VOER PLATFORM

VPC - API SERVICE SPECIFICATIONS

CHANGE RECORDS

1.0	Ha Pham	10 Nov 2012	Document created
1.1	Ha Pham	05 Jan 2013	Complete calls of Category, Author, Editor, Module
1.2	Ha Pham	10 Jan 2013	Change all appearances of Module to Material
1.3	Ha Pham	17 Jan 2013	Update the Material API
1.4	Ha Pham	29 Jun 2013	Update API of material PDF, material files
1.5	~	05 Nov 2013	Update Export flow and API for update material
1.6	~	12 Dec 2013	Add material extra information
1.7	~	17 Dec 2013	Add facet search counts
1.8	~	13 Jan 2014	Add favorites by person
1.9	~	27 Mar 2014	Add get linked collections of given module
2.0	~	17 May 2014	Add specs for getting top most-viewed, favorited, and rated materials
2.1	~	22 May 2014	Adjust result of get top ratings

1. VOER Platform Core API service:

API provided by VOER Platform Core (VPC) will adapt RESTful model, all requests will be made by calling specified URLs with corresponding parameters.

Request: Basic REST calls are made by some kinds of request:

POST	create
PUT	update
HEAD	get metadata
GET	read
DELETE	delete

Sample:

```
GET    http://api.voer.edu.vn/0.9/modules/0213123
```

# above call request the module #0213123, with API version 0.9

Response: Most of data format of responses are in JSON, other could be media file (VDP), this depends on the basis of request and returned information.

Sample:

```
200    OK
{
       "firstName": "John",
       "lastName": "Smith",
       "age": 25,
       "address": {
              "streetAddress": "21 2nd Street",
              "city": "New York",
              "state": "NY",
              "postalCode": 10021
       },
}
```

2. API Versioning

- API will also be versioned like the other components, or web site systems,...
- API calls need to specify the API version inside requested URL. Inside this example:

```
GET    http://api.voer.edu.vn/0.9/modules/0213123
```

# 0.9 is the API version of called command.

- There will not have a mark for latest version in API calls. Any call to API must specify the API version clearly. The reason is for reducing unexpected issues with running client systems.

3. API Content

API server serves information exchanging which relates to:

+ API Service
+ Authors
+ Editors
+ Categories
+ Materials

4. Authentication

All authentication steps will be performed under HTTPS.

1. (HTTPS) client application (CA) sends the client ID to the API server, URL:

   Request:

   ```
   POST   $URL/auth/<client_id>
   sugar = hello_im_sugar
   comb = md5(<client_key>+<sugar>)
   ```

   the comb argument is MD5 of concatenation of client (secret) key and the sugar argument. Argument sand is freely generated by the CA itself, and it cannot be blank.
2. (HTTPS) API server validates the received combined key then provides token if valid, or returns error message if not. Token has a predefined life-time, like, about 15 minutes.

   Response:

   ```
   200    OK
   {
          "client_id": <client_id>,
          "token": "asKJHSA&^29812123",
          "expires": "2013/01/123 13:12"
          "result": "ok",
          "error_message": "nothing",
   }
   ---
   406    NOT ACCEPTABLE
   {
          "detail": "Authentication failed (invalid combination)"
   }
   ```
3. CA uses the provided token inside every requests for a limited time.

~~* (HTTPS) CA can call refresh token when it's about to expired.~~

~~**Request:**~~

~~POST $URL/auth/<client_id>?refresh=1~~
~~token = asKJHSA&^29812123~~

~~**Response:**~~

~~200   OK~~

```
~~{~~
        ~~"token": "asKJHSA&^29812123",~~
        ~~"expires": "2013/01/123 13:12"~~
        ~~"result": "ok",~~
        ~~"error_message": "nothing",~~
~~}~~
```
* On server side, all requests with valid token will before putting onto API queue
* Token & client IP will be compared with stored db of API server
* When token is generated after validating request (new or refresh), the client IP will be recorded and saved along with token.

5. API Calls

Now we generalize calling addresses to $URL which contain API address and version, e.g.: http://api.voer.edu.vn/1.1/

List of API calls:

- Add new material
- Get material metadata
- Download material content
- Check-in material
- Search for materials (get list), authors, keywords, ...
- Delete material (all or specific version)
- Get all updates since a given time
- Get category information
- Authenticate
- Refresh token

Considering:
- Statistic log APIs
- parent and children of a given material, API
- Which collection/VDP/material refers to a given material?

