

REST Guidelines

OSS product creation - OSS Integrations – NE3S

Exported on 10/20/2020

Table of Contents

1	Background	6
2	Scope	8
3	Encoding.....	9
4	Authentication	10
5	Authorization.....	11
5.1	Using 404	11
5.2	Using 403	11
6	Media types	13
6.1	JSON media-types	13
6.1.1	Good examples.....	14
6.1.2	Bad examples	14
7	HATEOAS	15
7.1	JSON based APIs	15
7.2	XML based APIs.....	16
8	Versioning.....	18
8.1	URL encoded versioning.....	19
8.2	Media type based versioning.....	19
8.3	Supported versions.....	19
8.4	Not supported versions	20
9	URLs	21
9.1	Anatomy	21
9.2	General	22
9.2.1	Good URL examples.....	22
9.2.2	Bad URL examples	22
9.3	Queries on collections	23
9.3.1	Lengthy queries.....	23
9.3.1.1	Modeling lengthy queries as POST requests	23
9.3.1.2	Modeling lengthy queries as REST resources	24
9.4	Entry point.....	25

9.5	API building block	26
10	HTTP Verbs	27
10.1	Using PATCH.....	27
11	HTTP Headers.....	29
11.1	HTTP Headers from server.....	29
11.2	HTTP Headers from client	31
12	Responses.....	33
12.1	Headers.....	33
12.2	Error handling	34
12.3	Error codes	34
13	CORS	36
13.1	Simple requests	36
13.2	Preflight requests.....	36
14	Asynchronous operations.....	39
14.1	Asynchronous REST verbs	39
14.1.1	POST example	40
14.1.2	DELETE example	41
14.2	Jobs and workflows	41
14.3	Asynchronous REST verbs and jobs	42
15	Concurrency control	43
16	Pagination	45
17	Rate limiting	47
18	Non-REST interactions.....	48
19	REST API Documentation	49

Table of Contents

- [Background](#)(see page 6)
- [Scope](#)(see page 8)
- [Encoding](#)(see page 9)
- [Authentication](#)(see page 10)
- [Authorization](#)(see page 11)
 - [Using 404](#)(see page 11)
 - [Using 403](#)(see page 11)
- [Media types](#)(see page 13)
 - [JSON media-types](#)(see page 13)
 - [Good examples](#)(see page 14)
 - [Bad examples](#)(see page 14)
- [HATEOAS](#)(see page 15)
 - [JSON based APIs](#)(see page 15)
 - [XML based APIs](#)(see page 16)
- [Versioning](#)(see page 18)
 - [URL encoded versioning](#)(see page 19)
 - [Media type based versioning](#)(see page 19)
 - [Supported versions](#)(see page 19)
 - [Not supported versions](#)(see page 20)
- [URLs](#)(see page 21)
 - [Anatomy](#)(see page 21)
 - [General](#)(see page 22)
 - [Good URL examples](#)(see page 22)
 - [Bad URL examples](#)(see page 22)
 - [Queries on collections](#)(see page 23)
 - [Lengthy queries](#)(see page 23)
 - [Modeling lengthy queries as POST requests](#)(see page 23)
 - [Modeling lengthy queries as REST resources](#)(see page 24)
 - [Entry point](#)(see page 25)
 - [API building block](#)(see page 26)
- [HTTP Verbs](#)(see page 27)
 - [Using PATCH](#)(see page 27)
- [HTTP Headers](#)(see page 29)
 - [HTTP Headers from server](#)(see page 29)
 - [HTTP Headers from client](#)(see page 31)
- [Responses](#)(see page 33)
 - [Headers](#)(see page 33)
 - [Error handling](#)(see page 34)
 - [Error codes](#)(see page 34)
- [CORS](#)(see page 36)
 - [Simple requests](#)(see page 36)
 - [Preflight requests](#)(see page 36)
- [Asynchronous operations](#)(see page 39)
 - [Asynchronous REST verbs](#)(see page 39)
 - [POST example](#)(see page 40)
 - [DELETE example](#)(see page 41)
 - [Jobs and workflows](#)(see page 41)
 - [Asynchronous REST verbs and jobs](#)(see page 42)
- [Concurrency control](#)(see page 43)

- [Pagination](#)(see page 45)
- [Rate limiting](#)(see page 47)
- [Non-REST interactions](#)(see page 48)
- [REST API Documentation](#)(see page 49)

1 Background

Following resources have been used to compile these guidelines:

- <https://github.com/WhiteHouse/api-standards>
- [Short Guide On Creating RESTful Services¹](#) (OoD)
- [Facebook Graph API²](#)
- [Github API³](#)
- [EPC RESTful API design⁴](#)
- [Designing HTTP Interfaces and RESTful Web Services⁵](#)
- API Facade Pattern, by Brian Mulloy, Apigee
- Web API Design, by Brian Mulloy, Apigee
- [Fieldings Dissertation on REST⁶](#)
- [IANA link relations⁷](#)
- [HTTP status codes cheat sheet⁸](#)
- [HAL specification⁹](#)
- Key words for use in RFCs to Indicate Requirement Levels [RFC 2119¹⁰](#)
- [Please. Don't Patch Like An Idiot.¹¹](#)
- [Heroku Platform API design guide¹²](#)
- [OpenNMS ReST¹³](#)
- [Searches as resources article¹⁴](#)
- CORS
 - [W3 CORS¹⁵](#)
 - [Mozilla¹⁶](#)
- OAuth 2.0:
 - [Specification home page¹⁷](#): includes a list of server and client libraries
 - [Facebook and OAuth 2.0¹⁸](#): a blog post on how Facebook uses OAuth2.0, easy to read and perhaps best article I have found to understand OAuth2.0 in 5 minutes
- Hypermedia formats:
 - [JSON LD¹⁹](#)
 - [On choosing hypermedia format²⁰](#)
- Versioning:
 - <http://www.lexicalscope.com/blog/2012/03/12/how-are-rest-apis-versioned/>
 - https://www.mnot.net/blog/2011/10/25/web_api_versioning_smackdown

1 <https://confluence.ext.net.nokia.com/display/OoD/Short+Guide+On+Creating+RESTful+Services>

2 <https://developers.facebook.com/docs/graph-api>

3 <https://developer.github.com/v3/>

4 <https://sharenet-ims.inside.nokiasiemensnetworks.com/Open/D512299607>

5 <http://munich2012.drupal.org/program/sessions/designing-http-interfaces-and-restful-web-services>

