WORKING WITH THE REST ANNOTATION FRAMEWORK
Version: 1.0/draft version
Last updated: 26th november 2013, 18th december 2013
Author : LIMSI

I) Overview

The rest annotation framework, introduced in the deliverable, allows you to work with the 3M data. The framework provides you with a set of APIs that your application can interact with the 3M data through making http requests. Note that, this document supposes that you are working on localhost server. In the case you are working with the server flower.limsi.fr (provided by the project), you have to replace «localhost :3000» by flower.limsi.fr/data/, and then everything should work properly.

II) Getting started with the framework

You should be familiar with basic javascript concepts, as well as with the javascript object notation (json) data format. The core of the framework is implemented on node.js using express framework. All data are stored in mongodb using json format.

We built an access control list (ACL), which is a list of access control entries (ACE) attached to resources, for user-access management. Each ACE specifies which users or groups are granted access rights to a resource (corpus, media, layer, annotation), as well as what operations are allowed on the given resource. For instance, a corpus contains an entry ACE (Peter, D), meaning that Peter has permission to delete this corpus. Each user or group can be granted one of the five permissions: None (N), Read (R), Create (C), Delete (D), and Admin (A). When finding the exact right for a connected user on a given resource, we first check the user rights, then the group rights on that resource. If it is not found, we check the rights of the connected user in a higher level of resources, i.e., in the following order : annotation < layer < media < corpus, until we can determine the exact right of the user for the resource considered. If you have an A right on a given resource, you can assign rights for other users or groups to access that resource (see next section).

In this document, we will learn how to:

- login to the API framework
- logout
- create an ACL (access control list) for our authenticated application
- create a corpus/media/layer/annotation
- update a corpus/media/layer/annotation
- delete a corpus/media/layer/annotation
- get a resource, e.g, corpus/media/layer/annotation

II.1) Preparation

We use Node.js (npm is also installed as a node module) and Express (which is the middleware used to create the web server on the Node.js environment) to build the API web services. To start using the APIs, you need to do the following:

- Install Node.js and MongoDB if they're not already installed (just download them from their corresponding websites, and follow the corresponding installation instructions)
- Clone the camomile-node.js-rest-api repository into a new directory:
- $> git `clone ssh://git@git.renater.fr:2222/cmml-annotation.git`
- Go in the directory in which you cloned the camomile project, do the following commands to install the dependencies:
  - $> cd cmml-annotation
  - $> npm install

Now it's time to discover our APIs.

II.2) Using the APIs

Let's us recall some supported APIs built for the Camomile project:

- GET /corpus: returns all available corpus
- GET /corpus/:id : returns the corpus identified by the given :id
- GET /corpus/:id_corpus/media : returns all media of the corpus identified by the given id id_corpus
- GET /corpus/:id_corpus/media/:id_media : returns the media identified by the given :id_media
- GET /corpus/:id_corpus/media/:id_media/layer : returns all layers of the media identified by the given :id_media
- GET /corpus/:id_corpus/media/:id_media/layer/:id_layer : returns the layer identified by the given :id_layer
- GET /corpus/:id_corpus/media/:id_media/layer/:id_layer/annotation : returns all annotations of the layer identified by the given :id_layer
- GET /corpus/:id_corpus/media/:id_media/layer/:id_layer/annotation/:id_anno: returns the annotaion identified by the given :id_anno
- POST /corpus: creates a new corpus corresponding to the JSON object given in the body of the HTTP request
- POST /corpus/:id/media : creates a new media (corresponding to the JSON object given in the body of the HTTP request), under the corpus id
- POST /corpus/:id_corpus/media/:id_media/layer : creates a new layer, under the media id_media
- POST /corpus/:id_corpus/media/:id_media/layer/:id_layer/annotation: creates a new annotation, under the layer id_layer
- PUT /corpus/:id : updates the corpus with values of the JSON object given in the body of the HTTP request. Other resources can be updated in a similar way.
- DELETE /corpus/:id : deletes the corpus with the given id. Other resources can be deleted in a similar way.

