FileMiner API

Contents

About

The FileMiner API is the only one interface for FileMiner app. This document is designed for helping to create client apps and includes describing information about all features and functionality.

Server functions

Files and metadata:

/upload_file

Description	Uploads a file.
URL Structure	/upload_file
Method	POST
Request body	*required* the file contents to be uploaded.
Parameters	Upload – the file to be uploaded
Returns	The HTTP response with uploading result.
Errors	400 One of the fields is empty or wrong 500 Server error
Notes	The method is used for uploading files. There Is no support for large files loading, be careful. Uploading works only for new files. If you will try to upload file which was uploaded before, you will get error 400.

/update_file

Description	Overwrites file created before.
URL Structure	/update_file
Method	POST
Request body	*required* the file contents to be uploaded.
Parameters	Upload – the file to be uploaded
Returns	The HTTP response with uploading result.
Errors	400 One of the fields is empty or wrong 404 File is not found 500 Server error
Notes	The method is used for updating files which was uploaded before. If you will try to update file without uploading first by “upload_file”, you will get error 404. There Is no support for large files loading, be careful.

/get_file_data

Description	Download file.
URL Structure	/get_file_data
Method	POST
Parameters	File_name – the name of file to be downloaded
Returns	The standart HTTP response with file result.
Errors	400 One of the fields is empty or wrong 404 The specified file is not loaded. Upload the file before other actions. 500 Server error
Notes	The method is used for downloading files. There Is no support for large files loading, be careful.

/get_file_metadata

Description	Downloads metadata of the file.
URL Structure	/get_file_metadata
Method	POST
Parameters	File_name – the target file
Returns	The HTTP response with result in free form. Example body of the response: *<body>* *File metadata: Example1.txt{ "Change date" "time.struct_time(tm_year=2016, tm_mon=12, tm_mday=14, tm_hour=18, tm_min=15, tm_sec=54, tm_wday=2, tm_yday=349, tm_isdst=0)"* *"protection bits" "33188"* *"inode number" "654172"* *"device" "2049"* *"number of hard links" "1"* *"user id of owner" "1000"* *"group id of owner" "1000"* *"size of file in bytes" "177"* *"time of most recent access" "1481721497.44"* *"time of most recent content modification" "1481721354.34"* *"platform dependent" "1481721354.34"}* *</body>*
Errors	400 One of the fields is empty or wrong 500 Server error
Notes	The method is used for getting files metadata.

/get_files_list

Description	Getting list of all allowed files
URL Structure	/get_files_list
Method	POST
Parameters	No params
Returns	The HTTP response with list of the files. Example body of the response: *<body>* *Files list:* *"Example1.txt"* *"Example2.txt"* *</body>*
Errors	500 Server error
Notes	The method is used for getting list of all allowed files.

/login

Description	Manual testing interface
URL Structure	/login
Method	GET,POST
Parameters	No params
Returns	The HTTP response with list of the FileMiner's functions.
Errors	
Notes	The method is used for getting list of all allowed functions by FileMiner. Also it is possible start manual check of the function execution.

Error handling

Errors are returned using standard HTTP error code syntax. Any additional info is included in the body of the return call.

Example:

```
<body>
    <h1>Error: 400 Bad Request</h1>
    <p>Sorry, the requested URL <tt>&amp;#039;http://localhost:8080/update_file&amp;#039;</tt> caused an error:</p>
    <pre>Field file is empty</pre>
</body>
```

Standard API errors

Code	Description
400	Bad input parameter. Error message should indicate which one and why.
404	File or folder not found at the specified path.
500	Server error. Ask developer about the reason.

Development plans

- Optimization

It is planned to optimize upload and download files to make memory usage more effective. Files will be load in memory partially. Load in memory from source will continue when previous chunk uploaded to the reciver. It will make possible to work with real big files.

To prevent any conflicts with concurrent access it is planned to make a database. All actions that can cause conflicts will be checked before start for using in other process. After start of the process information will be recorded in database to make checks possible. At the end of the work transaction about it will be deleted. Also on this database will be created algorithm which will limit count of requests by client or/and ip.

- Security

To make work with FileMiner more effective will be created database

- to fix and determinate limits for all files size
- to check and limit for uploading and downloading in directories
- to make authorization
- to save information which can be helpful for restore forgotten login and password
- to bind files to the creator
- to create opportunity for choosing users which can read and write files of creator

Implementation features

API made on “python 2.7” with using library “bottle.py”. Manual tested in PyCharm with integrated webserver.