API-usage instructions Ultrahack (v1.0)

1. Content Discovery

Katsomo API is located under http://api.katsomo.fi/api/web/ and responses in both formats, XML and JSON. Default response format is XML.

1.2 Response formats

As stated earlier, API’s default response format is XML. In order to get data out as JSON, easiest way is to include “.json” suffix in API calls. Some examples below:

http://api.katsomo.fi/api/web/search/categories/33005002/assets.json

http://api.katsomo.fi/api/web/category/33/recursive/categories/hierarchical.json

1.3 Site structure (understanding and using it)

All content in Katsomo has been divided hierarchically under several sub trees and full structure can be retrieved using URL http://api.katsomo.fi/api/web/category/33/recursive/categories/hierarchical/. Main tree (33) is currently our top level tree. Second level under it has genres and third level contains actual shows. A different type of layers is marked with <level> element and below is description for each of them:

TOP_LEVEL = main level of our service

CATEGORY = genre, for instance “Sports” (Main category)

SHOW = program, for instance “The Bold and the Beautiful”

EXTRAS = additional content of program (Show addon

1.4 Publish rules

Whether or not specific category (genre or show) is published, <priority> element tells it. If its value is -1 or 0, category is hidden and it should be excluded from listings. Same rule applies also to cases where <level> element is empty or not found. Navigational elements should be sorted by priority element so that biggest value is listed first from left to right or top to bottom, depending on UI-layout.

Regarding assets, unwanted content should not be indexed. That way, when clients are using Search API, they will only get assets which are meant to be visible on selected platform. This leads to certain set of rules which content producers should remember from now on. These rules are described below. All listed properties are tree related.

Tree type	If empty, tree does not show in any listing but its assets will as long as web searchable is checked. Otherwise tree should be marked as one of the options listed above in section 2.
Priority	If -1 or 0, category is hidden and won't be listed in Search API. However, if subcategory of such **category is visible (priority > 0) and web searchable is checked**, content of subcategory will be listed in Search API.
Web searchable	If **checked (and priority > 0)**, assets will be shown in Search API. if not checked, assets won't be shown in Search API.

1.5 Building “All programs” element

To list all available shows, client must fetch complete tree structure using above mentioned hierarchical URL and read all <category> elements from it marked with <level>SHOW</level>. All SHOW elements marked with <priority>-1</priority> must be left out.

1.6 Building genre filters, used for filtering “All programs” element

When looping through fetched hierarchical tree structure, genres are the ones marked with level CATEGORY. Use these in order to build genre filters.

1.7 Building video grids on program pages

All video grid content should be fetched using Search API. For instance, to get latest assets for show “The bold and the beautiful”, client should use URL http://api.katsomo.fi/api/web/search/categories/33005002/assets. This will list content from tree 33005002 and all possible sub trees. Note that content is listed only from indexed trees (web searchable checkbox must be checked in Katsomo admin).

In addition to this, video grid may have sub-navigational elements described in the next chapter.

1.8 Building sub-navigational elements on program pages

Some Katsomo shows may have additional content or content separated under several sub trees. Such trees are marked with <level>EXTRAS</level> and priority > -1. Whenever such element is found, it should be added as sub-navigational element into video grid of corresponding program page. When SHOW contains EXTRAS, UI should also list sub-navigational element “kaikki” that will list all assets under that SHOW –level (if EXTRAS is not marked as -1). sub-navigational elements should be sorted by priority element so that biggest value is listed first right after "kaikki" element from left to right or top to bottom, depending on UI-layout.

1.9 Building most-popular sub-navigational element (video grid filter)

Most popular filter can be populated using SOLR search for certain category, and doing sort based on count.

for example Emmerdale:

http://api.katsomo.fi/api/web/search/categories/33005003/assets?query=(liveBroadcastTime:[NOW-5DAYS/DAY TO NOW])&sort=count&size=20

1.10 Building most recent video grids for different categories

Most recent video grid listings can be populated using SOLR search for certain category.

for example building most recent video grid for "Urheilu"-category:

http://api.katsomo.fi/api/web/search/categories/33002/assets?query=(liveBroadcastTime:[NOW-5DAYS/DAY TO NOW])&size=20

1.11 How to Get Non-DRM Content

Add query condition prodGroupID:5.

