Schnittstelle der Mainzelliste

Version 3.0.0 vom 09.07.2018

von M. Lablans[1], A. Borg[2] und G. Tremper[1]

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

Kontakt: info@mainzelliste.de

Versionshistorie

Version	Datum	Änderung/Status	Autor
2.0	11.04.2014	Erste veröffentlichte Version	ML, AB
2.0-1	08.07.2014	Tippfehler im Beispiel zum readPatients-Token behoben (gemeldet von Thomas Zumbrunn)	ML
2.0-2	29.10.2014	Klarstellung Gültigkeitsdauer von Tokens; Korrektur der Autorisierung beim Zugriff auf Sessions	AB
2.1.0	20.02.2015	Schnittstellenerweiterungen (siehe unten), „Cite as...“	AB, ML
2.2.0	29.12.2015	Erweiterung: Sprache per URL-Parameter	AB
3.0.0	09.07.2018	Erweiterung: Externe IDs; dadurch inkompatible Änderung editPatient-Token	AB, ML, GT

1. Einleitung

Die Mainzelliste bietet eine REST-basierte Schnittstelle zur Pseudonymisierung von Personendaten zur datenschutzgerechten Verarbeitung. Diese Schnittstelle bildet die folgenden Anwendungsfälle ab:

- Pseudonymisierung erster Stufe (IDAT → PID)
- Verwaltung multipler Pseudonyme (PSN, SIC, DeineID1, DeineID2, ...)
- Auflösung von Namen (Pseudonym → IDAT)
- Handling von temporären Pseudonymen

Dieses Dokument beinhaltet:

1. Ein Einführungsbeispiel (Abschnitt 1.5), welches die wichtigsten Elemente der Schnittstelle anhand eines typischen Anwendungsfalls vorstellt. Dieser Abschnitt bietet insbesondere „Neulingen“ einen ersten Einstieg in die Benutzung der Schnittstelle.
2. Eine formale Spezifikation der Schnittstelle (Abschnitt 2 und folgende). Diese dient als Referenz für Entwickler, welche die Schnittstelle client- oder serverseitig implementieren und als umfassender Überblick über die Funktionalität.

1.1 Cite as...

Als Referenz für die Schnittstelle der Mainzelliste in Publikationen kann der in 2015 bei BMC Medical Informatics and Decision Making open-access veröffentlichte Artikel [1] dienen.

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung

Gefördert durch DFG Deutsche Forschungsgemeinschaft

1.2 Versionshistorie

Die Schnittstelle der Mainzelliste unterliegt einer stetigen Weiterentwicklung. Daher unterstützt jede Komponente eine bestimmte Schnittstellenversion x.y. Die Nummerierung folgt dabei der Spezifikation des Semantic Versioning [2]:

- MAJOR-Versionen (x) für inkompatible Änderungen,
- MINOR-Versionen (y) für neue Funktionalität bei Erhaltung der Abwärtskompatibilität.

Die im Untertitel dieses Dokuments genannte Versionsnummer erhält zusätzlich eine dritte Nummer (PATCH-Version z) für Fehlerbehebungen. Die PATCH-Versionen von Schnittstellendokument und Implementierung sind grundsätzlich unabhängig voneinander.

Dieses Dokument beschreibt Version 2.2 der Mainzelliste-Schnittstelle.

Die verwendete Version der Schnittstelle (im Format MAJOR.MINOR) ist von Clients im HTTP-Header *mainzelliste-ApiVersion* oder als URL-Parameter (gleichen Namens) anzugeben.[1] Falls diese Angabe fehlt, wird Version 1.0 angenommen.

Änderungen in Version 3.0

- Erweiterungen für die Speicherung extern generierter Pseudonyme.
- Im Zusammenhang mit der Verarbeitung extern generierter Pseudonyme war eine nicht abwärtskompatible Änderung des editPatient-Tokens erforderlich – es müssen nun alle editierbaren Felder explizit genannt werden (siehe auch der Hinweis in der Tokenbeschreibung, Abschnitt 3.2, Seite 15).

Änderungen in Version 2.2

- Für die HTML-Formulare ist eine Sprachauswahl mittels HTTP-Header oder URL-Parameter möglich (siehe Abschnitt 2.5).

Änderungen in Version 2.1

- Mittels des *editPatient*-Tokens und einem PUT-Request können die IDAT eines Patienten geändert werden. Als Schnittstelle für menschliche Benutzer wird ein HTML-Formular bereit gestellt (S. 8, 12 und 15).
- Sitzungen und Tokens können mittels DELETE-Requests gelöscht werden (S. 10 und 11).
- In einem POST-Request können andere HTTP-Methoden (insbesondere PUT und DELETE) gekapselt werden („method override“; siehe Abschnitt 2.1).

Änderungen in Version 2.0

Version 2.0 ist die erste in diesem Dokument beschriebene Fassung der Schnittstelle. Die folgende Änderungsliste bezieht sich auf den Funktionsumfang der Mainzelliste-Software, Version 1.2, und auf das Dokument „Zugriff durch MDAT-Server im Anwendungsfall „behandlungsfernes Register“ in der Version 1.4.1.

- Mittels des *readPatients*-Tokens und eines GET-Requests können IDs und IDAT vorhandener Patienten ausgelesen werden (S. 8 und S. 15).
- Im Redirect-Aufruf kann die Token-ID mittels einer speziellen Template-Variable als URL-Parameter übergeben werden (S. 14).

[1] Die Angabe in der URL ist vor allem für Fälle vorgesehen, in denen eine Aufruf-URL an ein anderes weitergeben wird, beispielsweise wenn der Webbrowser eines Benutzers auf das Formular zum Anlegen eines Patienten verwiesen wird (vgl. Beispiel unten).

- Der Callback-Aufruf überträgt nicht mehr eine ID, sondern ein Array von IDs (S. 14).
- Entsprechend wird beim Anlegen eines Patienten im Falle einer JSON-Rückgabe (`Accept: application/json`) ein Array von IDs zurückgegeben (S. 7).

1.3 Lizenz, Verbreitung, Veränderung

Die Mainzelliste ist freie Software, vgl. http://www.mainzelliste.de. Ihre hier spezifizierte Schnittstelle ist offen, d.h. ihre Spezifikation für jedermann offengelegt und darf ohne unsere Genehmigung implementiert werden. Erweiterungen oder Änderungen sind ebenfalls erlaubt; im Sinne einer auch langfristig einheitlichen Schnittstelle bitten wir jedoch vor einer Verbreitung abgewandelter Versionen Änderungsvorschläge mit uns zu diskutieren. Bislang konnten wir alle Vorschläge in dieser offiziellen Schnittstellenspezifikation umsetzen.

