nr. des Benutzers)

Rückgabecodes:

- 700 – OK
- 221 - Geschäftspartner-Nr. konnte nicht aktualisiert werden.

Rückgabewerte bei Code 700: Keine.

6.1.18 updateUserEmail()

Aktualisiert die E-Mail Adresse des Benutzers.

Folgende Schlüssel werden erwartet:

- appKey (ApplicationKey)
- tmpTokenValue (temporaerer Token, siehe dazu Kapitel 6.1.26)
- userId (UserId des Benutzers)
- userEmail (E-Mail Adresse des Benutzers)

Rückgabecodes:

- 700 – OK
- 213 - E-Mail-Adresse konnte nicht aktualisiert werden.
- 510 - Temporärer Token ist ungültig.

Rückgabewerte bei Code 700: Keine.

6.1.19 updateUserPassword()

Aktualisiert das Passwort des Benutzers.

Folgende Schlüssel werden erwartet:

- appKey (ApplicationKey)
- tmpTokenValue (temporärer Token, siehe dazu Kapitel 6.1.26)
- userId (UserId des Benutzers)
- userPass (neues Passwort des Benutzer)

Rückgabecodes:

- 700 – OK
- 214 - Passwort konnte nicht aktualisiert werden.
- 510 - Temporärer Token ist ungueltig.

Rückgabewerte bei Code 700: Keine.

6.1.20 forgetUserPassword()

Verschickt eine E-Mail mit einem Link zum Setzen eines neuen Passworts.

Folgende Schlüssel werden erwartet:

- appKey (ApplicationKey)
- userEmail (E-Mail Adresse des Benutzers)

Rückgabecodes:

- 700 – OK

- 216 – E-Mail-Adresse ist unbekannt.
- 302 – Applikation gesperrt.
- 304 – Es konnten keine passenden E-Mail Texte für diese Applikation gefunden werden.

Rückgabewerte bei Code 700: Keine.

6.1.21 createUserAttribute()

Legt für einen Benutzer ein Attribut an.

Folgende Schlüssel werden erwartet:

- appKey (ApplicationKey)
- global (0 = nur Attribut der Applikation, 1 = globales Attribut)
- attrId (AttributId)
- userId (UserId des Benutzers)
- attrValue (Wert des Attributs)
- attrPeriod (JJJJ-MM-TT, unbefristet gültig = 0)
- attrStatus (Attribut Status, 1 = nicht gesperrt, 0 = gesperrt)

Rückgabecodes:

- 700 – OK
- 302 – Applikation konnte nicht authentifiziert werden.
- 603 – Attribut für diese Applikation existiert nicht.
- 604 – Attribut existiert bereits fuer diesen Benutzer
- 605 – Attribut kann für diesen Benutzer nicht angelegt werden.
- 608 - Dieser Benutzer existiert nicht.

Rückgabewerte bei Code 700: Keine.

6.1.22 deleteUserAttribute()

Löscht das Attribut eines Benutzers.

Folgende Schlüssel werden erwartet:

- appKey (ApplicationKey)
- userId (UserId des Benutzers)
- attrId (AttributId)

Rückgabecodes:

- 700 – OK
- 606 – Attribut kann für diesen Benutzer nicht gelöscht werden.

Rückgabewerte bei Code 700: Keine.

6.1.23 getUserSubscriptions()

Ermittelt die Abonnements eines Benutzers

Folgende Schlüssel werden erwartet:

- appKey (ApplicationKey)
- userGp (Geschäftspartner-Nr. des Benutzers)

Rückgabecodes:

- 700 – OK
- 222 – Es konnten keine Abonnements ermittelt werden.

Rückgabewerte bei Code 700:

user	0…n	pva
		start
		ende
		auftrag

6.1.24 setUserAttribute()

Aktualisiert ein Benutzerattribut.

Folgende Schlüssel werden erwartet:

- appKey (ApplicationKey)
- global (0 nur Attribut der Applikation, 1 = globales Attribut)
- userId = userId
- attrId = attributId
- attrValue = Wert des Attributs
- attrPeriod = JJJJ-MM-TT (unbefristet gültig = 0)
- attrStatus = Attribut Status 1/ 0 (0 = gesperrt)

Rückgabecodes:

- 700 – OK
- 302 - Applikation gesperrt.
- 602 - Attribut konnte nicht aktualisiert werden.
- 608 - Dieser Benutzer existiert nicht.

Rückgabewerte bei Code 700: Keine.

6.1.25 getUserAttribute()

Liefert alle Attribute eines Benutzers.

Folgende Schlüssel werden erwartet:

- appKey (ApplicationKey)
- userId (UserId des Benutzers)

Rückgabecodes:

- 700 – OK
- 302 – Applikation gesperrt.
- 601 – Es liegen keine Attribute für diesen Benutzer vor.

