Following the rule of limited trust not only changes a general approach to everyday life but also forces different branches of business to undertake additional actions and introduce procedures to make sure that the particular subject matter is well-defined and not modified without the knowledge of the other party. It affects, in particular, the cases when a reasonable suspicion or a doubt about the delivered data correctness occur, which is the matter of exchange. Among such examples are downloaded files, sent documents agreements, or regulations.

WHO IS AFFECTED (POLISH SCENE):

In Poland, the Consumer Rights Act, which came into force on the 25th of December 2014, imposes the information obligation on entrepreneurs in contracts concluded remotely or outside the company's premises. The Act identifies various ways of passing important information to the consumer (understood as communication). Such information can be written on paper or any other **durable medium** or forwarded in a way corresponding to the type of applied communication means. This law also introduces a new definition, which is a "durable medium". Simply put, it is any substance or tool allowing a consumer or an entrepreneur to store information dedicated and addressed strictly to him/her, and which can be accessed in the future as long as it is needed (timespan provided by law) in an unmodified form. In practice, we can consider the following things as durable media:

- paper,
- CD/DVD,
- pen drive,
- memory card,
- any other hard drive.

But also services, such as:

- email,
- SMS.

Recently, a solution based on the **blockchain technology** was introduced, deployed, and implemented as a customer service by PKO BP. It turned out that their unique approach is based on the blockchain technology, where nodes are represented by the bank itself and the KIR (National Clearing House). Such a software solution has many merits stemming from the applied technology, but also, we can assume that it could help to reduce the costs of printing documents on paper or burning CD/DVDs including the costs of a production process, implementation, and maintenance. A similar approach is planned by Alior Bank. These examples aren't the only ones embracing the problem of storing files and making sure that they remain unchanged, but they indicate possible challenges which may arise for entrepreneurs, institutions, parties, state offices, or even governments. Below, there is a list of different domains which may require some support in case of the authorized files:

- online bookstores selling digitally signed electronic versions of books,
- authorship,
- notarial acts or ownerships,
- marketing consents,
- privacy policy.

PROPOSED SOLUTION:

Before you jump in at the deep end and dive into the possibilities of blockchain, it is good to think things through. Applying blockchain technology guarantees immutability of the saved data, but apart from all the vast promises it also has its flaws which one should carefully consider against business needs. However, once we take the blockchain as a durable medium into consideration, we should be aware that the best approach is to store only the basic information, not the files themselves. By "basic information" I mean such data as names, dates added, extensions, authors, or any other properties which are stored by a file system or help identify the file among others. And here we come to the point made before that the best way to distinguish the file (not only from other files but also from versions of the same file) is knowing its hash. It's another level of security which hides the content of the file behind a fixed size sequence of letters and digits. Any modification of this file, regardless of intentions, meaning, or size, causes a complete change of hash, which gives a proof that there is or there isn't any consistency between the requested file and the file stored. This proof is as an argument supporting the case.

Having answered what we should store in the blockchain, another question arises. Where should we keep these files in practice? So, as decentralisation increases, it is tempting to use one of the distributed file systems. They are designed to consolidate information and facilitate file sharing while providing remote access at a local-like level. The main characteristics of such systems are high availability, reliability, data integrity, scalability, and heterogeneity (distributed system looks the same for every device which can be a part of such a system). An example of a distributed file system that has been functioning for some time is the InterPlanetary File System known as IPFS. It is a system which synthesises successful ideas from previous peer-to-peer systems including BitTorrent or Git. It also provides a high throughput content-addressed block storage model, with content address hyperlinks. Its structure, built upon DAG (Directed Acyclic Graph), lets us create a versioned file system or even a permanent web. IPFS consists of hashing tables, incentivised blocks exchange, and self-certified file system.

As each of the system components, it also has some demerits or aspects requiring some additional consideration, i.e. need for additional data backup or the way of securing the file replication in the network. The second issue is especially worthy of consideration because IPFS itself doesn't have any automatic replications. Nodes only store and/or distribute content they are explicitly intended to store and/or distribute. Simply put, devices that run IPFS nodes don't have to host files that they are not designed for. But when it comes to the first issue, nodes can refer to the requested file and from now on have its local copy. For more details and explanations, please visit IPFS homepage and read their whitepaper (**https://raw.githubusercontent.com/ipfs/papers/master/ipfs-cap2pfs/ipfs-p2p-file-system.pdf**).

After such introduction, we can go straight to the technical details of the solution which is expected to solve described problems using the mentioned technologies.

TECHNICAL ASPECTS:

In this section, I present a simple implementation of storing file hash (SHA-256) in the private blockchain built upon ETH protocol keeping the file in a distributed file system – IPFS.

Technology stack:

- Truffle (development framework for dapps based on the Ethereum blockchain: https://truffleframework.com/),
- Ganache (one click, in-memory blockchain: https://truffleframework.com/ganache),
- Solidity (contract-oriented programming language for writing smart contracts: https://solidity.readthedocs.io/en/v0.4.24/),
- Web3.js (Ethereum JavaScript API: https://github.com/ethereum/web3.js/),
- NodeJS (JavaScript run-time environment: https://nodejs.org/en/),
- React (JavaScript library for building user interfaces: https://reactjs.org/).

ENVIRONMENT PREPARATION AND INSTALLATION TOOLS:

Disclaimer: the instruction is based on Windows OS, some of the instructions may require OS specific approach such as using *sudo* in Linux-based OSes.

