INTERNET DRAFT
Document: draft-dejong-remotestorage-02

Intended Status: Proposed Standard
Expires: 13 June 2014

Michiel B. de Jong
(independent)
F. Kooman
(independent)
10 December 2013

remoteStorage

Abstract

This draft describes a protocol by which client-side applications, running inside a web browser, can communicate with a data storage server that is hosted on a different domain name. This way, the provider of a web application need not also play the role of data storage provider. The protocol supports storing, retrieving, and removing individual documents, as well as listing the contents of an individual directory, and access control is based on bearer tokens.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 13 June 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction

Many services for data storage are available over the internet. This specification describes a vendor-independent interface for such services. It is based on https, CORS and bearer tokens. The metaphor for addressing data on the storage is that of folders containing documents and subfolders. The actions the interface exposes are:

* GET a folder: retrieve the names and current versions of the documents and subfolders currently contained by the folder

* GET a document: retrieve its content type, current version, and contents

* PUT a document: store a new version, its content type, and contents, conditional on the current version

* DELETE a document: remove it from the storage, conditional on the current version

* HEAD a folder or document: like GET, but omitting the response body

The exact details of these four actions are described in this specification.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [WORDS].

"SHOULD" and "SHOULD NOT" are appropriate when valid exceptions to a general requirement are known to exist or appear to exist, and it is infeasible or impractical to enumerate all of them. However, they should not be interpreted as permitting implementors to fail to implement the general requirement when such failure would result in interoperability failure.

3. Storage model

The server stores data in nodes that form a tree structure. Internal nodes are called 'folders' and leaf nodes are called 'documents'. For a folder, the server stores references to nodes contained in the folder, and it should be able to produce a list of them, with for each contained item:

* item name
* item type (folder or document)
* current version
* content type
* content length

For a document, the server stores, and should be able to produce:

* current version
* content type
* content length
* content

4. Requests

Client-to-server requests SHOULD be made over https [HTTPS], and servers SHOULD comply with HTTP/1.1 [HTTP]. Specifically, they SHOULD support chunked transfer coding on PUT requests. Servers MAY also offer an optional switch from https to SPDY [SPDY].

The root folder of the storage tree is represented by the following URL:

    URI_ENCODE(<storage_root> '/')

Subsequently, if <parent_folder> is the URL of a folder, then the URL of an item contained in it is:

    URI_ENCODE(<parent_folder> <document_name>)

for a document, or:

    URI_ENCODE(<parent_folder> <folder_name> '/')

for a folder. Item names MAY contain all characters except '/' and the null character, and MUST NOT have zero length.

A document description is a map containing one string-valued 'ETag' field, one string-valued 'Content-Type' and one integer-valued 'Content-Length' field. They represent the document's current version, its content type, and its content length respectively. Note that content length is measured in octets (bytes), not in characters.

A folder description is a map containing a string-valued 'ETag' field, representing the folder's current version.

A successful GET request to a folder SHOULD be responded to with a JSON-LD [JSON-LD] document (content type 'application/json'),

containing as its 'items' field a map in which contained documents appear as entries <item_name> to a document description, and contained folders appear as entries <item_name> '/' to a folder description. It SHOULD furthermore contain an '@context' field with the value 'http://remotestorage.io/spec/folder-description'. For instance:

```
{
  "@context": "http://remotestorage.io/spec/folder-description",
  "items": {
    "abc": {
      "ETag": "DEADBEEFDEADBEEFDEADBEEF",
      "Content-Type": "image/jpeg",
      "Content-Length": 82352
    },
    "def/": {
      "ETag": "1337ABCD1337ABCD1337ABCD"
    }
  }
}
```

Empty folders are treated as non-existing, and therefore GET requests to them SHOULD be responded to with either a 404 response OR the JSON document representing an empty map ("{}"). However, an empty folder MUST NOT be listed as an item in its parent folder.