Rückgabewerte bei Code 700:

attribute	0…n	attributeName
		attributeDataId
		attributeId
		userId
		attributeDataValue
		attributeDataPeriod
		attributeDataStatus
		applicationId

6.1.26 getUserAttributeByName()

Liefert ein Attribut eines Benutzers anhand des Attributnamens.

Folgende Schlüssel werden erwartet:

- appKey (ApplicationKey)
- userId (UserId des Benutzers)
- attrName (Name des Attributs)

Rückgabecodes:

- 700 – OK
- 302 – Applikation gesperrt.
- 601 – Es liegen keine Attribute für diesen Benutzer vor.

Rückgabewerte bei Code 700:

attribute	0…n	attributeName
		attributeDataId
		attributeId
		userId
		attributeDataValue
		attributeDataPeriod
		attributeDataStatus
		applicationId

6.1.27 requestTmpToken()

Fordert einen temporären Token für „Neues Passwort“-, „Passwort vergessen“- und „E-Mail Adresse“-Funktionen an. Die Benutzung dieser Einaml-Tokens ist an eine UserId geknüpft.

Folgende Schlüssel werden erwartet:

- appKey (ApplicationKey)
- userId (UserId des Benutzers)

Rückgabecodes:

- 700 – OK
- 511 – Temporärer Token konnte nicht erzeugt werden.

Rückgabewerte bei Code 700:

token	tmpTokenValue

6.1.28 getApplicationAttribute()

Liefert alle Attribute einer Applikation.

Folgende Schlüssel werden erwartet:

- appKey (ApplicationKey)
- global (0 nur Attribute der Applikation, 1 = globale Attribute)

Rückgabecodes:

- 700 – OK
- 302 – Applikation gesperrt.
- 607 – Es existieren keine Attribute kann für diese Applikation.

Rückgabewerte bei Code 700:

attribute	0…n	attributeId
		applicationId
		attributeName
		attributeType
		attributeDescription
		attributeDataStatus
		attributeDataPeriod

6.1.29 getApplicationMailContentById()

Liefert einen bestimmten E-Mail Text anhand der emailId.

Folgende Schlüssel werden erwartet:

- appKey (ApplicationKey)
- emailId (E-Mail Id)
- default (0 Text des SSO-Servers, 1 = Text der anfragenden Applikation)

Rückgabecodes:

- 700 – OK
- 302 – Applikation gesperrt.
- 305 – Die übergebene EmailId ist ungültig oder für diese Applikation existiert der gesuchte E-Mail Text nicht.

Rückgabewerte bei Code 700:

application	emailId
	applicationId
	emailName
	emailSender

	emailSenderAddress
	emailSubject
	emailText

6.1.30 getApplicationMailContentList()

Liefert alle E-Mail Texte einer Applikation.

Folgende Schlüssel werden erwartet:

- appKey (ApplicationKey)

Rückgabecodes:

- 700 – OK
- 302 – Applikation gesperrt.
- 304 – Es konnten keine passenden E-Mail Texte für diese Applikation gefunden werden.

Rückgabewerte bei Code 700:

application	0…n	emailId
		emailName
		emailSender
		emailSenderAddress
		emailSubject
		emailText
		emailBcc
		emailStatus

6.2 Error-Codes

Je nachdem welche Methoden über die API aufgerufen werden, wird im Fehlerfall ein unterschiedlicher Fehlercode zurückgeliefert. Diese sind in 100er Blöcken unterteilt.

6.2.1 100-Codes

- 100 – „Datenbankverbindung OK!“
- 101 – „Fehler beim Aufbau der Datenbankverbindung.“

6.2.2 200-Codes

- 200 – „Authentifikation OK!“
- 201 – „Benutzername oder Passwort falsch.“
- 202 – „Benutzer gesperrt.“
- 203 – „Falsches Passwort.“
- 204 – „Benutzer unbekannt.“
- 205 – „Keine Benutzerdaten verfügbar.“
- 206 – „Logout fehlgeschlagen.“
- 207 – „Logout OK.“
- 208 – „Fehler beim Anlegen eines Benutzers.“
- 209 – „Benutzerdaten wurden nicht aktualisiert.“
- 210 – „Benutzer konnte nicht aktiviert werden.“

- 211 – „Benutzerstatus wurde nicht aktualisiert.“
- 212 – „Benutzername bereits vorhanden.“
- 213 – „E-Mail-Adresse konnte nicht geändert werden.“
- 214 – „Passwort konnte nicht geändert werden.“
- 215 – „Passwort konnte nicht zurückgesetzt werden.“
- 216 – „E-Mail-Adresse ist unbekannt.“
- 217 – „E-Mail-Adresse ist bereits vergeben.“
- 218 – „Das Unterkonto konnte nicht hochgestuft werden.“
- 219 – „Benutzer wurde bereits aktiviert“
- 220 – „Geschäftspartner-Nr. konnte nicht ermittelt werden.“
- 221 – „Geschäftspartner-Nr. konnte nicht aktualisiert werden.“

