API Documentation

M. Brunelle, H. Defrance, G. Derivery,
G. Feraud, V. Meilhac, A. Pasquini,
S. Rose, H. Xie

Revisions table

Date	Author	Section	Comment
06/08/2013	Hugo Defrance	All	Added titles & general presentation
07/08/2013	Hugo Defrance	All	Request details
07/11/2013	Hugo Defrance	All	Page layout, All sections updated and corrected
11/11/2013	Hugo Defrance	Playlists	Added playlist section
12/11/2013	Hugo Defrance	Music	Modification of the type of the time element of a song
14/11/2013	Hugo Defrance	All	Added a method to delete of POST for iOS compatibility
24/11/2013	Hugo Defrance	Friend lists	Added the username of a friend in the GET request + details of the return value of the request
28/11/2013	Hugo Defrance	Playlist	It is now possible to add a new song to a playlist
05/12/2013	Hugo Defrance	Friend lists	Modification of the delete method for a friend – it is now possible to delete more than one at a time
10/12/2013	M. Brunelle	All	Translation to English
12/12/2013	Hugo Defrance	User – Instant Messaging	Username must now be unique + added some information in instant messaging doc
03/01/2014	Hugo Defrance	User – Track – Instant Messaging	Add a method to reset a forgotten password + removal of useless attributes created_at, updated_at and deleted_at + added some information in instant messaging doc

Table of contents

Revisions table **2**

Table of contents **3**

API presentation **4**

Available requests **4**

1. **User** **5**
   1. Register – User account and token creation 5
   2. Connection – token creation 6
   3. Disconnection – Token destruction 6
   4. Update information of the user account 7
   5. Forgotten password retrieval 7
   6. Obtaining user information 8
   7. Obtain the User list 9

1. **Friend List** **10**
   1. Add a friend 10
   2. Delete friends 11
   3. Obtaining the friend list 12

2. **Music** **12**
   1. Get a song details 12
   2. Obtain the music of the user 13
   3. Adding a song 14
   4. Updating a song information 15
   5. Delete a song from the library 15
   6. Delete one or more songs from the library 16

3. **Playlist** **17**
   1. Creating a playlist 17
   2. Modify a playlist 18
   3. Delete a playlist 18
   4. Delete multiple playlists 19
   5. Adding multiple songs to a playlist 20
   6. Delete multiple songs in a playlist 21
   7. Delete a song from a playlist 22
   8. Get songs in a playlist 22
   9. List playlists 23
   10. Print the details of a playilst 23

4. **Instant messaging** **24**

API presentation

All the elements of the project (Android, iOS, PC & MAC apps and the website) will communicate with the main server through a REST type API. It will allow us to access to the main database of the project and be responsible of the authentication of the users and the socials functionalities.
This API will use the HTTPS protocol to guarantee security and also flexibility with its usage. A majority of modern programming languages (such as Java, Ruby or Python) implement this protocol natively, which allows the programmer to do request easily and securely.

Available requests

All the requests must be encoded in UTF-8 and must contain a “Content-Type” of type “application/json”. The parameters of the requests must be formatted in JSON valid format.

To authenticate users, all of the requests (except the create account, the connection requests and password forgotten request) must contain the GET “token” parameter containing the token. The requests will have the following format:
« /<request>?token=<token> ».

The server will respond to all of the requests using the answer codes of standard HTTP and will eventually contain data encoded with JSON.

1. User

1. Register – User account and token creation

This request is used to register a new user. Once the account created the user is connected automatically.

Request

POST /api/users

All the parameters except first_name and last_name are mandatory.

```
{
  "user":
           {
             "email" : "<user_email>",
             "password" : "<user_password>",
             "username" : "<user_username >",
             "first_name" : "<user_first_name >",
              "last_name" : "<user_last_name >",
            }
}
```

Response

If the request was successful you will receive a “201 Created” response containing the token. Here is an response example:

```
{
  "success" : true,
   "token" : "<token>",
   "id" : "<user id>",

}
```

Errors

In case of errors, you will receive a “403 Forbidden” message response. The error field only contains the first detected error. If there are multiple errors they will be sent once the first error is fixed. Here’s an example of a error response:

```
{
  "success": false,
  "error": <first error string>
}
```

2. Connection – token creation

This request generates a token for an existing account.

Request

POST /api/users/login

```
{
  "email" : "<user_email>",
  "password" : "<user_password>"
}
```

Response

If the request is a success, a « 201 Created » response will be return containing a token. Here's what the response looks like:

```
{
  "success" : true,
   "token" : "<token>"
}
```

Errors

In case of errors, you will receive a "403 Forbidden" response. Here's the format of the answer:

```
{
  "success" : false,
  "error" : "Invalid email or password."
}
```

3. Disconnection – Token destruction

This request allows you to destroy a token.

