FOXWEBALARMPLAYER

DOCUMENTATION

© 2004-2016 Foxstream SAS

**Document name: FoxWebAlarmPlayer documentation**

**Versions**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Description** |
| 0.1 | 17/08/2016 | Nicolas OTTEN | Document structure and API documentation |
| 0.2 | 27/08/2016 | Nicolas OTTEN | Translation and server description |
|  |  |  |  |
|  |  |  |  |

**SOMMAIRE**

1. General 4
   1. Deployment 4
2. Server 5
   1. Architecture 5
      1. UML diagram 5
      2. Description of the objects 7
   2. API 8
      1. Alarms 8
      2. Users 10
      3. Servers 13
      4. Live 15
3. Clients 16
   1. Translation 16

Update translation files 16

# General

## Deployment

The following gulp command can be run in the root of the application to create a ready-to-install directory :

gulp release

The command will concatenate and minify all the .scss and .js files for the front-end and create a release/ directory that can be placed on the production server. The release folder does not contain the node\_modules directory, as these dependencies are platform-specific and should be installed directly on the server. To do so, the following command should be run :

npm install

If successful, the application can then be started with this command :

npm start

# Server

## Architecture

### UML diagram

The following diagram is designed to give a quick idea of how the different JavaScript objects communicate on the server. Please note that for the sake of clarity, all the objects and JavaScript events are not shown on this diagram (for a complete description of the objects and their roles, see the next section).

### Description of the objects

#### Main

Entry point of the application. It creates all the required objects (WebServer, persistence objects, controllers, …) and calls some of their methods to get the application running (creating HTTP routes, starting serversManager, starting alarmRemover, …)

#### WebServer

Creates an Express web server listening on the port specified in the config files. This object is also responsible for creating the main HTTP routes of the application, such as the ones that deliver the HTML pages (routes designed for more specific actions are declared in the adequate controllers).

#### AlarmPersistence, ServerPersistence, UserPersistence

These objects are the only ones to handle database operations to manage the lists of alarms, users and servers.

The AlarmPersistence object keeps track of al the sites and cameras for which alarms have been received in two arrays (siteList and cameraList). This is useful to create filters for the alarms.

#### AlarmController, UserController, …

The controllers act as an interface between the clients and the database (through the persistence layer), by responding to HTTP routes they add to the web server (see section asdf for the complete description of the API).

#### FoxXmlClient

Stores the configuration of a server. Maintains a permanent connection to the server by sending ping messages every ten seconds.

#### XmlClient

The interface between a server and the corresponding FoxXmlClient object: it is responsible for sending and receiving XML messages.

#### ServersManager

As any modification of information on a server should not only update the database, but create, delete or update the corresponding FoxXmlClient object too, the ServerController object doesn’t communicate with the ServerPersistence directly. ServersManager acts as an interface between ServerController and ServerPersistence, and maintains the list of servers. It is also responsible for saving new alarms in the database.

#### AlarmRemover

Periodically removes old alarms from the database and their corresponding image files from the server/data directory.

## API

### Alarms

|  |  |
| --- | --- |
| GET | /alarms |

|  |  |
| --- | --- |
| **Description** | Returns the list of all alarms matching the conditions passed as a parameter.  Possible conditions are :   * date (only alarms raised during the 24 hours following this timestamp will be selected) * camera * site |
| **Access** | All users |
| **Parameters (data)** | Object { [date: timestamp[, site: String[, camera: String] ] ] } |
| **Status code** | 200 (Success), 400 (Bad request, si les paramètres fournis sont mal formatés), 500 (Internal Server Error) |
| **Response data** | * Array [ { id: Number, timestamp: Number, cameraname: String, hostname: String, sitename: String, handled: Number, nbimages: Number} ] if successful * Error message otherwise |

|  |  |
| --- | --- |
| GET | /alarms/sitelist |

|  |  |
| --- | --- |
| **Description** | Returns the list of sites for which alarms have been saved in the database. This is especially useful to expose filters on the client side. |
| **Access** | All users |
| **Status code** | 200 (Success), 500 (Internal Server Error) |
| **Response** | Array [String] |

|  |  |
| --- | --- |
| GET | /alarms/cameralist |

|  |  |
| --- | --- |
| **Description** | Returns the list of cameras for which alarms have been saved in the database. |
| **Access** | All users |
| **Status code** | 200 (Success), 500 (Internal Server Error) |
| **Response** | Array [Object { cameraname: String, sitename: String }] |

|  |  |
| --- | --- |
| GET | /alarms/:alarmid |

|  |  |
| --- | --- |
| **Description** | Returns information on a specific alarm. |
| **Access** | All users |
| **Parameters (url)** | alarmid : Number |
| **Status code** | 200 (Success), 204 (No Content), 500 (Internal Server Error) |
| **Response data** | * Object { id: Number, timestamp: Number, cameraname: String, hostname: String, sitename: String, handled: Number, nbimages: Number} if successful * Empty (along with status code 204) if **alarmid** doesn’t match an existing alarm in database. * Error message otherwise |