6 <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>

7 <http://www.iana.org/assignments/link-relations/link-relations.xhtml>

8 <http://www.restapitutorial.com/httpstatuscodes.html>

9 http://stateless.co/hal_specification.html

10 <https://www.ietf.org/rfc/rfc2119.txt>

11 <http://williamdurand.fr/2014/02/14/please-do-not-patch-like-an-idiot/>

12 <https://github.com/interagent/http-api-design>

13 <http://www.opennms.org/wiki/ReST>

14 <http://evertpot.com/dropbox-post-api/>

15 <http://www.w3.org/TR/cors/>

16 https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS

17 <http://oauth.net/2/>

18 <http://web.archive.org/web/20150905174403/http://www.socialipstick.com/?p=239>

19 <http://json-ld.org/>

20 <http://sookocheff.com/post/api/on-choosing-a-hypermedia-format/>

- <https://www.mnot.net/blog/2012/12/04/api-evolution>
- ETSI
 - http://www.etsi.org/deliver/etsi_gs/MEC/001_099/009/01.01.01_60/gs_MEC009v010101p.pdf (ETSI GS MEC009: General principles for Mobile Edge Service APIs)
 - https://nfvwiki.etsi.org/images/NFVSOL%2817%29000050r4_ETSI_NFV_SOL_REST_API_Conventions.pdf (ETSI NFV API guidelines)

2 Scope

The scope of these guidelines is Nokia Networks REST APIs, in other words, any REST API developed or provided by Nokia Networks shall follow these guidelines.

These guidelines are company confidential.

3 Encoding

- All XML and JSON data transferred via the APIs is expected to be UTF-8 encoded
 - UTF-16 encoded is intentionally not supported, the reason is that there is no known application in Nokia Networks that uses UTF-16: adding UTF_16 support would be artificial and would only increase complexity at no gain (also [HTML5 discourages](#)²¹ using UTF-16 and UTF-32)
- Date/time fields used in the API shall use [ISO 8601 formatted string](#)²², e.g.: 2014-02-27T15:05:06+01:00
 - [RFC3339](#)²³ shall be used as ISO 8601 profile for data and time expressions
 - These guidelines do not define a specific formatted string to be used, the specific formatted string is use case dependent (e.g.: addition of fraction of seconds, timezone offsets, etc...)
 - Dot '.' shall be used as fraction of seconds separator
 - Whenever time of day is presented then timezone offset shall be present
 - Notice that the formatted string can represent time, duration, time interval or repeating intervals

²¹ <http://www.w3.org/International/questions/qa-choosing-encodings>

²² <http://www.cl.cam.ac.uk/~mgk25/iso-time.html>

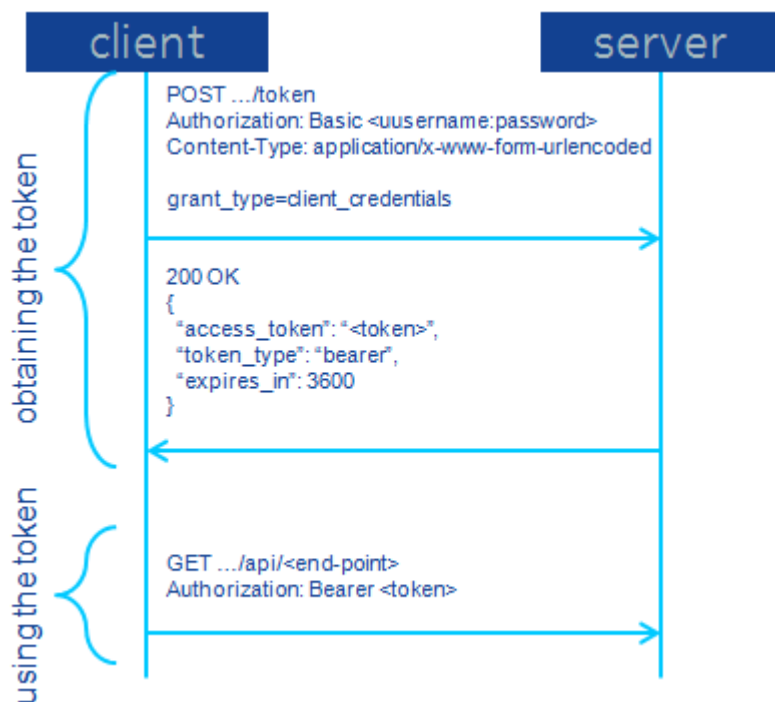
²³ <http://www.ietf.org/rfc/rfc3339.txt>

4 Authentication

- Whenever possible end user credentials (and not system credentials) shall be used over HTTPS

i Even though end user credentials should be used, there are several practical limitations that force systems to instead exchange system credentials, in these cases authentication and authorization are system based and therefore are more limited.

- API shall use OAuth2.0
- OAuth2.0 [client credentials grant](#)²⁴ shall be used
- OAuth2.0 [bearer tokens](#)²⁵ shall be used
- Scope may not be supported
- Tokens shall be transferred via [header](#)²⁶



- Each API provider (resource server) shall be an authorization server, unless a centralized OAuth2.0 authorization server is provided in the deployment.

²⁴ <http://tools.ietf.org/html/rfc6749#page-40>

²⁵ <http://tools.ietf.org/html/rfc6750>

²⁶ <http://tools.ietf.org/html/rfc6750#page-5>

5 Authorization

- Credentials provided during authentication shall be used for authorization purposes
- Server should avoid leakage of information to non-authorized users: this includes providing 401 / 403 status codes for non authorized users (as they leak information that the resource exists in the first instance), in these cases 404 is recommended

5.1 Using 404

Request

```
GET /users/mike2 HTTP/1.1
Host: acme.com
Accept: application/vnd.nokia-users+json
Authorization: Basic Zm9vOmJhcg==
```

Response (failed authorization, signaled via 404)

```
HTTP 404
Content-Type: application/json

{
  "type": "https://example.com/probs/failed-auhtorization",
  "title": "User is not authorized to perform operation",
  "detail": "The user that attempted to perform the operation is not authorized",
  "instance": "/users/mike2"
}
```

5.2 Using 403

See: <https://tools.ietf.org/html/rfc7231#section-6.5.3>

Request

```
PUT /users/mike HTTP/1.1
Host: acme.com
Accept: application/vnd.nokia-users+json
Authorization: Basic Zm9vOmJhcg==
```

Response (failed authorization, signaled via 403: not recommended as it leaks information)

HTTP 403

Content-Type: application/json

```
{
  "type": "https://example.com/probs/failed-auhtorization",
  "title": "User is not authorized to perform operation",
  "detail": "The user that attempted to perform the operation is not authorized",
  "instance": "/users/mike2"
}
```

