

Einstellung	Beschreibung
Reagieren bei	<p>Diese Einstellung bestimmt, zu welchen Ereignissen tenfold eine Aktualisierung der Postfächer vornimmt. Folgende Einstellungen sind möglich:</p> <ul style="list-style-type: none"> <li>• <b>Niemals:</b> Postfächer von Personen mit zugewiesener Ressource werden niemals aktualisiert.</li> <li>• <b>Bei angeforderter Adressänderung:</b> Postfächer von Personen mit zugewiesener Ressource werden aktualisiert, wenn sich die E-Mail-Adresse der Person ändert.</li> <li>• <b>Bei angeforderter Namensänderung:</b> Postfächer von Personen mit zugewiesener Ressource werden aktualisiert, wenn sich Vor- und/oder Nachname der Person ändern. <b>Achtung:</b> Diese Einstellung hat nur eine Wirkung, wenn die E-Mail-Adressen von tenfold erzeugt werden.</li> <li>• <b>Bei allen Stammdatenänderungen:</b> Postfächer von Personen mit zugewiesener Ressource werden bei allen Stammdatenänderungen der Personen aktualisiert.</li> </ul>
E-Mail-Alias	<p>Diese Einstellung legt fest, wie mit der aktuellen E-Mail-Adresse verfahren werden soll, wenn seitens tenfold die Adresse geändert wird. Sie haben folgende Auswahlmöglichkeiten:</p> <ul style="list-style-type: none"> <li>• <b>Alte E-Mail Adresse als Alias behalten:</b> Die aktuelle E-Mail-Adresse wird bei einer Änderung als Alias-Adresse behalten.</li> <li>• <b>Alte E-Mail Adresse entfernen:</b> Die aktuelle Adresse wird entfernt. Das Postfach ist ab der Änderung nur noch unter der neuen Adresse erreichbar.</li> </ul>

### Karteireiter "Mailbox löschen"

In diesem Karteireiter legen Sie fest, wie mit Postfächern umgegangen wird, wenn Personen in tenfold die Postfach-Ressourcen entfernt werden.

Einstellung	Beschreibung
Aktion	<p>Legt die Aktion fest, welche bei Entfernung der Ressource durchgeführt wird. Sie haben folgende Auswahlmöglichkeiten:</p> <ul style="list-style-type: none"> <li>• <b>Keine:</b> Es wird keine Aktion durchgeführt. Das Postfach bleibt normal bestehen.</li> <li>• <b>Exchange-Mailbox deaktivieren:</b> Bei Entfernung der Ressource wird das zugehörige Postfach deaktiviert.</li> </ul>

**Postfächer deaktivieren**

Beim Deaktivieren eines Postfaches wird das Postfach vom Active Directory-Konto getrennt. Das Postfach existiert noch eine gewisse Zeit lang in Exchange (Dauer kann in Exchange konfiguriert werden), wird daraufhin jedoch gelöscht. Wird für dieses Konto zu einem späteren Zeitpunkt ein Postfach aktiviert, so wird dabei ein neues Postfach angelegt. Solange die Daten des alten Postfaches noch in der Datenbank existieren, können diese in eine neue Mailbox übernommen werden. Danach sind die Daten verloren. Nähere Information dafür finden Sie unter <https://learn.microsoft.com/en-us/exchange/recipients/disconnected-mailboxes/connect-disabled-mailboxes?view=exchserver-2019>. **Beachten Sie, dass eine automatische Übernahme von Altdaten in ein neues Postfach von tenfold nicht unterstützt wird.** Dies muss über die Werkzeuge von Exchange manuell durchgeführt werden.

Wenn Sie mit den getroffenen Einstellungen zufrieden sind, betätigen Sie die Schaltfläche "Speichern", um die Konfiguration für diesen Exchange-Server abzuschließen. Sollte für diese Domäne & Bereitstellungsart noch keine Postfachressource existieren, wird diese an dieser Stelle angelegt.

#### Mehrfachkonfiguration

Sollten Sie versuchen, für eine Domäne & Bereitstellungsart eine weitere Konfiguration anzulegen, so wird Ihnen beim Speichern eine Fehlermeldung angezeigt und die Konfiguration kann nicht gespeichert werden.

Sie können, auf der Hauptmaske des Plugins, auch bestehende Konfigurationen über die Aktion "Bearbeiten" im Aktionsmenü der entsprechenden Zeile ändern. Bitte beachten Sie, dass eine Änderung der Einstellungen keine unmittelbaren Auswirkungen auf existierende Postfächer hat. Nur für Neuanlagen und gewisse Änderungen in der Zukunft haben die Änderungen eine Konsequenz.

Mit der Aktion "Löschen" können Sie eine Konfiguration (inklusive der zugehörigen Ressource und Zuordnungen) wieder entfernen. Beim Löschen der Zuordnungen werden keine Postfächer in Exchange deaktiviert, lediglich in tenfold werden die Zuordnungen entfernt.

### 14.4.3 Abgleich und Simulation

Da während der Verwendung von tenfold auch weiterhin die Möglichkeit besteht, Postfächer direkt mit den Werkzeugen von Exchange zu verwalten, kann nicht ausgeschlossen werden, dass es zu Differenzen im Datenbestand von tenfold und dem eigentlichen Zustand von Exchange kommt. Dies kann zu Fehlern führen, wenn tenfold davon ausgeht, dass Postfächer existieren, welche bereits entfernt wurden oder wenn tenfold Postfächer anlegen möchte, welche bereits existieren.

Um solche Fehler zu vermeiden, macht tenfold in regelmäßigen Abständen einen Datenabgleich, welcher vom Job "Exchange Mailbox Lifecycle Plugin - Sync" durchgeführt wird (siehe [Jobs](#) (siehe page 443)). Bei diesem Abgleich entfernt tenfold Zuordnungen von Postfachressourcen, deren Postfächer nicht mehr existieren, oder erstellt Zuordnungen von Postfachressourcen, wo für die Person das Vorhandensein eines Postfaches festgestellt werden konnte.

#### Aktionen in Exchange

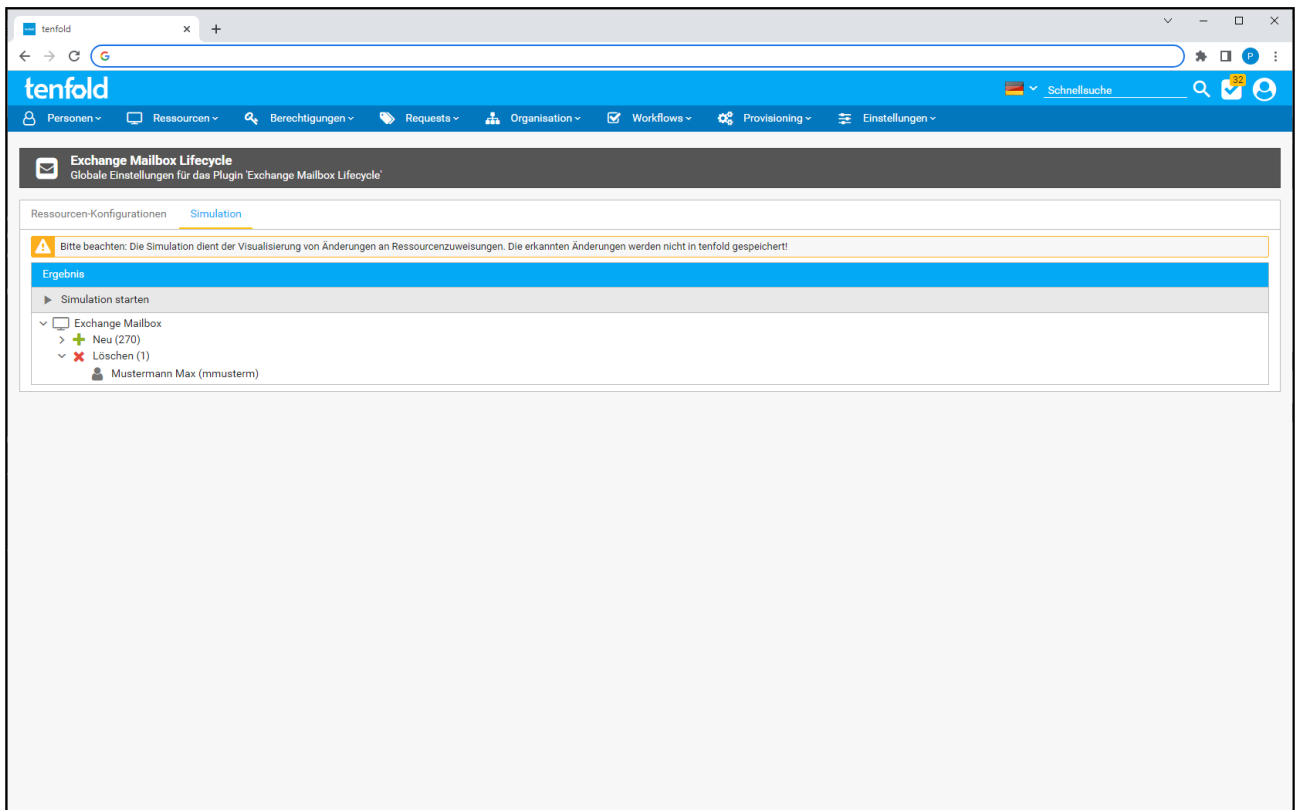
Dieser Abgleich führt **keine** Aktionen in Exchange durch. Es wird nur der Ist-Zustand des Datenbestandes in tenfold zum Soll-Zustand, welcher aus Exchange ermittelt wurde, angeglichen. Postfächer in Exchange werden durch den Abgleich weder angelegt, deaktiviert noch aktualisiert.

Wenn Sie prüfen möchten, welche Änderungen der nächste Datenabgleich in tenfold umsetzen würde, können Sie eine Simulation des Abgleichs durchführen. Hierfür steht Ihnen der Karteireiter "Simulation" auf

der Hauptmaske des Plugins zur Verfügung (im Menü unter *Provisioning > Plugins > Exchange Mailbox Lifecycle*).

### Ressourcen

Bevor Sie eine Simulation durchführen können, müssen Sie zunächst eine Ressource mittels der Ressourcenkonfigurationen (siehe [Allgemeine Einstellungen](#)(see page 693)) angelegt haben.



Nachdem Sie die Schaltfläche "Simulation starten" betätigt haben, erhalten Sie eine baumartige Darstellung aller Änderungen, welche der kommende Abgleich durchführen wird. Sollten Sie der Meinung sein, dass mit den Änderungen etwas nicht stimmt, deaktivieren Sie den Job "Exchange Mailbox Lifecycle Plugin - Sync" und prüfen die Konfiguration.

### Änderungen speichern

Die angezeigten Änderungen können an dieser Stelle nicht durchgeführt oder gespeichert werden. Wenn Sie die Änderungen durchführen wollen, starten Sie den Job "Exchange Mailbox Lifecycle Plugin - Sync" manuell in der Job-Verwaltung (siehe [Jobs](#)(see page 443)).

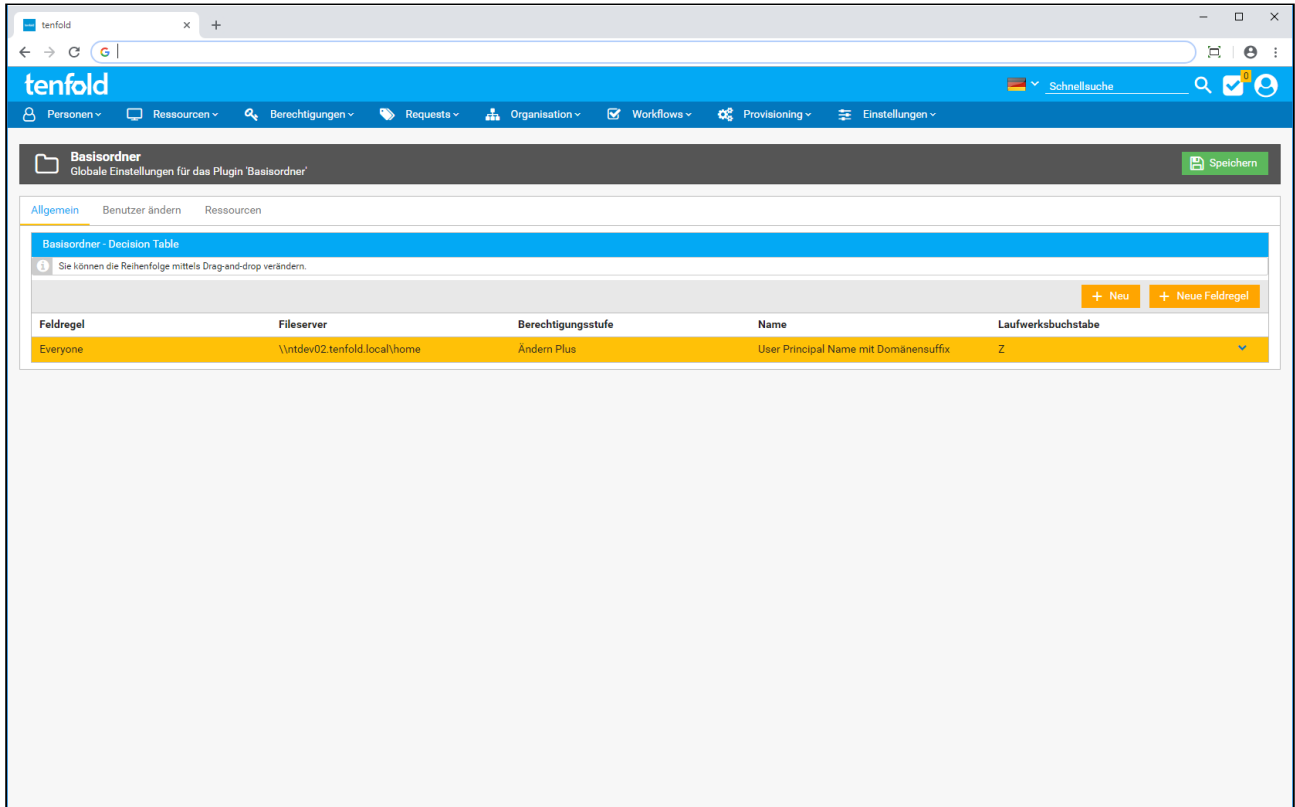
## 14.5 Basisordner

Die Aufgabe des Basisordner Plugin ist die Verwaltung des Basisordner für einen Active Directory Benutzer ("Home-Verzeichnis"). Das Plugin stellt folgende Funktionen zur Verfügung:

- Anlegen eines Basisordners (dies geschieht üblicherweise bei der Anlage eines Benutzers)
- Umbenennen bzw. Verschieben des Ordners bei relevanten Änderungen

Die Zuordnung eines Basisordners ist dabei durch eine Ressourcenzuordnung bei einer Person abgebildet. Somit hat eine Person, die eine Zuordnung der Ressource hat, einen Basisordner. Fehlt die Zuordnung, so existiert auch kein Basisordner.

### 14.5.1 Konfigurationsübersicht



Die Einstellungen für das Plugin sind auf folgenden Karteireitern verteilt:

- Allgemein / Decision Table: Hier werden die Einstellungen für die zu erstellenden Basisordner hinterlegt
- Benutzer ändern: Auf diesem Reiter befinden sich Einstellungen, die festlegen, wie das Plugin bei Änderungen an der Person reagieren soll
- Ressourcen: Auf diesem Tab können automatisch Ressourcen für das Plugin erzeugt werden

### 14.5.2 Allgemein

Auf diesem Karteireiter werden in der Entscheidungstabelle einer oder mehrere Einträge angelegt, die steuern, wie die Verzeichnisanlage funktioniert. Wie bei jeder Entscheidungstabelle, prüft das System die Einträge von oben nach unten, und wählt den ersten Eintrag aus, der zutreffend ist. Ob ein Eintrag auf die jeweilige Person anwendbar ist, wird anhand einer Feldregel überprüft (siehe dazu auch [Feldregeln](#) (see page 562)). Die betreffende Feldregel wird in der Entscheidungstabelle in der ersten Spalte angezeigt.

Für jeden Eintrag kann anschließend folgendes eingestellt werden:

Einstellung	Beschreibung
Fileserver	Legt den Fileserverpfad fest, in welchem der Basisordner angelegt wird. Der Pfad muss als UNC-Pfad eingegeben werden. Achtung: Das Dienstkonto, unter welchem der unter "Agent" ausgewählte Agent läuft, muss über Vollzugriff auf diesem Pfad verfügen.
Berechtigungsstufe	Legt die Berechtigungsstufe fest, welche dem Benutzer auf dem Basisordner gewährt wird. Es wird empfohlen, die Einstellung "Ändern" oder "Ändern Plus" zu wählen.
Name generieren aus	Bestimmt, welchen Namen das Verzeichnis erhalten soll. Zur Auswahl stehen der Benutzername (samAccountName) oder der UPN, jeweils mit oder ohne Domain-Suffix.
Agent	Legt fest, mit welchem Agent die Verzeichnisanlage und Berechtigungszuweisung erfolgen soll.
AD-Attribute setzen	Bestimmt, ob im Active Directory das angelegte Verzeichnis als Basisordner eingetragen werden soll.
Laufwerksbuchstabe	Wenn die Option "AD-Attribute setzen" gesetzt ist, lässt sich hiermit bestimmen, mit welchem Laufwerksbuchstaben der Basisordner bei der Anmeldung verbunden werden soll.
Wenn Verzeichnis existiert	Diese Einstellung dient dazu, festzulegen, was passieren soll, wenn ein Verzeichnis mit dem gewünschten Namen bereits existiert.

### 14.5.3 Benutzer ändern

Auf diesem Karteireiter kann bestimmt werden, welche Aktion ausgelöst werden soll, wenn eine Änderung an den Stammdaten einer Person dazu führt, dass eine andere Feldregel (siehe Karteireiter "Allgemeint") greift, als die bisher gültige und der Basisordner sich auf einem anderen Fileserver befinden sollte. Folgende Einstellungen sind dabei möglich:

- Niemals verschieben: Selbst, wenn ein anderer Fileserver vorgesehen wäre, wird der Basisordner auf dem alten Fileserver belassen
- bei (...): der Basisordner wird nur verschoben, wenn das jeweils ausgewählte Personenfeld verändert wurde
- bei allen Stammdatenänderungen: der Basisordner wird verschoben, egal welches Feld bearbeitet wurde (allerdings nur, wenn tatsächlich ein anderer Fileserver vorgesehen ist)

### 14.5.4 Ressourcen

Dieser Karteireiter dient dazu, neue Ressourcen für dieses Plugin anzulegen. Eine Ressource entspricht dabei einem Basisordner in einer bestimmten Domäne. Wird diese Ressource einer Person zugewiesen, so wird ein neuer Basisordner erstellt.

Folgende Optionen können zum Anlegen gewählt werden:

- Domäne: Legt fest, für welche Domäne eine neue Ressource erstellt werden soll
- Name: Legt den Namen für die neue Ressource fest
- Ressourcenkategorie: Bestimmt, in welcher Kategorie die neue Ressource angelegt werden soll

Durch Klicken des "Neu" Button wird die Ressource angelegt. Es werden dabei bereits alle Einstellungen, insbesondere die Konfiguration der Provisionierung, so gesetzt, dass die Ressource sofort einsatzfähig ist. Es wird empfohlen, die Einstellungen nicht zu verändern, um die Funktionsfähigkeit zu gewährleisten. Ausgenommen davon sind die Optionen für das überschreiben der globalen Pluginkonfiguration im Karteireiter "Provisioning" sowie Beschreibungstexte und Bilder.

### Information

Ressourcen, die durch das Plugin erstellt wurden, können auf dieser Maske nicht bearbeitet oder gelöscht werden. Nutzen Sie dazu bitte die Ressourcenverwaltung. Siehe auch [Ressourcenverwaltung](#)<sup>11</sup>.

## 14.6 E-Mail-Benachrichtigung

Das Plugin E-Mail-Benachrichtigung ist für das Versenden von E-Mails in unterschiedlichen Situationen zuständig. Es kann dazu genutzt werden, bestimmte vordefinierte Standard-E-Mails zu aktivieren und um neue E-Mail-Benachrichtigungen zu erstellen, die im Standardumfang nicht vorgesehen sind.

### 14.6.1 Konfigurationsübersicht

Die Konfiguration findet über mehrere Karteireiter statt, welche nachfolgende Aufgaben haben:

- Allgemein: Hier können systemweite Einstellungen festgelegt werden.

<sup>11</sup> <http://vi-srv-atlassian.prod.local:8090/display/ISMAD/Ressourcenverwaltung>

- Personendaten bis Dateneigentümer: Dient zur Aktivierung von Standardbenachrichtigungen
- Globale Ereignisse: An diesem Punkt können benutzerdefinierte Benachrichtigungen zu bestimmten, vorgegebenen Systemereignissen gesendet werden

## 14.6.2 Allgemein

### Absender

- Absender: Entweder kann die Systemeinstellung (E-Mails > Absender-Adresse) übernommen werden, oder es kann für das Plugin ein individueller Absender gewählt werden.
- Absender-Adresse: Es wird entweder die Systemeinstellung angezeigt oder es kann die individuelle Adresse eingegeben werden.

#### Achtung

Der Mailserver muss den konfigurierten Absender als gültigen Absender akzeptieren. Sollte es hierbei zu Problemen kommen, kontaktieren Sie den Administrator des Mailservers.

### Betreff überschreiben

Die Option "Betreff überschreiben" steht nur bei der Verwendung von Standardvorlagen zur Verfügung. Sie ermöglicht es, den Betreff der in der Vorlage definiert wurde, an dieser konkreten Stelle mit einem individuellen Text zu überschreiben. Der überschriebene Betreff gilt dann nur für die jeweilige Nachricht, bei welcher er hinterlegt wurde. Alle anderen E-Mails, bei denen die gleiche Standardvorlage verwendet wird, funktionieren nach wie vor mit dem Standardtext. Zusätzlich zur Möglichkeit, einen fixen Text einzugeben, kann über den Parameter `request` und dessen Felder auch auf alle Strukturen des jeweiligen Request verwiesen werden. Die entsprechenden Stellen werden dann mittels Ersetzung mit den Daten aus dem Request ausgetauscht. Um beispielsweise den Nachnamen des Empfängers eines Request im Betreff einzusetzen, kann der Betreff der Standardvorlage mit folgendem Wert überschrieben werden:

"Neuer Request für `${request.person.masterdata.lastName}`"

#### Achtung

Mehrsprachige Vorlagen werden von dieser Funktion nicht berücksichtigt. Wird der Betreff überschrieben, so wird dieser für E-Mails in allen Sprachen verwendet.

### Benachrichtigung bei fehlgeschlagenem Request

Diese Einstellung dient dazu, automatisch eine Benachrichtigung an den konfigurierten Empfänger zu senden, wenn ein Request fehlgeschlagen ist.

- Aktiv: Die Checkbox muss aktiviert werden, damit die Einstellungen sichtbar werden.
- Vorlageneinstellung: Es kann entweder die Standardvorlage verwendet werden, oder es kann eine individuelle Vorlage ausgewählt werden.
- Empfänger festlegen: Der Empfänger kann entweder manuell eingegeben werden (Fixer Wert), per Script festgelegt werden (Code Snippet) oder der Dateneigentümer der betroffenen Ressource kann informiert werden.
- Request-Typ und Request-Quelle: Die Benachrichtigung kann auf bestimmte Request-Typen und bestimmte Request-Quellen eingeschränkt werden. Für andere Typen und Quellen wird keine Benachrichtigung generiert.

## Benachrichtigung bei fehlgeschlagenem Job

Diese Einstellung dient dazu, automatisch eine Benachrichtigung an den konfigurierten Empfänger zu senden, wenn ein Job fehlgeschlagen ist (siehe zu Jobs auch [Jobs](#) (see page 443)).

- Aktiv: Die Checkbox muss aktiviert werden, damit die Einstellungen sichtbar werden.
- Vorlageneinstellung: Es kann entweder die Standardvorlage verwendet werden, oder es kann eine individuelle Vorlage ausgewählt werden.
- Empfänger festlegen: Der Empfänger kann entweder manuell eingegeben werden (Fixer Wert), per Script festgelegt werden (Code Snippet) oder der Dateneigentümer der betroffenen Ressource kann informiert werden.

### 14.6.3 Personendaten (und folgende Karteireiter)

The screenshot shows the 'E-Mail-Benachrichtigung' (Email Notification) settings for the 'Personendaten' (Person Data) tab. The interface is divided into several sections, each with an 'Aktiv' (Active) checkbox and specific configuration options.

- Genehmiger bei offener Genehmigung benachrichtigen** (Notifier when approval is pending):
  - Aktiv: ☒
  - Vorlageneinstellung: Standardvorlage verwenden
  - Link zur Requestprüfung einfügen: ☐
  - Link zur Requestanzeige einfügen: ☐
  - Link zur direkten Requestgenehmigung einfügen: ☐
  - Link zur direkten Requestablehnung einfügen: ☐
  - Request-Typ: Alle
  - Request-Quelle: Alle
- Ersteller bei Abbruch benachrichtigen** (Creator when cancelled):
  - Aktiv: ☒
  - Vorlageneinstellung: Standardvorlage verwenden
  - Request-Typ: Alle
  - Request-Quelle: Alle
- Ersteller bei Abschluss benachrichtigen** (Creator when completed):
  - Aktiv: ☐
- Betroffene Person bei Abschluss benachrichtigen** (Affected person when completed):
  - Aktiv: ☐

Auf dem Karteireiter Personendaten und den nachfolgenden Karteireitern können jeweils Benachrichtigungen aktiviert werden, die in bestimmten Situationen für den jeweiligen Request-Modus (Active Directory Gruppe, Fileserver, Ressource, etc.) ausgelöst werden. Der jeweilige Bereichstitel beschreibt das Ereignis und kann über die Checkbox "Aktiv" aktiviert werden. Anschließend sind jeweils spezifische Einstellungen verfügbar, mit denen die Benachrichtigung detailliert konfiguriert werden kann. Folgende Einstellungen finden sich darunter:

- Vorlageneinstellung: Mit dieser Einstellung kann jeweils festgelegt werden, ob die Standardvorlage für das jeweilige Ereignis verwendet werden soll, oder ob eine benutzerdefinierte Vorlage verwendet werden soll.
- Vorlage: Wird eine benutzerdefinierte Vorlage verwendet, so kann diese hier ausgewählt werden.
- Request-Typ: Einschränkung der Benachrichtigung auf einen bestimmten Request-Typ



- Request-Quelle: Einschränkung der Benachrichtigung auf eine bestimmte Request-Quelle
- Link zu (...) einfügen: Wird die jeweilige Checkbox aktiviert, so wird in der Vorlage automatisch ein Link zum Auslösen der jeweiligen Aktion hinterlegt. Diese Funktion ist nur verfügbar, wenn die Standardvorlage verwendet wird.
- Ressourcenabhängigkeit: Einschränkung der Benachrichtigung auf eine bestimmte Ressource

## 14.6.4 Globale Ereignisse

Auf dem Karteireiter Globale Ereignisse können individuelle Benachrichtigungen verschickt werden. Die Benachrichtigung muss sich dabei immer auf ein vordefiniertes Ereignis (siehe unten) beziehen, zu welchem die Benachrichtigung ausgelöst werden soll.

Mit dem Button "Hinzufügen" kann ein neuer Eintrag erstellt werden. Für jeden Eintrag ist folgendes festzulegen:

- Ereignis: Es muss das Ereignis selektiert werden, zu welchem die Benachrichtigung ausgelöst wird. Die Liste der Ereignisse ist systemdefiniert und kann nicht verändert oder erweitert werden.
- Vorlage: Die Vorlage, welche die Basis für die erstellte E-Mail bildet, muss ausgewählt werden.
- Bedingung: Hier kann ein Code Snippet hinterlegt werden, welches bestimmt, ob die Benachrichtigung tatsächlich ausgelöst wird oder nicht. Das Snippet muss einen Boolean (true/false) zurückliefern.
- Empfänger festlegen: Es kann zwischen einem fixen Empfänger, einem Code Snippet oder einem Dateneigentümer gewählt werden.
- Empfänger: Der Empfänger wird in diesem Textfeld eingegeben. Mehrere Empfänger können, durch ein Komma oder einen Zeilenumbruch getrennt, angegeben werden.

- **Code Snippet:** Ist für die Festlegung des Empfängers die Einstellung "Code Snippet" gewählt, muss hier das Snippet eingegeben werden. Das Snippet muss eine gültige E-Mail-Adresse (oder mehrere nach obenstehender Regel) zurückliefern.

Im Bereich "Parameter" können Parameter definiert werden, die an die E-Mail-Vorlage übergeben werden. Diese Funktion dient dazu, eine Vorlage für mehrere Ereignisse mit bestimmten variablen Inhalten verwenden zu können. Ein Parameter wird mit dem Button "Hinzufügen" erstellt. Es müssen dabei folgende Einstellungen festgelegt werden:

- **Parameter:** Der Name des Platzhalters in der E-Mail-Vorlage
- **Wert:** Der Wert, mit welchem der Platzhalter ersetzt werden soll

## 14.7 Import Plugin

Das Import Plugin wird dazu genutzt, Personendaten aus vorgelagerten Drittsystemen nach tenfold zu importieren. Es wird dazu genutzt, um neue Personen anzulegen, Stammdatenänderungen an Personen durchzuführen und nicht mehr existierende Personen zu löschen. Beispielsweise wird das Plugin genutzt, um von einem Personalmanagementsystem (HR-System) bestimmte Daten automatisch zu übernehmen, wenn darin ein neuer Mitarbeiter angelegt oder ein bestehender geändert wurde. Es kann aber beispielsweise auch dafür genutzt werden, um automatisch Rufnummern aus einer Telefonanlage zu übernehmen. Das Plugin hat für den gewöhnlich Anwender keine Berührungspunkte, da es vollständig im Hintergrund agiert. Es besteht aus einer oder mehreren Import-Konfigurationen sowie einem Job (siehe [Jobs\(see page 443\)](#)), der die Abgleiche durchführt.

### Allgemeiner Warnhinweis

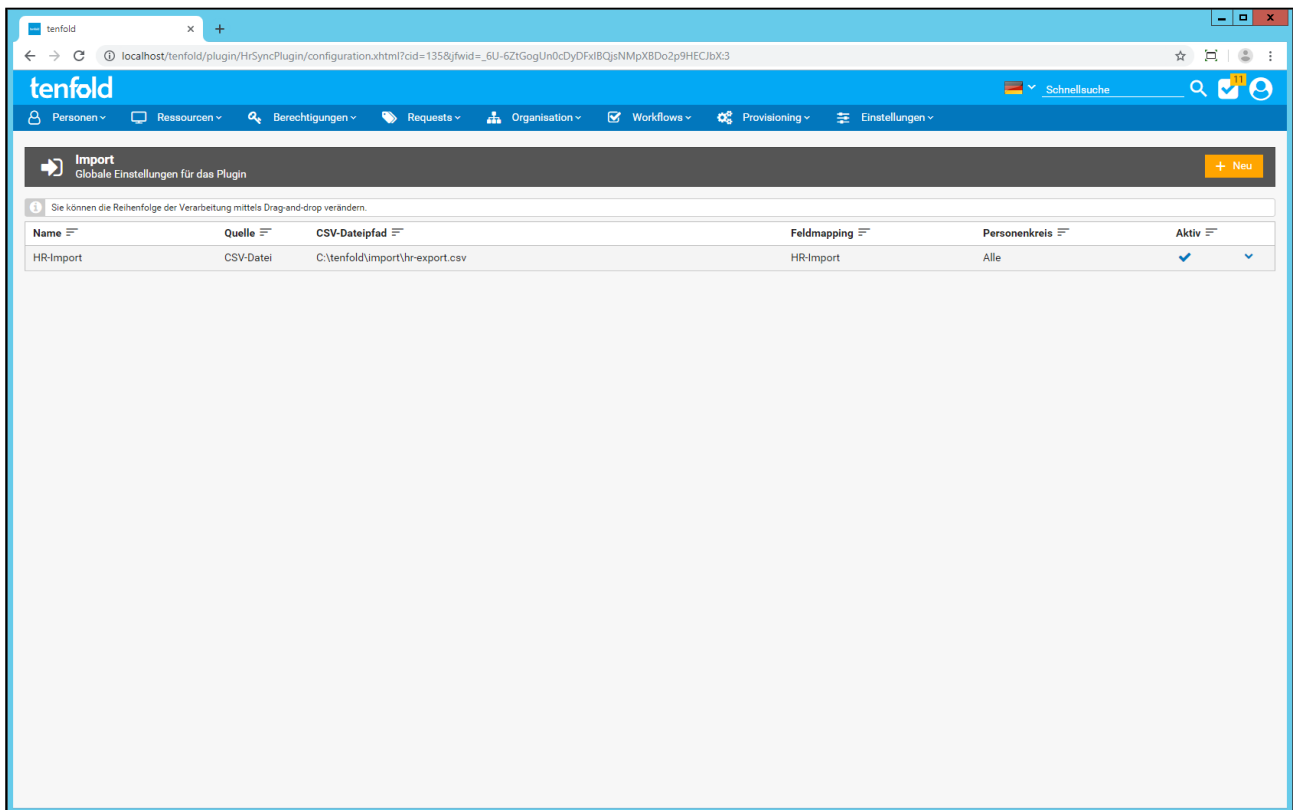
Das Import Plugin ist in seiner Konfiguration vergleichsweise komplex und kann bei falschen Einstellungen darüber hinaus enormen Schaden, bis zur Löschung aller Benutzeraccounts, zur Folge haben. Gehen Sie bei der Konfiguration extrem sorgfältig vor und nutzen Sie in jedem Fall bei jeder noch so unbedeutenden Änderung den Simulationsmodus, um die resultierenden Auswirkungen zu testen.

### 14.7.1 Konfigurationsübersicht

Für das Plugin können mehrere Konfigurationen angelegt werden. Jede Konfiguration stellt dabei eine Import-Quelle dar. Eine neue Konfiguration kann mit dem Neu-Button erstellt werden.

### Achtung

Bevor eine neue Konfiguration angelegt wird, sollte bereits das Feldmapping angelegt sein, da dieses zum Speichern benötigt wird (siehe mehr dazu unter "Feldmapping").



Auf der Übersichtsmaske der Konfigurationen kann via Drag-and-Drop die Reihenfolge der Konfigurationen verändert werden. Die Imports laufen anschließend in der festgelegten Reihenfolge ab. Der Startzeitpunkt wird über die Job-Verwaltung festgelegt. Es ist aktuell nicht möglich, für jede Konfiguration einen individuellen Startzeitpunkt festzulegen. Die Ausführung startet zur im Job definierten Zeit und führt anschließend die Import in der festgelegten Reihenfolge durch.

#### Import vorübergehend deaktivieren

Wenn ein Import vorübergehend nicht ausgeführt werden soll, so kann dieser über das Kontextmenü deaktiviert werden. Wenn der Import anschließend wieder ausgeführt werden soll, kann er wieder aktiviert werden.

## 14.7.2 Allgemein

The screenshot shows the 'Import' configuration page in the tenfold web interface. The 'Allgemein' tab is selected. The 'Einstellungen' section contains the following fields:

- Name:** HR-Import
- Beschreibung:** (empty text area)
- Quelle:** CSV-Datei (selected from a dropdown)
- CSV-Dateipfad:** C:\tenfold\import\hr-export.csv
- Trennzeichen:** ;
- Zeichenkodierung:** UTF\_8
- Kopfzeile:** ☒

Auf dem Karteireiter "Allgemein" werden der Name und eine optionale Beschreibung für den Import festgelegt. Der Bereich "Quelle" ist variabel und hängt von der Einstellung im Feld "Quelle" ab:

### Quelle "CSV-Datei"

Wenn die Quelle CSV-Datei gewählt wurde, dann liest tenfold die zu importierenden Daten aus einer CSV (Comma Separated Value) Datei aus. Dabei müssen folgende Einstellungen getroffen werden:

Einstellung	Beschreibung	Beispiel
CSV-Dateipfad	Gibt an, wo die zu importierende Datei gefunden werden kann. Es kann sowohl ein lokaler, als auch ein UNC-Netzwerkpfad angegeben werden.	C:\tenfold\import\hr-import.csv \\fs-srv-4711\exports\hr-import.csv
Trennzeichen	Legt fest, welches Zeichen zur Trennung der Felder in der Datei verwendet wird. Dieses Zeichen darf nicht in den Feldinhalten selbst vorkommen, da dies dazu führt, dass die Datei nicht eingelesen werden kann.	Semikolon ";"

Einstellung	Beschreibung	Beispiel
Zeichenkodierung	Diese Einstellung definiert, welche Kodierung die angegebene Datei aufweist. Ein guter Weg, um dies festzustellen ist, die Datei im Editor "Notepad++" zu öffnen und dann das Menü "Encoding" anzuwählen.	UTF_8
Kopfzeile	Wenn die Option aktiviert ist, wird die erste Zeile der Datei als Überschrift gewertet und nicht verarbeitet.	-

### Quelle "Code Snippet"

Wird die Einstellung hingegen auf "Code Snippet" gesetzt, so muss der Datenzugriff gescrripted werden. Wie bei allen Code Snippets steht der volle Funktionsumfang von Groovy zur Verfügung. Beispielsweise können hier Daten aus einer SQL-Datenbank oder einem anderen Dateiformat wie XML ausgelesen werden. Damit die Daten anschließend korrekt verarbeitet werden können, muss das Code Snippet den Datentyp `List<List<String>>` zurückgeben. Die innere Liste stellt dabei eine Zeile aus mehreren Werten dar (die erste "Spalte" befindet sich in der inneren Liste im Index 0 und so weiter). Die äußere Liste stellt dabei die Zeilen dar, wobei die erste Zeile in der äußeren Liste sich im Index 0 befindet.

