api-spec

Bergar Simonsen

April 10, 2013

Contents

1 api spec

1.1 M6, Movie Distribution Web Service API Specs

The web service uses RESTful concepts to provide a standardized and stateless programming interface, for interacting with the system.

Standard response
All responses consists of the following data:

Table 1: Response message objects

Message	Description
error_number	0 on success, otherwise a number indicating the relevant error. NOT TO BE CONFUSED WITH HTTP-Response-code! This is assuming the HTTP-Response-code is 200.
error_message	A message in English, describing the eventual error.

Security and encoding
All requests should be made with a valid API-key, and values hashed into a "Checksum". (TODO: Describe hashing details). Requests should be accompanied by a UTC timestamp and a nonce. All authenticated requests should contain an access token.
So every request should look something like:
`Resource]?auth=`AUTH-STRING]&parama=xxx&paramb=yyy

Where [AUTH-STRING] is something similar to: { "api-key" : "AB14" , "hmac" : "XZ45" , "time" : "213" , "nonce" : "XC98" ; "token" = "DE95" }

The only optional part of this auth object, is the token, which is only used after the user has successfully logged in.

Objects
Every area of the application has some certain objects with a well defined structure. These are described initially in the relevant section, and utilized throughout the API. This allows for client to receive and process these objects in a uniform manner.

Parameters
Are GET or POST arguments, depending on the request-type. The left side column of the table shows the argument name, while the right one describes the argument.

Response
Is an associative JSON object, with fields corresponding to the left side column of the table. The right side of the column describes the data in the field of the returned response.

1.2 Users (users, tokens & user data types)

The users are at the heart of the system. All actions are performed by users, and their identities are associated with both actions and data entities within the system. Users are very simple, but can be extended to application specific needs. This is done by creating new user data types (unique text indexes, that can be used to store data for each user), and assigning data to the users. Tokens are used for accessing the application as a specific user, and is given upon submission of a valid email and password combination.

Table 2: User objects

Field	Description
id	The users id
email	E-mail of the user. False if currently logged in user is not permitted to read.
user_data	All data that exists for this user as an associative structure

Table 3: Token objects

Field	Description
token	A token string, to be used in further queries
issued	Date and time for issuance
expires	Date and time for expiration

Get user

Url structure: /user/<ID>
Description: Who is the user with the id <ID>?
Method: GET
Parameters: None
Response:

Field	Description
user	user

Get currently logged in user

Url structure: /user/me
Description: Who is the currently logged in user?
Method: GET
Parameters: None
Authorization: Token
Response:

Field	Description
user	user

Get all users (with parameters)
Url structure: /user/
Description: Who are the users that match the given parameters
Method: GET
Parameters: **group_ids** Comma separated list of group-ids.
Only show users who are members of this/these group(s)
search_string The string to search for
search_fields Comma separated list of fields to use for matching the string
limit How many users to return?
page Should there be an offset? Default = 1 means no offset.
order_by Order by what column? Default = e-mail.
order Order which way? Default = ASC.
Authorization: Token
Response:

Field	Description
users	array[User]
count	Number of users in total, regardless of limit
count_pages	Number of pages needed for users with current limit

---

Post user access token
Url structure: /user/token
Description: Can i have an access-token with these credentials?
Method: POST
Parameters: **email** The users e-mail
password An SHA-1 hash of the users password.
Response:

Field	Description
token	Token

Renew user access token

Url structure: /user/token/renew
Description: Can I renew this token?
Method: POST
Parameters: None
Response:

Field	Description
token	Token

Create new user

Url structure: /user
Description: Create a new user with this data
Method: POST
Parameters: **e-mail** The users e-mail. Doubles as a username.
password The users password. SHA-1 hashed.
user_data Other data for this user as an associative array.
NB: All data must already be present as user data types.
Response: None

Delete user

Url structure: /user/<ID>
Description: Delete the user with this id
Method: DELETE
Parameters: None
Response: None

Update user

Url structure: /user/<ID>
Description: Update this user with this data
Method: PUT
Parameters: **e-mail (optional)** The users new e-mail
old-password (optional) The users current password.
SHA-1 hashed.

password (optional) The users new password.
SHA-1 hashed.
user_data Other data for this user as an associative array.
NB: All data must already be present as user data types.