1.4 Ressourcen

Dem REST-Paradigma folgend findet die Kommunikation über HTTP statt und verarbeitet sogenannte Ressourcen:

- `patients` – Die namensgebenden Patienten samt ihrer identifizierenden Attribute und Identifikatoren.
- `sessions` – Bilden eine Nutzersitzung ab, um Tokens zu gruppieren.
    - `tokens` – Repräsentieren Berechtigungen zum Zugriff auf Funktionen der Schnittstelle
- `html` – vordefinierte Formulare in HTML zur Anzeige im Webbrowser.

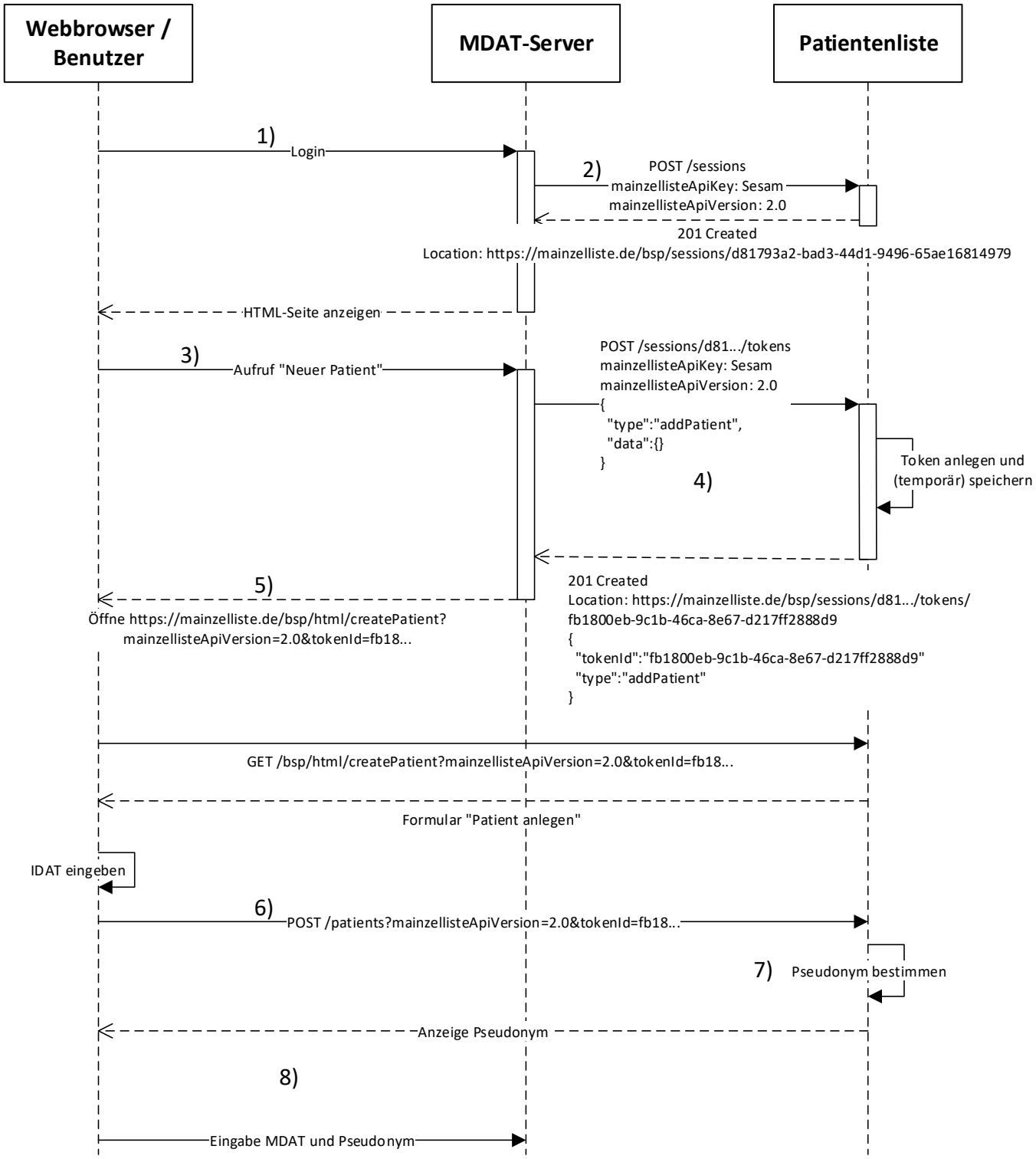

Abbildung 1: Sequenzenzdiagramm für Einführungsbeispiel

1.5 Einführungsbeispiel

Das folgende Beispiel gibt einen Überblick über die generelle Funktionsweise der Schnittstelle. Gegeben sei das folgende Szenario:

- In einem Register sollen medizinische Daten von Probanden (MDAT) pseudonymisiert gespeichert werden. Das Register bietet eine Webschnittstelle (HTML), über welche MDAT mitsamt des zugehörigen Pseudonyms (im Folgenden auch „ID“) eingegeben werden. Die Registersoftware verfügt über eine Nutzerverwaltung und eine Nutzerauthentifizierung (z.B. Eingabe von Benutzername und Passwort).
- Die MDAT stammen von verschiedenen behandelnden Kliniken. Jeder Klinik sind die identifizierenden Daten (IDAT) der dort behandelten Registerpatienten bekannt.
- Um Patienten institutionsübergreifend verfolgen zu können, kommt für die Pseudonymisierung eine zentrale Patientenliste mit der hier beschriebenen Schnittstelle zum Einsatz. Die zuständigen Dokumentare der behandelnden Kliniken geben hier IDAT von Registerpatienten ein. Die Patientenliste prüft mittels eines Record-Linkage-Algorithmus, ob der Patient schon eingetragen ist. Falls ja, wird das vorhandene Pseudonym zurückgegeben, falls nein, werden die IDAT gespeichert und ein neues Pseudonym erzeugt und zurückgegeben.

