strongTNC REST API Draft 1

Part of the strongTNC Bachelor's Thesis

Danilo Bargen, Christian Fässler, Jonas Furrer

Spring 2014

Contents

1 Introduction

The REST resource design is based on the recommendations of the *REST API Design Rulebook*[1] from O'Reilly.

URI Definition

URIs[1] are defined as follows, according to RFC 3986[2]:

```
URI = scheme "://" authority "/" path [ "?" query ] [ "#" fragment ]
```

Resource-Archetypes

We used the resource archetype terminology from Masse 2011[1]. The explanation texts are taken directly from said book.

Document A document resource is a singular concept that is akin to an object instance or database record. A document's state representation typically includes both fields with values and links to other related resources.

Collection A collection resource is a server-managed directory of resources. Clients may propose new resources to be added to a collection. However, it is up to the collection to choose to create a new resource, or not.

Store A store is a client-managed resource repository. A store resource lets an API client put resources in, get them back out, and decide when to delete them. On their own, stores do not create new resources; therefore a store never generates new URIs. Instead, each stored resource has a URI that was chosen by a client when it was initially put into the store.

Controller A controller resource models a procedural concept. Controller resources are like executable functions, with parameters and return values; inputs and outputs. Like a traditional web application's use of HTML forms, a REST API relies on controller resources to perform application-specific actions that cannot be logically mapped to one of the standard methods (create, retrieve, update, and delete, also known as CRUD).

[1]Uniform Resource Identifier

2 Archetype Representation

The JSON representation in this API documentation depends on the resource archetype.

Document A document returns a JSON object, which contains all relevant fields.
If a document contains references to other resources, then this reference is returned as an URI. Additionally, a `depth` query parameter can be specified to embed nested resources directly in the response.
Example:

```
{
  "field1": "<str,value-1>",
  "field2": <int,value-2>,
  "field3": <bool,value-3>,
  "subObject": "<uri,sub-object>"
  "subCollection": [
    {
      "uri": "<uri,sub-object>"
    },
    {
      "uri": "<uri,sub-object>"
    }
  ]
}
```

Example with `depth=1`:

```
{
  "field1": "<str,value-1>",
  "field2": <int,value-2>,
  "field3": <bool,value-3>,
  "subObject": {
    "field1": <int,value-1>,
    ...
    "uri": "<uri,sub-object>"
  }
  "subCollection": [
    {
      "field": <int,field>,
      ...
      "uri": "<uri,sub-object>"
    },
    {
      "field": <int,field>,
      ...
      "uri": "<uri,sub-object>"
    }
  ]
}
```

Collection A collection returns a (possibly filtered) list of all contained documents as JSON-objects. The documents are annotated with an additional field called `uri`, which contains the URI to the document resource.

Collections too allow the usage of a `depth` parameter to embed nested objects.

Example:

```
{
  {
    "field1": "<str,value-1>",
    "subObject": "<uri,sub-object>",
    "uri": "<uri,objek>"
  },
  {
    "field1": "'<str,value-1>",
    "subObject": "<uri,sub-object>",
    "uri": "<uri,resource>"
  },
}
```

Example with `depth=1`:

```
{
  {
    "field1": "<str,value-1>",
    "subObject": {
      "field": "<str,value>",
    },
    "uri": "<uri,resource>"
  },
  {
    "field1": "'<str,value-1>",
    "subObject": {
      "field": "<str,value>",
    },
    "uri": "<uri,resource>"
  },
}
```

Controller The output of a controller varies, depending on the implementation.

3 HTTP Status Codes

The result of a request is communicated through HTTP status codes. In some cases, additional information is returned in the response body.

The following status codes can be expected:

200 OK The request has succeeded.

201 Created The request has been fulfilled and resulted in a new resource being created.

204 No Content The server has fulfilled the request but does not need to return an entity-body.

400 Bad Request Generic client side error.

404 Not Found The server has not found anything matching the Request-URI.

405 Method Not Allowed The method specified in the Request-Line is not allowed for the resource identified by the Request-URI.

409 Conflict The entity to be created already exists.

412 Precondition Failed Additional steps are necessary to successfully process the sent request.

500 Internal Server Error Generic server side error.

If a resource uses additional error codes, then it is documented in the resource documentation.