6.2.3 300-Codes

- 300 – „ApplicationKey wurde nicht übergeben.“
- 301 – „Applikation unbekannt.“
- 302 – „Applikation gesperrt.“
- 303 – „Applikation konnte nicht authentifiziert werden.“
- 304 – „Es konnten keine passenden E-Mail Texte für diese Applikation gefunden werden.“
- 305 – „Die übergebene EmailId ist ungültig oder für diese Applikation existiert der gesuchte E-Mail Text nicht.“

6.2.4 400-Codes

- 401 – „Authentifikation fehlgeschlagen“

6.2.5 500-Codes

- 500 – „Token ist gültig.“
- 501 – „Token ist ungültig.“
- 502 – „Token kann nicht erstellt werden.“
- 503 – „Token kann nicht aktualisiert werden.“
- 504 – „Token-Wert kann nicht ermittelt werden.“
- 505 – „Token kann nicht ermittelt werden.“
- 510 – „Temporärer Token ist ungültig.“
- 511 – „Temporärer Token konnte nicht erzeugt werden.“

6.2.6 600-Codes

- 601 – „Es liegen keine Attribute für diesen Benutzer vor.“
- 602 – „Attribut konnte nicht aktualisiert werden.“
- 603 – „Attribut für diese Applikation existiert nicht.“
- 604 – „Attribut existiert bereits für diesen Benutzer“
- 605 – „Attribut kann für diesen Benutzer nicht angelegt werden.“
- 606 – „Attribut kann für diesen Benutzer nicht gelöscht werden.“
- 607 – „Es existieren keine Attribute kann für diese Applikation.“

6.2.7 700-Codes

- 700 – „OK“

6.2.8 800-Codes

- 801 – „Es wurden unbekannte Parameter übergeben.“
- 802 – „Aktivierungs-E-Mail konnte nicht verschickt werden.“

6.2.9 900-Codes

- 900 – „Test Meldung“

6.3 Kommunikation mittels SOAP

Die WSDL-URI lautet https://<SSO_SERVER_URL>/wsdl/api.wsdl. Einen Test Soap-Client finden Sie unter: https://<SSO_SERVER_URL>/soapClient.php. Um den Test Soap-Client nutzen zu können ist ein Login erforderlich. Verwenden Sie dazu die Backend-Login Anmeldedaten.

Jede Methode erwartet als Parameter ein assoziatives Array. Je nach Methode werden weitere Schlüssel nötig. Eine Kurzbeschreibung welche Methode welche Schlüssel erwartet, finden Sie unter https://<SSO_SERVER_URL>/wsdl/index.php (siehe dazu auch Kapitel 6.1).

Als Rückgabe-Wert liefert die MSP SSO-API immer ein assoziatives Array zurück – entweder die Rückgabe der angeforderten Daten oder eine Fehlermeldung (siehe dazu Kapitel 6.2).

Zwei Beispiele wie ein SOAP-Aufruf in PHP realisiert wird:

Der Aufruf der Methode „__testMethod()“:

```
$params = array(‘appKey‘ => ‘b3aacf7e54c7de4f35368089c1a18d8c‘);
$client = new SoapClient('http://<SSO_SERVER_URL>/wsdl/api.wsdl,
                array('soap_version' => SOAP_1_2));

$res = $client->__testMethod($params);
```

Der Aufruf der Methode „getUserData()“:

```
$params = array(‘appKey‘ => ‘b3aacf7e54c7de4f35368089c1a18d8c ‘,
                ‘tokenId‘ => ‘00000059000182545997137119597 9‘);
$client = new SoapClient('http://<SSO_SERVER_URL>/wsdl/api.wsdl',
                array('soap_version' => SOAP_1_2));

$res = $client->getUserData($params);
```

6.4 Kommunikation mittels JSON

Die Abfrage der JSON-API findet über einen POST-Request statt. Als Ergebnis wird ein JSON-Objekt zurückgeliefert. Die JSON-API finden Sie unter https://<SSO_SERVER_URL>/json/api.php. Einen Test JSON-Client können Sie unter https://<SSO_SERVER_URL>/jsonClient.php aufrufen. Um den Test JSON-Client nutzen zu können ist ein Login erforderlich. Verwenden Sie dazu die Backend-Login Anmeldedaten.

Jede Methode erwartet als Parameter ein JSON-Objekt. Je nach Methode werden weitere Schlüssel nötig. Eine Kurzbeschreibung welche Methode welche Schlüssel erwartet, finden Sie unter https://<SSO_SERVER_URL>/wsdl/index.php (siehe dazu auch Kapitel 6.1).