Response: None

Get a list of all user data types

Url structure: /userdatatype
Description: Get all user data types for this system
Method: GET
Parameters: **name** Select all user data types with this name.
Used to test if a given data type exists.
Response:

Field	Description
userdatatypes	An array of user data types as strings

Create a new user data type

Url structure: /userdatatype/<NAME>
Description: Make a new user data type
Method: POST
Parameters: **name** The name of the new user data type.
Response: None

Delete a user data type

Url structure: /userdatatype/<NAME>
Description: Delete user data type with this name <NAME>
Method: DELETE
Parameters: None
Response: None

1.3 Media (Media & Media Category)

Table 4: Media objects

Field	Description
id	A unique id of the media
media_category	The id of the media's category
media_category_name	The name of the media's category
user	The id of the user who uploaded
file_location	The location of the connected file
title	The title of the media
description	The description of the media
media_length	The length of the media in minutes
format	The format of the file
tags	A list of tags connected to the media

MediaCategory

Field	Description
id	A unique id
name	The name of the media category

Get media with specific id
Url structure: /media/<ID>
Description: Get a specific media, based on it's id
Method: GET
Parameters: None
Response:

Field	Description
media	Media

Get all medias (with parameters)
Url structure: /media
Description: Get all media matching the giver criteria.
Can be used for listings and searches.
Method: GET
Parameters: **andTags** A list of tags where the media has to match all of them

orTags A list of tags where the media has to match one of them
mediaCategoryFilter A media category id that filters the medias.
nameFilter A string that filters the medias
page The page you are on
limit The amount of medias per page

Response:

Field	Description
pageCount	Amount of pages
medias	array[Media]

Create a new media

Url structure: /media
Description: Create a new media and get a path for your upload. This will only create an entry in the database with the meta data provided. Returns id.
Method: POST
Parameters: None
Content-Type **application/json**
media_category The id of the media's category
title The title of the media
description The description of the media
media_length The length of the media in minutes
format The format of the media file
tags A list of tags connected to the media

Response:

Field	Description
id	The id of the new media

Upload a media file associated with a media

Url structure: /mediaFiles/<ID>
Description: Upload a media file. You give the ID connected the posted meta data and the file you want to upload.
Method: POST
Parameters: None
Content-Type File Stream
Response: Response message

Update media

Url structure: /media/<ID>
Description: Update the metadata of a media.
Method: PUT
Parameters: None
Content-Type **application/json**
media_category The id of the media's category
title The title of the media
description The description of the media
media_length The length of the media in minutes
format The format of the media file
tags A list of tags connected to the media
Response: Response message.

Delete media

Url structure: /media/<ID>
Description: Delete a media. This will also delete the file connected to the media.
Method: DELETE
Parameters: None
Response: Response message.

Get all media categories

Url structure: /mediaCategory
Description: Get a list of all media categories
Method: GET
Parameters: None
Response:

Field	Description
media_categories	array[MediaCategory]

Get media category with specific id

Url structure: /mediaCategory/<ID>
Description: Get a media category
Method: GET
Parameters: None

Response:

Field	Description
media_categories	MediaCategory

New media category

Url structure: /mediaCategory
Description: Creates a new media category
Method: POST
Parameters: None
Content-Type **application/json**
name The name of the media category.
Response:

Field	Description
id	The unique id of the media category posted

Update media category

Url structure: /mediaCategory/<ID>
Description: Update media category
Method: PUT
Parameters: None
Content-Type **application/json**
name The name of the media category.
Response: Response message

