Mainzelliste – Konfigurationshandbuch

Version 1.7.0 vom 09.07.2018

von A. Borg[1] und M. Lablans[2]

[1] Universitätsmedizin der Johannes Gutenberg-Universität Mainz
[2] Deutsches Krebsforschungszentrum, Heidelberg

Kontakt: info@mainzelliste.de

Versionshistorie

Version	Datum	Änderung/Status	Autor
0.1	15.03.2013	Initiale Version nach aktuellem Implementierungsstand	Andreas Borg
1.0	05.05.2013	Feinschliff für Freigabe	Martin Lablans
1.1	12.07.2013	Information zu mitgelieferter Beispielkonfiguration; Kleinere Korrekturen	Andreas Borg, Maximilian Ataian
1.2	08.11.2013	Ergänzung um Parameter „sessionTimeout“	Andreas Borg
1.3	03.02.2014	Listen mit Tokentypen und Berechtigungen, Ergänzung Callback-Format	Andreas Borg
1.3.1	11.04.2014	Liste mit reservierten Wörtern	Andreas Borg
1.3.2	17.04.2014	Paketnamen unter „field.{name}.comparator“ korrigiert	Andreas Borg
1.4.0	20.02.2015	Ergänzungen zu Version 1.4.0	Andreas Borg
1.5.0	23.12.2015	Ergänzungen zu Version 1.5.0	Andreas Borg
1.6.0	30.01.2017	Ergänzungen zu Version 1.6.0	Andreas Borg
1.7.0	09.07.2018	Ergänzung ExternalIdGenerator	Andreas Borg, Martin Lablans

1. Allgemeines

Die Konfiguration der Patientenliste erfolgt über eine Textdatei im Format, das von java.util.Properties.load akzeptiert wird. Im Wesentlichen heißt das:

- Kommentarzeilen beginnen mit ‚#‘.
- Konfigurationsparameter werden als Schlüssel-Wert-Paare notiert, getrennt durch „:“ oder „=“.

Für Details sei auf die Java-API-Dokumentation verwiesen. In Abweichung zum standardmäßigen Verhalten der Properties-Klasse werden Leerzeichen am Ende der Parameterwerte entfernt.

Eine Beispielkonfiguration wird in der Datei *conf/mainzelliste.conf.default* (im Quellcode) bzw. *WEB-INF/classes/mainzelliste.conf.default* (in der Binärdistribution) mitgeliefert. Für einen Testbetrieb sind mindestens die Datenbankverbindung (2.1) sowie der Zugangsschlüssel („apiKey“, 2.2) anzupassen.

Diese Fassung bezieht sich auf Version 1.4 der Mainzelliste.

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung
Gefördert durch DFG Deutsche Forschungsgemeinschaft

1.1 Format und Reservierte Wörter

Benutzerdefinierte Bezeichner (Feldnamen, ID-Typen etc.) dürfen Buchstaben, Ziffern und Unterstriche („_“) enthalten. Konfigurationsparameter sowie die durch Punkte getrennten Teile davon sollten nicht als Bezeichner verwendet werden. Darüber hinaus sollten die folgenden Zeichenketten nicht als eigene Bezeichner verwendet werden, weil sie in der Schnittstelle der Mainzelliste besondere Bedeutung haben und eine Verwendung als Bezeichner zu Mehrdeutigkeiten führen kann:

`tokenId`

2. Konfigurationsparameter

2.1 Datenbankkonfiguration

Die Datenbank, die die Patientenliste nutzt, muss existieren und der angegebene Benutzer muss darauf Schreibrechte besitzen. Für die Initialisierung beim ersten Start muss der verwendete Datenbankbenutzer insbesondere berechtigt sein, Tabellen und Indizes anzulegen. Am praktikabelsten ist es, einen eigenen Benutzer anzulegen, der Besitzer der verwendeten Datenbank ist.

db.driver

Vollständiger Klassenname des JDBC-Treibers, z.B. `com.mysql.jdbc.Driver`. Die Klasse muss im Classpath auffindbar sein.

db.url

JDBC-URL der Datenbankverbindung, zum Beispiel: `jdbc:mysql://localhost:3306/mainzelliste`.

db.username

Benutzername des Datenbankbenutzers, über den sich die Patientenliste verbindet.

db.password

Passwort des des Datenbankbenutzers, über den sich die Patientenliste verbindet.