4 Further Remarks

4.1 Notation

For the request- and response-format of a resource, pseudo-JSON notation is used. The values are denoted as a tuple in angle brackets. The first part of the tuple denotes the type, while the second is a description of the value.

The following types are used:

int Integer
num Decimal
str String
bool Boolean
xml An XML document
hex A HEX string
uri The fully qualified URI of a resource
doc The resource document

4.2 Default behavior

The following items describe the default behavior of resources:

Depth Query-Parameter Each resource that returns an URI to another resource, supports a `depth` query parameter. This parameter embeds the nested resources into the returned document up to the depth specified by the query parameter. If no parameter is specified, the depth `0` is used.

Filter Query-Parameter Filter query parameters are – if available – always optional. They can be used to filter the returned data.
All Collections support – unless specified otherwise – generic filters on all fields except foreign keys. The parameter looks like this: `fieldName=query`. Filters can also be used on document resources – if the document field values don't match, then a HTTP 404 status code is returned.

Fields Query-Parameter The `fields` query parameter can be used to limit the returned fields of a resource. This is similar to a `SELECT field1, field2, fieldn` SQL statement.

5 Daten Definition

5.1 Policy Arguments

Folgende Workitem, bzw. Policy Typen sind momentan vorhanden:

`00:` `RESVD` Deny
`01:` `PCKGS` Installed Packages
`02:` `UNSRC` Unknown Source
`03:` `FWDEN` Forwarding Enabled
`04:` `PWDEN` Default Password Enabled
`05:` `FREFM` File Reference Measurement
`06:` `FMEAS` File Measurement
`07:` `FMETA` File Metadata
`08:` `DREFM` Directory Reference Measurement
`09:` `DMEAS` Directory Measurement
`10:` `DMETA` Directory Metadata
`11:` `TCPOP` Open TCP Listening Ports
`12:` `TCPBL` Blocked TCP Listening Ports
`13:` `UDPOP` Open UDP Listening Ports
`14:` `UDPBL` Blocked UDP Listening Ports
`15:` `SWIDT` SWID Tag Inventory
`16:` `TPMRA` TPM Remote Attestation

Einige Typen benötigen unterschiedliche Argumente, Gemeinsamkeiten lassen sich wie folgt gruppieren:

Keine Argumente 00, 01, 02, 03, 04
Datei Pfad 05, 06, 07
Verzeichnis Pfad 08, 09, 10
Port Liste 11, 12, 13, 14
SWID Request Flags 15
TPM Attestation Flags 16

Dementsprechend sollen die Argumente der Workitems übermittelt werden. Folgende Objekte werden je nach Type übermittelt, als Verweis wird `<doc,policy-argument>` verwendet.

Keine Argumente

```
{}
```

Datei Pfad

```
{
  "file": "<str,file-path>"
}
```

Verzeichnis Pfad

```
{
  "directory": "<str,directory-path>"
}
```

Port Liste

```
{
  "portList": [
    "<str,port or port-range>"
  ]
}
```

SWID Request Flags

```
{
  "swidFlags": [
    "<str,swid-flag>"
  ]
}
```

TPM Attestation Flags

```
{
  "tpmFlags": [
    "<str,tpm-flag>"
  ]
}
```

5.2 Recommendation Types

Folgende Recommendation Types bzw. Actions sind vorhanden:

`0:` `ALLOW` Die Empfehlung/Aktion ist, den Client zuzulassen
`1:` `BLOCK` Die Empfehlung/Aktion ist, den Client zu blockieren
`2:` `ISOLATE` Die Empfehlung/Aktion ist, den Client in einem isoliertem Segment zu platzieren
`3:` `NONE`

Diese Werte werden von der API als Integer Id verwendet. Es wird kein Objekt für die Repräsentation erstellt.

5.3 Hash-Set

Ein Hash-Set wird innerhalb eines File-Documents ausgeliefert, der Verweis lautet `<doc,hash-set>` und ist wie folgt aufgebaut:

```
[
  {
    "algorithm": "<str,algorithm-name>",
    "hash": "<hex,hash> ",
    "product": "<uri,product>"
  }
]
```

Falls ein Algorithmus noch nicht existiert, wird er erfasst, momentan sind folgende Algorithmen erfasst: `SHA384, SHA256, SHA1, SHA1-IMA`