To start testing our APIs, open two different terminal sessions, and type on each one:

```
$> mongod
$> node appCam.js
```

If everything is ok, you will see the following message:

```
Express server listening on port 3000 Connected to Mongoose
Connected to Mongoose:
Connect mongodb session success...
```

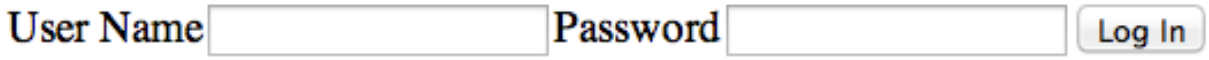
It means that your node js server is running and listening on the port 3000, and you have successfully connected to the Mongodb database. Now you can use the APIs of the framework. These APIs are usually invoked in a client application through jQuery or Ajax calls.

1) Admin pannel

   a) Login

You can now connect to the API services by typing on your browser:
http://localhost:3000/

You will see a welcome message, and below it a link to login (log in) to the APIs. Click on that link, or type http://localhost:3000/login on your browser, to login:



In the appeared form, enter your username and password. Note that, you have to contact the cordinator or the Webmaster of the project to receive an account. If you entered a valid account, you will receive the following message:

```
You have been successfully logged in
```

Alternatively, you can make a post to login:

```
curl -i -X POST http://localhost:3000/login --data '{"username":"ta", "pass":"****"}' -
H "Content-Type: application/json"
```

And now you can access the APIs.

   b) Logout

To logout, run (or make a GET request to):

```
http://localhost:3000/logout
```

a) Create a user
To create an account, login as root user and make a post like:

curl -i -X POST http://localhost:3000/signup --data '{"username":"ta", "pass":"****", "affiliation": "LIMSI", "role" : "user"}' -H "Content-Type: application/json"

Where username, pass, affiliation, and role are the required fields that need to be filled in.

You can also login as a root account and simply type:

http://localhost:3000/signup

Each account has a role, which can be either user (normal) or admin (root). An admin (root) account can access all resources, while a user role has less permissions, which are resource-dependent.

d) Change the role of an existing user

The role of the created user is assigned to "user" by default, but you can then change it by making a post operation, for example:

curl -i -X POST http://localhost:3000/chmodUser --data '{"username":"ta", "role":"admin"}' -H "Content-Type: application/json"

Alternatively, you can use a window form to enter the username and its new role, for instance:

http://localhost:3000/chmodUser

f) Remove a user

Login as an admin account, you can then remove a user through its loginname by:

$ curl -i -X DELETE http://localhost:3000/removeUserByname --data '{"username":"ta"}' -H "Content-Type: application/json"

g) Create a group

Login as an admin account and make a post request, for example:

curl -i -X POST http://localhost:3000/Group --data '{"groupname":"CRP", "description":"People at CRP"}' -H "Content-Type: application/json"

As an alternative way, you can do so by filling up a form, which can be obtained from a GET request:

http://localhost:3000/addGroup

h) Remove a group

Login as an admin user, and then perform a delete operation, e.g.:

```
$ curl -i -X DELETE http://localhost:3000/removeGroupByName --data
'{"groupname":"CRP"}' -H "Content-Type: application/json"
```

i) Add a user to a group

Login as an admin user, and perform a post request like:

```
curl -i -X POST http://localhost:3000/addUser2Group --data '{"groupname":"CRP",
"username":"bredin"}' -H "Content-Type: application/json"
```

Alternatively, you can fill in a form by making the following GET:

http://localhost:3000/addUser2Group

j) Get all groups

To list all available groups, login as an admin account and type (or make a GET request to):

http://localhost:3000/allGroups

h) Get all users

To list all available users, login as an admin account and type (or make a GET request to):

http://localhost:3000/allUsers

2) Working with the data (frequently used APIs)

In this section, you will learn how to manipulate with the 3M data through a set of APIs. The following guide supposes that you are connected to the framework (see the above section).

a) Create a corpus

Let's add a new corpus, namely repere, by using curl, for example :

curl -i -x post http://localhost:3000/corpus --data '{"name":"repere"}' -h "content-type: application/json"