Beispiel

```
db.driver = com.mysql.jdbc.Driver
db.url = jdbc:mysql://localhost:3306/mainzelliste
db.username = mainzelliste
db.password = mainzelliste
```

2.2 Zugriffsberechtigungen

Jeder zugreifende Server wird durch einen Satz `servers.{n}…`-Parameter definiert, wobei für {n} die fortlaufende Nummer, beginnend mit 0, einzusetzen ist.

servers.{n}.apiKey

Ein geheimer Schlüssel, mit dem sich der Server gegenüber der Patientenliste identifiziert (wird im Header `mainzellisteApiKey` bei Requests mitübertragen). Zur Authentifizierung dient darüber hinaus die Adresse des Servers (vgl. `servers.{0}.allowedRemoteAdresses`).

servers.{n}.permissions

Berechtigungen des Servers. Mögliche Berechtigungen:

- *createSession*: Erlaubt, Sessions anzulegen.
- *createToken*: Erlaubt, Token anzulegen
- *tt_{tokenType}*: Erlaubt, Token vom Typ *{tokenType}* anzulegen. Tokentypen sind:[1]
  - *tt_addPatient*: Token zum Anlegen eines Patienten.
  - *tt_readPatients*: Token zum Lesen von Patientendaten (IDs und IDAT).

servers.{0}.allowedRemoteAdresses

Netzwerkadressen (IPv4 und IPv6) oder Adressbereiche in CIDR-Notation (nur IPv4), von denen der Server zugreift, getrennt durch Semikolon. Dient in Kombination mit servers.{n}.apiKey der Authentifizierung. Beispiele:

- Nur Zugriffe von localhost:

```
servers.0.allowedRemoteAdresses = 127.0.0.1;0:0:0:0:0:0:0:1
```

- Zugriffe aus einem typischen privaten Netzwerk:

```
servers.0.allowedRemoteAdresses = 192.168.56.0/24
```

- Kombination von beidem:

```
servers.0.allowedRemoteAdresses = 127.0.0.1;0:0:0:0:0:0:0:1;192.168.56.0/24
```

servers.allowedOrigins

Hostnamen, von denen ursprungsübergreifende Zugriffe (Cross-Origin Resource Sharing, CORS) zugelassen werden, getrennt durch Semikola. Bei ursprungsübergreifenden Zugriffen muss die Angabe im Header „Origin“ mit einem dieser Werte übereinstimmen. Beispiel:

```
servers.allowedOrigins = https://mdat.example.org;https://dev.mdat.example.org
```

callback.allowedFormat

Erlaubtes Format für Callback-URLs (verwendet im *addPatient*-Token) als regulärer Ausdruck. Damit lässt sich zum Beispiel die mögliche Callbackadresse auf einen bestimmten Domainnamen oder auf sichere (HTTPS-) Verbindungen beschränken.

Beispiel:

```
servers.0.apiKey = mdat1234
servers.0.permissions = createSession;;createToken;tt_addPatient;tt_readPatients
servers.0.allowedRemoteAdresses = 127.0.0.1;0:0:0:0:0:0:0:1
callback.allowedFormat = https://.*
```

[1] Details siehe Schnittstellendokument.

callback.allowSelfsigned

Falls `true`, werden bei Callbackaufrufen selbstsignierte Zertifikate des aufgerufenen Hosts akzeptiert. Diese Einstellung sollte nur für Test- und Entwicklungszwecke benutzt werden.

2.3 Logging

log.filename

Pfad der Logdatei.

log.level

Loglevel im log4J-Format (vgl. http://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/lf5/LogLevel.html). Unterstützt werden `INFO`, `DEBUG`, `WARN`, `ERROR`, `FATAL`. Standardwert: `DEBUG`.

