Detailbeschreibung der REST-Schnittstelle, v1.0

Jiri Kraml

22. Januar 2015

1 Einleitung

Dieses Dokument entält eine detaillierte Beschreibung der Ressourcen der REST-Schnittstelle in Version 1.0.

Jede Ressource hat einen eigenen Abschnitt mit ihrem Pfad als Titel. Dieser gliedert sich weiter nach den HTTP-Methoden, mit denen Anfragen an die Ressource gestellt werden können.

Ressourcen mit verwandten Aufgaben sind zu größeren Abschnitten zusammengefasst.

Die Beschreibungen der verfügbaren Methoden einer Ressource listen die Parameter der Anfrage und die möglichen Rückgabewerte der Antwort auf. Parameter sind entweder vom Typ Form oder vom Typ Query.

Form-Parameter werden im Inhalt der HTTP-Anfrage übertragen. Formulare in HTML-Dokumenten (Forms) erzeugen solche Anfrage, wenn sie mit den Methoden POST oder PUT abgeschickt werden.

Query-Parameter sind Teil des sogenannten Query-Strings, einer Zeichenkette, die an die URL einer Anfrage angehägt werden kann.

Es folgt ein Beispiel für eine Ressource.

1.1 /Pfad/zu/BeispielRessource

Beschreibung der Ressource.

1.1.1 GET

(Query)	*Name eines Query-String-Parameters*
...	...

Beschreibung des Effekts einer GET-Anfrage auf die Ressource.

`200`	OK	Bedeutung und ggf. Inhalt der Antwort
`412`	Precondition Failed	Bedeutung und ggf. Inhalt der Antwort
...		...

1.1.2 POST

(Form)	*Name eines Formular-Parameters*
...	...

Beschreibung des Effekts einer POST-Anfrage auf die Ressource.

`201`	Created	Bedeutung und ggf. Inhalt der Antwort
`400`	Bad Request	Bedeutung und ggf. Inhalt der Antwort
...		...

2 Accountverwaltung

Accounts unterscheidet die REST-Schnittstelle anhand ihrer ID. Diese ist zwar ein numerischer Wert, wird aber als Zeichenkette übergeben. In Javascript geschriebene Clients könnten diese großen ganzzahligen Werte sonst auf die nächste Gleitkommazahl runden, da diese Programmiersprache keine „long“-Werte unterstützt.

2.1 /active_account

Diese Ressource verwaltet den derzeit aktiven Account und kann ihn umschalten.

2.1.1 GET

— keine Parameter —

Gibt ein JSON-Objekt mit der Nummer des aktiven Accounts zurück.

Format:

```
{ activeAccount: string }
```

`200` OK —

2.1.2 PUT

(Form) *accountNumber*

Schaltet auf den Account mit Nummer *accountNumber* um.

`200` OK	—
`400` Bad Request	*accountNumber* ist keine gültige Zahl
`404` Not Found	Account mit der Nummer *accountNumber* existiert nicht

---

2.2 /accounts

Diese Ressource enthält eine Liste mit den verfügbaren Account-IDs.

2.2.1 GET

— keine Parameter —

Gibt eine Liste der vorhandenen Account-Identifier zurück.

Format:

```
[ string , string , string , ... ]
```

`200` OK —

2.2.2 POST

(Form) *file*
(Form) *password*
(Form) *successRedirectUrl*
(Form) *failureRedirectUrl*

Importiert die Account aus der Datei *file*, die optional mit Password *password* Verschlüsselt ist.

`200` OK	Import erfolgreich
`303` See Other	Import erfolgreich, Weiterleitung zu *successRedirectUrl* (falls angegeben)
`400` Bad Request	Import fehlgeschlagen, da Anfrage nicht verstanden oder keine Datei enthalten
`412` Precondition Failed	Import fehlgeschlagen (möglicherweise Passwort falsch)
`303` See Other	Import fehlgeschlagen, Weiterleitung zu *failureRedirectUrl* (falls angegeben)

2.3 /accounts/export

Diese Ressource ermöglicht den Export der Accounts als XML-Datei.