Also, folders SHOULD be created silently, as necessary to contain newly added items. This way, PUT and DELETE requests only need to be made to documents, and folder management becomes an implicit result.

A successful GET request to a document SHOULD be responded to with the full document contents in the body, the document's content type in a 'Content-Type' header, its content length in octets (not in characters) in a 'Content-Length' header, and the document's current version in an 'ETag' header. Content-Range headers on GET requests MAY be supported by the server [HTTP].

A successful PUT request to a document MUST result in:

* the request body being stored as the document's new content,
* parent and further ancestor folders being silently created as necessary, with the document (name and version) being added to its parent folder, and each folder added to its subsequent

parent,
* the value of its Content-Type header being stored as the document's new content type,
* its version being updated, as well as that of its parent folder and further ancestor folders, using a strong validator [HTTP, section 13.3.3].

The response MUST contain a strong ETag header, with the document's new version (for instance a hash of its contents) as its value.

A successful DELETE request to a document MUST result in:

* the deletion of that document from the storage, and from its parent folder,
* silent deletion of the parent folder if it is left empty by this, and so on for further ancestor folders,
* the version of its parent folder being updated, as well as that of further ancestor folders.

A successful OPTIONS request SHOULD be responded to as described in the CORS section below.

A successful HEAD request SHOULD be responded to like to the equivalent GET request, but omitting the response body.

5. Response codes

Response codes SHOULD be given as defined by [HTTP, section 10] and [BEARER, section 3.1]. The following is a non-normative checklist of status codes that are likely to occur in practice:

* 500 if an internal server error occurs,
* 429 if the client makes too frequent requests or is suspected of malicious activity,
* 414 if the request URI is too long,
* 416 if Range requests are supported by the server and the Range request can not be satisfied,
* 401 for all requests that don't have a bearer token with sufficient permissions,
* 404 for all DELETE and GET requests to nodes that do not exist on the storage,
* 304 for a conditional GET request whose pre-condition fails (see "Versioning" below),

* 409 for a PUT request where any folder name in the path clashes with an existing document's name at the same level, or where the document name coincides with an existing folder's name at the same level.
* 412 for a conditional PUT or DELETE request whose pre-condition fails (see "Versioning" below),
* 507 in case the user's account is over its storage quota,
* 4xx for all malformed requests (e.g. foreign characters in the path), as well as for all PUT and DELETE requests to folders,
* 2xx for all successful requests.

Clients SHOULD also handle the case where a response takes too long to arrive, or where no response is received at all.

6. Versioning

All successful requests MUST return an 'Expires: 0' header, and an 'ETag' header [HTTP] with, in the case of GET, the current version, in the case of PUT, the new version, and in case of DELETE, the version that was deleted. PUT and DELETE requests MAY have an 'If-Match' request header [HTTP], and MUST fail with a 412 response code if that doesn't match the document's current version.

GET requests MAY have a comma-separated list of revisions in an 'If-None-Match' header [HTTP], and SHOULD be responded to with a 412 response if that list includes the document or folder's current version. A PUT request MAY have an 'If-None-Match: *' header [HTTP], in which case it MUST fail with a 412 response code if the document already exists.

In all 'ETag', 'If-Match' and 'If-None-Match' headers, revision strings should appear inside double quotes (").

A provider MAY offer version rollback functionality to its users, but this specification does not define the user interface for that.

7. CORS headers

All responses MUST carry CORS headers [CORS]. The server MUST also reply to OPTIONS requests as per CORS. For GET requests, a wildcard origin MAY be returned, but for PUT and DELETE requests, the response MUST echo back the Origin header sent by the client.