2.4 Felddefinition

field.{name}.type

Definiert ein Feld mit dem Namen `{name}`. Der Name muss eindeutig sein und mit dem `name`-Attribut des korrespondierenden Eingabefelds, z.B. im HTML-Formular, übereinstimmen. Als Wert des Konfigurationsparameters ist der Typ des Felds anzugeben. Gültig sind alle nicht-abstrakten Java-Klassen, die von der Klasse de.pseudonymisierung.mainzelliste.Field erben und im Classpath liegen.

Der Lieferumfang der Patientenliste enthält folgende vordefinierte Typen[2]:

- PlainTextField: Felder, die eine Zeichenkette (String) enthalten.
- IntegerField: Felder, die eine Integerzahl enthalten.

Bei Klassen im Paket de.pseudonymisierung.mainzelliste (d.h. die vordefinierten), muss nur der einfache Klassenname angegeben werden, bei anderen der „fully qualified class name“ (mit Paket).

Um zu Testzwecken ein Feld zu deaktivieren, genügt es, den entsprechenden Eintrag `field.{name}.type` auszukommentieren.

Field.{name}.label

Ein Feldname zur Anzeige in Fehlermeldungen.

field.{name}.transformers

Definiert eine Liste von Transformationen, die auf das Feld `{name}` angewendet werden. Erlaubt sind nicht-abstrakte Java-Klassen, die von de.pseudonymisierung.mainzelliste.matcher.FieldTransformer erben. Mehrere Transformationen können in einer kommaseparierten Liste angegeben werden. Dabei gelten folgende Regeln:

- Die erste Transformation erhält die Benutzereingabe als Objekt der Klasse, die beim zugehörigen Parameter field.{name}.type angegeben ist. Der Eingabetyp der Transformation muss zum Feldtyp kompatibel sein (vgl. API-Dokumentation zu `de.pseudonymisierung.mainzelliste.matcher.FieldTransformer.getInputClass()`).

[2] Die ebenfalls bereitgestellten Typen HashedField und CompoundField werden zur Zeit nur intern verwendet.

- Der Eingabetyp der folgenden Transformationen muss zur Ausgabe der vorhergehenden kompatibel sein (vgl. API-Dokumentation zu `de.pseudonymisierung.mainzelliste.matcher.FieldTransformer.getOutputClass()`).
- Die Ausgabe der letzten Transformation muss kompatibel zur Eingabe des zugehörigen Feldvergleichs (siehe `field.{name}.comparator`) sein.

Vordefinierte Transformationen sind:

- StringTrimmer (PlainTextField -> PlainTextField): Entfernt aus einer Zeichenkette führende und folgende Leerzeichen (gemäß der Definition der Java-Methode String#trim(), also alle Unicode-Zeichen <= 0020, was auch Tabulatoren und andere, spezielle Leerzeichen beinhaltet).
- StringNormalizer (PlainTextField -> PlainTextField): Normalisierung von Zeichenketten, z.B. Umwandlung in Großbuchstaben und Ersetzen von Umlauten.
- BloomFilterTransformer (PlainTextField -> HashedField): Erzeugen von als BloomFilter kodierten n-Grammen nach dem Verfahren von Schnell et al. (Schnell, Bachteler, and Reiher 2009)
- FirstNameDecomposer (PlainTextField -> PlainTextField): Auftrennen von Vornamen in drei Komponenten.
- GermanLastNameDecomposer (PlainTextField -> PlainTextField): Auftrennen von deutschen Nachnamen in drei Komponenten, besondere Behandlung von Spezialfällen wie akademischen und Adelstiteln.

Bei Klassen im Paket de.pseudonymisierung.mainzelliste liegen (d.h. die vordefinierten), muss nur der einfache Klassenname angegeben werden, bei anderen der „fully qualified class name“ (mit Paket).

field.{name}.comparator

Definiert einen Attributvergleich für das Feld `{name}`. Gültig sind nicht-abstrakte Klassen, die von de.unimainz.mainzelliste.matcher.FieldComparator erben. Falls die Klasse nicht im Paket de.unimainz.mainzelliste.matcher liegt, ist der fully qualified class name anzugeben. Derzeit sind folgende vordefinierte Vergleiche vorhanden:

BinaryFieldComparator

Prüft Gleichheit der Eingaben durch Aufruf der Methode `.equals()` der Eingabeklasse. Rückgabe: 0 im Fall `false`, 1 im Fall `true`.