> *http://api.katsomo.fi/api/web/search/categories/33011107/assets?query=**prodGroupId:5**&size=30*
>
> *MTV news clips:*
>
> *http://api.katsomo.fi/api/web/search/categories/33001002/assets?query=prodGroupId:5%20AND%20drmProtected:false%20AND%20liveBroadcastTime:%5B*%20TO%20NOW%5D&size=30*
>
> *Sport clips:*
>
> *http://api.katsomo.fi/api/web/search/categories/33002/assets?query=prodGroupId:5%20AND%20drmProtected:false%20AND%20liveBroadcastTime:%5B*%20TO%20NOW%5D&size=30*
>
> *F1 videos:*
>
> *http://api.katsomo.fi/api/web/search/categories/33002001/assets?query=prodGroupId:5%20AND%20drmProtected:false%20AND%20liveBroadcastTime:%5B*%20TO%20NOW%5D&size=30*

1.12 Retrieve a single video

Next, when a user clicks on the video, we want to load the asset page. It's usually recommended to have a separate page for asset playback, so that it can be shared to social media, bookmarked and display additional personalized functionality, such as a playlist. To retrieve a single asset, use the following API call:

> *HTTP·*
> *cURL: application/json;v=2·*
> *cURL: application/xml*

```
GET /api/web/asset/:assetId?expand=metadata
```

> ***API response***
> *application/json;v=2·*
> *application/xml*

```
{
  "asset": {
    "@id": "1954",
    "@channelId": "0",
    "@categoryId": "1231",
    "@uri": "/api/web/asset/1954",
    "archive": true,
    "aspect16x9": true,
```