Das nachfolgende Beispiel zeigt dies in vereinfachter Weise:

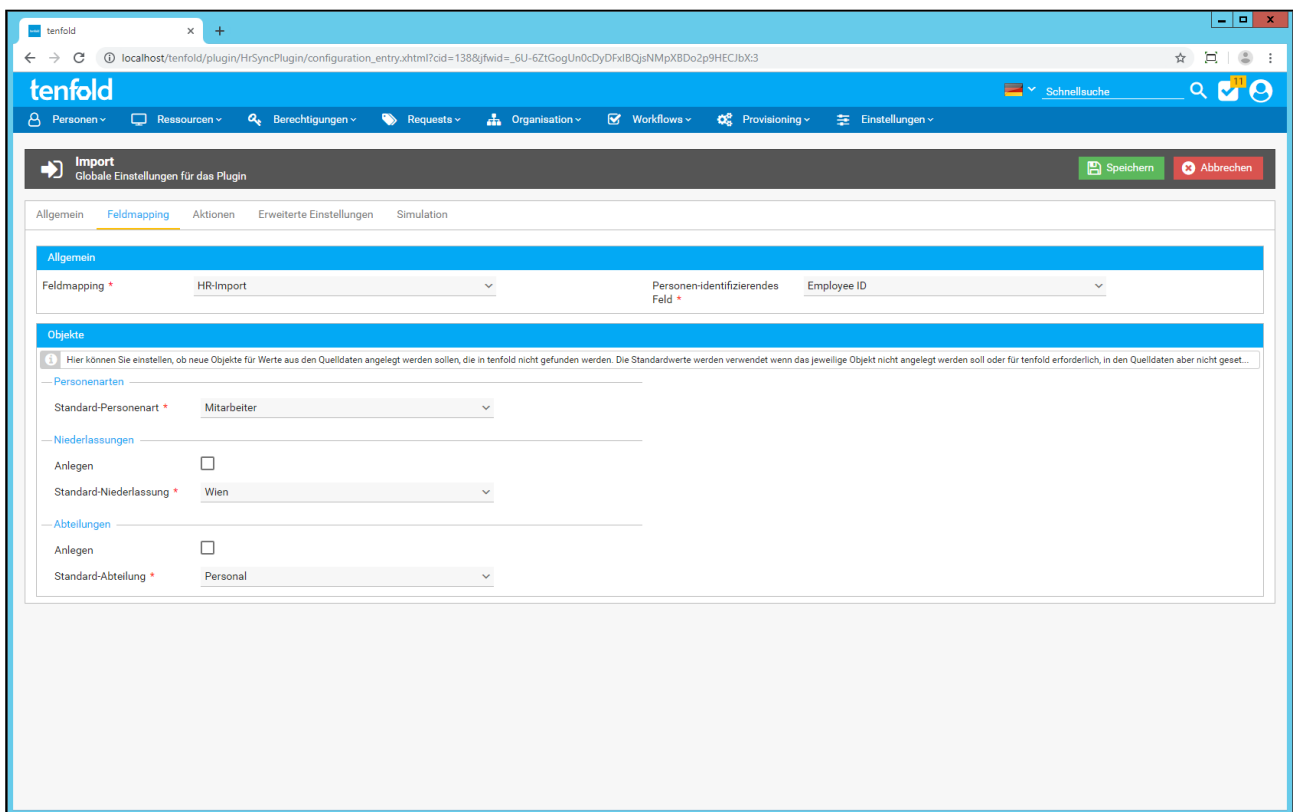
```
List<List<String>> table = new ArrayList<List<String>>()
while (source.hasNext())
{
    List<String> line = source.next()
    table.add (line)
}
return table
```

#### Hinweis zum Feldmapping

Das in weiterer Folge beschriebene Feldmapping ist unabhängig von der Quelle. Das bedeutet "Column01" bezieht sich entweder auf die erste Spalte der CSV-Datei oder auf die erste Spalte der vom Code Snippet zurückgelieferten Tabelle.

### 14.7.3 Feldmapping

Das Feldmapping des Imports legt fest, in welcher Weise die Felder der Importdatei (oder Tabelle) in die entsprechenden Personfelder in tenfold geschrieben werden.



Auf diesem Karteireiter wird lediglich festgelegt, welches Feldmapping zum Einsatz kommen soll. Das Feldmapping muss folglich bereits konfiguriert sein, bevor der Import gespeichert werden kann. Es muss beim Feldmapping auf jeden Fall der Typ "Import - CSV" ausgewählt werden (dies gilt auch, wenn die Quelle "Code Snippet" ist). In der Einstellung "Feldmapping" wird das gewünschte Feldmapping eingetragen, welches zuvor schon angelegt wurde.

## Personen-identifizierendes Feld

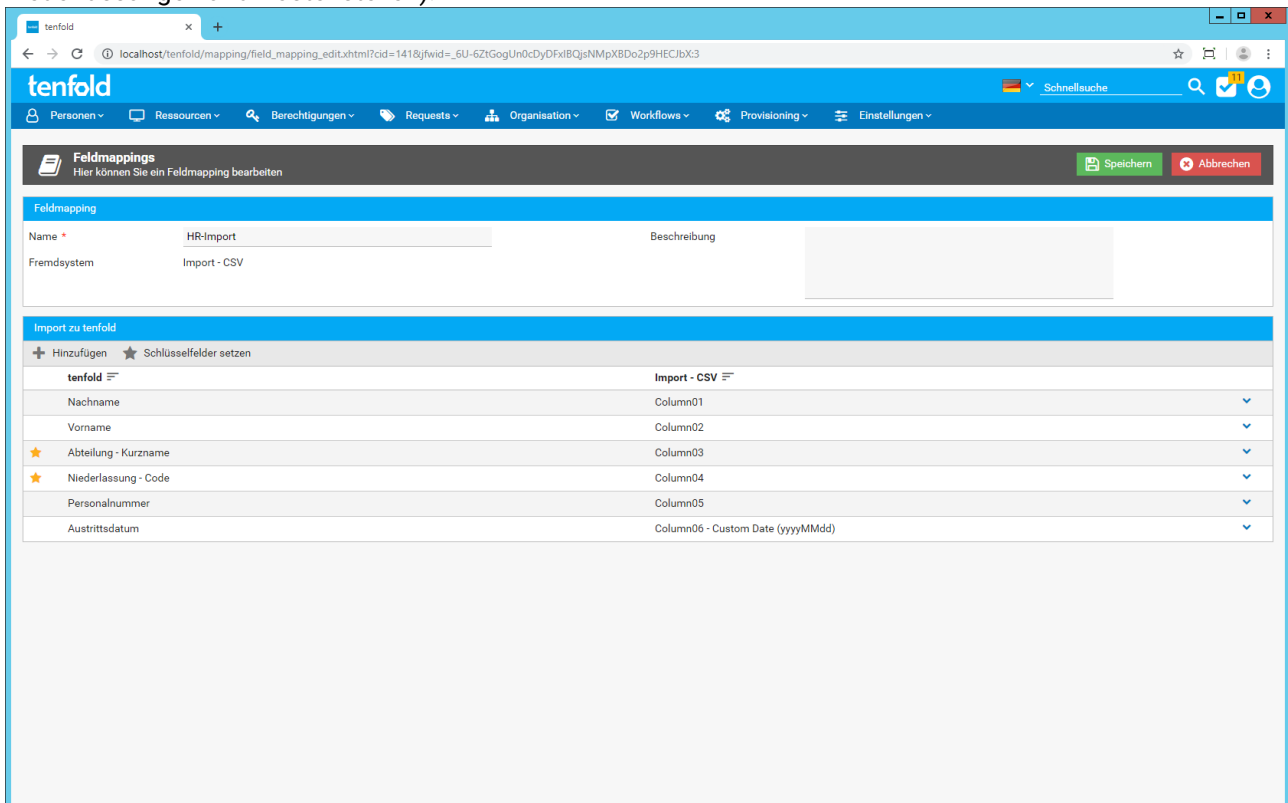
In dieser Einstellung muss das Personenfeld eingetragen werden, welches als Schlüsselfeld dienen soll. In der Importquelle muss eine Spalte existieren, welches eine Person eindeutig identifiziert. Das ist notwendig, damit tenfold den Datensatz eindeutig einer Person in der tenfold-Datenbank zuordnen kann. Es werden grundsätzlich alle Personenfelder angezeigt, weshalb Sie darauf achten müssen, dass das gewünschte Personenfeld im Feldmapping auch einer Spalte der Importquelle zugeordnet ist.

### Hinweis zur Eindeutigkeit

Bei einem Importvorgang liest tenfold zuerst alle betreffenden Personendatensätze aus der Datenbank. Im nächsten Schritt wird überprüft, ob der Wert für das hier ausgewählte Personenfeld bei diesen geladenen Personen tatsächlich eindeutig ist. Ist das nicht der Fall, bricht der Import an dieser Stelle bereits ab und liefert eine Fehlermeldung. Dieses Verhalten ist notwendig, da tenfold sonst die betreffende Person nicht eindeutig identifizieren kann. Beispiel: Ist als identifizierendes Feld "Employee ID" eingetragen (diese Feld muss im Feldmapping einer bestimmte Spalte der Importquelle zugeordnet sein) und gibt es unter den betreffenden Personen zwei Personen, die im Feld "Employee ID" (Personalnummer) den gleichen Wert eingetragen haben, so wird der Import fehlschlagen.

## Objekte

Der Bereich "Objekte" ist variabel und hängt vom zugrundeliegenden Feldmapping ab. Es werden alle "Objekte" angezeigt, die im Feldmapping konfiguriert sind. (Objekte in diesem Sinne sind Personenfelder, die keinen Textinhalt haben, sondern sich auf andere Einträge in tenfold beziehen, wie zum Beispiel Abteilungen, Niederlassungen und Kostenstellen).



Wird in der Importquelle, wie in diesem Beispiel, der Kurzname der Abteilung bereitgestellt, so muss das Feldmapping sich eindeutig auf dieses Feld (Kurzname) der Abteilung beziehen (es kämen noch andere Felder, wie zum Beispiel der volle Name in Betracht). Die unterschiedlichen Einstellungen im Bereich "Objekte" legen fest, wie tenfold reagieren soll, wenn die Übersetzung von der Importquelle (Kurzname der Abteilung) auf das Objekt in tenfold ("Abteilung") nicht funktioniert, da der Feldinhalt in der Importquelle im entsprechenden tenfold-Feld nicht bekannt ist.

- Die Einstellung "Anlegen" bedeutet, dass, wenn in diesem konkreten Beispiel ein Abteilungskurzname in der Importquelle geliefert wird, der in tenfold bei keiner Abteilung eingetragen ist, diese Abteilung tenfold automatisch neu angelegt wird. Es ist jedoch wichtig zu beachten, dass dazu alle notwendigen Felder des Objekts (dies lässt sich über die jeweilige Stammdatenmaske anhand der roten Stern-Icons erkennen) in der Importquelle vorhanden sind und über das Feldmapping zugeordnet sind. Ansonsten ist für tenfold nicht möglich, das Objekt in der Datenbank anzulegen, da die Datenbank das Vorhandensein aller benötigten Attribute voraussetzt. Eventuell ist beim Anlegen neuer Einträge das zusätzliche Anlegen von Sub-Objekten erforderlich (zum Beispiel "Region" für "Abteilung"). Dafür gilt analog, dass die benötigten Felder in der Importquelle verfügbar und über Feldmapping zugeordnet sein müssen.
- Ist diese Einstellung jedoch nicht aktiviert, so wird im Falle, dass das Objekt nicht zugeordnet werden kann, automatisch das Objekt im Feld "Standard-" zugeordnet. Findet tenfold somit eine in der

Importquelle benannte Abteilung nicht, erhält die Person automatisch die Abteilungszugehörigkeit der "Standard-Abteilung".

## 14.7.4 Aktionen

Der Bereich "Aktionen" steuert, welche Operationen der Import grundsätzlich ausführen soll und wie auf bestimmte Ereignisse reagiert werden soll.

The screenshot shows the 'Import' configuration page in the tenfold system. The 'Aktionen' tab is selected, showing settings for actions performed during imports. Under 'Personen-Aktionen', 'Personen anlegen', 'Personen ändern', and 'Personen sperren' are checked, while 'Personen löschen' is not. The 'Personenkreis' section has a dropdown set to 'Alle'. The 'Requests' section has 'Genehmigungsmodus' set to 'Requests automatisch genehmigen' and 'Request-Quelle' set to 'HR System'. A warning section at the bottom is currently inactive.

### Personen-Aktionen

Folgende Einstellungen steuern, ob der Import grundsätzliche Operationen ausführt:

- Personen anlegen
- Personen ändern
- Personen sperren
- Personen löschen

#### Hinweis

Ist eine der Operationen nicht aktiviert, bedeutet dies, dass tenfold diese Operation grundsätzlich nicht ausführt (für keinen Datensatz aus der Importquelle), selbst wenn die restliche Konfiguration dies vorsehen würde.

Zusätzliche stehen nachfolgende Optionen zur Verfügung, wobei die betroffene Person je nach Einstellung entweder gesperrt oder gelöscht wird, oder die Einstellung ignoriert werden soll. Beide Einstellungen beziehen sich auf das Sperren oder Löschen von Personen, die in der Importquelle als solches markiert sind, oder gänzlich fehlen. Welche Einstellung zu wählen ist, hängt davon ab, wie die Importquelle aufgebaut ist:



- Wenn eine Person in der Quelle als gelöscht gilt, wenn der zugehörigen Datensatz nicht mehr geliefert wird (somit von einem zum nächsten Import "fehlt"), dann ist die Einstellung "Aktion, wenn Person nicht in den Quelldaten vorkommt" relevant.
- Wenn die Person jedoch als gelöscht gilt, wenn ein bestimmtes (zugeordnetes) Austrittsdatum erreicht oder überschritten wurde, so ist die andere Einstellung relevant.

Die Einstellungen verhalten sich dabei wie folgt:

- Aktion bei Erreichen des Austrittsdatum: Diese Option kommt zum Tragen, wenn das Feld "Leaving Date" (Austrittsdatum) im Feldmapping zugeordnet wurde und das Datum im entsprechenden Datensatz der Importquelle erreicht oder überschritten wurde. Achtung: Die Einstellung steht auch zur Verfügung, wenn das Feld nicht zugeordnet wurde. In diesem Fall, sollte die Option auf "Ignorieren" gestellt werden.
- Aktion, wenn Person nicht vorkommt: Diese Option kommt zum Tragen, wenn ein Datensatz als "zu löschen" oder "zu sperren" interpretiert werden soll, wenn er plötzlich in der Importquelle fehlt (und zuvor vorhanden war). Das Fehlen eines Datensatzes wird festgestellt durch den Vergleich der in der Importquelle vorhandenen Personen im Vergleich zur Menge der Personen, die im Bereich "Personenkreis" definiert sind.

#### **Achtung - Schadensmöglichkeit**

Die Option "Aktion, wenn Person nicht vorkommt" ist mit größter Vorsicht zu verwenden. Sollte es beispielsweise dazu kommen, dass die Importquelle auf einen Fehler läuft (Export aus HR-System schlägt fehl) und eine leere Datei exportiert, würde dies zur Folge haben, dass alle Personen aus dem Personenkreis gelöscht oder gesperrt werden (da alle Personen in der Importquelle fehlen). Um den Effekt abzufedern, kann die Option "Warnung bei Überschreitung" (siehe unterhalb) genutzt werden. Allerdings entsteht in diesem Fall dennoch ein (beschränkter) Schaden an den Personendaten. Es ist somit dringend empfehlenswert, ein Dateiformat zu wählen, in welchem gelöschte Personen durch ein Austrittsdatum identifiziert werden.

## **Personenkreis**

Der Personenkreis legt fest, welche Personen zu Beginn des Import aus der tenfold-Datenbank geladen werden und für Vergleiche zwischen Datenbank und Importquelle herangezogen werden. Der Personenkreis umfasst entweder alle Personen, oder kann über eine Feldregel (siehe [Feldregeln](#) (see page 562)) eingeschränkt werden. Beispielsweise kann der Personenkreis auf eine bestimmte Personenart reduziert werden.

#### **Hinweis**

Der Personenkreis sollte so gewählt werden, dass er nur die Personen umfasst, die auch in der Importquelle vorhanden sind (zum Beispiel alle internen Mitarbeiter, oder alle Mitarbeiter an einer bestimmten Niederlassung).

## **Requests**

Dieser Abschnitt legt Einstellungen fest, welche die Request-Erstellung und Behandlung der Workflows festlegen. Die Einstellung "Genehmigungsmodus" umfasst folgende Möglichkeiten:

Option	Verhalten	Workflow	Provisionierung	Profile
Mit der Request-Person genehmigen	Diese Einstellung bedeutet, dass der bei der jeweiligen Personenart hinterlegte Workflow gewöhnlich abläuft	Normal	Normal	Abgleich erfolgt
Requests automatisch genehmigen	Mit dieser Option werden alle vorgesehenen Workflow-Schritte automatisch durch das System freigegeben	Automatisch freigegeben	Normal	Abgleich erfolgt
Requests als abgeschlossen anlegen	Die Einstellung wird lediglich bei initialen Datenkorrekturen verwendet (siehe Hinweis unterhalb)	Wird übersprungen	Wird übersprungen	Abgleich erfolgt

### Hinweise zur Option "Requests als abgeschlossen anlegen"

Diese Option sollte lediglich für initiale Datenkorrekturen verwendet werden. Das Verhalten der Option sieht vor, dass beim Import keine Workflows zur Anwendung kommen, und auch die Provisionierung, die bei der Personenart konfiguriert ist, übersprungen wird (damit werden die empfangenen Änderungen nicht in Drittsysteme übertragen). Es erfolgt lediglich eine Anpassung der Daten in der tenfold-Datenbank.

#### Warnung

Bei dieser Option findet, trotz fehlender Provisionierung der Schritte, die in der Personenart hinterlegt sind, allerdings sehr wohl der Abgleich der Profile statt. Ergeben sich für einen Datensatz Änderungen in der Profilzuordnung werden die resultierenden Requests angelegt und auch die Provisionierung für diese Requests gewöhnlich durchgeführt. Es sollte somit sichergestellt werden, dass, wenn diese Option zur Anwendung kommt, kein Profilabgleich mit der entsprechenden Request-Quelle konfiguriert ist.

Außerdem ist zu beachten, dass die Ereignisse "Request - Before Close" und "Request - After Close" für alle Requests aufgerufen werden. Eventuelle Plugin-Konfigurationen, die Aktionen mit diesen Ereignissen verknüpfen (beispielsweise das Versenden einer Mail) werden demnach gewöhnlich ausgeführt.

### Hinweise zur Nutzung von Profilen

Wenn die Option "Mit der Request-Person genehmigen" oder "Requests automatisch genehmigen" im Rahmen des Regelbetriebs (nicht für initiale Korrekturen) gewählt wurde, so muss beachtet werden, dass ein Profilabgleich nur dann stattfindet, wenn dieser explizit für die gewählte Request-Quelle konfiguriert wurde. Eine Änderung der Abteilung, ohne konfigurierten Profilabgleich, führt zwar zu einer Anpassung der Stammdaten in tenfold, und über die Provisionierung in der Personenart, auch in den Drittsystemen, aber es findet keine Zuordnung oder Entfernung von Profilen statt. Um die automatische Anpassung der Profile zu aktivieren, muss ein Eintrag auf der Maske "Automatischer Abgleich" hinzugefügt werden, wo die Request-Quelle der im jeweiligen Import angegeben Quelle entspricht.

Modus	Beschreibung	Request-Quelle	Request-Typ	Gültig ab	Person anlegen
Regel	All HR requests	HR System	*	03.07.2019 22:24:09	user demo (demouser) ▼
Regel	All tenfold requests	tenfold	*	03.07.2019 22:24:09	user demo (demouser) ▼

## Request-Quelle

Die Einstellung Request-Quelle gibt an, welche Quelle in die angelegten Requests eingetragen werden soll.

### Hinweis

Als Request-Quelle sollte keinesfalls "tenfold" oder "Active Directory" ausgewählt werden. Empfohlen ist, für jeden Import eine eigene Quelle zu erstellen und anzugeben, denn damit lassen sich Requests auf der Request-Liste sehr einfach nach Ursprung filtern.

Objekt	Angefordert für	Angefordert am	Ticket	Zuletzt genehmigt von	Request-Quelle	Request-Typ	Request-Status
Person masterdata change	Fred Belspiel	23.07.2019 13:35:27			HR System	+ Neu	FERTIG
Person masterdata change	Mayer Klaus	23.07.2019 13:35:09			HR System	+ Neu	FERTIG

## Warnung

Das Plugin bietet die Möglichkeit, eine Warnung zu generieren (und den Import abubrechen), wenn ein Durchgang des Imports eine Anzahl von Änderungen erzeugen würde, die über einem konfigurierten Schwellwert liegt. Damit soll verhindert, bzw. abgemildert werden, dass ein Importvorgang aufgrund von fehlerhaften Daten in der Importquelle oder aufgrund von fehlerhafter Konfiguration des Plugins, eine große Menge von Daten zerstört.

Um die Warnung zu aktivieren, müssen folgende Einstellungen hinterlegt werden:

- Max. Anzahl von Änderungen: Gibt an, wie viel Änderungen in einem Durchgang maximal generiert werden dürfen, ohne, dass es zu einer Warnung kommt
- Vorlageneinstellung: Legt die E-Mail-Vorlage fest, welche aufgrund der Warnung verschickt wird (eine Warnung auf der Oberfläche ist nicht möglich, da der Import über den Job im Hintergrund abläuft)
- Empfänger: Hier können ein oder mehrere E-Mail-Adressen hinterlegt werden, welche die Warnung erhalten sollen

Kommt es zu einer Warnung, so wird die E-Mail an die eingestellten Empfänger versendet. Der Import kann nicht erfolgreich abgeschlossen werden. Die Warnung kann nunmehr zwei Ursachen haben, die folgendermaßen zu behandeln sind:

- Die hohe Anzahl der Änderungen ist nicht legitim: Es liegt ein Fehler in den Importdaten oder der Konfiguration vor. Der Fehler muss analysiert und korrigiert werden. Anschließend kann der Import wieder erfolgreich abgeschlossen werden.
- Die Anzahl ist tatsächlich legitim: Handelt es sich tatsächlich um eine sehr hohe Anzahl von gewollten Änderungen, so muss die Option "Max. Anzahl von Änderungen" vorübergehend auf einen noch höheren Wert gestellt werden (so hoch, dass es zu keiner Warnung kommt). Anschließend muss der

Import-Job manuell gestartet werden. Nachdem der Job erfolgreich abgeschlossen wurde, sollte die Anzahl wieder auf den ursprünglichen Wert gesetzt werden.

### 14.7.5 Erweiterte Einstellungen

In diesem Abschnitt können Code Snippets hinterlegt werden, die dazu dienen, die Verarbeitung per Script spezifisch anzupassen und zu beeinflussen. Folgende Snippets können konfiguriert werden:

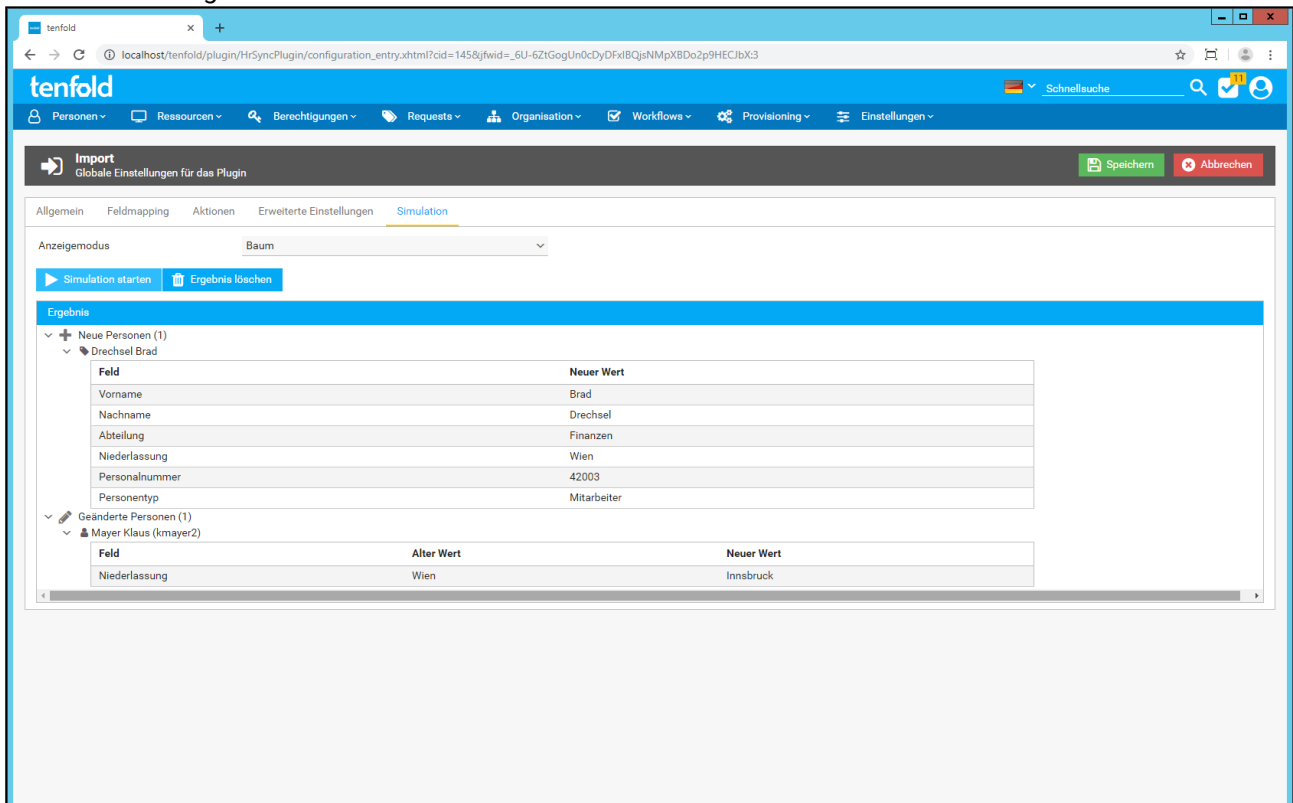
Code Snippet	Beschreibung	Input
nach Auslesen der Quelldaten	Wird ausgeführt, nachdem die Quelldaten vollständig gelesen wurden und bevor das Feldmapping durchgeführt wird. Das Snippet eignet sich dazu, die Quelldaten zu manipulieren, bevor sie verarbeitet werden.	<ul style="list-style-type: none"> <li>Parameter: <b>"rows"</b> Datentyp: <b>List&lt;Row&gt;</b> beinhaltet alle Daten.</li> <li>Innerhalb des Objekts <b>Row</b>: <ul style="list-style-type: none"> <li><b>List&lt;String&gt; columns</b>: beinhaltet die Spaltenwerte (Index 0 entspricht Spalte 1)</li> <li><b>int index</b>: beinhaltet die aktuelle Zeilennummer</li> </ul> </li> </ul>
nach Feldmapping für neu erstellte Person	Wird ausgeführt, nachdem das Feldmapping erfolgt ist. Es wurde bereits ein Person-Objekt im Speicher erzeugt, in welchem die Attribute befüllt sind. Dieses Snippet wird nur für Datensätze ausgeführt, die zur Neuanlage einer Person führen	<ul style="list-style-type: none"> <li>Parameter <b>"result"</b> Datentyp <b>"PersonSnippetResult"</b> beinhaltet: <ul style="list-style-type: none"> <li><b>"ignorePerson"</b> Datentyp <b>boolean</b>: Wird dieses Feld gesetzt, wird die aktuelle Person bei der Verarbeitung ignoriert</li> </ul> </li> <li>Parameter <b>"person"</b> Datentyp <b>Person</b>: Das Person-Objekt nach dem Feldmapping</li> <li>Parameter <b>"personMasterdata"</b> Datentyp <b>PersonMasterdata</b>: das PersonMasterdata-Objekt nach dem Feldmapping</li> <li>Parameter <b>columns</b> Datentyp <b>List&lt;String&gt;</b>: beinhaltet die Spaltenwerte (Index 0 entspricht Spalte 1) vor dem Feldmapping</li> </ul> <p>Siehe dazu auch: <a href="#">Datentypen</a>(see page 608)</p>
nach Feldmapping für geänderte Person	Wird ausgeführt, nachdem das Feldmapping erfolgt ist. Es wurde bereits ein Person-Objekt im Speicher erzeugt, in welchem die Attribute befüllt sind. Dieses Snippet wird nur für Datensätze ausgeführt, die zur Änderung einer bestehende Person führen	Analog zu oberhalb

#### Input / Output Behandlung

Alle Snippets erhalten als Input Objekte, welche anschließend verändert werden können. Die im Snippet veränderten Objekte werden wieder an die weitere Verarbeitung zurückgegeben. Es ist nicht notwendig, dass die Snippets Daten via "return"-Statement zurückgegeben werden.

## 14.7.6 Simulation

Die Simulation dient dazu, vor der Aktivierung einer initialen oder geänderten Konfiguration zu sehen, welche Requests tenfold bei einer produktiven Ausführung erzeugen würde. Die Requests werden dabei nicht tatsächlich erzeugt.



Um eine Simulation zu starten, drücken Sie den Button "Simulation starten". Mit dem Button "Ergebnis löschen" können sie die aktuelle Anzeige entfernen, um anschließend eine neue Simulation zu starten. Es stehen grundsätzlich zwei Modi zur Verfügung, welche jeweils folgende Vorteile bieten:

- **Baum:** Dieser Modus zeigt alle Änderungen auf Personen- und Feldebene detailliert und übersichtlich an. Verarbeitungsinformationen werden dabei ausgeblendet.
- **Text:** Dieser Modus zeigt nicht alle Feldänderungen an, gibt dabei aber Auskunft über bestimmte Verarbeitungseignisse im Hintergrund, beispielsweise wenn ein Objekt (Abteilung, etc. - siehe oben) nicht übersetzt werden kann.

## 14.8 SAP User Lifecycle

Das SAP User Lifecycle Plugin dient zur Verwaltung Benutzern in SAP ERP (ehemals R/3) auf Basis des NetWeaver ABAP Stack. Der Funktionsumfang entspricht im wesentlichen dem [Active Directory User Lifecycle](#)(see page 665) Plugin. Ein SAP-System wird durch eine Ressource repräsentiert.

## 14.8.1 Konfiguration

Für jedes SAP-System muss über den Button "Neu" ein Eintrag im Plugin angelegt werden. Für jeden Eintrag muss anschließend die Konfiguration vorgenommen werden.

### Hinweis

Mit dem Button "Verbindung testen" kann überprüft werden, ob mit den angegebenen Verbindungsdaten eine Verbindung zum SAP-System aufgebaut werden kann. Es ist empfohlen, diesen Test durchzuführen, wenn die Verbindungsdaten initial erfasst oder geändert werden, um Fehler ausschließen zu können, die aus einer fehlenden Verbindung zum SAP-System entstehen.

Folgende Einstellungen stehen zur Verfügung:

Einstellung	Beschreibung
<b>Verbindungseinstellungen</b>	
Name	Der Anzeigename des Systems auf der tenfold-Oberfläche
Mode	Aktuell ist nur "SAP Instanz" für Einzelsysteme verfügbar. In Zukunft wird es eine Einstellung "SAP ZBV" geben, welche die Einbindung eines ganzen ZBV-Verbunds erlaubt
Zugangsdaten	Auswahl der Zugangsdaten, welche den Hostnamen, RFC-Benutzer und Passwort für die Verbindung zu NetWeaver beinhaltet

Instanznummer	Instanznummer des jeweiligen SAP-Systems
Client	Legt fest mit welchem Mandant des SAP-Systems die Verbindung aufgebaut werden soll
Sprache	Legt fest, mit welcher Spracheinstellung die Verbindung aufgebaut wird. Dies hat lediglich Einfluss etwaige Fehlermeldungen, die von SAP geliefert werden.
Debug-Informationen ausgeben	Wenn die Option aktiviert ist, werden zusätzliche Debug-Information vom RFC-Client in das Logfile geschrieben
<b>Benutzernamensvergabe</b>	
Benutzername	Der Benutzername für neue SAP-Benutzer kann entweder aus dem Personenfeld "Benutzername" übernommen werden, oder extra für SAP generiert werden. In diesem Fall muss der Benutzernamengenerator, der verwendet werden soll, angegeben werden. Beim Generator ist darauf zu achten, dass dieser so konfiguriert ist, dass er die relevanten SAP-Ressourcen auf Duplikate prüft.
<b>Sync-Einstellungen</b>	
Sync aktiv	Diese Einstellung legt fest, ob eine Rücksynchronisierung der Daten aus SAP stattfinden soll. Wenn diese Einstellung aktiviert ist, werden folgende Aktionen ausgelöst (vorausgesetzt, der Benutzer kann auf eine tenfold-Person übersetzt werden (siehe Einstellung "Attribut für Personenzuordnung")): <ul style="list-style-type: none"> <li>• Zuordnung der entsprechenden SAP-System-Ressource, wenn der Benutzer in SAP existiert, aber in tenfold noch nicht zugeordnet ist</li> <li>• Entfernung der SAP-System-Ressource, wenn der Benutzer in SAP nicht mehr existiert</li> <li>• Zuordnung der Rollen als Berechtigungen</li> </ul> Achtung: Es erfolgt keine Aktualisierung der Stammdaten von SAP zu tenfold. Benutzer, welche nicht übersetzt werden können, werden ignoriert.
Attribut für Personenzuordnung	Legt fest, welches Attribut für die Zuordnung eines SAP-Benutzers zu einer Person in tenfold verwendet werden soll. Es kann entweder der Benutzername gewählt werden (dabei muss der SAP-Benutzername dem Personenfeld "Benutzername" entsprechen), oder es kann ein Code Snippet verwendet werden.
<b>Passwort Benutzeranlage</b>	
Empfänger	Diese Option legt fest, wer nach der Anlage eines neuen Benutzers das Initialpasswort erhalten soll. Für die Auswahl "Vorgesetzter" muss das Personenfeld konfiguriert und gepflegt sein.



Vorlageneinstellung	Es kann mit dieser Einstellung festgelegt werden, welche E-Mail-Vorlage für den Versand des Initialpassworts verwendet wird.
<b>Password-Reset Einstellungen</b>	
Temporäres Passwort	Mit dieser Option kann bei der Passwortrücksetzung ein temporäres Passwort versendet werden.
Alternative Empfänger	Für den Fall, dass das Email nicht zugestellt werden kann, muss ein alternativer Empfänger angegeben werden.
Vorlageneinstellung	Auswahl ob die Standard-, oder eine benutzerdefinierte Vorlage verwendet werden soll.
<b>Passwortrichtlinie</b>	
Passwortrichtlinie	Mit dieser Option kann eine Passwortrichtlinie ausgewählt werden, nach dessen Komplexitätsanforderungen sämtliche automatisch generierten Passworte erzeugt werden.