NGramComparator

Ähnlichkeitsvergleich auf Zeichenketten. Für beide Eingaben wird die Menge der enthaltenen Teilzeichenketten einer definierten Länge *n* („N-Gramme“) bestimmt. Aus diesen Mengen (im Folgenden S_1 und S_2 bezeichnet) wird die Ähnlichkeit durch folgende Formel berechnet:

$$2 \cdot \frac{|S_1 \cap S_2|}{|S_1| + |S_2|}$$

DiceFieldComparator

Berechnet das gleiche Ähnlichkeitsmaß wie *NGramComparator*, basiert statt auf Klartext aber auf Bloom-Filtern, die durch den BloomFilterTransformer (siehe field.{name}.transformers) erzeugt wurden.

exchangeGroup.{n}

Kommaseparierte Feldnamen. Definiert eine Reihe vom Feldnamen, die als austauschbar definiert sein sollen. Wie diese Information verwertet wird, liegt in der Verantwortung des Matchers. Für die eingebauten Matcher (Epilink-Matcher und ThreadedEpilinkMatcher) gilt Folgendes: Beim Vergleich zweier Datensätze werden die Felder einer exchange group in allen möglichen Permutationen über Kreuz verglichen und diejenigen mit der höchsten Übereinstimmung gehen in das Gesamtergebnis ein. Es seien zum Beispiel definiert:

```
exchangeGroup.0 = vorname, nachname, geburtsname
```

Dann werden für zwei Datensätze `rec1` und `rec2` folgende Vergleiche vorgenommen:

	Vergleiche `rec1.vorname` mit:	Vergleiche `rec1.nachname` mit:	Vergleiche `rec1.geburtsname` mit:
1	`rec2.vorname`	`rec2.nachname`	`rec2.geburtsname`
2	`rec2.vorname`	`rec2.geburtsname`	`rec2.nachname`
3	`rec2.nachname`	`rec2.vorname`	`rec2.geburtsname`
4	`rec2.nachname`	`rec2.geburtsname`	`rec2.vorname`
5	`rec2.geburtsname`	`rec2.vorname`	`rec2.nachname`
6	`rec2.geburtsname`	`rec2.nachname`	`rec2.vorname`

Diejenige Kombination mit dem höchsten Gesamtvergleichswert (Epilink-Gewicht) wird ausgewählt.

Beispiel für Felddefinitionen, -transformationen und -vergleiche

```
field.vorname.type = PlainTextField
field.nachname.type = PlainTextField
field.geburtsname.type = PlainTextField
field.geburtstag.type = IntegerField
field.geburtsmonat.type = IntegerField
field.geburtsjahr.type = IntegerField
field.plz.type = PlainTextField

field.vorname.transformers = StringNormalizer, FirstNameDecomposer
field.nachname.transformers = StringNormalizer, GermanLastNameDecomposer
field.geburtsname.transformers = StringNormalizer, GermanLastNameDecomposer

field.vorname.comparator = NGramComparator
field.nachname.comparator = NGramComparator
field.geburtsname.comparator = NGramComparator
field.geburtstag.comparator = BinaryFieldComparator
field.geburtsmonat.comparator = BinaryFieldComparator
field.geburtsjahr.comparator = BinaryFieldComparator
field.plz.comparator = BinaryFieldComparator
field.ort.comparator = NGramComparator
```

2.5 Serverseitige Validierung

Validatoren prüfen die Eingabefelder eines neuen Patienten auf bestimmte Bedingungen. Falls diese nicht erfüllt sind, weist der Server mit Fehlercode 400 (Bad Request) die Anfrage zurück.

validator.field.{name}.required

Wertebereich: true / false. Gibt an, ob das Feld {name} obligatorisch ist, d.h. nicht leer sein darf.

validator.field.{name}.format

Wert: Ein regulärer Ausdruck. Gibt an, dieser Ausdruck auf das das Feld {name} passen muss. Es gilt die Syntax, die von java.util.regex.Pattern verwendet wird (vgl. http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html).