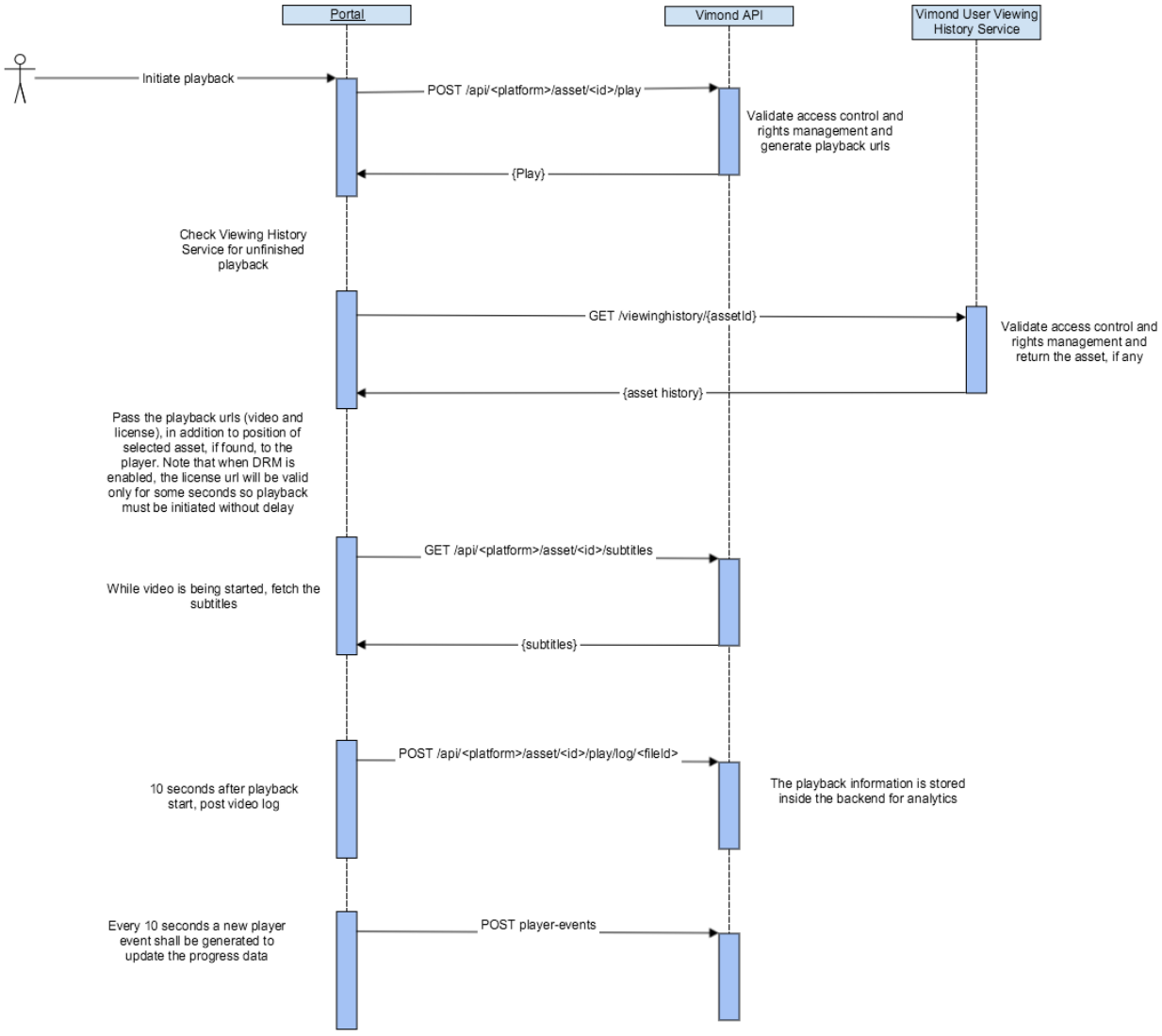
```
    "assetRelations": "",
    "assetTypeId": 0,
    "autoDistribute": false,
    "autoEncode": false,
    "autoPublish": true,
    "category": {
      "@uri": "/api/web/category/1231",
      "available": false
    },
    "copyLiveStream": false,
    "createTime": "2015-04-11T06:47:02Z",
    "deleted": false,
    "distributed": 0,
    "drmProtected": false,
    "accurateDuration": 0,
    "duration": 0,
    "encoderGroupId": 0,
    "externalAssets": "",
    "imageUrl": "https://image.yoursite.com/api/v2/img/552bfc99e4b097a8928f398b-1428946073902?
width=144&height=81",
    "imageVersions": {
      "image": {
        "@type": "original",
        "url": "https://image.yoursite.com/api/v2/img/552bfc99e4b097a8928f398b-1428946073902?w
idth=144&height=81"
      }
    },
    "itemsPublished": false,
    "items": {
      "@uri": "/api/web/asset/1954/items"
    },
    "labeledAsFree": true,
    "live": false,
    "liveBroadcastTime": "2015-04-11T06:46:15Z",
    "metadata": {
      "@uri": "/api/metadata/asset/1954",
      "title": {
        "@xml:lang": "en_US",
        "$": "A Walkthrough of VCC - The Basics"
      },
      "image-pack": {
        "@xml:lang": "*",
        "$": "552bfc99e4b097a8928f398b-1428946073902"
      },
      "description-short": {
        "@xml:lang": "en_US"
      }
    },
    "onDemandTimeBegin": 0,
    "onDemandTimeEnd": 0,
    "parent": true,
    "playbackStrategy": "DEFAULT",
    "playback": {
      "@uri": "/api/web/asset/1954/play"
    },
    "product": {
      "@uri": "/api/web/asset/1954/productgroups"
    },
    "encoderProfileId": 0,
    "showInEpg": false,
    "title": "A Walkthrough of VCC - The Basics",
    "updateTime": "2016-02-08T15:13:52Z",
```

```
    "views": 0
  }
}
```

2. Video Playback

Sequence diagram

Here is a simple sequence diagram overview on how to enable video playback and optionally allow resuming playback.



Step 1:Fetch playback information

If there is no device registration required you can perform a GET request instead

```
HTTP·
```