Im Folgenden wird der Fall betrachtet, dass ein Benutzer einen neuen Patienten in die Registersoftware eintragen möchte und dafür ein Pseudonym für diesen Patienten anfordern muss. Dieser Ablauf ist im Sequenzdiagramm in Abbildung 1 dargestellt. Die Ziffern in den folgenden Anmerkungen beziehen sich auf dieses Diagramm.

Authentifizierung und Autorisierung

Die Schnittstelle der Mainzelliste verzichtet auf eine gesonderte Authentifizierung individueller Benutzer, da diese üblicherweise bereits von einem anderen Server geleistet wird. In diesem Fall wird angenommen, dass dies der Webserver der MDAT-Anwendung (im Folgenden „MDAT-Server“) ist. Dieser authentifiziert sich gegenüber der Patientenliste und stellt für Zugriffe der Benutzer Zugriffstokens aus, wodurch ein Single-Sign-On-Konzept realisiert wird. Die Schritte 1-4 im Sequenzdiagramm illustrieren dieses Konzept:

(1) Der Benutzer meldet sich am MDAT-Server an, zum Beispiel durch Benutzername und Passwort oder ein Smartcard-Zertifikat. Der MDAT-Server erstellt eine Browsersession für den Benutzer.
(2) Der MDAT-Server erstellt eine Session auf der Mainzelliste. Diese dient quasi als Erweiterung der Browsersession über einen weiteren Server hinaus. Die Erstellung der Session erfolgt über eine POST-Anfrage auf die Ressource */sessions/* der Mainzelliste. Der MDAT-Server identifiziert sich über einen (in der Patientenliste konfigurierbaren) Zugangsschlüssel, der im HTTP-Header *mainzellisteApiKey* übergeben wird. Der Session wird ein eindeutiger Bezeichner *{sessionId}* und die Ressource */sessions/{sessionid}/* zugewiesen. Die vollständige URL der Session wird dem MDAT-Server zurückgegeben.
(3) Der Benutzer wählt die Funktion „Neuer Patient“ aus.
(4) Um dem Benutzer die Berechtigung zum Pseudonymisieren des Patienten zu erteilen, legt der MDAT-Server auf der Patientenliste ein Zugriffstoken an. Tokens werden innerhalb der Session durch den Pfad */tokens/* repräsentiert. Das Anlegen eines Tokens erfolgt durch ein POST auf die Ressource */sessions/{sessionId}/tokens/*. Die wichtigsten Eigenschaften eines Tokens sind:
    a. Der Tokentyp (Parameter *type* im übergebenen Objekt). Dies ist ein Bezeichner, der festlegt, für welche Funktion das Token als Berechtigung gilt. Der Typ wird als Parameter *type* in der Anfrage zum Anlegen des Tokens mitgeteilt. In diesem Fall kommt der Typ *addPatient* zum Einsatz, welcher die Pseudonymisieren genau eines Patienten erlaubt.
    b. Die Token-ID. Dieser Bezeichner identifiziert das Token. Um das Token einzulösen (d.h. damit eine Funktion der Mainzelliste aufzurufen) ist nur die Token-ID nötig. Sie wird durch eine zufällige Zeichenkette realisiert, die hinreichend lang ist, um Kollisionen und das Erraten praktisch unmöglich zu machen.

Das Token wird als JSON-Objekt zurückgegeben, so dass der MDAT-Server die Token-ID aus der Rückgabe auslesen kann.

Durch die Verwendung von Zugriffstokens kann auf die Verwaltung individueller Benutzerkonten auf Seiten der Mainzelliste verzichtet werden. Das verhindert die doppelte Erfassung dieser Information, die ja ohnehin schon auf dem MDAT-Server vorliegen muss, und erleichtert die Administration der Mainzelliste durch externe Dienstleister.

Pseudonymisierung

Die eigentliche Pseudonymisierung muss der Benutzer nun auf einer Webseite der Patientenliste vornehmen. Die Eingabe von IDAT in ein vom MDAT-Server ausgeliefertes Formular würde der strikten Datentrennung zwischen medizinischen und identifizierenden Daten widersprechen, da IDAT vom MDAT-Server mitgelesen werden könnten. Dies läuft wie folgt ab:

(5) Der MDAT-Server leitet den Benutzer auf die Ressource /html/createPatient der Patientenliste weiter, beispielsweise durch einen Redirect oder einen Link. Die Token-ID des in Schritt (4) erzeugten Tokens wird als URL-Parameter *tokenID* angehängt. Die Patientenliste liefert daraufhin ein HTML-Formular, in das der Benutzer die identifizierenden Daten des neuen Patienten eingeben kann.
(6) Beim Abschicken des Formulars werden die eingegebenen Daten per POST-Anfrage auf die Ressource */patients* abgeschickt. Wie im letzten Schritt wird die Token-ID als URL-Parameter übergeben. Zur Autorisierung der Anfrage prüft die Patientenliste, ob ein Token mit dieser Token-ID existiert und gültig ist (d.h. noch nicht benutzt wurde).
(7) Die Patientenliste bestimmt ein Pseudonym für den Patienten – ein existierendes, falls bereits ein Eintrag mit diesen Daten vorhanden ist, ein neues, falls es sich um einen neuen[2] Patienten handelt.
(8) Das Pseudonym wird dem Benutzer angezeigt. Dieser kann es in seine Akten übernehmen und dafür verwenden, in der MDAT-Anwendung Daten einzutragen.

2. Ressourcen und Methoden

2.1 Allgemeines

Zur Authentifizierung ist, falls nicht anders angegeben, der konfigurierte Zugangsschlüssel im Header `mainzellisteApiKey` zu übermitteln. Zusätzlich wird die IP-Adresse des aufrufenden Systems geprüft. Für jeden Zugriff ist außerdem die spezifische Berechtigung angegeben, die in der Konfiguration dem zugreifenden Server zugeteilt werden muss (vgl. Konfigurationshandbuch, Abschnitt „Zugriffsberechtigungen“). Ausgenommen sind solche Aktionen, die die Angabe einer Token-ID (in der Regel als URL-Parameter tokenId übergeben) erfordern. In diesem Falle reicht die Kenntnis einer gültigen Token-ID als Berechtigung aus.

Die Schnittstelle unterstützt das sogenannte „method override“, d.h. mittels eines POST-Requests können auch andere HTTP-Methoden genutzt werden. Das ermöglicht es zum Beispiel, mittels HTML-Formularen in Browsern Aufrufe der Methoden PUT und DELETE durchzuführen.[3] Die eigentlich intendierte Methode muss dann auf eine der folgenden Arten mitgeteilt werden:

- im Header *X-HTTP-Method-Override* oder
- als URL-Parameter *_method*.

