TSD API integration

Contents

Generated on: 2019-09-20

Overview

For questions about integration please contact: `tsd-drift@usit.uio.no`.

Production environment

The production API is available at `https://api.tsd.usit.no`. TSD has a test project named `p11` against which apps can be developed. Note that this project is not intended to store sensitive data of any kind.

Client registration

Signup

To register POST your email address and a cient name. The email address should be one that can be used by TSD to reach the administrator of the client in question. The combination of email + client_name should be unique. *Client names should be ASCII and not contain spaces.*

```
POST /v1/p11/auth/basic/signup

{
    "email": "your.email@whatever.com",
    "client_name": "test1"
}
```

This will return the status of your client and a persistent client ID.

Confirmation

Next you need to request this confirmation token from TSD. Use the contact email provided above. This token lasts 24 hours so you need to complete this step before it times out. To confirm the validity of the provided email address, and to obtain a persistent API password POST your client_id and the confirmation token to the API:

```
POST /v1/p11/auth/basic/confirm

{
    "client_id": <your_id>,
    "token": <confirmation_token>
}
```

This will return a client password. At this point your client still has to be verified by TSD before you can use your ID and password to request a persistent API key. Contact tsd-drift about this.

Getting an API key

Once verified you can request an API key:

```
POST /v1/p11/auth/basic/api_key

{
    "client_id": <some_id>,
    "pass": <your_password>
}
```

This will return a long-lived API key: `{ "api_key": <key> }`. This key lasts for a year, and times out automatically. It is your responsibility to request a new one using your client_id and password, and to distribute it to your application in time. If you forget your application will not be able to request access tokens, no matter which method you use. When you request a new one before your current key expires, both will remain active while you update your configuration. It is recommended to revoke the old key once it is not in use anymore. The key is just a JWT, so you can decode it in your application to see if it has expired.

You use this API key as a bearer token in the header when getting short-lived access tokens, so the TSD API can authenticate your application. You can only request five long-lived API keys in total, so do not lose it. If you by some unfortunate accident lose it, then contact TSD for help. If you abuse the API then this token will be revoked.

You can revoke an existing API key in the following way:

```
DELETE /v1/p11/auth/basic/api_key

{
    "client_id": <some_id>,
    "pass": <some_password>,
    "api_key": <key>
}
```

Client password reset

To request a new password you still need access to your current one. Otherwise your client has to be registered again. Using your password you can request a new one as such:

```
POST /v1/p11/auth/basic/reset_password

{
    "client_id": <some_id>,
    "pass": <some_password>
}
```

A summary of the sign-up flow is as follows:

```
    client                                      API
    ------                                      ---
    client_name, email          -------------->  /auth/basic/signup
                                <--------------  client_id
                                <--------------  confirmation_token (via email)
    client_id, confirmation_token -------------->  /auth/basic/confirm
                                <--------------  password
                                                 (details verified by TSD)
    client_id, password         -------------->  /auth/basic/api_key
                                <--------------  api_key
```

During the confirmation step you will be in ccontact with staff from TSD who will assess your use case and grant your application the right to use one or more authentication methods for one or more TSD projects. The API supports two authentication methods: basic authentication, TSD Two Factor Authentication.

Authentication and Authorization

Authorization

TSD projects have three defualt groups controlling data access rights: `import`, `export`, and `admin` groups. The API issues access tokens for each of these groups, depending on whether the identity of the authenticator is a member of these groups.

To import data you need to be a member of the `import` group, and the API client has to request an import token `?type=import`. To export data, and export token should be requested `?type=export`. You can implement your own privileged behaviour and require the admin token for that `?type=admin`.

Authentication

Basic authentication

Using just the API key, you can get a short-lived access token that will allow your application to import data to TSD, for the project which you have authorization.

```
POST /v1/p11/auth/basic/token
Authorization: Bearer $apikey
```

Because basic authentication has a low level of assurance, we require that your app runs on a host with a fixed IP address/range. Apps using basic authentication cannot export any data.