## 14.8.2 Zugangsdaten

In den verwendeten Zugangsdaten sind folgende Einstellungen zu treffen:

- Name: Anzeigenname für die tenfold-Oberfläche
- Verbindung 1: DNS-Hostname oder IP-Adresse des SAP-Systems
- Benutzername: Benutzername des RFC-Benutzers, mit welchem die Verbindung aufgebaut wird.
- Passwort: Passwort des RFC-Benutzers

tenfold

localhost/tenfold/settings/credentials/credentials\_edit.xhtml?cid=1648&jfwid=\_6U-6ZtGogUn0cDyDFxIBQjsNMpXBDo2p9HECibX9

tenfold

Personen Ressourcen Berechtigungen Requests Organisation Workflows Provisioning Einstellungen

**Zugangsdaten bearbeiten**  
Die Einstellungen für die Verbindung sind je nach System unterschiedlich zu interpretieren

Speichern Abbrechen

**Zugangsdaten**

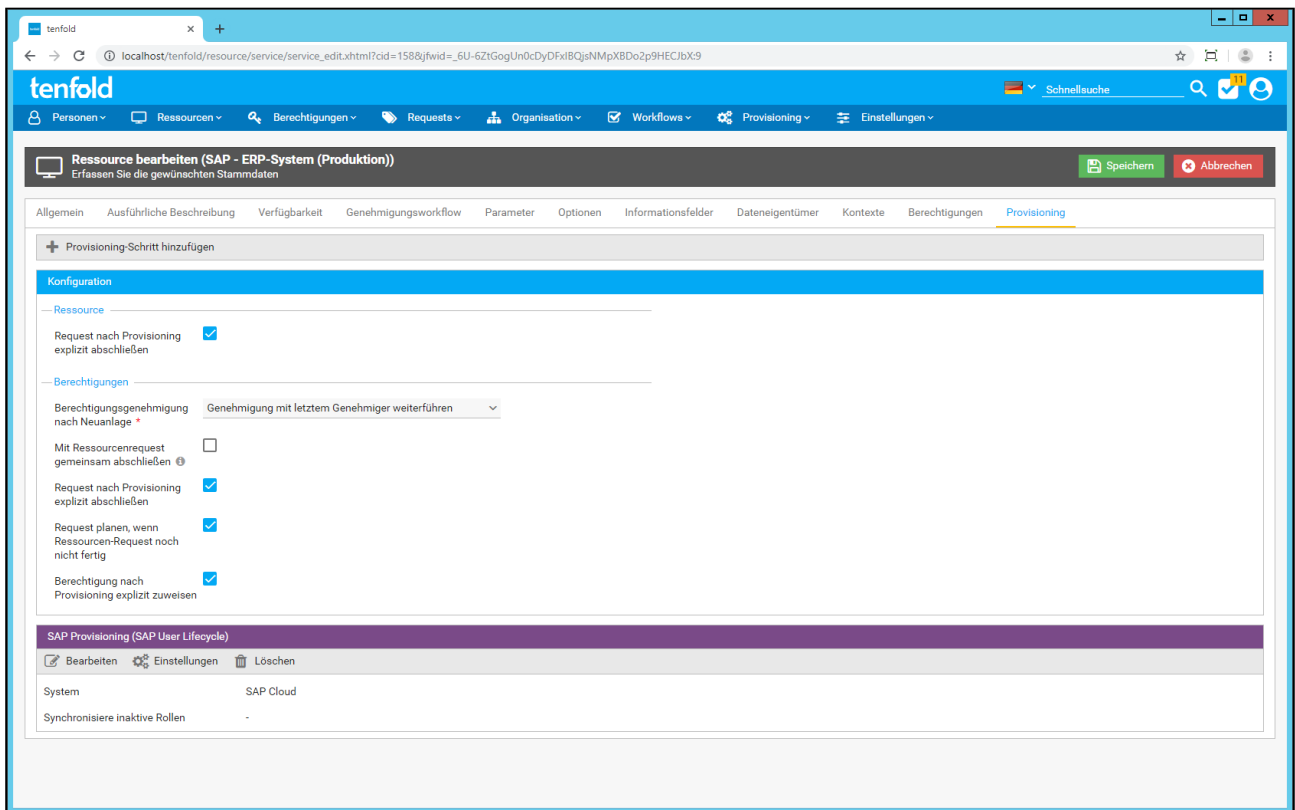
Name *	SAP Credentials
Verbindung 1	ec2-99-1999-999-99.eu-central-1.compute.amazonaws.com
Verbindung 2	
Benutzername *	SAP*
Passwort *	.....
Passwort bestätigen *	.....

### Hinweis

Beachten Sie, dass der RFC-Benutzer über die notwendigen Berechtigungen verfügen muss, um die Transaktion SU01 bedienen zu können. Zusätzlich sind RFC-Berechtigungen für alle BAPI, welche mit BAPI\_USER\_ beginnen, zu vergeben. Darüber hinaus müssen die Firewall-Einstellungen vorsehen, dass eine Verbindung vom tenfold-Server zum SAP-System über RFC möglich ist.

## 14.8.3 Ressource

Es existiert aktuell keine Funktion, um direkt aus der Plugin-Konfiguration heraus eine SAP-System-Ressource anzulegen. Dieser Schritt ist daher manuell durchzuführen.



Um eine Ressource anzulegen, die das SAP-System darstellt, muss folgendes beachtet werden:  
Folgende Optionen müssen bei der Ressource aktiviert werden:

- Aktiv
- Sperrbar
- Berechtigungsverwaltung aktiv
- Ablaufdatum anzeigen
- Benutzername: nicht generieren

Folgende Elemente sollten nicht verwendet werden:

- Parameter
- Optionen
- Informationsfelder

### Restliche Einstellungen

Zusätzlich sollten im Bereich "Kontext" keine Änderung gemacht werden, sodass nur der "DEFAULT"-Kontext verfügbar ist. Die restlichen Einstellungen, etwa zu Verfügbarkeit und Workflows können frei gewählt werden.

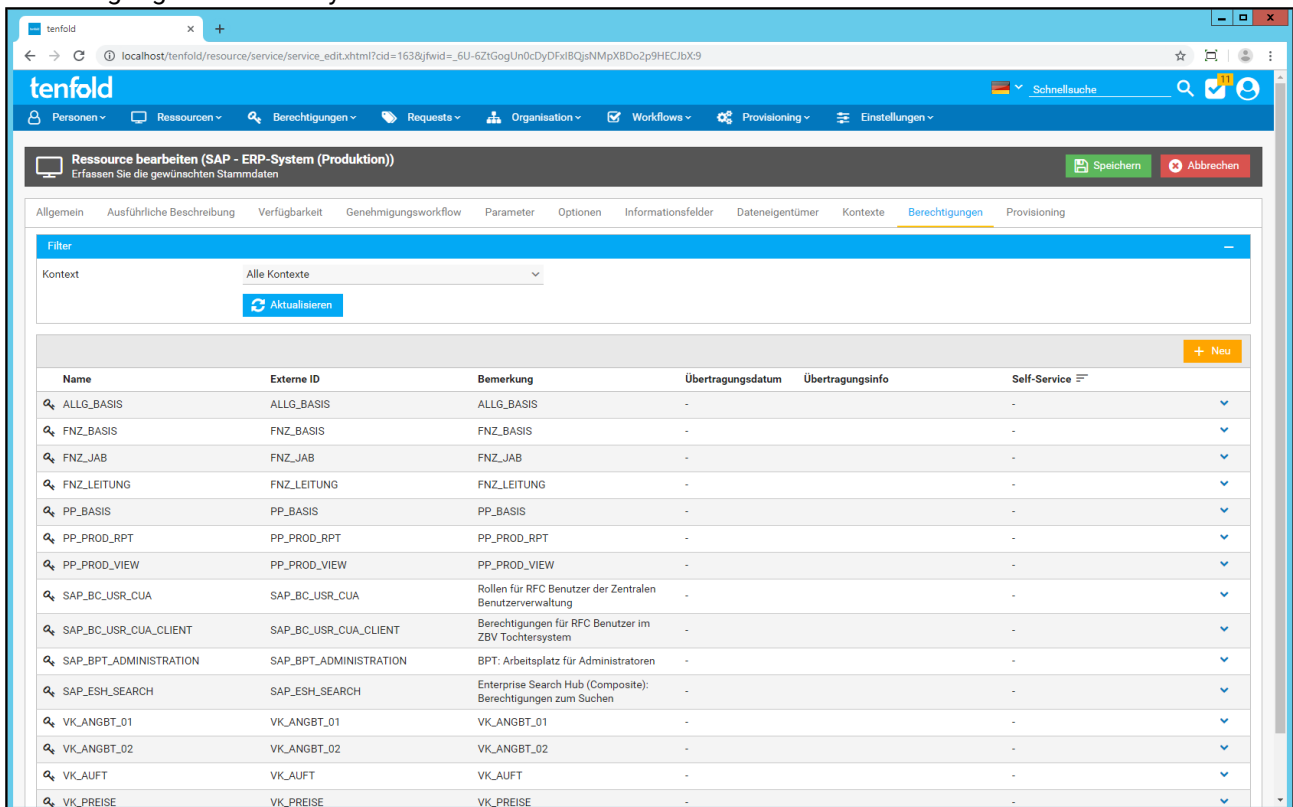
Um die Ressource nunmehr mit dem Plugin zu verknüpfen, muss im Abschnitt "Provisioning" ein Schritt hinzugefügt werden:

- Name: Bezeichnung für den Schritt, beispielsweise "SAP Provisioning"
- Plugin: SAP User Lifecycle

Anschließend muss das SAP-System, welches im vorangegangenen Schritt konfiguriert wurde, in der Option "System" eingestellt werden. Die Option "Synchronisiere inaktive Rollen" steuert, ob Rollen als Berechtigungen in tenfold angelegt werden sollen, auch wenn sie aktuell keinem SAP-Benutzer zugeordnet sind.

## 14.8.4 Synchronisierung

Anschließend muss die Synchronisierung gestartet werden, damit bestehende SAP-Benutzer und -Rollen nach tenfold übernommen werden. Für die Synchronisierung ist der Job "SAP Plugin - Sync" verantwortlich. Nachdem der Job gestartet und erfolgreich abgeschlossen wurde, müssen die SAP-Rollen im Karteireiter "Berechtigungen" der SAP-System-Ressource aufscheinen.



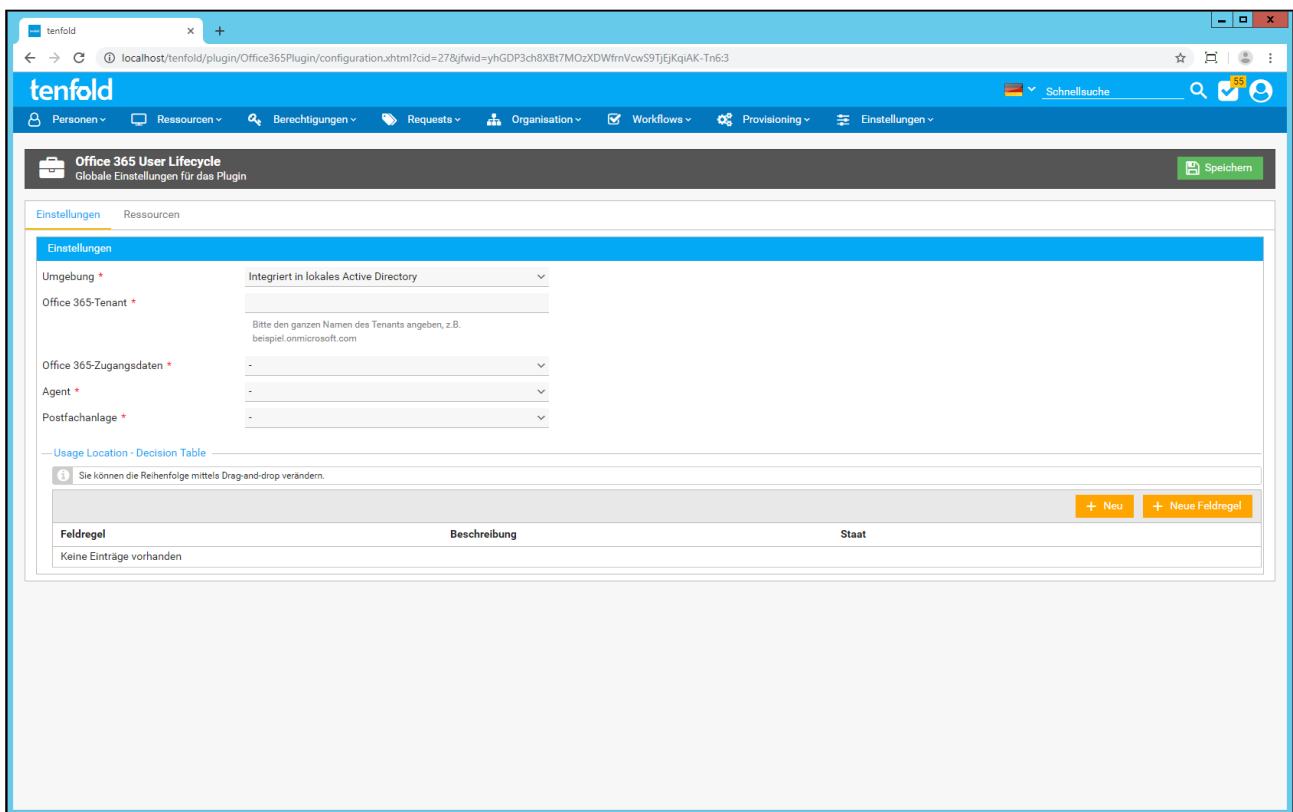
Name	Externe ID	Bemerkung	Übertragungsdatum	Übertragungsinfo	Self-Service
ALLG_BASIS	ALLG_BASIS	ALLG_BASIS	-	-	-
FNZ_BASIS	FNZ_BASIS	FNZ_BASIS	-	-	-
FNZ_JAB	FNZ_JAB	FNZ_JAB	-	-	-
FNZ_LEITUNG	FNZ_LEITUNG	FNZ_LEITUNG	-	-	-
PP_BASIS	PP_BASIS	PP_BASIS	-	-	-
PP_PROD_RPT	PP_PROD_RPT	PP_PROD_RPT	-	-	-
PP_PROD_VIEW	PP_PROD_VIEW	PP_PROD_VIEW	-	-	-
SAP_BC_USR_CUA	SAP_BC_USR_CUA	Rollen für RFC Benutzer der Zentralen Benutzerverwaltung	-	-	-
SAP_BC_USR_CUA_CLIENT	SAP_BC_USR_CUA_CLIENT	Berechtigungen für RFC Benutzer im ZBV Tochtersystem	-	-	-
SAP_BPT_ADMINISTRATION	SAP_BPT_ADMINISTRATION	BPT: Arbeitsplatz für Administratoren	-	-	-
SAP_ESH_SEARCH	SAP_ESH_SEARCH	Enterprise Search Hub (Composite): Berechtigungen zum Suchen	-	-	-
VK_ANGBT_01	VK_ANGBT_01	VK_ANGBT_01	-	-	-
VK_ANGBT_02	VK_ANGBT_02	VK_ANGBT_02	-	-	-
VK_AUFT	VK_AUFT	VK_AUFT	-	-	-
VK_PREISE	VK_PREISE	VK_PREISE	-	-	-

## 14.9 Office 365 User Lifecycle

Mit dem Office 365 User Lifecycle Plugin können Benutzern in Office 365 automatisch Lizenzen zugewiesen und entzogen werden. Darüber hinaus kann mit dem Plugin automatisch ein Postfach in Exchange Online erzeugt werden, oder, in Zusammenspiel mit dem Exchange Mailbox Lifecycle Plugin, ein Postfach in einer hybriden Exchange-Infrastruktur erzeugt werden.

### 14.9.1 Konfiguration

Die Einstellungen für das Plugin erfolgen global für die gesamte tenfold-Instanz. Es ist daher über das Plugin nicht möglich, mehr als einen Tenant einzurichten.



Auf dem Karteireiter Einstellungen müssen zuerst folgende Elemente konfiguriert werden:

Einstellung	Beschreibung
Umgebung	Aktuell wird nur die Einstellung "Integriert in lokales Active Directory" unterstützt. Das bedeutet, dass die Anlage von neuen Benutzern und Gruppen nur über das lokale Active Directory möglich ist, und auch nur synchronisierte Objekte in tenfold sichtbar sind. Die Anlage der Objekte in Office 365 erfolgt durch Azure AD Connect. (siehe <a href="https://docs.microsoft.com/de-at/azure/active-directory/hybrid/whatis-azure-ad-connect">https://docs.microsoft.com/de-at/azure/active-directory/hybrid/whatis-azure-ad-connect</a> <sup>12</sup> )
Office 365-Tenant	Hier muss der Name des einzubindenden Tenant angegeben werden, zum Beispiel "contoso.onmicrosoft.com"
Office 365-Zugangsdaten	Hier müssen die Zugangsdaten ausgewählt werden, mit welchen die Verbindung zu Office 365 hergestellt werden kann. Der ausgewählte Benutzer muss über Administratorenrechte im angegebenen Tenant verfügen. In den Zugangsdaten muss der volle Office 365-Benutzername angegeben werden (zum Beispiel Administrator@contoso.onmicrosoft.com).
Agent	Es muss der Agent ausgewählt werden, der dazu genutzt wird, um sich via PowerShell zu Office 365 zu verbinden. Die benötigten Cmdlets werden automatisch geladen. Es muss sichergestellt werden, dass der Agent eine Netzwerkverbindung zum Internet aufbauen darf.

<sup>12</sup> <https://docs.microsoft.com/de-at/azure/active-directory/hybrid/whatis-azure-ad-connect>

Einstellung	Beschreibung
Postfachanlage	Diese Einstellung steuert, ob die Anlage der Mailbox über das Exchange Mailbox Lifecycle Plugin abgehandelt werden soll (in diesem Fall erhält der Benutzer nur dann eine Mailbox, wenn ihm die entsprechende Ressource zugewiesen wurde) oder ob die Mailbox in Exchange Online automatisch bei der Zuweisung einer Lizenz erstellt werden soll.
Usage Location (Speicherort)	Hier kann über Feldregeln (siehe <a href="#">Feldregeln</a> (see page 562)) festgelegt werden, welcher Speicherort für die unterschiedlichen Benutzer in Office 365 festgelegt werden soll.

## 14.9.2 Ressource

### Anlage der Ressource

Über den Karteireiter "Ressourcen" kann eine Ressource angelegt werden, welche anschließend verwendet werden kann, um Benutzern Lizenzen zuzuweisen. Die Vorgehensweise ist analog zur Anlage einer Domain-Ressource im Active Directory User Lifecycle Plugin (siehe [Active Directory User Lifecycle](#)(see page 665)).

### Einstellen der Lizenztypen

Das Anlegen der Ressource alleine ist jedoch nicht ausreichend, um Benutzern Lizenzen zuweisen zu können. Die unterschiedlichen Lizenztypen (E1, E3, etc.), die in Ihrem Tenant zur Verfügung stehen, sind in tenfold über Berechtigungen innerhalb der Office 365 Ressource abgebildet. Wird einer Person anschließend die Ressource sowie eine oder mehrere Berechtigungen (die jeweils einem Lizenztyp entsprechen)

zugewiesen, so wird der Benutzer in Office 365 aktiviert. Das Auslesen der verfügbaren Lizenztypen erfolgt dabei aktuell nicht automatisch. Es muss somit für jeden Lizenztyp, der im Tenant verfügbar ist, eine Berechtigung in tenfold innerhalb der Office 365-Ressource angelegt werden. Dabei geht man folgendermaßen vor:

1. Bearbeiten Sie die zuvor angelegte Office 365-Ressource und wechseln Sie auf den Tab Berechtigungen. Legen Sie nun für jeden nachfolgenden Eintrag eine neue Berechtigung an.
2. Zuerst muss über PowerShell abgefragt werden, welche Lizenztypen zur Verfügung stehen. Dieser Vorgang ist hier beschrieben: <https://docs.microsoft.com/en-us/powershell/module/msonline/get-msolaccountsku?view=azureadps-1.0>
3. Die AccountSkuld muss anschließend in der neu angelegten Berechtigung in tenfold in das Feld "EID" übertragen werden (die AccountSkuld ist zum Beispiel tenfoldcontoso:E3)
4. Das Feld Name kann grundsätzlich frei gewählt werden. Es ist jedoch ratsam die Bezeichnung des Lizenztyps (zum Beispiel "E3") zu verwenden, da der Name auf der tenfold Oberfläche zur Anzeige genutzt wird.

### 14.9.3 Sync-Job

Das Plugin installiert automatisch einen Job "Office 365 User Lifecycle Plugin - Sync", welche im eingestellten Intervall die in Office 365 angelegten und lizenzierten Benutzer prüft und gegebenenfalls der entsprechenden Person in tenfold die Office 365-Ressource und die eingestellten Lizenzen (über Berechtigungen, siehe oben) zuweist. Wenn eine Zuweisung über tenfold gemacht wird, so wird dies automatisch in der Datenbank eingetragen. Der Sync erzeugt daher nur Einträge für Lizenzzuweisungen, die außerhalb von tenfold, direkt auf der Administrationsoberfläche von Azure AD gemacht wurden.

## 14.10 Generic Connector

Das Generic Connector Plugin erlaubt es Ihnen selbst Anbindungen für Systeme zu entwickeln, für die es keine vorgefertigten Anbindungen in tenfold gibt. Damit ist es möglich zum Beispiel Anbindungen für In-House Software zu erstellen. Der Generic Connector arbeitet mittels eines REST APIs, welches vom Plugin vorgegeben wird. Sie können dieses API mit einer von Ihnen bevorzugten Technologie (z.B. Java, .net, node.js) implementieren, um es mit dem Generic Connector zu verwenden.

tenfold benutzt dieses API auf zwei Arten:

- Als Datenquelle für die Synchronisierung der IST-Daten in Ihrem System
- Als Dienst um Änderungen in Ihrem System vorzunehmen

Eine genauere Beschreibung dieses APIs finden Sie später in diesem Kapitel.

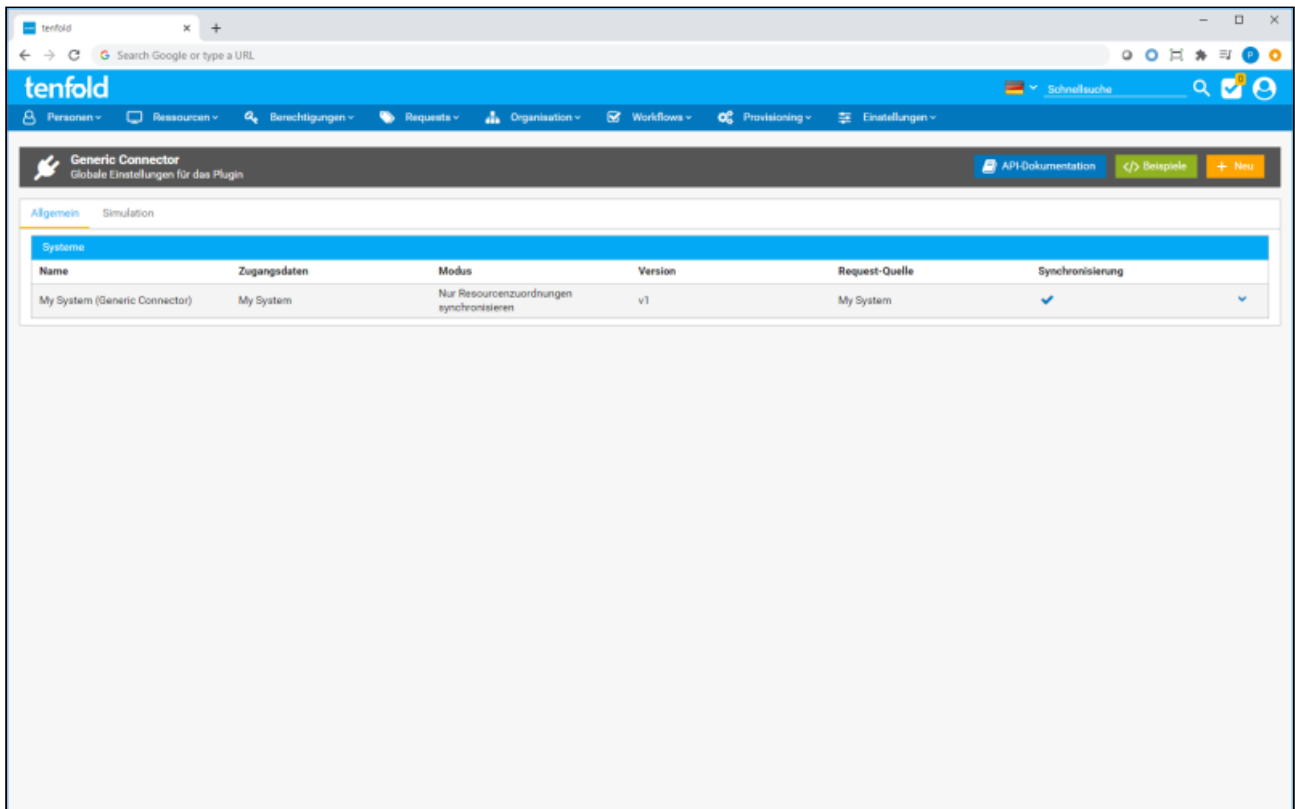
### Terminologie

Der Generic Connector wurde entwickelt, um tenfold-Kunden die Möglichkeit zu geben, In-House entwickelte Software oder Software von Drittherstellern, welche eine Anbindung entwickeln können, anzubinden. Sie könnten den Generic Connector aber auch einfach dafür verwenden, um tenfold um gewisse Funktionen zu erweitern, welche Sie selbst entwickeln. Im folgenden Kapitel wird folgende Terminologie verwendet:

- **Schnittstelle, REST-API, REST-Schnittstelle:** Dies ist die API, welche für das Zusammenspiel mit dem Generic Connector entwickelt wird.
- **System:** Die Anwendung oder sonstige Anbindung für welche die Schnittstelle implementiert wird.
- **Benutzer:** Ein Anwender oder Zugang für das System.

- **Person:** Eine von tenfold verwaltete Person, für welche Benutzer im System angelegt werden können.
- **Ressource:** Eine Ressource in tenfold (siehe [Ressourcen](#)(see page 125)) mit welcher über den Generic Connector Benutzer im System verwaltet werden.

### 14.10.1 Plugin-Einstellungen



In den Plugin-Einstellungen im Karteireiter "Allgemein" finden Sie zunächst eine Liste sämtlicher konfigurierter Systeme, falls vorhanden. Hier können Sie bestehende Systeme zur Bearbeitung auswählen, sowie neue Systeme hinzufügen. Unter "System" ist in diesem Zusammenhang ein Fremdsystem zu verstehen, für welches eine Implementierung des Generic Connector REST APIs existiert.

#### Neues System anlegen

Durch Betätigen der Schaltfläche "Neu" können Sie ein neues System für den Generic Connector anlegen. Mit dem Anlegen eines neuen Systems wird auch automatisch eine neue Ressource angelegt, welche den Lifecycle dieses Systems in tenfold abbildet.

#### Zugangsdaten

Zur Anlage eines neuen Systems benötigen Sie Zugangsdaten vom Typ "Generic Connector", welche bereits in tenfold gespeichert sind. Bevor Sie ein neues System anlegen, sollten Sie solche Zugangsdaten bereits vorbereitet haben. Siehe [Zugangsdaten](#)(see page 552) für weitere Informationen.



## Allgemeine Einstellungen

Im Karteireiter "Allgemein" legen Sie die Grundlegenden Einstellungen zur Verbindung für Ihr System fest.

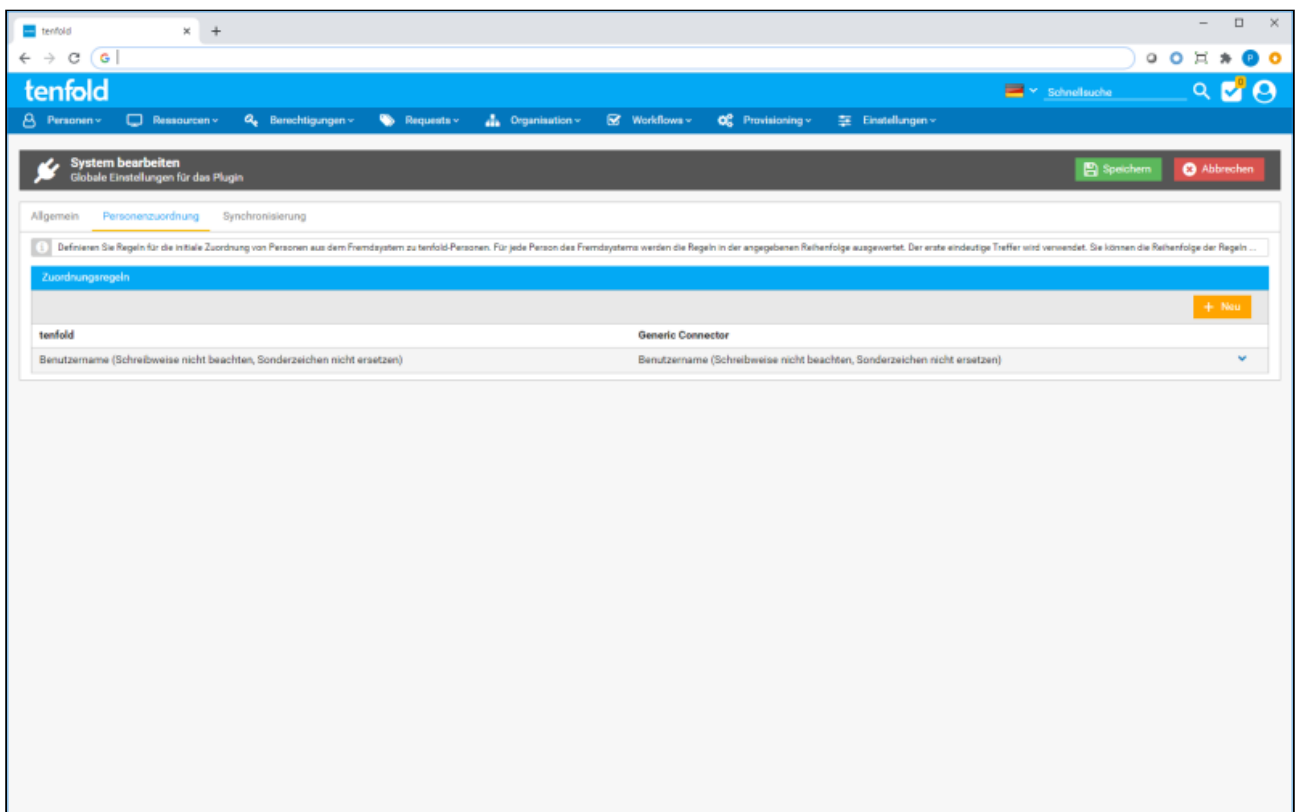
Folgende Einstellungen können hierbei getroffen werden:

Einstellung	Beschreibung
<b>Verbindungseinstellungen</b>	
Name	Der Name des neuen Systems. Dieser Name dient zur Anzeige in tenfold und ist gleichzeitig auch der Name der Ressource, welche für das System angelegt wird.
Zugangsdaten	Mit dieser Einstellung legen Sie die Zugangsdaten fest, mit welchen tenfold auf den erstellten REST Service zugreift. Diese Daten beinhalten den Benutzernamen, das Passwort sowie den Basis-URL des Services.
Authentifizierungsart	Dies legt die Authentifizierungsart fest, mit welcher sich tenfold am REST Service anmeldet. Zur Zeit wird nur HTTP BASIC Authentifizierung unterstützt.
Zertifikatsvalidierung	Sollte die Kommunikation mittels HTTPS stattfinden (was stark empfohlen wird, da Passwörter sonst als Klartext übermittelt werden), so kann mit dieser Einstellung festgelegt werden, ob tenfold alle Zertifikate akzeptiert oder ob tenfold nur Zertifikate in seinem Zertifikatsspeicher akzeptiert.

Version	Legt die API Version des Generic Connector REST Services fest. Aktuell steht nur Version 1 ("v1") zur Verfügung. Um in Zukunft Abwärtskompatibilität mit bestehenden Service Implementierungen gewährleisten zu können, muss die verwendete Version hier explizit angegeben werden.
<b>Passwort</b>	
Passwortrichtlinie	Diese Einstellung legt fest, nach welchem Schema Passwörter für neue Benutzer generiert werden. Dies wird benötigt, sollte tenfold am System neue Benutzer anlegen müssen.
Empfänger	Hier stellen Sie ein, an wen das nach obiger Richtlinie generierte Passwort gesendet werden soll. Mögliche Einstellungen hierfür sind "Anforderer" (die Person, welche den neuen Benutzer anfordert), "Vorgesetzter" (der Vorgesetzte, der Person für welche der Zugang angefordert wird, "Fixe E-Mail-Adresse" (eine E-Mail Adresse welche fest in den Einstellungen hinterlegt wird) und "Code Snippet" (Ein Stück Groovy-Code welcher dynamisch eine E-Mail Adresse zurückliefern kann).
Fixe E-Mail-Adresse	Diese Einstellung ist nur Verfügbar wenn als Empfänger "Fixe E-Mail-Adresse" ausgewählt wurde und legt eine statische E-Mail-Adresse fest, an welche alle Passwörter gewendet werden. Mehrere E-Mail-Adressen können kommagetrennt hinterlegt werden.
Code-Snippet	Diese Einstellung ist nur verfügbar wenn als Empfänger die Auswahl "Code-Snippet" getroffen wurde. Mit diesem Snippet kann Groovy-Code hinterlegt werden, welcher dynamisch E-Mail-Adressen (kommagetrennt) erzeugen kann, an welche das erzeugte Passwort gesendet wird.
Vorlageneinstellungen	Hier kann konfiguriert werden, ob die Standardvorlage für das Passwort-E-Mail verwendet werden soll oder ob eine eigens erstellte Vorlage benutzt werden soll.
Vorlage	Sollte in der obigen Einstellung "Selbsterstellte Vorlage verwenden" ausgewählt worden sein, lässt sich hier die zu verwendende E-Mail-Vorlage auswählen.

## Personenzuordnung

Im Karteireiter "Personenzuordnung" lässt sich festlegen wie tenfold bei der Synchronisierung von Daten Benutzer aus dem System mit Benutzern in tenfold verknüpft.



Bei der Anlage eines Systems ist bereits eine Regel hinterlegt, welche Personen mittels des Benutzernamens verknüpft.

Die Personenzuordnung wird nur für die erste Zuordnung eines Systemusers zu einer tenfold-Person verwendet. Nachdem eine Person und ein User zum ersten Mal über die Personenzuordnung verknüpft werden konnten, speichert tenfold die ID des Users im System bei der Ressourcenzuordnung der Systemressource zur Person und verwendet im Anschluss immer die ID des Users.