8. Session description

The information that a client needs to receive in order to be able to connect to a server SHOULD reach the client as described in the 'bearer token issuance' sections below. It consists of:

* <storage_root>, consisting of 'https://' followed by a server host, and optionally a server port and a path prefix as per [IRI]. Examples:
  * 'https://example.com' (host only)
  * 'https://example.com:8080' (host and port)
  * 'https://example.com/path/to/storage' (host, port and path prefix; note there is no trailing slash)
* <access_token> as per [OAUTH]. The token SHOULD be hard to guess and SHOULD NOT be reused from one client to another. It can however be reused in subsequent interactions with the same client, as long as that client is still trusted. Example:
  * 'ofb24f1ac3973e70j6vts19qr9v2eei'
* <storage_api>, always 'draft-dejong-remotestorage-02' for this alternative version of the specification.

The client can make its requests using https with CORS and bearer tokens, to the URL that is the concatenation of <storage_root> with '/' plus one or more <folder> '/' strings indicating a path in the folder tree, followed by zero or one <document> strings, indicating a document. For example, if <storage_root> is "https://storage.example.com/bob", then to retrieve the folder contents of the /public/documents/ folder, or to retrieve a 'draft.txt' document from that folder, the client would make requests to, respectively:

* https://storage.example.com/bob/public/documents/
* https://storage.example.com/bob/public/documents/draft.txt

9. Bearer tokens and access control

A bearer token represents one or more access scopes. These access scopes are represented as strings of the form <module> <level>, where the <module> string SHOULD be lower-case alphanumerical, other than the reserved word 'public', and <level> can be ':r' or ':rw'. The access the bearer token gives is the sum of its access scopes, with each access scope representing the following permissions:

'*:rw') any request,

'*:r') any GET or HEAD request,

<module> ':rw') any requests to paths that start with
'/' <module> '/' or '/public/' <module> '/',

<module> ':r') any GET or HEAD requests to paths that start with
'/' <module> '/' or '/public/' <module> '/',

As a special exceptions, GET requests to a document (but not a folder) whose path starts with '/public/' are always allowed. They, as well as OPTIONS requests, can be made without a bearer token. All other requests should present a bearer token with sufficient access scope, using a header of the following form (no double quotes here):

```
Authorization: Bearer <access_token>
```

In addition, providing the access token via a HTTP query parameter for GET requests MAY be supported by the server, although its use is not recommended, due to its security deficiencies; see [BEARER, section 2.3].

10. Application-first bearer token issuance

To make a remoteStorage server available as 'the remoteStorage of <user> at <host>', exactly one link of the following format SHOULD be added to the webfinger record [WEBFINGER] of <user> at <host>:

```
{
  "href": <storage_root>,
  "rel": "remotestorage",
  "properties": {
    "http://remotestorage.io/spec/version": <storage_api>,
    "http://tools.ietf.org/html/rfc6749#section-4.2": <auth-dialog>,
    "http://tools.ietf.org/html/rfc6750#section-2.3": <query-param>,
    "https://tools.ietf.org/html/rfc2616#section-14.16": <ranges>
  }
}
```

Here <storage_root> and <storage_api> are as per "Session description" above, and <auth-dialog> SHOULD be a URL where an OAuth 2.0 implicit-grant flow dialog [OAUTH] is presented, so the

user can supply their credentials (how, is out of scope), and allow or reject a request by the connecting application to obtain a bearer token for a certain list of access scopes.

The <query-param> variable SHOULD have the boolean value true if the server supports passing the bearer token in the URI query parameter as per section 2.3 of [BEARER], and false otherwise.

The <ranges> variable SHOULD have a string value of "GET" if Content-Range headers are supported for GET requests as per [HTTP, section 14.16], and the boolean value false if not.

The server SHOULD NOT expire bearer tokens unless they are revoked, and MAY require the user to register applications as OAuth clients before first use; if no client registration is required, then the server MAY ignore the client_id parameter in favor of relying on the redirect_uri parameter for client identification.

11. Storage-first bearer token issuance

The provider MAY also present a dashboard to the user, where they have some way to add open web app manifests [MANIFEST]. Adding a manifest to the dashboard is considered equivalent to clicking 'accept' in the dialog of the application-first flow. Removing one is considered equivalent to revoking its access token.