Hier ein Beispielaufruf der Methode „__testMethod()“ mit jQuery:

```
var params = {
 'method': '__testMethod',
 'appKey': 'b3aacf7e54c7de4f35368089c1a18d8c'
}

$.post('https://<SSO_SERVER_URL>/json/api.php',
   params,
   function(result) {
      if (result) {
         console.log(result.application.applicationName);
      }
}, 'json');
```

Achtung: Es handelt sich hierbei um ein Beispiel. Aus Sicherheitsgründen sollten Sie zu keiner Zeit den ApplicationKey im HTML/JavaScript-Quellcode hinterlegen.

6.5 Kommunikation mittels OAuth2-Protokoll

Das MSP SSO unterstützt das OAuth2-Protokoll. Am SSO-Server sind dafür drei OAuth2-Endpunkte eingerichtet, die im Folgenden beschrieben werden.

6.5.1 Der Authorize-Endpunkt

Über den Authorize-Endpunkt kann sich der User (im RFC ist das der Resource-Owner) einloggen. Der MSP-SSO-Server ist so konfiguriert, dass der anfragenden Applikation der Zugriff auf die Daten des Users (Authorization Grant) implizit gegeben wird.

URL: https://<SSO-SERVER>/oauth2/authorize.php

GET-Parameter:

- response_type=code
- client_id=<Client-ID>
- redirect_uri=<registrierte Rücksprung-URL>
- state=<frei wählbarer Status> (optional)
- scope=<Scope-Parameter> (optional)

6.5.2 Der Token-Endpunkt

Über den Token-Endpoint am SSO-Server tauscht die Client-Applikation den vom Authorize-Endpunkt erhaltenen Authorisierungs-Code gegen einen länger gültigen Zugriffs-Token (access token) sowie ein „Erneuerungs“-Token (refresh token) aus.

URL: https://<SSO-SERVER>/oauth2/token.php

POST-Parameter:

- grant_type=authorization_code
- code=<code von Authorize-Endpunkt>
- redirect_uri=<registrierte URL, die auch im Authorize-Endpunkt benutzt wurde>
- client_id=<Client-ID>
- client_secret=<Client-Secret>

client_id und client_secret können als HTTP-Basic-Authorization-Header oder ebenfalls als POST-Parameter mit übergeben werden

6.5.3 Der API-Endpunkt

Der API-Endpunkt (oder Resource-Endpunkt) gewährt der anfragenden Applikation mit dem aus dem Token-Endpunkt erhaltenen access token Zugriff auf die Daten des eingeloggten Users. Dafür können die in Kapitel 6.1 beschriebenen Methoden benutzt werden.

URL: https://<SSO-SERVER>/oauth2/ssoapi.php

POST-Parameter:

- access_token=<access_token aus Token-Endpoint>
- method=<API-Methode>
- appKey=<Client-Secret>
- weitere laut API-Dokumentation erforderliche Parameter
- wo die API einen Parameter tokenId erfordert, wird dieser weggelassen; die tokenId ist mit dem access_token verknüpft.

Als Ergebnis wird ein JSON-String geliefert mit dem Rückgabe-Wert des Aufrufs.

6.5.4 Zusammenspiel der Endpunkte

Bevor eine Applikation Zugriff auf den SSO-Server bekommen kann, muss die Applikation mit einer oder mehreren Redirect-URLs im SSO-Server eingetragen werden. Für den Zugriff auf die API benötigt die Applikation (im Weiteren auch Client genannt) dann eine *Client-ID* und ein *Client-Secret*. Beides wird von einem Administrator des SSO-Servers zur Verfügung gestellt.

Der Authorize-Endpunkt wird in der Applikation (z.B. als Link) wie folgt benutzt:

https://<SSO_SERVER>/oauth2/authorize.php?response_type=code& client_id=<Client-ID>&redirect_uri=<registrierte Redirect-URL>

Klickt der User auf den Link, erscheint die Login-Maske des SSO-Servers, falls der User noch nicht eingeloggt ist. Nach dem Login erfolgt der Redirect zur übergebenen Redirect-URL. Ist der User schon eingeloggt, erfolgt der Redirect sofort.