Materials, categories, or other content could be retrieved and added separately. But to have a rich-info material content, we encourage users to add missing items (categories, authors...) before issueing calls for creating materials of dependant content.

Validate API requests:

At beginning of each content API call, provided token & client_id will be check to ensure the request is from an authenticated client.

Values of token and client_id need to be put as COOKIES or GET variables, with names:
vpr_token: Valid token value

vpr_client: Valid API client ID

If the request cannot pass the token check, a error response will be returned, somehow like this:

```
HTTP 401 UNAUTHORIZED
{
    "details": "Permission denied due to invalid API token"
}
```

5.1 Categories:

A category content has basic information like below:

id	NUMBER	Unique ID of the category
name	TEXT	Name of the category
parent	NUMBER	ID of the parent category
description	TEXT	Description of the category

Listing and retrieving calls from normal clients are always accepted, while creating and deleting are only for users with manager/administrator role.

5.1.1 Listing categories:

Request:

```
GET    $URL/categories/
```

Response:

```
HTTP 200 OK
{
    "count": 2,
    "next": null,
    "previous": null,
    "results": [
        {
            "id": 1,
            "name": "Mathematic",
            "parent": 0,
            "description": "Just for calculation"
        },
        {
            "id": 2,
            "name": "Photography",
            "parent": 0,
            "description": "Entertainment"
        }
    ]
}
```

5.1.2 Create category:

Just using POST with data for category name, parent ID, and its description:

```
POST   $URL/categories/
name = 'Photography'
parent = 0
description = 'Capture the moments'
```

```
---
HTTP 201 CREATED
Content-Type: text/html
{
    "id": 3,
    "name": "Photography",
    "parent": 0,
    "description": "Capture the moments"
}
```

5.1.3 Retrieve category:

Using GET with category ID, all category info will be returned

```
GET    $URL/categories/3
---
HTTP 200 OK
{
    "id": 3,
    "name": "Mathematic",
    "parent": 0,
    "description": "Just for calculation"
}
```

In case of including material in each category returned, add *?count=1* into query

```
GET    $URL/categories/3?count=1
---
HTTP 200 OK
{
    "id": 3,
    "name": "Mathematic",
    "parent": 0,
    "description": "Just for calculation"
    "material": 123
}
```

5.1.4 Update category:

Using PUT method with category ID, all category info will be returned

```
PUT    $URL/categories/3
name = 'Mathematic'
parent = 0
description = 'Just for calculation and counting money'
---
HTTP 200 OK
{
    "id": 3,
    "name": "Mathematic",
    "parent": 0,
    "description": "Just for calculation and counting money"
}
```

5.1.5 Delete category:

Using DELETE method with category ID

```
DELETE $URL/categories/3
---
HTTP 204 NO CONTENT
```

5.2 Persons

Person is a general model for author, editor, or other roles. The model has below properties::

```
id            ID of the author record
fullname      Fullname of the author
user_id       ID of person at client system
first_name
last_name
email
title
homepage
affiliation
affiliation_url
national
biography     Some additional information of author
client_id
avatarImage for user avatar
```

Just like category, person has the same set of API calls.

5.2.1 Listing Authors

Result of listing call only shows compact infomation of persons. To retrieve complete person metadata, we need to call the specific person detail request.

```
GET    $URL/persons/
---
HTTP 200 OK
Content-Type: text/html
{
    "count": 2,
    "next": null,
    "previous": null,
    "results": [
        {
            "id": 1,
            "fullname": "Test Author",
            "user_id": "james_bond"
             ...
        },
        {
```

```
                 "id": 2,
                 "fullname": "Stephen King",
                 "user_id": "micky_mouse"
                 ...
            }
            ]
        }
```

5.2.2 Create New Person

```
POST   $URL/persons/
       fullname = 'JJK Tolkien'
       biography = 'Father of the Middle Earth'
       avatar = <image-file>
       ...
---
HTTP 201 CREATED
Content-Type: text/html
{
    "id": 3,
    "fullname": "JJK Tolkien",
    "biography": "Father of the Middle Earth"
       ...
}
```

5.2.3 Retrieve Person

```
GET    $URL/persons/3/
---
HTTP 200 OK
Content-Type: text/html
{
    "id": 1,
    "fullname": "JJK Tolkien",
    "biography": "Father of the Middle Earth"
       ...
}
```