cURL

GET /api/:platform/asset/:assetId/play

API response

application/json;v=2


```
{
   playback:{
      title:"Guardians of the Galaxy",
      live:"false",
      aspect16x9:"true",
      drm_protected:"false",
      playback_status:"OK",
      has_items:"false",
      live_broadcast_time:"2015-12-28T09:27:21Z",
      items:{
         item:[
            {
               log:{
                  uri:"/api/web/asset/3101/play/log/7925/"
               },
               bitrate:"0",
               file_size:"10337",
               media_format:"ismusp",
               scheme:"http",
               server:"video.site.com",
               base:"http://video.site.com",
               url:"http://video.yoursite.com/62f91/guardiansOfTheGalaxy_310(3101_ISMUSP)_v3.i
sm/Manifest"
            },
            {
               log:{
                  uri:"/api/web/asset/3101/play/log/7925/"
               },
               bitrate:"0",
               file_size:"10337",
               media_format:"ismusp",
               scheme:"http",
               server:"video.site2.com",
               base:"http://video.site2.com/video/origin",
               url:"http://video.yoursite.com/video/origin/62f91/guardiansOfTheGalaxy_310(3101
_ISMUSP)_v3.ism/Manifest"
            }
         ]
      },
      log_data:"3833ADE526936DC1BB707DF5906AEA6E38B09708C9EA09C29FA9C2815F1897FEA2AEC681BBC672
8A40F693261423EB0944FD8636EB06B117927AD46362202EA8F14726D8E3BF6EB497C630CD752B10E32B9B6812377C
FA9A30FCDA381958973FD7EE23AC9DEBEE957BD1FAE56D0C4D16184FA2547736771D16BE9898D6B5282491F13215D8
18DD4F70F36BEF4EB252467EC90AAFEC9B5E0F98FB4FCA5E521F06CD555912B8466C28AB50C5B99F613A331F08FCB3
92E28E013DF013773DB89817400AFE1082471CA590F74925BA1DF81196C5F5142D01C9187D7056D97AA492BBA18F65
86247CF54F5F5D39E2C33E3E29EA400621CDF326A069D50511B8617BC49C73A765ED5882989773 26D1C3AFC2E048C7
5AB924E5B727052078FD3BD15F4D",
      stream_rule:{
         max_subscription_streams:"2147483647"
      },
      asset_id:"3101"
   }
}
```


ERROR CODES

Code	Comment
ASSET_NOT_FOUND	Asset not found
ASSET_PLAYBACK_INVALID_VIDEO_FORMAT	No playback url available for this protocol/video format

SERVER_NOT_FOUND	No playback url available for this protocol/video format
PERMISSION_DENIED	The user is not allowed to view this video
ASSET_PLAYBACK_INVALID_GEO_LOCATION	User is outside of the allowed Geo region
ASSET_NO_ACCESS	User does not have a valid subscription to watch this video
DEVICE_INFO_REQUIRED	The user device info were not part of the request
DEVICE_LIMIT_EXCEEDED	The user has used its quota of allowed devices

Those are the common error codes that can be used by the front end in order to provide various error messages. Note that in addition to the mentioned error codes, you might want to look at the **playback_status** inside the response, when set to NOT_STARTED, it means that the asset is visible but it is not yet available for playback. This can be used among others for promotion of live events such as Lynx

PROTOCOL SELECTION

The Vimond platform is able to present video url in middle streaming format: HLS, DASH, ISM.

It is up to the player/device to request the correct protocol as a query parameter upon playback. See example below:

> *HLS·*
> *DASH·*
> *ISM*
>
> `POST https://api.yoursite.com/api/web/asset/3101/play?protocol=HLS&app_name=ios_player`

If the protocol is missing, the API will select the default protocol configured for this platform.

It should be noted that when using protocol=DASH also Smooth Stream manifest files are returned. If protocol parameter is set to MPD, only dash manifest is returned.

PLAYBACK VALIDITY

Both the url to the video file and for the license may contain a temporary token. Therefore, the playback API must be called right before playback is initiated on the device. This endpoint can not be cached or store in any middle layer logic, this is the last thing that must happen when the user press play.

FALLBACK LOGIC

The playback endpoint will return not a single but a list of playback urls with the preferred url first in the result set. If the player has implemented a fallback logic, it could automatically try out the second url if playback fails with the first one

Step 2: Check if video has view history for resume playback

Upon pressing **play**, if the selected video has previously been started and not completed, the users might be prompted with the choices of **resuming from where they left off** or **playing it from the start**, or the asset may resume by default, depending on your implementation.