6 REST Ressourcen

6.1 Session Steuerung und Ablauf

6.1.1 Controller

URI Path `/sessionss/start/`

Archetype Controller

Methods POST

Request Parameter

- `connectionId` strongSwan Connection Id
- `clientIdentity` strongSwan Client-Identity
- `hardwareId` Die ID, welche das Gerät identifiziert, so zum Beispiel AIK, Android-ID, DBUS Machine-ID, o.ä. Dies entspricht dem `value` Feld in der `device` Tabelle in der Datenbank
- `productName` Der Productname ist der Name des OS wie er in der `product` Tabelle der Datenbank steht

JSON Format Response

```
{
  "sessionId": <int,id>,
  "workitems": [
     <doc,workitem>,
     ...
  ],
  "uri": "<uri,session>"
}
```

Beschreibung Dieser Controller erstellt und startet eine Session, das Device, welches der Session zugeordnet werden soll, wird anhand der `hardwareId` und dem `productName` bestimmt. Falls eines der Objekte noch nicht existiert wird dieses durch den Controller erstellt. Die ID, die im Response Dokument zurück geliefert wird, dient zur zukünftigen Identifikation der soeben gestarteten Session. Ausserdem erhält man eine Liste von Workitems, die für diese Session abgearbeitet werden müssen.

URI Path `/sessions/{id}/end/`

Archetype Controller

Methods POST

Request Parameter

- `recommendation` Endgültiges Resultat/Empfehlung für diese Session.

Beschreibung Dieser Controller schliesst die Session ab und startet alle zusätzlich nötigen Vorgänge, so werden zum Beispiel die Workitems dieser Session abgeräumt und die jeweiligen Resultate gespeichert und können via `/sessions/{id}/results` abgerufen werden.

6.1.2 Documents und Collections

URI Path `/sessions/{id}/`

Archetype Readonly Document

Methods GET

JSON Format Response

```
{
  "id": <int,id>,
  "uri": "<uri,resource>",
  "time": <int,time>,
  "identity": "<uri,identity>",
  "connectionId": <int,connection-id>,
  "device": "<uri,device>",
  "recommendation": <int,rec>
}
```

Beschreibung Informationen zu einer bestimmten Session. Sessions sollen nicht direkt geändert werden, sondern nur über die entsprechenden Controller, der Grund dafür ist, dass im Hintergrund noch zusätzliche Operationen vorgenommen werden müssen.

URI Path `/sessions/`

Archetype Readonly Collection

Filter Query

timeFrom `<int,timestamp>`

timeTo `<int,timestamp>`

Methods GET

JSON Format Response

```
[<doc,session>, ...]
```

Beschreibung Liste aller Sessions. Neue Sessions können nur über den entsprechenden Controller erstellt werden.

Workitems

URI Path `/sessions/{id}/workitems/{id}/`
Archetype Readonly Document
Methods GET
JSON Format Response

```
{
  "id": <int,id>,
  "uri": "<uri,resource>",
  "session": "<uri,session>",
  "type": <int,type>,
  "argument": <policy-argument>
}
```

Beschreibung Ein zu einer bestimmten Session zugehöriges Workitem. Workitems können nicht direkt erstellt werden, sondern werden beim Erstellen einer Session anhand der Enforcements erstellt. Workitems gibt es nur, so lange eine Session nicht beendet wurde. Nach dem Ende der Session wird ein `HTTP 404` Statuscode zurückgeliefert.

URI Path `/sessions/{id}/workitems/`
Archetype Readonly Collection
Filter Query
  type `<int,policy-type>`
Methods GET
JSON Format Response

```
[<doc,workitem>, ...]
```

Beschreibung Eine Liste aller Workitems, die zu einer bestimmten Session gehören. Workitems können nicht direkt erstellt werden, sondern werden beim Erstellen einer Session anhand der Enforcements erstellt. Workitems gibt es nur, so lange eine Session nicht beendet wurde. Nach dem Ende der Session wird eine leere Liste geliefert.

URI Path `/sessions/{id}/workitems/{id}/result/`

Archetype Document

Request Parameter

`recommendation` Resultat/Empfehlung für dieses Workitem.

`comment` Kommentar zum Resultat.

Methods GET, POST

Response Statuscodes

201 Created Resultat wurde erfolgreich gespeichert.

409 Conflict Resultat existiert bereits.

JSON Format Response

```
{
  "recommendation": <int,type>,
  "comment": <str,comment>
}
```

Beschreibung Auf dieser Ressource soll das Resultat des jeweiligen Workitems eingetragen werden. Diese Resultate werden beim Beenden der Session von den Workitems in die Session-Resulate übertragen.

