The smart contract consists of two parts. The first part defines structures of 2 multi-index tables. The second part defines two actions: *getfile* and *addfile*.

Actions:

The ***addfile*** action takes as input parameters the metadata of a file uploaded to the IPFS network and the name of the user's blockchain account. After authorization, it writes the parameters to the multi-index table *‘ipfsstore’* and adds the entire operation in the multi-index table *‘activityreg’*.

[[eosio::action]] addfile\_return addfile (name creator, string filename, string hash, uint64\_t size, string description, eosio::asset price, bool is available)

input parameters:

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| creator | name | file’s creator account name |
| filename | string | filename without path (to be stored after downloading from ipfs with the correct filename) |
| Hash | string | Ipfs hash of the file |
| Size | unit64\_t | File size (for checking purposes) |
| description | string | File description in common language |
| Price | asset | Price of the file. Optional for future development |
| is\_available | bool | An indicator of whether the file is accessible (for example, if it needs to be backed up or is not visible) |

Return parameters:

|  |
| --- |
| struct addfile\_return {uint64\_t id; eosio::time\_point\_sec timestamp;}; |

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| Id | unit64\_t | The unique id of the entry in the multi-index table to be searched for later use. it can be stored in an external database. |
| timestamp | time\_point\_sec | The transaction’s timestamp |

The ***getfile*** action takes as input parameters the name of the user's blockchain account and the unique key of the searched file, contained in the multi-index table *‘ipfsstore’*. This key is received in advance in the App, through searching the catalogue of files available and selecting one of them. The catalogue can be retrieved from external database or, by extracting data from the ‘*ipfsstore’* multi-index table on the blockchain. After authentication, the action sends a payment from the user's account to the file publisher's account (optional). If the payment is successful smart contract returns to the App the hash of the selected file and adds the entire operation in the multi-index table *‘activityreg’* in the blockchain.

|  |
| --- |
| [[eosio::action]] getfile\_return getfile (name actor, uint64\_t id) |

input parameters:

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| Actor | name | User’s account name (authority/permission check). |
| Id | unit64\_t | The unique id of the entry in the multi-index table. |

return parameters:

|  |
| --- |
| struct getfile\_return {string hash; string filename; uint64\_t size; eosio::time\_point\_sec timestamp;}; |

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| Hash | string | Hash of the file for further use in IPFS. |
| filename | string | Name of the file used for local store operation. |
| Size | unit64\_t | File size (for checking purposes) |
| timestamp | time\_point\_sec | The timestamp parameters when the file is uploaded |

**Multi-index tables**

IPFS files registry:

|  |
| --- |
| TABLE store {  uint64\_t id;  std::string filename;  std::string hash;  uint64\_t size;  name created\_by;  eosio::time\_point\_sec created\_at;  eosio::asset price;  bool is\_available;  string description;  uint64\_t primary\_key() const { return id; }  uint64\_t third\_key() const { return created\_at.sec\_since\_epoch(); }  uint64\_t available\_key() const {  uint64\_t ret = 1;  if (!is\_available) { ret = 0; };  return ret;  }  }; |

Activity register:

|  |
| --- |
| TABLE activity {  uint64\_t uid;  name actor;  time\_point\_sec timestamp;  uint64\_t act; // 1 - add file, 2 - get file  uint64\_t object\_id;  uint64\_t primary\_key() const { return uid; }  uint64\_t time\_key() const { return timestamp.sec\_since\_epoch(); }  uint64\_t action\_key() const { return act; }  uint64\_t object\_key() const { return object\_id; }  uint64\_t actor\_key() const { return actor.value; }  }; |