Request

DELETE /api/users/logout

Response

If the token is valid, a « 200 OK » response will be answered:

```
{
  "success" : true,
  "message" : "Token deleted"
}
```

Errors

If the token is invalid, a « 404 Not Found » type response will be returned:

```
 {
   "success" : false,
   "message" : "Invalid token."
}
```

4. Update information of the user account

This request allows you to update the information of a user account (such as the password). It's possible to update all the fields in the User object (see account creation for more information). In the following example, the password is updated:

Request

PUT /api/users

```
{
   "user" :
               {
                  "password" : "<new password>"
               }
}
```

Response

If the parameters are valid, a « 201 Created » type response will be returned:

```
{
  "success" : true,
  "message" : "User has been updated"
}
```

Errors

If the token is invalid, a « 403 Forbidden » type response will be returned:

```
{
  "success" : false,
  "message" : "Token is invalid"
}
```

5. Forgotten password retrieval

This request let a user reset his password when forgotten by sending a confirmation email.

Request

POST /api/users/reset-password-link

This request requires an email address or a username in its parameters and do not require an authentication token.

```
{
  "email":<email de l'utilisateur>
}
```

OR

```
{
   "username":<username de l'utilisateur>
}
```

Response

If the email address or the username match with a valid user, an email will be sent to the user with the password recovery procedure. A « 200 OK » type response will be returned by the API:

```
{
  success: true
  message: "You will receive an email with the password reset procedure."
}
```

Errors

If the email address or the username do not match with any user, a « 404 Not Found » type response will be returned:

```
{
   success: false
   error: "User not found"
}
```

6. Obtaining user information

This request allows you to get information of a user.

Request

GET /api/users/<user_id>
GET /api/users/me

The second request is a shortcut that allows access to the connected user's profile. The same type of answer will be returned.

Response

If the token is valid and that the user's ID exists, a « 200 OK » type response will be returned. Here's a response example:

```
{
  "success" : true,
  "user": {
```

```
            "email" : <user_email>,
            "email_verified": <boolean>
            "id" : <user_id>,
            "first_name" : "<user_first_name>",
            "last_name" : "<user_last_name>",
            "username" : "<user_username>"
           }
}
```

Errors

If the token is missing or invalid, a « 403 Forbidden » type response will be returned:

```
{
  success : false,
  error : "token field is missing"
}
```

If the user doesn't exist, a « 404 Not Found » type response will be returned:

```
{
  "success" : false,
  "error" : "User does not exist"
}
```

7. Obtain the User list

This request allows you to obtain a user list with their public informations.

Request

GET /api/users

Response

If the token is valid, a « 200 OK » type response will be returned:

```
{
  "success" : true,
  "list":
```

```
              [
                {<user 1>}, {…}, {<user n>}
              ]
}
```

Errors

If the token is missing or invalid, a « 403 Forbidden » type response will be returned (in this example, the token is missing) :

```
{
  success : false,
  error : "token field is missing"
}
```

1. Friend List

1. Add a friend

This request allows you to add a friend to the friend list of the user.

Request

POST /api/friends

```
{
  "friend_email":<friend_email (string)>
}
```

Response

If the parameters are valid, a « 201 Created » type response will be returned. If both users are already friends, a« 200 Created » type response will be returned:

```
{
  "success" : true,
  "friend" : <Utilisateur>
}
```

Errors

If the ID of the friend is invalid, a « 404 Not Found » type response will be returned:

```
{
   "success" : false,
   "error" : "Friend does not exists"
}
```

If the token is missing or invalid, a « 403 Forbidden » type response will be returned (in the following example, the token is missing):

```
{
   success : false,
   error : "token field is missing"
}
```

If the “friend_email” is missing , a “400 Bad Request” type response will be returned:
```
{
   success: false
   error: "friend_email field is missing"
}
```

2. Delete friends

This request allows you to delete friends from the user’s friend list.