6 Media types

- Clients shall use media types in Accept header to request for specific payload contents
- Servers shall provide media types in Content-Type header
- All payloads shall have own media type, anatomy of a media type:
 - application/vnd.nokia-[payload]-v[version]+[xml/json]
- Payload forwarding entities are not enforced to parse payload contents in order to derive accurate media type version information:
 - A forwarding entity shall provide media type value as agreed in the interface (e.g.: application/vnd-nokia-test-v1+json)
 - If the actual payload version is different but backwards compatible to the one provided by the forwarding entity in Content-Type then this shall not represent an error
- Clients may use application/json and application/xml as shortcuts to the latest version of the expected payload
 - This is useful when navigating a REST API via browser
- The default format to be used as payload is JSON (unless JSON is not one of the payloads used by the API)
 - This means that if a client can accept the latest version of the expected payload in JSON format then it is not necessary to provide Accept header

6.1 JSON media-types

- JSON payload shall not include values in keys
- If a property requires quotes, double quotes must be used. All property names must be surrounded by double quotes. Property values of type string must be surrounded by double quotes. Other value types (like boolean or number) should not be surrounded by double quotes
- Comments should not be included in JSON objects
- Property names must conform to the following guidelines:
 - Property names should be meaningful names with defined semantics.
 - Property names must be camel-cased, ascii strings.
 - The first character must be a letter, an underscore (_) or a dollar sign (\$).
 - Subsequent characters can be a letter, a digit, an underscore, or a dollar sign.
 - Reserved JavaScript keywords should be avoided:


```
abstract
boolean break byte
case catch char class const continue
debugger default delete do double
else enum export extends
false final finally float for function
goto
if implements import in instanceof int interface
let long
native new null
package private protected public
return
short static super switch synchronized
this throw throws transient true try typeof
var volatile void
while with
yield
```
- Array types should have plural property names. All other property names should be singular

- The spec at [JSON.org](http://www.json.org/)²⁷ specifies exactly what type of data is allowed in a property value. This includes Unicode booleans, numbers, strings, objects, arrays, and null
 - Dates should be formatted as recommended by RFC 3339
 - "lastUpdate": "2007-11-06T16:34:41.000Z"
 - Time durations should be formatted as recommended by ISO 8601.
 - "duration": "P3Y6M4DT12H30M5S"
 - Latitudes/Longitudes should be formatted as recommended by ISO 6709. Furthermore, they should favor the $\pm DD.DDDD\pm DDD.DDDD$ degrees format.
 - "statueOfLiberty": "+40.6894-074.0447"
- Properties can be in any order within the JSON object.

6.1.1 Good examples

No values in keys:

```
"tags": [
  {"id": "125", "name": "Environment"},
  {"id": "834", "name": "Water Quality"}
],
```

6.1.2 Bad examples

Values in keys:

```
"tags": [
  {"125": "Environment"},
  {"834": "Water Quality"}
],
```

²⁷ <http://www.json.org/>

7 HATEOAS

- APIs shall use [HATEOAS](#),²⁸ (See [About HATEOAS](#)²⁹ for more information.)
- References shall be relative whenever possible
- References shall be provided as part of the HTTP response body (instead of Link headers)
- References shall be used (at least) with:
 - 202 responses
 - Pagination
 - Entry points
 - Collections
- API clients shall use HATEOAS in order to discover resource end-point, in other words API clients need only to hard-code the end-point `https://<subdomain.domain.top-level-domain>/<path>/api/<api-building-block>` the rest can be discovered via resource links. This allows API servers to change end-points without impact into clients (notice that changing end-points for resources that can be discovered is considered a backwards compatible change)

7.1 JSON based APIs

- APIs shall follow [HAL](#)³⁰ when requiring links to other resources
- APIs shall use HAL when listing resources under a collection (notice that collections can be represented as HAL links or as HAL embedded resources)

Request

```
GET /users HTTP/1.1
Host: acme.com
Accept: application/vnd.nokia-users+json
```

²⁸ <http://en.wikipedia.org/wiki/HATEOAS>

²⁹ <https://confluence.ext.net.nokia.com/display/NE3S/About+HATEOAS>

³⁰ http://stateless.co/hal_specification.html

Response

```
{
  "_links": {
    "self": {
      "href": "/users"
    },
    "user": [
      {
        "href": "/users/mike",
        "title": "Mike Kelly",
        "type": "application/vnd.nokia-user+json"
      },
      {
        "href": "/users/mture",
        "title": "Mark Ture",
        "type": "application/vnd.nokia-user+json"
      }
    ]
  }
}
```

- APIs may provide links to self, but it is not required
- APIs may provide documentation links, but it is not required
 - When providing documentation links it is recommended to provide them using HAL curies
- APIs should provide link media type (via [optional type property](#)³¹) for better discoverability

7.2 XML based APIs

- APIs shall use Atom links when requiring links to other resources
- APIs shall use Atom links when listing resources under a collection

Request

```
GET /users HTTP/1.1
Host: acme.com
Accept: application/vnd.nokia-users+xml
```

³¹ <http://tools.ietf.org/html/draft-kelly-json-hal-05#section-5.3>

Response

```
<?xml version="1.0" encoding="utf-8"?>
<users xmlns="urn:vnd.nokia-users" xmlns:atom="http://www.w3.org/2005/Atom">
  <user>
    <name>Mike Kelly</name>
    <atom:link rel="item" type="application/vnd.nokia-user+xml" href="/users/mike"/>
  </user>
  <user>
    <name>Mark Ture</name>
    <atom:link rel="item" type="application/vnd.nokia-user+xml" href="/users/mture"/>
  </user>
</users>
```

- Link rel attribute shall follow [IANA pre-defined link relations](http://www.iana.org/assignments/link-relations/link-relations.xhtml)³² whenever possible

³² <http://www.iana.org/assignments/link-relations/link-relations.xhtml>

8 Versioning

- All APIs shall be versioned
- Versions shall be prefixed with 'v', e.g.: v1, v2, v3
- Versions shall be integers (no usage of semver)
- All APIs should provide at least one major version backwards compatibility
- Major version shall be incremented whenever a non-discoverable change is introduced
- All APIs shall use both URL encoded versioning and media type based versioning

About non-discoverable changes

What is a non-discoverable change? non-discoverable changes are those that a client cannot discover by using HATEOAS principles, e.g.:

- Minor resource modeling refactoring: API has changed resource model, e.g.: an API v1 provides *"alarms"* as resources and on v2 the API provides *"events"* as resources (where *"alarm"* is a sub-type of *"event"*). In this situation clients will try to find *"alarm"* resource but it will not be found
 - Notice that this API change can be made discoverable by clients if the API provides duplicate resources for *"alarms"* (represented as *"alarm"* and as *"event"*)
- Major resource modeling refactoring: API has changed resource model, e.g.: an API v1 provides *"alarms"* as resources and on v2 the API provides *"managedObjects"* as resources (where *"alarms"* is a property of *"managedObject"*). In this situation clients will try to find *"alarm"* resource but it will not be found
 - Notice that the only way to make this API change discoverable is to provide both resource model views in parallel, which is basically the same as to provide both API versions under different URLs

Why semver is not supported?