2.3.1 GET

(Query) *password*

Exportiert eine XML-Datei mit allen Accounts, optional Verschlüsselt mit *password*

`200` OK Eine Datei namens „accounts.xml“ wird gesendet

2.4 /account/*accountNumber*

Für jeden Account gibt es eine solche Ressource. Sie enthält Details zum Account und der Account kann über sie gelöscht werden.

2.4.1 GET

— keine Parameter —

Liefert Details zu Account *accountNumber*.

Format:

```
{
  accountNumber: string,
  expiresOn: number,
  hasExpired: boolean,
  currentVolume: number,
  monthlyVolume: number,
  isActive: boolean,
  isBlocked: boolean
}
```

200	OK	Gibt ein JSON-Objekt mit Details zurück
400	Bad Request	*accountNumber* ist keine gültige Zahl
404	Not Found	Account mit der Nummer *accountNumber* existiert nicht

2.4.2 DELETE

— keine Parameter —

Löscht Account *accountNumber*.

200	OK	Löschung erfolgreich
400	Bad Request	*accountNumber* ist keine gültige Zahl
404	Not Found	Account mit der Nummer *accountNumber* existiert nicht

---

2.5 /account/*accountNr*/update

Diese Ressource löst ein Update der Accountdaten aus. Normalerweise ist es nicht nötig, diese Ressource anzusprechen, da der JonDoController die Daten automatisch regelmäßig aktualisiert.

2.5.1 POST

— keine Parameter —

Löst ein Update der Informationen für Account mit ID *accountNr* aus.

200	OK	Update wurde ausgelöst

2.6 /coupons

Diese Ressource enthält eine Liste der vergebenen Ticket-IDs. Es gibt eine Ticket-ID wird für jeden Einlösevorgang eines Couponcodes. Das Anstoßen neuer Einlösevorgänge ist ebenfalls Aufgabe dieser Ressource.

2.6.1 GET

— keine Parameter —

Gibt eine Liste der vorhandenen Ticket-IDs zurück.

Format:

```
[ string , string , string , ... ]
```

`200` OK Rückgabe der Liste

2.6.2 POST

(Form) *couponCode*
(Form) *redirectUrl*

Startet das Einlösen des Couponcodes *couponCode* und weist den Vorgang eine Ticket-ID zu, die an den Client zurückgeschickt wird. Der Client kann im Feld *redirectUrl* optional eine URL angeben, zu der er weitergeleitet wird, wenn die Anfrage erfolgreich ist.

`201` Created	Vorgang wurde gestartet
`303` See Other	Vorgang wurde gestartet, Weiterleitung zu *redirectUrl* (falls angegeben)
`400` Bad Request	*couponCode* ist nicht enthalten

2.7 /coupon/*ticketId*

Diese Ressource repräsentiert einen Einlösevorgang für einen CouponCode.

2.7.1 GET

— keine Parameter —

Gibt ein JSON-Objekt mit Status des Couponcode-Vorgangs mit Ticket-ID *ticketId* zurück. Das Objekt enthält die angeforderte Ticket-ID, den Code, der eingelöst wird/wurde, und den Status des Vorgangs.

Der Status wird sowohl als textuelle („name“), als auch als numerische („value“) Konstante übergeben. Die möglichem Werte sind wie folgt:

Zahl	Text	Bedeutung
0	IN_PROGRESS	Der Vorgang ist nocht nicht abgeschlossen
200	SUCCESS	Der Couponcode wurde eingelöst
400	INVALID_CODE	Der Couponcode hat ein ungültiges Format
500	FAILURE	Das Einlösen ist fehlgeschlagen

Format des JSON-Objekts:

```
{
  ticketId : string,
  code : string,
  name : string,
  value : number
}
```

200	OK	JSON-Objeckt wird zurückgegeben
404	Not Found	Es existiert kein Vorgang mit der ID *ticketId*

2.7.2 DELETE

— keine Parameter —

Löscht das Einlöse-Ticket mit ID *ticketId*.