As an equivalent to OAuth's 'scope' parameter, a 'remotestorage' field SHOULD be present in the root of such an application manifest document, as a JSON array of strings, each string being one access scope of the form <module> <level>.

When the user gestures they want to use a certain application whose manifest is present on the dashboard, the dashboard SHOULD redirect to the application or open it in a new window. To mimic coming back from the OAuth dialog, it MAY add 'access_token' and 'scope' parameters to the URL fragment.

Regardless of whether 'access_token' and 'scope' are specified, it SHOULD add a 'remotestorage' parameter to the URL fragment, with a value of the form <user> '@' <host>. When the application detects this parameter, it SHOULD resolve the webfinger record for <user> at <host> and extract the <storage_root> and <storage_api> information.

If no access_token was given, then the application SHOULD also extract the <auth_endpoint> information from webfinger, and continue as per application-first bearer token issuance.

Note that whereas a remoteStorage server SHOULD offer support of the application-first flow with webfinger and OAuth, it MAY choose not to support the storage-first flow, provided that users will easily remember their <user> '@' <host> webfinger address at that provider. Applications SHOULD, however, support both flows, which means checking the URL for a 'remotestorage' parameter, but giving the user a way to specify their webfinger address if there is none.

If a server provides an application manifest dashboard, then it SHOULD merge the list of applications there with the list of issued access tokens as specified by OAuth into one list. Also, the interface for revoking an access token as specified by OAuth SHOULD coincide with removing an application from the dashboard.

12. Example wire transcripts

The following examples are not normative ("\" indicates a line was wrapped).

12.1. WebFinger

In application-first, an in-browser application might issue the following request, using XMLHttpRequest and CORS:

```
        GET /.well-known/webfinger?resource=acct:michiel@michielbdejon\
g.com HTTP/1.1
        Host: michielbdejong.com
```

and the server's response might look like this:

```
        HTTP/1.1 200 OK
        Access-Control-Allow-Origin: *
        Access-Control-Allow-Methods: GET
        Access-Control-Allow-Headers: If-Match, If-None-Match
        Access-Control-Expose-Headers: ETag, Content-Type, Content-Len\
gth

        {
          "links":[{
```

```
            "href": "https://michielbdejong.com:7678/inbox",
            "rel": "post-me-anything"
          }, {
            "href": "https://michielbdejong.com/me.jpg",
            "rel": "avatar"
          }, {
            "href": "https://3pp.io:4439/storage/michiel",
            "rel": "remotestorage",
            "properties": {
              "http://remotestorage.io/spec/version": "draft-dejong-re\
motestorage-02",
              "http://tools.ietf.org/html/rfc6750#section-4.2": "https\
://3pp.io:4439/oauth/michiel",
              "http://tools.ietf.org/html/rfc6750#section-2.3": false,
              "http://tools.ietf.org/html/rfc2616#section-14.16": false
            }
          }]
        }
```

12.2. OAuth dialog form

Once the in-browser application has discovered the server's OAuth end-point, it will typically redirect the user to this URL, in order to obtain a bearer token. Say the application is hosted on https://drinks-unhosted.5apps.com/ and wants read-write access to the user's "myfavoritedrinks" scope:

```
GET /oauth/michiel?redirect_uri=https%3A%2F%2Fdrinks-unhosted.5\
apps.com%2F&scope=myfavoritedrinks%3Arw&client_id=https%3A%2F%2Fdrinks-\
unhosted.5apps.com&response_type=token HTTP/1.1
Host: 3pp.io
```

The server's response might look like this (truncated for brevity):

```
HTTP/1.1 200 OK

<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Allow access?</title>
...
```