There are few reasons why semver is not supported:

- REST API version should not describe software version: the REST API version describes the evolution of the contract between consumer and producer, not the evolution of the internal components of the producer. **Don't couple software version to REST API version**
- REST API version evolution should be limited to non-backwards compatible changes, by definition semver provides minor and patch versions which provide insights into backwards compatible changes, those are of no interest to REST API clients. **Don't pollute REST API with irrelevant information**
- Ideally resources should be uniquely identified by their URLs, using semver leads to a situation where resources end up being identified by a multitude of URLs, e.g.: resource *"alarm with id 101"* is identified by */fm/v1.0.0/alarms/101*, */fm/v1.0.1/alarms/101*, */fm/v1.0.2/alarms/101*, */fm/v1.1.0/alarms/101*, */fm/v1.1.1/alarms/101*... This leads to a situation where many web mechanisms become invalid or broken (caches, spiders, forms...)
- REST APIs should be designed so that visible version changes (non-backwards compatible changes) are extremely rare (large internet providers stay on same REST API version for several years), on the other hand semver introduces a culture of *"version changes are welcomed"*, where every single software build introduces a new version

8.1 URL encoded versioning

The version of the API is part of the URL, e.g.:

- <https://api.myservice.com/api/service1/v1/foo/bar>
- <https://api.myservice.com/api/service2/v2/foo/bar>³³

 URL encoded version is used to track the whole resource model structure changes

8.2 Media type based versioning

The version of the API is requested by client as part of the Accept media type headers, e.g.:

- v1

```
GET /products HTTP/1.1
Host: acme.com
Accept: application/vnd.nokia-myservice-v1+xml
```

- v2

```
GET /products HTTP/1.1
Host: acme.com
Accept: application/vnd.nokia-myservice-v2+xml
```

 Media type version is used to track specific resource content changes

8.3 Supported versions

APIs shall provide the list of supported versions

Request
<pre>GET /(api-building-block) Host: (subdomain.domain.top-level-domain) Accept: application/vnd.nokia-versioning+json</pre>

³³ <https://api.myservice.com/service2/v2/foo/bar>

Response

```
{
  "_links": {
    "versions": [
      {
        "href": "/v1"
      },
      {
        "href": "/v2"
      }
    ]
  }
}
```

8.4 Not supported versions

- Major versions that are not anymore supported shall return 404 code (NOT FOUND)

9 URLs

9.1 Anatomy

									Purpose
https	<subdomain.domain.top-level-domain>	[<path>]	/api	/<api-building-block>					List all available versions
					/v<version-number>				Entry point
						/<resource-name-in-plural>			List all resources
						/<resource-id>			Single resource
						?<filter-criteria>=<filter-value>			Filtering resources by query
						/<resource-id>	/<resource-name-in-plural>		Hierarchy of resources
						/<resource-id>?fields=<a,b,c>			Filtering resource attributes

The above URL anatomy assumes following:

- HTTPS shall be always used
- *path* shall be empty unless imposed by the hosting server
- *path* shall not contain the product or API name, or other similar identifiers that come from the interface content
- *api-building-block* represents a major and distinguishable building block of the API
- *version-number* represents the version in use of the API, notice that API building blocks are versioned independently

9.2 General

- URL shall identify a resource
- URLs shall include nouns
- URLs shall not include verbs
 - Use HTTP verbs (GET, POST, PUT, DELETE) to operate on the collections and elements.
- Plural shall be used with nouns
 - URLs shall avoid using collection verbage (e.g.: customerList)
 - In case the noun represents a singleton then singular can be used
- URL segments shall use lower-case and hyphens ('-') to separate words
- URLs should not go deeper than resource/identifier/resource
- Optional fields shall be provided as a comma separated list.
- URLs shall not include media type extensions to indicate resource media type (e.g.: http://www.example.gov/api/v1/magazines.json), instead accept header shall be used:

Request

```
GET /api/cms/v1/magazines
Host: www.example.gov
Accept: application/vnd.nokia-raml-v21+json
```

9.2.1 Good URL examples

- List of magazines:
 - http://www.example.gov/api/cms/v1/magazines
- Filtering is a query:
 - http://www.example.gov/api/cms/v1/magazines?year=2011&sort=desc
 - http://www.example.gov/api/cms/v1/magazines?topic=economy&year=2011
- A single magazine:
 - http://www.example.gov/api/cms/v1/magazines/1234
- All articles in (or belonging to) this magazine:
 - http://www.example.gov/api/cms/v1/magazines/1234/articles
- Specify optional fields in a comma separated list:
 - http://www.example.gov/api/cms/v1/magazines/1234?fields=title,subtitle,date
- Add a new article to a particular magazine:
 - POST http://example.gov/api/cms/v1/magazines/1234/articles

9.2.2 Bad URL examples

- Non-plural noun:
 - http://www.example.gov/api/cms/v1/magazine
 - http://www.example.gov/api/cms/v1/magazine/1234
 - http://www.example.gov/api/cms/v1/publisher/magazine/1234

- Verb in URL:
 - <http://www.example.gov/api/cms/v1/magazines/1234/create>
- Filter outside of query string
 - <http://www.example.gov/api/cms/v1/magazines/2011/desc>

9.3 Queries on collections

- APIs may provide possibility to query collections
- The default query mechanism is to provide filtering by resource attributes in the URL:
 - <http://www.example.gov/api/cms/v1/magazines?topic=economy&year=2011>
- APIs may provide sort attribute in collection requests (where the values are asc or desc)
 - <http://www.example.gov/api/cms/v1/magazines?year=2011&sort=desc>
- Responses to queries on collections should take into account pagination

9.3.1 Lengthy queries

There is a [limitation](#)³⁴ on how large URLs can be, this limitation varies depending on browser and server but 2000 characters is considered to be the limit for URL length.

This imposes as well a limitation on how many attributes can fit into URL attributes.

Two different approaches are provided in order to solve this limitation:

- Modeling lengthy queries as POST requests
- Modeling lengthy queries as REST resources

The first approach is not fully REST but it is a good compromise when simplicity is required (avoids state management of query criteria) and when it is not expected that query criteria will be re-used across multiple calls / clients.