[2] „Neu“ und „vorhanden“ ist hier aus Sicht der Patientenliste gemeint. Ein Patient, der zum Beispiel vorher von einer anderen Klinik angelegt wurde, ist aus Sicht des Benutzers neu, auch wenn seine Daten schon in der Patientenliste vorhanden sind.
[3] Anwendungsbeispiel: Aufruf von PUT /patients/tokenId/{tokenId} aus dem Browser bei Editieren von IDAT mittels HTML-Formular.

2.2 /patients

Die namensgebenden Patienten mitsamt Ihrer identifizierenden Attribute und Identifikatoren.

Anlegen eines Patienten (POST)

Legt einen Patienten an und ordnet diesem IDs zu.

- Methode: POST
- Aufrufender: Beliebiger Tokeninhaber; z.B. Webbrowser im Auftrag eines Nutzers
- Content-Type der Eingabe (Header Content-Type): application/x-www-form-urlencoded
- Content-Type der Ausgabe (Header Accept):
  - text/html (Browserschnittstelle)
  - application/json (Maschinenschnittstelle)
- URL-Parameter:
  - tokenId: ID eines *addPatient*-Tokens
- Daten:
  - Felder (identifizierende Daten) als Schlüssel-Wert-Paare
  - Extern generierte IDs als Schlüssel-Wert-Paare (Schlüssel ist der ID-Typ).
  - Reserviertes Feld „sureness“: Boolean, *false*, falls nicht angegeben. Teilt mit, ob die eingegebenen Daten als sicher (d.h. fehlerfrei) anzusehen sind. Vgl. Behandlung unsicherer Fälle.

Es wird geprüft, ob in der Datenbank bereits ein Patient mit den gegebenen identifizierenden Daten enthalten ist. Falls nein, wird ein neuer Patient angelegt und für diesen werden die konfigurierten IDs erzeugt und zurückgegeben bzw. weitergeleitet. Falls ja, werden die vorhandenen IDs benutzt. Für den Aufrufenden ist nicht erkennbar, ob der Patient bereits vorhanden war oder neu angelegt wurde.

Falls eine extern generierte ID übergeben wird und diese bereits einem anderen Patienten zugeordnet ist, schlägt das Anlegen fehl (HTTP 409).

Eingabefelder

Zu den in der Anfrage übergebenen Feldern kommen gegebenenfalls die im Token vordefinierten Felder (vgl. Beschreibung des *addPatient*-Tokens) hinzu. Jedes in der Konfiguration definierte Feld muss in den Formulardaten oder im Token vorkommen, auch wenn es leer ist (in diesem Fall ist ein leerer String anzugeben).

Behandlung unsicherer Matches

Falls nicht sicher festgestellt werden kann, ob die Daten zu einem vorhandenen Patienten passen oder einen neuen Eintrag darstellen, erfolgt folgende Fallunterscheidung nach dem Wert des Eingabefelds *sureness*:

- *true*: Es wird ein neuer Patient angelegt. Die generierten IDs werden als vorläufig gekennzeichnet (Feld *tentative*, siehe Abschnitt ID). Damit wird signalisiert, dass eine nachträgliche Zusammenführung mit einem existierenden Fall möglich ist.
- *false* (bzw. nicht vorhanden): Aufgrund der Unsicherheit der Eingabe erfolgt keine Rückgabe einer ID, es wird als Statuscode 409 (Conflict) zurückgegeben. Dies gibt dem Eingebenden die Möglichkeit, die Daten zu überprüfen und ggfls. zu korrigieren oder zusätzliche Angaben (d.h. in nicht-obligatorischen Feldern) zu machen.

Die Entscheidung, ob ein unsicherer Match vorliegt, ist implementierungsabhängig. In der Standardkonfiguration der Mainzelliste ist dies der Fall, wenn das höchste Matchgewicht zwischen die Grenzwerte für sichere Matche und sichere Non-Matche fällt (vgl. Konfigurationshandbuch).

Rückgabe (nach Statuscodes):

- 201 Created: Der Patient wurde angelegt. Die generierte(n) ID(s) werden als HTML-Seite (Fall *text/html*) oder als Array von ID-Objekten (Fall *application/json*) zurückgegeben.
- 400 Bad Request: Die Eingabedaten sind ungültig (z.B.: Feld fehlt, Pflichtfeld ist leer, falsches Datumformat). Es wird eine Fehlermeldung ausgegeben, die den Grund genauer angibt.
- 401 Unauthorized: Das übergebene Token ist ungültig (vgl. Hinweise zur Gültigkeit eines Tokens in Abschnitt 2.4).
- 409 Conflict: Es wurde ein unsicherer Match gefunden (vgl. Hinweise zu unsicheren Matches oben), eine übergebene externe ID ist bereits einem anderen Patienten zugeordnet oder eine übergebene externe ID ist für den gefundenen Patienten bereits mit einem anderen Wert definiert.[4]

In den Fällen 400 und 409 bleibt das verwendete Token gültig.

Patientenliste abrufen (GET)

Ruft eine Liste aller Patienten ab, auf die das übergebene Token Zugriff verschafft. Das Token spezifiziert, welche Patienten und welche Daten (IDAT, IDs) dieser Patienten abgerufen werden (siehe Erläuterung des Tokentyps *readPatients* in Abschnitt 3.2).

- Methode: GET
- Parameter:
    - tokenId: Id eines Tokens vom Typ *readPatients*

Rückgabe: Array von Patientenobjekten mit den angeforderten Daten.

Daten eines Patienten ändern (PUT /patients/tokenId/{tokenId})

- Methode: PUT
- Content-Type der Eingabe (Header Content-Type):
    - application/json
    - application/x-www-form-urlencoded
- URL-Parameter (im Pfad):
    - *tokenId*: ID eines gültigen *editPatient*-Tokens
- Erforderliche Berechtigung: Kenntnis der Token-ID
- Daten: Die zu schreibenden Daten als Schlüssel-Wert-Paare. Dies können sowohl Felder (Schlüssel ist der Feldname) als auch extern erzeugte IDs (Schlüssel ist der ID-Typ) sein.