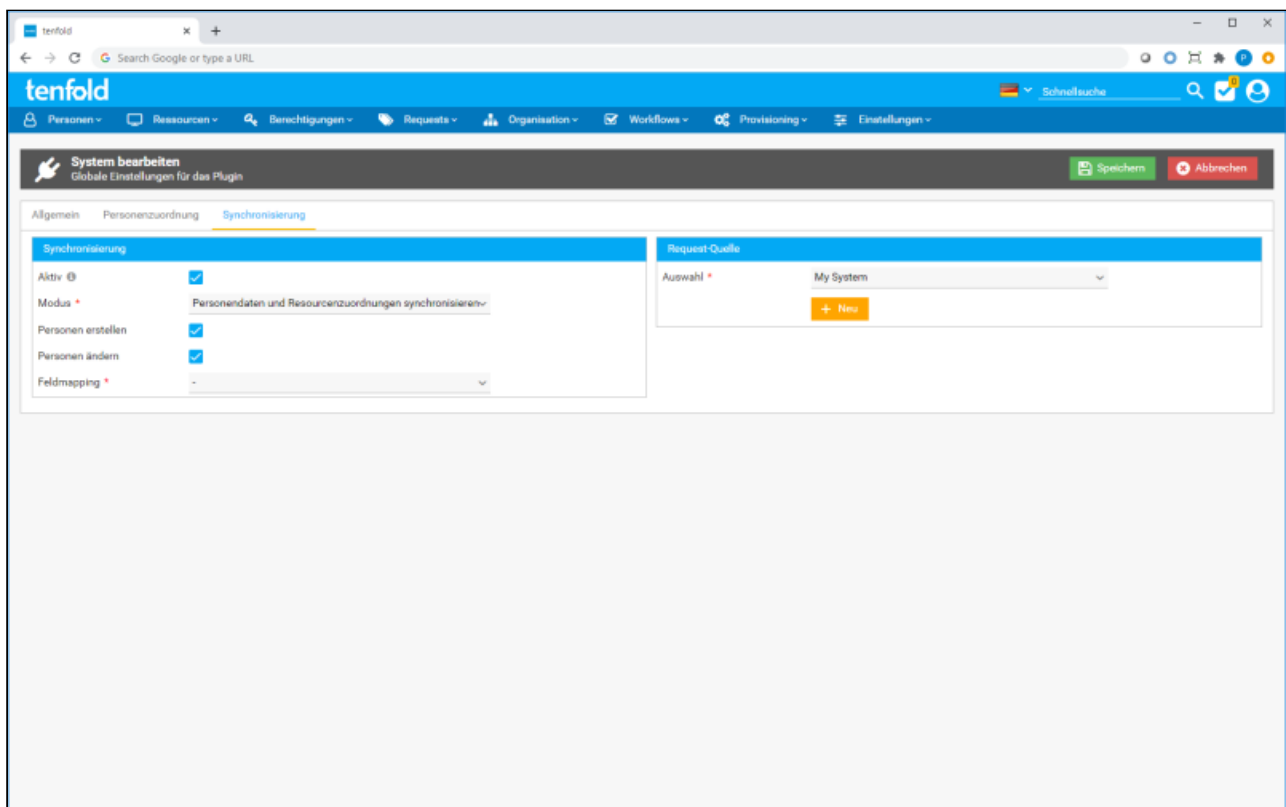
Auf diese Art kann gewährleistet werden, dass eine spätere Änderung der geprüften Felder im System nicht zu einem Verlust der Verknüpfung in tenfold führt. (Beispiel: User und Personen werden über den Benutzernamen zugeordnet. Sollte sich im System der Benutzername ändern, führt dies später nicht dazu, dass die Zuordnung verloren geht, da nur mehr die ID des Systems benutzt wird).

## Synchronisierung

Im Karteireiter "Synchronisierung" können Sie erweiterte Einstellungen zum Synchronisierungsprozess festlegen.

### Zeitpunkt und Intervall

Zeitpunkt bzw. Intervall des Synchronisierungsvorganges lassen sich im Menü Einstellungen > Jobs > Verwaltung treffen. Der Datenabgleich wird vom Job "Generic Connector Plugin - Sync" durchgeführt. Nähere Informationen siehe [Jobs](#)(see page 443).



Folgende Einstellungen lassen sich für den Datenabgleich treffen:

Einstellung	Beschreibung
<b>Synchronisierung</b>	
Aktiv	Mit dieser Einstellung können Sie festlegen, ob der Datenabgleich zwischen dem System und tenfold aktiv ist. Diese Einstellung legt nur fest, ob beim regelmäßigen Datenabgleich die Daten dieses Systems ausgelesen und nach tenfold übertragen werden sollen. Unabhängig von dieser Einstellung, werden Änderungen in tenfold immer an das System übertragen.
Modus	Hier können Sie festlegen, ob nur die Ressourcenzuordnungen und Berechtigungen nach tenfold übertragen werden sollen oder ob auch die Stammdaten aus dem System nach tenfold übertragen werden sollen.
Personen erstellen	Diese Einstellung legt fest ob Personen aus dem System in tenfold angelegt werden sollen, wenn Sie keiner bereits bestehenden Person zugeordnet werden können. Diese Einstellung ist nur aktiv, wenn im Übertragungsmodus eingestellt wurde, dass auch die Stammdaten übertragen werden sollen.
Personen ändern	Diese Einstellung legt fest, ob die Stammdaten von bestehenden Personen, welche aus dem System zugeordnet werden konnten, bei einer Änderung im System, nach tenfold übertragen werden sollen. Diese Einstellung ist nur verfügbar, wenn auch die Stammdaten übermittelt werden sollen.

Feldmapping	Mit dieser Einstellung lässt sich das Feldmapping definieren, welches für den Datenabgleich verwendet werden soll. Da die Datenstruktur des Generic Connector APIs nahezu ident mit den tenfold-Datenstrukturen ist, wird das Feldmapping hauptsächlich dazu benötigt, festzulegen, welche Felder übertragen werden sollen.
<b>Request-Quelle</b>	
Auswahl	Mit dieser Einstellung lässt sich konfigurieren, mit welcher Quelle die Requests angelegt werden sollen, welche durch den Datenabgleich entstehen. Mit der Schaltfläche "Neu" unterhalb dieser Einstellung, lassen sich neue Quellen definieren, sollten Ihnen die bestehenden Quellen nicht genügen.

## Systeme bearbeiten/löschen

Mit der Aktion "Bearbeiten" im Aktionsmenü eines jeden Systems lassen sich die Daten des Systems bearbeiten. Die Einstellungen sind dabei ident zu den Einstellungen bei der Neuanlage. Die verknüpfte Ressource, welche bei der Anlage des Systems erstellt wurde, bleibt nach einer Änderung des Systems unverändert. Sollten Sie den Namen des Systems ändern, wird keine neue Ressource angelegt und die verknüpfte Ressource wird nicht umbenannt. Wenn Sie die Ressource umbenennen wollen, müssen Sie dies manuell über die Ressourcenverwaltung erledigen (siehe [Ressourcenverwaltung](#)(see page 125)). Auch ein Umbenennen der Ressource führt nicht dazu, dass das System umbenannt wird. Während die Verknüpfung bestehen bleibt, können die Daten des Systems und der Ressource nach der Anlage getrennt bearbeitet werden.

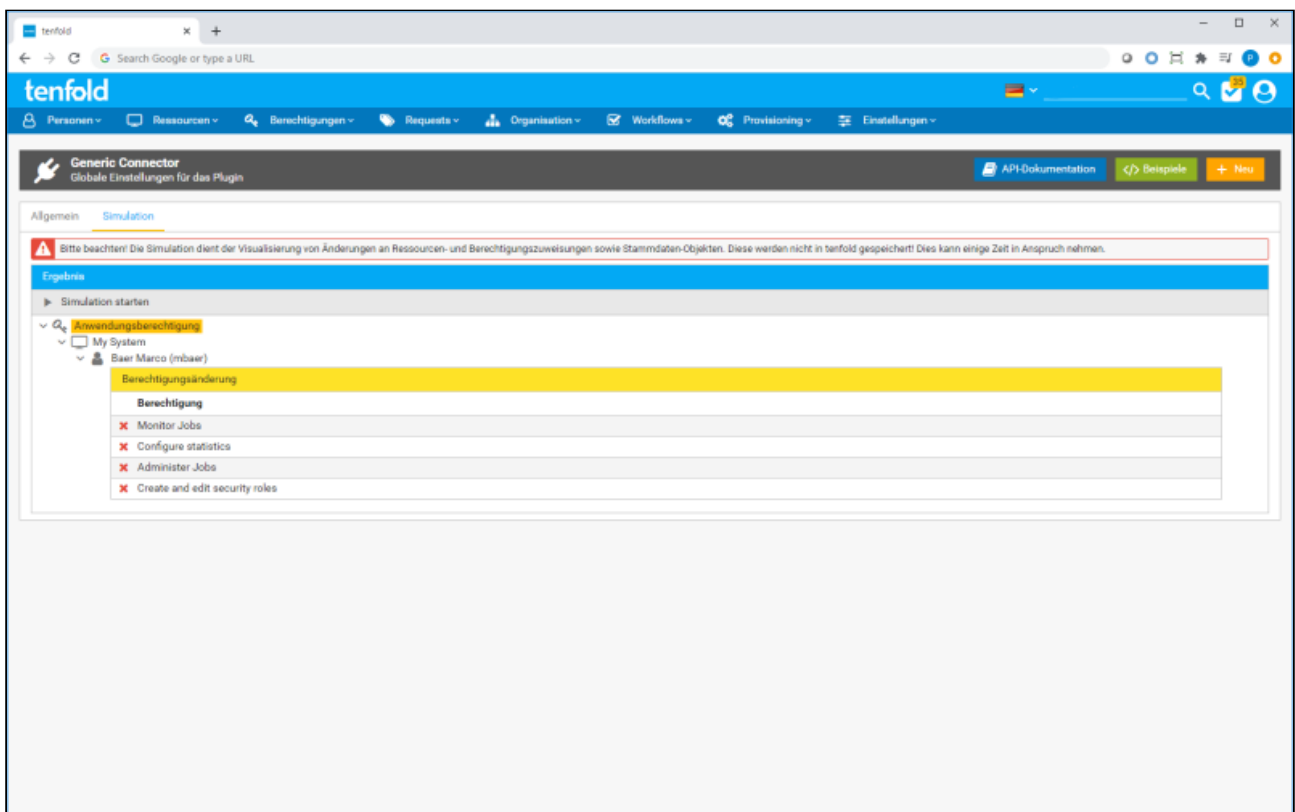
Über den Eintrag "Löschen" im Aktionsmenü lassen sich erstellte Systeme wieder entfernen. Sollten Sie das System löschen, bleibt die zugehörige Ressource weiterhin bestehen und muss manuell in der Ressourcenverwaltung gelöscht werden (siehe [Ressourcenverwaltung](#)(see page 125)).

### Ressource löschen

Sollten Sie die verknüpfte Ressource löschen, so kann keine neue Ressource mit einem noch bestehenden System verknüpft werden. Sie müssen daher das System neu Anlegen, sollten Sie es zu einem späteren Zeitpunkt noch einmal verwenden wollen.

## Simulation

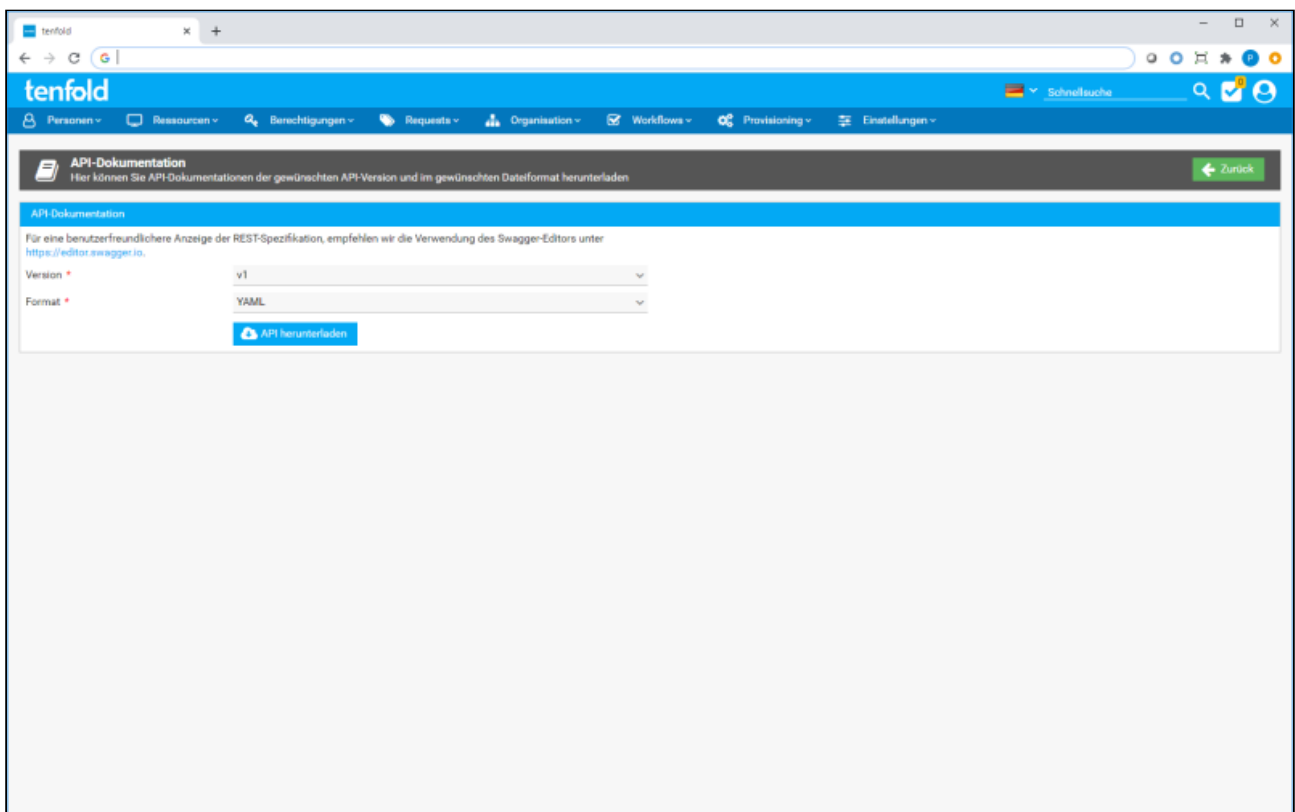
Sie haben im Karteireiter "Simulation" des Hauptschirms der Plugin-Einstellungen die Möglichkeit, sich eine Prognose des nächsten Datenabgleiches liefern zu lassen.



Durch Betätigen der Schaltfläche "Simulation starten" ermittelt tenfold sämtliche Änderungen, die sich anhand des aktuellen Datenbestandes jener Systeme ergeben, für welche die Synchronisation aktiv ist. Dies eignet sich besonders gut dazu, zu prüfen ob geänderte oder neue Einstellungen die gewünschten Auswirkungen auf den Datenbestand von tenfold haben.

## API-Dokumentation

Mit der Schaltfläche "API-Dokumentation" im Kopfbereich des Hauptschirmes gelangen Sie auf einen Schirm auf welchem Sie die API-Dokumentation des Generic Connector REST Services herunterladen können.



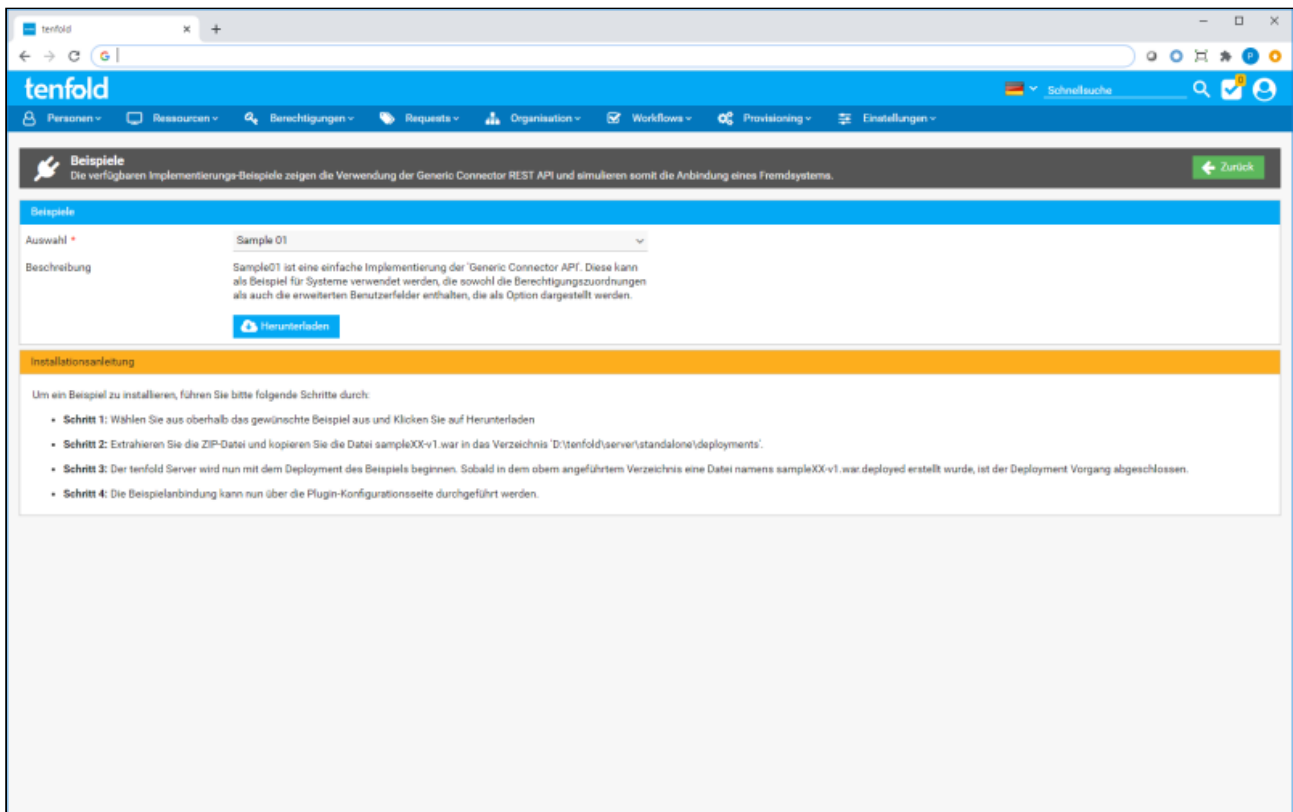
Folgende Einstellungen können Sie zur Erzeugung der API treffen:

Einstellung	Format
Version	Mit dieser Einstellung können Sie festlegen, zu welcher Version Sie die Dokumentation herunterladen können. Aktuell steht nur die Version 1 ("v1") zur Verfügung. Um in Zukunft jedoch Abwärtskompatibel zu bestehenden Services zu bleiben, werden die Änderungen des APIs versioniert. Ältere Versionen können dann an dieser Stelle weiterhin heruntergeladen werden.
Format	Legt fest, ob das API als JSON- oder YAML-Dokument exportiert werden soll.

Nachdem Sie das API-Dokument heruntergeladen haben können Sie dies zum Beispiel benutzen, um sich mit Tools wie <https://editor.swagger.io> eine Übersicht über das API geben zu lassen. Mit diesem Web-Tool lassen sich auch Code-Gerüste für die verschiedensten Technologien erstellen um Services für dieses API entwickeln zu können. Selbstverständlich können Sie auch jedes andere Tool benutzen welches Code-Gerüste aus APIs im Swagger-Format generieren kann.

## Beispiele

Mit der Schaltfläche "Beispiele" im Kopfbereich des Hauptschirmes gelangen Sie auf einen Schirm auf welchem Sie Beispiel-Implementierungen des APIs herunterladen können.



Wählen Sie einfach im Auswahlménü eines der Beispiele und klicken Sie anschließend auf die Schaltfläche "Herunterladen". Sie erhalten daraufhin eine Zip-Datei, welche sowohl eine ausführbare Anwendung enthält, die sich auf dem tenfold-Server deployen lässt, als auch den dazugehörigen Source-Code.

Die Beispielanwendungen wurden mit Java und JAX-RS entwickelt und als .war Datei kompiliert. Die Anwendung lässt sich direkt auf dem tenfold-Server, durch Kopieren in das Verzeichnis <tenfold-Installation>\server\standalone\deployments installieren und starten. Für die Installation auf einem eigenen Java Server, ziehen Sie bitte die Dokumentation Ihrer Server-Software zu Rate.

Die Anwendungen arbeiten mit einer In-Memory-Datenbank. Dies bedeutet, dass die Daten bei jedem Neustart Ihrer Anwendung verloren gehen.

#### Internetverbindung

Bitte beachten Sie, dass die Beispielanwendungen nicht mit tenfold mitgeliefert werden, sondern, wie auch das Plugin selbst, vom tenfold Marketplace bezogen werden müssen. Eine Internetverbindung auf dem tenfold-Server ist daher notwendig, um die Beispielanwendungen herunterladen zu können.

## 14.10.2 API-Dokumentation

Wie im vorangegangenen Abschnitt beschrieben, benötigt das Generic Connector Plugin einen Service, welcher das Generic Connector API implementiert. Nachfolgend finden Sie eine Referenz der verwendeten Datenstrukturen und der zu implementierenden Operationen. Die folgende Dokumentation bezieht sich auf die Version v1 des Generic Connector APIs. Diese Dokumentation richtet sich an Entwickler, mit Kenntnissen über die Entwicklung von REST-APIs.



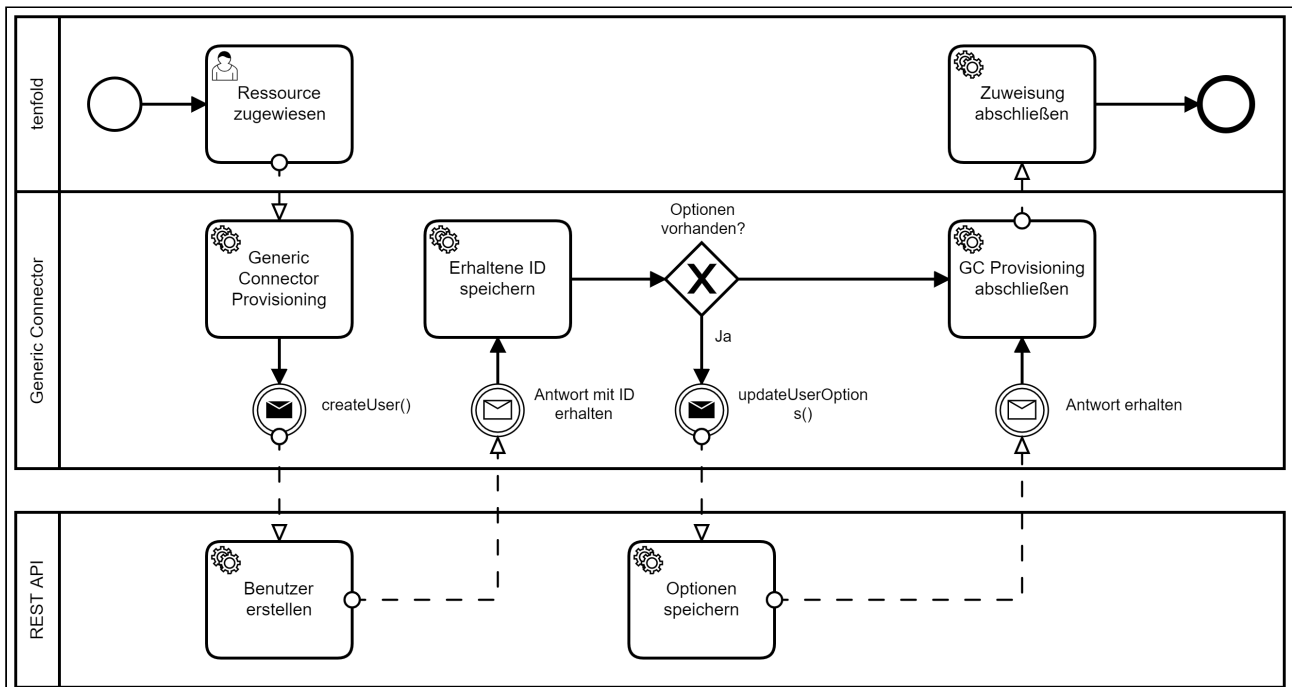
## Prozesse

In den folgenden Abschnitten wird das Zusammenspiel zwischen tenfold, dem Generic Connector und dem REST Api dargestellt. Eine Beschreibung der hier dargestellten Operationen folgt im Abschnitt [Operationen](#) (see page 0).

### Fehlerfälle

In den nachfolgenden Prozessen sind der Einfachheit halber mögliche Fehlerfälle nicht abgebildet. Sollte das API zu irgend einem Zeitpunkt einen Fehler melden, so geht der auslösende Request in tenfold in den Status "Fehlgeschlagen" über. In tenfold kann der Request jederzeit wiederholt werden. In diesem Fall beginnt der Prozess von neuem. Dabei werden möglicherweise bereits erfolgreiche Operationen ebenso wieder ausgeführt.

## Benutzeranlage



Die Anlage eines Benutzers findet in einem Mehrstufigen Prozess statt.

Gestartet wird die Benutzeranlage durch Zuweisung einer der Generic Connector Ressourcen an eine Person in tenfold. tenfold startet daraufhin das Provisioning des Generic Connectors. Dieser ruft zuerst die API Methode `createUser()` auf, welche die kompletten in tenfold vorhandenen Personendaten übergeben bekommt. Diese kann das API nutzen, um sämtliche für das System interessanten Daten zu speichern und muss daraufhin eine eindeutige ID für den Benutzer an den Generic Connector zurückliefern, welcher in der Zukunft dafür verwendet wird, sich auf diesen Benutzer zu beziehen.

Im nächsten Schritt werden, sofern vorhanden, die ausgewählten Optionen der Ressource an die API-Methode `updateUserOptions()` übergeben. Hier wird bereits die zuvor ermittelte ID benutzt, um den Benutzer im API zu referenzieren. Das API kann hier erneut die relevanten Informationen aus den Optionen in der eigenen Datenquelle speichern.

Im Anschluss schließt tenfold das Provisioning der Ressource ab.

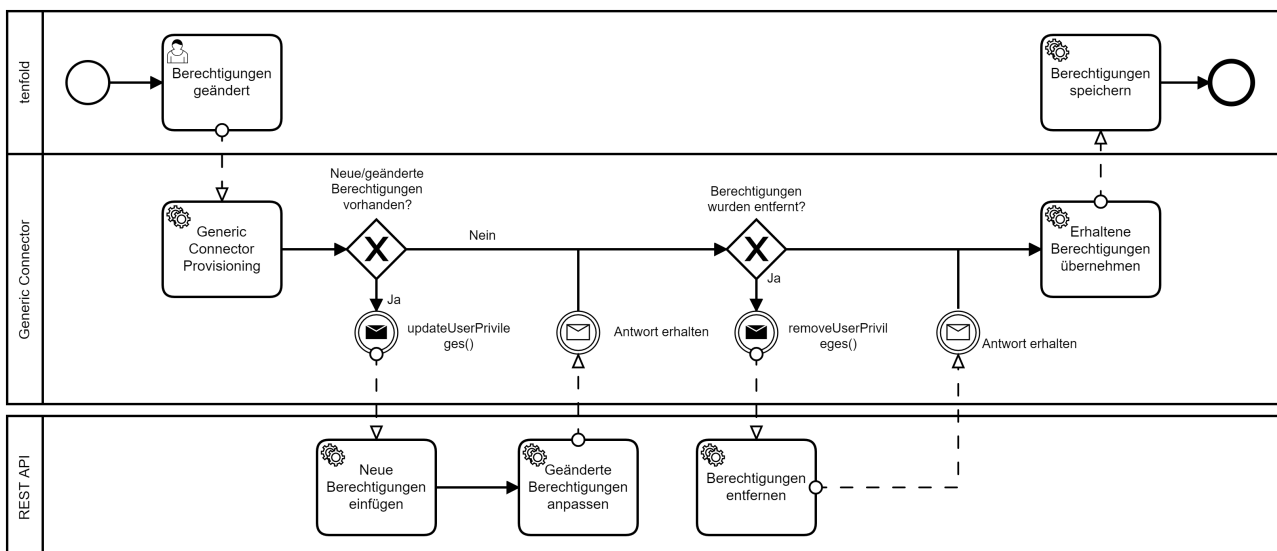
## Optionen

Optionen stellen in tenfold Daten dar, welche nicht direkt der Person zugeordnet sind, sondern der Ressourcenzuordnung der jeweiligen Ressource zu der Person in tenfold. Während die Personendaten immer fest an der Person hängen, können die Daten der Optionen von Zuordnung zu Zuordnung anders sein. Zum Beispiel kann eine Person mehrere Zugänge zur selben Software erhalten. In den Optionen werden daher die Informationen gespeichert, welche von Zugang zu Zugang unterschiedlich sein können, während sich zum Beispiel der Vor- und Nachname der Person nicht unterscheiden wird.

Nähere Informationen zu Berechtigungen und Optionen finden Sie in den Kapiteln [Optionen für Ressourcen](#)(see page 151).

Die Abwicklung der ausgewählten Berechtigungen für den neu angelegten Benutzer finden, ebenso wie spätere Berechtigungsänderung, im gesonderten Prozess "Berechtigungen ändern" statt. Genauerer hierzu finden Sie im folgenden Abschnitt.

## Berechtigungen ändern



Wenn in tenfold die Berechtigungen der Zuordnung einer Generic Connector Ressource hinzugefügt, geändert oder entfernt werden, so ermittelt der Generic Connector die Änderungen und ruft im ersten Schritt die API-Methode `updateUserPrivileges()` auf. An diese Methode werden sämtliche Berechtigungen übergeben, welche entweder hinzugefügt oder verändert wurden. Im Anschluss werden alle entfernten Berechtigungen an die API-Methode `removeUserPrivileges()` übergeben. Beide Methoden werden nur bei Bedarf aufgerufen. Das bedeutet die Methode `updateUserPrivileges()` wird nur aufgerufen wenn es tatsächlich hinzugefügte/geänderte Berechtigungen gibt. Ebenso wird die Methode `removeUserPrivileges()` nur aufgerufen wenn auch Berechtigungen entfernt wurden.

## Berechtigungsänderungen

Mit Berechtigungsänderung ist die Änderung einer Eigenschaft von Berechtigungszuordnungen gemeint. Zum Beispiel kann eine vergebene Berechtigung ein Ablaufdatum erhalten. Das Ändern dieses Ablaufdatums wäre in diesem Kontext dann eine Berechtigungsänderung.

Nachdem die beiden API-Methoden verarbeitet wurden, speichert tenfold die angeforderten Berechtigungen.

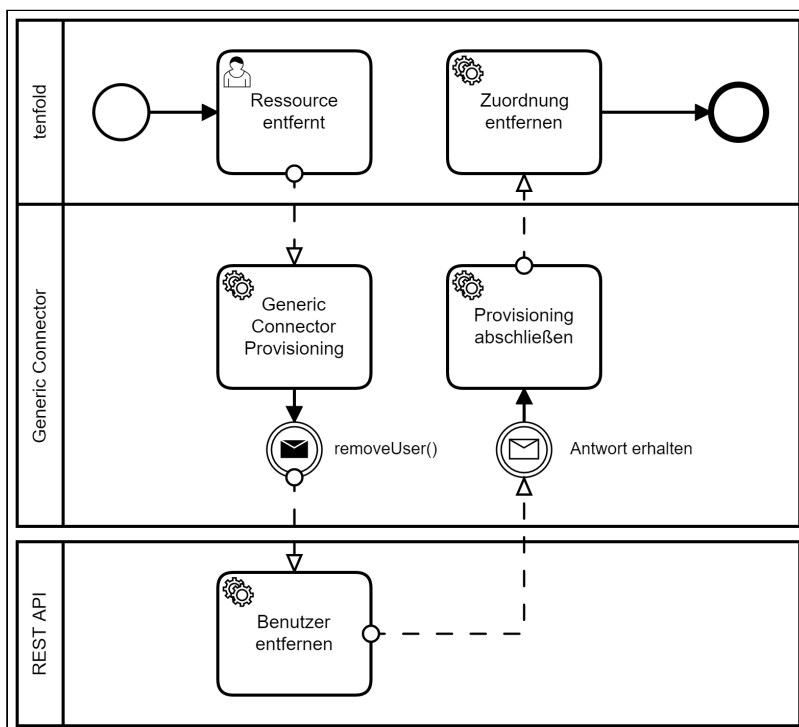
### Vererbte Berechtigungen

Sollte die Vergabe oder der Entzug von Berechtigungen kaskadierende Auswirkungen auf andere Berechtigungen haben, so werden diese in diesem Schritt nicht berücksichtigt, sondern erst mit dem nächsten Datenabgleich.

### Berechtigungen, die keine Berechtigungen sind.

Im Regelfall bilden die Berechtigungen einer Ressourcenzuordnung Zugangsberechtigungen in einem Softwaresystem ab. Dies muss jedoch nicht zwangsläufig der Fall sein. Mit Berechtigungen können theoretisch alle möglichen 1:N Attribute abgebildet werden. Zum Beispiel könnten dies Newsletter-Subscriptions in einer Newsletter-Anwendung sein.

## Benutzer entfernen



Wird eine Generic Connector Ressource einer Person in tenfold entzogen, so ruft der Generic Connector die API-Methode `removeUser()` auf. An dieser Stelle sollten im System sämtliche Vorkehrungen getroffen werden, damit der Benutzer sich nicht mehr am System anmelden kann. Dies muss nicht zwangsläufig bedeuten, dass der Benutzer tatsächlich gelöscht wird.

### Berechtigungen beim Entfernen

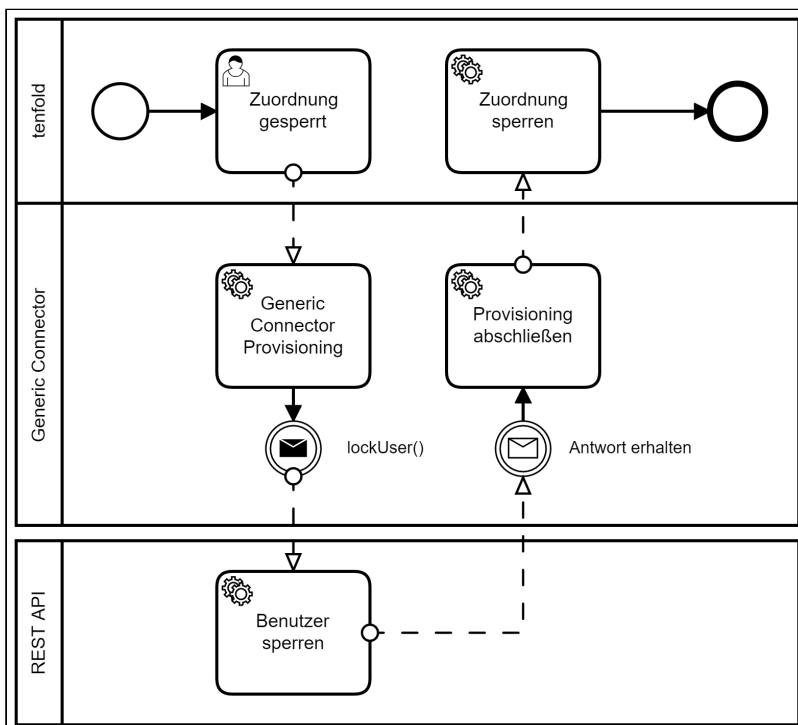
Normalerweise werden beim Entfernen einer Ressourcenzuordnung keine extra Anfragen zum Entfernen der Berechtigungen getätigt. Falls notwendig, können Ressourcen jedoch so konfiguriert werden, dass beim Entfernen auch noch der Prozess "Berechtigung ändern" angestoßen wird. Weitere Details finden Sie unter [Ressourcen](#) (see page 125).

Nach der Entfernung der Ressource geht tenfold davon aus, dass der User im Fremdsystem entfernt wurde, unabhängig davon, ob der Benutzer im System noch existiert oder nicht. Es werden anschließend keine Updates der Benutzerdaten bei Personenänderungen durchgeführt.

### Benutzer neu anlegen

Sollte eine Person in tenfold zu einem späteren Zeitpunkt die Ressource erneut zugeordnet bekommen, geht tenfold davon aus, dass der Benutzer im System nicht mehr existiert und startet erneut den Prozess "Benutzeranlage". Die zuvor ermittelte ID geht dabei verloren und wird nicht wieder übertragen. Sollten im System Benutzer nicht tatsächlich gelöscht werden, liegt es in der Hand des APIs zu ermitteln ob es einen vorhandenen inaktiven Benutzer gibt, der reaktiviert werden muss.

### Benutzer sperren



Wenn man möchte, dass die Anmeldung eines Benutzers (vorübergehend) deaktiviert wird, die Benutzerdaten jedoch nicht entfernt werden sollen, so kann man in tenfold eine Ressourcenzuordnung auch sperren. In diesem Fall ruft der Generic Connector die API-Methode `lockUser()` auf.