The second approach complies with REST but it increases system complexity but requiring state management / persistency of query criteria. This solution is optimal when query criteria are re-used across multiple calls / clients.

9.3.1.1 Modeling lengthy queries as POST requests

- In this approach the query shall be provided as a JSON object on a POST request
- The POST request shall include X-HTTP-Method-Override: GET in headers
- The response shall take into account pagination
 - In case of pagination a queryId attribute shall be provided
 - The queryId is only valid for a limited period of time, i.e.: clients shall not make assumptions about durability of the queryId

³⁴ <http://stackoverflow.com/questions/417142/what-is-the-maximum-length-of-a-url-in-different-browsers>

Request

```
POST /users
Host: (subdomain.domain.top-level-domain)
Content-Type: application/vnd.nokia-user+json
Accept: application/vnd.nokia-users+json
X-HTTP-Method-Override: GET
```

```
{
  "isActive": true,
  "isAdmin": false,
  "organization": "NOKIA",
  [...]
}
```

Response

```
HTTP 200
Content-Type: application/vnd.nokia-users+json
```

```
{
  "_links": {
    "next": { "href": "/users?after=MTAxNTExOTQ1MjAwNzI5NDE&limit=30&queryId=hty4658Jrtrh" },
    "user": [
      {
        "href": "/users/mike",
        "title": "Mike Kelly"
      },
      [...]
    ]
  }
}
```

9.3.1.2 Modeling lengthy queries as REST resources

- In this approach queries are modeled as resources

Request

```
POST /queries
Host: (subdomain.domain.top-level-domain)
Content-Type: application/vnd.nokia-query+json
Accept: application/vnd.nokia-query+json
```

```
{
  "isActive": true,
  "isAdmin": false,
  "organization": "NOKIA",
  [...]
}
```

Response

HTTP 201

```
{
  "id": 1,
  "_links": {
    "self": {
      "href": "/queries/1"
    }
  }
}
```

9.4 Entry point

- APIs shall have entry point


Request

```
GET /(api-building-block)/v1
Host: (subdomain.domain.top-level-domain)
Accept: application/com.nokia.x+json
```

Response

```
{
  "_links": {
    "magazines": { "href": "/magazines" },
    "dogs": { "href": "/dogs" }
  }
}
```

9.5 API building block

 The term API building block is quite lame, please come up with something better (my preference is to avoid using the term "fragment" as it is quite loaded term already in NE3S WS)

Represents a major and distinguishable building block of Nokia Networks REST API

Building blocks present following characteristics:

- Can be combined with other API building blocks to satisfy a high level use case, i.e.:
 - A single API building block rarely satisfies a high level use case by its own
 - A single API building block typically serves multiple use cases
- Is responsible for own set of resources where:
 - The API building block is handling resources that no other API building block is handling, or
 - The API building block is handling a specific concern of the resources that no other API building block is handling

10 HTTP Verbs

Verb	Description
HEAD	Can be issued against any resource to get just the HTTP header info.
GET	Used for retrieving resources.
POST	Used for creating resources
PATCH	Used for updating resources with partial data
PUT	Used for replacing resources or collections. For PUT requests with no body attribute, be sure to set the Content-Length header to zero.
DELETE	Used for deleting resources.
OPTIONS	Used to discover how client can interact with the resource.

- GET, HEAD and OPTIONS HTTP methods are defined as [safe](#)³⁵ and [idempotent](#)³⁶: they are only intended for retrieving data.
- PUT, PATCH and DELETE methods are defined to be [idempotent](#)³⁷
- POST method is not idempotent

In some cases there is a need to perform an update over a resource that is not idempotent, e.g.: increment a counter on a resource, in these cases it is best to use POST instead of PUT.

10.1 Using PATCH

- When APIs use PATCH they shall satisfy its special semantics (see [RFC 5789](#)³⁸):
 - PATCH body shall contain the set of changes that should be performed
 - When using JSON media types APIs shall use [RFC 7396](#)³⁹ to describe the set of changes
 - When using XML media types APIs shall use [RFC 5261](#)⁴⁰ to describe the set of changes

³⁵ <http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html>

³⁶ <http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html>

³⁷ <http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html>

³⁸ <https://tools.ietf.org/html/rfc5789>

³⁹ <https://tools.ietf.org/html/rfc7396>

⁴⁰ <http://tools.ietf.org/html/rfc5261>

```

PATCH /users/123 HTTP/1.1
Host: example.org
Content-Type: application/merge-patch+json

{
  "email": "new.email@example.org",
  "another-field": {
    "a-field-we-want-to-remove-in-this-patch-operation": null
  },
  "a-field-to-add-in-this-patch-operation": 123
}

```

⚠ Notice that [RFC 7396](#)⁴¹ performs poorly on arrays, as there is a need to provide the whole array even if only one element needs to be modified.

ⓘ Before [RFC 7396](#)⁴² was available these guidelines were recommending the usage of [RFC 6902](#)⁴³ and [RFC 6901](#)⁴⁴ for JSON media types

```

PATCH /users/123

[
  { "op": "replace", "path": "/email", "new.email@example.org" }
]

```

⁴¹ <https://tools.ietf.org/html/rfc7396>

⁴² <https://tools.ietf.org/html/rfc7396>

⁴³ <http://tools.ietf.org/html/rfc6902>

⁴⁴ <http://tools.ietf.org/html/rfc6901>

11 HTTP Headers

Notice that not all the HTTP headers are listed here, only those that are considered of special relevance.

11.1 HTTP Headers from server

header	when to use	client support
Content-Type	Always, indicates the media type of the payload provided	mandatory
Date	The server provides the data and time that the message was sent (in HTTP date format as defined by RFC 7231 ⁴⁵)	optional
Expires	If the server wants to provide a hint to the client on how long it will take for the resources to become stale	optional
ETag	If the server wants to provide a hash of the resource contents for later caching negotiation or if concurrency control is desired	optional
Content-Encoding	If the client requested a compressed version of the resource via "Accept-Encoding" header Notice that in case Content-Encoding is used it shall always have "gzip" as value	optional
Content-Length	The server provides the length of the response body in octets	mandatory (except when using chunked transfer encoding)
Transfer-Encoding	If the server is providing the data in a series of "chunks" (the value of this header is "chunked"). In this case the Content-Length is not used. Typically used when gzip encoding is used.	mandatory (except when using non-chunked transfer encoding)
Allow	If the server wants to provide information about what operations are valid for a resource. When this header is provided then the contents shall take into account the current authorization context	optional