Der Datensatz des beim Anlegen des Tokens mitgeteilten Patienten wird wie folgt geändert:

- Alle Felder und IDs, die im Aufruf mit einem nichtleeren Wert enthalten sind, werden auf den übergebenen Wert gesetzt.
- Felder und IDs, die im Aufruf mit einem leeren Wert (`null` oder leerer String) enthalten sind, werden gelöscht. Der Feldinhalt wird dabei auf einen leeren String gesetzt, auch wenn als Eingabe `null` übergeben wird.
- Felder und IDs, die im Aufruf nicht enthalten sind (d.h. der Feldname kommt nicht als Schlüssel vor), werden nicht geändert.

[4] Zur Motivation: „ The 409 (Conflict) status code indicates that the request could not be completed due to a conflict with the current state of the target resource. This code is used in situations where the user might be able to resolve the conflict and resubmit the request.” [3]

- Validierungsfunktionen, welche die zu ändernden Felder betreffen, werden angewendet. Ist eine Datumsvalidierung konfiguriert, werden gegebenenfalls auch nicht zu ändernde Felder berücksichtigt (Beispiel: Versuch, für einen Patienten mit Geburtsdatum 30.01.2000 den Geburtsmonat auf „02“ zu setzen, schlägt fehl). Schlägt eine einzige Validierung fehl, hat der vollständige Aufruf keinen Effekt.

Rückgabe (nach Statuscodes):

- Der Statuscode im Erfolgsfall (d.h. die Patientendaten wurden mit den angegebenen Daten aktualisiert) richtet sich nach dem Eingabedatenformat und danach, ob eine Redirect-Adresse gesetzt wurde (vgl. Beschreibung des Tokens, S. 15):
  - application/json, kein Redirect gesetzt: 204 No Content.[5]
  - application/x-www-form-urlencoded, kein Redirect gesetzt: 200 OK. Es wird eine HTML-Seite ausgegeben, die über die erfolgreiche Änderung informiert.
  - Redirect gesetzt: 303 See Other. Der Header *Location* wird auf die im Token angegebene Redirect-URL gesetzt.
- 400 Bad Request: Die Eingabedaten sind ungültig (z.B.: Pflichtfeld ist leer, falsches Datumformat). Es wird eine Fehlermeldung ausgegeben, die den Grund genauer angibt.
- 401 Unauthorized: Das übergebene Token fehlt, ist ungültig (z.B. bereits benutzt, durch Ablaufen der Session ungültig geworden, oder von anderem Typ als *editPatient*) oder es wird versucht, ein Feld oder eine extern generierte ID zu schreiben, für welches das Token nicht gültig ist.
- 404 Not Found: Der angegebene Patient existiert nicht.

2.3 /sessions

Sitzungen bilden einen Container für Daten, die für einen Benutzer während einer Sitzung gespeichert werden. In der vorliegenden Schnittstellenversion umfassen diese ausschließlich die Zugriffstokens (siehe Abschnitt 2.4). Die Modellierung als Ressource erlaubt es,

a) REST-konform Daten über einen Aufruf hinaus zu speichern,
b) alle zu einer Nutzersitzung gehörigen Daten, wie z.B. Zugriffstickets, gemeinsam zu verlängern und beim Logout zu löschen.

Die Gültigkeitsdauer einer Sitzung kann konfigurationsabhängig beschränkt sein (vgl. Konfigurationshandbuch).

Sitzung anlegen (POST)

- Methode: POST
- Content-Type der Eingabe (Header Content-Type): nicht relevant
- Content-Type der Ausgabe (Header Accept): application/json
- Erforderliche Berechtigung: createSession

Rückgabe (nach Statuscodes)

- 201 Created: Session wurde angelegt. Rückgabe des Sessionsobjekts sowie URI der Session im Header Location.

[5] Grundsätzlich ist nicht auszuschließen, dass in einer zukünftigen Schnittstellenversion eine Entität zurückgegeben wird und damit ein Statuscode 200 OK zurückgegeben werden muss. Für größtmögliche Kompatibilität sollten Clients deshalb dazu in der Lage sein, beide Fälle als „erfolgreich“ zu erkennen, z.B. indem alle 2xx-Statuscodes akzeptiert werden.

- 401 Unauthorized: Die Authentifizierung ist fehlgeschlagen (falscher oder fehlender Authentifizierungsschlüssel, IP-Adresse des aufrufenden Systems ist nicht freigeschaltet oder Berechtigung für diese Aktion fehlt).

Sitzung lesen (GET /sessions/{session})

- Methode: GET
- Content-Type der Eingabe: n.a.
- Content-Type der Ausgabe (Header Accept): application/json
- Erforderliche Berechtigung: keine (Kenntnis der Session-ID reicht aus).

Rückgabe (nach Statuscodes)

- 200 OK: Rückgabe des Sessionobjekts
- 404 Not Found: Die angegebene Session existiert nicht

Sitzung löschen (DELETE /sessions/{sid})

- Methode: DELETE
- Erforderliche Berechtigung: keine (Kenntnis der Session-ID reicht aus).

Ein Aufruf für eine nicht (mehr) existierende Sitzung führt nicht zu einem Fehler, da das gewünschte Endergebnis (Session existiert nicht) erreicht wird.

Rückgabe (nach Statuscodes)

- 204 No Content: Die Session wurde gelöscht (oder existierte nicht).

2.4 /sessions/{sid}/tokens

Tokens zu einer Sitzung. Ein *Token* repräsentiert eine Zugriffsberechtigung für eine bestimmte Aktion, zum Beispiel das Anlegen eines Patienten. Jedes Token wird identifiziert durch eine zufällige Zeichenkette, die *Token-ID*, die lang genug ist, um das Erraten eines Tokens praktisch unmöglich zu machen. Um ein Token einlösen zu können, also die von ihm autorisierte Funktion aufzurufen, reicht die Kenntnis der Token-ID aus. Dadurch ist es möglich, Berechtigungen an Drittsysteme (zum Beispiel dem Webbrowser eines Benutzers) zu übertragen, auch wenn diese der Mainzelliste nicht als zugreifendes System bekannt sind.

Ein Token kann nur eingelöst werden, solange es *gültig* ist. Dafür müssen folgende Bedingungen erfüllt sein:

- Die Session, zu der das Token gehört, existiert. Beispielsweise wird ein Token ungültig, sobald die beinhaltende Session durch einen Timeout abläuft (vgl. Hinweis zur Gültigkeitsdauer einer Session in Abschnitt 2.3).
- Das Token existiert.
- Der Typ des Tokens passt zur aufgerufenen Funktion, zum Beispiel *addPatient* zum Anlegen eines neuen Patienten.