To get count of collections/modules belong to this person, add **?count=1** to above URL

```
GET    $URL/persons/3?count=1
---
HTTP 200 OK
Content-Type: text/html
{
    "id": 1,
    "fullname": "JJK Tolkien",
    "biography": "Father of the Middle Earth"
    "author": {
           "collection": 0,
           "module": 3
    },
    "translator": {
           "collection": 0,
                "module": 0
            },
            ...
        }
```

To get the person avatar as an image, add /avatar behind the requested URL.

```
        GET    $URL/persons/3/avatar
```

To delete the avatar, just add ?delete=1 to above call:

```
        GET    $URL/persons/3/avatar?delete=1
```

To get list of favorited materials by a given person:

```
        GET    $URL/persons/3/favorites
        ---
        {
            "count": 21,
            "previous": null,
            "results": [
                {
                    "author": "8",
                    "title": "A module here",
                    "material_type": 1,
                    "material_id": "434ddc73",
                    "version": 1,
                    "categories": [
                        3
                    ]
                },
                {
                    "author": "91",
                    "title": "Another module here",
                    "material_type": 1,
                    "material_id": "64a5f170",
                    "version": 1,
                    "categories": [
                        1
                    ]
                },
                ...
                ],
            "next": "http://localhost:8000/1/persons/62/favorites?page=2"
        }
```

5.2.4 Update Person

```
        PUT    $URL/persons/3/
               fullname = 'JJK Tolkien'
               biography = 'Not Father of the Middle Earth.'
               …
        ---
        HTTP 200 OK
        Content-Type: text/html
        {
            "id": "1",
            "fullname": "JJK Tolkien",
            "biography": "Father of the Middle Earth."
              ...
        }
```

5.2.5 Delete Person

```
DELETE       $URL/persons/1/
---
HTTP 204 NO CONTENT
```

5.4 Materials

Material is the primary content object of VOER Platform, this contains a specific educational content and its metadata

material_id	ID of the material, auto generated
material_type	Type of the material
text	Body text of the material
version	version of current content
title	Material title
description	Material description
categories	IDs of belonging categories
author	IDs of material author(s)
editor	IDs of material editor(s)
contributor	IDs of material contributor(s)
licensor	IDs of material licensor(s)
translator	IDs of material translator(s)
maintainer	IDs of material maintainer(s)
keywords	Material keywords
image	Cover image of the material
language	Language code (en, vi,...) of material
license_id	ID of implemented license
modified	Time of the last modification
derived_from	ID of parent material

5.4.1 List Materials

This below is for getting all materials (having paging), this could be prevented due to ineffective when published.

```
GET    $URL/materials/
---
HTTP 200 OK
Content-Type: text/html
```

```
{
    "count": 1,
    "next": null,
    "previous": null,
    "results": [
        {
            "material_id": "754000ee1f7d6bc160ceb1505b376096",
            "text": "Hello",
            "version": "1",
             ...
        }
    ]
}
```

User also can get a list of specific material with all of its versions:

```
GET    $URL/materials/<material_id>/all/
---
HTTP 200 OK
. . .
```

5.4.2 Create Material

One of the new change of this model/type is having image field and ability of having multiple attachment files.

To add multiple attached files, just add those files into posted form as follow:

"attach01"	<file-01>
"attach01_name"	"File 01 name"
"attach01_description"	"File 01 description"
"attach02"	<file-02>
"attach02_name"	"File 02 name"
"attach02_description"	"File 02 description"
...	

```
POST   $URL/materials/
       material_type = 1,
       text = "Hello",
       version = "1",
       title = "Hello",
       description = "Hello",
       categories = "1, 2, 3",
       author = "3",
       editor = "1",
       licensor = "2",
       keywords = "mobile\ncomputer",
       image = <file 'abc'>,
       attach01 = <file 'hello-01'>,
       attach01_name = 'Additional 01 File',
       attach01_description = 'Description of first file',
       attach02 = <file 'hello-02'>,
       attach02_name = 'Additional 02 File',
       attach02_description = 'Description of first file',
       language = "en",
        license_id = 1,

       ---
       HTTP 201 CREATED
       Content-Type: text/html
       ...
```

5.4.3 Check In Material

User has to to specify the exact material (with version) to perform check in new content. Check in call using PUT method, with updated content in sent request.

```
PUT   $URL/materials/<material_id>/<version=latest>
---
HTTP 201 CREATED
Content-Type: text/html
Allow: GET, PUT, DELETE, HEAD, OPTIONS
{
    "material_id": "095a365194638330f5e67731d32f4296",
    "title": "No title",
    "text": "no text",
    "version": 2,
    "description": "some more changes",
    "categories": "1",
        ...
}
```

5.4.4 Browse Materials

VPC API also provides calls to get list of materials base on specific content, like: category, author, editor,... Those use the similar URL to listing materials in section 5.1.1.

