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 folder, 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 2 June 2017.

Copyright Notice

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 five 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 MUST comply with HTTP/1.1 [HTTP]. Specifically, they
   MUST support chunked transfer coding on PUT requests. Servers MAY
   also offer an optional switch to HTTP/2 [HTTP/2].

   A request is considered successful if the HTTP response code is in
   the 2xx range (e.g. 200 OK, 201 Created), and unsuccessful if an
   error occurred or a condition was not met, e.g. response code 404
   Not Found, 304 Not Modified.

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

        URI_ENCODE(<storage_root> '/')

   Subsequently, let <parent_folder> be the URL of a folder, i.e. ends
   with a '/', 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, before URI_ENCODE, except '/'
   and the null character '\0' and MUST NOT have zero length. Item
   names MUST NOT be equal to '.' or to '..', as those have a special
   semantic in URIs (Section 5.2.4 of [URI]).

   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 MUST be responded to with a
JSON-LD [JSON-LD] document (content type 'application/ld+json'),
containing as its 'items' field a map in which contained documents
appear as entries <item_name> to a document description, and
contained non-empty folders appear as entries <item_name> '/' to a
folder description. It MUST also 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"
    }
  }
}
```

GET requests to empty folders SHOULD be responded to with a folder
description with no items (the items field set to '{}'). However, an
empty folder MUST NOT be listed as an item in its parent folder.

PUT and DELETE requests only need to be made to documents, and never
to folders. A document PUT will make all ancestor folders along its
path become non-empty; deleting the last document from a subtree
will make that whole subtree become empty. Folders will therefore
show up in their parent folder descriptions if and only if their
subtree contains at least one document.

In contexts outside of this document, non-empty folders may be
called 'existent', while empty folders may be called 'non-existent'.

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 as a strong ETag in an 'ETag' header.

Note that the use of strong ETags prohibits changing the response
body based on request headers; in particular, the server will not be
able to serve the same document uncompressed to some clients and
compressed to other clients when requested, since the two bodies
would not be identical byte-for-byte.

Servers MAY support Content-Range headers [RANGE] on GET requests,
but whether or not they do SHOULD be announced both through the
"http://tools.ietf.org/html/rfc7233" option mentioned below in
section 10 and through the HTTP 'Accept-Ranges' response header.

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 7.2].

If not exactly one Content-Type header was received as part of a
PUT request, the Content-Type header value contains non-ASCII
characters, or it is unreasonably long, the server MAY refuse to
process the request, and instead respond with a descriptive error
message in the body, and a http response code from the 4xx range.

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

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

5. Response codes

   Response codes SHOULD be given as defined by [HTTP, section 6] and
   [BEARER, section 3.1]. The following is a non-normative list of
   status codes that are likely to occur in practice:

     * 500 if an internal server error occurred,
     * 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 require a valid bearer token and
           where no valid one was sent (see also [BEARER, section
           3.1]),
     * 403 for all requests that have insufficient scope, e.g.
           accessing a <module> for which no scope was obtained, or
           accessing data outside the user's <storage_root>,
     * 404 for all DELETE, GET and HEAD requests to documents that do
           not exist on the storage,
     * 304 for a conditional GET request whose precondition
           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 precondition
           fails (see "Versioning" below),
     * 507 in case the account is over its storage quota,
     * 4xx for all malformed requests, e.g. reserved characters in the
           path [URI, section 2.2], 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 GET, HEAD, PUT and DELETE requests MUST return an
     'ETag' header [HTTP] with, in the case of GET and HEAD the current
     version, in the case of PUT, the new version, and in case of DELETE,
     the version that was deleted. All successful GET requests MUST
     return a 'Cache-Control: no-cache' header. PUT and DELETE requests
     MAY have an 'If-Match' request header [COND], and MUST fail with a
     412 response code if that does not match the document's current
     version.

     All successful requests MUST return 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. All
     successful GET requests MUST return an 'Cache-Control: no-cache'
     header. PUT and DELETE requests MAY have an 'If-Match' request
     header [COND], and MUST fail with a 412 response code if that
     does not match the document's current version.

     GET requests MAY have a comma-separated list of revisions in an
     'If-None-Match' header [COND], and SHOULD be responded to with a 304
     response if that list includes the document or folder's current
     version. A PUT request MAY have an 'If-None-Match: *' header [COND],
     in which case it MUST fail with a 412 response code if the document
     already exists.

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

7. CORS headers

     All responses MUST carry CORS headers [CORS]. The server MUST also
     reply to preflight OPTIONS requests as per CORS.

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-08' 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 relative to <storage_root>
               that start with '/' <module> '/' or
               '/public/' <module> '/',

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

As a special exceptions, GET and HEAD 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. Unless [KERBEROS] is used (see section 10 below), 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]. If supported, this SHOULD be announce through the
"http://tools.ietf.org/html/rfc6750#section-2.3" WebFinger property
as per section 10 below.

10. Application-first bearer token issuance

To make a remoteStorage server available as 'the remoteStorage of
the person identified by <uri>', exactly one link of the following
format SHOULD be added to the WebFinger record [WEBFINGER] for
<uri>:

```
{
  "href": <storage_root>,
  "rel": "http://tools.ietf.org/id/draft-dejong-remotestorage",
  "properties": {
    "http://remotestorage.io/spec/version": <storage_api>,
    "http://tools.ietf.org/html/rfc6749#section-4.2": <auth-dialog>,
    "...": "...",
  }
```

    }

    A common way of identifying persons as <user> at <host> is through a
    URI of the format "acct:<user>@<host>". Persons who use a personal
    domain name, not shared with any other users, can be identified by
    a URI of the format "http://<host>/" (see [WEBFINGER, section 4.1]).

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

    If <auth-dialog> is a URL, the user can supply their credentials
    for accessing the account (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. Note that an account
    will often belong to just one human user, but may also belong to a
    group of multiple users (the remoteStorage of <group> at <host>).

    If <auth-dialog> is null, the client will not have a way to obtain
    an access token, and SHOULD send all requests without Authorization
    header, and rely on Kerberos [KERBEROS] instead for requests that
    would normally be sent with a bearer token, but servers SHOULD NOT
    impose any such access barriers for resources that would normally
    not require an access token.

    The '...' ellipses indicate that more properties may be present.
    Non-breaking examples that have been proposed so far, include a
    "http://tools.ietf.org/html/rfc6750#section-2.3" property, set to
    the string value "true" if the server supports passing the bearer
    token in the URI query parameter as per section 2.3 of [BEARER],
    instead of in the request header.

    Another example is "http://tools.ietf.org/html/rfc7233" with a
    string value of "GET" if Content-Range headers are supported for
    GET requests as per [RANGE].

    Both these proposals are non-breaking extensions, since the client
    will have a way to work around it if these features are not present
    (e.g. retrieve the protected resource asynchronously in the first
    case, or request the entire resource in the second case).

    A "http://remotestorage.io/spec/web-authoring" property has been

proposed with a string value of the fully qualified domain name to
which web authoring content is published if the server supports web
authoring as per [AUTHORING]. Note that this extension is a breaking
extension in the sense that it divides users into "haves", whose
remoteStorage accounts allow them to author web content, and
"have-nots", whose remoteStorage account does not support this
functionality.

The server MAY expire bearer tokens, and MAY require the user to
register applications as OAuth clients before first use; if no
client registration is required, the server MUST ignore the value of
the client_id parameter in favor of relying on the origin of the
redirect_uri parameter for unique client identification. See section
4 of [ORIGIN] for computing the origin.

11. Storage-first bearer token issuance

To request that the application connects to the user account
<account> ' ' <host>, providers MAY redirect to applications with a
'remotestorage' field in the URL fragment, with the user account as
value.

The appplication MUST make sure this request is intended by the
user. It SHOULD ask for confirmation from the user whether they want
to connect to the given provider account. After confirmation, it
SHOULD connect to the given provider account, as defined in Section
10.

If the 'remotestorage' field exists in the URL fragment, the
application SHOULD ignore any other parameters such as
'access_token' or 'state', to ensure compatibility with servers
that implement older versions of this specification.

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: *
        Content-Type: application/jrd+json

        {
          "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": "http://tools.ietf.org/id/draft-dejong-remotestorag\
e",
            "properties": {
              "http://remotestorage.io/spec/version": "draft-dejong-re\
motestorage-08",
              "http://tools.ietf.org/html/rfc6749#section-4.2": "https\
://3pp.io:4439/oauth/michiel",
              "http://tools.ietf.org/html/rfc6750#section-2.3": null,
              "http://tools.ietf.org/html/rfc7233": null,
              "http://remotestorage.io/spec/web-authoring": null
            }
          }]
        }
```