Request

DELETE /api/friends ?friends_email[]=<email 1>&friends_email[]=<email n>
POST /api/friends/delete

If the POST method is used, an email list to deleted must be placed in the parameters.

```
{
  "friends_email":[<email 1>, <email 2>, <email n>]
}
```

Response

If the parameters are valid, a « 200 Ok » type response will be returned:

```
{
  "success" : true,
  "message" : "<nb_deleted> friend have been deleted from your friendlist"
  "nb_friends": <number of friends remaining in friend list (int)>
   "nb_deleted": <number of friends deleted from the friend list (int)>
}
```

Errors

If no email matches with a friend’s email, a « 404 Not Found » type response will be returned:
```
{
 "success" : false,
 "message" : "<nb deleted> friend were deleted from your friendlist",
 "nb_friends":<user’s number of friend>,
 "nb_deleted":<nb deleted friends>
}
```

If the “friends_email” field is missing, a "400 Bad Request" type response will be returned:
```
{
   success: false
```

```
  error: "friends_email field is missing"
}
```

3. Obtaining the friend list

This request allows you to obtain the user's friend list with their information.

Request

GET /api/friends

Response

If the token is invalid, a "200 OK " type response will be returned:

```
{
  success: true,
  friends: [
            {
              "id": <friendship id (int)>,
              "friend_id": <friend id (int)>,
              "user_id": <user id (int)>,
              "created_at": <datetime (string)>
              "first_name": <friend first_name (string)>,
              "last_name": <friend last_name (string)>,
              "username": <friend username (string)>,
              "email": <friend email (string)>
            },
            <user 2>, ..., <user n> ],
  nb_friends : <number of friends in the list (int)>
}
```

Errors

If the token is missing or invalid, a « 403 Forbidden » type response will be returned (In the following example, the token is missing) :

```
{
  success : false,
  error : "token field is missing"
}
```

2. Music

1. Get a song details

This request allows you to obtain the information of a user's song.

Request

GET /api/tracks/<track_id>

Response

If the token is valid and the song belongs to the user, a “200 OK” will be returned:

```
{
   "success":true,
   "track":
              {
                "album":"<track album >",
                "artist":"<track artist>",
                "genre":<track genre>,
                "id":<track id>,
                "numberTrack":<track number>,
                "song_updated_at":<datetime field>,
                "time":<time in seconds>,
                "title":<track title>,
                "user_id":<user id>
              }
}
```

Errors

If the token is missing or invalid or if the user does not own the song, a « 403 Forbidden » type response will be returned (in the following example the token is missing):

```
{
  success : false,
  error : "invalid token"
}
```

2. Obtain the music of the user

This request allows you to get the music list of the user.

Request

GET /api/tracks

Response

If the token is valid, a “ 200 OK ” will be returned:

```
{
  "success":true,
  "list":[ <Music>, <>...]
}
```

Errors

If the token is missing or invalid, a « 403 Forbidden » type response will be returned. (In the following example the token is missing):

```
{
  success : false,
  error : "invalid token"
}
```

3. Adding a song

This request allows you to add a song and song related information.

Request

POST /api/tracks/

The request takes relative information to the music in parameters:

```
{
  "tracks":
             [
              {
                "album":<track album (string)>,
                "artist":<track artist (string)>,
                "genre":<track genre (string)>,
                "numberTrack":<track number (int)>,
                "time":<track duration in seconds (int)>,
                "title":<track title (string REQUIRED)>
              }, <autre track>, ...
            ]
}
```

Response

If the token is valid and at least one song is added, a « 201 Created » type response will be returned:

```
{
  "success":true,
  "message":"Tracks have been created"
}
```

Errors

If the token is missing or invalid, a « 403 Forbidden » type response will be returned (In the following example the token is missing) :

```
{
  success : false,
   error : "invalid token"
```

}

4. Updating a song information

This request allows you to update information of a song.

Request
PUT /api/tracks/<track_id>

The request takes for parameters the fields that can be updated (A more detailed list is available in the Music section):
```
{
    "title":"New Title"
}
```

Response
If the token and the ID are correct and if the user owns the song, the music informations will be updated and the response message will be :
```
{
  "success":true,
  "message":"Track information have been updated."
}
```

Errors
If the token is missing, invalid or if the user does not own the song, a « 403 Forbidden » will be returned (in the following example, the token is missing) :
```
{
  success : false,
  error : "invalid token"
}
```

If the ID in the parameter field does not match with any songs, a « 404 Not Found » will be returned :
```
{
  "success":false,
  "error":"The track cannot be found"
}
```

5. Delete a song from the library

This request allows you to delete a song from a library.

Request
DELETE /api/tracks/<track_id>

Response

If the token is valide and if the song has been deleted, a "200 OK" type response will be returned.

```
{
 "success":true,
 "message":"Track has been deleted."
}
```

Errors

If the token is missing or invalid, a « 403 Forbidden » type response will be returned (In the following example, the token is missing) :

```
{
  success : false,
  error : "invalid token"
}
```

If the song doesn't exist or doesn't belong to the user, a « 404 Not Found » type response will be returned.