12.3. OAuth dialog form submission

When the user submits the form, the request would look something like this:

```
    POST /oauth HTTP/1.1
    Host: 3pp.io:4439
    Origin: https://3pp.io:4439
    Content-Type: application/x-www-form-urlencoded
    Referer: https://3pp.io:4439/oauth/michiel?redirect_uri=https%3\
A%2F%2Fdrinks-unhosted.5apps.com%2F&scope=myfavoritedrinks%3Arw&client_\
id=https%3A%2F%2Fdrinks-unhosted.5apps.com&response_type=token

    client_id=https%3A%2F%2Fdrinks-unhosted.5apps.com&redirect_uri=\
https%3A%2F%2Fdrinks-unhosted.5apps.com%2F&response_type=token&scope=my\
favoritedrinks%3Arw&state=&username=michiel&password=something&allow=Al\
low
```

To which the server could respond with a 302 redirect, back to the origin of the requesting application:

```
    HTTP/1.1 200 OK
    Location:https://drinks-unhosted.5apps.com/#access_token=j2YnGt\
XjzzzHNjkd1CJxoQubA1o%3D&token_type=bearer&state=
```

12.4. OPTIONS preflight

When an in-browser application makes a cross-origin request which may affect the server-state, the browser will make a preflight request first, with the OPTIONS verb, for instance:

```
    OPTIONS /storage/michiel/myfavoritedrinks/ HTTP/1.1
    Host: 3pp.io:4439
    Access-Control-Request-Method: GET
    Origin: https://drinks-unhosted.5apps.com
    Access-Control-Request-Headers: Authorization
    Referer: https://drinks-unhosted.5apps.com/
```

To which the server can for instance respond:

```
    HTTP/1.1 200 OK
    Access-Control-Allow-Origin: https://drinks-unhosted.5apps.com
    Access-Control-Allow-Methods: GET, PUT, DELETE
```

```
        Access-Control-Allow-Headers: Authorization, Content-Length, Co\
ntent-Type, Origin, X-Requested-With, If-Match, If-None-Match
```

12.5. Initial PUT

An initial PUT may contain an 'If-None-Match: *' header, like this:

```
        PUT /storage/michiel/myfavoritedrinks/test HTTP/1.1
        Host: 3pp.io:4439
        Content-Length: 91
        Origin: https://drinks-unhosted.5apps.com
        Authorization: Bearer j2YnGtXjzzzHNjkd1CJxoQubA1o=
        Content-Type: application/json; charset=UTF-8
        Referer: https://drinks-unhosted.5apps.com/?
        If-None-Match: *

        {"name":"test","@context":"http://remotestorage.io/spec/modules\
/myfavoritedrinks/drink"}
```

And the server may respond with either a 201 Created or a 200 OK status:

```
        HTTP/1.1 201 Created
        Access-Control-Allow-Origin: https://drinks-unhosted.5apps.com
        ETag: "1382694045000"
```

12.6. Subsequent PUT

A subsequent PUT may contain an 'If-Match' header referring to the ETag previously returned, like this:

```
        PUT /storage/michiel/myfavoritedrinks/test HTTP/1.1
        Host: 3pp.io:4439
        Content-Length: 91
        Origin: https://drinks-unhosted.5apps.com
        Authorization: Bearer j2YnGtXjzzzHNjkd1CJxoQubA1o=
        Content-Type: application/json; charset=UTF-8
        Referer: https://drinks-unhosted.5apps.com/?
        If-Match: "1382694045000"

        {"name":"test", "updated":true, "@context":"http://remotestorag\
e.io/spec/modules/myfavoritedrinks/drink"}
```

And the server may respond with a 412 Conflict or a 200 OK status:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: https://drinks-unhosted.5apps.com
ETag: "1382694048000"
```

12.7. GET

A GET request would also include the bearer token, and optionally an If-None-Match header:

```
GET /storage/michiel/myfavoritedrinks/test HTTP/1.1
Host: 3pp.io:4439
Origin: https://drinks-unhosted.5apps.com
Authorization: Bearer j2YnGtXjzzzHNjkd1CJxoQubA1o=
Referer: https://drinks-unhosted.5apps.com/?
If-None-Match: "1382694045000", "1382694048000"

{"name":"test", "updated":true, "@context":"http://remotestora\
ge.io/spec/modules/myfavoritedrinks/drink"}
```

And the server may respond with a 304 Not Modified or a 200 OK status:

```
HTTP/1.1 304 Not Modified
Access-Control-Allow-Origin: https://drinks-unhosted.5apps.com
ETag: "1382694048000"
```

12.8. DELETE

A DELETE request may look like this:

```
DELETE /storage/michiel/myfavoritedrinks/test HTTP/1.1
Host: 3pp.io:4439
Origin: https://drinks-unhosted.5apps.com
Authorization: Bearer j2YnGtXjzzzHNjkd1CJxoQubA1o=
Content-Type: application/json; charset=UTF-8
Referer: https://drinks-unhosted.5apps.com/?
If-Match: "1382694045000"
```

And the server may respond with a 412 Conflict or a 200 OK status:

```
HTTP/1.1 412 Conflict
```

```
Access-Control-Allow-Origin: https://drinks-unhosted.5apps.com
ETag: "1382694048000"
```

13. Distributed versioning

This section is non-normative, and is intended to explain some of the design choices concerning ETags and folder listings. At the same time it will hopefully help readers who intend to develop an application that uses remoteStorage as its per-user data storage. When multiple clients have read/write access to the same document, versioning conflicts may occur. For instance, client A may make a PUT request that changes the document from version 1 to version 2, after which client B may make a PUT request attempting to change the same document from version 1 to version 3.

In this case, client B can add an 'If-Match: "1"' header, which would trigger a 412 Conflict response code, since the current version ("2") does not match the version required as a condition by the header If-Match header ("1").

Client B is now aware of the conflict, and may consult the user, saying the update to version 3 failed. The user may then choose, through the user interface of client B, whether version 2 or version 3 should be kept, or maybe the document should be reverted on the server to version 1, or a merged version 4 is needed. Client B may then make a request that puts the document to the version the user wishes; this time setting an 'If-Match: "2"' header instead.

Both client A and client B would periodically poll the root directory of each scope they have access to, to see if the version of the root directory changed. If it did, then one of the versions listed in there will necessarily have changed, and the client can make a GET request to that child directory or document, to obtain its latest version.

Because an update in a document will result in a version change of its containing folder, and that change will propagate all the way to the root folder, it is not necessary to poll each document for changes individually.

As an example, the root folder may contain 10 directories, each of which contain 10 directories, which each contain 10 documents, so their paths would be for instance '/0/0/1', '/0/0/2',

etcetera. Then one GET request to the root folder '/' will be enough to know if any of these 1000 documents has changed.

Say document '/7/9/2' has changed; then the GET request to '/' will come back with a different ETag, and entry '7/' will have a different value in its JSON content. The client could then request '/7/', '/7/9/', and '/7/9/2' to narrow down the one document that caused the root directory's ETag to change.

Note that the remoteStorage server does not get involved in the conflict resolution. It keeps the canonical current version at all times, and allows clients to make conditional GET and PUT requests, but it is up to whichever client discovers a given version conflict, to resolve it.

14. Security Considerations

To prevent man-in-the-middle attacks, the use of https instead of http is important for both the interface itself and all end-points involved in webfinger, OAuth, and (if present) the storage-first application launch dashboard.