> *HTTP·*
> *cURL*
>
> `GET /microservices/viewing-history/:assetId`
>
> ***API Response***
>
> *JSON*

```
{
  "title": "This is the title of an asset",
  "assetId": "12345",
  "totalTime": 240000,
  "progress": {
    "position": 100000,
    "date": "2016-03-22T14:09:15.088Z",
    "updateTime": "2016-03-22T14:09:16.558Z"
  }
}
```

Step 3: Subtitles

We start by getting a list of available subtitles for the video:

```
HTTP·
cURL
```

```
GET /api/web/asset/:assetId/subtitles
```

API Response

```
application/json;v=2
```

```
[
  {
    "assetId": 3101,
    "contentType": "text/srt;charset=\"UTF-8\"",
    "locale": "th",
    "name": "Thai",
    "id": 4183,
    "type": "NORMAL",
    "uri": "/api/web/asset/3101/subtitle/4183"
  },
  {
    "assetId": 3101,
    "contentType": "text/srt;charset=\"UTF-8\"",
    "locale": "ms",
    "name": "Malay",
    "id": 4182,
    "type": "NORMAL",
```

Authentication

```
"uri": "/api/web/asset/3101/subtitle/4182"
  },
  {
    "assetId": 3101,
    "contentType": "text/srt;charset=\"UTF-8\"",
    "locale": "en",
    "name": "English",
    "id": 4181,
    "type": "NORMAL",
    "uri": "/api/web/asset/3101/subtitle/4181"
  }
]
```

In this case we have three language options for text/srt; Thai, Malay, and English. By using the URI we can download the appropriate subtitle for our user:

```
HTTP·
cURL
```

```
GET /api/web/asset/:assetId/subtitle/:subtitleId
```

API Response

```
text/srt
```

```
1
00:00:29,570 --> 00:00:33,380
Forget it. It's too risky.
I'm through doing that shit.

2
00:00:33,380 --> 00:00:35,840
You always say that.
The same thing every time.

3
00:00:35,920 --> 00:00:38,840
"I'm through, never again, too dangerous."

4
00:00:38,930 --> 00:00:41,180
I know that's what I always say.
I'm always right too.

5
00:00:41,260 --> 00:00:44,890
- But you forget about it in a day or two.
- The days of me forgetting are over.

...
```

Step 4: Playback log

Once the playback has been initiated, the player should send back log data to the API about 10 seconds later. This is to actually register that the playback has started and not only that the playback endpoint was called. In order to do so, the player should call the **log.uri** endpoint provided in the playback data and send the **log_data**

Building on above example, the following endpoint would be called and data posted:

```
HTTP·
cURL
```

```
POST /api/web/asset/:assetId/play/log/:logId/
```

Step 5: Player events

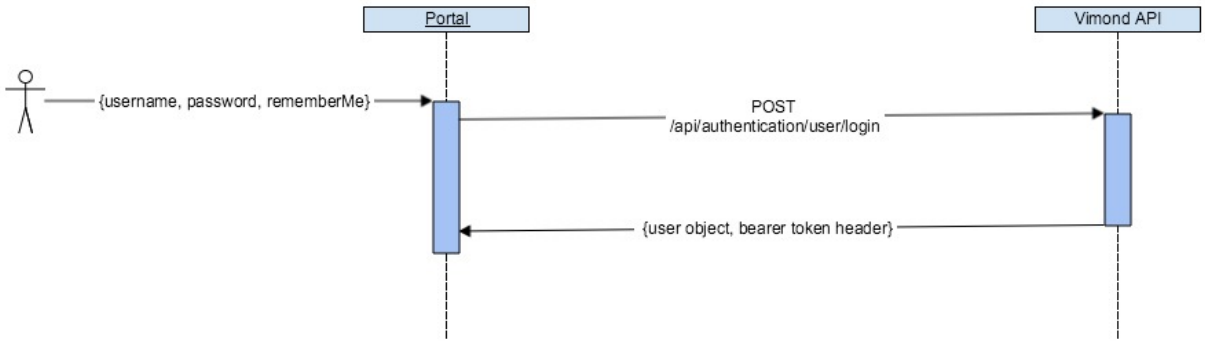
> *Once playback is going, the player should send periodic events (every 10th seconds) in order to provide the progress information. Below is an example of player-event where **originatorId** is the Users unique userId and **ssoToken** is the users authorization Bearer token.*

```
HTTP·
cURL
```

```
POST /microservices/event-service/player-events
```

3. Logging in a User

Here is a simple sequence diagram overview on how to login and receive an access token.



In order to remain logged in, thus being able to authenticate with api services, a valid session cookie and an authorization token is needed. This information can be found in the response.