By category:
```
GET    $URL/materials?categories=1,2,3
```
By author:
```
GET    $URL/materials?author=1,2,3
```
By editor:
```
GET    $URL/materials?editor=1,2,3
```
By language:
```
GET    $URL/materials?language=1,2,3
```
By modified type:
```
GET    $URL/materials?modified=1,2,3
```
By material type:
```
GET    $URL/materials?material_type=1,2,3
```
Browsed items could be combined in a single API call
```
GET    $URL/materials?categories=1&author=2
```

Returned results can be sorted by a given field

```
GET    $URL/materials?categories=1,2,3&sort_on=modified
GET    $URL/materials?categories=1,2,3&sort_on=-modified
GET    $URL/materials?categories=1,2,3&sort_on=-modified,author
---
HTTP 200 OK
Content-Type: text/html
Allow: GET, PUT, DELETE, HEAD, OPTIONS
Content-Type: text/html
{
    "count": 1,
    "next": null,
    "previous": null,
    "results": [
        {
            "material_id": "754000ee1f7d6bc160ceb1505b376096",
            "text": "Hello",
            "version": "1",
              ...
        }
    ]
}
```

Only with categories, user can select between OR or AND (default) conditions:
Search with AND condition:

```
GET    $URL/materials?categories=1,2,3
```

Search with OR condition:

```
GET    $URL/materials?categories=1,2,3&or=categories
```

5.4.5 Get Material Attached Files

Material attached files don't be included inside the returned body of the material. In order to get those files information, we have to make separate requests:

1. Get list of IDs of attached files:

```
GET    $URL/materials/<material_id>/<version=latest>/mfiles/
---
[2, 3,...]
```

2. Get specific information of attached file using material file ID:

```
GET    $URL/mfiles/<mfile_id>
---
{
    "id": 1,
    "material_id": "106ee38d5c7f81625477b4cff082c64d",
    "version": 1,
    "name": "Hello this is the first file",
    "description": "",
    "mfile": "mfiles/test-image_3.png",
    "mime_type": ""
}
```

3. Download material file:

```
GET    $URL/mfiles/<mfile_id>/get/
---
```

5.4.6 Download material PDF file

Material export (under PDF) is generated by other component, named VPT, and stored inside VPR storage. The first time when export file is not ready, a request will be made toward VPT for conversion order. Client side should continously browse the below URL in order to knowing the status of exporting progress.

```
GET    $URL/materials/<material_id>/<version>/pdf/
```

When the making PDF file is under progress, response is:

```
HTTP 102 PROCESSING
Content-Type: text/html
```

When the conversion is completed and PDF is ready for download, resonse is:

```
HTTP 200 SUCCESS
Content-Type: application/pdf
…
```

5.4.7 Get similar materials

In order to get similar materials, just use the request below, which will return list of maximum 10 IDs with similarity decreased.

```
GET    $URL/materials/<material_id>/<version>/similar/
---
HTTP 200 OK
Content-Type: text/html
[
       {
           "version": 1,
           "material_type": 1,
           "modified": "2013-08-22T10:46:58",
           "material_id": "12b0def6",
           "title": "H\u1ed3 Huron"
       },
       {
           "version": 1,
           "material_type": 1,
           "modified": "2013-08-22T11:15:56",
           "material_id": "6a2d625d",
           "title": "H\u1ed3 Superior"
       },
       …
]
```

5.4.8 Update material

This feature could be enabled only for a moment or limited to some specific users or clients. It allow stored material being updated (anything except its ID and version).

```
PUT   $URL/materials/<material_id>/<version>?magic-update=1
---
HTTP 200 OK
Content-Type: text/html
Allow: GET, PUT, DELETE, HEAD, OPTIONS
{
    "title": "No title",
    "text": "no text",
    "version": 2,
    "description": "some more changes",
    "categories": "1",
      …
}
```

5.4.9 Material comments

Each material can have multiple comments made by every person who has comment permission. The commenting feature is now common for all clients of VPs.