### Sperren und Löschen

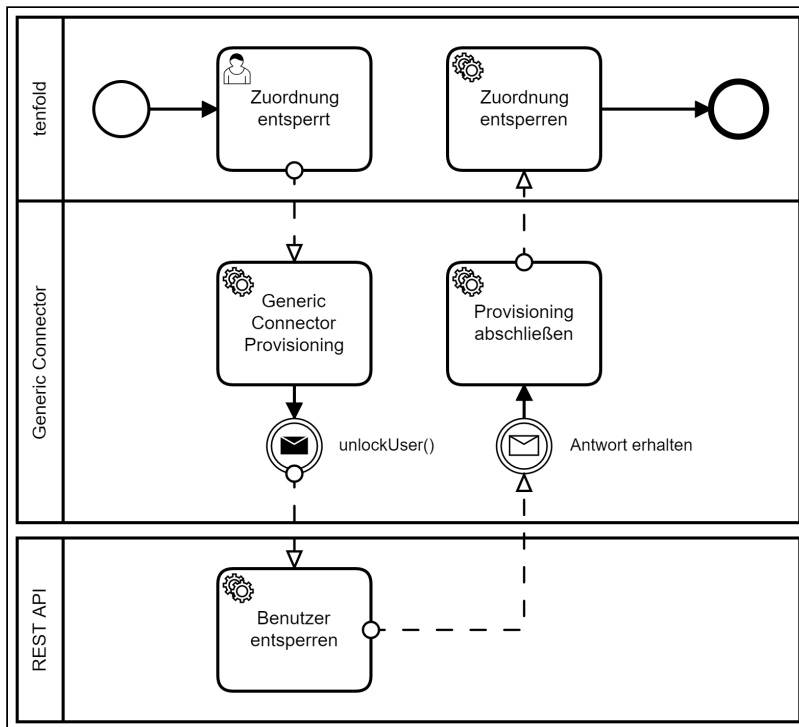
Sowohl beim Sperren als auch beim Löschen einer Zuordnung erwartet tenfold, dass nach dem Aufruf der entsprechenden API-Methoden eine Benutzeranmeldung nicht mehr möglich ist. Der Unterschied besteht darin, dass beim Sperren einer Person, tenfold weiterhin Benutzerdatenupdates bei Personenänderungen sendet und weiterhin über die ID des Benutzers im System verfügt.

### Ressourcenzuordnungen sperren

In den Einstellungen einer Ressource muss explizit freigegeben werden, dass diese sperrbar ist. Bei der Anlage einer Ressource ist dieser Haken nicht standardmäßig gesetzt. Nähere Informationen hierzu finden Sie unter [Ressourcenverwaltung](#)(see page 125).

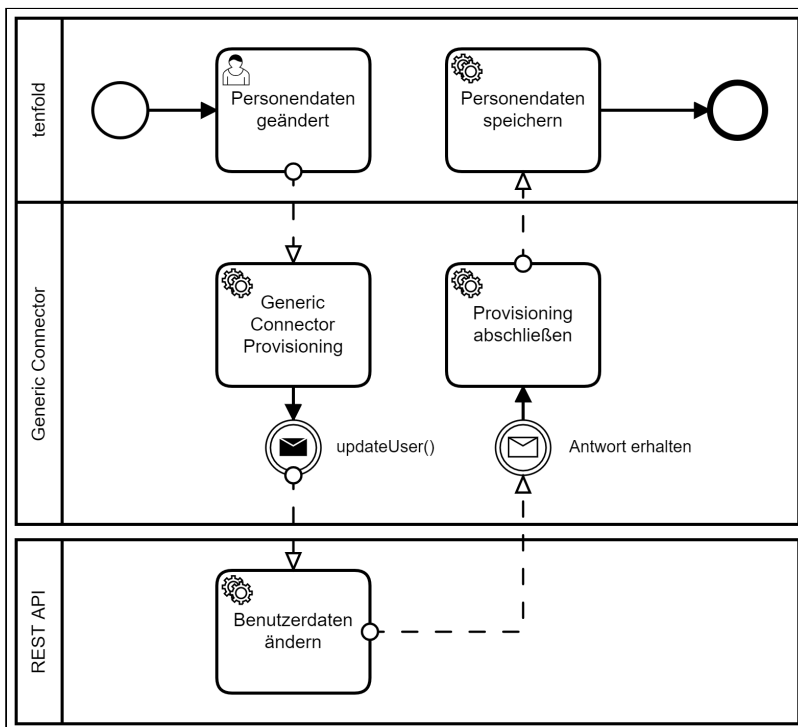
Wenn ein Benutzer gesperrt wurde, kann er durch den im nachfolgenden Abschnitt beschriebenen Prozess "Benutzer entsperren" wieder reaktiviert werden.

### Benutzer entsperren



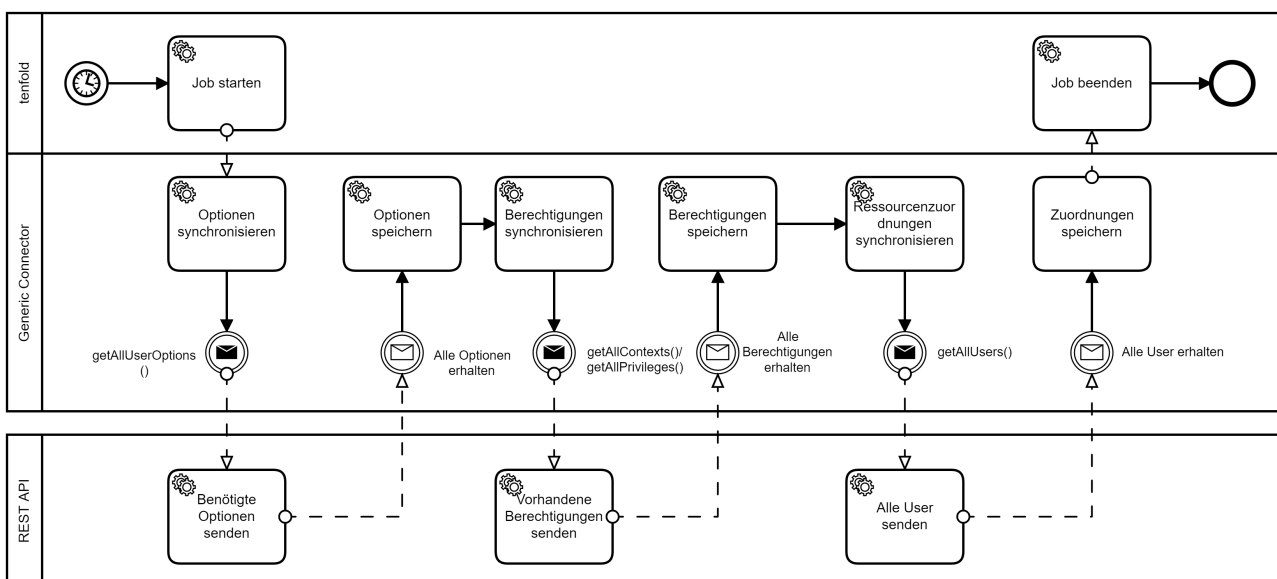
Sollte eine Ressourcenzuordnung gesperrt worden sein, lässt sie sich in tenfold einfach wieder entsperren. Der Generic Connector ruft hierbei einfach die API-Methode `unlockUser()` auf. Nach dem Aufruf dieser Methode sollte der Benutzer wieder normal eine Anmeldung durchführen können.

## Personendatenänderung



Werden in tenfold die Daten einer Person geändert, welche eine Generic Connector Ressource zugeordnet hat, so ruft der Generic Connector die updateUser() Methode des APIs auf. Damit soll sichergestellt werden, dass die Personendaten in tenfold und die Benutzerdaten im System gleich sind. Dieser Methode wird die ID übergeben, welche bei der Personenanlage übermittelt wurde. tenfold sendet hierbei den gesamten Personendatensatz an das API. Es liegt hierbei am API zu entscheiden welche Daten relevant sind oder nicht.

## Datenabgleich



In konfigurierbaren Zeitintervallen startet tenfold einen Datenabgleich zwischen dem System und den eigenen Datenbeständen. Hierbei wird folgendermaßen vorgegangen:

Zuerst ermittelt tenfold alle Generic Connector Ressourcen. Für jede dieser Ressourcen wird zuerst die API-Methode `getAllUserOptions()` aufgerufen. Anhand der erhaltenen Daten werden die vorhandenen Optionen der Ressource angepasst.

Anschließend ermittelt tenfold alle Berechtigungen des Systems mithilfe der beiden API-Methoden `getAllContexts()` und `getAllPrivileges()`. Diese ermitteln zuerst alle verfügbaren Kontexte für Berechtigungen und anschließend die sich in den Kontexten befindlichen Berechtigungen.

Zuletzt ruft der Generic Connector noch die API-Methode `getAllUsers()` auf, welche alle Daten der im System bestehenden User zurückliefert. Diese Daten beinhalten sowohl die dort vorhandenen Benutzerdaten als auch die Berechtigungen bzw Optionen. Mithilfe dieser Daten gleich tenfold in jedem Fall ab ob und wer die entsprechenden Ressourcen zugeordnet hat, als auch die dazugehörigen Berechtigungen und Optionen. Optional gleicht tenfold an dieser Stelle auch die gewünschten Benutzerdaten ab.

## Endpoints

Folgende Operationen müssen vom API des Systems implementiert werden. Zu jeder Operation folgt eine kurze Beschreibung sowie die Eckdaten des Aufrufes. Für eine Beschreibung der entsprechenden Datentypen konsultieren Sie bitte den nachfolgenden Abschnitt.

Folgende allgemeine Fehlercodes können von jeder Operation verwendet werden und werden daher nicht explizit bei jeder Operation aufgeführt.

HTTP Code	Bedeutung	Erwartete Ausgabe
400	Die Anfrage enthielt ungültige Daten	keine
401	Nicht autorisiert	keine
403	Darf vom angemeldeten Benutzer nicht durchgeführt werden	keine
500	Interner Fehler	<i>apiError</i>

Alle Datenobjekte welche an das API gesendet werden und als Rückgabe erwartet werden, sind im JSON-Format auszuführen.

## Benutzer auflisten

GET	/users	
Parameter		
keine		
Antworten		
Code	Daten	Beschreibung
200	user[]	Eine Liste aller user-Objekte im System

Diese Operation listet alle bestehenden User auf, welche von tenfold übernommen werden sollen. tenfold benutzt diese Operation, um:

- Personen in tenfold die Ressourcen des Systems zuzuordnen und entsprechende Berechtigungen zu setzen.
- Die Stammdaten einer Person in tenfold zu aktualisieren.

**Abgleich von Stammdaten**

Für den Abgleich der Stammdaten muss ein entsprechendes Feldmapping in der Konfiguration des APIs im Generic Connector hinterlegt werden (siehe [Synchronisierung](#)(see page 735)). Es werden nur jene Felder in tenfold beim Datenabgleich aktualisiert, welche im eingerichteten Feldmapping vorkommen. Dennoch können Sie unabhängig davon sämtliche Daten liefern, welche Ihrem System bekannt sind. Anschließend lassen sich die übertragenen Attribute einfach über das Feldmapping, an einer zentralen Stelle, steuern, ohne später das API anpassen zu müssen.

**Einzelnen User mit ID ermitteln**

GET	/users/{id}	
Parameter		
name	Ort	Beschreibung
id	Pfad	Die Id des zu ermittelnden <i>user</i> -Objektes
Antworten		
Code	Daten	Beschreibung
200	user	Ein user mit entsprechender Id wurde gefunden und wird zurückgeliefert.
404	-	Ein Benutzer mit der geforderten Id existiert nicht.

Dieser Endpoint findet ein einzelnes User-Objekt mittels der ID des Objektes. Aktuell wird diese Operation seitens tenfold nicht verwendet und ist vorerst nur der Vollständigkeit halber im API enthalten. Es wird jedoch empfohlen sie dennoch zu implementieren, da es in Zukunft möglicherweise gebraucht werden könnte.

**Neuen Benutzer anlegen**

POST	/users	
Parameter		
name	Ort	Beschreibung
user	Body: user	Ein Objekt vom Typ <i>user</i> , welcher alle Stammdaten des anzulegenden Benutzers enthält. Dieses Objekt enthält kein <i>id</i> -Attribut.
Antworten		
Code	Daten	Beschreibung
201	user	Ein <i>user</i> -Objekt mit den Daten des angelegten Benutzers. In diesem Objekt sollte das <i>id</i> -Attribut gesetzt sein.

Diese Operation legt einen neuen Benutzer im System an. Dieser Operation werden nur die Stammdaten der Person aus tenfold übergeben. Die Werte von Optionen und Berechtigungen werden in separaten Endpoints abgehandelt. Das zurückgelieferte *user*-Objekt sollte die Daten enthalten die auch an den Endpoint



übergeben wurden. Wichtig ist jedoch, dass im Gegensatz zum übergebenen Objekt, im zurückgelieferten Objekt das *id*-Attribut enthalten sein muss. Der Generic Connector speichert diese *id*, bei der Ressourcenzuordnung in tenfold ab und benutzt diese für zukünftige Referenzen (Updates, Löschen, etc) auf diesen Benutzer.

Der Rest der zurückgelieferten Daten wird aktuell vom Generic Connector nicht verwendet. Es ist jedoch eine gute Praxis dennoch die korrekten Daten zurückzuliefern.

#### Datenänderung

Sollten Sie andere Daten im *user*-Objekt zurückliefern als Sie erhalten, führt der Generic Connector dennoch keine Personenänderung in tenfold durch. Sollten Sie jedoch zum Beispiel eine Telefonanlage oder Ähnliches anbinden wollen, welche bei der Anlage der Person in tenfold eine Durchwahl (*phone*-Attribute) zuweisen möchte, so muss dies mit dem periodischen Datenabgleich des Generic Connectors geschehen (siehe [Synchronisierung](#)(see page 735)).

#### Übergebene Daten

Der Generic Connector sendet immer den vollständigen Datensatz der Person aus tenfold an das API. Das API muss nicht sämtliche dieser Daten speichern nur um diese bei Abfragen zurückliefern zu können. Es genügt vollkommen nur jene Daten zu verarbeiten welche für das System relevant sind.

### Vorhandenen Benutzer aktualisieren

PUT	/users/{id}	
Parameter		
name	Ort	Beschreibung
id	Pfad	Die Id des zu aktualisierenden Benutzers. Dies ist die id, welche an tenfold in der Operation "Neuen Benutzer anlegen" zurückgeliefert wurde.
user	Body: user	Ein Objekt vom Typ user, welcher alle Stammdaten des anzulegenden Benutzers enthält.
Antworten		
Code	Daten	Beschreibung
200	user	Ein user-Objekt mit den Daten des aktualisierten Benutzers.
404	-	Wird gesendet, falls ein Benutzerobjekt mit der angegebenen Id nicht gefunden werden konnte.

Dieser Endpoint aktualisiert ein Benutzerobjekt im System. Er muss ein Benutzerobjekt mit der angegebenen Id ermitteln und anschließend alle relevanten Daten, welche übergeben wurden im System updaten.

#### Übergebene Daten

Der Generic Connector sendet immer den vollständigen Datensatz der Person aus tenfold an das API. Das API muss nicht sämtliche dieser Daten speichern nur um diese bei Abfragen zurückliefern zu können. Es genügt vollkommen nur jene Daten zu verarbeiten welche für das System relevant sind.

## Benutzer entfernen

DELETE	/users/{id}	
Parameter		
name	Ort	Beschreibung
id	Pfad	Die Id des zu aktualisierenden Benutzers. Dies ist die id, welche an tenfold in der Operation "Neuen Benutzer anlegen" zurückgeliefert wurde.
Antworten		
Code	Daten	Beschreibung
204	-	Gibt an, dass der Benutzer erfolgreich entfernt wurde.
404	-	Wird gesendet, falls ein Benutzerobjekt mit der angegebenen Id nicht gefunden werden konnte.

Der Endpoint entfernt einen Benutzer mit der angegebenen Id aus dem System. Wie dieser Vorgang genau implementiert wird, hängt vom System ab. Folgende Kriterien sollten jedoch erfüllt sein:

- Der Benutzer wird vom Datenabgleich nicht mehr mitgeliefert.
- Mögliche Anmeldungen am System des Benutzers sollten deaktiviert sein.

### Intention des Löschens

Wird einer Person in tenfold eine Ressource entzogen, so geht tenfold davon aus, dass der Benutzer aus dem System entfernt wird. Genauso als würde man den Benutzer durch eine Anwendung/ Interaktion des Systems löschen. Unabhängig davon ob der Datensatz des Benutzers tatsächlich gelöscht wurde oder nicht, das System sollte sich im Anschluss zumindest so verhalten, als würde der Benutzer nicht mehr existieren.

## Berechtigungen hinzufügen/aktualisieren

PUT	/users/{id}/privileges	
Parameter		
name	Ort	Beschreibung
id	Pfad	Die Id des zu aktualisierenden Benutzers. Dies ist die id, welche an tenfold in der Operation "Neuen Benutzer anlegen" zurückgeliefert wurde.

privileges	Body: <i>privilegeAssignment[]</i>	Eine Liste von Berechtigungen in Form von <i>privilegeAssignment</i> -Objekten.
<b>Antworten</b>		
<b>Code</b>	<b>Daten</b>	<b>Beschreibung</b>
200	<i>user</i>	Die Berechtigungen wurden erfolgreich aktualisiert und die Daten des Benutzers werden zurückgeliefert.
404	-	Wird gesendet, falls ein Benutzerobjekt mit der angegebenen Id nicht gefunden werden konnte.

Diese Methode fügt neue Berechtigungen hinzu oder aktualisiert erweiterte Optionen von bestehenden Berechtigungszuordnungen. Sollte der Kontext der Berechtigung keine erweiterten Optionen definieren, so werden an dieser Stelle immer nur neu hinzugefügte Berechtigungen übergeben werden. Sollte der Kontext über erweiterte Optionen verfügen, so können hier auch zu bestehenden Berechtigungszuordnungen geänderte Optionen übertragen werden. In diesem Fall kann die Unterscheidung am Vorhandensein des *id*-Attributes im jeweiligen *privilegeAssignment*-Objekt festgestellt werden.

#### Datenabgleich

Es existiert keine getrennte Operation, um die Berechtigungen einzeln auszulesen. Die gesetzten Berechtigungen müssen jedoch im Datenabgleich, welcher mit der Operation "GET /users" durchgeführt wird, widerspiegelt sein. Andernfalls wird der nächste Datenabgleich die geänderten Berechtigungen in tenfold wieder umschreiben.

## Berechtigungen entfernen

DELETE	/users/{id}/privileges	
Parameter		
name	Ort	Beschreibung
id	Pfad	Die Id des zu aktualisierenden Benutzers. Dies ist die id, welche an tenfold in der Operation "Neuen Benutzer anlegen" zurückgeliefert wurde.
privileges	Body: privilegeAssi gnment[]	Eine Liste von Berechtigungen in Form von privilegeAssignment-Objekten.
Antworten		
Code	Daten	Beschreibung
200	user	Die Berechtigungen wurden erfolgreich entfernt und die Daten des Benutzers werden zurückgeliefert.

404	-	Wird gesendet, falls ein Benutzerobjekt mit der angegebenen Id nicht gefunden werden konnte.
-----	---	--

Diese Operation ist dafür zuständig, Berechtigungen welche der Benutzer im System hat, wieder zu entfernen. Sollte der Kontext der jeweiligen Berechtigung über erweiterte Optionen verfügen, so wird im entsprechenden *privilegeAssignment*-Objekt eine id mitgeliefert um zwischen verschiedenen Zuordnungen derselben Berechtigung mit unterschiedlichen Optionen unterscheiden zu können.

Ist kein *id*-Attribut gesetzt, so wurden beim Kontext keine erweiterten Optionen eingerichtet. Daher geht tenfold davon aus, dass ein Benutzer einfach eine Berechtigung hat oder nicht. Es genügt aus der Sicht von tenfold daher zu sagen, dass eine bestimmte Berechtigung entfernt werden muss. Existieren erweiterte Optionen beim Kontext, so geht tenfold davon aus, dass die Berechtigung mehrfach mit unterschiedlichen Einstellungen zugeordnet werden kann. Daher ist in diesem Falle auch eine Id notwendig, welche der Generic Connector mitliefert.

Ob und wie die Berechtigungen tatsächlich entfernt werden, spielt für den Generic Connector keine Rolle. Es muss nur sichergestellt sein, dass beim Datenabgleich die entfernten Berechtigungen auch nicht mehr mitgeliefert werden.

#### Datenabgleich

Es existiert keine getrennte Operation um die Berechtigungen einzeln auszulesen. Die gesetzten Berechtigungen müssen jedoch im Datenabgleich, welcher mit der Operation "GET /users" durchgeführt wird, widerspiegelt sein. Andernfalls wird der nächste Datenabgleich die geänderten Berechtigungen in tenfold wieder umschreiben.

### Optionen aktualisieren

PUT	/users/{id}/options	
Parameter		
name	Ort	Beschreibung
id	Pfad	Die Id des zu aktualisierenden Benutzers. Dies ist die id, welche an tenfold in der Operation "Neuen Benutzer anlegen" zurückgeliefert wurde.
privileges	Body: userOptionAssignment[]	Eine Liste von Optionen in Form von userOptionAssignment-Objekten.
Antworten		
Code	Daten	Beschreibung
200	user	Die Berechtigungen wurden erfolgreich entfernt und die Daten des Benutzers werden zurückgeliefert.
404	-	Wird gesendet, falls ein Benutzerobjekt mit der angegebenen Id nicht gefunden werden konnte.

Da seitens tenfold davon ausgegangen werden muss, dass für eine Person in einem System mehrere Benutzer existieren könnten, ist es notwendig gewisse Datenbestände nicht an der Person zu speichern,

sondern an der entsprechenden Ressourcenzuordnung in tenfold. Dieser Endpoint wird vom Generic Connector aufgerufen, um diese Datenbestände zu aktualisieren.

Eine Person kann zum Beispiel in einem System für Zutrittskontrollen mehrere Schlüsselkarten haben. Jede Schlüsselkarte kann hierbei ein eigenes Ablaufdatum haben (Zutrittskarten zu Hochsicherheitsbereichen müssten hier eventuell einmal im Monat neu ausgestellt werden). Eine Person in tenfold hat hierbei jedoch nur einen Stammdatensatz und jedes Feld ist immer nur einmal vorhanden. Um diese Einschränkung zu umgehen, können mittels Optionen die Daten, wie erwähnt, neben der Zuordnung gespeichert werden, statt bei der Person.

Auch können Optionen benutzt werden, um Datenbestände abzubilden, welche im Personenstammdatensatz von tenfold keine Entsprechung finden.

Hierbei spielt es keine Rolle, ob die Datenbestände im System direkt am Benutzer gespeichert werden oder getrennt vom Benutzer, so wie sie in tenfold getrennt von der Person gespeichert werden. Das API hat hierbei die Aufgabe die Daten des Systems korrekt auf die einzelnen Optionen zu übertragen.

#### Datenabgleich

Es existiert keine getrennte Operation um die Optionen einzeln auszulesen. Die gesetzten Optionen müssen jedoch im Datenabgleich, welcher mit der Operation "GET /users" durchgeführt wird, widerspiegelt sein. Andernfalls wird der nächste Datenabgleich die geänderten Optionen in tenfold wieder umschreiben.

### Passwort setzen

PUT	/users/{id}/password	
Parameter		
name	Ort	Beschreibung
id	Pfad	Die Id des zu aktualisierenden Benutzers. Dies ist die id, welche an tenfold in der Operation "Neuen Benutzer anlegen" zurückgeliefert wurde.
password	Body: string	Das neue Passwort.
Antworten		
Code	Daten	Beschreibung
200	user	Das Passwort wurde erfolgreich gesetzt und die Daten des Benutzers werden zurückgeliefert.
404	-	Wird gesendet, falls ein Benutzerobjekt mit der angegebenen Id nicht gefunden werden konnte.

Diese Operation setzt ein neues Passwort für den Benutzer im System. Der Generic Connector ruft diese Operation auf, wenn in tenfold die Passwort-Reset Funktionen für eine Person benutzt werden. Sollte das API keinen Passwort-Reset unterstützen, so muss bei der Ressource in tenfold die Option "Passwortänderung möglich" deaktiviert sein (siehe [Ressourcenverwaltung](#)(see page 125)).

## Benutzer sperren

PUT	/users/{id}/lock	
Parameter		
name	Ort	Beschreibung
id	Pfad	Die Id des zu aktualisierenden Benutzers. Dies ist die id, welche an tenfold in der Operation "Neuen Benutzer anlegen" zurückgeliefert wurde.
Antworten		
Code	Daten	Beschreibung
200	user	Der Benutzer wurde erfolgreich gesperrt und die Daten des Benutzers werden zurückgeliefert.
404	-	Wird gesendet, falls ein Benutzerobjekt mit der angegebenen Id nicht gefunden werden konnte.

Diese Operation wird vom Generic Connector aufgerufen, um einen Benutzer im System zu sperren. Die Intention dahinter ist, dass der Benutzer nach diesem Zeitpunkt keine Anmeldung am System mehr durchführen kann. Vergleichbar ist dies mit dem Deaktivieren eines Benutzerkontos im Active Directory. Der Benutzer ist nach wie vor vorhanden, und die Benutzerdaten sind einseh- und änderbar. Eine Anmeldung mit diesem Konto ist dann allerdings nicht mehr möglich.

Sollte Ihr System und/oder API eine vorübergehende Deaktivierung nicht unterstützen, so sollte die entsprechende Generic Connector Ressource als nicht sperrbar konfiguriert werden (siehe [Ressourcenverwaltung\(see page 125\)](#)).

## Benutzer entsperren

PUT	/users/{id}/unlock	
Parameter		
name	Ort	Beschreibung
id	Pfad	Die Id des zu aktualisierenden Benutzers. Dies ist die id, welche an tenfold in der Operation "Neuen Benutzer anlegen" zurückgeliefert wurde.
Antworten		
Code	Daten	Beschreibung
200	user	Der Benutzer wurde erfolgreich entsperrt und die Daten des Benutzers werden zurückgeliefert.
404	-	Wird gesendet, falls ein Benutzerobjekt mit der angegebenen Id nicht gefunden werden konnte.

Diese Operation wird vom Generic Connector aufgerufen um einen Benutzer im System zu entsperren. Dies ist die entgegengesetzte Operation zum Sperren aus dem vorhergehenden Abschnitt. Nach dem Aufruf dieses Endpoints sollte eine Anmeldung für den Benutzer wieder möglich sein.

Sollte Ihr System und/oder API eine vorübergehende Deaktivierung nicht unterstützen, so sollte die entsprechende Generic Connector Ressource als nicht sperrbar konfiguriert werden (siehe

[Ressourcenverwaltung](#)([see page 125](#))).

### Mögliche Optionen auflisten

GET	/users/options	
Parameter		
name	Ort	Beschreibung
Keine		
Antworten		
Code	Daten	Beschreibung
200	userOption[]	Eine Liste aller möglichen Optionen wird zurückgeliefert

Dieser Endpoint liefert eine Liste von allen Optionen für Benutzer zurück. Der Generic Connector benutzt diese Liste, um die entsprechende Ressource zu konfigurieren, so dass diese anschließend die Eingabe der aufgelisteten Optionen, bei der Zuordnung oder danach, zulässt. Sollte Ihr System oder die Schnittstelle keine Optionen benötigen, so sollte diese Operation dennoch implementiert werden, eine leere Liste zu liefern, statt nicht zu existieren.

#### IDs und Namen

Die Ids, welche die einzelnen `userOption`-Objekte enthält, werden für zukünftige Abgleiche und Aktualisierungen herangezogen. Eine Id ist daher zwingend erforderlich. Sollten Optionen in Ihrem System keine Id haben sondern Beispielsweise über den Namen identifiziert werden, sollten Sie einfach den Namen auch im id-Attribut liefern. In tenfold wird der Name nur zur Anzeige auf der Oberfläche verwendet, nicht jedoch für Abgleiche oder Ähnliches.

### Anmeldung prüfen

GET	/login/{username}	
Parameter		
name	Ort	Beschreibung
username	Pfad	Der Benutzername des anzumeldenden Benutzers.
password	Body: string	Das zu prüfende Passwort.
Antworten		

Code	Daten	Beschreibung
200	bool	Der Benutzer wurde gefunden und das Passwort wurde akzeptiert (true) oder der Benutzer wurde gefunden, aber die Anmeldung konnte mit dem angegebenen Passwort nicht durchgeführt werden (false).
404	-	Wird gesendet, falls ein Benutzerobjekt mit dem angegebenen Benutzernamen nicht gefunden werden konnte.

Diese Operation wird vom Generic Connector aktuell nicht verwendet. Die Operation wurde vorsorglich in das API aufgenommen, um für die Zukunft die Möglichkeit zu bieten, den Generic Connector als alternative zu Active Directory für die Anmeldung in tenfold zu verwenden.

### Mögliche Berechtigungen auflisten

GET	/privileges	
Parameter		
name	Ort	Beschreibung
Keine		
Antworten		
Code	Daten	Beschreibung
200	privilege[]	Eine Liste aller möglichen Berechtigungen wird zurückgeliefert

Dieser Endpoint wird vom Generic Connector benutzt, um die zugehörigen Ressourcen zu konfigurieren. Dieser Endpoint sollte alle Berechtigungen zurückliefern, welche über tenfold vergeben werden sollen. Diese Auflistung ist unabhängig von den einzelnen Benutzern, sondern eine Auflistung aller verfügbaren Berechtigungen. Sollte Ihr System nicht über Berechtigungen verfügen, so sollte diese Operation trotzdem implementiert werden und eine leere Liste liefern.

#### IDs und Namen

Die Id, welche die einzelnen *privilege*-Objekte enthalten, wird für zukünftige Abgleiche und Aktualisierungen herangezogen. Eine Id ist daher zwingend erforderlich. Sollten Berechtigungen in Ihrem System keine Id haben sondern Beispielsweise über den Namen identifiziert werden, sollten Sie einfach den Namen auch im id-Attribut liefern. In tenfold wird der Name nur zur Anzeige auf der Oberfläche verwendet, nicht jedoch für Abgleiche oder Ähnliches.

Die möglichen erweiterten Optionen, welche für Berechtigungen möglich sind, werden nicht an dieser Stelle definiert, sondern bei den Kontexten, welche von der folgenden Operation behandelt werden.

### Kontexte auflisten

<b>GET</b>	<b>/contexts</b>
<b>Parameter</b>	



name	Ort	Beschreibung
Keine		
<b>Antworten</b>		
Code	Daten	Beschreibung
200	<code>context[]</code>	Eine Liste aller möglichen Kontexte wird zurückgeliefert

Diese Operation liefert eine Liste aller im System vorhandenen Kontexte für Berechtigungen. Jede Berechtigung hat immer einen zugehörigen Kontext. Dies dient dazu um Systeme abzubilden in welcher einzelne Berechtigungen in verschiedenen Bereichen vergeben werden können. Beispielsweise kann es die Berechtigungen "Lesen" und "Schreiben" für die Kontexte "Lager", "Rechnungen" oder Ähnliches geben. Auf der tenfold-Oberfläche werden die Berechtigungen optisch getrennt nach Kontext dargestellt. Damit lassen sich in Systemen mit vielen Berechtigungen zur besseren User-Experience die Berechtigungen auch kategorisieren, selbst wenn die eigentliche Berechtigungslogik des Systems keine unterschiedlichen Berechtigungsbereiche definiert.

Zuletzt bietet jeder Kontext noch an, eine Reihe an Optionen zu definieren, welche bei den Berechtigungenszuordnungen der im Kontext enthaltenen Berechtigungen, angegeben werden können. Sollten erweiterte Optionen definiert werden oder das Attribut "validityEditable" true sein, so benutzt tenfold bei der Bearbeitung der Berechtigungen einen erweiterten Eingabemodus, statt einer einfachen Checkboxliste, in welcher die hier definierten Optionen eingetragen werden können. Diese Einstellungen sind über die Benutzeroberfläche von tenfold nicht einstellbar und müssen über das API geliefert werden. Für eine Ressource in tenfold gibt es immer einen Default-Kontext. Ihr API muss dies respektieren indem es immer zumindest einen Kontext zurückliefert, egal ob er verwendet wird oder nicht. Die Berechtigungen müssen sich in diesem Fall auch immer auf diesen Kontext beziehen.

## Datenmodell

Im folgenden wird das Datenmodell des APIs beschrieben. Auf diesem Datenmodell bauen die Operationen auf, welche zu implementieren sind.

### User

Ein User-Objekt repräsentiert einen Benutzer oder Account im System. [Dies ist das zentrale Datenobjekt, mit welchem tenfold Benutzerdaten aus dem System ausliest und für das Anlegen oder Bearbeiten von Benutzern an das API übermittelt.](#) Bei den meisten der Felder handelt es sich um Stammdaten, die zur Synchronisierung mit Personenstammdaten in tenfold verwendet werden können. Wenn das System beispielsweise eine Telefonanlage ist, so kann das System die Telefonnummern der jeweiligen User an tenfold übermitteln, um diese in tenfold aktuell gemäß der Telefonanlage zu halten. Dieses Objekt wird auch von tenfold verwendet, um Änderungen an das System zu übermitteln. Also wenn ein neuer Benutzer im System angelegt werden soll, oder die Daten eines bestehenden Users angepasst werden sollen. Die meisten dieser Felder haben eine 1 zu 1 Entsprechung im Personenstammdatensatz von tenfold.

#### Flexible Stammdaten

Die tatsächlich verwendeten Felder, sind für jede Personenart frei konfigurierbar (siehe [Personenarten\(see page 81\)](#)). Je nach der Konfiguration, können daher einige der unten angeführten Felder nicht befüllt sein.

### Senden und Empfangen von Daten

Sendet der Generic Connector die Personendaten an die REST-Schnittstelle, so sendet er immer den kompletten Datensatz, so wie er in tenfold abgebildet ist. Sollte die Schnittstelle so konfiguriert worden sein, dass die Personendaten auch von tenfold zurücksynchronisiert werden, muss ein Feldmapping (siehe [Feldmappings\(see page 556\)](#)) verwendet werden, welches angibt, welche Felder tatsächlich aus der Schnittstelle übernommen werden. Achten Sie darauf, dass die im Feldmapping eingestellten Mappings, auch wieder zu den korrekten Feldern gemappt werden. Für die Datenübertragung von tenfold an die Schnittstelle wird kein Feldmapping verwendet.

Feld	Datentyp	Beschreibung
id	string	Eine eindeutige ID des Users im System. Dies kann zum Beispiel die ID des User-Objektes aus der Datenbank sein, oder eine für das API generierte UUID. Entscheidend ist, dass die ID eindeutig ist und zwischen einzelnen Aufrufen des APIs unverändert bleibt. Diese ID wird zur Verknüpfung verwendet und in der Externen ID der Ressourcenzuordnung in tenfold gespeichert.
tenfoldId	string	Die ID des Personenobjektes aus tenfold. Diese ID wird an das API übergeben, wenn tenfold User-Änderungen anfordert, muss aber vom API nicht zurückgegeben werden.
userName	string	Der Benutzername des Users im System. Dieser kann sich vom Benutzernamen der Personenstammdaten in tenfold unterscheiden. Sollte der User von tenfold aus dem System ausgelesen werden, wird dieser Benutzername in der Ressourcenzuordnung der Systemressource gespeichert.
password	string	Das Passwort des Users. Dieses Feld wird von tenfold genutzt, um bei der Useranlage ein Passwort vorzugeben. Dieses Feld sollte beim Auslesen der Userdaten nicht an tenfold zurückgegeben werden.
mustChangePassword	boolean	Dieses Feld wird von tenfold bei der Useranlage benutzt, um dem System mitzuteilen, dass der User bei der nächsten Anmeldung sein Passwort ändern muss. Dieses Feld muss beim Auslesen der Userdaten nicht an tenfold übertragen werden.
tenfoldUserName	string	Dies ist der Benutzername des Personenstammdatenobjektes in tenfold. Dieser kann, muss aber nicht, mit dem Benutzernamen des Systems übereinstimmen. tenfold wird den Benutzernamen bei Änderungen an das System überliefern, das System muss dieses Feld jedoch weder beachten, noch zurücksenden.

Feld	Datentyp	Beschreibung
firstName	string	Der Vorname des Benutzers.
firstName2	string	Der Vorname des Benutzers, in welchem Sonderzeichen ersetzt wurden. (ö wird zu o, etc.)
lastName	string	Der Nachname des Benutzers.
lastName2	string	Der Nachname des Benutzers, in welchem Sonderzeichen ersetzt wurden. (ö wird zu o, etc.)
middleName	string	Der zweite Vorname des Benutzers.
middleName2	string	Der zweite Vorname des Benutzers, in welchem Sonderzeichen ersetzt wurden. (ö wird zu o, etc.)
fax	string	Die Faxnummer des Benutzers.
fax2	string	Eine alternative Faxnummer.
personalFax	string	Die persönliche Faxnummer des Benutzers.
phone	string	Die berufliche Telefonnummer des Benutzers.
phone2	string	Eine Alternative Telefonnummer.
phoneHome	string	Die private Telefonnummer des Benutzers
mobile	string	Die Mobiltelefonnummer des Benutzers.
email	string	Die E-Mailadresse des Benutzers.
jobTitle	string	Die Stellenbeschreibung des Benutzers als Freitextfeld.
department	department	Die Abteilung des Benutzers.
costCenter	costCenter	Die Kostenstelle des Benutzers.
personType	personType	Die tenfold Personenart des Benutzers (siehe <a href="#">Personenarten</a> (see page 81)).
building	building	Das Gebäude in welchem der Benutzer üblicherweise tätig ist.
office	office	Die Adresse/Niederlassung an welcher der Benutzer üblicherweise tätig ist.
gender	gender	Das Geschlecht des Benutzers.
ITCostCenter	costCenter	Die für die IT relevante Kostenstelle des Benutzers.