Install IPFS or run it other way (read below):

1. Go to https://dist.ipfs.io/#go-ipfs and download IPFS for your platform.
2. Extract zip to a new folder.
3. Open Command-line Interface (CLI) and move pointer to the folder from the previous step.
4. Run in CLI:

```
instal.sh
```

5. Run in CLI:

```
ipfs init
```

6. Run in CLI:

```
ipfs bootstrap rm -all
```

(this step ensures creating local and private storage)

7. Run in CLI:

   ```
   1. ipfs daemon
   ```

   (allow access if Windows firewall requires your confirmation, and leave the process running)

After these steps, you should have IPFS node set up and running. For more information, please visit IPFS documentation, it is well-written and includes many available options. For your benefit, it is good to leave one node just as it is, but in real case scenario when a complete configuration is required, it would be better to add additional nodes to the IPFS network (different machines) using the command:

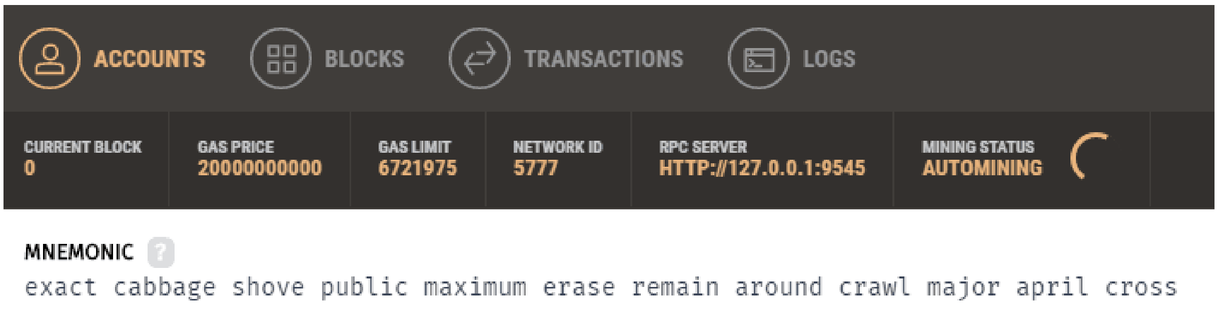
```
ipfs bootstrap add /ip4/<node_ip>/tcp/4001/ipfs/<hash_which_appears
after ipfs init_from_step5>
```

– after clearing bootstrap nodes in step 6. Our setup will use default values, for example, 127.0.0.1 as a server address and 5001 as its port.

There are other options to run the IPFS node such as docker containers or using Infura which is a hosted Ethereum node cluster that lets you run the application without requiring to set up your own Ethereum node or wallet but only for public network instances like Mainnet or Rinkeby.

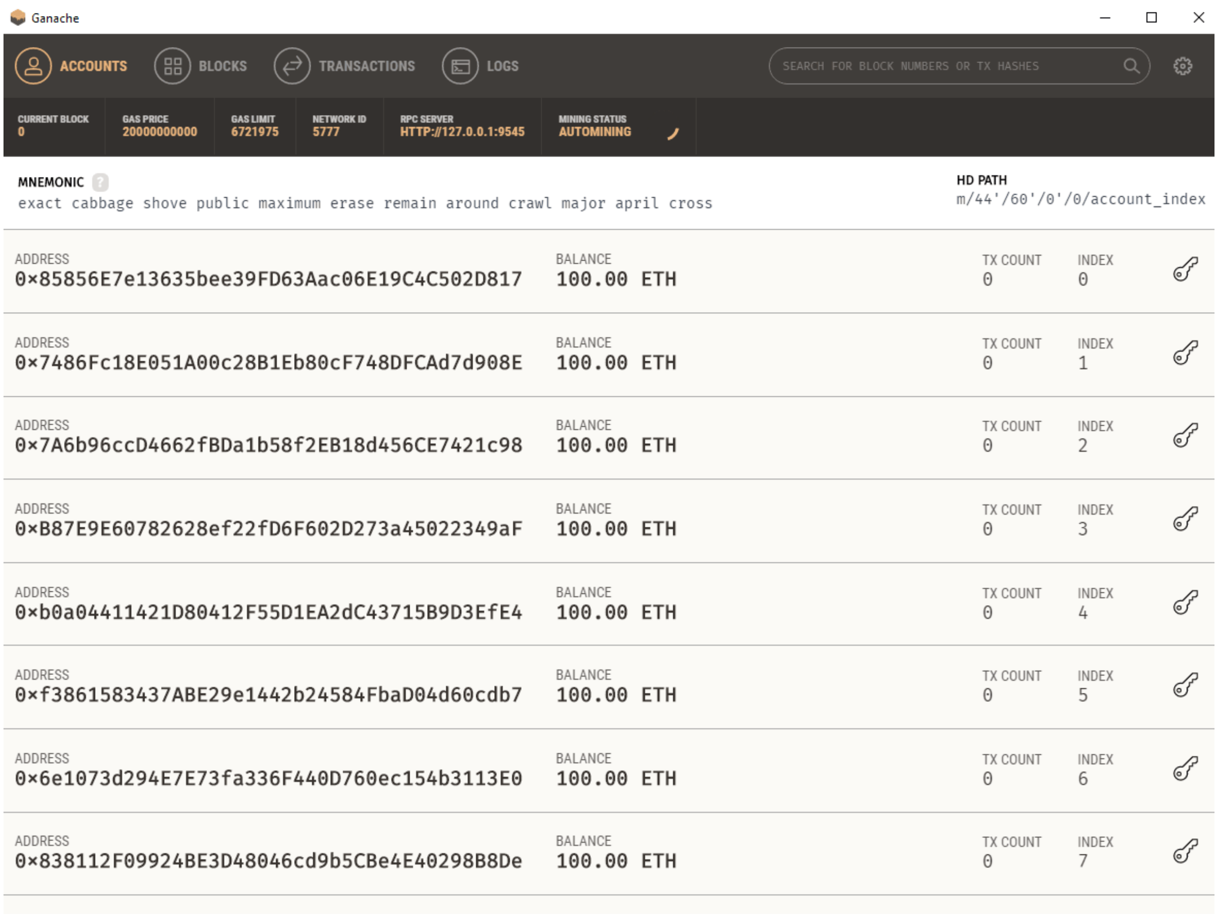
Install Ganache.

1. Go to https://truffleframework.com/ganache and download a version dedicated to your operating system.
2. Install by a double click, then run.
3. Ganache runs with default values which should be the same or similar to these on-screen. The crucial part is a section defining RPC Server.
4. Leave it running.