6. Delete one or more songs from the library

This request allows you to delete one or multiple songs.

Request

DELETE /api/tracks ?tracks_id[]=<id 1>&tracks_id[]=<id n>
POST /api/tracks/delete

If the POST method is used, a song ID list must be passed into the parameters to delete the songs.

```
{
  "tracks_id":[<id 1>, <id 2>, <id n>]
}
```

Response

If the token is valid, a "200 OK" type response will be returned, even if the music IDs were invalid.

```
{
 "success":true,
 "message":"Elements have been deleted."
}
```

Errors
If the token is missing or invalid, a « 403 Forbidden » type response will be returned (In the following example, the token is missing) :

```
{
  success : false,
  error : "invalid token"
}
```

3. Playlist

1. Creating a playlist

Request
POST /api/playlist
The “title” attribute must be of minimum 5 characters.

```
{
  "title":<Playlist name>
}
```

Response
If the parameters are invalid, a « 201 Created » type response will be returned:

```
{
 "success":true,
 "message":"Playlist has been created",
 "id":   <id de la liste de lecture>
}
```

Errors
If the token is missing or invalid, a « 403 Forbidden » type response will be returned. (In the following example, the token is missing) :

```
{
  success : false,
  error : "invalid token"
}
```

If the parameters are invalid, a « 400 Bad Request » response will be returned. It will contain details about the errors:

```
{
   success: false,
   error: "title has already been taken"
}
```

2. Modify a playlist

Request

PUT /api/playlist/<id>

```
{
  "title":<nom de la piste de lecture>
}
```

Response

If the parameters are valid, a « 200 Ok » type reponse will be returned :

```
{
 "success":true,
 "message":"Playlist has been created"
}
```

Errors

If the token is missing or invalid, a « 403 Forbidden » type response will be returned (in the following example, the token is missing):

```
{
  success : false,
  error : "invalid token"
}
```

If the parameters are invalid, a « 400 Bad Request » type response will be returned containing the error detail :

```
{
   success: false,
   error: "title has already been taken"
}
```

3. Delete a playlist

Request

DELETE /api/playlist/<id>

Response

If the playlist exists and belongs to the user, a « 200 Ok » type response will be returned:

```
{
 "success":true,
 "message":"Playlist has been deleted"
}
```

Errors
If the token is missing or invalid, a « 403 Forbidden » type response will be returned (in the following example, the token is missing) :
```
{
  success : false,
  error : "invalid token"
}
```

If the playlist doesn't exist or doesn't belong to the user, a « 404 Not Found » type response will be returned:
```
{
   success: false,
   error: " Playlist does not exist"
}
```

4. Delete multiple playlists

Request
DELETE /api/playlist/ ?playlist_ids[]=<id 1>&playlist_ids[]=<id n>
POST /api/playlist/delete

If the POST method is used, the following parameters are mandatory:
```
{
  "playlist_ids":[<id 1>, <id 2>, <id n>]
}
```

Response
If at least one playlist exists and belongs to the user, a « 200 Ok » type response will be returned:
```
{
 "success":true,
 "message":"Playlist has been deleted"
}
```

Errors
If the token is missing or invalid, a « 403 Forbidden » type response will be returned (In the following example, the token is missing):
```
{
  success : false,
  error : "invalid token"
}
```

If none of the playlist exist or belong to the user, a « 404 Not Found » type response will be returned:

```
{
   success: false,
   error: " Playlist does not exist"
}
```

5. Adding multiple songs to a playlist

Request

It’s possible to add a song to a playlist either by giving its ID or by giving information on the song. In the second case scenario, the song will be added to the library then added to the playlist.

POST /api/playlist/<id>/tracks

```
{
  "tracks_id" : [<track_id 1>, <track_id 2>, <track_id n>]
  "tracks" : [<track 1>, <track 2>, <track n>]
}
```

Response

If the parameters are invalid, a « 201 Created » type response will be returned:

```
{
 "success":true,
  "message": "Tracks have been added to the playlist"
}
```

Errors

If the token is missing or invalid, a « 403 Forbidden » type response will be returned (In the following example the token is missing) :

```
{
  success : false,
  error : "invalid token"
}
```

If at least a song doesn’t exist or doesn’t belong to the user, a « 404 Not Found » type response will be returned:

```
{
   success: false
   error: "Couldn't find track <track_id>"
}
```