Results

URI Path `/sessions/{id}/results/{id}/`
Archetype Readonly Collection
Methods GET
JSON Format Response

```
{
  "id": <int,id>,
  "uri": "<uri,resource>",
  "enforcement": "<uri,enforcement>",
  "recommendation": <int,type>,
  "comment": "<str,comment>"
}
```

Beschreibung Einzelnes Resultat. Im Gegensatz zur Resultat-Resource unter der Workitem Resource enthält dieses Document auch einen Link zum zugehörigen Enforcement.

URI Path `/sessions/{id}/results/`
Archetype Readonly Collection
Methods GET
JSON Format Response

```
[<doc,result>, ...]
```

Beschreibung Nach dem Beenden einer Session können die eingetragenen Resultate in dieser Collection abgefragt werden. Diese Collection ist readonly, damit die Session-Ergebnisse nicht nachträglich geändert werden können.

6.2 SWID Erweiterung

6.2.1 SWID Tags: Messung

URI Path `/sessions/{id}/swid-measurement/`
Archetype Controller
Methods POST
Content-Type `application/json; charset=utf-8`
Request Parameter
`softwareId` Software-IDs als JSON-Liste.
Response Statuscodes
200 OK SWID Tags der übermittelten Software-IDs sind eingetragen und wurden für die übermittelte Session eingetragen.
404 Not Found Session mit der spezifizierten ID wurde nicht gefunden.
412 Precondition Failed Es existieren nicht alle SWID Tags für die übertragenen Software-IDs. Als Payload werden die fehlenden Software-IDs übertragen.
JSON Format Response

```
["<str,software-id>", ...]
```

Beschreibung Die strongSwan Komponente sendet eine Liste von Software-IDs an die strongTNC Schnittstelle. Die Software-IDs wurden zuvor auf dem Client gemessen und widerspiegeln die momentan installierten Software Pakete.
Wenn bereits SWID Tags für alle übertragenen Software-IDs bestehen, können diese direkt eingetragen werden.

6.2.2 SWID Tags: Erstellung

URI Path `/swid/add-tags/`
Archetype Controller
Methods POST
Content-Type `application/json; charset=utf-8`
Request Parameter
`xmlData` SWID Tag als JSON-Liste.
Response Statuscodes
200 OK SWID Tags wurden erfolgreich verarbeitet.
400 Bad Request Fehlerhafter Request. Details zum Fehler werden im Response-Body zurückgesendet.
Beschreibung Die übermittelten Tags werden gelesen und in die Datenbank gespeichert. Bereits vorhandene Tags werden übersprungen.

6.3 CRUD Resourcen

6.3.1 Products

URI Path `/products/{id}/`
Archetype Document
Methods GET, PUT, PATCH
JSON Format Response

```
{
  "id": <int,id>,
  "uri": "<uri,resource>",
  "name": "<str,productname>"
}
```

URI Path `/products/`
Archetype Collection
Methods GET, POST
JSON Format Response

```
[
  {
    "id": <int,id>,
    "uri": "<uri,resource>",
    "name": "<str,productname>"
  }
]
```

URI Path `/products/{id}/default-groups/`
Archetype Collection
Methods GET, POST
Request Parameter

`groupId` group-id

JSON Format Response

```
[<doc,group>, ...]
```

6.3.2 Packages

URI Path /packages/{id}/
Archetype Document
Methods GET, PUT, PATCH
JSON Format Response

```
{
  "id": <int,id>,
  "uri": "<uri,resource>",
  "name": "<str,packagename>"
}
```

URI Path /packages/
Archetype Collection
Methods GET, POST
JSON Format Response

```
[
  {
    "id": <int,id>,
    "uri": "<uri,resource>",
    "name": "<str,packagename>",
  }
]
```

URI Path /packages/{id}/versions/
Archetype Collection
Methods GET, POST
JSON Format Response

```
[<doc,version>, ...]
```

6.3.3 Versions

URI Path `/versions/{id}/`

Archetype Document

Methods GET, PUT, PATCH, DELETE

JSON Format Response

```
{
  "id": <int,id>,
  "uri": "<uri,resource>",
  "package": "<uri,package>",
  "product": "<uri,product>",
  "release": "<str,release>",
  "securtiy": <int,security>,
  "blacklist": <bool,blacklist>,
  "time": <int,time>
}
```