Als Antwort auf einen erfolgreichen Login bekommt der Client über den Parameter `code` einen Zugriffs-Code übermittelt, den er innerhalb der Gültigkeitsdauer des Codes (1 Minute) mit einem POST-Request am Token-Endpunkt gegen ein Access-Token eintauschen kann. Die Antwort sieht z.B. wie folgt aus:

http://<registrierte_Redirect_URL>?code=8681ae7ebc92f32ffb3e7d866921cc4b2dfa576e

Danach setzt der Client folgenden POST-Request für den Eintausch eines Access-Tokens ab:

https://<SSO_SERVER>/oauth2/token.php

POST-Parameter:

```
grant_type=authorization_code&
code=8681ae7ebc92f32ffb3e7d866921cc4b2dfa576e&
redirect_uri=<registrierte_Redirect_URL>&
client_id=<Client-ID>&
client_secret=<Client-Secret>
```

Der SSO-Server antwortet mit einem JSON-String, in dem das Access-Token übergeben wird; z.B.:

```
{ "access_token": "b31bc23d9e7702590f4a658eff5e27bb4a3f37b1",
  "expires_in": 3600,
  "token_type": "Bearer",
  "scope": "",
  "refresh_token": "f13e15027cc3b95f641df542c276967ec81ac6ba" }
```

Das Access-Token kann für die nächsten 3600 Sekunden für den Zugriff auf die SSO-API benutzt werden, danach kann wenn nötig mit dem Refresh-Token ein neues Access-Token angefordert werden.

Ein POST-Request auf die SSO-API sieht z.B. so aus:

https://<SSO_SERVER>/oauth2/ssoapi.php

POST-Parameter:

```
access_token=b31bc23d9e7702590f4a658eff5e27bb4a3f37b1&
method=getUserData&
appKey=<Client-Secret>
```

Mit dem Parameter `appKey` muss das Client-Secret übergeben werden. Die Antwort des SSO-Servers erfolgt wieder als JSON-String und könnte etwa so aussehen:

```
{"error":{
  "code":"700",
  "text":"OK"
},
"user":{
  "userId":"26244",
  "userGp":"",
  "userName":"Max",
  "userSurname":"Mustermann",
  "userLogin":"max.mustermann",
  "userEmail":"max.mustermann@sso-server.test",
  "userAlias":"",
  "userStatus":"1",
  "userComment":null,
  "isSubAccount":"0",
  "subAccountId":"0",
  "subAccountMainId":"0",
  "accountCreateOn":"2014-05-05 15:34:33",
  "accountActivateOn":"2014-05-05 15:35:56",
  "lastLogin":"2014-06-17 14:58:01",
  "userDataSalutation":"",
  "userDataTitle":null,
  "userDataStreet":"",
  "userDataStreetnumber":"",
  "userDataStreetaddon":null,
  "userDataZipcode":"",
  "userDataCity":"",
  "userDataCountry":"Deutschland",
  "userDataPhone":null,
  "userDataCell":null,
  "attribute":false}}
```

Welche Attribute der Client angezeigt bekommt, wird im SSO-Admin-Bereich konfiguriert. Hierüber kann auch ausgegeben werden, über welche Zeitungs- oder E-Paper-Abos der Kunde verfügt.

Die Antworten der SSO-API sind in Kapitel 6 beschrieben.

6.5.5 Beispiel

Das Zusammenspiel dieser Methoden kann man beispielhaft unter der folgenden URL beobachten:

https://<SSO-SERVER>/testscripts/testOAuth2.php

7. Frontend

Das MSP SSO stellt ein Frontend für den Benutzer zur Verwaltung seiner persönlichen Daten bereit. Dieses Frontend kann, muss aber nicht, verwendet werden. Die Nutzung einer eigenen Frontend-GUI ist möglich, der Abruf der Daten erfolgt dann über die MSP SSO-API. Bei der Installation des MSP SSO wird standardmäßig ein Test Benutzerkonto (siehe dazu Kapitel 8.1) eingerichtet. Die Zugangsdaten dafür lauten:

Login:	mustermann
Passwort:	mustermann

7.1 Unterkonten

Jeder Benutzer kann eine bestimmte Anzahl von Unterkonten[4] anlegen. Dabei werden sämtliche Attribute des Hauptkontos an das Unterkonto vererbt. Jedes Unterkonto kann jederzeit zu einem Hauptkonto hochgestuft werden und verliert dabei alle seine Attribute. Dadurch wird unberechtigter Zugriff auf Ressourcen oder Anwendungen nach einer Hochstufung verhindert.

7.2 Persönliche Daten

Persönliche Angaben wie Adresse, Telefonnummer usw. werden im Untermenüpunkt „Persönliche Daten“ bearbeitet.

7.3 Passwort ändern

Um das Passwort zu ändern ist die Eingabe des aktuellen Passwortes erforderlich.

7.4 E-Mail Adresse ändern

Zum Ändern der E-Mail-Adresse ist eine zweimalige Eingabe der neuen E-Mail-Adresse erforderlich. Daran anschließend wird eine E-Mail mit einem Aktivierungslink (dieser ist nur 60 Minuten gültig) an die aktuelle E-Mail-Adresse verschickt. Erst durch Klick des Aktivierungslinks erfolgt eine endgültige Aktualisierung.