A malicious party could link to an application, but specifying a remoteStorage user address that it controls, thus tricking the user into using a trusted application to send sensitive data to the wrong remoteStorage server. To mitigate this, applications SHOULD clearly display to which remoteStorage server they are sending the user's data.

Applications could request scopes that the user did not intend to give access to. The user SHOULD always be prompted to carefully review which scopes an application is requesting.

An application may upload malicious html pages and then trick the user into visiting them, or upload malicious client-side scripts, that take advantage of being hosted on the user's domain name. The origin on which the remoteStorage server has its interface SHOULD therefore NOT be used for anything else, and the user SHOULD be warned not to visit any web pages on that origin. In particular, the OAuth dialog and launch dashboard or token revokation interface SHOULD be on a different origin than the remoteStorage interface.

Where the use of bearer tokens is impractical, a user may choose to

store documents on hard-to-guess URLs whose path after <storage_root> starts with '/public/', while sharing this URL only with the intended audience. That way, only parties who know the document's hard-to-guess URL, can access it. The server SHOULD therefore make an effort to detect and stop brute-force attacks that attempt to guess the location of such documents.

The server SHOULD also detect and stop denial-of-service attacks that aim to overwhelm its interface with too much traffic.

15. IANA Considerations

This document registers the 'remotestorage' link relation, as well as the following WebFinger properties:

* "http://remotestorage.io/spec/version"
* "http://tools.ietf.org/html/rfc6749#section-4.2"
* "http://tools.ietf.org/html/rfc6750#section-2.3"
* "https://tools.ietf.org/html/rfc2616#section-14.16

16. Acknowledgements

The authors would like to thank everybody who contributed to the development of this protocol, including Kenny Bentley, Javier Diaz, Daniel Groeber, Bjarni Runar, Jan Wildeboer, Charles Schultz, Peter Svensson, Valer Mischenko, Michiel Leenaars, Jan-Christoph Borchardt, Garret Alfert, Sebastian Kippe, Max Wiehle, Melvin Carvalho, Martin Stadler, Geoffroy Couprie, Niklas Cathor, Marco Stahl, James Coglan, Ken Eucker, Daniel Brolund, elf Pavlik, Nick Jennings, Markus Sabadello, Steven te Brinke, Matthias Treydte, Rick van Rein, Mark Nottingham, Julian Reschke, and Markus Lanthaler, among many others.

17. References

17.1. Normative References

[WORDS]
Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[IRI]
Duerst, M., "Internationalized Resource Identifiers (IRIs)", RFC 3987, January 2005.

[WEBFINGER]
Jones, P., Salguerio, G., Jones, M, and Smarr, J., "WebFinger", RFC7033, September 2013.

[OAUTH]
"Section 4.2: Implicit Grant", in: Hardt, D. (ed), "The OAuth 2.0 Authorization Framework", RFC6749, October 2012.

17.2. Informative References

[HTTPS]
Rescorla, E., "HTTP Over TLS", RFC2818, May 2000.

[HTTP]
Fielding et al., "Hypertext Transfer Protocol -- HTTP/1.1", RFC2616, June 1999.

[SPDY]
Mark Belshe, Roberto Peon, "SPDY Protocol - Draft 3.1", http://www.chromium.org/spdy/spdy-protocol/spdy-protocol-draft3-1, September 2013.

[JSON-LD]
M. Sporny, G. Kellogg, M. Lanthaler, "JSON-LD 1.0", W3C Proposed Recommendation, http://www.w3.org/TR/2013/PR-json-ld-20131105/, November 2013.

[CORS]
van Kesteren, Anne (ed), "Cross-Origin Resource Sharing -- W3C Candidate Recommendation 29 January 2013", http://www.w3.org/TR/cors/, January 2013.

[MANIFEST]
Mozilla Developer Network (ed), "App manifest -- Revision 330541", https://developer.mozilla.org/en-US/Apps/Developing/Manifest$revision/482369, October 2013.

[BEARER]
M. Jones, D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC6750, http://tools.ietf.org/html/rfc6750#section-2.3, October 2012.

18. Authors' addresses

Michiel B. de Jong
(independent)

Email: michiel@michielbdejong.com

F. Kooman
(independent)

Email: fkooman@tuxed.net