Note that, only admin accounts are allowed to create new corpus.

b) Retrieve corpus

To retrieve all available corpus, make a Get operation, e.g :

curl -i http://localhost:3000/corpus

It will return all corpus, on which you have at least a R (read) access right. The output looks like :

```
[
        {
                "name": "Repere",
                "_id": "5223404ef22f9e1305000001"
        }
]
```

If you want to get a specific corpus with a given id, you make:

curl -i http://localhost:3000/corpus/5223404ef22f9e1305000001

c) Create a new media

Once you have created a corpus, you can insert media into the corpus. The following example shows how to insert a new media to the corpus Repere:

```
curl -i -X POST http://localhost:3000/corpus/5223404ef22f9e1305000001/media --data
'{"id_corpus" : "5223404ef22f9e1305000001","name":"Video 01"}' -H "Content-Type:
application/json"
```

d) Create new layers or annotations

In a similar way, you can add a layer or an annotation. You have to send correct post requests to the server, i.e., requests with correct data types and parameters (see the spec of the project).

e) Update a corpus/media/layer/annotation

Let's update the name of the corpus by setting its value to "Repere train phase1" (its current name is "Repere"):

```
curl -i -X PUT http://localhost:3000/corpus/5223404ef22f9e1305000001 --data
'{"name":"Repere train phase1"}' -H "Content-Type: application/json"
```

For resources having several fields, e.g., layers or annotations, we can indicate only the fields that you want to update.

f) Delete a corpus/media/layer/annotation

Note that, if you delete a corpus, all resources belonging to this corpus will also be deleted. Here is an example showing the deletion of a media:

```
Curl -i -X DELETE

http://localhost:3000/corpus/5223404ef22f9e1305000001/media/52234d74f22f9e1305000002
```

In the case of success, you will recieve a list of ids of the resources which have been deleted. Otherwise, it returns an error message.

3) Working with acl

This section describes how to use the authentication of the api-tool.

a) View all acls

Login as an admin user, and type:

http://localhost:3000/allACLs

It will return a list of ACL entries.

b) Retrieve ACLs of a resource (a corpus, a medium, a layer, an annotation):

Type the following command (only users which have an admin rights on the requested resource can view or modify its content) to check ACLs of a specific corpus:

http://localhost:3000/corpus/id/acl

For retrieving ACLs of media:

http://localhost:3000/corpus/id_corpus/media/id_media/acl

For getting ACLs of a layer:

http://localhost:3000/corpus/id_corpus/media/id_media/layer/id_layer/acl

And for an annnotation:

http://localhost:3000/corpus/id_corpus/media/id_media/layer/id_layer/annotation/id_anno/acl

c) Update the right (ACL) of a resource
You can either add a username right or a groupname right to a resource. If the username or groupname being updated exists in the resource, its corresponding new right will be updated.

To update a username and its right for a specific resource id, execute the following command:

curl -i -X PUT http://localhost:3000/corpus/id/acl --data '{"username":"ta", "userright" : "C"}' -H "Content-Type: application/json"

Or for a group:

curl -i -X PUT http://localhost:3000/corpus/id/acl --data '{"groupname":"CRP", "groupright" : "A"}' -H "Content-Type: application/json"

Similarly, we can update a username right of a media, a layer, or an annotation:

curl -i -X PUT http://localhost:3000/corpus/id_corpus/media/id_media/acl --data '{"username":"ta", "userright" : "C"}' -H "Content-Type: application/json"

curl -i -X PUT http://localhost:3000/corpus/id_corpus/media/id_media/layer/id_layer/acl --data '{"username":"ta", "userright" : "C"}' -H "Content-Type: application/json"

curl -i -X PUT http://localhost:3000/corpus/id_corpus/media/id_media/layer/id_layer/annotation/id_anno/acl --data '{"username":"ta", "userright" : "C"}' -H "Content-Type: application/json"

Note that, an ACL entry specified by the name of the connected user (as username) and an A (admin) right (as userright) is automatically added into the ACL list whenever a new resource is created. In the case of deletion, when a resource is deleted, all its ACL entries will be automatically removed.