⁴⁵ <http://tools.ietf.org/html/rfc7231>

header	when to use	client support
X-Request-Id	If the server wants to provide an UUID that uniquely identifies this request, this helps during troubleshooting / debugging if both the server and client log the request identifier	optional
X-RateLimit-Limit	The maximum number of requests that the consumer is permitted to make per time window.	optional (used with rate limiting)
X-RateLimit-Remaining	The number of requests remaining in the current rate limit window.	optional (used with rate limiting)
X-RateLimit-Reset	The time at which the current rate limit window resets in UTC epoch seconds ⁴⁶ .	optional (used with rate limiting)
Access-Control-Allow-Origin	Used during CORS interactions, if the server has got a request with "Origin" header. If the value of this header is wildcard ("*") then "Access-Control-Allow-Credentials" should not be false If the value of this header is an origin host (not "*") then the server must include Origin in the Vary response header	optional (used with CORS)
Access-Control-Expose-Headers	Used during CORS interactions, if the server wants to white-list headers that clients are allowed to access	optional (used with CORS)
Access-Control-Max-Age	Used during CORS interactions (preflight), if the server wants to indicate how long results of a preflight request can be cached	optional (used with CORS)
Access-Control-Allow-Credentials	Used during CORS interactions, if the server wants to indicate whether clients can perform CORS requests with credentials (Cookie)	optional (used with CORS)
Access-Control-Allow-Methods	Used during CORS interactions (preflight), if the server wants to indicate the method or methods allowed when accessing the resource	optional (used with CORS)

⁴⁶ http://en.wikipedia.org/wiki/Unix_time

header	when to use	client support
Access-Control-Allow-Headers	Used during CORS interactions (preflight), if the server wants to indicate the HTTP headers that can be used when performing the actual request	optional (used with CORS)

11.2 HTTP Headers from client

header	when to use	server support
Last-Modified	<p>The client wants to get a full response from the server if the resource has been modified since the provided date</p> <p>Notice that only provides second granularity and requires certain level of client/server time synchronization</p>	optional
If-Match	<p>The client wants to get a full response from the server if the resource matches the etag hash provided</p> <p>Notice that this can also be used in order to detect concurrent updates when performing PUT: optimistic concurrency support</p>	optional
If-None-Match	<p>The client wants to get a full response from the server if the resource does not match the etag hash provided</p> <p>Notice that this can also be used in order to detect concurrent updates when performing PUT: optimistic concurrency support</p>	optional
Accept	The client specifies the content types that are acceptable for the response	mandatory except when client is able to understand default content type provided by that end-point (which is the latest version of the media type)

header	when to use	server support
Accept-Encoding	<p>The client wants to obtain a compressed version of the resource in order to save network capacity.</p> <p>Notice that in case server supports compression then gzip shall be used</p> <p>Notice that in case server does not support compression, or does not support provided compression schema (i.e.: it is not gzip) then the server will respond with HTTP 406 (NOT ACCEPTABLE)</p>	optional
Expect	<p>The client wants to enforce asynchronous or synchronous operations:</p> <ul style="list-style-type: none"> ▪ Asynchronous: "Expect: 202-accepted" ▪ Synchronous: "Expect: 200-ok/201-created/204-no-content" 	<p>mandatory</p> <p>(server will respond with "417 Expectation failed" if it cannot satisfy the expectation)</p>
X-HTTP-Method-Override	The client is using a collection query that does not fit into URL attributes and instead it is using a POST that should be translated into GET	mandatory when supporting queries that do not fit into URL attributes
Origin	Used during CORS interactions, indicates the origin of the cross-site access request or preflight request	optional (used with CORS)
Access-Control-Request-Method	Used during CORS interactions (preflight), indicates the HTTP method that will be used when the actual request is made	optional (used with CORS)
Access-Control-Request-Headers	Used during CORS interactions (preflight), indicates the HTTP headers that will be used when the actual request is made	optional (used with CORS)
Accept-Language	Used to indicate client's preference on natural language used in server responses	optional (used in servers with multi-lingual REST API support)

12 Responses

12.1 Headers

HTTP method	expected response	description
GET	200 (OK)	Resource in response body
	202 (ACCEPTED)	Resource is not yet ready (creation happens asynchronously)
	304 (NOT MODIFIED)	Resource has not been modified according to the conditional GET (cache) criteria
POST	201 (CREATED)	Creation performed / update performed (used with non idempotent update)
	202 (ACCEPTED)	Creation accepted (creation happens asynchronously) / update accepted (used with non idempotent update)
	204 (NO CONTENT)	Update performed (used when no response body is needed) (used with non idempotent update)
	412 (PRECONDITION FAILED)	Update not performed: the etag provided in the If-Match header does not match the current resource state (used with non idempotent update)
PUT	200 (OK)	Update performed (used when a response body is needed)
	202 (ACCEPTED)	Update accepted (update happens asynchronously)
	204 (NO CONTENT)	Update performed (used when no response body is needed)
	412 (PRECONDITION FAILED)	Update not performed: the etag provided in the If-Match header does not match the current resource state
DELETE	200 (OK)	Delete performed (used when a response body is needed)
	204 (NO CONTENT)	Delete performed (used when no response body is needed)

12.2 Error handling

- Error responses shall include a body that presents information about the actual error
- The body shall be encoded using JSON
- The body shall follow [RFC7807](https://tools.ietf.org/html/rfc7807)⁴⁷

RFC7808

```
{
  "type": "https://example.com/probs/out-of-credit",
  "title": "You do not have enough credit.",
  "detail": "Your current balance is 30, but that costs 50.",
  "instance": "/account/12345/msgs/abc",
  "balance": 30,
  "accounts": ["/account/12345",
               "/account/67890"]
}
```

12.3 Error codes

Notice that error code and HTTP status codes are aligned as much as possible, but do not necessarily need to match (there can be multiple error codes for one single HTTP status code)

Error code	Associated HTTP status	Description	Embedded resources	What to do
1	401 UNAUTHORIZED	Failed authentication	none	Check credentials
3	403 FORBIDDEN	Failed authorization	none	Check that user has enough privileges to perform operation
4	404 NOT FOUND	Resource not found	none	<ul style="list-style-type: none"> • Check resource URI • Check that user has enough privileges to perform operation • In case of asynchronous DELETE operation this error message is expected