Add comment to specific material:

```
POST  $URL/materials/<material_id>/<version>/comments/
      person = 123,
      comment = "Hello, this is the first comment",
---
HTTP 201 CREATED
{
    "id": 1,
    "comment": ""Hello, this is the first comment",
    "person": 123,
    "modified": "2013-12-12T08:28:06.402",
}
```

View all comments of specific material:

```
GET   $URL/materials/<material_id>/<version>/comments/
---
HTTP 200 OK
{
    "count": 5,
    "next": null,
    "previous": null,
    "results": [
        {
            "id": 3,
            "person": 100,
            "comment": "Hello there",
            "modified": "2013-12-10T16:50:43Z"
        },
        {
            "id": 5,
            "person": 100,
            "comment": "Hello there",
            "modified": "2013-12-10T16:52:28Z"
        },
       ...
}
```
Update & delete comments

5.4.10 Material view counter

The counter of each material accepts only 3 actions: PUT, GET, and DELETE. The first one is for increment count value, and the latter is for getting the current view count.

```
        GET    $URL/materials/<material_id>/<version>/counter/
        ---
        HTTP 200 OK
        {
            "last_visit": "2013-12-10T20:53:13Z",
            "view": 4
        }
```

Use PUT to update material count value, if "increment" value is set to 1 if missing.

```
        PUT    $URL/materials/<material_id>/<version>/counter/
               increment = 2
        ---
        HTTP 200 OK
        {
            "last_visit": "2013-12-10T20:53:13Z",
            "view": 6
        }
```

To delete or reset the view count value:

```
        DELETE $URL/materials/<material_id>/<version>/counter/
        ---
        HTTP 200 OK
        {
            "last_visit": None,
            "view": 0
        }
```

Get most viewed materials:

```
        GET    $URL/stats/materials/counter/
        ---
        HTTP 200 OK

        [
            {
                "count": 12159,
                "title": "Number One Material",
                ...
            },
            {
                "count": 12120,
                "title": "Number Two Material",
                ...
            },
            ...
        ]
```

5.4.11 Material ratings

Every authenticated user (person) can perform rate upon specific material. Rating points are integers and selected from 1-5. User also can remove the rate that he/she made on single material.

To add rate:

```
POST   $URL/materials/<material_id>/<version>/rates/
       person = 123
       rate = 4
---
HTTP 200 OK
{
    "rate": 4.5,
    "count": 2
}
```

To get aggregated rate of single material:

```
GET    $URL/materials/<material_id>/<version>/rates/
---
HTTP 200 OK
{
    "rate": 4.5,
    "count": 2
}
```

To get single rate of given person on specific material:

```
GET    $URL/materials/<material_id>/<version>/rates?person=123
---
HTTP 200 OK
{
    "rate": 4
}
```

To delete rate submitted by single person to a material:

```
DELETE $URL/materials/<material_id>/<version>/rates?person=123
---
HTTP 200 OK
{
    "rate": 5,
    "count": 1
}
```

To reset/remove all rates of single material:

```
DELETE $URL/materials/<material_id>/<version>/rates
---
HTTP 200 OK
{
    "rate": null,
    "count": 0
}
```

Get top rated materials:

```
GET    $URL/stats/materials/rates/
---
HTTP 200 OK

[
    {
        "rating": 5,
        "rating_count": 123,
        "title": "Number One Material",
        ...
    },
    {
        "rating": 4.5,
        "rating_count": 12120,
        "title": "Number Two Material",
        ...
    },
    ...
]
```

5.4.12 Material favorite

Every authenticated user (person) can select/mark specific material as favorite for him/herself. User also can remove the mark that he/she made from single material.
To mark favorite:

```
POST   $URL/materials/<material_id>/<version>/favorites/
       person = 123
---
HTTP 200 OK
{
    "favorite": 2
}
```

To get number of selection of single material:

```
GET    $URL/materials/<material_id>/<version>/favorites/
---
```

```
HTTP 200 OK
{
    "favorite": 2
}
```

To check if person has marked a material as favorite or not:

```
GET    $URL/materials/<material_id>/<version>/favorites?person=123
---
HTTP 200 OK
{
    "favorite": 1
}
```

To delete favorite submitted by single person to a material:

```
DELETE $URL/materials/<material_id>/<version>/favorites?person=123
---
HTTP 200 OK
{
    "favorite": 1
}
```

Get top favorite materials:

```
GET    $URL/stats/materials/favorites/
---
HTTP 200 OK

[
    {
        "favorites": 50,
        "title": "Number One Material",
        ...
    },
    {
        "favorites": 41,
        "title": "Number Two Material",
        ...
    },
    ...
]
```