URI Path `/versions/`

Archetype Collection

Filter Query

productName `<str,product-name>`

packageName `<str,package-name>`

Methods GET, POST

JSON Format Response

```
[
  {
    "id": <int,id>,
    "uri": "<uri,resource>",
    "package": "<uri,package>",
    "product": "<uri,product>",
    "release": "<str,release>",
    "securtiy": <int,security>,
    "blacklist": <bool,blacklist>,
    "time": <int,time>
  }
]
```

6.3.4 Identities

URI Path `/identities/{id}/`
Archetype Document
Methods GET, PUT, PATCH
JSON Format Response

```
{
  "id": <int,id>,
  "uri": "<uri,resource>",
  "type": <int,type>,
  "value": "<str,value>"
}
```

URI Path `/identities/`
Archetype Collection
Methods GET, POST
JSON Format Response

```
[
  {
    "id": <int,id>,
    "uri": "<uri,resource>",
    "type": <int,type>,
    "value": "<str,value>"
  }
]
```

6.3.5 Devices

URI Path `/devices/{id}/`
Archetype Document
Methods GET, PUT, PATCH
JSON Format Response

```
{
  "id": <int,id>,
  "uri": "<uri,resource>",
  "description": "<str,description>",
  "value": "<str,value>",
  "product": "<uri,product>",
  "created": <int,created>
}
```

URI Path `/devices/`
Archetype Collection
Methods GET, POST
JSON Format Response

```
[
  {
    "id": <int,id>,\\
    "uri": "<uri,resource>",
    "description": "<str,description>",
    "value": "<str,value>",
    "product": "<uri,product>",
    "created": <int,created>,
  }
]
```

URI Path `/device/{id}/sessions/`
Archetype Readonly Collection
Filter Query

- **timeFrom** `<int,timestamp>`
- **timeTo** `<int,timestamp>`

Methods GET
JSON Format Response

```
[<doc,session>, ...]
```

URI Path `/device/{id}/sessions/{id}/results/`
Archetype Readonly Collection
Methods GET
JSON Format Response

```
[<doc,result>, ...]
```

URI Path `/device/{id}/groups/`
Archetype Collection
Methods GET, POST
JSON Format Response

```
[<doc,group>, ...]
```

URI Path `/device/{id}/swid-tags/`
Archetype Readonly Collection
Methods POST
JSON Format Response

```
[<doc,swid-tag>, ...]
```

6.3.6 Enforcements

URI Path `/enforcements/{id}/`
Archetype Readonly Document
Methods GET
JSON Format Response

```
{
  "id": <int,id>,
  "uri": "<uri,resource>",
  "policy": "<uri,policy>",
  "group": "<uri,group>",
  "failRecommendation": <int,rec_fail>,
  "noresultRecommendation": <int,rec_noresult>",
  "maxAge": <int,max_age>
}
```

URI Path `/enforcements/`
Archetype Readonly Collection
Methods GET
Filter Query

- **groupName** `<str,group-name>`
- **policyName** `<str,policy-name>`

JSON Format

```
[
  {
    "id": <int,id>,
    "uri": "<uri,resource>",
    "policy": "<uri,policy>",
    "group": "<uri,group>",
    "failRecommendation": <int,rec_fail>,
    "noresultRecommendation": <int,rec_noresult>",
    "maxAge": <int,max_age>
  }
]
```

URI Path `/enforcements/{id}/groups/`
Archetype Readonly Collection
Methods GET
JSON Format

```
[<doc,group>, ...]
```

6.3.7 Policies

URI Path `/policies/{id}/`
Archetype Readonly Document
Methods GET
JSON Format Response

```
{
  "id": <int,id>,
  "uri": "<uri,resource>",
  "type": <int,type>,
  "name": "<str,name>",
  "argument": <policy-argument>,
  "failRecommendation": <int,rec_fail>,
  "noresultRecommendation": <int,rec_noresult>",
}
```

URI Path `/policies/`
Archetype Readonly Collection
Methods GET
JSON Format

```
[
  {
    "id": <int,id>,
    "uri": "<uri,resource>",
    "type": <int,type>,
    "name": "<str,name>",
    "argument": <policy-argument>,
    "failRecommendation": <int,rec_fail>,
    "noresultRecommendation": <int,rec_noresult>"
  }
]
```