CURRENT BLOCK	GAS PRICE	GAS LIMIT	NETWORK ID	RPC SERVER	MINING STATUS
0	20000000000	6721975	5777	HTTP://127.0.0.1:9545	AUTOMINING

MNEMONIC
exact cabbage shove public maximum erase remain around crawl major april cross



CURRENT BLOCK	GAS PRICE	GAS LIMIT	NETWORK ID	RPC SERVER	MINING STATUS
0	20000000000	6721975	5777	HTTP://127.0.0.1:9545	AUTOMINING

MNEMONIC
exact cabbage shove public maximum erase remain around crawl major april cross

HD PATH
m/44'/60'/0'/0/account_index

ADDRESS	BALANCE	TX COUNT	INDEX
0×85856E7e13635bee39FD63Aac06E19C4C502D817	100.00 ETH	0	0
0×7486Fc18E051A00c28B1Eb80cF748DFCAd7d908E	100.00 ETH	0	1
0×7A6b96ccD4662fBDa1b58f2EB18d456CE7421c98	100.00 ETH	0	2
0×B87E9E60782628ef22fD6F602D273a45022349aF	100.00 ETH	0	3
0×b0a04411421D80412F55D1EA2dC43715B9D3EfE4	100.00 ETH	0	4
0×f3861583437ABE29e1442b24584FbaD04d60cdb7	100.00 ETH	0	5
0×6e1073d294E7E73fa336F440D760ec154b3113E0	100.00 ETH	0	6
0×838112F09924BE3D48046cd9b5CBe4E40298B8De	100.00 ETH	0	7

After these four steps, you have a local blockchain instance working with fast transaction confirmation and a pretty nice explorer which can help you check transactions in the mined blocks. If you want to check whether proposed blockchain is functional, you can connect to it with Metamask as to a custom RPC server and make some transactions between your accounts.

Build webservice code from repository and run it.

1. Install node.js and make sure that it is added to the environment variables (https://nodejs.org/en/download/).

2. Install Truffle Framework by opening CLI and run the command:

```
npm install -g truffle
```

3. Download code from repository: git clone **https://github.com/FutureProcessing/DocuHash**

4. Enter the folder (repository root) cd DocuHash and run the following command:

```
npm install
```

5. Open in editor (i.e. Notepad++) the config located in /services/config. If you have different values than the custom ones (i.e. Ganache configuration), please adjust them to your needs. Make sure that:

    - *config.eth_url* corresponds to the Ganache RPC Server,
    - *config.wallet_passphrase* corresponds to the Ganache mnemonic,
    - *config.network_id* corresponds to the Ganache network id,
    - *config.hs_contract_address* is left for now just as it is, we will change it soon,
    - *config.ipfs_api_address* corresponds to your IPFS API address,
    - *config.ipfs_api_port* corresponds to your IPFS API port.

6. The next step is to compile and deploy smart contracts. In CLI run, truffle console uses the command: *truffle console –network dev*. First of all, if your system does not recognise a truffle command, you have to add it to the environment variables. Find a path to truffle.cmd file (it can probably be found here: *"C:\Users\<username>\AppData\Roaming\npm\node_modules\.bin"* where the C drive is your system drive) and add new variable. For instance, using a variable name: "TRUFFLE" and variable value as path to folder containing (as above presented) *truffle.cmd*. If you see *truffle(dev);* then your console is ready to work with.

7. Run the command:

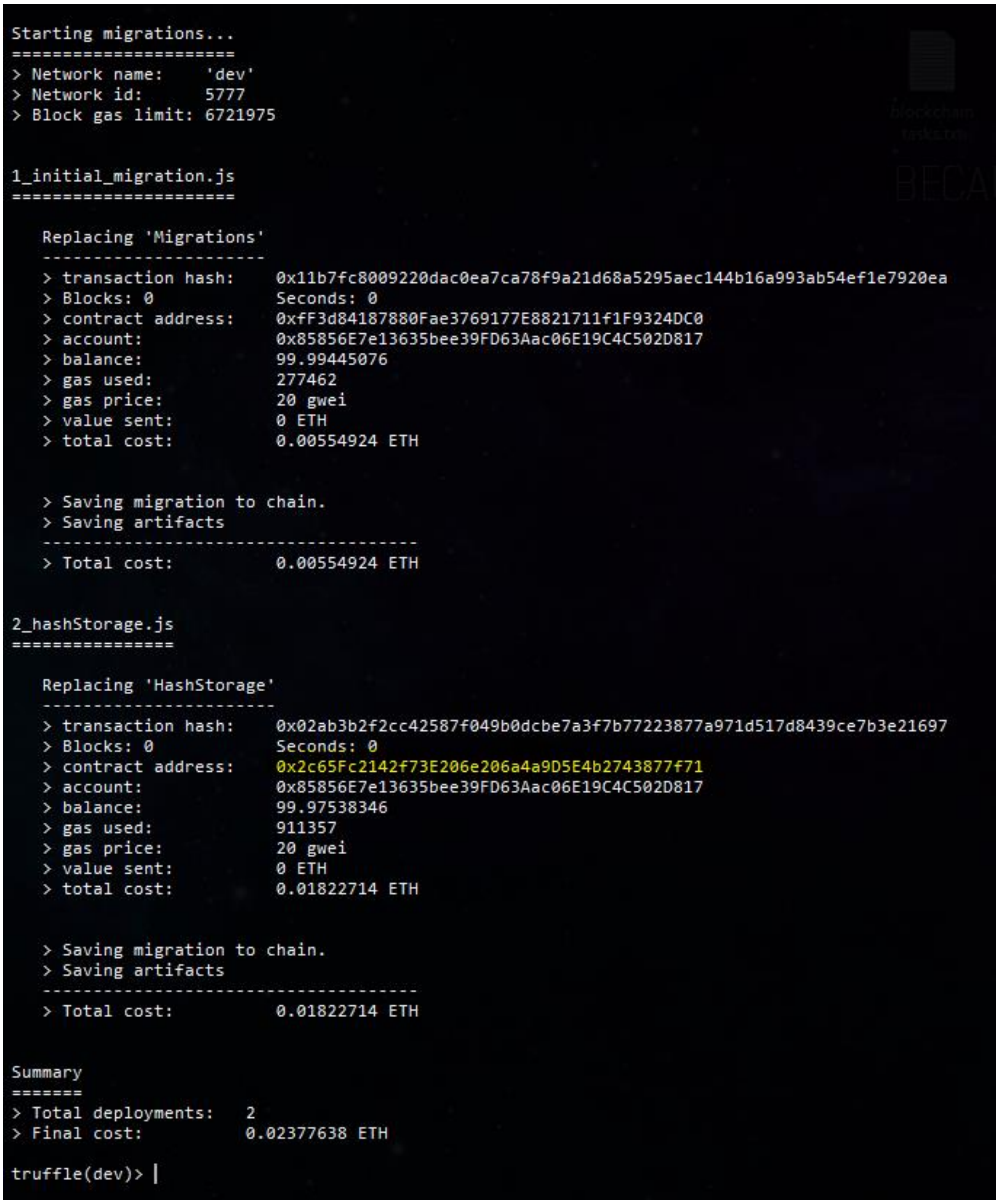
```
compile -all
```

in the running truffle console. It is responsible for compiling “all” contracts, no matter if previous compiled versions are there or not. Successful output should be ended with the following line: *Writing artifacts to .\build\contracts.*