|  |  |
| --- | --- |
| GET | /alarms/:alarmid/image/:imgid |

**Redirects to /alarms/:alarmid/image/:imgid/jpg**

|  |  |
| --- | --- |
| GET | /alarms/:alarmid/image/:imgid/jpg |

|  |  |
| --- | --- |
| **Description** | Returns a specific image recorded for an alarm. |
| **Access** | All users |
| **Parameters (url)** | alarmid : Number, imgid : Number |
| **Status code** | 200 (Success), 404 (Not Found), 500 (Internal Server Error) |
| **Response data** | * If successful, the image encoded with base64 * Error message if an error occurred while reading the file |

|  |  |
| --- | --- |
| GET | /alarms/:alarmid/image/:imgid/osd |

|  |  |
| --- | --- |
| **Description** | Return the OSD (On-Screen Display) for a specific alarm image |
| **Access** | All users |
| **Parameters (url)** | alarmid : Number, imgid : Number |
| **Status code** | 200 (Success), 404 (Not Found), 500 (Internal Server Error) |
| **Response data** | * Array [ Array [ Object {x: Number, y: Number} ] ] * Error message if an error occurred while reading the file |

|  |  |
| --- | --- |
| PUT | /alarms/:alarmid/handled |

|  |  |
| --- | --- |
| **Description** | Mark an alarm as handled. |
| **Access** | All users |
| **Parameters (url)** | alarmid : Number |
| **Status code** | 200 (Success), 404 (Not Found), 500 (Internal Server Error) |
| **Response data** | * Empty if successful * Error message otherwise |

### Users

#### General notes

All users have a type: administrator (type = 1) or normal (type = 0).

A user is the only one who can change his own password. However, an administrator can reset the password of any other user. The latter will then be able to log in with an empty password (he should therefore be asked to choose a new one after his next connection).

Once the user is logged in, a **user** object is passed in the **req** parameter of every HTTP request, containing his account information such as login, displayname and type. This object also contains a field called “shouldChangePassword”, which is set to 1 if the user’s password is empty.

#### Routes

|  |  |
| --- | --- |
| GET | /users |

|  |  |
| --- | --- |
| **Description** | Returns the list of registered users. |
| **Access** | Administrators only |
| **Status code** | 200 (Success), 500 (Internal Server Error) |
| **Response** | * Array[Object] : [ { id: Number, username: String, … }, {…}, … ] if successful * Error message if necessary |

|  |  |
| --- | --- |
| GET | /users/me |

|  |  |
| --- | --- |
| **Description** | Returns information on the user who sent the request |
| **Access** | All users |
| **Status code** | 200 (Success), 500 (Internal Server Error) |
| **Response** | * Object : {username: String, displayname: String, …} * Error message if necessary |

|  |  |
| --- | --- |
| GET | /users/:userid |

|  |  |
| --- | --- |
| **Description** | Returns information on a specific user |
| **Access** | Administrators only |
| **Parameters (URL)** | **userid** : Number |
| **Status code** | 200 (Success), 204 (No Content), 500 (Internal Server Error) |
| **Response data** | * Object : {username: String, displayname: String, …} * Empty (along with status code 204) if **userid** doesn’t match an existing user in database. * Error message if necessary |

|  |  |
| --- | --- |
| PUT | /users/:userid |

|  |  |
| --- | --- |
| **Description** | Updates a specific user (except password) |
| **Access** | Administrators only |
| **Parameters (URL)** | **userid** : Number |
| **Parameters (data)** | Object { id: Number, displayname: String, type: Number } |
| **Status code** | 200 (Success), 400 (Bad request, in case the object passed as a parameter doesn’t have the right format), 404 (Not Found), 500 (Internal Server Error) |
| **Response data** | * Empty if successful * Error message if necessary |

|  |  |
| --- | --- |
| PUT | /users/me/displayname |

|  |  |
| --- | --- |
| **Description** | Enables users to update their displayed name |
| **Access** | All users |
| **Parameters (data)** | Object { displayname: String } |
| **Status code** | 200 (Success), 400 (Bad request, si l’objet passé en paramètre est mal formé), 500 (Internal Server Error) |
| **Response data** | * Empty if successful * Error message if necessary |

|  |  |
| --- | --- |
| PUT | /users/me/password |

|  |  |
| --- | --- |
| **Description** | Enables users to update their password |
| **Access** | All users |
| **Parameters (data)** | Object { oldPassword: String, newPassword: String }. |
| **Status code** | 200 (Success), 400 (Bad request, si l’objet passé en paramètre est mal formé), 500 (Internal Server Error) |
| **Response data** | * Empty if successful * Error message if necessary |

|  |  |
| --- | --- |
| POST | /users |

|  |  |
| --- | --- |
| **Description** | Save a new user in the database |
| **Access** | Administrators only |
| **Parameters (data)** | Object {login: String, displayname: String, type: Number} |
| **Status code** | 200 (Success), 400 (Bad request) in case the object passed as a parameter doesn't have the right format, 500 (Internal Server Error) |
| **Response data** | * Object { id: Number, login: String, … } : object representing the new user if successful * Error message otherwise |