TSD auth

To get a token:

```
POST /v1/p11/auth/tsd/token?type=<type>
Authorization: Bearer $apikey

{
    "user_name":"p11-test",
    "otp":"453627",
    "password": "dhfbjhb"
}
```

File import and export

Simple file upload and download

To stream file contents:

```
PUT /v1/p11/files/stream/filename?group=p11-data-group
Authorization: Bearer $import_token
```

Notice the `group` parameter in the URI. The value of this specifies which file group should have access to the file once it is uploaded to TSD. Access tokens issued by the auth API contain a claim listing the identity's group memberships. Clients can therefore set the value of the group parameter to any one of these groups.

To get a list of all files available for export:

```
GET /v1/p11/files/export
Authorization: Bearer $export_token
```

To export a file:

```
GET https://test.api.tsd.usit.no/v1/p11/files/export/myfile
Authorization: Bearer $export_token
```

Resumable file upload

A reference client implementation using the resumable import and export API can be found online.

HTTP method overview:

```
GET /files/resumables
GET /files/resumables/filename?id=<UUID>
PATCH /files/stream/file?chunk=<chunknum,end>&id=<UUID>&group=<group-name>
DELETE /files/resumables/filename?id=<UUID>
```

Starting a new resumable upload

The client, having chunked the file, starts by initiating a PATCH, uploading the first chunk:

```
PATCH /files/stream/filename?chunk=<num>&group=<group-name>
Authorization: Bearer $import_token

{
    filename: str,
    max_chunk: int,
    id: uuid
}
```

Using the UUID returned by the server in the response, the client can continue sending succesive chunks, in sequence:

```
PATCH /files/stream/filename?chunk=<num>&id=<UUID>&group=<group-name>
Authorization: Bearer $import_token

{
    filename: str,
    max_chunk: int,
    id: uuid
}
```

Resuming prior uploads

GET requests provide information necessary to resume a file upload.

Firstly, to list all resumables for the authenticated user:

```
GET /files/resumables
Authorization: Bearer $import_token

{
    resumables: [
        {
            filename: str,
            max_chunk: int,
            chunk_size: int,
            id: uuid,
            pevious_offset: int,
            next_offset: <int,'end'>,
            md5sum: str,
            warning: str,
            group: str
        },
        {...}
    ]
}
```

Secondly, the client can optionally specify a given filename without an upload id, and the server will return the resumable with the most data on the server (if there is more than one):

```
GET /files/resumables/myfile
Authorization: Bearer $import_token
```

```
{
    resumables: [
        {...},
        {...}
    ]
}
```

And lastly, the information for a speific upload can be requested by including the uplooad id in addition to the filename:

```
GET /files/resumables/myfile?id=<UUID>
Authorization: Bearer $import_token

{
    filename: str,
    max_chunk: int,
    chunk_size: int,
    id: uuid,
    pevious_offset: int,
    next_offset: <int,'end'>,
    md5sum: str
    warning: str,
    group: str
}
```

In this way, the GET endpoints provide the client a way to either discover previous uploads which can be resumed, or to get direct information.

Each resumable upload has: - a filename - a chunk number - a chunk size - a UUID - previous offset (number of bytes sent so far minus the last chunk size) - next offset (number of bytes sent so far, or an instruction to 'end' the sequence) - chunk md5 - a warning message, for if data is inconsistent - the group which will own the upload (can be used for granular access)

The combination of the filename and UUID allow the client to resume an upload of a specific file for a specific prior request. The chunk size and number allow the client to seek locally in the file before sending more chunks to the server, avoiding sending the same data more than once. The md5 digest of the latest chunk, combined with the offset information allow clients to verify chunk integrity.

The server will attempt to repair any data inconsistencies which may have arised due to server crashes or filesystem issues. If it cannot get the resumable data back into a consistent state, the `next_offset` field will be set to `end`. Client are recommended to either end the upload, or delete it.