Feld	Datentyp	Beschreibung
title	title	Die Anrede des Benutzers (Herr/Frau/etc).
preTitle	title	Akademische Titel, welche vor dem Namen geführt werden (Dr., Mag., etc)
postTitle	title	Akademische Titel, welche nach dem Namen geführt werden (MSc, BSc, etc)
region	region	Die Region in welcher der Benutzer üblicherweise tätig ist.
position	position	Die Position des Benutzers im Unternehmen.
valueGroup[1-5]	valueGroup	Die Auswahl aus einer Dropdownliste, deren Inhalt frei konfigurierbar ist. Es gibt insgesamt 5 dieser Felder, valueGroup1, valueGroup2, etc.
check[1-5]	boolean	Status einer von 5 Checkboxes, check1, check2, etc.
employeeID	string	Die Personalnummer des Benutzers.
joiningDate	string	Das Eintrittsdatum des Benutzers.
leavingDate	string	Das Austrittsdatum des Benutzers.
expiryDate	string	Das Ablaufdatum des Benutzers (üblicherweise das Ablaufdatum des Accounts in Active Directory).
roomNumber	string	Die Büronummer des Benutzers.
status	status	Der aktuelle Status des Benutzers.
superior	user	Der Vorgesetzte des Benutzers.
date[1-5]	string	Frei definierbare Datumsfelder, date1, date2, etc.
userPrincipalName	string	Der User Principal Name des Benutzers aus dem Active Directory. Hinweis: In den meisten Fällen wird der User Principal Name von tenfold dynamisch generiert und in das Active Directory geschrieben, statt mit diesem Feld synchronisiert zu werden.
customAttributes	hash	Schlüssel/Wert-Paare mit Benutzerfeldern, welche durch Generic Connector-Einstellungen festgelegt werden. Diese Felder sind für mögliche zukünftige Erweiterungen des Generic Connectors vorgesehen und werden aktuell nicht verwendet.

Feld	Datentyp	Beschreibung
privileges	Liste: privilegeAssignment	Eine Liste aller Berechtigungen, über welche der Benutzer in der Schnittstelle verfügt.
options	Liste: userOptionAssignment	Eine Liste aller Optionen über welche der Benutzer in der Schnittstelle verfügt. Diese Optionen werden an der Generic Connector Ressource definiert und befinden sich nicht im eigentlichen Stammdatensatz des Benutzers. Diese sind dann von Bedeutung, wenn es keine zwingende 1:1 Beziehung von tenfold Personen und Benutzern in der Schnittstelle gibt. Daten welche von Benutzer zu Benutzer derselben Person unterschiedlich sein können, werden daher als Optionen abgebildet.

## privilege

Im Allgemeinen stellt ein Objekt vom Typ *privilege* eine Berechtigung im System der Schnittstelle dar. Es repräsentiert also eine Form von Zugriffsberechtigung auf das System. In speziellen Anwendungsfällen können privileges jedoch auch andere Objekte darstellen, für welche bei der Ressourcenzuordnung eine Mehrfachauswahl getroffen werden kann. Zum Beispiel könnte es sich um die Mitgliedschaft in einem Newsletter in einer Newsletteranwendung handeln oder eine Auswahl von Extra-Features für Hardwaregeräte für eine Gerätemanagementsoftware. Sollte Ihre Schnittstelle über verschiedene Typen von Objekten verfügen, welche hier vergeben werden können, so achten Sie bitte darauf diese in entsprechende Kontexte zu gliedern um dem tenfold-Anwender die Bedienung leichter zu machen. Nähere Informationen zu den Kontexten finden Sie im entsprechenden Abschnitt des Datenmodells.

Im Normalfall wird davon ausgegangen, dass Berechtigungen von Ihrer Schnittstelle vorgegeben werden und im Synchronisationsjob an tenfold übermittelt werden (GET /privileges). tenfold konfiguriert die zugehörige Ressource automatisch anhand der von Ihnen übertragenen Berechtigungen. In seltenen Fällen kann es jedoch auch vorkommen, dass die Berechtigungen in tenfold konfiguriert werden. Sie sollten dann jedoch darauf achten, dass Ihre Schnittstelle die vergebenen Berechtigungen in irgendeiner Art abspeichert und zurückliefert, da tenfold sonst die eingerichteten Berechtigungen bei der nächsten Synchronisation verwirft.

Feld	Datentyp	Beschreibung
id	string	Die id der Berechtigung im System. In tenfold wird dies im Feld "Externe ID" gespeichert, für das System ist es die interne id zur referenzierung der Berechtigung.
name	string	Der Name der Berechtigung. Diese wird als Anzeigename in tenfold verwendet. Zum Beispiel: "Administrator"
shortName	string	Ein Kürzel für die Berechtigung. Zum Beispiel "ADM".
description	string	Eine ausführliche Beschreibung der Berechtigung.
privilegeType	privilegeType	Die Art von Berechtigung. Zum Beispiel: Rolle, Einzelberechtigung
context	context	Der Kontext. Dies dient zur Gruppierung von Berechtigungen auf der tenfold-Oberfläche.

Feld	Datentyp	Beschreibung
remark	string	Ein Kommentar.
assignable	boolean	Legt fest, ob die Berechtigung über tenfold bestellt werden kann. Im Falle von false muss die Berechtigung über das System verwaltet werden und wird in tenfold nur zu Auswertungszwecken angezeigt.
searchInfo	string	Kann für die Suche nach Berechtigungen verwendet werden. Ein alternativer Text nach dem auch gesucht wird. wenn in tenfold die Berechtigungssuche verwendet wird.

### privilegeType

Ein Objekt vom Typ *privilegeType* legt die Art der Berechtigung fest. Jedes Objekt vom Typ *privilege* verfügt dabei auch über einen *privilegeType*. Für tenfold hat dies keinerlei Auswirkungen und wird nur zu Informationszwecken angezeigt. Übliche Typen von Berechtigungen sind zum Beispiel: "Rolle" oder "Privileg".

Feld	Datentyp	Beschreibung
id	string	Die id des Typs im System.
name	string	Der Anzeigenname des Typs.

### context

Bei einem *context* Objekt handelt es sich um eine beliebig definierbare Gruppierung für *privilege* Objekte. Jedes *privilege* Objekt muss zwingend einen *context* haben. Dies ist eine zwingende Grundvoraussetzung für tenfold. Sollte Ihr System keine Kontexte/Gruppierungen kennen, so muss die Schnittstelle zumindest einen Pseudokontext definieren. Ein Kontext kann dabei auch eine Reihe von Eigenschaften, sogenannten Optionen, definieren, welche für jede Berechtigungszuordnung extra vergeben werden können. Für einen Kontext kann zum Beispiel definiert werden ob für die Berechtigungen darin ein Start- und Enddatum eingerichtet werden kann, oder auch erweiterte Einstellungen wie eine Zusätzliche Auswahl von zum Beispiel "Schreiben", "Vollzugriff" für eine Berechtigung "Rechnungen verwalten".

Sollten erweiterte Optionen definiert werden, so ändert tenfold auf der Oberfläche die Berechtigungsauswahl zu einem erweiterten Dialog. Sind keine Optionen definiert, bleibt die Berechtigungsauswahl eine einfache Checkboxliste.

Im Normalfall wird davon ausgegangen, dass Kontexte von Ihrer Schnittstelle vorgegeben werden und im Synchronisationsjob an tenfold übermittelt werden (GET /contexts). tenfold konfiguriert die zugehörige Ressource automatisch anhand der von Ihnen übertragenen Kontexte.

Feld	Datentyp	Beschreibung
id	string	Die id des <i>context</i> im System.
name	string	Ein Name welcher in tenfold angezeigt wird.
shortName	string	Eine Kurzform des Namens oder eine Art Code.

Feld	Datentyp	Beschreibung
validityEditable	boolean	Gibt an, ob Berechtigungen im Kontext über ein Start- und Enddatum verfügen.
options	Liste: privilegeOption	Eine liste von erweiterten Eigenschaften, welche für alle Berechtigungen des Kontexts vergeben werden kann.

### privilegeAssignment

Ein *privilegeAssignment* Objekt stellt eine Zuordnung von einem Benutzer zu einer Berechtigung dar. Für jede Berechtigung, die ein Benutzer im System besitzt, gibt es in der Liste *privileges* des *user* Objektes einen Eintrag eines *privilegeAssignment* Objektes. An diesem Objekt werden auch alle erweiterten Optionen gespeichert, sofern der Kontext der Berechtigung solche definiert.

Verfügt der Kontext über Optionen oder ist die Einstellung *validityEditable* true, so ist es erforderlich, dass das API jedem *privilegeAssignment* eine eindeutige *id* zuweist, damit tenfold die Zuweisung identifizieren kann. Sollte dies nicht der Fall sein, kann diese *id* entfallen.

Wenn der Generic Connector den Synchronisationsjob durchführt, ruft er die Operation GET /users auf. In den dort gelieferten user Objekten müssen alle privilegeAssignments enthalten sein, sonst verwirft tenfold die den Personen zugeordneten Berechtigungen. Für das hinzufügen oder entfernen von Berechtigungen übermittelt tenfold privilegeAssignment Objekte an die Operation PUT /users/{id}/privileges bzw DELETE /users/{id}/privileges. Die privilegeAssignments, welche ein Aufruf an PUT /users/{id} zum Update von Personendaten sendet, sind folglich nicht zu verarbeiten.

Feld	Datentyp	Beschreibung
id	string	Die eindeutige ID der Berechtigungszuordnung im System. Diese kann entfallen, wenn der Kontext der Berechtigung keine erweiterten Optionen definiert.
userId	string	Die ID des Benutzers im System.
privilegeId	string	Die ID der Berechtigung im System.
contextId	string	Die ID des Kontexts im System.
requestReference	ineger	Die ID des Requests in tenfold, welcher eine Berechtigungsänderung ausgelöst hat. Diese wird von tenfold an die Schnittstelle gesendet, wenn Berechtigungen tenfold-Seitig geändert werden. Das System kann diese Information nutzen, wenn gewünscht. Das API muss diese Information nicht aufbewahren und bei zukünftigen Anfragen zurücksenden.
startDate	string(\$date-time)	Das Datum ab/seit dem die Zuordnung gültig ist. Wird nur verwendet wenn im Kontext die Einstellung <i>validityEditable</i> aktiv ist.

Feld	Datentyp	Beschreibung
endDate	string(\$date-time)	Das Datum bis zu dem die Zuordnung gültig ist. Wird nur verwendet wenn im Kontext die Einstellung <i>validityEditable</i> aktiv ist.
inherited	boolean	Gibt an, ob die Berechtigung vergeben wurde (false), oder in Abhängigkeit zu einer vergebenen Berechtigung besteht (true)
optionValues	Liste: privilegeOptionAssignment	Die tatsächlich getätigten Einstellungen zur Zuordnung, welche als Optionen beim Kontext definiert wurden.

### userOption

Objekte vom Typ *userOption* definieren, welche zusätzlichen Einstellungen bei der Zuordnung einer Ressource in tenfold getätigt werden können. Während die Personendaten wie Vorname oder Nachname für alle Benutzer der Person gleich sind, können mit Optionen Eigenschaften definiert werden, welche sich für die einzelnen Benutzer der Person unterscheiden.

Es wird davon ausgegangen, dass die Optionen vom API definiert werden (GET /users/options) und vom Generic Connector Synchronisationsjob täglich abgeglichen werden. Die Optionen werden daher in tenfold automatisch konfiguriert, wenn sie im API enthalten sind.

Feld	Datentyp	Beschreibung
id	string	Eine eindeutige id mit welcher tenfold die Option referenzieren kann. Sollte das System keinen eigenen Datensatz haben, und werden die Optionswerte direkt beim Benutzer gespeichert, muss die Schnittstelle selber eine id für jede Option definieren.
name	string	Der Name der Option, welche in tenfold angezeigt wird.
datatype	string	Ein Datentyp, welcher das Eingabefeld in tenfold definiert. Mögliche Werte sind: INTEGER, FLOAT, STRING, DATE, BOOLEAN, MULTILINE_STRING, SELECTION
description	string	Eine detailliertere Beschreibung, welche in tenfold zum Eingabefeld der Option angezeigt wird.
optionValues	Liste: optionValue	Eine reihe von Werten, welche als Dropdown in tenfold angezeigt werden, sollte der <i>datatype</i> SELECTION sein.

### privilegeOption

Objekte vom Typ *privilegeOption* werden von context Objekten definiert und legen die erweiterten Optionen eines Kontextes fest. Diese sind direkt im context Objekt enthalten und werden daher mit GET /contexts



mitübertragen und vom Generic Connector beim Abgleich ausgewertet. Ein Kontext muss keine Optionen definieren. Die Liste von *privilegeOption* Objekten kann auch leer sein (aber nicht null).

Feld	Datentyp	Beschreibung
id	string	Eine eindeutige id anhand derer die Option von tenfold referenziert werden kann. Sollte das System über keine solcher ids verfügen, muss das API selber ids definieren.
name	string	Der Name der Option welcher bei der Berechtigungseingabe angezeigt wird.
contextId	string	Die id des Kontexts zu welchem die Option gehört.
datatype	string	Ein Datentyp, der die Art des Eingabefeldes in tenfold bestimmt. Mögliche Werte sind: INTEGER, FLOAT, STRING, DATE, BOOLEAN, MULTILINE_STRING, SELECTION
description	string	Eine detaillierte Beschreibung, welche in tenfold auf der Eingabemaske angezeigt werden kann.
optionValues	Liste: optionValue	Eine reihe von Werten, welche als Dropdown in tenfold angezeigt werden, sollte der <i>datatype</i> SELECTION sein.

### userOptionAssignment

Eine Zuordnung von Optionen zu einem Benutzer. In tenfold werden diese Optionen bei der Ressourcenzuordnung gespeichert und nicht im Personenstammdatensatz. Im System kann es sich aber durchaus um Daten handeln, welche direkt beim Benutzer gespeichert werden. Es liegt an der Schnittstelle diese Werte entsprechend zu transformieren.

Feld	Datentyp	Beschreibung
optionId	string	Die eindeutige id der entsprechenden <i>userOption</i> .
userId	string	Die id des Benutzers im System.
simpleValue	string	Der Wert der Option als Text, welcher für alle Datentypen außer SELECTION verwendet wird. Das API muss etwaige Datenkonvertierungen (z.B. von string auf int) selbst durchführen.
complexValue	optionValue	Falls der Datentyp der Option SELECTION ist, befindet sich hier der ausgewählte Wert des in tenfold erzeugten Dropdown-Menüs.

## privilegeOptionAssignment

Dieses Objekt stellt den tatsächlich eingegebenen Wert einer Kontext-Option zu einer Berechtigungszuordnung dar.

Feld	Datentyp	Beschreibung
id	string	Die id der Berechtigungszuordnung im System.
optionId	string	Die Kontextoption des Systems.
simpleValue	string	Der Wert der Option als Text, welcher für alle Datentypen außer SELECTION verwendet wird. Das API muss etwaige Datenkonvertierungen (z.B. von string auf int) selbst durchführen.
complexValue	optionValue	Falls der Datentyp der Option SELECTION ist, befindet sich hier der ausgewählte Wert des in tenfold erzeugten Dropdown-Menüs.

## optionValue

Kontext- oder Benutzeroptionen vom Datentyp SELECTION stellen eine Auswahlliste dar, aus welcher ein Wert ausgewählt werden kann. Der Typ optionValue repräsentiert eine Auswahlmöglichkeit aus so einer Liste.

Feld	Datentyp	Beschreibung
id	string	Die id der getätigten Auswahl. Diese wird von tenfold benutzt um die entsprechende Auswahl zu referenzieren.
name	Name	Ein Anzeigename welcher in tenfold angezeigt wird.

## country

Dieser Typ stellt ein Land dar, welches bei Niederlassungen(office) in tenfold hinterlegt wird und bietet mehrere Eigenschaften zur genaueren Identifizierung.

Feld	Datentyp	Beschreibung
name	string	Der Name des Landes.
iso31661a2	string	Der Ländercode nach iso31661a2-Norm
iso31661a3	string	Der Ländercode nach iso31661a3-Norm
iso31661n	string	Der Ländercode nach iso31661n-Norm

## company

Objekte vom Typ *company* stellen die in tenfold hinterlegten Unternehmensstammdaten dar. Diese werden bei Niederlassungen hinterlegt und sind daher auch Teil des *office*-Objektes. Ein *user*-Objekt selbst hat keinen direkten Bezug zu einem Unternehmen, dieser Bezug wird einzig und allein über das *office*-Objekt des Benutzers hergestellt.

Feld	Datentyp	Beschreibung
id	string	Die eindeutige id des Unternehmens. Dieses Feld enthält den Wert des Feldes "Externe ID" im tenfold-Datensatz des Unternehmens.
name	string	Der Name des Unternehmens.
code	string	Ein beliebig wählbarer Code. Dieser hat für tenfold keine spezielle Bedeutung, kann aber zum Beispiel verwendet werden, um Kurzformen aus der Buchhaltung oder Ähnliches abzubilden. Beispiel: Das Unternehmen tenfold Software GmbH hat den Code TNF.
department	department	Eine Standardabteilung für Unternehmen. Gibt es in Ihrem Unternehmen zum Beispiel eine Tochtergesellschaft für die IT, so kann hier die IT-Abteilung hinterlegt werden. Diese Eigenschaft ist zumeist rein Informativ.
company	company	Das übergeordnete Unternehmen. Wird zur Abbildung von hierarchischen Unternehmensstrukturen verwendet.

### office

Ein Objekt vom Typ *office* stellt eine Niederlassung oder Adresse dar, an welcher Personen üblicherweise ihre Tätigkeiten verrichten. Dieses Objekt ist Teil des *user*-Objektes. Standort und Niederlassung sind in diesem Zusammenhang als Synonyme zu betrachten.

Feld	Datentyp	Beschreibung
id	string	Die eindeutige id der Niederlassung. Hier wird der Wert des Feldes "Externe ID" im tenfold-Datensatz der Niederlassung übertragen. Anhand dieser ID kann tenfold die Niederlassung matchen, sollte der Abgleich der Personendaten aktiv sein.
code	string	Ein Code zur Abkürzung der Niederlassung, welcher zum Beispiel aus dem Buchhaltungssystem kommen könnte. Zum Beispiel: Der Standort Wien hat den Code VIE.
name	string	Der Name des Standortes.
phone	string	Die allgemeine Telefondurchwahl des Standortes.
fax	string	Die allgemeine Faxdurchwahl des Standortes.
address	string	Die Adresse des Standortes (Straßenname und Nummer).
city	string	Die Stadt in welcher sich der Standort befindet.
zip	string	Die Postleitzahl an welcher sich der Standort befindet.
state	string	Das Bundesland in welcher sich die Niederlassung befindet.

Feld	Datentyp	Beschreibung
country	country	Das Land in welchem sich der Standort befindet.
ou	organizationUnit	Die Organisationseinheit zu welcher die Niederlassung gehört. Dies ist im Sinne der Organisationsstruktur zu verstehen und nicht zwangsläufig als OU im Active Directory.
company	company	Das Unternehmen zu welchem der Standort gehört.

## building

Der Typ *building* stellt ein Gebäude dar in welchem Personen Ihre Tätigkeiten verrichten. Zu jedem Standort kann es hierbei mehrere Gebäude geben. Einer Person kann dann eines der Gebäude am jeweilig ausgewählten Standort zugeordnet werden. Der Generic Connector sendet mit dem *user*-Objekt das ausgewählte Gebäude an das API. Die Zuordnung von einem Gebäude zur Niederlassung wird nicht an das API übermittelt.

Feld	Datentyp	Beschreibung
id	string	Eine eindeutige ID mit welche tenfold das Gebäude referenzieren kann.
eid	string	Eine id aus einem Fremdsystem, welche tenfold verwenden kann um das Gebäude in diesem Fremdsystem zu identifizieren.
code	string	Ein Code zur Abkürzung des Gebäudes, welches Beispielsweise aus der Buchhaltung kommen könnte. Beispiel: Das Hauptgebäude hat den Code HPT.
name	string	Der Name des Gebäudes. Zum Beispiel: Hauptgebäude.

## department

Objekte des Typs *department* stellen eine Abteilung im Unternehmens dar. tenfold unterstützt zwar eine Hierarchische Abbildung von Abteilungen (Eltern-Kind Beziehungen), diese werden im REST-API jedoch nicht abgebildet. In tenfold gibt es unterhalb der Abteilung keine weitere organisatorischen Strukturen mehr. Sollten in Ihrem Unternehmen unterhalb der Abteilung weitere Untergliederungen stattfinden, so werden diese in tenfold sehr wahrscheinlich auch durch *department*-Objekte abgebildet.

Feld	Datentyp	Beschreibung
id	string	Eine ID zur Erkennung der Abteilung. tenfold kann diese ID benutzen, um Abteilungen zu matchen, sollte der Datenabgleich für Personendaten aktiv sein. Dieses Attribut bezieht sich auf das Feld "Externe ID" des Abteilungseintrages in tenfold.
name	string	Der Name der Abteilung, zum Beispiel: Information Technology.
shortName	string	Eine Kurzform des Abteilungsnamens, zum Beispiel: IT
description	string	Eine Beschreibung für die Abteilung.

Feld	Datentyp	Beschreibung
region	region	Die örtliche Region, in welcher die Abteilung operiert.

### organizationUnit

Der Typ *organizationUnit* stellt eine organisatorische Einheit innerhalb des Unternehmens dar. Eine *organizationUnit* hängt hierbei immer an einem *office*-Objekt und ist darüber mit dem *user*-Objekt verknüpft.

Feld	Datentyp	Beschreibung
id	string	Das Feld "Externe ID" des tenfold-Datensatzes. Kann verwendet werden, um beim Datenabgleich, im Falle das der Abgleich von Benutzerdaten aktiviert ist, die Organisationseinheit in tenfold zu matchen.
name	string	Der Name der Organisationseinheit.
code	string	Ein Code für die Organisationseinheit.

### gender

Objekte vom Typ *gender* stellen eine Auswahlmöglichkeit für Geschlechter dar. Diese Liste ist in tenfold frei verwaltbar.

Feld	Datentyp	Beschreibung
id	string	Eine eindeutige ID für den Datensatz. Dies entspricht dem Feld "Externe ID" in tenfold.
name	string	Ein frei wählbarer Name. Dieser wird in einer Dropdown-Liste auf der tenfold-Oberfläche angezeigt. Zum Beispiel: Mann, Frau, Divers.
code	string	Eine frei wählbarer Code. Welcher für beliebige Zwecke verwendet werden kann.

### valueGroup

Ein Objekt vom Typ *valueGroup* stellt einen Eintrag aus einer in tenfold frei definierbaren Dropdown-Liste dar. Da bei jeder Personenart für die Felder *valueGroup1-5* unterschiedliche Dropdownlisten hinterlegt werden können, muss hier auf den entsprechenden Kontext der Person geachtet werden.

Feld	Datentyp	Beschreibung
id	string	Eine eindeutige ID für den Datensatz. Dies entspricht dem Feld "Externe ID" der Auswahl in tenfold.
name	string	Der Name der Auswahl. Diese wird auf der Oberfläche in der Dropdown-Liste angezeigt.
code	string	Ein frei wählbarer Code, welcher für verschiedene Zwecke definiert werden kann.

## region

*region*-Objekte sind örtliche Untergliederungen für Abteilungen(department). Jede Abteilung kann hierbei einer Region zugeordnet werden.

Feld	Datentyp	Beschreibung
id	string	Eine eindeutige ID für den Datensatz. Dies entspricht dem Feld "Externe ID" der Region in tenfold.
name	string	Der Name der Region. Diese wird auf der Oberfläche in der Dropdown-Liste angezeigt bei Abteilungen angezeigt.
code	string	Ein frei wählbarer Code welcher für verschiedene Zwecke definiert werden kann.

## position

Objekte des Typs *position* stellen getroffene Auswahlen aus einer Liste von frei konfigurierbaren Positionen für eine Person dar. Dies ist vergleichbar mit dem Attribut "jobTitle" des *user*-Objektes, welches ein Freitextfeld ist.

Feld	Datentyp	Beschreibung
id	string	Eine eindeutige ID für den Datensatz. Dies entspricht dem Feld "Externe ID" der Position in tenfold.
name	string	Der Name der Position. Diese wird auf der Oberfläche in der Dropdown-Liste angezeigt.
code	string	Ein frei wählbarer Code welcher für verschiedene Zwecke definiert werden kann.

## title

Objekte des Typs *title* stellen getroffene Auswahlen aus einer Liste von frei konfigurierbaren Titeln und Anreden für eine Person dar. Bei einem *user*-Objekt gibt es die drei Felder *preTitle*, *postTitle* und *title*, welche jeweils Titel welche vor bzw Nach dem Namen geführt werden, sowie allgemeine Anreden darstellen. Alle diese Attribute verwenden Werte vom Typ *title*.

Feld	Datentyp	Beschreibung
id	string	Eine eindeutige ID für den Datensatz. Dies entspricht dem Feld "Externe ID" des Titels in tenfold.
name	string	Der Name des Titels. Diese wird auf der Oberfläche in der Dropdown-Liste angezeigt.
code	string	Ein frei wählbarer Code welcher für verschiedene Zwecke definiert werden kann.

## personType

*personType*-Objekte stellen eine Form von Gruppierung von Personen in tenfold dar. In tenfold kann für jede Personenart eine ganze Reihe an Einstellungen getroffen werden (siehe [Personenarten](#)(see page 81)), für das API haben diese jedoch keine Relevanz und werden daher nicht mitgeliefert.

Feld	Datentyp	Beschreibung
id	string	Eine eindeutige ID für den Datensatz. Dies entspricht dem Feld "Externe ID" der Personenart in tenfold.
name	string	Der Name der Personenart. Diese wird auf der Oberfläche in der Dropdown-Liste angezeigt.

## costCenter

*costCenter*-Objekte sind Abbildungen von Kostenstellen des Unternehmens, welche in tenfold eingetragen wurden. In *user*-Objekten können zwei unterschiedliche Kostenstellen eingetragen werden. Einmal im Attribut *costCenter*, welches die allgemeine Kostenstelle der Person angibt und das Attribut *ITCostCenter*, welches eine IT spezifische Kostenstelle für die Person angibt.

Feld	Datentyp	Beschreibung
id	string	Eine eindeutige ID für den Datensatz. Dies entspricht dem Feld "Externe ID" der Kostenstelle in tenfold.
name	string	Der Name des Kostenstelle. Diese wird auf der Oberfläche in der Auswahlliste angezeigt beim Bearbeiten von Personen.
code	string	Der Buchhaltungs-Code der Kostenstelle.

## datatype

Bei *datatype* handelt es sich um einen einfachen Aufzählungstypen, welcher folgende Werte beinhalten kann:

- INTEGER
- FLOAT
- STRING
- DATE
- BOOLEAN
- MULTILINESTRING
- SELECTION

Durch die Angabe eines *datatype* bei den verschiedenen Datenstrukturen des APIs, kann tenfold bei der Eingabe in tenfold entsprechende Eingabefelder verwenden.

## status

*status* ist ein Aufzählungstyp, welcher den Status eines *user*-Objektes angibt und kann folgende Werte enthalten:

- ACTIVE
- LOCKED
- DELETED

## apiError

Der Typ *apiError* definiert eine Nachricht, welche im Fehlerfall versendet werden kann. tenfold erwartet *apiError*-Objekte immer dann wenn das API eine Fehlermeldung mit Nummer 500 (Internal Server Error) liefert, um eine nähere Beschreibung des Fehlers zu liefern.

Feld	Datentyp	Beschreibung
message	string	Eine detaillierte Beschreibung des Problems.

### 14.10.3 Implementierung eines APIs

Wie in den vorangegangenen Abschnitten beschrieben wurde, muss zur Verwendung des Generic Connector Plugins eine Implementierung des Generic Connector APIs vorliegen. Diese Implementierung kann grundsätzlich mit jeder von Ihnen bevorzugten Technologie zur Implementierung eines REST-APIs durchgeführt werden.

In den folgenden Abschnitten wird anhand eines Beispiels beschrieben wie Sie eine eigene Schnittstelle für den Generic Connector implementieren können.

Für die folgenden Beispiele wird zur Realisierung ausschließlich Java verwendet. Sie können dennoch jede beliebige Sprache und/oder Technologie verwenden.

#### Schnittstellenbeschreibung herunterladen

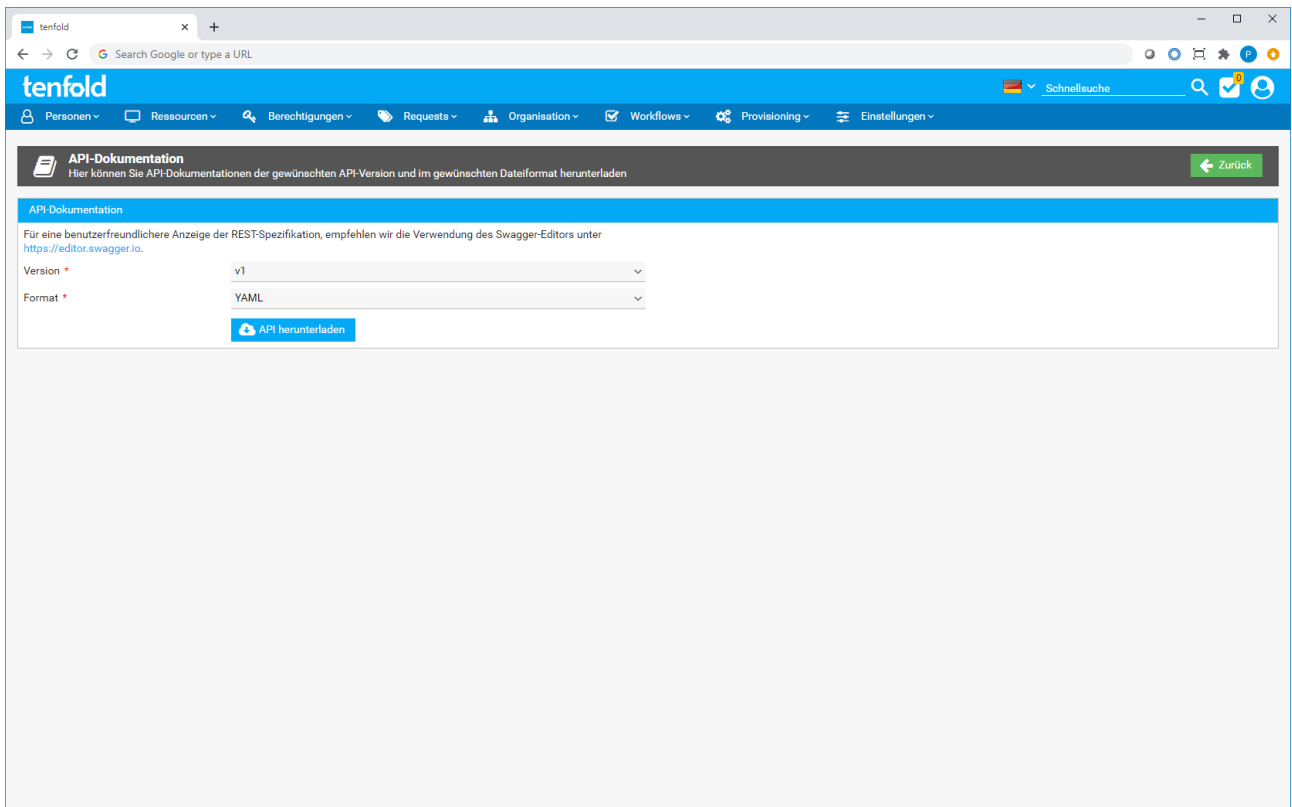
Sie können die Schnittstelle von Grund auf nach der Beschreibung im vorhergehenden Kapitel selbst einrichten, es wird jedoch empfohlen das API mittels der mitgelieferten Swagger Schnittstellenbeschreibung zu implementieren (<https://swagger.io>). Swagger ist ein Set an Tools welche es erlaubt eine REST-Schnittstelle mittels einer Datei im JSON oder YAML-Format zu beschreiben.

Dies beruht auf demselben Prinzip wie es mit einer WSDL-Datei auch im SOAP-Umfeld schon vorhanden ist. Während WSDL-Schnittstellenbeschreibungen jedoch im SOAP-Standard bereits enthalten sind, bietet REST kein eigenes Mittel zur Beschreibung des APIs. Swagger hat sich jedoch als der De-Facto Standard zur Schnittstellen-Beschreibung von REST APIs herauskristallisiert.

Damit die Implementierung eines APIs also einfach und reibungslos ablaufen kann, liefert das Generic Connector Plugin die Beschreibung des APIs im Swagger-Format bereits mit. Die Beschreibung kann hierbei sowohl als JSON- als auch YAML-Datei heruntergeladen werden.

Navigieren Sie hierzu auf die Einstellungsmaske des Generic Connector Plugins unter Provisioning > Plugins > Generic Connector und betätigen die Schaltfläche „API-Dokumentation“ in der Titelleiste.





Sie können daraufhin das gewünschte Datei-Format auswählen und mit der Schaltfläche „API-Herunterladen“ die Datei auf Ihrem Gerät abspeichern. Danach können Sie mittels dieser Datei jedes von Ihnen präferierte Tool benutzen, um damit ein Grundgerüst für Ihr API erstellen zu lassen.

#### API Server

Beachten Sie hierbei, dass der Server des APIs zu erstellen ist, da tenfold hier den Part des Clients übernimmt.

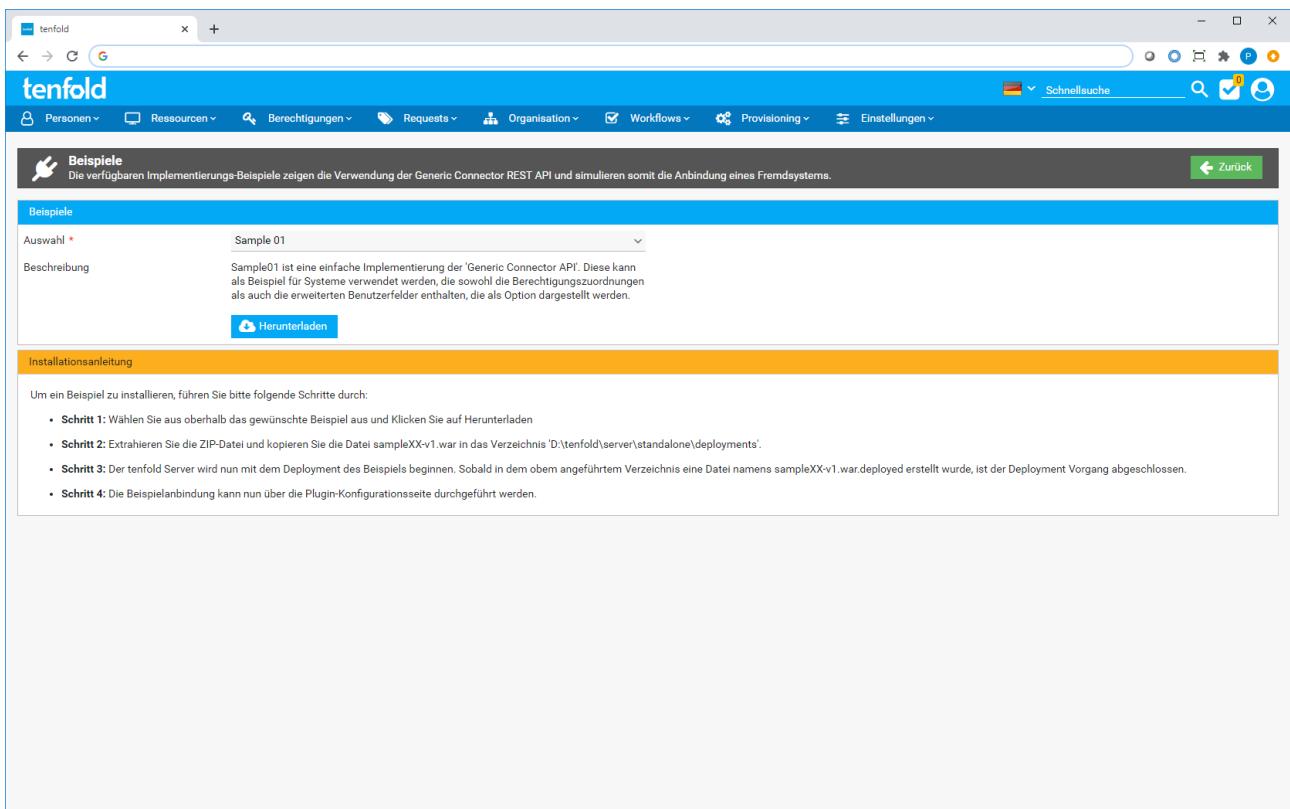
Auf <https://editor.swagger.io> können Sie mittels dieser Dateien eine grafische Darstellung des APIs erzeugen und auch Servergerüste erstellen lassen.