Es ist zu beachten, dass Escapesequenzen mit doppeltem Backslash zu schreiben sind, da Java selbst Escapesequenzen benutzt.

validator.date.{n}.fields

Kommaseparierte Liste von Feldern, die Komponenten eines Datums sind. {n} ist eine Zahl >=0, welche das Datum identifiziert.

validator.date.{n}.format

Formatdefinition des Datums mit dem Index {n}. Definiert das Format des Strings, der aus der Konkatenation der im zugehörigen Parameter validator.date.{n}.fields genannten Felder entsteht. Die Formatdefinition entspricht derjenigen von java.text.SimpleDateFormat (vgl. http://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html).

Beispiel

Der reguläre Ausdruck im Beispiel bedeutet: Alle Zeichenketten, die aus Buchstaben, inklusive deutschen Umlauten, Punkten, Bindestrichen, Apostrophen und Leerzeichen bestehen, wobei mindestens ein Zeichen ein Buchstabe sein muss.

```
validator.field.vorname.required = true

validator.field.vorname.format = [A-Za-zÄäÖöÜüß\\.\\-' ]*[A-Za-zÄäÖöÜüß]+[A-Za-
zÄäÖöÜüß\\.\\-' ]*

validator.date.0.fields = geburtstag, geburtsmonat, geburtsjahr
validator.date.0.format = ddMMyyyy
```

2.6 Matcher

matcher

Definiert die Klasse, die das Matchen übernimmt. Dies muss das Interface de.pseudonymisierung.mainzelliste.matcher.Matcher implementieren. Falls die Klasse nicht im Paket de.pseudonymisierung.mainzelliste.matcher liegt, ist der fully qualified class name anzugeben.

Mitgeliefert ist das Verfahren „Epilink“, ein einfaches gewichtsbasiertes Record-Linkage-Verfahren nach (Contiero et al. 2005). Es ist implementiert in der Klasse de.pseudonymisierung.mainzelliste.matcher.EpilinkMatcher. Die Klasse ThreadedEpilinkMatcher im gleichen Paket bietet eine Epilink-Implementierung, die für Mehrprozessorsysteme optimiert ist. In Funktion und Konfigurationsmöglichkeiten verhalten sich beide gleich.

Für einen Matcher mit dem Namen `{name}` werden alle Konfigurationsparameter, die mit `matcher.{name}.` beginnen, an den Matcher weitergereicht. Die Parameter für den EpilinkMatcher bzw. Epilinkmatcher werden im folgenden Abschnitt erklärt.

EpilinkMatcher / ThreadedEpilinkMatcher

Epilink berechnet Matchgewichte nach dem folgenden Verfahren:

- Es sei s_i der Ähnlichkeitswert (=Ausgabe des zugehörigen FieldComparators), f_i die relative Häufigkeit (siehe matcher.epilink.{fieldname}.frequency) und e_i die angenommene Fehlerrate (siehe matcher.epilink.{fieldname}.errorRate) für das i-te Feld.
- Das Gesamtgewicht eines Datensatzes errechnet sich aus der Formel:
- $\frac{\sum_i w_i s(x_i^1, x_i^2)}{\sum_i w_i}$
- Es sei w_i das maximale Gewicht aller betrachteten Datensatzpaare. Falls w_i größer oder gleich dem durch matcher.epilink.threshold.match festgelegtem Wert ist, wird der zugehörige Datensatz aus der Datenbank als Match zurückgegeben.
- Falls w_i kleiner als matcher.epilink.threshold.match und größer oder gleich matcher.epilink.threshold.non_match ist, kommt der zugehörige Datensatz potentiell als Match in Frage. In diesem Fall wird dem Benutzer die Möglichkeit gegeben, die Daten zu bestätigen oder zu korrigieren.
- Falls w_i kleiner als matcher.epilink.threshold.non_match ist, wird der eingegebene Datensatz als neu betrachtet und ein neues Pseudonym erzeugt.