200	OK	Rückgabe der Liste
404	Not Found	Es existiert kein Vorgang mit der ID *ticketId*

---

3 Verwaltung der Mixkaskaden

3.1 /blacklist/*serviceId*

Diese Ressource stellt einen Service auf der Blacklist da. Sie ist nur vorhanden, wenn sich ein Service auf der Blacklist befindet.

3.1.1 PUT

— keine Parameter —

Versucht den Service mit ID *serviceId* auf die Blacklist zu setzen. Falls ein solcher Service nicht existiert, bleibt der Aufruf ohne Folgen.

200 OK —

3.1.2 DELETE

— keine Parameter —

Versucht den Service mit ID *serviceId* von der Blacklist zu nehmen. Falls der Service nicht auf der Blacklist ist, oder ein solcher Service nicht existiert, bleibt der Aufruf ohne Folgen.

200 OK —

3.2 /services

Diese Ressource enthält die Liste der verfügbaren Anonymisierungsdienste.

3.2.1 GET

(Query) *filter*

Liefert eine Liste der verfügbaren Services. Über den optionalen Parameter *filter* lässt sich die Auswahl auf bezahlte oder kostenlose Angebote einschränken. Zulässige Werte für *filter* sind „all“, „free“, „premium“. Auf Groß- und Kleinschreibung muss nicht geachtet werden. Falls der Wert ungültig oder nicht vorhanden ist, wird „all“ angenommen und alle Dienste werden in der Liste zurückgegeben.

Die Elemente der Liste sind keine einfachen Identifier, sondern Dienstobjekte (eins pro Dienst). Jedes der Objekte enthält Namen und Identifier des jeweiligen Dienstes und gibt an, ob es sich um einen Premiumdienst handelt und ob der Dienst auf der Blacklist steht.

Format der Liste JSON-Objekts:

[*Dienstobjekt* , *Dienstobjekt* , *Dienstobjekt* , ...]

Format der Dienstobjekte:

```
{
  serviceId : string,
  serviceName : string,
  isBlacklisted : boolean,
  isPremium : boolean
}
```

200 OK Rückgabe der Liste

3.3 /service/*serviceId*

Für jeden Anonymisierungsdienst gibt es eine Ressource, die Details über ihn liefert.

3.3.1 GET

— keine Parameter —

Liefert ausführliche Informationen zum Service mit ID *serviceId*. Der Service wird durch ein kompliziertes JSON-Objekt mit einigen Unterobjekten beschreiben.

Format des Hauptobjekts:

```
{
  id: string,
  name: string,
  isBlacklisted: boolean,
  isPremium: boolean,
  userCount: integer,
  userLimit: integer,
  mixInfos: Liste von Mixinfo-Objekten,
  speed: string,
  delay: string,
  exitIP: string,
  lastMixCountryCode: string
}
```

Format der Mixinfo-Objekte:

```
{
  mixName: string,
  location:
    {
      countryCode: string,
      city: string,
      state: string,
      longitude: string,
      latitude: string
    },
  operator:
    {
      countryCode: string,
      email: string,
      web: string,
      organization: string,
      organizationUnit: string,
      organizationShort: string
    }
}
```

Die Ländercodes („countryCode“, „lastMixCountryCode“) in den Objekten bestehen aus zwei Buchstaben und entstammen der Norm ISO 3166-1.

`200` OK Rückgabe des JSON-Objekts

3.4 /service/*serviceId*/update

Diese Ressource löst ein Update der Daten zur Mixkaskade aus. Normalerweise ist es nicht nötig, diese Ressource anzusprechen, da der JonDoController die Daten automatisch regelmäßig aktualisiert.

3.4.1 POST

— keine Parameter —

Triggert ein Update der Informationen für den Service *serviceId*.

`200` OK —

---

4 Verbindungsmanagement

4.1 /connection

Die Ressource repräsentiert den Verbindungszustand des JonDoControllers.