If no songs are in the parameters, a « 400 Bad Request » type response will be returned:

```
{
  success : false,
   error : "No track found in params"
}
```

If the playlist doesn't exist or doesn't belong to the user, a « 404 Not Found » type response will be returned:
{
  success : false,
   error : "Playlist does not exist"
}

6. Delete multiple songs in a playlist

Request

**DELETE /api/playlist/<id>/tracks ?tracks_id[]=<track_id 1>&
tracks_id[]=<track_id n>
POST /api/playlist/<id>/tracks/delete**

If the POST method is used, the following parameters are mandatory:
{
  "tracks_id" : [<track_id 1>, <track_id 2>, <track_id n>]
}

Response

If the parameters are valid, a « 200 Ok » type response will be returned:
{
 "success":true,
 "tracks": "<n> tracks have been deleted from the playlist",
 "nb_tracks": <new number of musics in the library>,
 "nb_deleted": <number of songs removed from the playlist>,
}

Errors

If the token is missing or invalid, a « 403 Forbidden » type response will be returned (In the following example the token is missing) :
{
  success : false,
  error : "invalid token"
}

If the playlist doesn't exist, a « 404 Not Found » response will be returned:
{
  success : false,
  error : " Playlist does not exist"
}

If none of the songs exist in the playlist, a « 404 Not Found » response will be returned:
```
{
  success : false,
  error : "0 track has been deleted from the playlist"
}
```

7. Delete a song from a playlist

Request

DELETE /api/playlist/<playlist_id>/tracks/<track_id>

Response

If the parameters are invalid, a « 200 Ok » response will be returned:
```
{
 "success":true,
 "tracks":  " Track <track_id> has been deleted from the playlist",
 "nb_tracks":  <nouveau nombre de musique dans la liste de musique>,
}
```

Errors

If the token is missing or invalid, a « 403 Forbidden » type response will be returned (in the following example the token is missing) :
```
{
  success : false,
  error : "invalid token"
}
```

If the playlist doesn't exist, a « 404 Not Found » response will be returned:
```
{
  success : false,
  error : " Playlist does not exist"
}
```

If none of the songs exist in a playlist, a « 404 Not Found » response will be returned:
```
{
  success : false,
  error : "0 track were deleted from the playlist"
}
```

8. Get songs in a playlist

Request

GET /api/playlist/<id>/tracks

Response
If the token is valid, a « 200 Ok » response will be returned:
```
{
 "success":true,
 "id": <playlist id>
 "tracks":[<Track 1>, <Track 2>, <Track n>]
}
```

Errors
If the token is missing or invalid, a « 403 Forbidden » response will be returned (In the following example the token is missing) :
```
{
  success : false,
  error : "invalid token"
}
```

9. List playlists

Request
GET /api/playlist

Response
If the token is valid, a « 200 Ok » response will be returned:
```
{
 "success":true,
 "list":[<playlist 1>, <playlist 2>, <playlist n>]
}
```

Errors
If the token is missing or invalid, a « 403 Forbidden » response will be returned (in the following example the token is missing) :
```
{
  success : false,
  error : "invalid token"
}
```

10. Print the details of a playilst

Request
GET /api/playlist/<id>

Response
If the token is valid and the playlist belongs to a user, a « 200 Ok » response will be returned:
```
{
  success: true
  playlist:
            {
              id: <id>
              nb_tracks: <nombre de musique dans la liste de lecture>
              title: <nom de la liste de lecture>
              user_id: <id de l'utilisateur possédant la liste de lecture>
            }
}
```

Errors

If the token is missing or invalid, a « 403 Forbidden » type response will be returned (in the following example the token is missing) :

```
{
  success : false,
  error : "invalid token"
}
```

If the playlist doesn't exist or doesn't belong to the user, a « 404 Not Found » response will be returned:

```
{
  success : false,
  error : "Playlist does not exist"
}
```

4. Instant messaging

The chat between users is implemented with a server compatible with Jabber and XMPP. You need to use a compatible library to implement it.

The server's URL is https://audiowire.co and the server uses standard ports. AudioWire's users automatically have an account on the Jabber server.

Friends adding and deletion must be done using the API and not the Jabber server directly. The API will automatically update the Jabber server.

The username to use to connect on the Jabber server is <username>@audiowire.co and the password is the same as the AudioWire's account password.

It is not possible to update the Jabber password directly. The password and all the information stored on the Jabber server are synchronized with the AudioWire server's information. Therefore, you need to use the API to update these information.

The Jabber server's admin interface is available at http://audiowire.co:9090. Please contact an administrator to get an access.