8. Backend

Im MSP SSO Backend werden die Benutzerdaten, Benutzerkonten (sowohl Haupt- als auch Unterkonten), Applikationen, E-Mail-Texte und Attribute verwaltet. Bei der Installation des Backends wird standardmäßig ein Super-Administrator (siehe dazu Kapitel 8.1) angelegt. Die Zugangsdaten lauten:

[4] Die Anzahl der Unterkonten ist konfigurierbar.

Login: superadmin
Passwort: mspsso

Ändern Sie bitte nach erstmaliger Anmeldung umgehend das Passwort. Dieser Benutzer kann nicht über das Backend gelöscht werden. Sollte der Benutzer über die Datenbank gelöscht werden, ist kein Login am Backend mehr möglich (sofern keine weiteren Administratoren vorhanden sind). Eine Wiederherstellung ist dann nur direkt über die Datenbank möglich.

8.1 Administratorentypen

Der Zugriff auf die einzelnen Bereiche ist vom Administratortyp abhängig. Das MSP SSO Backend unterscheidet hier zwischen drei Typen, die mit unterschiedlichen Zugriffsrechten ausgestattet sind:

	Verwalten Benutzerkonten	Verwalten Applikationen	Verwalten Attribute	Verwalten E-Mail Texte
Mitarbeiter	x			
Administrator	x	x		
Super-Administrator	x	x	x	x

Das Verwalten beinhaltet immer jeweils die Aktionen Anlegen, Bearbeiten und Löschen.

8.2 Benutzerkonten

Ein Aufruf des Untermenüpunktes „Übersicht“ listet jeweils 20[5] Benutzerkonten auf einer Seite auf. Gesperrte Benutzerkonten werden in roter Schrift dargestellt, nicht aktivierte Benutzerkonten dagegen in blau. Alle anderen Benutzerkonten sind aktiv und gültig.

8.2.1 Benutzerkonten Anlegen, Bearbeiten und Löschen

Folgende Aktionen sind möglich:

	Aktiviert das Benutzerkonto.
	Zeigt alle Unterkonten eines Hauptkontos. Dieses Icon ist nur sichtbar wenn für ein Hauptkonto Unterkonten angelegt worden sind.
	Legt ein neues Unterkonto für dieses Hauptkonto an. Es kann für ein Unterkonto kein weiteres Unterkonto angelegt werden.
	Ein Unterkonto wird zu einem Hauptkonto hochgestuft. Dieses Icon ist nur bei Unterkonto sichtbar.
	Zeigt weitere Details (sofern verfügbar) wie Adressdaten oder Telefon/Mobilnummer an. Detailansichten sind nur für Hauptkonten möglich.
	Erzeugt ein neues Passwort für dieses Benutzerkonto. Das Passwort wird an die E-Mail-Adresse verschickt, die für dieses Benutzerkonto hinterlegt wurde.
	Das Formular für die Bearbeitung der Benutzerdaten wird aufgerufen.
	Das Formular für die Bearbeitung der Benutzerattribute wird aufgerufen. (siehe dazu Kapitel 8.2.2)
	Das Benutzerkonto wird gesperrt. Je nach Status des Kontos ist das Icon für sperren bzw. entsperren sichtbar. Beachten Sie, dass eine E-Mail-Adresse und ein Login nur einmal im MSP SSO verwendet werden können. Daher wird bei der Erfassung die Eingabe geprüft und

[5] Die Anzahl ist konfigurierbar.

	gegebenenfalls auf ein Vorhandensein hingewiesen. Wenn ein Hauptkonto gesperrt wird, so werden alle dazugehörigen Unterkonten ebenfalls gesperrt.
✔	Das Benutzerkonto wird entsperrt. Je nach Status des Kontos ist das Icon für entsperren bzw. sperren sichtbar. Wenn ein Hauptkonto entsperrt wird, werden alle dazugehörigen Unterkonto nicht automatisch entsperrt. Diese müssen separat entsperrt werden.
🗑	Das Benutzerkonto wird vollständig gelöscht. Wenn ein Hauptkonto gelöscht wird, werden die dazugehörigen Unterkonten sowie alle Attribute unwiderruflich gelöscht.
+	Legt ein neues Hauptkonto an.

8.2.2 Benutzerattribute

Um sich die Attribute (sofern vorhanden) eines Benutzerkontos anzusehen, wählen Sie zuvor die gewünschte Applikation aus. Danach können Sie die Attribute editieren und löschen. Über den Link „Attribut hinzufügen“ können Sie dem Benutzerkonto ein neues Attribut zuweisen. Füllen Sie das Formular aus und klicken Sie anschließend auf „Attribute anlegen“.

✎	Das Formular für die Bearbeitung des Attributs wird aufgerufen.
🗑	Löscht das Attribut. Beachten Sie, dass das Löschen eines Attributes ein Fehlerverhalten auf seitens der Applikationen verursachen kann. Möglicherweise wird das Attribut für einen Prozess verwendet der nun nicht mehr korrekt abgearbeitet werden kann. Eine Wiederherstellung des Attributs kann das Fehlerverhalten wieder korrigieren.