matcher.epilink.{fieldname}.frequency

Wertebereich: Reelle Zahl (double) im Intervall [0,1]. Gibt die relative Wertehäufigkeit des Feldes {fieldname} an und entspricht der u-Wahrscheinlichkeit im Fellegi-Sunter-Modell (Fellegi and Sunter 1969). In der Regel nimmt man den Kehrwert der Anzahl an Werten, die dieses Feld annehmen kann bzw. in einem hinreichend großen Testdatensatz annimmt. Das Beispiel zeigt den typischen Fall für Monat (1/12) und Tag im laufenden Monat(1/30):

```
matcher.epilink.geburtstag.frequency = 0.0333
# 1 / 12
matcher.epilink.geburtsmonat.frequency = 0.833
```

matcher.epilink.{fieldname}.errorRate

Wertebereich: Reelle Zahl (double) im Intervall [0,1]. Angenommene Fehlerrate für das Attribut {feldname}. Diese kann aus hinreichend großen Testdaten mit bekanntem Matchingstatus eruiert werden, in der Praxis reicht aber oft ein einheitlicher ad-hoc-Wert für alle Felder, z.B. 0.05.

```
matcher.epilink.vorname.errorRate = 0.01
matcher.epilink.nachname.errorRate = 0.008
```

matcher.epilink.threshold.match

Wertebereich: Reelle Zahl (double) im Intervall [0,1]. Gibt das minimale Gewicht an, welches der Vergleich eines Datensatzpaares ergeben muss, damit dieses Paar als definitiver Match in Frage kommt.

matcher.epilink.threshold.non_match

Wertebereich: Reelle Zahl (double) im Intervall [0,1]. Gibt das minimale Gewicht an, welches der Vergleich eines Datensatzpaares ergeben muss, damit dieses Paar als potentieller Match in Frage kommt. Datensatzpaare mit Gewichten echt kleiner diesem Wert kommen als Match definitiv nicht in Frage (definitive Non-Matche).

2.7 ID-Erzeugung

idgenerators

Kommaseparierte Liste von eindeutigen Bezeichnern, die die verwalteten ID-Typen identifizieren. Jeder Eintrag der Liste benötigt einen korrespondierenden Eintrag `idgenerator.{identifier}`.

idgenerator.{identifier}

Gibt an Algorithmus an, welcher die IDs für den jeweiligen ID-Typ erzeugt. Gültig sind nicht-abstrakte Klassen, die das Interface de.pseudonymisierung.mainzelliste.IDGenerator implementieren. Falls die Klasse nicht im Pfad de.pseudonymisierung.mainzelliste liegt, muss der fully qualifed class name angegeben werden.

Mitgelieferte ID-Generatoren sind:

- PIDGenerator: Nicht-sprechende, fehlertolerante Pseudonyme nach dem Algorithmus von (Faldum and Pommerening 2005).
- SimpleIDGenerator: Gibt die Folge der natürlichen Zahlen, beschränkt durch den Wertebereich eines Integers, als Pseudonym aus. Für Testzwecke oder zur Benutzung als Datenbankschlüssel auf einem MDAT-Server.
- ExternalIDGenerator: Erzeugt selbst keine IDs, sondern steht stellvertretend für eine externe Generierung von IDs. IDs, die mit diesem Generator konfiguriert sind, müssen von externen Systemen oder Nutzern in der Mainzelliste gespeichert werden.

Konfiguration des PIDGenerator

Zu technischen Details siehe (Faldum and Pommerening 2005).

idgenerator.{name}.k1, idgenerator.{name}.k2, idgenerator.{name}.k3

Wertebereich jeweils: Ganze Zahl im Intervall [0, 2147483647]. Geheime Schlüssel, die zur Erzeugung des PID verwendet werden.

idgenerator.{name}.rndwidth

Wertebereich: Ganze Zahl im Intervall [0,12]. Anzahl der randomisierten Bits. Standardwert ist 0.