Tokens vom Typ *addPatient* werden bei erfolgreicher Benutzung invalidiert, können also nur einmal benutzt werden.

Eine Beschreibung der definierten Tokentypen findet sich in der Beschreibung der Datentypen (Abschnitt 3.2).

Token anlegen (POST)

- Content-Type der Eingabe (Header Content-Type): application/json

- Content-Type der Ausgabe (Header Accept): application/json
- Header:
  - mainzellisteApiKey: Authentifizierungsschlüssel
- Daten: Tokenobjekt gemäß Datenbeschreibung (vgl. Abschnitt 3.2). Die Felder id und uri müssen nicht vorhanden sein und werden ansonsten ignoriert (ID und URI werden vom Server generiert und zurückgegeben).

Rückgabe (nach Statuscodes)

- 201 Created: Token wurde angelegt. Rückgabe des Tokenobjekts sowie URI der Session im Header Location.
- 400 Bad Request: Es liegt ein Fehler in den übergebenen Daten vor. Zur genaueren Fehlerdiagnose wird eine Fehlermeldung ausgegeben.
- 401 Unauthorized: Die Authentifizierung ist fehlgeschlagen (falscher oder fehlender Authentifizierungsschlüssel, IP-Adresse des aufrufenden Systems ist nicht freigeschaltet oder Berechtigung für diese Aktion fehlt).
- 404 Not Found: Die angegebene Session existiert nicht.

Token abrufen (GET /sessions/{sid}/tokens/{tid}

Abruf des Tokens mit Token-ID *{tid}* aus der Session mit der ID *{sid}*.

- Aufrufender: MDAT-Server.
- Content-Type der Ausgabe (Header Accept): application/json
- Erforderliche Berechtigung: Keine (Kenntnis von Session-ID und Token-ID reicht aus)

Rückgabe (nach Statuscodes)

- 200 Ok: Anfrage erfolgreich, Rückgabe des Tokenobjekts.
- 404 Not Found: Es existiert kein Token unter der angegebenen URI. Das beinhaltet den Fall, dass die in der Aufruf-URI angegebene Session nicht existiert.

Token löschen (DELETE /sessions/{sid}/tokens/{tid})

Löscht das Token mit der ID *{tid}* aus der Session mit der ID *{sid}*.

- Aufrufender: MDAT-Server.
- Erforderliche Berechtigung: Keine (Kenntnis von Session-ID und Token-ID reicht aus).

Das Löschen eines nicht existierenden Tokens einer existierenden Session führt nicht zu einem Fehler, da das gewünschte Endergebnis (Token existiert nicht) erreicht wird. Dagegen muss die angegebene Session existieren, da die Kenntnis der Session-ID zur Autorisierung erforderlich ist.

Rückgabe (nach Statuscodes)

- 204 No Content: Anfrage erfolgreich. Das Token wurde gelöscht (oder existierte nicht).
- 404 Not Found: Es existiert keine Session mit der angegebenen ID.

2.5 /html

Webseiten für den Zugriff mit einem Webbrowser. Alle Zugriffe erfolgen, falls nicht abweichend genannt, per GET. In der Regel wird der Benutzer vom Clientsystem (z.B. Registersoftware), welches die Patientenliste anspricht, auf die jeweilige Seite weitergeleitet (siehe auch Anwendungsfall „Mainzelliste – Zugriff durch MDAT-Server im Anwendungsfall ‚behandlungsfernes Register'").

Abhängig von der Implementierung können die Webseiten in mehreren Sprachen angeboten werden. Die Sprachwahl erfolgt über Angabe des Sprachcodes entweder als URL-Parameter language (dieser hat Vorrang) oder im Header Accept-Language (vgl. [1], Abschnitt 5.3.5).

/html/createPatient: Formular zum Anlegen eines Patienten

Gibt ein Formular zur Anzeige im Webbrowser zurück, mit dem ein menschlicher Benutzer mittels Eingabe identifizierender Daten einen Patienten anlegen kann.

- Aufrufender: Webbrowser eines menschlichen Benutzers
- Methode: GET
- Content-Type der Ausgabe (Header Accept): text/html
- URL-Parameter: *tokenId*: ID eines gültigen *addPatient*-Tokens

Rückgabe (nach Statuscodes)

- 200 OK: Rückgabe des Formulars
- 401 Unauthorized: Das übergebene Token ist ungültig.

/html/editPatient: Formular zum Ändern eines Patienten

Gibt ein Formular zur Anzeige im Webbrowser aus, mit dem ein menschlicher Benutzer die identifizierenden Daten eines Patienten ändern kann.

- Aufrufender: Webbrowser eines menschlichen Benutzers
- Methode: GET
- Content-Type der Ausgabe (Header Accept): text/html
- URL-Parameter: *tokenId*: ID eines gültigen *editPatient*-Tokens

Rückgabe (nach Statuscodes)

- 200 OK: Rückgabe des Formulars
- 401 Unauthorized: Das übergebene Token ist ungültig.

3. Datentypen

Im Folgenden werden die Datentypen, welche die Schnittstelle ausgibt, beschrieben. Die Beispiele sind im JSON-Format. Gemäß der JSON-Spezifikation [4] kommen als Datenstrukturen unter anderem Arrays (geordnete Listen) sowie Objekte (assoziative Arrays, also Schlüssel-Wert-Paare) vor. Alle Daten sind UTF-8-kodiert. Umbrüche und Einrückungen in den Beispielen wurden der Lesbarkeit halber hinzugefügt und müssen nicht Teil der tatsächlichen Ausgabe sein.

3.1 Session

Objekt mit den Feldern:

- sessionId: String. Ein Universally Unique Identifier (UUID [5]), der die Session identifiziert.
- uri: String. URI der Session.