8.2.3 Benutzerkonten Suchen

In der Standardsuche wird nach dem Namen, dem Login sowie die E-Mail-Adresse des Benutzers mittels einer ODER-Verknüpfung gesucht. Hierbei wird nicht zwischen Groß- und Kleinschreibung unterschieden.

🔧	Um ein genaueres Suchergebnis zu erzielen, kann die Suche über eine Auswahl einzelner Parameter eingeschränkt werden. Durch das Anklicken des Schraubenschlüssel-Symbols neben dem Suchfeld werden weitere Suchparameter sichtbar. Wählen Sie die Parameter aus, auf die die Suche eingeschränkt werden soll.

8.3 Applikation

Ein Aufruf des Untermenüpunktes „Übersicht“ listet alle eingetragenen Applikationen auf. Gesperrte Applikationen werden in roter Schrift dargestellt. Bei der Installation des MSP SSO wird standardmäßig der SSO-Server als eigenständige Applikation angelegt. Dieser kann nicht gelöscht werden.

8.3.1 Applikation Anlegen, Bearbeiten und Löschen

Folgende Aktionen sind möglich:

↻	Erzeugt einen neuen ApplicationKey für eine Applikation, der an die E-Mail-Adresse verschickt wird, die für die Applikation hinterlegt wurde.
👁	Zeigt weitere Details (sofern verfügbar) wie Beschreibung und hinterlegte E-Mail-Adresse.
✎	Das Formular für die Bearbeitung der Applikationsdaten wird aufgerufen.
🗁	Das Formular für das Setzen der Zugriffsrechte der Benutzerdaten wird aufgerufen. Durch das Setzen der Haken wird bestimmt, welche Daten an die Applikation geliefert werden darf. Wenn weitere Inhalte übergeben werden müssen – die beispielsweise aus anderen Quellen stammen – muss das Feld „Erweitert“ angehakt sein. Zu beachten ist, dass bei der

	Datenübergabe (z. B. in einem Addon) der Schlüssel „userDataExtend“ dazu verwendet werden muss. Hier kann als Wert ein String oder auch ein Array übergeben werden. Beispiele: $result[‘userDataExtend‘] = value; $result[‘userDataExtend‘] = array(‘key‘ => ‘value‘);
	Das Formular für die Bearbeitung der Attribute wird aufgerufen.
	Die Applikation wird gesperrt. Je nach Status der Applikation ist das Icon für sperren bzw. entsperren sichtbar.
	Die Applikation wird entsperrt. Je nach Status der Applikation ist das Icon für entsperren bzw. sperren sichtbar.
	Die Applikation wird vollständig gelöscht.
	Fügt eine neue Applikation hinzu.

8.3.2 Attribute

Um die Attribute einer Applikation aufzulisten, wählen Sie den Untermenüpunkt „Übersicht“ und dann die gewünschte Applikation aus.

	Das Formular für die Bearbeitung der Attribute wird aufgerufen.
	Das Attribut wird gelöscht. Beachten Sie, dass durch das Löschen eines Attributes alle dazugehörigen Attribut-Werte aller Benutzerkonten ebenfalls gelöscht werden. Das kann zur Folge haben, dass die Applikation möglicherweise die betroffenen Benutzer nicht mehr korrekt zuordnen / ver- bzw. bearbeiten kann.

8.3.3 E-Mail Texte

Um die E-Mail Texte einer Applikation aufzulisten, wählen Sie den Untermenüpunkt „Übersicht“ und dann die gewünschte Applikation aus.

	Der kompletten E-Mail Text anzeigen.
	Das Formular für die Bearbeitung der Attribute wird aufgerufen.
	Das Attribut wird gelöscht. Beachten Sie, dass durch das Löschen eines Attributes alle dazugehörigen Attribut-Werte aller Benutzerkonten ebenfalls gelöscht werden. Das kann zur Folge, dass die Applikation möglicherweise die betroffenen Benutzer nicht mehr korrekt zuordnen / ver- bzw. bearbeiten kann.

8.4 Administration

Ein Aufruf des Untermenüpunktes „Übersicht“ listet alle Administratoren auf. Klicken Sie auf das Plus-Symbol um einen neuen Administrator anzulegen. Nur die Administratorentypen „Administrator“ und „Super-Administrator“ haben Zugriff auf diesen Bereich. Demzufolge können auch nur diese Administratorentypen neue Mitarbeiter und Administratoren anlegen, bearbeiten und löschen.

	Das Formular für die Bearbeitung des Administrators wird aufgerufen.