Beispiel

```
idgenerators = pid, intid

idgenerator.pid = PIDGenerator
idgenerator.pid.k1 = 1
idgenerator.pid.k2 = 2
idgenerator.pid.k3 = 3
# rndwidth is optional (default 0)
idgenerator.pid.rndwidth = 0

idgenerator.intid = SimpleIDGenerator
```

2.8 Anzeige des Ergebnisses / Benutzeroberfläche

result.show

Wertebereich: true / false. Legt fest, ob nach dem Anlegen eines Patienten eine Ergebnisseite mit dem erzeugten Pseudonym angezeigt wird. Diese Einstellung greift nur, wenn dabei eine Redirect-Adresse zur direkten Weiterleitung des Benutzers im zugehörigen Token angegeben ist.

result.printIdat

Wertebereich: true / false. Fall true, werden die eingebenen Patientendaten zusammen mit dem PID angezeigt. Mit Klick auf den Button „Drucken und Patient anlegen“ werden die Daten gedruckt (Druckdialog öffnet sich) und über die Redirect-Adresse erfolgt der Rücksprung in die MDAT-Applikation.

language

Stellt eine feste Sprache für die Benutzeroberfläche ein. Gültige Werte: Alle Sprachcodes, für die eine entsprechende Sprachdatei verfügbar ist (z.B. „en“ oder „de“). Falls nicht definiert, erfolgt die Sprachauswahl per URL-Parameter oder HTTP-Header (Details siehe Schnittstellenbeschreibung) mit Englisch als Standard. Falls hier eine Sprache definiert ist, ist diese fest konfiguriert, d.h. die Mechanismen zur dynamischen Sprachauswahl sind außer Kraft gesetzt.

2.9 Sonstiges

debug

Wertebereich: true / false. Schaltet den Debug-Modus ein. Falls true, wird das Tokenhandling abgestellt, d.h. eine Anfrage, die ein Token erfordern, sind ohne Token möglich. Darüber hinaus verfallen erzeugt Tokens nicht. Vor Produktiveinsatz unbedingt abstellen!

dist

Name der Mainzelliste-Instanz. Dieser Name wird bei Aufruf der Root-Resource angezeigt.

sessionTimeout

Wertebereich: Positive ganze Zahl, interpretiert als Zeit in Minuten. Gibt an, wie lange eine Session inaktiv sein darf, bevor sie gelöscht wird. Eine Session ist in einem bestimmten Zeitraum inaktiv, wenn während dieser Zeit keine Zugriffe auf die Session (Erstellen und Lesen von Tokens) erfolgen und keines der zugehörigen Tokens erfolgreich eingelöst wird. Die tatsächliche Dauer der erlaubten Inaktivität kann aus technischen Gründen bis zu einer Minute länger dauern. Falls nicht angegeben, gilt ein Standardwert von 10 Minuten.

operator.contact

Kontaktdaten des Betreibers. Diese werden unterhalb aller HTML-Formulare angezeigt. Sinnvollerweise wird hier eine Kontaktadresse für Supportfälle angegeben.

operator.logo

Pfad einer Grafikdatei, die als Logo oben rechts in allen HTML-Formularen angezeigt wird. Möglich ist:

- Ein absoluter Pfad auf dem Server.
- Ein relativer Pfad innerhalb der Applikation (.war-Datei).
- Ein relativer Pfad innerhalb des Verzeichnisses META-INF/resources in einer .jar-Datei auf dem class path.

3. Literatur

Contiero, P, A Tittarelli, G Tagliabue, A Maghini, S Fabiano, P Crosignani, and R Tessandori. 2005. “The EpiLink Record Linkage Software: Presentation and Results of Linkage Test on Cancer Registry Files.” *Methods of Information in Medicine* 44 (1): 66–71. http://www.ncbi.nlm.nih.gov/pubmed/15778796.

Faldum, Andreas, and Klaus Pommerening. 2005. “An Optimal Code for Patient Identifiers.” *Computer Methods and Programs in Biomedicine* 79 (1) (July 1): 81–8. doi:10.1016/j.cmpb.2005.03.004. http://www.cmpbjournal.com/article/S0169-2607(05)00067-2/abstract.

Fellegi, Ivan P., and Alan B. Sunter. 1969. “A Theory for Record Linkage.” *Journal of the American Statistical Association* 64 (328) (December): 1183. doi:10.2307/2286061. http://www.citeulike.org/user/dbauer/article/590229.

Schnell, Rainer, Tobias Bachteler, and Jörg Reiher. 2009. “Privacy-preserving Record Linkage Using Bloom Filters.” *BMC Medical Informatics and Decision Making* 9 (1) (January): 41. doi:10.1186/1472-6947-9-41. http://www.biomedcentral.com/1472-6947/9/41.