Beispiel:

```
{
  "sessionId":"d81793a2-bad3-44d1-9496-65ae16814979",
  "uri":"https:\/\/mainzelliste.de\/bsp\/sessions\/d81793a2-bad3-44d1-9496-
65ae16814979",
```

```
  "tokens":[
    {
      "id":"",
      "uri":"",
    },
    {
      "id":"",
      "uri":"",
    },
  ]
}
```

3.2 Token

(Berechtigungs-)Objekt mit den Feldern:

- id: String. Eindeutiger Bezeichner.
- type: String. Bezeichnet den Tokentyp (s.u.), und damit die Art der Berechtigung.
- data: Objekt mit weiteren Daten, abhängig vom Tokentyp.
- uri: URI des Tokens

Es folgt eine Beschreibung der möglichen Tokentypen, ihrer Funktion sowie der möglichen Felder im Datenobjekt.

addPatient

Berechtigt zum Anlegen genau eines neuen Patienten, d.h. zum Abruf einer ID mittels Eingabe von IDAT. Das Token wird nach erfolgreicher Benutzung ungültig.

Datenfelder:

- idTypes: Liste von Strings: Pseudonymtypen, die erzeugt und zurückgegeben werden sollen (vgl. Abschnitt 3.3). Falls für die Instanz nur ein ID-Typ konfiguriert ist, kann auf diese Angabe verzichtet werden.
- fields: Objekt mit vordefinierten Feldern (Feldnamen als Schlüssel). Damit können Eingabefelder serverseitig ausgefüllt werden.
- ids: Objekt mit vordefinierten IDs (ID-Typen als Schlüssel). Damit können extern generierte IDs serverseitig definiert werden.
- callback: String. Callback-URL (siehe Beschreibung der Callback-Funktionalität).
- redirect: String. Redirect-URL (siehe Beschreibung der Redirect-Funktionalität).

Beispiel:

```
{
  "id":"97d2424f-dd3d-42de-ad9f-8b604b1c57ad",
  "type":"addPatient",
  "data":{
    "idtypes":["pid"],
    "fields":{
      "aufnahmedatum":"12.1.2014",
      "aufnahmeort":"Mainz"
    },
    "ids":{
      "extid":"12345"
    },
    "callback":"https:\/\/example.org/mdat/patientCreated.php",
```

```
    "redirect":
       "https:\/\/example.org/mdat/addNewPatientMDAT.php?pid={pid}&tid={tokenId}"
  },
  "uri":"http:\/\/localhost:8080\/mainzelliste\/sessions\/7629d1dd-5d9b-4c4c-8b90-
7f64b44a3159\/tokens\/97d2424f-dd3d-42de-ad9f-8b604b1c57ad"
}
```

Redirect

Über den Parameter *redirect* kann eine URL angegeben werden, auf die der Browser nach erfolgreichem Abfragen einer ID verwiesen wird. Damit kann zum Beispiel auf ein Formular zur Eingabe von medizinischen Daten auf dem MDAT-Server weitergeleitet werden. Die URL kann gemäß des Formats für URL-Templates [6][6] parametrisiert werden. Dabei gilt:

- Als Parameter sind die Bezeichner der für die Instanz konfigurierten ID-Typen möglich.
- Die Parameter werden in geschweifte Klammern ersetzt.
- In der Aufrufadresse wird jeder Parameter durch die generierte ID des jeweiligen Typs ersetzt.
- Der spezielle Parameter {tokenId} wird durch die ID des zugehörigen Tokens ersetzt.

Beispiel:

- In einem Token sei definiert:
  `"redirect":"https://example.org/mdat/?pid={pid}&token={tokenId}"`.
- Die zurückgegebene Token-ID sei `a593edf7-2a86-4c34-a6d7-46a5935adb1e`.
- Es werde ein Patient mit der ID „000LP0WE“ vom Typ „pid“ angelegt.

Der Redirect-Aufruf lautet in diesem Fall:

GET `https://example.org/mdat/?pid=000LP0WE&token=a593edf7-2a86-4c34-a6d7-46a5935adb1e`

Da die Redirect-URL im Browser des Benutzers sichtbar ist, darf sie nur zur Übertragung von Informationen genutzt werden, die der Benutzer kennen darf. Für die Übertragung ‚geheimer‘ Daten, zum Beispiel eines nur MDAT- und IDAT-Server bekannten Pseudonyms, sollte die Callback-Funktionalität (siehe folgender Abschnitt) genutzt werden. Eine Zuordnung zwischen Callback- und Redirectaufruf ist über die Token-ID, die ohnehin für den Benutzer sichtbar ist, möglich.

Callback

Die im Feld *callback* angegebene URL wird nach erfolgreichem Anlegen des Patienten, aber vor Übermittlung der Antwort auf den Request *POST /patients* aufgerufen. Dabei wird mittels eines POST-Requests ein JSON-Objekt mit folgenden Feldern übermittelt:

- *tokenId*: ID des Tokens, mit dem der Patient angelegt wurde. Erlaubt zum Beispiel die Zuordnung des gelieferten PIDs zu den Patientendaten auf dem MDAT.
- *ids*: Array von IDs, Format wie in Abschnitt 3.3 beschrieben. Enthält die für den Patienten vergebenen IDs aller im Feld *idTypes* des Tokens angegebenen ID-Typen.

Beispiel für das im Callback übermittelte Objekt:

```
{
  "tokenId":"a593edf7-2a86-4c34-a6d7-46a5935adb1e",
  "ids":
  [
```

```
    {
      "idType":"pid",
      "idString":"000LP0WE"
    }
  ]
}
```

readPatients

Berechtigt zum Auslesen von Patientendaten (IDAT und/oder Pseudonyme). Dieses Token bleibt für die Lebensdauer der beinhaltenden Session gültig.

Datenfelder:

- *searchIds*: Array von IDs, die den gesuchten Patienten zugeordnet sind. Format der IDs wie im Abschnitt 3.3 beschrieben.
- *resultFields*: Array von Strings. Die Felder, die ausgegeben werden sollen.
- *resultIds*: Array von Strings. Namen der IDs, die ausgegeben werden sollen.

Beispiel: Zu den Patienten mit den IDs „9U5HWV3E“ und „AG3JK81L“ vom Typ „pid“ sollen jeweils Geburtsjahr, Wohnort sowie die ID vom Typ „labid“ ausgegeben werden. Das anzulegende Token hat folgende Form:

```
{
  "type": "readPatients",
  "data": {
    "searchIds": [
      {
        "idType":"pid",
        "idString":"9U5HWV3E"
      },
      {
        "idType":"pid",
        "idString":"AG3JK81L"
      }
    ],
    "resultFields":["geburtsjahr","wohnort"],
    "resultIds":["labid"]
    }
}
```

editPatient

Berechtigt zum Ersetzen (Editieren) von Patientendaten (vgl. Abschnitt „Daten eines Patienten ändern“, S. 8).