	Erzeugt ein neues Passwort, das an die E-Mail-Adresse verschickt wird, die für diesen Administrator hinterlegt wurde.
	Der Administrator wird gesperrt. Je nach Status das Icon für sperren bzw. entsperren sichtbar.
	Der Administrator wird entsperrt. Je nach Status das Icon für sperren bzw. entsperren sichtbar.
	Der Administrator wird gelöscht.

8.4.1 Addons

Dieser Menüpunkt steht nur zur Auswahl wenn etwaige Addons in der Konfiguration aktiviert worden sind. Derzeit steht aber nur das MSP SAP Addon zur Auswahl. Anhand der ausgewählten Felder (wenn Haken gesetzt) erfolgt ein Geschäftspartner Abgleich mit Ihrem SAP-System.

8.5 Einstellungen

Dieser Bereich steht jedem Mitarbeiter und Administrator zur Verfügung und dient zum Editieren ihrer persönlichen Daten und Passwort.

9. FAQ

1. *Der Benutzer kann sich mit einer seiner E-Mail-Adresse nicht registrieren.*
   Jede E-Mail Adresse kann nur einmal verwendet werden. Ist der Kunde oder ein Familienmitglied bereits mit dieser E-Mail Adresse registriert? Verwendet der Benutzer eine sogenannte Junk-E-Mail-Adresse (z. B. Mailinator oder Trash-Mail), so werden diese Domains geblockt.

2. *Gibt es eine Liste welche Junk-E-Mail-Adressenanbieter geblockt werden?*
   Eine Liste findet sich unter class/helper/invalidDomains.txt

3. *Der Rücksprung auf die Adresse aus der service-URL funktioniert nicht.*
   Ein Präfix der Rücksprung-Adresse muss im Backend bei der Applikation konfiguriert sein.

Anhang

Beispiel für die Kommunikation beim Login-Prozess über die JSONP-Schnittstelle zwischen der Client-Applikation http://www.eins.test/client/test/testapp1.php und dem SSO-Server http://login.nwzonline.test/

1. Nachfragen, ob der User vielleicht schon am SSO-Server eingeloggt ist:

```
$.ajax({
    "url": "http://login.ssoserver.test/json/authenticate.php?callback=?",
    "data": {
        "action": "validate"
    },
    "dataType": "jsonp"
});
```

Über die JSONP-Schnittstelle wird angefragt, ob der User am SSO-Server eingeloggt ist; in diesem Fall existiert kein Login-Cookie, welches der SSO-Server überprüfen kann, daher die Antwort:

Antwort des SSO-Servers:

```
{
    "error": {
        "code": "505",
        "text": "Token kann nicht ermittelt werden."
    }
}
```

2. Login am SSO-Server über JSONP:

```
$.ajax({
    "url": "http://login.ssoserver.test/json/authenticate.php?callback=?",
    "data": {
        "action": "authenticate",
        "userLogin": "joe@email.com",
        "userPass": "Me1nPassw0rt"
    },
    "dataType": "jsonp"
});
```

Aus einem Formular mit Username/Passwort werden mittels JSONP der Username und das Passwort mit der „action“ „authenticate“ an die JSONP-Schnittstelle des SSO-Servers übertragen; der SSO-Server überprüft die Login/Passwort-Kombination und loggt den User dann ein; die Userdaten werden im „user“-Container zurückgeliefert. Der Login-Token (user.tokenId) wird anschließend an die Applikation weitergegeben – mit diesem können User-spezifische Anfragen über die API gestellt werden.

Antwort des SSO-Servers:

```
{
    "error": {
        "code": "700",
        "text": "OK"
    },
    "user": {
        "userId": "46130",
        "userGp": "40623064",
        "userName": "Joe",
        "userSurname": "User",
        "userLogin": "joe@email.com",
        "userEmail": "joe@email.com",
        "userAlias": null,
        "userStatus": "1",
        "userComment": null,
        "isSubAccount": "0",
        "subAccountId": "0",
        "subAccountMainId": "0",
        "accountCreateOn": "2009-12-08 12:42:02",
        "accountActivateOn": "2011-06-30 16:57:01",
        "lastLogin": "2015-12-09 15:55:01",
        "userFlag": "bearbeitet",
        "tokenId": "00046130000027359510144967290 1"
    }
}
```

3. Weiterreichen des Login-Tokens an die Applikation:

```
$.ajax({
    "url": "/client/test/testapp1.php?callback=?",
    "data": {
        "mspsso_action": "validate",
        "mspsso_token": "00046130000027359510144967290 1"
    },
    "dataType": "jsonp"
});
```

4. Antwort eines späteren Validate-Requests (siehe 1.):

```
$.ajax({
    "error": {
        "code": "700",
        "text": "OK"
    },
    "token": {
```

```
        "tokenId": "00046130000027359510144967290 1"
    }
});
```

Ein Validate-Request liefert, wenn der User eingeloggt ist, den gültigen Login-Token zurück.