8. Run the command:

   ```
   migrate –reset
   ```

   in the truffle console. It is responsible for deploying contracts to the blockchain. “Reset” parameter will make sure that the whole migration process is executed even if some of the contracts are already deployed. You should see the effect similar to the one shown below:



```
Starting migrations...
======================
> Network name:    'dev'
> Network id:      5777
> Block gas limit: 6721975


1_initial_migration.js
======================

   Replacing 'Migrations'
   ----------------------
   > transaction hash:    0x11b7fc8009220dac0ea7ca78f9a21d68a5295aec144b16a993ab54ef1e7920ea
   > Blocks: 0            Seconds: 0
   > contract address:    0xfF3d84187880Fae3769177E8821711f1F9324DC0
   > account:             0x85856E7e13635bee39FD63Aac06E19C4C502D817
   > balance:             99.99445076
   > gas used:            277462
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.00554924 ETH


   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:          0.00554924 ETH


2_hashStorage.js
================

   Replacing 'HashStorage'
   -----------------------
   > transaction hash:    0x02ab3b2f2cc42587f049b0dcbe7a3f7b77223877a971d517d8439ce7b3e21697
   > Blocks: 0            Seconds: 0
   > contract address:    0x2c65Fc2142f73E206e206a4a9D5E4b2743877f71
   > account:             0x85856E7e13635bee39FD63Aac06E19C4C502D817
   > balance:             99.97538346
   > gas used:            911357
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.01822714 ETH


   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:          0.01822714 ETH


Summary
=======
> Total deployments:   2
> Final cost:          0.02377638 ETH

truffle(dev)> |
```

9. Find the contract hash storage address (marked to yellow on the screen above), copy and paste it to the config file mentioned in step 5. Remember to save a file after introducing changes. In our case config file looks like this:

```
const result = require('dotenv').config();

CONFIG = {}
CONFIG.app = process.env.APP   || 'development';
CONFIG.port = process.env.PORT  || '3002';

CONFIG.eth_url = process.env.ETH_URL || 'http://127.0.0.1:9545';
CONFIG.wallet_passphrase = process.env.HD_WALLET_PASSPHRASE || 'exact cabbage shove public maximum erase remain around crawl major april cross';
CONFIG.eth_network_id = process.env.ETH_NETWORK_ID || '5777';

CONFIG.hs_contract_address = "0x2c65Fc2142f73E206e206a4a9D5E4b2743877f71";

CONFIG.ipfs_api_address = '127.0.0.1';
CONFIG.ipfs_api_port = '5001';
CONFIG.ipfs_url = CONFIG.ipfs_api_address + ':8080/ipfs/';

CONFIG.clientUrl = 'http://127.0.0.1:3003';
```

10. Run:

```
npm start
```

in repository root directory in CLI. After a while you should see: *DocuHash is running on port 3002*

Build and run a web page.

1. Open another CLI, go to the cloned repository and in CLI enter the app folder using the command:
   `cd app`

2. Run:

   ```
   npm install
   ```

   in CLI.

3. Taking default values into account, you don't have to change the config file, in other case it is placed in *app/src/config.js*; please adjust *apiServerAddress* to the address which corresponds with your webservice. In our case it is the same as in the repository.

   ```
   const config = {
       apiServerAddress: 'http://127.0.0.1:3002'
   };

   export default config;
   ```