Datenfelder:

- *patientId*: ID des zu ändernden Patienten.
- *fields*: Array von Strings. Bezeichnet die Felder, die mit dem Token geändert werden dürfen.
- *ids*: Array von Strings. Bezeichnet alle externen IDs (ID-Typen), die mit dem Token geändert werden dürfen.
- *redirect*: URL, auf die nach Abschließen des Editiervorgangs weitergeleitet werden soll. Falls dieses Feld gesetzt ist, antwortet die Mainzelliste auf den PUT-Request zum Ändern des Patienten mit dem Statuscode 303 (See other) und setzt den Header *Location* auf diesen Wert (vgl. S. 8). Kann zum Beispiel genutzt werden, um nach dem Editieren in die Stammdatenansicht in der MDAT-Anwendung zurückzuleiten.

Ab Schnittstellenversion 3.0 muss mindestens eines der Felder fields und ids definiert sein. Die bisherige Möglichkeit, dass bei Angabe nur von patientId und Auslassen von fields alle Felder bearbeitet werden können, ist ab Schnittstellenversion 3.0 ungültig.

Beispiel 1 (Ändern der Felder „vorname“ und „nachname“ möglich, mit Redirect):

```
{
  "type": "editPatient",
  "data": {
    "patientId":
    {
      "idType":"pid",
      "idString":"9U5HWV3E"
    },
    "fields":["vorname", "nachname"],
    "redirect":"https://mdat.example.org/patientList"
  }
}
```

Beispiel 2 (Ändern der IDs „EXT-ID“ und „LAB-ID“ möglich):

```
{
  "type": "editPatient",
  "data": {
    "patientId":
    {
      "idType":"pid",
      "idString":"9U5HWV3E"
    },
  "ids":["EXT-ID","LAB-ID"]
}
```

Beispiel 3 (Ändern der ID „EXT-ID“ und des Felds „wohnort“ möglich):

```
{
  "type": "editPatient",
  "data": {
    "patientId":
    {
      "idType":"pid",
      "idString":"9U5HWV3E"
    },
  "fields":["wohnort"],
  "ids":["EXT-ID"]
}
```

3.3 ID

Repräsentiert ein Pseudonym. Objekt mit den Feldern:

- *idType*: String. Typ des Pseudonyms (s.u.)
- *idString*: String. Wert des Pseudonyms.
- *tentative*: Boolean. Gibt an, ob das Pseudonym als vorläufig zu betrachten ist, d.h. der zugehörige Patient möglicherweise ein Duplikat ist (vgl. „Behandlung unsicherer Matches“, S. 7).
- *uri*: URI der ID (für zukünftige Anwendungsfälle).

Der in *idString* enthaltene Wert ist das Pseudonym an sich, d.h. die Zeichenkette, die den Patienten identifiziert. Um einen Patienten mehrere Pseudonyme zuordnen zu können (z.B. für verschiedene Subsysteme), ist für jedes Pseudonym außerdem ein Typbezeichner festgelegt. Vgl. auch die Beschreibung von ID-Typen im Konfigurationshandbuch (Abschnitt „ID-Erzeugung“).

Beispiele:

```
 {
  "idType":"pid",
  "idString":"000GR0W0",
  "tentative":false,
  "uri":"https:\/\/example.org/mainzelliste/patients/pid/000GR0W0"
}
```

```
{
  "idType":"labid",
  "idString":"xzf439",
  "tentative":true,
  "uri":"https:\/\/example.org/mainzelliste/patients/labid/xzf439"
}
```

3.4 Patient

Repräsentiert einen Patienten der Patientenliste.

Felder:

- *fields*: IDAT-Felder als Schlüssel-Wert-Paare (Feldnamen als Schlüssel).
- *ids*: Array von IDs des Patienten.

Ob und welche dieser Informationen in einem zurückgegebenen Objekt tatsächlich vorhanden sind, ist kontextabhängig. Zum Beispiel enthält ein Patientenobjekt, das auf eine Anfrage „GET /patients“ zurückgegeben wird, nur diejenigen Felder und IDs, die im zugehörigen *readPatients*-Token angegeben sind (vgl. „Patientenliste abrufen (GET)“, S. 8, sowie Tokentyp *readPatients*, S. 15). Wird in einem *readPatients*-Token ein externer ID-Typ angefordert, kann es außerdem vorkommen, dass für einen Patienten keine ID dieses Typs verfügbar ist – diese fehlt dann in der Ausgabe. Zugreifende Anwendungen sollten deshalb keine Annahmen treffen, welche Felder und IDs in einem Objekt tatsächlich vorhanden sind und gegebenenfalls eine Fehlerbehandlung vornehmen.

Beispiel (korrespondierend zum Beispiel des *readPatients*-Token in Abschnitt 3.2, in diesem Fall wurden Wohnort, Geburtsjahr und das Pseudonym *labid* angefordert):

```
[
  {
    "fields":{
      "geburtsjahr":"1950",
      "wohnort":"Darmstadt"
    },
    "ids":[
      {
        "idType":"labid",
        "idString":"12345"
      }
    ]
```

```
  },
  {
    "fields":{
      "geburtsjahr":"1980",
      "wohnort":"Frankfurt a.M."
    },
    "ids":[
      {
        "idType":"labid",
        "idString":"54321"
      }
    ]
  }
]
```

4. Literatur

1. Lablans M, Borg A, Ückert F (2015) A RESTful interface to pseudonymization services in modern web applications. BMC Med Inform Decis Mak 15: 2. doi: 10.1186/s12911-014-0123-5
2. Preston-Werner T (2015) Semantic Versioning 2.0.0. Verfügbar unter http://semver.org/spec/v2.0.0.html
3. Fielding R, Reschke J (2015) Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. Verfügbar unter http://tools.ietf.org/html/rfc7231
4. Bray T (2015) The JavaScript Object Notation (JSON) Data Interchange Format. Verfügbar unter http://tools.ietf.org/html/rfc7159
5. Leach P, Mealling M, Salz R (2015) A Universally Unique IDentifier (UUID) URN Namespace. Verfügbar unter http://tools.ietf.org/html/rfc4122
6. Gregorio J, Fielding R, Hadley M, Nottingham M, Orchard D (2015) URI Template. Verfügbar unter http://tools.ietf.org/html/rfc6570