### Stub herunterladen

Als Alternative zum Herunterladen und Erzeugen des Servergerüsts, können Sie auch den Beispiel-Stub herunterladen, welchen Ihnen tenfold zur Verfügung stellt. Dieser ist ein bereits fertig erzeugtes Grundgerüst für Ihr API, in welchem Sie nur noch die gewünschte Funktionalität implementieren müssen.

Dieser Stub wurde in Java geschrieben und verwendet dort gängige Webtechnologien.

Diesen Stub finden Sie, gemeinsam mit anderen Beispielen, auf der Beispielseite des Generic Connector Plugins. Navigieren Sie hierfür auf die Einstellungsseite des Generic Connector Plugins unter Provisioning > Plugins > Generic Connector und betätigen die Schaltfläche „Beispiele“ in der Titelleiste.



Wählen Sie hier „Server Stub“ aus und klicken auf die Schaltfläche „Herunterladen“. Sie erhalten daraufhin eine Zip-Datei mit dem kompletten Code-Gerüst für einen Server. Dieses beinhaltet auch ein Maven-Projekt (für nähere Informationen siehe <https://maven.apache.org>) mit welchem Sie das Codegerüst bauen können. In diesem Stub befinden sich bereits alle notwendigen Quellcodedateien, um ein leeres API zu erzeugen. Um dieses API mit der von Ihnen gewünschten Funktionalität auszustatten müssen die entsprechenden Java-Klassen mit Programmlogik befüllt werden.

## Erklärung des Stubs

Der Stub basiert im wesentlichen auf den folgenden Java Frameworks.

- Java Web
- Resteasy (<https://resteasy.github.io/>)
- CDI - Contexts and Dependency Injection (<https://weld.cdi-spec.org/>)
- Jackson (<https://github.com/FasterXML/jackson>)

Da das Java Web Framework Teil der Java EE Platform (<https://jakarta.ee/>) ist, lässt sich diese Anwendung ohne Weiteres um die zahlreichen erweiterten Funktionen der Enterprise Platform erweitern.

### Java Webcontainer

Der generierte Stub lässt sich problemlos auf jedem gängigen Java Web Server wie Tomcat (<https://tomcat.apache.org/>) betreiben. Sollten Sie weitere Module der Java Enterprise Platform in Anspruch nehmen, so können diese nur mehr auf vollständigen Java Enterprise Platform Servern wie JBoss WildFly (<https://www.wildfly.org/>) ausgeführt werden.

### Deployment auf dem tenfold Server

Da tenfold auf einem JBoss WildFly server ausgeführt wird, können Sie den Stub, sowie alle Beispiele auch einfach auf dem tenfold Server installieren. Kopieren Sie hierfür die von Ihnen erzeugte .war oder .ear Datei nach <tenfold Installationsverzeichnis>\server\standalone\deployments.

Alle Quellcodedateien befinden sich im Ordner *src*. Dieser Ordner ist unterteilt in die Ordner *gen* und *main*.

Ordner	Zweck
gen	In diesem Ordner befinden sich sämtliche Quellcodedateien, welche von swagger.io erzeugt wurden und die Java Webinfrastruktur beinhalten, die zur Ausführung des APIs notwendig ist. Da diese Klassen das Grundgerüst der Anwendung darstellen befindet sich an dieser Stelle kein Code, welcher für Ihre spezielle Implementierung verantwortlich ist. In einem Großteil der Fälle ist es nicht notwendig oder empfehle, Dateien in diesem Ordner und dessen Unterordnern zu bearbeiten.
main	In diesem Ordner befinden sich die Klassen, welche für die individuelle Funktionalität Ihres APIs verantwortlich sind. Ein großteil der Anpassungen wird in diesem Ordner stattfinden.

## Das Datenmodell

Das Datenmodell befindet sich im Java-Paket *com.tenfoldsecurity.gcstub.model* im *gen* Ordner des Stubs. Diese Klassen sollten unter keinen Umständen bearbeitet werden, da sie die Datenstruktur abbilden, welcher zur Kommunikation mit dem tenfold Server verwendet wird. Eine Änderung dieser Klassen kann sehr leicht dazu führen, dass die Datenstruktur, welche von tenfold gesendet und erwartet wird, nicht mehr mit Ihrem API zusammenpasst und daher Fehlermeldungen erzeugt. Die Klassen in diesem Paket sind strukturell identisch mit dem im Abschnitt [Datenmodell\(see page 757\)](#) beschriebenen Modell.

Zu jedem Datensatz im Datenmodell gibt es eine analoge Klasse in diesem Paket. Die Felder des Datenmodells sind dabei in der gängigen Java Eigenschaftsnotation implementiert. Dies bedeutet, dass es zum Feld *id* im Datensatz *User* eine entsprechende Methode *getId()* für den lesenden und eine Methode *setId()* für den schreibenden Zugriff auf das Feld gibt. Sämtliche Felder haben hierbei entweder den Datentyp *java.lang.String* oder eine andere Klasse aus dem Datenmodell.

Die meisten API-Methoden werden Datenstrukturen aus diesem Paket erwarten oder zurückliefern. Es ist daher anzuraten eine gewisse Vertrautheit mit dem Datenmodell herzustellen.

## Die Schnittstellendefinitionen

Die Schnittstellendefinitionen befinden sich im Java-Paket *com.tenfoldsecurity.gcstub.api* des *gen* Ordners im Stub. Diese spiegeln die REST-Schnittstellendefinitionen des APIs wieder. Eine Änderung an dieser Stelle führt ebenso wie beim Datenmodell dazu, dass Ihr REST API aller Wahrscheinlichkeit nach nicht mehr von tenfold verarbeitet werden kann. Eine Änderung an dieser Stelle ist dringend abgeraten.

Die folgenden Klassen und deren Methoden sind dafür zuständig, die REST-Schnittstelle abzubilden:

Methode	HTTP-Methode	URL
<b>ContextsApi</b>		
getAllContexts	GET	/contexts

LoginApi		
login	GET	/login/{username}
PrivilegesApi		
getAllPrivileges	GET	/privileges
UsersApi		
changePassword	PUT	/users/{id}/password
createUser	POST	/users
deleteUser	DELETE	/users/{id}
getAllUserOptions	GET	/users/options
getAllUsers	GET	/users
lockUser	PUT	/users/{id}/lock
removeUserPrivileges	DELETE	/users/{id}/privileges
unlockUser	PUT	/users/{id}/unlock
updateUser	PUT	/users/{id}/
updateUserOptions	PUT	/users/{id}/options
updateUserPrivileges	PUT	/users/{id}/privileges

Eine Beschreibung der einzelnen API Methoden finden Sie unter [API Dokumentation](#)(see page 740).

Diese Klassen stellen lediglich die REST-Schnittstellenbeschreibung dar. Diese Klassen implementieren nicht die Funktionalität des APIs. Hierfür befinden sich in diesem Paket weitere Java-Schnittstellen, eine für jede \*Api Klasse.

REST-API Klasse	Java-Schnittstelle
ContextsApi	ContextsApiService
LoginApi	LoginApiService
PrivilegesApi	PrivilegesApiService

REST-API Klasse	Java-Schnittstelle
UsersApi	UsersApiService

Jede der Java-Schnittstellen deklariert denselben Satz an Methoden wie die dazugehörige REST-API Klasse. Jede dieser Schnittstellen findet eine Implementierung, welche im folgenden Abschnitt beschrieben wird.

### Schnittstellenimplementierung

Im Ordner *main* befindet sich das Java-Paket *com.tenfoldsecurity.gcstub.api.impl*. In diesem Paket befinden sich Implementierungen der Schnittstellen, welche im vorhergehenden Abschnitt beschrieben wurden. Diese tragen denselben Namen der Schnittstelle gefolgt von *Impl*.

Schnittstelle	Implementierung
ContextsApiService	ContextsApiServiceImpl
LoginApiService	LoginApiServiceImpl
PrivilegesApiService	PrivilegesapiServiceImpl
UsersApiService	UsersApiServiceImpl

Diese Klassen realisieren die eigentliche Funktionalität des APIs und sind jene Klassen, die für die eigene Implementierung des APIs angepasst werden müssen. Alle Methoden in diesen Klassen sind bereits vorimplementiert und liefern einen einfachen Erfolgs-Code an den Client, ohne konkrete Daten zu liefern. Als Beispiel finden Sie hier die Methode *getAllContexts()* der Klasse *ContextsApiServiceImpl*.

Beispiel eines API Methodengerüsts	
1	<code>public Response getAllContexts(SecurityContext securityContext)</code>
2	<code>throws NotFoundException</code>
3	<code>{</code>
4	<code>    // TODO: Implement Logic</code>
5	<code>    return Response.ok().entity(new</code>
6	<code>        ApiResponseMessage(ApiResponseMessage.OK, "magic!")).build();</code>
	<code>}</code>

An jede der Methoden wird, zusätzlich zu den Parametern des REST-APIs, auch eine Instanz der Klasse *SecurityContext* übergeben. Mithilfe dieses Objektes können Sie auf die Informationen des Users zugreifen, mit welchem das API aufgerufen wurde. Dies ist nur dann notwendig, wenn Sie planen, Ihr API mit Berechtigungen abzusichern. tenfold selbst wird immer dieselben Zugangsdaten für den Aufruf verwenden. Sollten Sie dieses API also keinem anderen Client zur Verfügung stellen wollen, benötigen Sie dieses Objekt nicht.

Die Methode benutzt Funktionen der Resteasy Bibliothek, um eine JSON-Repäsentation einer einfachen Erfolgsnachricht an den Client zu übermitteln. Es wird hierfür *Response.ok().entity(...).build()* verwendet. Das Objekt, welches an die *entity()*-Methode übergeben wird, wird von Jackson in eine JSON-Repräsentation selbiger umgewandelt.

Eine einfache Implementierung dieser Methode könnte zum Beispiel einen vordefinierten Default-Kontext zurückliefern.

### Einfache Implementierung

```

1  public Response getAllContexts(SecurityContext securityContext)
2      throws NotFoundException
3  {
4      Context myDefaultContext = new Context();
5      myDefaultContext.setId("1");
6      myDefaultContext.setName("DEFAULT");
7      return Response.ok().entity(myDefaultContext).build();
8  }

```

Auf diese Art und Weise müssen alle benötigten API-Methoden implementiert werden. Im folgenden Abschnitt finden Sie eine einfache Implementierung des Generic Connector APIs beschrieben.

### Vom Stub zum API: Tutorial 1

In diesem Abschnitt wird eine einfache Implementierung des Generic Connector APIs erzeugt. Diese Implementierung hält einfach alle Daten im Hauptspeicher. Dadurch gehen die Daten zwischen einzelnen Dienst-Neustarts verloren und damit eignet sich diese Implementierung für den Produktionseinsatz. Sie dient lediglich zur Veranschaulichung des Implementierungsprozesses.

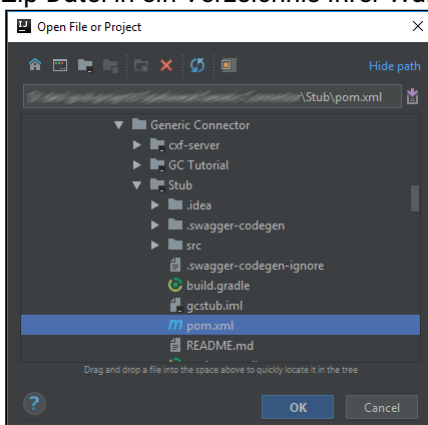
Als Grundlage dieses APIs dient der im vorangegangenen Abschnitt erwähnte Stub, welchen Sie über die Einstellungsseite des Generic Connector Plugins herunterladen können. Auf dieser Seite können Sie ebenso den Code dieses APIs herunterladen.

Für die Entwicklung dieser Anwendung wurde IntelliJ IDEA Community Edition verwendet. Alle beschriebenen Schritte basieren daher auf dieser Entwicklungsumgebung. Sie können auch jede andere Entwicklungsumgebung verwenden, welche Ihnen vertraut ist, müssen dann jedoch die einzelnen Schritte gegebenenfalls auf Ihre Entwicklungsumgebung übertragen.

Im folgenden finden Sie eine Schritt-für-Schritt-Anleitung, wie Sie aus dem Stub diese einfache Anwendung erzeugen.

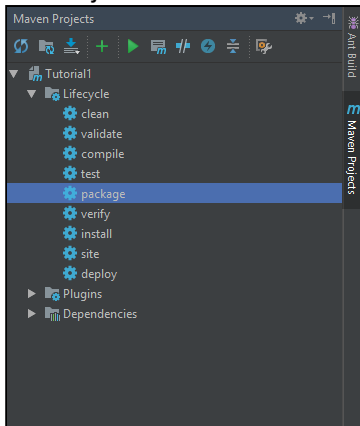
### Stub herunterladen, bauen, testen

Laden Sie zunächst den Stub herunter wie im vorhergehenden Kapitel beschrieben wurde. Entpacken Sie die Zip-Datei in ein Verzeichnis Ihrer Wahl und öffnen die darin enthaltene .pom-Datei mit IntelliJ.



IntelliJ fragt Sie daraufhin, ob Sie die Datei als einzelne Datei oder als Projekt öffnen möchten. Öffnen Sie die Datei als Projekt. IntelliJ erzeugt daraufhin die eigens benötigten Projektstrukturen. In Zukunft können Sie den Stub dann als normales IntelliJ Projekt öffnen.

Sobald das Projekt geöffnet ist, können Sie die "Maven Project"-Ansicht auf der rechten Seite ausklappen und das Projekt bauen. Wählen Sie hierfür das Maven-Goal "package" aus und führen es durch.



Nachdem Maven alle Abhängigkeiten heruntergeladen und das Projekt gebaut hat, finden Sie im Projektverzeichnis unter `target\gcstub-1.0` war eine fertige Java-Webanwendung, welche Sie auf einem beliebigen Java-Webserver ausführen können. Sie können diese Datei zum Beispiel einfach in das Verzeichnis `server\standalone\deployments` Ihrer tenfold-Installation kopieren, woraufhin das API auf demselben Rechner wie tenfold ausgeführt wird.

Das API ist dann verfügbar unter <https://ihr-server/gc/v1>.

Sie können diesen Pfad anpassen, indem Sie die Datei `src/main/webapp/jboss-web.xml` bearbeiten.

```

1 <jboss-web>
2   <context-root>/gc/v1</context-root>
3 </jboss-web>

```

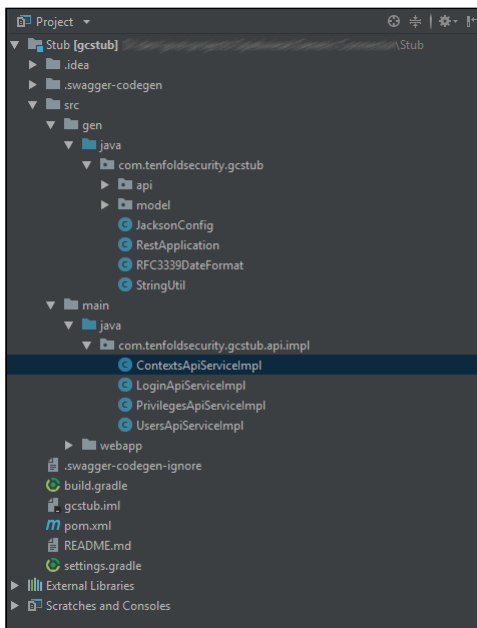
Passen Sie hier einfach den Wert des Tags `context-root` an.

### Deploymenteinstellungen

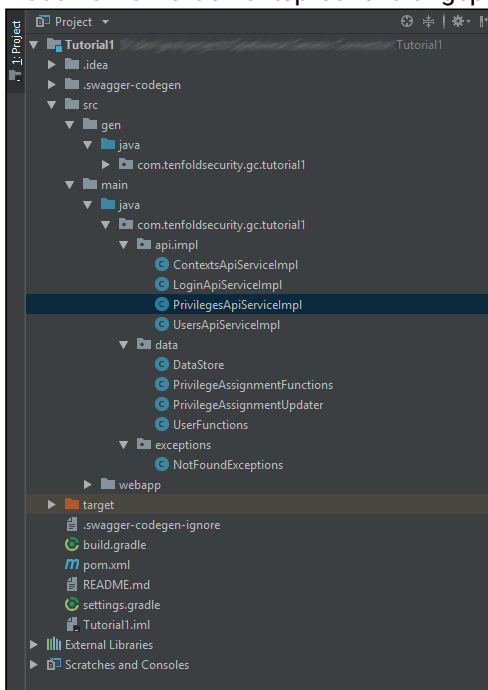
Bitte beachten Sie, dass diese Einstellung nur für Deployment auf dem tenfold-Server oder anderen WildFly kompatiblen Java Servern (<https://www.wildfly.org/>), wie etwa dem JBoss EAP (<https://www.redhat.com/de/technologies/jboss-middleware/application-platform>) funktioniert. Sollten Sie Ihr API auf einem anderen Server installieren, konsultieren Sie bitte die Dokumentation Ihrer Server-Software wie Sie den Kontextpfad der Anwendung dort konfigurieren können.

### Stub anpassen

Auf der linken Seite finden Sie die Projektansicht des Stubs.

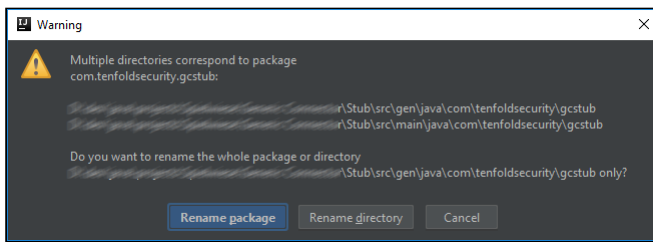


An dieser Stelle können Sie das Java-Paket `com.tenfoldsecurity.gcstub` auf ein Paket Ihrer wahl anpassen. In diesem Beispiel wurde das Paket geändert in `com.tenfoldsecurity.gc.tutorial1`. Auch der Projektname und der Modulname wurden entsprechend angepasst.



Erreichen können Sie dies über das Menü `File > Project Structure...` um das Projekt und Modul umzubenennen. Um das Package umzubenennen verwenden Sie einfach die Refactoring-Funktion von IntelliJ. Wählen Sie dafür eines der beiden Root-Packages (`com.tenfoldsecurity.gcstub` unter `gen` oder `main`) an und wählen `Refactoring > Rename...` im Kontextmenü des Knotens oder verwenden den Shortcut `Umschalt+F6`. IntelliJ erkennt automatisch, dass dieses Package auf mehrere Ordner aufgeteilt ist und fragt nach ob Sie lieber das gesamte Paket oder nur den einzelnen Ordner umbenennen möchten. Wählen Sie hier die Option "Rename package".





Nun sind die ersten Vorbereitungen abgeschlossen und die eigentliche Implementierung kann beginnen.

## Datenquelle für Sync Job vorbereiten

Zunächst benötigen wir einen Ort, an dem wir unsere Daten ablegen können. Normalerweise würde man an dieser Stelle mit einer Datenbank arbeiten oder andere REST/SOAP APIs als Datenquelle heranziehen. Der Einfachheit halber wird hier jedoch eine einfache Klasse implementiert, welche die benötigten Daten im Speicher hält.

Erstellen Sie hierfür zunächst das Paket *com.tenfoldsecurity.gc.tutorial.data* und legen darin eine Klasse *DataStore* an.

Die Klasse wird als einfacher Singleton realisiert, weswegen eine statische Variable für die Singletoninstanz sowie eine Zugriffsmethode erstellt werden:

```
1 private static final DataStore instance = new DataStore();
```

```
1 public static DataStore getInstance()
2 {
3     return instance;
4 }
```

Dieses API definiert lediglich einen Default-Kontext sowie die Berechtigungen "Administrator" und "User". Da sich dies niemals ändert, kann dies einfach in einem static-Block initialisiert werden, was dann wie folgt aussieht:

```
1 private static final Context DEFAULT_CONTEXT;
2
3 private static final List<Privilege> PRIVILEGES;
4
5 static
6 {
7     DEFAULT_CONTEXT = new Context();
8     DEFAULT_CONTEXT.setId("1");
9     DEFAULT_CONTEXT.setName("DEFAULT");
10    DEFAULT_CONTEXT.setShortName("DEFAULT");
11    DEFAULT_CONTEXT.setValidityEditable(false);
12    DEFAULT_CONTEXT.setOptions(new ArrayList<>());
13
14    Privilege adminPrivilege = new Privilege();
15    adminPrivilege.setName("Administrator");
16    adminPrivilege.setId("1");
17    adminPrivilege.setAssignable(true);
18    adminPrivilege.setContext(DEFAULT_CONTEXT);
```

```

19     adminPrivilege.setDescription("Administrator privileges for the whole
20     system.");
21     adminPrivilege.setShortName("adm");
22
23     Privilege userPrivilege = new Privilege();
24     userPrivilege.setName("User");
25     userPrivilege.setId("2");
26     userPrivilege.setAssignable(true);
27     userPrivilege.setContext(DEFAULT_CONTEXT);
28     userPrivilege.setDescription("Normal user privileges.");
29     userPrivilege.setShortName("usr");
30
31     PRIVILEGES = Arrays.asList(adminPrivilege, userPrivilege);

```

Dann werden auch noch die entsprechenden Zugriffsmethoden benötigt.

```

1  public synchronized List<Context> getAllContexts()
2  {
3      return Arrays.asList(DEFAULT_CONTEXT);
4  }
5
6  public synchronized List<Privilege> getAllPrivileges()
7  {
8      return new ArrayList<>(PRIVILEGES);
9  }

```

Mit diesen Methoden können dann bereits die ersten API-Methoden implementiert werden. Öffnen Sie hierfür zunächst die Klasse `com.tenfoldsecurity.gc.tutorial1.api.impl.ContextsApiServiceImpl`. Dort benötigen Sie zunächst eine Referenz auf unseren `DataStore`, fügen Sie also eine Instanzvariable zur Klasse hinzu:

```

1  private DataStore dataStore = DataStore.getInstance();

```

### Fehlende Imports

Da die Klasse `DataStore` in einem anderen Paket abgelegt ist als die Klasse `ContextsApiServiceImpl`, ist die Klasse nur mit Ihrem Namen noch nicht verfügbar. Es fehlt daher der Import. Diesen können Sie in IntelliJ einfach mit ALT-Enter einfügen lassen. Andere Entwicklungsumgebungen bieten diese Funktion ebenso.

Die Stub-Methode ändern Sie dann wie folgt:

```

1  public Response getAllContexts(SecurityContext securityContext)
2      throws NotFoundException
3  {
4      return Response.ok().entity(dataStore.getAllContexts()).build();
5  }

```

Analog verfahren Sie mit der Klasse `com.tenfoldsecurity.gc.tutorial1.api.impl.PrivilegesApiServiceImpl`. Fügen Sie dort ebenso eine Referenz auf den `DataStore` ein und ändern die Stub-Methode so, dass sie die Berechtigungen zurückliefert.

```
1 return Response.ok().entity(dataStore.getAllPrivileges()).build();
```

Um den Datenabgleich erfolgreich durchführen zu können, müssen noch zwei weitere Stub-Methoden implementiert werden. Dafür öffnen Sie die Klasse `com.tenfoldsecurity.gc.tutorial1.api.impl.UsersApiServiceImpl` und lassen die Methoden `getAllUserOptions()` und `getAllUsers()` leere Listen zurückliefern:

```
1 public Response getAllUserOptions(SecurityContext securityContext)
2     throws NotFoundException
3 {
4     return Response.ok().entity(Collections.emptyList()).build();
5 }
6
7 public Response getAllUsers(SecurityContext securityContext)
8     throws NotFoundException
9 {
10    return Response.ok().entity(Collections.emptyList()).build();
11 }
```

Bauen Sie nun die Anwendung, deployen Sie sie auf einem Server Ihrer Wahl und richten, wie in den vorangegangenen Kapiteln, eine Ressource mit dem Generic Connector Plugin ein. Wenn Sie daraufhin den Job "Generic Connector Plugin - Sync" starten, werden Sie feststellen, dass in der vom Plugin erstellten Ressource nun die Berechtigungen "User" und "Administrator" sowie ein Default-Kontext vorhanden sind. Im nächsten Schritt wird die Datenquelle erweitert, so dass Benutzerdaten gespeichert werden können.

### Datenquelle erweitern: Benutzer speichern

Aktuell werden die Benutzerdaten noch nicht gespeichert. Wenn Sie einer Person in tenfold die neue Ressource zuordnen, wird Ihnen tenfold diese Aktion mit einem Fehler quittieren. Dies liegt daran, dass im Stub eine einfache "OK"-Nachricht geantwortet wird. Dafür muss zunächst der `DataStore` erweitert werden. Fügen Sie dafür dem `DataStore` eine weitere Instanzvariable hinzu:

```
1 private final List<User> users = new ArrayList<>();
```

Jetzt fügen Sie dem `DataStore` noch eine Methode hinzu, welche neue Benutzer in diese Liste speichert:

```
1 public synchronized User insertUser(User user)
2 {
3     if(user.getOptions() == null)
4         user.setOptions(new ArrayList<>());
5     if(user.getPrivileges() == null)
6         user.setPrivileges(new ArrayList<>());
7     user.setId(UUID.randomUUID().toString());
8     users.add(user);
9     return user;
```

```
10 }

```

Dem Benutzer wird hier eine neue ID vergeben. Diese wird mittels der Java-Klasse UUID erzeugt und im Objekt gespeichert. Zusätzlich werden die Listen der Benutzeroptionen- und Berechtigungen mit leeren Listen initialisiert, sollten diese *null* sein.

Abschließend wird der Benutzer in die Liste der Benutzer eingefügt und zurückgegeben. Mit dieser Funktion im Datenspeicher können Sie nun die API-Methode `createUser()` realisieren. Fügen Sie dafür zunächst einen Verweis auf den `DataStore` in die Klasse `com.tenfoldsecurity.gc.tutorial1.api.impl.UsersApiServiceImpl` ein.

```
1 private DataStore dataStore = DataStore.getInstance();

```

Mit der Referenz auf den `DataStore` lässt sich nun sehr einfach die `createUser()` Methode realisieren:

```
1 public Response createUser(User user, SecurityContext securityContext)
2     throws NotFoundException
3 {
4     user = dataStore.insertUser(user);
5     return Response.ok().entity(user).build();
6 }

```

Wenn Sie die Anwendung nun erneut bauen und wieder auf Ihrem Webserver deployen, werden Sie feststellen, dass eine neue Zuordnung der Ressource an Personen in tenfold funktioniert. Wenn Sie jedoch den Datenabgleich erneut ausführen, werden alle erstellten Zuordnungen wieder entfernt.

Dies liegt daran, dass die Methode `getAllUsers()` noch nicht implementiert wurde. Dies lässt sich jedoch leicht ändern. Fügen Sie zunächst eine neue Methode in den `DataStore` hinzu, mit welcher die Benutzer abgefragt werden können:

```
1 public synchronized List<User> getAllUsers()
2 {
3     return new ArrayList<>(users);
4 }

```

Hier wird eine Kopie der Liste zurückgegeben, welche der `DataStore` im Speicher behält. Dies verhindert ein unbeabsichtigtes Ändern der Userliste, außerhalb des `DataStores`. Mithilfe dieser Methode kann nun die API-Methode `getAllUsers()` sehr einfach implementiert werden. Benutzen Sie einfach die vorhin erstellte Referenz des `DataStores` in der Klasse `UsersApiServiceImpl` und liefern diese zurück.

```
1 public Response getAllUsers(SecurityContext securityContext)
2     throws NotFoundException
3 {
4     return Response.ok().entity(dataStore.getAllUsers()).build();
5 }

```

Nun liefert das API alle gespeicherten User zurück und der Datenabgleich verwirft die Zuordnungen nicht mehr (solange der Prozess läuft). Damit ist bereits ein guter Teil der notwendigen Funktionalität implementiert. Aktuell kann das API jedoch nur neue Benutzer anlegen. Es ist nicht möglich bestehende Benutzerdaten anzupassen oder diese wieder zu löschen.

Im folgenden Abschnitt wird das API um diese Möglichkeiten erweitert.

## Datenquelle erweitern: Bestehende Benutzer verwalten

Die Benutzerdaten werden in einer Liste verwaltet. Für Anfragen zur Anpassung von Benutzern liefert der Generic Connector zwar die zuvor generierte ID mit, man muss jedoch zuerst die entsprechende Benutzerinstanz anhand der ID aus der Liste finden. Bevor das API also erweitert werden kann, benötigt man entsprechende Filtermethoden. Diese werden in der Klasse `com.tenfoldsecurity.gc.tutorial.data.UserFunctions` implementiert. Die Implementierung sieht wie folgt aus:

```

1  public class UserFunctions
2  {
3      public static final Comparator<User> COMPARE_BY_ID =
        Comparator.nullsFirst(Comparator.comparing(User::getId,
        String::compareToIgnoreCase));
4
5      public static final Predicate<User> equals(final User user)
6      {
7          return u -> COMPARE_BY_ID.compare(u, user) == 0;
8      }
9
10     public static final Predicate<User> hasId(String id)
11     {
12         return u -> u.getId().equalsIgnoreCase(id);
13     }
14 }
```

Zunächst verfügt diese Klasse über eine Instanz der Klasse `java.util.Comparator`, mit welcher Benutzer verglichen werden können. Dieser Vergleich findet über die ID des Benutzers statt. Bei Gleichheit der ID zweier Benutzerobjekte erhält man mit der `compareTo`-Methode dieses Comparators den Wert 0. Zusätzlich sind zwei Methoden implementiert, die Instanzen der `java.util.function.Predicate` zurückliefern, die wiederum benutzt werden können, um zu prüfen, ob zwei Benutzer anhand der ID ident sind und ob ein Benutzer eine bestimmte übergebene ID besitzt. Mit diesen Helfermethoden wird nun im `DataStore` eine Methode implementiert, welche es erlaubt ein Benutzerobjekt in der Liste anhand eines an die API übergebenen Benutzers zu finden.

In der `DataStore` Klasse fügen Sie also folgende Methode hinzu:

```

1  private User matchToExisting(User user) throws NotFoundException
2  {
3      return users.stream()
4          .filter(UserFunctions.equals(user))
5          .findFirst()
6          .orElseThrow(NotFoundExceptions::userWithIdNotFound);
7  }
```

Diese Methode benutzt das Java Stream API und die zuvor erstellte HelferMethode `UserFunctions.equals()`, um aus der Liste ein Userobjekt zu finden, welches dieselbe ID hat, wie der übergebene User. Wird kein passendes Objekt gefunden, wirft die Methode eine `NotFoundException`, welche von den REST-Funktionen in einen 404 Fehler umgewandelt wird.

Zu diesem Zeitpunkt fehlt jedoch noch die `Exceptions`-Klasse, welche an dieser Stelle benutzt wird, um eine passende Exception mit entsprechender Fehlermeldung zu erzeugen. Erstellen Sie daher die Klasse `com.tenfoldsecurity.gc.tutorial1.exceptions.NotFoundExceptions` und fügen dieser folgende Methode hinzu:

```

1  public static NotFoundException userWithIdNotFound()
2  {
3      return new NotFoundException(404, "User with given ID not found.");
4  }

```

Mittels ALT-Enter lässt sich der fehlende Import in der Klasse DataStore, wie auch zuvor, hinzufügen. Nun existiert eine Methode, welche es erlaubt, aus der Liste der vorhandenen User einen passenden User zu finden. Mit dieser Methode kann nun die Funktionalität zur Änderung oder Löschung von Benutzern realisiert werden.

Folgende Methoden fügen Sie also der DataStore Klasse hinzu:

```

1  public synchronized User updateUser(User user) throws NotFoundException
2  {
3      User existingUser = matchToExisting(user);
4      int existingIndex = users.indexOf(existingUser);
5      users.set(existingIndex, user);
6      return user;
7  }
8
9  public synchronized void deleteUser(User user) throws NotFoundException
10 {
11     User existingUser = matchToExisting(user);
12     users.remove(existingUser);
13 }

```

Diese Methoden benutzen die vorhergehend entwickelte Methode, um das Benutzerobjekt aus der Liste zu filtern. Die Methode updateUser() ermittelt daraufhin den Index des bestehenden Benutzerobjektes und ersetzt es mit dem neuen Userobjekt. Die deleteUser()-Methode entfernt den Benutzer einfach aus der Liste. In der Klasse UsersApiServiceImpl kann nun die deleteUser() API-Methode entwickelt werden. Die vorhandene Stub-Methode wird dabei einfach durch folgende Methode ersetzt:

```

1  public Response deleteUser(String id, SecurityContext securityContext)
2      throws NotFoundException
3  {
4      User user = datastore.getUserById(id);
5      datastore.deleteUser(user);
6      return Response.ok().entity(new
7      ApiResponseMessage(ApiResponseMessage.OK, "Ok")).build();

```

Da der Generic Connector keinen bestimmten Retourwert dieser Methode erwartet, bleibt hier einfach die normale "Ok"-Nachricht bestehen. Danach muss noch die updateUser()-Methode implementiert werden. Diese sieht wie folgt aus:

```

1  public Response updateUser(String id, User user, SecurityContext
2      securityContext)
3      throws NotFoundException
4  {
5      user = datastore.updateUser(user);

```

```

5     return Response.ok().entity(user).build();
6 }

```

Auch hier wird einfach der DataStore herangezogen, um den Benutzer anzupassen. Daraufhin wird das Benutzerobjekt zurückgeliefert. Wenn Sie das Projekt nun wieder bauen und erneut deployen, können Sie nun wie erwartet die Ressourcenzuordnungen löschen und auch Benutzerdaten werden bei Personenänderungen korrekt vom API verarbeitet.

Als letzte Aufgabe für diesen Abschnitt fügen wir dem DataStore noch eine Methode hinzu, mit welcher einfach ein Benutzerobjekt mit einer bestimmten ID ermittelt werden kann. Diese Methode wird für den folgenden Abschnitt gebraucht.

```

1 public synchronized User getUserById(String id) throws NotFoundException
2 {
3     return users.stream()
4         .filter(UserFunctions.hasId(id))
5         .findFirst()
6         .orElseThrow(NotFoundExceptions::userWithIdNotFound);
7 }

```

Nun müssen nur mehr die API-Methoden zur Verarbeitung der Berechtigungen implementiert werden. Dies erfolgt im nächsten Abschnitt.

## Berechtigungen verwalten

Wenn Sie die API-Methoden `removeUserPrivileges()` oder `updateUserPrivileges()` betrachten, so fällt auf, dass im Gegensatz zur Methode `updateUser()` hier kein ganzes Benutzerobjekt mit ID übergeben wird, sondern lediglich die ID des Benutzers mit den entsprechenden Berechtigungsdaten.

```

1 public Response removeUserPrivileges(String id, List<PrivilegeAssignment>
2     privileges, SecurityContext securityContext)
3     throws NotFoundException
4 {
5     // TODO: Implement Logic
6     return Response.ok().entity(new
7     ApiResponseMessage(ApiResponseMessage.OK, "magic!")).build();
8 }

```

Daher wurde auch im letzten Abschnitt die Methode `getUserById()` in den DataStore hinzugefügt. Diese kann nun verwendet werden, um das passende Benutzerobjekt mit der übergebenen ID zu ermitteln.