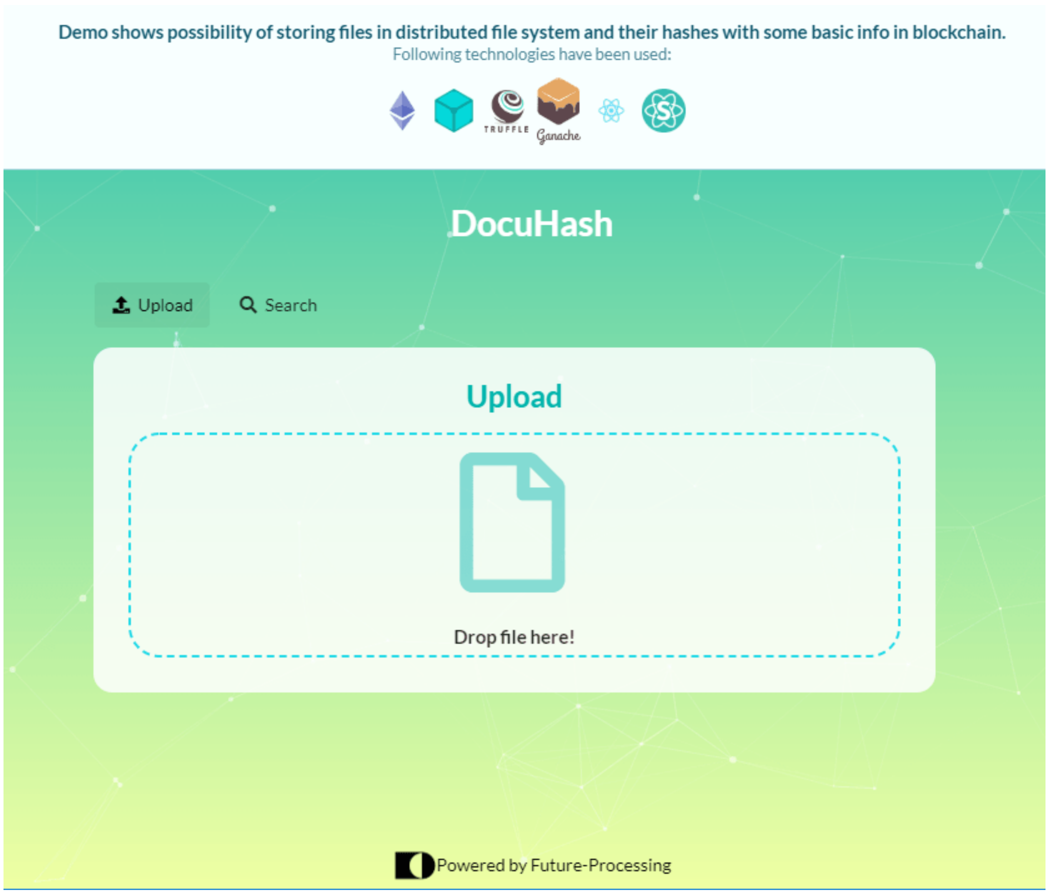
4. Run

   ```
   npm start
   ```

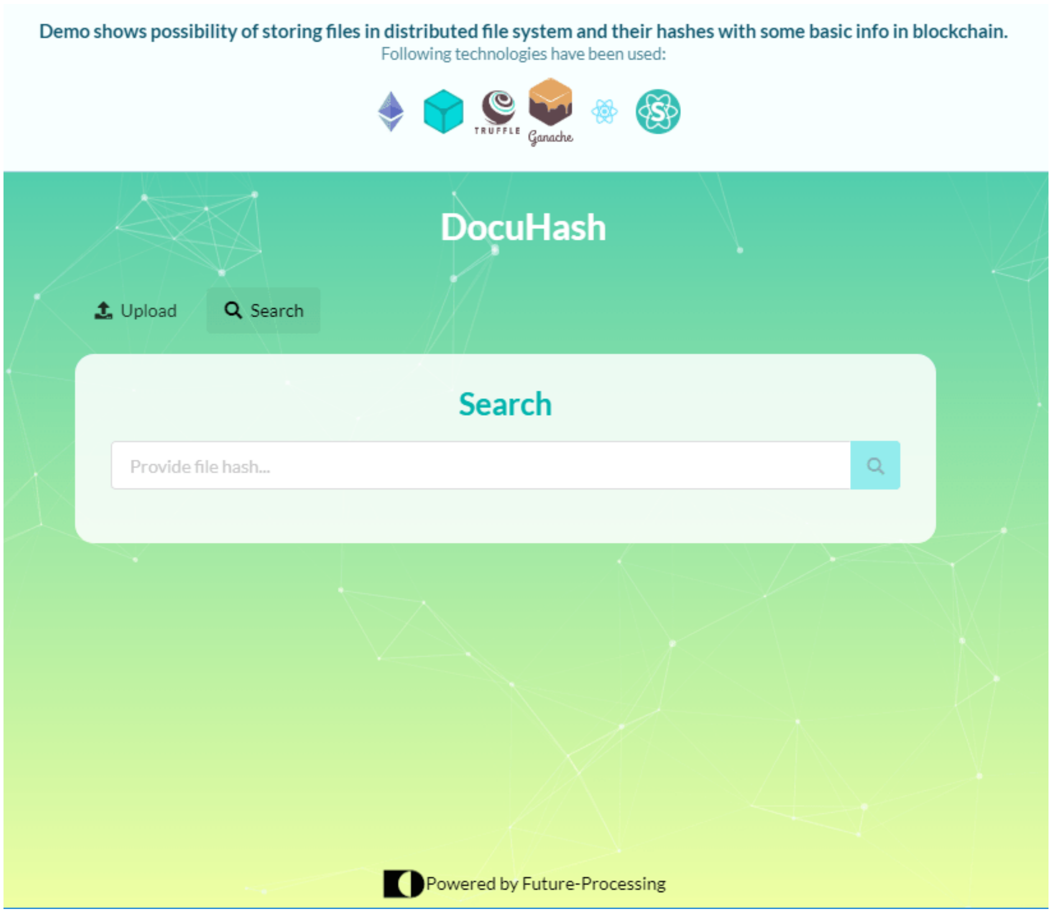
   in CLI.

   After a while you should see a message in CLI: *"Compiled successfully!"* and then how to access this page through a browser.

5. Finally, check how it works! Play with it for a while and come back for some further reading.



Demo shows possibility of storing files in distributed file system and their hashes with some basic info in blockchain.
Following technologies have been used:



Demo shows possibility of storing files in distributed file system and their hashes with some basic info in blockchain.
Following technologies have been used:

DETAILED DESCRIPTION OF SOME MAJOR ELEMENTS:

1. Smart contract HashStorage.sol in Solidity.
    a. Smart contract contains a structure which composes of file information:
        i. string ipfsHash – variable type string responsible for storing addresses/links (in fact, hash) in the decentralised file system;
        ii. uint dateAdded – variable type unsigned integer containing information of when a file was added in the unix timestamp (a number of seconds which have elapsed since the 1st of January 1970; midnight UTC/GMT);
        iii. bool exist – variable boolean type helpful in defining whether a file hash has been added to the blockchain or not. Why do we need that? It stems from the fact that variables in Solidity are initialized with default values, which in case of boolean is false.

    b. The smart contract also contains mapping which can be seen as a hash table. In general, mappings are virtually initialised for every possible key which exists (in our example hash). We can easily find a value for every possible key, even for those which have not been used previously; the keys mapping will return the value which byte-representation is a type's default value.

    c. In this Smart contract we can find two functions:
        i. get – which receives one parameter – file hash. This function allows to check whether a specific file stored in IPFS is available and, finally, date added. All this data is returned as a result of execution.
        ii. add – which expects three parameters during its invocation: IPFS hash, file hash, date added. The execution of this function is limited to an owner, which means that any other party cannot successfully invoke this method. It is ensured by Ownable.sol contract from OpenZeppelin (a library for secure smart contract development). Our owner is set up in the constructor, which is executed during a contract deployment to the network. An owner can be changed over time, but it's beyond our consideration. Getting back to the function: after some initial validation, the function adds a newly created object to the mentioned mapping.