3.1 Session based authentication

For manually storing cookies from a response, the following Curl call will do the trick:

Storing cookies to file

Example:

> *application/json;v=2·*
> *application/xml*

```
curl -c cookies.txt -i -X POST
  -H "Accept: application/json;v=2"
  -H "Content-Type:application/json;v=2"
  -d
'{
  "username":"your_user@your_domain.com",
  "password":"yourPassword"",
  "rememberMe":"false"
}'
  "https://api.yoursite.com/api/authentication/user/login"
```

3.2 Header based authentication

The Authorization header in the response contains the bearer token required for user authentication.
In order to store the bearer token, get the contents of response header named "Authorization".

Code sample

> *In order to access certain api functions regarding a specific user account, one is required to log in to said account.*
>
> *Upon sending a valid login request, a response containing an Authorization header is returned. For example:*

> *HTML*

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Authorization: Bearer dc386b8a-ab09-44d9-9f8f-0986bd3c57b9
Content-Type: application/xml;charset=UTF-8
Content-Length: 230
Date: Tue, 01 Mar 2016 09:54:40 GMT

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<response>
    <code>AUTHENTICATION_OK</code>
    <description>Authentication successful</description>
    <reference>53c4460aa4df43f7</reference>
    <userId>01234</userId>
</response>
```

Code sample
Java snippet showing how to get bearer token from response header:

```
Java
```

```
String bearerToken = response.getHeader("Authorization");
```

3.3 Logging in with a user

Now that we you how to handle the response, let's look at a request example:

```
HTTP·
cURL
```

```
POST /api/authentication/user/login
```

3.4 Further use of cookies and authorization token

For manually adding cookies to a request, the following Curl call will do the trick:

Code sample adding cookies from file
Example:

```
cURL
```

```
curl -b cookies.txt -X GET
-H "Accept: application/json"
-H "Authorization: Bearer dc37201c-4481-c216-8f5d-5c211k2649"
"https://api.yoursite.com/api/web/user"
```

In order to access api functions that require being logged in, include "Authorization" header with the bearer token as value in request.

Code Samples: Inserting bearer token into HTTP Requests

The Authorization header for

```
HTTP Header
Authorization:<space>Bearer<space>[BearerToken]
```

Example of request with Authorization header using cURL

```
cURL
```

```
curl -X GET -H "Accept: application/json" -H "Authorization: Bearer
49bb2921-4e9b-4k19-9eb7-1995236f1abd" "https://api.yoursite.com/api/web/user"
```

3.4 Logging out a User

A session will typically expire after 30 minutes of inactivity. It's also possible to explicitly log out.

Calling logout while providing a Bearer token will by default also invalidate your token from future use. This behavior can be disabled via configuration.

```
HTTP·
cURL
```

```
DELETE /api/authentication/user/logout
```

API Response

```
application/json;v=2·
application/xml
```

```
{
  "response": {
    "code": "SESSION_INVALIDATED",
    "description": "Session invalidated",
    "reference": "e2417b5376aa38ce"
  }
}
```

4. Viewing History

In the user portal, you may want to display viewing history of some sort. For example you might want to display one or more of the latest assets that the user has been watching without completing, or a list of previously completed assets. The user may also want to clear his or her viewing history.

4.1 Getting viewing history

There are several kinds of view histories to display for a user:

All asset history

```
HTTP·
cURL
```

```
GET /microservices/viewing-history/all
```

All assets in progress

```
HTTP·
cURL
GET /microservices/viewing-history/progress
```

4.2 History for a given asset

API Reference

> *HTTP·*
> *cURL*
>
> `GET /microservices/viewing-history/:assetId`

4.3 All completed assets

> *HTTP·*
> *cURL*
>
> `GET /microservices/viewing-history/completed`

4.4 Categories in which the user has watched something

> *HTTP·*
> *cURL*
>
> `GET /microservices/viewing-history/categories/all`

4.5 Deleting viewing history

A user may want to clear his or her viewing history.

The secret key should Delete all viewing history

> *HTTP·*
> *cURL*
>
> `DELETE /microservices/viewing-history`

Delete a specific asset's viewing history

> *HTTP·*
> *cURL*
>
> `DELETE /microservices/viewing-history/:assetId`