⁴⁷ <https://tools.ietf.org/html/rfc7807>

Error code	Associated HTTP status	Description	Embedded resources	What to do
8	408 REQUEST TIMEOUT	Request timeout	none	Retry the request using asynchronous mode (Expect: 202-accepted)
9	409 CONFLICT	<p>The request could not be completed due to a conflict with the current state of the resource.</p> <p>Typically this error will happen when trying to create a resource that already exists</p>	<pre>"user": { "_links": { "self": { "href": "/user/mike" } } }</pre>	The resource has been previously created, find the resource in the embedded resources
10	410 GONE	Resource not anymore available under this URI (redirect not possible)	none	<ul style="list-style-type: none"> Upgrade client to new version of the API In case of asynchronous DELETE operation this error message is expected
12	412 PRECONDITION FAILED	Concurrency error	<pre>"etags": { "providedETag": "686897696a7c876b7e", "currentETag": "675946ae43146ff456" }</pre>	<ul style="list-style-type: none"> Abort the update, or Retry the update without concurrency requirement (If-Match header)
29	429 TOO MANY REQUESTS	Rate limit exceeded	none	Stop sending requests to server until time window has expired

13 CORS

Cross-site HTTP requests are HTTP requests for resources from a different domain than the domain of the resource making the request.

13.1 Simple requests

A simple cross-site request is one that:

- Only uses GET, HEAD or POST. If POST is used to send data to the server, the Content-Type of the data sent to the server with the HTTP POST request is one of application/x-www-form-urlencoded, multipart/form-data, or text/plain.
- Does not set custom headers with the HTTP Request (such as X-Modified, etc.)

Simple GET request

```
GET /users/mike
Accept: application/vnd.nokia-user+json

Origin: http://domain.com
```

Response

```
HTTP 200
Content-Type: application/vnd.nokia-user+json
Access-Control-Allow-Origin: *
{
  [...]
}
```

13.2 Preflight requests

For HTTP request methods that can cause side-effects on user data (in particular, for HTTP methods other than GET, or for POST usage with certain MIME types), the specification mandates that browsers "preflight" the request, soliciting supported methods from the server with an HTTP OPTIONS request method, and then, upon "approval" from the server, sending the actual request with the actual HTTP request method.

Unlike simple requests (discussed above), "preflighted" requests first send an HTTP request by the OPTIONS method to the resource on the other domain, in order to determine whether the actual request is safe to send. Cross-site requests are preflighted like this since they may have implications to user data. In particular, a request is preflighted if:

- It uses methods other than GET, HEAD or POST. Also, if POST is used to send request data with a Content-Type other than application/x-www-form-urlencoded, multipart/form-data, or text/plain, e.g. if the POST request sends an XML payload to the server using application/xml or text/xml, then the request is preflighted.
- It sets custom headers in the request (e.g. the request uses a header such as X-PINGOTHER)

Preflight request

```
OPTIONS /users
Accept: text/plain
Access-Control-Request-Method: POST
Origin: http://domain.com
```

Preflight response

```
HTTP 200
Content-Type: text/plain
Access-Control-Allow-Origin: http://domain.com

Access-Control-Allow-Methods: POST, GET, OPTIONS
Access-Control-Max-Age: 1728000
Vary: Accept-Encoding, Origin
```

POST request

```
POST /users
Accept: application/vnd.nokia-user+json
Content-Type: application/vnd.nokia-user+json
Access-Control-Request-Method: POST
Origin: http://domain.com
```

```
{
  "username": "mike",
  "realName": "Mike Kelly"
}
```

POST response

```
HTTP 201
Content-Type: application/vnd.nokia-user+json

Access-Control-Allow-Origin: http://domain.com

Vary: Accept-Encoding, Origin
```

```
{
  "username": "mike",
  "realName": "Mike Kelly"
  "_links": {
    "self": {
      "href": "/users/mike"
    }
  }
}
```


14 Asynchronous operations

Two major types of REST asynchronous operations are considered:

1. Standard REST verbs interactions (PUT, POST, DELETE) that take too long to perform in a single synchronous call, i.e.: the resource to create or update takes too long to be ready to be synchronous
2. General support for operations that take too long to perform as single REST interactions, e.g.: operations that include multiple REST resources or that require orchestration across multiple parties

14.1 Asynchronous REST verbs

Whenever a REST verb (POST, DELETE) interaction is going to take too long to be performed synchronously then the following pattern shall be used:

- The response to a PUT, PATCH, POST that requires asynchronous operation will be HTTP 202, where the body of the response provides a link to the resource being created / updated (this requires that the ID of the resource is reserved even if the resource is not yet fully created)
- The response to a DELETE that requires asynchronous operation will be HTTP 202, where the body of the response provides a link to the resource being deleted
- The client will regularly poll the link to the resource with GET:
 - The response will be HTTP 202 (where the body includes the link to self) when the resource is not yet ready
 - In case of PUT, PATCH and POST the response will be HTTP 200 (where the body is the actual resource) when the resource is ready
 - In case of DELETE the response will be HTTP 404 or HTTP 410 when the resource is finally deleted:
 - HTTP 404 is used by default as response
 - HTTP 410 may be used to indicate that the resource has been deleted, this requires that the resource cannot be re-created and the server keeps state information about deletion
- In case there is an error while creating the resource in the server then the server shall respond with an error code that best represents the error situation (see [Error handling\(see page 34\)](#) and [Error codes\(see page 34\)](#))

What does it mean "too long" in this context? A typical HTTP response time-out is 30 seconds, and this is considered to be already quite high for most interactions, this means that an operation that takes over 10 seconds should be considered "too long" to be synchronous

How often should clients poll? This is to be decided by each specific API as it is dependent on use case (e.g.: some resources may take 1 minute to build, which means that 5 seconds polling is a good strategy, while other resources may take 1 day to build, which means that 10 minutes polling is better fit)

In case the client wants to avoid asynchronous request (or wants to enforce asynchronous request) for a specific request then the "Expect" header shall be used

14.1.1 POST example

Initial POST request

```
POST /users
Content-Type: application/vnd.nokia-user+json
Accept: application/vnd.nokia-user+json, application/vnd.nokia-accepted-response+json

{
  "username": "mike",
  "realName": "Mike Kelly"
}
```

Accepted response

```
HTTP 202
Content-Type: application/vnd.nokia-accepted-response+json

{
  "_links": {
    "user": {
      "href": "/users/mike"
    }
  }
}
```

Polling GET request

```
GET /users/mike
Accept: application/vnd.nokia-user+json, application/vnd.nokia-accepted-response+json
```

OK response

```
HTTP 200
Content-Type: application/vnd.nokia-user+json

{
  "_links": {
    "user": {
      "href": "/users/mike"
    }
  }
  "username": "mike",
  "realName": "Mike Kelly"
}
```


14.1.2 DELETE example

Initial DELETE request

```
DELETE /users/mike
Accept: application/vnd.nokia-user+json, application/vnd.nokia-accepted-response+json
```

Accepted response

```
HTTP 202
Content-Type: application/vnd.nokia-accepted-response+json

{
  "_links": {
    "self": {
      "href": "/users/mike"
    }
  }
}
```

Polling GET request

```
GET /users/mike
Accept: application/vnd.nokia-user+json, application/vnd.nokia-accepted-response+json
```

"OK" response

```
HTTP 404
Content-Type: application/vnd.nokia-error-response+json

{
  "error": {
    "developerMessage": "Resource not found",
    "userMessage": "Resource not found",
    "errorCode": 4,
    "moreInfo": "http://www.example.gov/path/to/help/for/4"
  }
}
```

14.2 Jobs and workflows

It is expected that certain tasks will be abstracted as jobs or workflows, where the REST resource is a "job".