Assuming data is consistent, the client then proceeds as follows:

```
PATCH /files/stream/filename?chunk=<num>?id=<UUID>&group=<group-name>
Authorization: Bearer $import_token

{
    filename: str,
    max_chunk: int,
    id: uuid
}
```

Completing a resumable upload

To finish the upload the client must explicitly indicate that the upload is finished by sending an empty request as such:

```
PATCH /files/stream/filename?chunk=end&id=<UUID>&group=<group-name>
Authorization: Bearer $import_token
```

This will tell the server to assemble the final file. Setting the group is optional as normal.

Cancelling a resumable upload

To avoid wasting disk space, partially completed uploads which were not resumed to completion, and abandoned, can be removed as such:

```
DELETE /files/resumables/filename?id=<UUID>
Authorization: Bearer $import_token
```

Implementation

Server

When a new resumable request is made, the server generates a new UUID, and creates a directory with the name of that UUID which will contain the successive chunks, and writes each chunk to its own file in that directory, e.g.:

```
/cb65e4f4-f2f9-4f38-aab6-78c74a8963eb
    /filename.txt.chunk.1
    /filename.txt.chunk.2
    /filename.txt.chunk.3
```

Once the client has sent the final chunk in the sequence, the server will merge the chunks, move the merged file to its final destination, remove the chunks, their accumulating directory, and respond to the client that the upload is complete.

Clients

Client are expected to split files into chunks, and upload each one as a separate request, *in order*. Since the server will return information about chunks, the client does not have to keep state if and when a resumable file upload fails, but it can if it wants to, since each request return enough information to resume the upload in the event of failure.

If a resumable upload fails and the client has lost track of the upload id, the client can, before initiating a new resumable request for a file, ask the server whether there is a resumable for the given file. If so, it will recieve the chunk size and sequence numner, and the UUID which identifies the upload. Using this, the given file upload can be resumed. The client chunks the file, seeks to the relevant part, and continues the upload.

When uploading the last chunk, the client must explicitly indicate that it is the last part of the sequence.

Resumable downloads

Or, how to perform conditional range requests, per file.

Starting a resumable download

Clients can get resource information before starting a download as follows:

```
HEAD /files/export/filename
Authorization: Bearer $export_token
```

The server will return an `Etag` header, containing an ID which uniquely identifies the resource content. Addtionally, the server will return the `Content-Length` in bytes. Clients can store the `Etag` to make sure that if they resume a download, they can check with the server that the resource has not changed in the meantime.

Downloads are started as usual:

```
GET /files/export/filename
Authorization: Bearer $export_token
```

Resuming a partially complete download

If a download is paused or fails before completing, the client can count the number of bytes in the local partial download, and request the rest from the server, using the `Range` header. *Importantly, range requests specify ranges with a 0-based index. So if the client already has 103 bytes of a file (bytes 0-102, with an 0-based index), and it wants the rest, then it should ask for*:

```
GET /files/export/filename
Range: bytes=103-
Authorization: Bearer $export_token
```

A specific index range can also be requested, if relevant:

```
GET /files/export/filename
Range: bytes=104-200
Authorization: Bearer $export_token
```

And to ensure resource integrity it is recommended that the value of the `Etag` ias included, thereby performing a conditional range request:

```
GET /files/export/filename
If-Range: 0g04d6de2ecd9d1d1895e2086c8785f1
Range: bytes=104-
Authorization: Bearer $export_token
```

The server will then only send the requested range if the resource has not been modified. Multipart range requests are not supported.

Generic JSON API

Overview

The generic JSON API gives clients the ability to store arbitrary JSON data and associated metadata. There are no restrictions on the structure of the data. The API provides a query language which client can use to filter data based on keys and values. Clients can also ask the API to send data formatted as CSV. The sections below elaborate on these capabilities.

The generic JSON API is available from outside of TSD at `api.tsd.usit.no`, and inside TSD from project VMs at `internal.api.tsd.usit.no`. Externally, data export requires `export` token types, while internally `member` token types are sufficient. In both cases `admin` tokens are required for DELETE requests.