URI Path `/policies/{id}/enforcements/`
Archetype Readonly Collection
Methods GET
JSON Format

```
[<doc,enforcement>, ...]
```

6.3.8 Groups

URI Path `/groups/{id}/`
Archetype Readonly Document
Methods GET
JSON Format Response

```
{
  "id": <int,id>,
  "uri": "<uri,resource>",
  "name": "<str,name>",
  "parent": "<uri,group>"
}
```

URI Path `/groups/`
Archetype Readonly Collection
Methods GET
JSON Format

```
[
  {
    "id": <int,id>,
    "uri": "<uri,resource>",
    "name": "<str,name>",
    "parent": "<uri,group>"
  }
]
```

URI Path `/groups/{id}/devices/`
Archetype Collection
Methods GET, POST
JSON Format Response

```
[<doc,device>, ...]
```

URI Path `/groups/{id}/enforcements/`
Archetype Readonly Collection
Methods GET
JSON Format Response

```
[<doc,enfocement>, ...]
```

6.3.9 Files

URI Path `/files/{id}/`
Archetype Document
Methods GET, PUT, PATCH
JSON Format Response

```
{
  "id": <int,id>,
  "uri": "<uri,resource>",
  "name": "<str,name>",
  "hashes: <doc,hash-set>,
  "dir": "<uri,directory>"
}
```

URI Path `/files/`
Archetype Collection
Methods GET, POST
JSON Format Response

```
[
  {
    "id": <int,id>,
    "uri": "<uri,resource>",
    "name": "<str,name>"
    "hashes: <doc,hash-set>,
    "dir": "<uri,directory>"
  }
]
```

6.3.10 Directories

URI Path `/directories/{id}/`
Archetype Document
Methods GET, PUT, PATCH
JSON Format Response

```
{
  "id": <int,id>,
  "uri": "<uri,resource>",
  "path": "<str,path>"
}
```

URI Path `/directories/`
Archetype Collection
Methods GET, POST
JSON Format Response

```
[
  {
    "id": <int,id>,
    "uri": "<uri,resource>",
    "path": "<str,path>"
  }
]
```

URI Path `/directories/{id}/files/`
Archetype Collection
Methods GET, POST
JSON Format Response

```
[<doc,file>, ...]
```

6.3.11 SWID Tags

URI Path `/swid-tags/`
Archetype Readonly Collection
Methods GET
JSON Format Response

```
[<doc,swid-tag>, ...]
```

URI Path `/swid-tags/{id}/`
Archetype Readonly Document
Methods GET
Filter Query

- **packageName** `<str,package-name>`
- **version** `<str,version>`
- **uniqueId** `<str,unique-id>`

JSON Format Response

```
{
  "id": <int,id>,
  "uri": "<uri,resource>",
  "packageName": "<str,package-name>",
  "version": "<str,version>",
  "uniqueId": "<str,unique-id>",
  "entities": [
    {
    "entity": "<uri,entity>",
    "role":  <int,role>
    }
  ],
  "tagXml": "<xml,swid-tag>"
}
```

URI Path `/swid-tags/{id}/files/`
Archetype Readonly Collection
Methods GET
JSON Format Response

```
[<doc,file>, ...]
```

6.3.12 Entities

URI Path `/swid-entities/{id}/`
Archetype Readonly Document
Methods GET
JSON Format Response

```
{
  "id": <int,id>,
  "uri": "<uri,resource>",
  "name": "<str,name>",
  "regid": "<str,regid>"
}
```

URI Path `/swid-entities/`
Archetype Readonly Collection
Methods GET
JSON Format Response

```
[
  {
    "id": <int,id>,
    "uri": "<uri,resource>",
    "name": "<str,name>",
    "regid": "<str,regid>",
  }
]
```

URI Path `/swid-entities/{id}/swid-tags/`
Archetype Readonly Collection
Methods GET
JSON Format Response

```
[<doc,swid-tag>, ...]
```

References

[1] Masse, Mark. *REST API design rulebook.* O'Reilly Media, Inc., 2011.

[2] Berners-Lee, Tim, Roy Fielding, and Larry Masinter. *RFC 3986: Uniform resource identifier (uri): Generic syntax.* The Internet Society (2005).