Then at the end, it emits event as a clear signal that everything has gone right.

```
pragma solidity ^0.4.23;

import "./Ownable.sol";

contract HashStorage is Ownable{
    mapping (string => DocInfo) collection;
    struct DocInfo {
        string ipfshash;
        uint dateAdded; //in epoch
             bool exist;
         }

         event HashAdded(string ipfshash, string filehash, uint dateAdded);

         constructor () public {
             owner = msg.sender;
         }

         function add(string _ipfshash, string _filehash, uint _dateAdded) public onlyOwner {
             require(collection[_filehash].exist == false, "this hash already exists in contract");
             DocInfo memory docInfo = DocInfo(_ipfshash, _dateAdded, true);
             collection[_filehash] = docInfo;

             emit HashAdded(_ipfshash, _filehash, _dateAdded);
         }

         function get(string _hash) public view returns (string, string, uint, bool) {
             return (
                 _hash,
                 collection[_hash].ipfshash,
                 collection[_hash].dateAdded,
                 collection[_hash].exist
             );
```

```
            }
        }
```

1. Web service in node.js.
    a. Service.js defines two methods of handling user requests:
        i. addfile – the POST method expects file as input. Under the hood, it basically prepares file for uploading to IPFS, and compute the hash to store it in the blockchain. Finally adds a file to the IPFS and then save the hash in blockchain smart contract.
        ii. getfile – the GET method expects file hash as parameter, and it is used as a lookup value. The time order is different, first we check value in the blockchain and if it exists, then we make sure that it is available in IPFS.
    b. Hs-service.js also defines two methods responsible for fetching data from and pushing into the blockchain via initialized instance of the contract (hs-contract-provider.js). The mechanism of it is pretty simple: we prepare data (by converting it to a byte representation) and send it straight to the blockchain.
    c. web3-provider.js – inside this file, the web3 instance is created by setting user accounts based on provided mnemonic:
        i. Config.js – located in the services folder, contains all the necessary variables to run and configure application service.
        ii. Config.eth_url – it's the RPC address to the blockchain node you try to connect to;
        iii. Config.wallet_passphrase – it's the mnemonic allowing access to the accounts generated by Ganache;
        iv. Config.eth_network_id – an identifier of the Ganache network;
        v. Config.hsContractAddress – the address of HashStorage contract;
        vi. Config.ipfsAPIAddress – the address of IPFS node;
        vii. Config.ipfsPort – the port which helps to address to the IPFS node;
        viii. Config.ipfsUrl – the address to IPFS File Viewer.
2. Web App – React

    a. App.js – located in app/src/. This file is a main element of the web page, it combines a user interface with web service invocations. We won’t get into details. If you are not familiar with React, please take a look at some tutorials or just visit **https://reactjs.org/**.

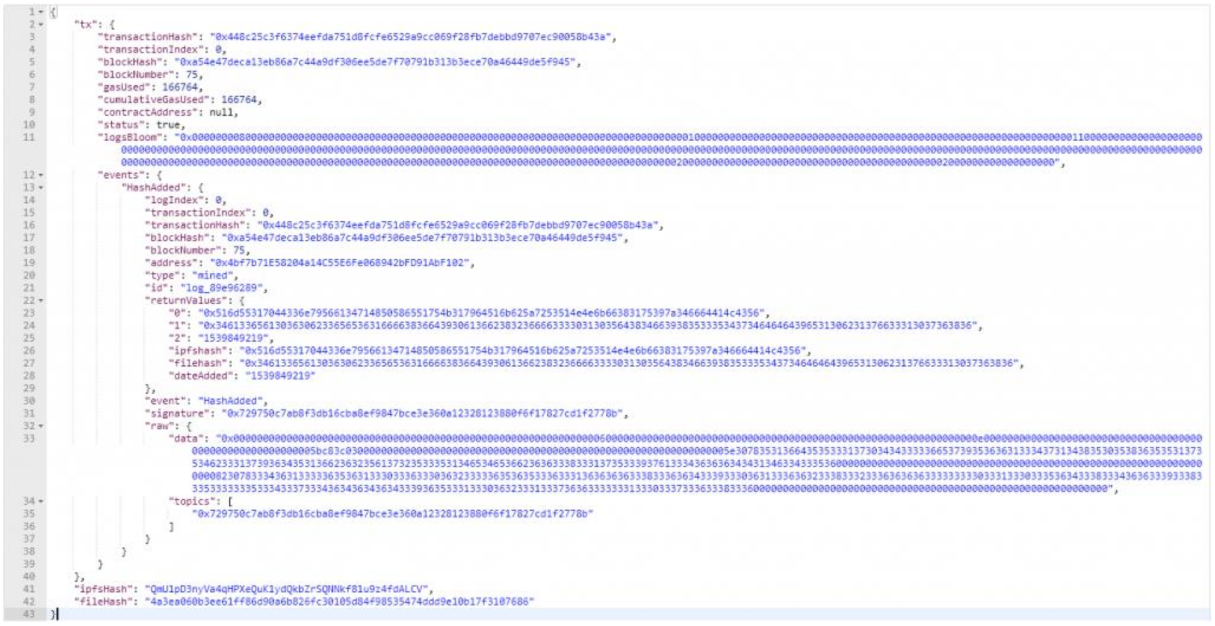
    b. Config.js – located in app/src/. This file contains only variables which refer to our web service API.

    c. WebService.js – located in app/src/. It’s responsible for calling a web service and handling error responses which may come as a result of the invalid request or configuration error.

3. Truffle-config.js – this file defines network with which we are going to work. It can have many networks defined, but every needs to have its unique name. Simple definition requires a host, where we put our local node 127.0.0.1, a port, where we put the RPC port, and a network_id. When a network_id is defined as asterisk, it means that it will connect to any network available under specified address and port.