An API building block shall be created to generically support job/workflow management and shall include following functions:

- Defining a job

- Starting a job
- Scheduling a job
- Stopping a job
- Deleting a job
- Listing job history
- Obtaining logs related to a job
- Providing feedback on job execution and life-cycle via callback

14.3 Asynchronous REST verbs and jobs

- APIs may provide extended functionality to asynchronous REST verbs via job API building block, this is achieved by providing a link to the job created by the asynchronous REST verb
- Linking asynchronous REST verbs to jobs has the benefit that clients have better visibility over the progress of the operation (e.g.: logs, feedback, ETA) and have control over the operation (e.g.: cancel operation)

Example of POST request response:

Accepted response

```
HTTP 202
Content-Type: application/vnd.nokia-accepted-response+json

{
  "_links": {
    "user": { "href": "/users/mike" },
    "job": { "href": "/jobs/5647" }
  }
}
```

15 Concurrency control

- APIs that require concurrency control and can provide such control via optimistic locking shall use etags and If-Match headers capabilities
- A typical interaction looks as follows:

GET request

```
GET /users/mike
Accept: application/vnd.nokia-user+json
```

OK response with etag

```
HTTP 200
Content-Type: application/vnd.nokia-user+json
ETag: "686897696a7c876b7e"

{
  "_links": {
    "user": {
      "href": "/users/mike"
    }
  }
  "username": "mike",
  "realName": "Mike Kelly"
}
```

PATCH request with etag

```
PATCH /users/mike
Content-Type: application/vnd.nokia-user+json
If-Match: "686897696a7c876b7e"

[
  { "op": "replace", "path": "/email", "new.email@example.org" }
]
```

Pre-condition failed response because etags don't match (someone else updated the resource before us)

HTTP 412

Content-Type: application/vnd.nokia-error-response+json

```
{
  "error": {
    "developerMessage": "The operation could not be performed because etags do not match",
    "userMessage": "The operation could not be performed because of a concurrency error",
    "errorCode": 12,
    "moreInfo": "http://www.example.gov/path/to/help/for/12",
    "_embedded": {
      "etags": {
        "providedETag": "686897696a7c876b7e"
        "currentETag": "675946ae43146ff456"
      }
    }
  }
}
```

16 Pagination

- REST APIs that return multiple resources shall be paginated to a default number of items, the default number of items used is specified by each API building block independently (a good rule of thumb is to use 30 as default)
 - Notice that in case there is a known upper limit of resources to be provided in the response, and this amount is considered to fit one single response then there is no need for pagination
- REST APIs shall use cursor based pagination
- REST APIs shall provide selection of cursor using ?before or ?after parameters
 - ?before parameter refers to items before the provided cursor ID
 - ?after parameter refers to items after the provided cursor ID
- Clients shall not make any assumptions about cursor ID values (they can be randomly generated UUID that are not following lexicographical order)
- Clients shall not make any assumptions about durability of cursor IDs (they might become invalidated with time)
- REST APIs should provide custom size of the page using ?limit parameter
- REST APIs shall provide cursor navigation via links in the responses:
 - prev: reference to the previous page
 - next: reference to the next page
- A typical interaction looks as follows:

GET request to a list of resources

```
GET /users
Accept: application/vnd.nokia-user+json
```

Response with first page (no previous link)

```
{
  "_links": {
    "next": { "href": "/users?after=MTAxNTExOTQ1MjAwNzI5NDE&limit=30" },
    "user": [
      {
        "href": "/users/mike",
        "title": "Mike Kelly"
      },
      [...]
    ]
  }
}
```

GET second (and last) page

```
GET /users?after=MTAxNTExOTQ1MjAwNzI5NDE&limit=30
Accept: application/vnd.nokia-user+json
```

Response with second (and last) page (no next link)

```
{
  "_links": {
    "prev": { "href": "/users?previous=NDMyNzQyODI3OTQw&limit=30" },
    "user": [
      {
        "href": "/users/mture",
        "title": "Mark Ture"
      },
      [...]
    ]
  }
}
```

17 Rate limiting

In case an API wants to provide rate limiting capabilities the following shall be taken into consideration:

- API shall document rate limiting policies:
 - number of requests allowed per client
 - time window used to reset the number of allowed requests
- API shall document what end-points are protected by rate limiting
- API shall use the following headers to indicate rate limiting status:

Header Name	Description
X-RateLimit-Limit	The maximum number of requests that the consumer is permitted to make per time window.
X-RateLimit-Remaining	The number of requests remaining in the current rate limit window.
X-RateLimit-Reset	The time at which the current rate limit window resets in UTC epoch seconds ⁴⁸ .

- In case a client exceeds rate limits then HTTP 429 shall be returned

⁴⁸ http://en.wikipedia.org/wiki/Unix_time

18 Non-REST interactions

Even though the aim when designing REST APIs is that they are RESTful, there are some cases where modeling interactions in a RESTful manner might lead to unnecessary complexity and might make the API difficult to understand and use. In such cases there might be a desire to use HTTP "lightweight" interactions (i.e.: non-SOAP) even if the resulting API is not RESTful.

Most of the guidelines depicted above are applicable to such HTTP "lightweight" APIs:

- Encoding
- Authentication
- Authorization
- Media types
- Versioning
- URL structure (except resource leafs)
- HTTP Headers
- Error handling
- Asynchronous operations

19 REST API Documentation

REST APIs shall use [S⁴⁹wagger⁵⁰UI⁵¹](http://swagger.io/swagger-ui/) as documentation framework

⁴⁹ <http://swagger.io/swagger-ui/>

⁵⁰ <http://swagger.io/swagger-ui/>

⁵¹ <http://swagger.io/swagger-ui/>