4.1.1 GET

— keine Parameter —

Liefert Informationen zur aktuellen Verbindung in Form eines JSON-Objekts. Neben den Identifier des aktiven Accounts und des aktuell verbundenen Dienstes werden der Verbindungszustand des JonDoControllers, das verbrauchte und das noch verfügbare Datenvolumen zurückgegeben. Der Verbindungszustand wird durch die boolschen Werte „isConnected“ und „isConnecting“ beschreiben. Zu beachten ist, dass der Controller — während er noch im Prozess des Verbindens ist — vorgibt, verbunden zu sein und den Service, zu dem er sich gerade verbindet, als den aktuellen Service angibt.

Die Felder „remainingCredit“ und „trafficBytes“ machen ihre Angaben in Bytes. Ersteres beschreibt dabei das noch verfügbare Datenvolumen für Premiumdienste.

Format des Objekts:

```
{
  activeAccount: string,
  isConnected: boolean,
  isConnecting: boolean,
  currentService: string,
  trafficBytes: number,
  remainingCredit: number
}
```

`200` OK —

4.2 /connection/connect

4.2.1 POST

(Query) *serviceId*

Baut Verbindung zu Service mit ID *serviceId* auf.

`200`	OK	Verbindung wird aufgebaut
`404`	Not Found	Es existiert kein Service mit ID *serviceId*
`403`	Forbidden	Der Service steht auf der Blacklist

4.3 /connection/disconnect

4.3.1 POST

— keine Parameter —

Trennt aktuelle Verbindung. Falls keine Verbindung besteht, hat der Aufruf keine Wirkung.

`200`	OK	Verbindung wird getrennt (falls eine besteht)

4.4 /connection/http_filter

4.4.1 GET

— keine Parameter —

Gibt aktuellen Status des HTTP Filters in einem einfachen JSON-Objekt zurück.

Format des Objekts:

```
{ enabled: boolean }
```

`200 OK` Verbindung wird getrennt (falls eine besteht)

4.4.2 PUT

(Form) *serviceId*

Aktiviert oder deaktiviert den HTTP Filter, je nach Wert im Feld *enabled*. Als boolscher Wert „wahr“ werden alle Zeichenketten interpretiert, die „true“ entsprechen, Groß- und Kleinschreibung ist dabei egal. Alle anderen werte gelten als „falsch“.

`200 OK` Wert wird setzt

---

5 Andere Ressourcen

5.1 /extraSettings

Die Extra-Settings-Ressource ermöglicht dem Client das Speichern beliebiger Daten in der REST-Schnittstelle. Das Programm, zu dem die Schnittstelle gehört, kann diese Daten einsehen und verändern, wodurch eine Kommunikation mit dem Client möglich ist.

5.1.1 GET

— keine Parameter —

Gibt alle Schlüssel-Wert-Paare zurück, die zuvor mit PUT gesetzt wurden. Die Paare werden als JSON-Objekt ausgegeben.

Format des Objekts:

```
{
  Schluessel1: Wert1,
  Schluessel2: Wert2,
   ...
}
```

`200 OK` Werte werden ausgegeben

5.1.2 PUT

(Form) *beliebig*
(Form) ...

Speichert alle übergebenen Schlüssel-Wert-Paare, diese können dann per GET abgerufen werden.

`200` OK Werte wurden gesetzt

5.2 /update_data

Diese Ressource löst ein Update der Daten von Infoservice aus.. Normalerweise ist es nicht nötig, diese Ressource anzusprechen, da der JonDoController die Daten automatisch regelmäßig aktualisiert.

5.2.1 POST

(Form) *beliebig*
(Form) ...

Stößt ein Update aller Daten vom Infoservice an.

`200` OK Update wurde angestoßen

5.3 /version

Diese Ressource steht für die Version der REST-Schnittstelle aus sicht des Clients. Die Version des Softwarepakets, das die Schnittstelle implementiert, wird hier nicht angegeben. Veränderungen an der Schnittstelle sollen zu verschiedenen Versionsnummern führen.

5.3.1 GET

— keine Parameter —

Gibt die Version der Schnittstelle als einfache Zeichenkette zurück.

`200` OK Version wird zurückgegeben