5.4.12 Get Material Collection Links

To list all collections containing a given module:

```
GET    $URL/materials/<material_id>/<version>/links/
---
HTTP 200 OK
[
      {
```

```
                "version": 1,
                "material_id": "45df218a",
                "title": "L\u1ecbch s\u1eed T\u1ef1 nhi\u00ean Vi\u1ec7t Nam"
        },
        {
                "version": 1,
                "material_id": "...",
                "title": "..."
        },
        …
]
```

5.5 Search Content

5.5.1 Normal search

All text data of authors, editors and materials are indexed and searchable via common searching tool. User can perform search on each kind of data base on given value of the parameter "?on=m" or "?on=p":

Implicit search (~ search on material):

```
GET    $URL/search?kw=<keywords>
GET    $URL/s?kw=<keywords>
```

Search on Material only:

```
GET    $URL/search?kw=<keywords>&on=m
GET    $URL/s?kw=<keywords>&on=m
```

Search on Person only:

```
GET    $URL/search?kw=<keywords>&on=p
GET    $URL/s?kw=<keywords>&on=p
```

Search with selected material type (collection or module):

```
GET    $URL/search?kw=<keywords>&on=m&type=1

GET    $URL/s?kw=<keywords>&on=m&type=1
```

--- sample result of material search:

```
HTTP 200 OK
Content-Type: text/html
Allow: GET, PUT, DELETE, HEAD, OPTIONS
Content-Type: text/html
{
    "count": 2,
    "next": null,
    "previous": null,
    "results": [
        {
            "material_id": "dbb0c9ebb17d6c07ce2fc6eb5cd2afa2",
            "material_type": "2013-01-28T09:53:41.372",
            "title": "Second one, check it out",
            "description": "The 787 development and production has involved...",
            "modified": "2013-01-28T09:53:41.372"
        },
        {
            "material_id": "c8dddba58950088faff712543be62ba9",
            "material_type": "2013-01-24T13:58:33Z",
            "title": "The first material",
            "description": "Editor's note: Christopher",
            "version": null,
            "authors": null,
            "editor_id": null,
            "modified": "2013-01-24T13:58:33Z"
        }
    ]
}
```

--- sample result of person search:

```
HTTP 200 OK
Content-Type: text/html
{
    "count": 6,
    "next": null,
    "previous": null,
    "results": [
        {
            "id": "vpr_content.person.3",
            "fullname": null,
            "user_id": "tranbinhminh",
            "email": "voer9@yahoo.com.vn",
            "client_id": null
        },
       ...
       ]
}
```

5.5.2 Faceted search:

VPR also supports providing information to faceting feature from web component. Returned faceted data contains corresponding number of: Categories, Language, Material type. Faceted search accepts input query similar to which of material browsing feature (see 5.4.1).

Get global faceted counts:

```
GET    $URL/facet/
```

Get faceted counts with conditions (notice: only one category in query allowed):

```
GET    $URL/facet?categories=1
GET    $URL/facet?categories=1&language=vi
---
HTTP 200 OK
Content-Type: text/html
```

```
{
    "fields": {
        "material_type": [
            [1,349],
            [2,3]
        ],
        "language": [
            ["vi",352]
        ],
        "categories": [
            [4,352]
        ]
    },
    "dates": {},
    "queries": {}
}
```

5.6 Get Update Information

This feature will return IDs of all modules which have check-in since a given time specified in sent request. This could help front-end component knows which are the newest changes without querying the whole database and compare to their own time records.

Caller also can enter argument **to=** to determine the time interval (to= is optional while since= is mandatory). Time value must be entered under YYYYMMDD format.

Calling of ../updates/ without module ID or time in past will receive error as result.

Request:

```
GET    $URL/updates?since=20121110
GET    $URL/updates?since=20121110&to=20121130
GET    $URL/updates/12323221?since=20121110
GET    $URL/updates/12323221/1.2/
```

Response:

```
200    OK
{
       "modules": {
              "id":"12323221",
              "versions":{
                     "1.1":”2012-11-11”,
                     “1.2”:”2012-11-12”,
              }
       },
       {
              "id":"12323222",
              "versions":{
                     "1.1":”2012-11-11”,
                     “1.2”:”2012-11-12”,
                }
            }
        }
```

5.7 Validate API Token

This part is mostly for internal component, like Transformer, for checking the client requests submitted on its side, not on VPR.

```
GET     $URL/token/<token>?cid=<client_id>
---
HTTP    200     OK
HTTP    401     UNAUTHORIZED
```

6. Error handling