Delete media category

Url structure: /mediaCategory/<ID>
Description: Delete media category
Method: DELETE
Parameters: None
Response: Response message

1.4 Tags

Table 5: Tag objects

Field	Description
id	A unique id
name	The name of the tag
simple_name	The short version of the name
tag-group	Tag group

Table 6: Tag group objects

Field	Description
id	A unique id
name	The name of the tag group
description	The tag group description

Get all tags
Url structure: /tags
Description: Get a list of all tags
Method: GET
Parameters: **tagGroupFilter** The id of the tag group you want to filter by
limit Amount of tags per page
page The page number
Response:

Field	Description
countPage	The amount of pages
tags	array[Tag]

Get tag with specific id
Url structure: /tags/<ID>
Description: Get a tag
Method: GET
Parameters: None
Response:

Field	Description
Tag	Tag

New tag

Url structure: /tags
Description: Create a new tag
Method: POST
Parameters: None
Content-Type **application/json**
name The name of the tag
simple_name The short version of the name
tag-groups A list of tag groups
Response:

Field	Description
id	The unique of the posted tag

Update tag

Url structure: /tags/<ID>
Description: Update a tag
Method: PUT
Parameters: None
Content-Type **application/json**
name The name of the tag
simple_name The short version of the name
tag-groups A list of tag groups
Response: Response message

Delete tag

Url structure: /tags/<ID>
Description: Delete a tag
Method: DELETE
Parameters: None
Response: Response message

Get tag group

Url structure: /tagGroups/<ID>
Description: Get a tag group
Method: GET
Parameters: **limit** Amount of tags per page.
page The page number.
Response:

Field	Description
tag_groups	array[TagGroup]

New tag group

Url structure: /tagGroups
Description: Create a new tag group
Method: POST
Parameters: None
Content-Type **application /json**
name The name of the new tag group
description The tag group description
Response:

Field	Description
id	The unique id of the new tag group

Update tag group

Url structure: /tagGroups/<ID>
Description: Update a tag group
Method: PUT
Parameters: None
Content-Type **application /json**
name The name of the new tag group
description The tag group description
Response: Response message.

Delete tag group

Url structure: /tagGroups/<ID>
Description: Delete a tag group (this will also delete tags connected to the tag group, or delete the connection)

Method: DELETE
Parameters: None
Response: Response message.

1.5 Orders

Url structure: /transactionHistory
Description: Get transaction history for a user.
Method: GET
Parameters: **user** The "owner" of the transaction history.
Response:

Field	Description
Transaction	array[Transaction] All transactions for the user
Order	array[order] All orders for the user
Promise	array[promise] All promises for the user

Url structure: /transaction
Description: Creates a new transaction for when the user wants to purchase additional functionality.
Method: POST
Parameters: None.
Content-Type **application/json**
promise array[promise]
order array[order]
order_line Order line containing all orders
Response:

Field	Description
Field	Description
Transaction_id	Id of the posted transaction
Response_data	Text describing the status of the transaction

1.6 rating

Get all rating for specific media
Url structure: /rating/<media>
Description: Returns all the ratings / comments on a specific media.
Method: GET
Parameters: None.
Response:

Field	Description
user_id	The user who has rated
media_id	Id of the media that the rating belongs to
starts	Amount of stars given in the rating
comment_title	Title of the comment
comment	Content of the comment

New rating for media

Url structure: /rating
Description: Posts a new rating for a media
Method: POST
Parameters: **user_id** The id of the user (current user)
media_id Id of the media to comment on
stars Number of stars to give to the media
comment_title Title of the comment
comment Content of the comment
Response: Response message

Edit rating

Url structure: /rating<ID>
Description: Edits an already existing comment.
Method: PUT
Parameters: **id** Id of the rating to edit
comment_title Title of the new comment
comment Content of the new comment
stars New amount of stars
Response: Response message