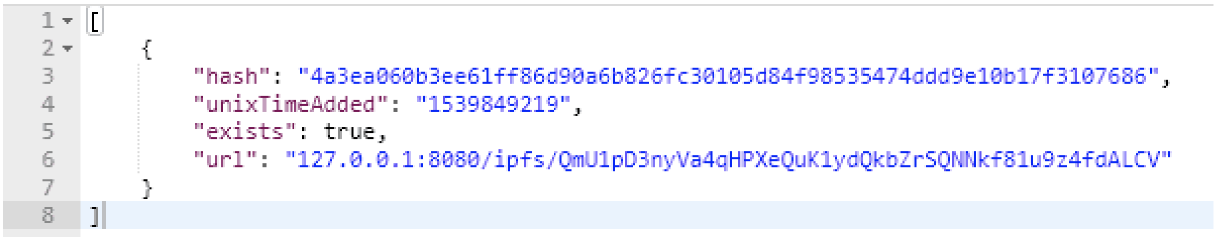
SAMPLE WEB SERVICE RESPONSES

AddFile:



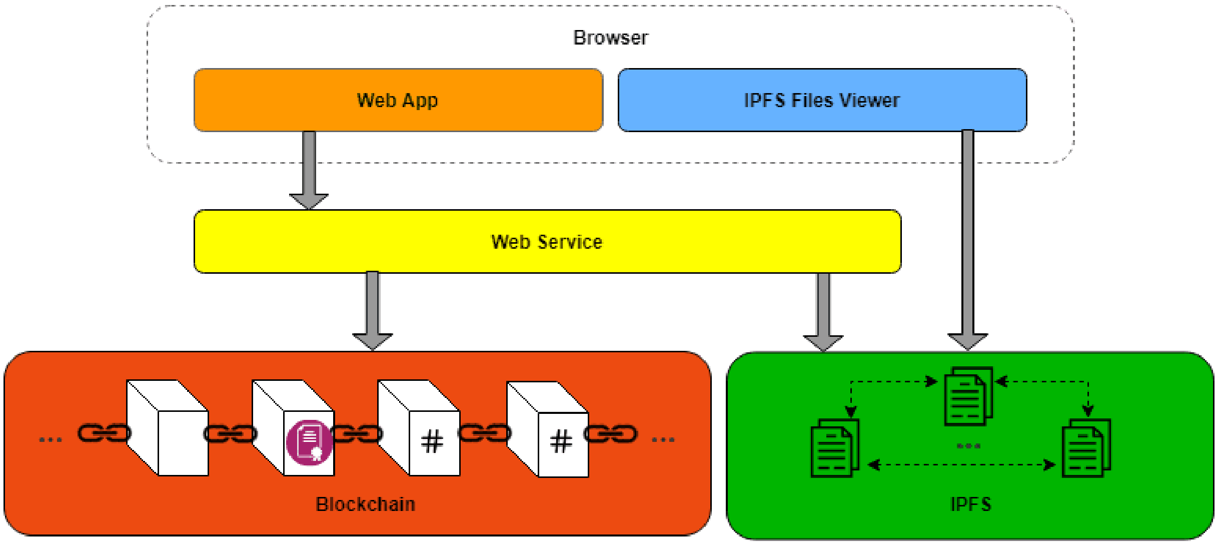
```
{
    "tx": {
        "transactionHash": "0x448c25c3f6374eefda751d8fcfe6529a9cc069f28fb7debbd9707ec90058b43a",
        "transactionIndex": 0,
        "blockHash": "0xa54e47deca13eb86a7c44a9df306ee5de7f70791b313b3ece70a46449de5f945",
        "blockNumber": 75,
        "gasUsed": 166764,
        "cumulativeGasUsed": 166764,
        "contractAddress": null,
        "status": true,
        "logsBloom": "0x000000008000000000000000000000000000000000000000000000000000000000000000000000000000000010000000000000000000000000000000000000000000000000000000000000110000000000000000000000
            000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
            00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000200000000000000000000000000000000000000002000000000000000000",
        "events": {
            "HashAdded": {
                "logIndex": 0,
                "transactionIndex": 0,
                "transactionHash": "0x448c25c3f6374eefda751d8fcfe6529a9cc069f28fb7debbd9707ec90058b43a",
                "blockHash": "0xa54e47deca13eb86a7c44a9df306ee5de7f70791b313b3ece70a46449de5f945",
                "blockNumber": 75,
                "address": "0x4bf7b71E58204a14C55E6Fe068942bFD91AbF102",
                "type": "mined",
                "id": "log_89e96289",
                "returnValues": {
                    "0": "0x516d55317044336e79566134714850586551754b317964516b625a7253514e4e6b66383175397a346664414c4356",
                    "1": "0x34613365613036306233656536316666383664393061366238323666633330313035643834663938353335343734646464396531306231376633313037363836",
                    "2": "1539849219",
                    "ipfshash": "0x516d55317044336e79566134714850586551754b317964516b625a7253514e4e6b66383175397a346664414c4356",
                    "filehash": "0x34613365613036306233656536316666383664393061366238323666633330313035643834663938353335343734646464396531306231376633313037363836",
                    "dateAdded": "1539849219"
                },
                "event": "HashAdded",
                "signature": "0x729750c7ab8f3db16cba8ef9847bce3e360a12328123880f6f17827cd1f2778b",
                "raw": {
                    "data": "0x00000000000000000000000000000000000000000000000000000000000000600000000000000000000000000000000000000000000000000000000000000000e0000000000000000000000000000000000000
                        000000000000000000005bc83c030000000000000000000000000000000000000000000000000000000000005e3078353136643535333137303434333336653739353636313334373134383530353836353531373
                        534623331373936343531366236323561373235333531346534653662363633383331373533393761333436363634343134633433353600000000000000000000000000000000000000000000000000000000000000
                        000082307833343631333336353631333033363330363233333635363533363331363636363338333636343339333036313336363233383332333636363633333333303331333033353634333833343636333933383
                        33533333335333433373334363436343634333936353331333036323331333736363333333133303337333633383336000000000000000000000000000000000000000000000000000000000000",
                    "topics": [
                        "0x729750c7ab8f3db16cba8ef9847bce3e360a12328123880f6f17827cd1f2778b"
                    ]
                }
            }
        }
    },
    "ipfsHash": "QmU1pD3nyVa4qHPXeQuK1ydQkbZrSQNNkf81u9z4fdALCV",
    "fileHash": "4a3ea060b3ee61ff86d90a6b826fc30105d84f98535474ddd9e10b17f3107686"
}
```

GetFile:



```
[
    {
        "hash": "4a3ea060b3ee61ff86d90a6b826fc30105d84f98535474ddd9e10b17f3107686",
        "unixTimeAdded": "1539849219",
        "exists": true,
        "url": "127.0.0.1:8080/ipfs/QmU1pD3nyVa4qHPXeQuK1ydQkbZrSQNNkf81u9z4fdALCV"
    }
]
```