Endpoints and methods

For working with data:

```
GET /v1/{pnum}/tables/generic
GET /v1/{pnum}/tables/generic/{table_name}
PUT /v1/{pnum}/tables/generic/{table_name}
PATCH /v1/{pnum}/tables/generic/{table_name}
DELETE /v1/{pnum}/tables/generic/{table_name}
```

And for metadata:

```
GET  /v1/{pnum}/tables/generic/metadata/{table_name}
PUT /v1/{pnum}/tables/generic/metadata/{table_name}
PATCH /v1/{pnum}/tables/generic/metadata/{table_name}
DELETE /v1/{pnum}/tables/generic/metadata/{table_name}
```

Querying

When making HTTP request for data, clients can filter data returned by the API by choosing which keys they want, conditional on the values of a combination of keys, much like SQL queries on tables. Some examples are:

```
GET /v1/{pnum}/tables/generic/{table_name}?select=col1,col2&col3=eq.5&col2=not.is.null&order=col1.desc
PATCH /v1/{pnum}/tables/generic/{table_name}?set=col1.5&col3=eq.5
DELETE /v1/{pnum}/tables/generic/{table_name}?col3=eq.5
```

Formatting

To get results as csv, simply add the `Accept: text/csv` header to your request.

Example

Storing and getting data in a table:

```
PUT /v1/p11/tables/generic/mytable
Authorization: Bearer $import_token

{
    "key1": 6,
    "key2": 7
}

PUT /v1/p11/tables/generic/mytable
Authorization: Bearer $import_token

{
    "key3": 60,
    "key4": 79,
    "key5": "value1"
}

GET /v1/p11/tables/generic/mytable
Authorization: Bearer $export_token

{
    "data": [
        {
            "key1": 6,
            "key2": 7
        },
        {
            "key3": 60,
            "key4": 79,
            "key5": "value1"
        }
    ]
}

GET /v1/p11/tables/generic/mytable
Authorization: Bearer $export_token
Accept: text/csv

key1,key2,key3,key4,key5
6,7,,,
,,60,79,value1
```

PostgreSQL JSON API

This API is not available for TSD projects by default, so please contact tsd-drift about setting it up.

Storing JSON data

The storage API is responsible for marshalling JSON data into a database in a TSD project area.

Conventions

For variable names use utf-8, camelcase, no spaces, no special characters like `;,.'|~` etc. So, this is a good variable name: `national_id_number` while this is a bad one: `National-id.number;`. The maximum permitted length for variable names is 63 characters.

Creating tables

Tables can also be created using a generic definition. The following postgres data types are supported:

`int, int[], text, text[], json, jsonb, real, date, timestamp, timestamptz, boolean, cidr, inet, interval, macaddr, decimal, serial, time, timetz, xml, uuid, bytea`

Clients can also specify which column should be a primary key and whether columns should have `not null` constraints.

```
POST v1/p11/storage/rpc/create_table
Authorization: Bearer $import_token

{
    "definition": {
        "table_name": "mytable",
        "columns": [
            {"name": "col1", "type": "int", "constraints": {"primary_key": true}},
            {"name": "col2", "type": "text[]", "constraints": {"not_null": true}},
            {"name": "col3", "type": "serial"} ]
        },
    "type": "generic"
}
```

Sending new definitions which refer to the same table will add columns if they were not present, but never delete existing ones.

Retrieving JSON data

The retrieval API makes it possible to get access to project data *outside* of TSD. Tables created by the storage API are not available for export by default.

Getting data as JSON

To GET data from a published table, for example:

```
GET v1/p11/retrieval/example_report
Authorization: Bearer $export_token
```

A more interesting query, selecting report data where `x > 2 and y = 'yes'`, for example, would be:

```
GET /v1/p11/retrieval/example_report?select=y,x,z&x=gte.2&y=eq.yes
Authorization: Bearer $export_token
```

Getting data as CSV

Data can be retrieved in CSV format instead of JSON by passing `Accept: text/csv` as a header in the GET request.

Query capabilities

Both the storage and retrieval APIs use postgrest as an application server which has a rich API for querying data. You can read about it on the project website.