Zur einfacheren Abarbeitung der im Benutzerobjekt enthaltenen Berechtigungen werden nun zwei weitere Hilfsklassen entwickelt. Zunächst eine Klasse analog zur `UserFunctions` Klasse welche Filtermethoden für eine Liste von Berechtigungen bereitstellt. Diese Klasse erstellen Sie im Paket `com.tenfoldsecurity.gc.tutorial1.data` und nennen Sie `PrivilegeAssignmentFunctions`.

```

1 public class PrivilegeAssignmentFunctions
2 {
3     public static final Comparator<PrivilegeAssignment> COMPARE_BY_ID
4         =
5     Comparator.nullsFirst(Comparator.comparing(PrivilegeAssignment::getId,
6     String::compareToIgnoreCase));
7 }

```

```

6      public static final Predicate<PrivilegeAssignment> equals(final
PrivilegeAssignment user)
7      {
8          return u -> COMPARE_BY_ID.compare(u, user) == 0;
9      }
10
11     public static final Predicate<PrivilegeAssignment> hasId(String id)
12     {
13         return u-> u.getId().equalsIgnoreCase(id);
14     }
15
16     public static final boolean isNew(PrivilegeAssignment assignment)
17     {
18         return assignment.getId() == null;
19     }
20
21     public static final boolean isExisting(PrivilegeAssignment assignment)
22     {
23         return assignment.getId() != null;
24     }
25 }

```

Auch hier findet sich wieder ein Comparator, welcher Berechtigungszuordnungen anhand der ID vergleicht. Desweiteren gibt es mehrere Methoden, mit welcher Listen von Berechtigungen gefiltert werden können. Damit ist der erste Baustein gelegt. Jetzt wird noch eine Klasse implementiert, welche die Berechtigungen in einem Benutzerobjekt verarbeitet, sprich hinzufügt, verändert oder entfernt. Erstellen Sie eine neue Klasse *PrivilegeAssignmentUpdater* im Paket *com.tenfoldsecurity.gc.tutorial1.data*. Zunächst benötigt die Klasse ein Benutzerobjekt, auf welchem sie arbeiten kann. Fügen Sie daher eine Klassenvariable und einen dazu passenden Konstruktor in die Klasse ein.

```

1      private User targetUser;
2
3      public PrivilegeAssignmentUpdater(User targetUser)
4      {
5          this.targetUser = targetUser;
6      }

```

Daraufhin erstellen Sie eine neue Methode zum Hinzufügen neuer Berechtigungen.

```

1      public void addAssignment(PrivilegeAssignment privilegeAssignment)
2      {
3          privilegeAssignment.setId(UUID.randomUUID().toString());
4          if(targetUser.getPrivileges() == null)
5              targetUser.setPrivileges(new ArrayList<>());
6          targetUser.getPrivileges().add(privilegeAssignment);
7      }

```

Beachten Sie, dass hier der neuen Berechtigungszuordnung eine ID vergeben wird. Diese ID wird immer dann benötigt, wenn ein Benutzer eine Berechtigung mehrfach, mit unterschiedlichen Eigenschaften, zugeordnet haben kann. Es gibt zum Beispiel System in welchen ein Benutzer eine einzelne Berechtigung mehrfach mit unterschiedlichen Gültigkeitszeiträumen zugeordnet haben kann. Außerdem kann Ihr API weitere



Eigenschaften definieren, die Berechtigungen zugeordnet werden können. In diesem Fall ist es zwingend erforderlich, dass Sie dem Generic Connector eine eindeutige ID für die Berechtigungszuordnung zurückliefern.

Diese simple Implementierung des APIs hat zwar keine solchen Zusatzeigenschaften, jedoch macht es eine ID einfacher, hinzugefügte oder veränderte Berechtigungen später zu erkennen.

Um eine bestehende Berechtigung anhand eines übergebenen Berechtigungsobjektes finden zu können, wird auch hier eine Methode benötigt, welche die entsprechende Instanz aus der Liste zuordnen kann. Diese sieht wie folgt aus:

```

1  private PrivilegeAssignment matchToExisting(final PrivilegeAssignment
2  privilegeAssignment)
3  {
4      return targetUser.getPrivileges()
5          .stream()
6          .filter(PrivilegeAssignmentFunctions.hasId(privilegeAssignment
7          .getId()))
8          .findFirst()
9          .orElse(null);
10 }
```

Im Gegensatz zur Methode für Benutzer wird hier einfach null zurückgeliefert, statt eine Exception zu werfen. Nicht erkannte Berechtigungen werden daher einfach ignoriert.

Mithilfe dieser Methode lassen sich nun die eigentlichen Methoden zur Aktualisierung und Entfernung der Berechtigungen schreiben.

```

1  public void updateAssignment(PrivilegeAssignment privilegeAssignment)
2  {
3      PrivilegeAssignment existingAssignment =
4      matchToExisting(privilegeAssignment);
5      if(existingAssignment != null)
6      {
7          int index =
8          targetUser.getPrivileges().indexOf(existingAssignment);
9          targetUser.getPrivileges().set(index, privilegeAssignment);
10     }
11 }
12
13 public void removeAssignment(PrivilegeAssignment privilegeAssignment)
14 {
15     PrivilegeAssignment existingAssignment =
16     matchToExisting(privilegeAssignment);
17     if(existingAssignment != null)
18         targetUser.getPrivileges().remove(existingAssignment);
19 }
```

Damit ist nun der letzte Baustein gelegt, mit welchem die letzten noch notwendigen Methoden des APIs realisiert werden können. Öffnen Sie daher noch einmal die Klasse *UsersApiServiceImpl* und ändern Sie die *updateUserPrivileges()* wie folgt:

```

1  public Response updateUserPrivileges(String id, List<PrivilegeAssignment>
2      privileges, SecurityContext securityContext)
3      throws NotFoundException
4  {
5      User user = datastore.getUserById(id);
6      PrivilegeAssignmentUpdater updater = new
7      PrivilegeAssignmentUpdater(user);
8      privileges.stream()
9          .filter(PrivilegeAssignmentFunctions::isExisting)
10         .forEach(updater::updateAssignment);
11
12     privileges.stream()
13         .filter(PrivilegeAssignmentFunctions::isNew)
14         .forEach(updater::addAssignment);
15     return Response.ok().entity(user).build();
16 }

```

Zunächst wird der DataStore aufgerufen, um den Benutzer mit der übergebenen ID zu ermitteln. Danach wird eine Instanz der eben erstellten Klasse *PrivilegeAssignmentUpdater* erstellt und der gefundene Benutzer wird übergeben. Danach werden die neuen Berechtigungen ermittelt und an den Updater übergeben damit diese in das Benutzerobjekt eingefügt werden. Anschließend werden noch die veränderten Berechtigungen übergeben und im Benutzerobjekt ersetzt. Abschließend wird das Benutzerobjekt mit den veränderten Berechtigungen an den Generic Connector zurückgesendet.

Wie erwähnt, unterstützt dieses API im Moment keine Eigenschaften für Berechtigungen. Daher sollten eigentlich keine Aktualisierungen durchgeführt werden, sondern immer nur Neuanlagen. Sollten Sie jedoch selber versuchen wollen Eigenschaften hinzuzufügen, so unterstützt die API-Funktion nun bereits die Möglichkeit diese Eigenschaften zu aktualisieren.

Zu Guter Letzt muss noch die Methode *removeUserPrivileges()* implementiert werden.

```

1  public Response removeUserPrivileges(String id, List<PrivilegeAssignment>
2      privileges, SecurityContext securityContext)
3      throws NotFoundException
4  {
5      User user = datastore.getUserById(id);
6      PrivilegeAssignmentUpdater updater = new
7      PrivilegeAssignmentUpdater(user);
8      privileges.stream()
9          .forEach(updater::removeAssignment);
10     return Response.ok().entity(user).build();
11 }

```

Wie zuvor wird auch hier der passende Benutzer zur übergebenen ID ermittelt und an ein Updaterobjekt übergeben. Anschließend werden die übergebenen Berechtigungen, eine nach der anderen, aus dem Benutzerobjekt entfernt. Abschließend wird das veränderte Benutzerobjekt an den Generic Connector zurückgeliefert.

Damit ist diese einfache Implementierung des APIs abgeschlossen.

## Tutorial 1 Beispiel API

Sie können die in diesem Kapitel erstellte API auch auf der Plugin Seite des Generic Connectors herunterladen. Der gesamte Quelltext ist dort fertig kompilierbar vorhanden und mit zusätzlichen Kommentaren versehen, welche in diesem Kapitel des Platzes halber ausgelassen wurden. Sollten bei der Implementierung, welche in dieser Anleitung beschrieben wurde, Probleme auftauchen, ist es ein guter erster Schritt den Ihren Code mit dem Code aus dieser Anwendung zu vergleichen. Der gesamte Code in diesem Beispiel basiert auf dieser Anwendung.

### Weitere Schritte

Das API ist nun fertig implementiert und funktionsfähig. Als gute Übung erweist es sich, dieses API um Benutzeroptionen und Kontextoptionen zu erweitern. Obwohl diese aktuell nicht definiert sind, ist eine Verarbeitung dieser bereits fertig implementiert. Sie können also einfach die API-Methoden `getAllPrivileges()`, `getAllContexts()` und `getAllUserOptions()` erweitern und verfolgen wie sich daraufhin das Verhalten der Generic Connector Ressource in tenfold ändert.

Eine weitere gute Übung wäre es, eine echte Datenquelle, wie zum Beispiel eine Datenbank, an diese Implementierung anzubinden, damit die Daten nicht bei jedem Neustart verloren gehen.

In den folgenden Kapiteln finden Sie eine Erklärung der weiteren Beispielanwendungen, welche Sie auf der Einstellungsmaske des Generic Connector Plugins herunterladen können. Diese implementieren, durch das Speichern der Daten in einer Datenbank, eine realitätsnähere Anwendung.

### Beispielanwendung: Sample 01

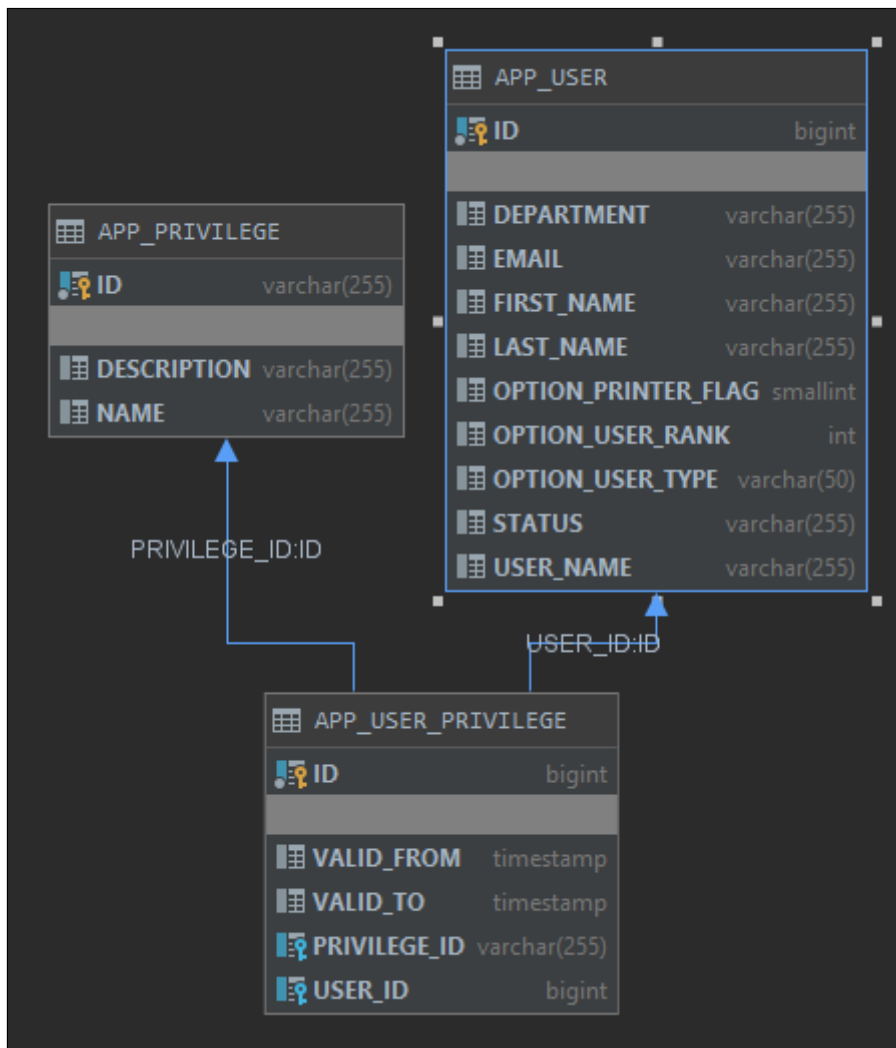
Nachdem Sie im vorhergehenden Kapitel damit vertraut gemacht wurden, wie Sie ein einfaches API realisieren können, folgt in diesem Kapitel nun eine Beispielanwendung mit mehr Realitätsbezug. Dieses API definiert sowohl Optionen für die Ressource als auch Gültigkeitszeitbereiche für Berechtigungen. Außerdem werden die Daten in einer Datenbank abgespeichert. Bei dieser Datenbank handelt es sich um eine mitgelieferte In-Process-Datenbank (H2: <https://www.h2database.com/>), die Installation eines eigenen Datenbankservers ist also nicht notwendig.

Damit wird der Zugriff auf eine Anwendung simuliert, auf welche tenfold mittels des Generic Connectors Zugriffsberechtigungen vergeben kann.

Wenn Sie die Anwendung heruntergeladen haben, finden Sie in der ZIP-Datei sowohl den Quellcode des APIs, als auch eine fertig gebaute Anwendung, welche Sie auf einem beliebigen Server deployen können.

Da die Grundzüge der Implementierung einer REST-Schnittstelle bereits im vorhergehenden Kapitel erläutert wurden, wird in diesem Kapitel darauf verzichtet. Stattdessen werden die speziellen Eigenschaften der Beispielanwendung beleuchtet, beginnend beim Datenmodell im folgenden Abschnitt.

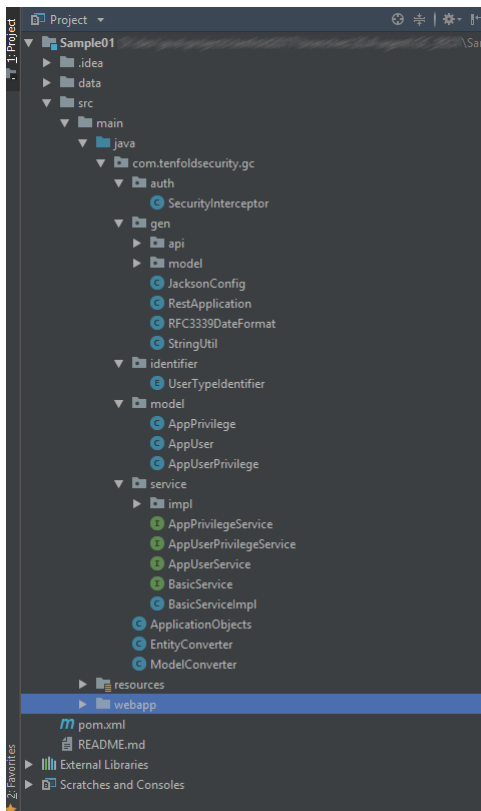
## Datenmodell



Das Datenmodell ist sehr simpel und besteht nur aus drei Tabellen: *APP\_USER*, *APP\_USER\_PRIVILEGE* und *APP\_PRIVILEGE*.

Tabelle	Beschreibung
APP_USER	Enthält die vom API verarbeiteten und gespeicherten Benutzerdaten, wie etwa Vor- und Nachnamen.
APP_PRIVILEGE	Enthält alle vorhandenen Berechtigungen des APIs. Enthält Namen und Beschreibungen der Berechtigungen, die an tenfold übermittelt werden.
APP_USER_PRIVILEGE	Enthält alle Zuordnungen von Berechtigungen zu den jeweiligen Benutzern. Enthält auch einen Gültigkeitszeitraum der Berechtigungszuweisung.

## Projektaufbau



Betrachtet man den Aufbau dieses Projektes so fällt einem auf, dass, im Gegensatz zur Tutorial-Anwendung, hier keine Trennung zwischen von Swagger generiertem Code und Benutzercode existiert. Alle Quellcode-Dateien befinden sich im Ordner `src/main`. Je nach Einstellungen des Code-Generators lassen sich hier die generierten und zu implementierenden Klassen in verschiedenen Ordnern ablegen oder nicht. In der Tutorial-Anwendung wurde diese Trennung benutzt, um nur den notwendigen Code im Blick zu haben.

Bis auf diese Trennung ist der Aufbau sonst jedoch sehr ähnlich. Das Root-Paket der Anwendung ist `com.tenfoldsecurity.gc`. Hier ein Überblick über die wichtigsten Pakete, ausgehend vom Root-Paket:

Paket	Beschreibung
<b>com.tenfoldsecurity.gc</b>	
.	Allgemeine Hilfsklassen
.auth	Enthält einen Standardinterceptor für die Anmeldelogik des Rest-APIs
.gen.api	Enthält die Klassen und Schnittstellen, welche das API nach außen publizieren.
.gen.api.impl	Enthält die eigentliche Implementierung der einzelnen REST-Operationen
.gen.model	Enthält die Definitionen für die JSON-Datenstrukturen, welche vom API übermittelt werden

Paket	Beschreibung
<b>com.tenfoldsecurity.gc</b>	
.identifizier	Enthält definierte Konstanten.
.model	Enthält eine Abbildung des Datenmodell der Datenbank. (Nicht zu verwechseln mit gen.model)
.service	Schnittstellen für die Datenbankzugriffslogik.
.service.impl	Klassen, welche die Datenbankzugriffe implementieren.

## Datenzugriff

### Datenbankzugriff

Für den Zugriff auf die Datenbank verwendet diese Beispielanwendung die Java Bibliothek Hibernate. Nähere Information dazu finden Sie unter <https://hibernate.org/orm/>.

Die Grundlage des APIs bildet der Zugriff auf die Datenbank. Als Datenbank wird H2 verwendet, welche eine in Java geschriebene In-Process Datenbank ist und daher keinen dedizierten Server benötigt. Die Anwendung ist damit unabhängig lauffähig. Der erste Blick daher sollte in das Paket `com.tenfoldsecurity.model` fallen, welche das im vorhergehenden Abschnitt beschriebene Datenmodell als Java-Klassen abbildet. Sie finden dabei auf allen Klassen und Feldern die entsprechenden Hibernate-Annotationen, um Hibernate mitzuteilen, wie diese Objekte in der Datenbank abgelegt werden müssen.

```

1  @Entity
2  @Table(name = "APP_PRIVILEGE")
3  @NamedQueries({
4      @NamedQuery(name = "AppPrivilege.selectAll",
5                  query = "SELECT p FROM AppPrivilege p")
6  })
7  public class AppPrivilege implements Serializable

```

```

1  @Id
2  @Column(name = "ID")
3  private String id;
4
5  @Basic
6  @Column(name = "NAME")
7  private String name;
8
9  @Basic
10 @Column(name = "DESCRIPTION")
11 private String description;

```

Mit diesen Annotationen ist es einfach möglich, Hibernate eine Instanz der entsprechenden Klasse zu überreichen und Hibernate kann diese automatisch in die Datenbank speichern und aus der Datenbank

entsprechende Listen von Instanzen generieren. Damit erspart man sich die mühevollen Aufgabe, eigene Datenbankzugriffslogik zu implementieren.

Wichtig ist auch, dass diese Datenobjekte eine `equals()`-Methode über die Schlüsselspalte definieren. Dies ist etwas, das im Tutorial nicht möglich war, da die gespeicherten Klassen gleichzeitig die API-Objekte waren, welche nicht verändert werden durften.

```

1  @Override
2  public boolean equals(Object o)
3  {
4      if (this == o) return true;
5      if (o == null || getClass() != o.getClass()) return false;
6      AppPrivilege appPrivilege = (AppPrivilege) o;
7      return Objects.equals(id, appPrivilege.id);
8  }
```

Mit diesen Datenklassen als Grundlage, kann mit Hibernate sehr einfach der Datenzugriff implementiert werden.

Statt im Rest der Anwendung direkt mit Hibernate zu arbeiten wurden im Paket `com.tenfoldsecurity.gc.service` mehrere Schnittstellen definiert, mit denen auf die Datenbankobjekte zugegriffen wird.

Schnittstelle	Funktion
BasicService	Definiert generisch alle grundlegenden CRUD-Operationen für Datenzugriffe. Die anderen Schnittstellen leiten von dieser Schnittstelle ab.
AppPrivilegeService	Filterfunktionen zum Lesen der Berechtigungen.
AppUserPrivilegeService	Filterfunktionen zum Lesen der User-Berechtigungszuordnungen.
AppUserService	Filterfunktionen zum Lesen der Benutzer.

Darüber hinaus gibt es in diesem Paket noch die Klasse `BasicServiceImpl`, welche generisch die CRUD-Operationen für alle Datenobjekte implementiert. Alle anderen Zugriffsservices leiten von dieser Klasse ab. Im darunterliegenden Paket `com.tenfoldsecurity.gc.service.impl` befinden sich die Klassen, welche die einzelnen spezifischen Zugriffsoptionen für die verschiedenen Datenobjekte mittels Hibernate-Queries implementieren.

## Datenkonvertierung

Im Gegensatz zur Tutorial-Anwendung werden hier nicht einfach die an das REST-API übergebenen Objekte gespeichert, sondern, wie im vorangegangenen Kapitel beschrieben, eigene Datenobjekte, welche die Datenbanktabellen repräsentieren.

Daher ist es notwendig zwischen diesen Objekten konvertieren zu können. Im Paket `com.tenfoldsecurity.gc` befinden sich zwei Klassen, welche diese Konvertierung vornehmen.

Klasse	Beschreibung
EntityConverter	Diese Klasse konvertiert REST-API Objekte in Datenbankzugriffsobjekte.

Klasse	Beschreibung
ModelConverter	Diese Klasse konvertiert Datenbankzugriffsobjekte in REST-API Objekte.

Diese Konvertierung von REST-API Objekten zu Datenbankzugriffsobjekten ist dabei größtenteils äußerst unkompliziert. Wie im folgenden Beispiel werden einfach die relevanten Eigenschaften der REST-Klassen in die Datenbankklassen geschaufelt.

```

1  public static AppUser createAppUser(User userDto)
2  {
3      AppUser entity = new AppUser();
4      entity.setDepartment(userDto.getDepartment().getName());
5      entity.setEmail(userDto.getEmail());
6      entity.setFirstName(userDto.getFirstName());
7      entity.setLastName(userDto.getLastName());
8      entity.setStatus(userDto.getStatus().name());
9      entity.setUsername(userDto.getUsername());
10
11     updateUserOptions(entity, userDto.getOptions());
12
13     return entity;
14 }
```

Die Konvertierung der Datenbankklassen in REST-Klassen gestaltet sich hierbei nur unwesentlich komplizierter. Die einzige Hürde besteht hier darin, dass die Datenbankklassen wesentlich flacher gestaltet sind als die REST-Klassen, weswegen mehr Objekte erzeugt werden müssen.

```

1  public static User createUserDto(AppUser entity, List<AppUserPrivilege>
2  appUserPrivileges)
3  {
4      User userDto = new User();
5      userDto.setId(entity.getId().toString());
6      userDto.setDepartment(createDepartmentDto(entity.getDepartment()));
7      userDto.setEmail(entity.getEmail());
8      userDto.setFirstName(entity.getFirstName());
9      userDto.setLastName(entity.getLastName());
10     userDto.setStatus(Status.valueOf(entity.getStatus()));
11     userDto.setUsername(entity.getUsername());
12
13     userDto.setOptions(createUserOptionAssignmentDtos(entity));
14
15     userDto.setPrivileges(createPrivilegeAssignmentsDtos(appUserPrivileges));
16
17     return userDto;
18 }
19
20 public static Department createDepartmentDto(String departmentName)
21 {
22     Department department = new Department();
23     department.setId(departmentName);
24 }
```



```

22     department.setName(departmentName);
23
24     return department;
25 }

```

Da in der REST-Schnittstelle die Abteilung (Department) als eigenes Datenobjekt mit ID und mehreren Feldern definiert ist, so wie es auch der tenfold-Datenstruktur entspricht, in der Datenbank der API jedoch nur ein einfaches Textfeld ist, muss hier ein wenig Zusatzarbeit geleistet werden um das User-Objekt zu befüllen, wie das vorangehende Beispiel zeigt.

Mithilfe dieser Klassen wird nun das REST-API im Paket *com.tenfoldsecurity.gc.gen.api.impl* implementiert.

## API-Implementierung

Die Implementierung des APIs ist hierbei sehr Ähnlich zur Implementierung des Tutorials. Statt jedoch alle Objekte einfach im Speicher zu halten werden die Datenbankzugriffsklassen aus den vorangegangenen Abschnitten verwendet, um die erhaltenen Daten in eine Datenbank zu speichern.

Zuerst ist zu beachten, wie das API Instanzen unserer Datenbankzugriffsklassen erhält. Dies zeigt Ihnen ein Blick in die Klasse *com.tenfoldsecurity.gc.api.impl.UsersApiServiceImpl*.

```

1  @Inject
2  private AppUserService appUserService;
3
4  @Inject
5  private AppPrivilegeService appPrivilegeService;
6
7  @Inject
8  private AppUserPrivilegeService appUserPrivilegeService;

```

Anstatt die Objekte selbst zu instanzieren oder, wie im Tutorial, über das Singleton-Pattern eine Instanz zu holen, werden diese Objekte über das CDI-API erzeugt. Wie Sie sehen, werden hier einfach Felder des Schnittstellentyps in der Klasse deklariert und mit der Annotation `@Inject` versehen. Dies bewirkt, dass wenn der Java Webserver eine Instanz der Klasse *UsersApiServiceImpl* erzeugt, diese Felder auch gleich mit Instanzen der Service-Klassen befüllt werden.

Da es in der Anwendung jeweils nur eine Klasse gibt, welche diese Schnittstellen implementieren kann, weiß das CDI-Framework automatisch, von welchen Klassen Instanzen erzeugt werden müssen, um diese Felder zu befüllen.

Ein weiterer Blick auf die Definition der Klasse selbst zeigt auch eine Neuerung, welche es im Tutorial nicht gab:

```

1  @RequestScoped
2  @Transactional
3  @javax.annotation.Generated(value =
4  "io.swagger.codegen.languages.JavaResteasyServerCodegen", date =
   "2020-03-25T15:02:54.255Z")
   public class UsersApiServiceImpl implements UsersApiService

```

Die Annotation `@Transactional` auf der Klasse sagt dem Java-Server, dass er um jeden Aufruf einer der REST-Operationen eine Datenbanktransaktion spannen muss, so das alles was in einer dieser Methoden passiert immer innerhalb einer Transaktion stattfindet. Im Fehlerfall wird die Transaktion automatisch zurückgerollt. Damit muss man sich nicht explizit selber um Transaktionslogik kümmern.

Der Rest der Implementierung zeigt kaum Neuerungen.

```

1  public Response createUser(User user, SecurityContext securityContext)
2         throws NotFoundException
3  {
4      AppUser entity = EntityConverter.createAppUser(user);
5      entity = appUserService.persist(entity);
6
7      return Response.ok().entity(ModelConverter.createUserDto(entity,
8      appUserPrivilegeService.getByUser(entity))).build();
9  }

```

Es wird einfach mit Hilfe der Datenzugriffsklassen eine Zeile in der Datenbank abgelegt und das erzeugte Objekt wird wieder in ein REST Userobjekt umgewandelt und an den Client zurückgeliefert. Die anderen Methoden verhalten sich analog dazu.

### Einrichtung in tenfold

Sie können die Beispielanwendung einfach neben tenfold deployen. Kopieren Sie dazu einfach die .war Datei welche sich in der heruntergeladenen ZIP-Datei befindet in das Verzeichnis server/standalone/deployments Ihrer tenfold Installation.

Beim ersten Deployment der Anwendung wird automatisch die mitgelieferte H2 Datenbank initialisiert.

Danach ist das API unter <https://ihr-server/sample1/v1> abrufbar.

Sie können das API nun in tenfold konfigurieren und testen. Erstellen Sie dafür zunächst neue Zugangsdaten vom Typ "Generic Connector" unter Provisioning > Zugangsdaten. Befüllen Sie die Felder mit folgenden Werten:

Feld	Wert
Name	Generic Connector Sample 01
Server URL	<a href="https://localhost/sample1/v1">https://localhost/sample1/v1</a>
Benutzername	tenfold
Passwort	tenfold
Passwort bestätigen	tenfold

Speichern Sie die Zugangsdaten ab und richten anschließend eine neue Generic Connector Verbindung unter Provisioning > Plugins > Generic Connector ein.

Tragen Sie im Kartereiter Allgemein folgende Werte ein (belassen Sie die Standardeinstellung nicht aufgeführter Felder bei):

Feld	Wert
Name	Generic Connector Sample 1
Zugangsdaten	Wählen Sie "Generic Connector Sample 1" aus dem Drop Down Menü aus.
Passwortrichtlinie	Wählen Sie eine beliebige Richtlinie aus.

Im Kartereiter "Benutzerzuordnung" wurde bereits eine passende Regel angelegt, welche die Benutzer mit dem Benutzernamen zuordnet. Lassen Sie diese einfach bestehen.

Zuletzt legen Sie noch im Karteireiter Synchronisierung eine neue Request-Quelle "GC Sample 1" an und wählen diese aus. Belassen Sie alle anderen Einstellungen wie sie sind. Speichern Sie nun die Verbindung.

### Einstellungen

Eine Detaillierte Beschreibung zu allen Einstellungen finden Sie unter [Plugin-Einstellungen](#) (see page 732)

Unter Ressourcen > Verwaltung sollten Sie nun eine neue Ressource mit dem Namen "Generic Connector Sample 1" finden. Lassen Sie nun noch einmal den Job "Generic Connector Plugin - Sync" laufen (siehe [Jobs](#) (see page 443)). Die Ressource sollte nun alle Optionen und Berechtigungen des APIs importiert haben.

Sie können diese Ressource nun wie gewohnt Personen zuordnen, entziehen und die Berechtigungen/Optionen ändern, um die Personen im API zu verarbeiten.

## 14.11 macmon

Mit dem macmon-Plugin können über tenfold gewisse Funktionen der macmon-BYOD Dienste angesprochen werden.

### 14.11.1 Verwendung

Nachdem das Plugin eingerichtet wurde, (näheres unter [Plugin-Konfiguration](#) (see page 799)) stehen in tenfold zwei neue Ressourcen zur Verfügung, *BYOD Access* und *BYOD Device*. Durch die Zuordnung der Ressource *BYOD Access*, erhält die Person Zugriff auf die BYOD Webkonsole von macmon. Mittels dieser Webkonsole, kann die Person Geräte einrichten, mit welchen die Person über das Netzwerk auf das Internet zugreifen darf.

Sobald eine Person über die Webkonsole von macmon BYOD-Geräte eingerichtet hat, werden diese über einen regelmäßigen Datenabgleich mit macmon von tenfold aufgegriffen. Jedes Gerät erzeugt dabei eine Zuordnung der Ressource *BYOD Device* zu einer Person. Damit erhält man in tenfold eine Übersicht über alle Geräte welche Personen registriert haben.

Diese Geräte können im Anschluss auch wieder entfernt werden, indem die Zuordnung der jeweiligen Device-Ressource der entsprechenden Person entzogen wird. Damit lässt sich zum Beispiel sehr einfach verhindern, dass nach dem Austreten einer Person aus dem Unternehmen Zugänge in macmon, durch vergessene Löschungen, offen bleiben.

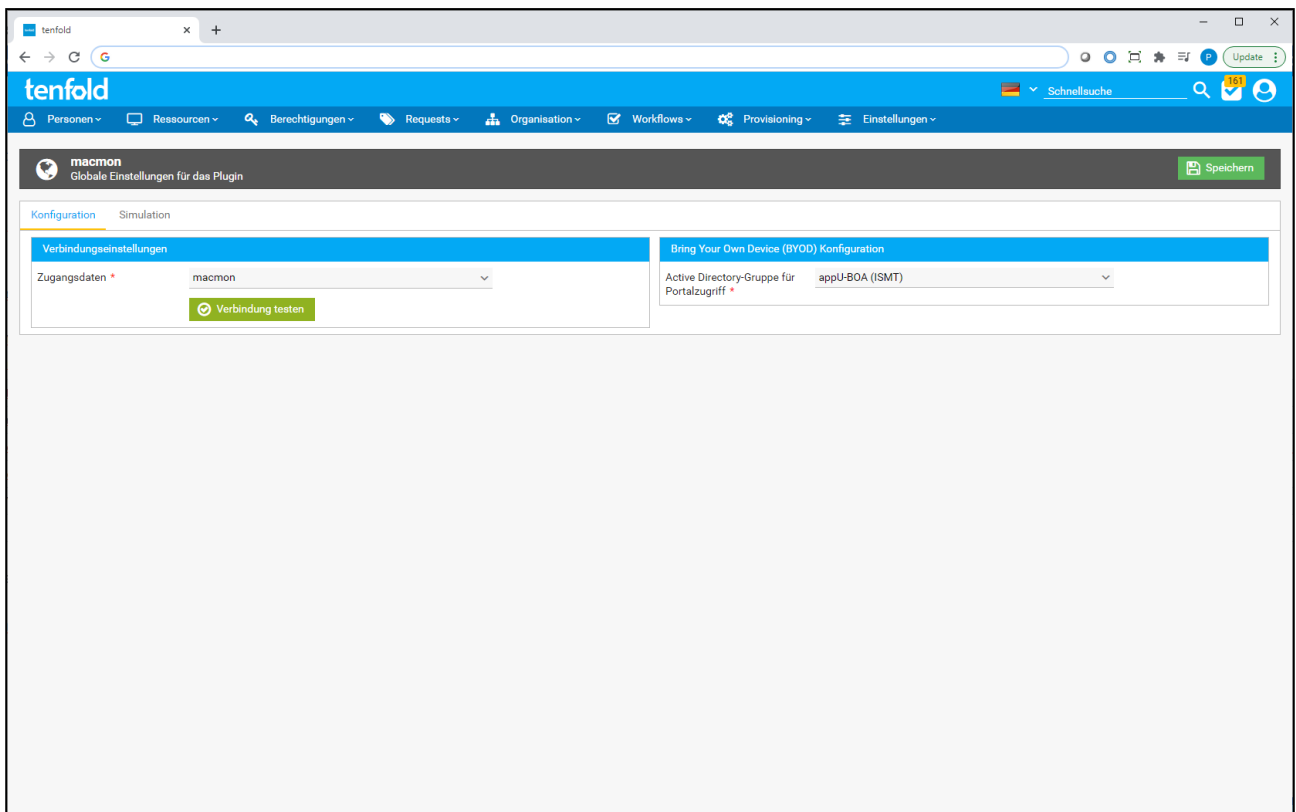
### BYOD-Geräte registrieren

Das Hinzufügen von Geräten im Self-Service oder Administrator-Modus von tenfold wird nicht unterstützt. Die Registrierung von Geräten findet ausschließlich über die Web-Konsole von macmon statt.

### 14.11.2 Plugin-Konfiguration

Die Einstellungen des macmon-Plugins finden Sie im Menü unter Provisioning > Plugins > macmon.

## Allgemeine Einstellungen



Um das macmon-Plugin verwenden zu können, müssen zuerst folgende Einstellungen im Karteireiter "Konfiguration" getätigt werden können.

Einstellung	Beschreibung
<b>Verbindungseinstellungen</b>	
Zugangsdaten	Wählen Sie hier die Zugangsdaten aus, mit welchen tenfold eine Verbindung zum macmon-Dienst herstellen soll. Diese Zugangsdaten enthalten nicht nur Benutzername und Passwort, sondern auch den URL für den Dienst. Näheres unter <a href="#">Zugangsdaten</a> (see page 802).
Verbindung testen	Mit dieser Schaltfläche kann geprüft werden ob mit den angegebenen Zugangsdaten eine Verbindung zum macmon-Dienst hergestellt werden kann.
<b>Bring Your Own Device (BYOD) Konfiguration</b>	
Active Directory-Gruppe	Mit dieser Einstellung wird festgelegt, welche Active Directory-Gruppe für den Zugriff auf die macmon-BYOD Webkonsole notwendig ist.