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 account'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&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/1.1 302 Found
        Location: https://drinks-unhosted.5apps.com/#access_token=j2YnG\
            tXjzzzHNjkd1CJxoQubA1o%3D&token_type=bearer

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"
```

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

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

Or a 200 OK status, plus a response body:

```
     HTTP/1.1 200 OK
     Access-Control-Allow-Origin: https://drinks-unhosted.5apps.com
     Content-Type: application/json; charset=UTF-8
     Content-Length: 106
```

```
        ETag: "1382694048000"
        Cache-Control: no-cache

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

   If the GET URL would have been "/storage/michiel/myfavoritedrinks/",
   a 200 OK response would have a folder description as the response
   body:

```
        HTTP/1.1 200 OK
        Access-Control-Allow-Origin: https://drinks-unhosted.5apps.com
        Content-Type: application/ld+json
        Content-Length: 171
        ETag: "1382694048000"
        Cache-Control: no-cache

        {"@context":"http://remotestorage.io/spec/folder-version","ite\
ms":{"test":{"ETag":"1382694048000","Content-Type":"application/json; \
charset=UTF-8","Content-Length":106}}}
```

   If the GET URL would have been a non-existing document like
   "/storage/michiel/myfavoritedrinks/x", the response would have a 404
   Not Found status, and no ETag header:

```
        HTTP/1.1 404 Not Found
        Access-Control-Allow-Origin: https://drinks-unhosted.5apps.com
```

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
   folder of each scope they have access to, to see if the version
   of the root folder 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 folder 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 folder'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 account 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 revocation 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 [CAPABILITIES] 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 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"
   * "http://tools.ietf.org/html/rfc7233"
   * "http://remotestorage.io/spec/web-authoring"

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, Markus Lanthaler,
and Markus Unterwaditzer, 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.

     [URI]
          Fielding, R., "Uniform Resource Identifier (URI): Generic
          Syntax", RFC 3986, 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.

     [ORIGIN]
          "Section 4: Origin of a URI", in: Barth, A., "The Web Origin
          Concept", RFC6454, December 2011.

17.2. Informative References

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

     [HTTP]
          Fielding et al., "Hypertext Transfer Protocol (HTTP/1.1):
          Semantics and Content", RFC7231, June 2014.

     [COND]
          Fielding et al., "Hypertext Transfer Protocol (HTTP/1.1):
          Conditional Requests", RFC7232, June 2014.

     [RANGE]
          Fielding et al., "Hypertext Transfer Protocol (HTTP/1.1):
          Conditional Requests", RFC7233, June 2014.

     [HTTP/2]
          M. Belshe, R. Peon, M. Thomson, Ed. "Hypertext Transfer Protocol
          Version 2 (HTTP/2)", RFC7540, May 2015.

     [JSON-LD]
          M. Sporny, G. Kellogg, M. Lanthaler, "JSON-LD 1.0", W3C
          Proposed Recommendation,

          http://www.w3.org/TR/2014/REC-json-ld-20140116/, January 2014.

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

     [KERBEROS]
          C. Neuman et al., "The Kerberos Network Authentication Service
          (V5)", RFC4120, July 2005.

     [BEARER]
          M. Jones, D. Hardt, "The OAuth 2.0 Authorization Framework:
          Bearer Token Usage", RFC6750, October 2012.

     [AUTHORING]
          "Using remoteStorage for web authoring", reSite wiki, retrieved
          September 2014. https://github.com/michielbdejong/resite/wiki
          /Using-remoteStorage-for-web-authoring

     [CAPABILITIES]
          J. Tennison (ed.), "Good Practices for Capability URLs",
          http://www.w3.org/TR/capability-urls/, February 2014.

18. Authors' addresses

     Michiel B. de Jong
     (independent)

     Email: michiel@unhosted.org

     F. Kooman
     (independent)

     Email: fkooman@tuxed.net