|  |  |
| --- | --- |
| POST | /users/:userid/passwordReset |

|  |  |
| --- | --- |
| **Description** | Resets the password of a specific user |
| **Access** | Administrators only |
| **Parameters (url)** | **userid :** Number |
| **Status code** | 200 (Success), 404 (Not Found), 500 (Internal Server Error) |
| **Response data** | * Empty if successful * Error message (along with status code 404) if **userid** doesn’t match an existing user in the database * Error message otherwise |

|  |  |
| --- | --- |
| DELETE | /users/:userid |

|  |  |
| --- | --- |
| **Description** | Deletes a user |
| **Access** | Administrators only |
| **Parameters (url)** | **userid :** Number |
| **Status code** | 200 (Success), 404 (Not Found), 500 (Internal Server Error) |
| **Response data** | * Empty if successful * Error message (along with status code 404) if **userid** doesn’t match an existing user in the database * Error message otherwise |

### Servers

|  |  |
| --- | --- |
| GET | /servers |

|  |  |
| --- | --- |
| **Description** | Returns the list of registered servers. |
| **Access** | All users |
| **Status code** | 200 (Success), 500 (Internal Server Error) |
| **Response** | * Array[Object] : [ { id: Number, address: String, … }, {…}, … ] * Error message if necessary |

|  |  |
| --- | --- |
| GET | /servers/:serverid |

|  |  |
| --- | --- |
| **Description** | Returns information on a specific server |
| **Access** | All users |
| **Parameters (URL)** | **serverid** : Number |
| **Status code** | 200 (Success), 204 (No Content), 500 (Internal Server Error) |
| **Response data** | * Object : {address: String, description: String, …} * Empty (along with status code 204) if **serverid** doesn’t match an existing server in the database. * Error message if necessary |

|  |  |
| --- | --- |
| PUT | /servers/:serverid |

|  |  |
| --- | --- |
| **Description** | Updates a specific server |
| **Access** | Administrators only |
| **Parameters (URL)** | **serverid** : Number |
| **Parameters (data)** | Object { address: Number, description: String… } |
| **Status code** | 200 (Success), 400 (Bad request, in case the object passed as a parameter doesn’t have the right format), 404 (Not Found), 500 (Internal Server Error) |
| **Response data** | * Empty if successful * Error message if necessary |

|  |  |
| --- | --- |
| POST | /servers |

|  |  |
| --- | --- |
| **Description** | Save a new server in the database |
| **Access** | Administrators only |
| **Parameters (data)** | Object {address: String, description: String, …} |
| **Status code** | 200 (Success), 400 (Bad request) in case the object passed as a parameter doesn't have the right format, 500 (Internal Server Error) |
| **Response data** | * Object { id: Number, address: String, … } : object representing the new server if successful * Error message otherwise |

|  |  |
| --- | --- |
| DELETE | /servers/:serverid |

|  |  |
| --- | --- |
| **Description** | Deletes a server |
| **Access** | Administrators only |
| **Parameters (url)** | **serverid** |
| **Status code** | 200 (Success), 404 (Not Found), 500 (Internal Server Error) |
| **Response data** | * Empty if successful * Error message (along with status code 404) if **serverid** doesn’t match an existing server in the database * Error message otherwise |

### Live

|  |  |
| --- | --- |
| GET | /cameras |

|  |  |
| --- | --- |
| **Description** | Returns the list of available cameras. |
| **Access** | All users |
| **Status code** | 200 (Success), 204 (No Content) |
| **Response data** | Object { site1: [ {id: Number, name: String, …}, …], site2: […] } |

|  |  |
| --- | --- |
| GET | /cameras/:server/:camid/live |

|  |  |
| --- | --- |
| **Description** | Sends a live image from a specific camera |
| **Access** | All users |
| **Parameters (URL)** | server: Number id of the server the camera is connected to  camid: Number id of the camera |
| **Status code** | 200 (Success), 500 (Internal Server Error) |
| **Response data** | The live image encoded in base64 |

# Clients

## Translation

The client side of the application must be available in several languages. To achieve internationalisation, every displayed text passes through a **translate** Angular filter.

Any piece of text that will be displayed in several languages should be identified by a unique key, containing only uppercase letter and underscores (ex: ERROR\_MESSAGE).

All the used keys should then be referenced in the **locale-\*** file of each supported language, located in the client/locale directory. These files contain a JSON representation of all the keys and their translations :

**File : locale-en.json**

{

…

‘ERROR\_MESSAGE’: “An error occured”,

…

}

To display translated text in **pug** views, the following syntax should be used :

**span** {{ ‘ERROR\_MESSAGE’ | translate }}

If a translation is needed inside a controller, the following syntax should be used:

$translate.instant(‘ERROR\_MESSAGE’)

## Update translation files

The script **update-locale-files.js** can be run in the client/ directory to automatically update the translation files :

FoxWebAlarmPlayer/client> node update-locale-files.js

The script will read all the views (.pug files) and all the controllers to find all the translation keys that are being used in the application. It will then check the locale-\* files and add any missing keys.