DESIGN STRUCTURE:



A quick explanation on how it has been structured. At the top, there is a browser with two components. The first one – Web App – allows us to send a request to the hosted Web Service in a simple and clear way. This request received by the web service is processed and, as a result, data is pushed further to the blockchain and IPFS network. It handles two types of requests – the first which writes and the second which only reads and doesn't make any changes. The second component in a browser area is the IPFS File Viewer which helps to explore files stored in a distributed file system. It comes with our IPFS node configuration, and it can display a file or even play a movie if its extension is recognized by the system. IPFS File Viewer searches for file without any intermediaries, providing that it is fulfilled by user. He or she has to know the file's path or hash (these two terms can be used interchangeably) generated by the system, in order to view the file. The hash from IPFS is completely different from the one generated by the file content, therefore we store both in the smart contract.

USER INTERFACE:

The Upload view is visible by default.

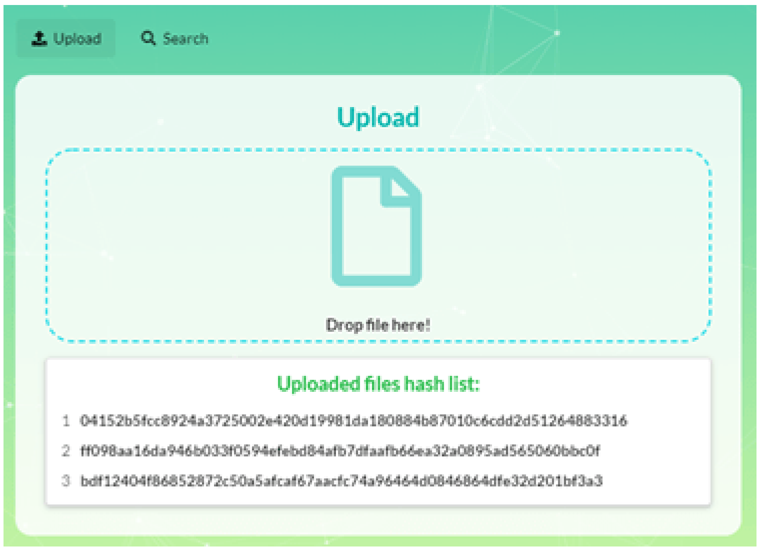
a. Thanks to a drag & drop mechanism applied to the first tab of our sample application, a user is able to upload a file.

b. Immediately after dropping a file in a dashed area, the request of this file is sent.

c. A user has to wait a short period of time which is measured by a swirl in the middle of the area.

d. After a successful operation, a new list with file hashes appears below the dropping area. The title is: “Uploaded files hash list:”.

e. In case of error, a proper box with a message appears.



Uploaded files hash list:

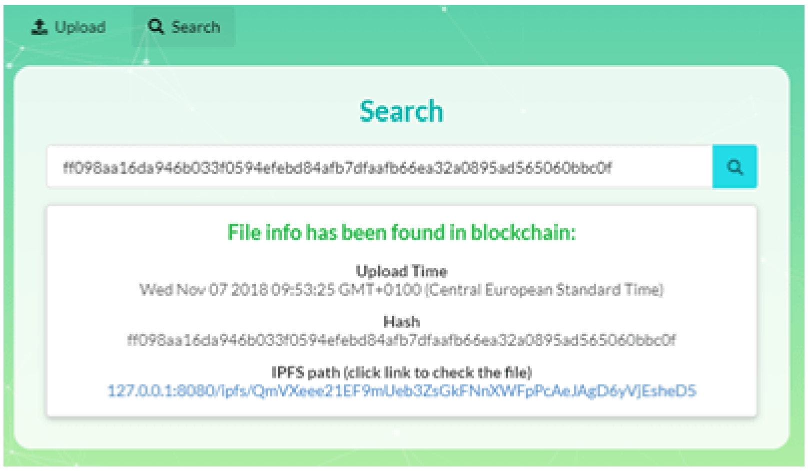
```
1  04152b5fcc8924a3725002e420d19981da180884b87010c6cdd2d51264883316
2  ff098aa16da946b033f0594efebd84afb7dfaafb66ea32a0895ad565060bbc0f
3  bdf12404f86852872c50a5afcaf67aacfc74a96464d0846864dfe32d201bf3a3
```

The user can switch to the Search view by clicking on the Search icon (with a magnifier):

a. Having the file hash, the user can search this file in the given application. After providing the hash to the search field and clicking a magnifier icon, the new lookup request is sent.

b. After a successful operation, the new block with detailed file information appears. It is titled: “File info has been found in blockchain:”. The last item on the list is a clickable link.

c. In case of error, a relevant message appears.



Search

ff098aa16da946b033f0594efebd84afb7dfaafb66ea32a0895ad565060bbc0f

File info has been found in blockchain:

Upload Time
Wed Nov 07 2018 09:53:25 GMT+0100 (Central European Standard Time)

Hash
ff098aa16da946b033f0594efebd84afb7dfaafb66ea32a0895ad565060bbc0f

IPFS path (click link to check the file)
127.0.0.1:8080/ipfs/QmVXeee21EF9mUeb3ZsGkFNnXWFpPcAeJAgD6yVJEsheD5

SUMMARY:

The proposed solution is not a comprehensive example. In order to deepen the issue, we should consider adding more nodes to a distributed file system, migrating the application to the public network, encrypting the file before sending and, finally, we should try to evaluate trust towards the server and decide who should pay the network fees. There is probably much more than that, but it is some food for thought. This article shows some possibilities and strengths of applying a blockchain technology to the described problem. If you have any questions or want to expand this solution, feel free to contact me or leave a comment. The code is available on github: https://github.com/FutureProcessing/DocuHash

SOURCES:

1. https://www.studiamba.wsb.pl/baza-wiedzy/trwaly-nosnik-co-oznacza-dla-przedsiebiorcy-w-transakcjach-z-konsumentem#trwaly-nosnik
2. https://fintech.pkobp.pl/blockchain-w-banku/
3. https://ipfs.io/