Groep 9
Bart Viaene
Guy Van Hoe
Nick Vandewalle

Solidity contracts

The core development is done in the documentStore.sol contract. The other contracts are used for the CMC setup. Functions in this contract can only be called by the "documentManager.sol" contract. Doug.sol, DougEnabled.sol and Owned.sol are technical contracts to set ownership and manage the addresses to link the contracts to each other. We will only explain documentStore.sol in detail.

documentStore.sol

Events

Only 1 event is configured, the event "DocumentRegistered" returns a Boolean "success" and the hash of the newly uploaded document once a new document is successfully stored in the contract.

Data Storage

All the information to run the contract is stored in 2 mappings.
The mapping "documents" contains the list of documents with their metadata in the form of a variable "Document". Document is a struct that consists of title, releaseDate, autor, and isEntity (this is a technical attribute to make it easy to check if a hash already exists).

The second mapping is "documentsOfAuthor" were all the hashes are linked to their author, it is created for the efficiency of the search function.

Groep 9
Bart Viaene
Guy Van Hoe
Nick Vandewalle

Functions

Each function contains follow code, as part of the CMC setup:

```
require(DOUG != 0x0);
address documentManager = ContractProvider(DOUG).contracts("documentmanager");
require(msg.sender == documentManager);
```

A function can only be called from the "DocumentManager" contract and if the DOUG address in the dougenabled contract isn't empty.

Function: IsDocumentRegistered	Assembly: no
Description: checks if a document is already registered, if so it returns the metadata linked to this hash	
Input variables: • docHash	Output variables: • Boolean true/false (does this hash already exist?) • Metadata document
Stored information: no	Triggered events: No

Function: registerDocument	Assembly: Yes, registerDocumentAsm
Description: this function registers the hash if it doesn't not exist yet	
Input variables • DocHash • Metadata document	Output variables • Boolean true/false (is the hash registered?)
Stored information: • Hash and metadata added in mapping "documents" • Hash linked to author in mapping "documentsOfAuthor"	Triggered events: • DocumentRegistered

Function: getAmountOfDocumentsFromAuthor	Assembly:yes
Description: this is a technical function required because we can't call dynamic tables in a CMC setup. This function shall be called before "getDocumentListFromAuthor" and then the documents shall be called in batch.	
Input variables • AuthorName	Output variables • nrOfDocuments, an integer with the number of documents registered on the author name
Stored information: No	Triggered events: No

Function:GetdocumentListFromauthor	Assembly: yes, GetdocumentListFromauthorASM
Description: returns titles registered to a certain author	
Input variables • AuthorName	Output variables • AuthorMail • Titles
Stored information: no	Triggered events: no

Assembly

Benefits of using inline assembly:

- **Reduced gas cost.**

  Solidity's assembly generates a lot of redundant opcodes and odd jumps.
  Using inline assembly can optimize the code significantly.
  The compiled binaries can be quite smaller than their Solidity equivalent.

- **Processing entire words at once.**

  Inline assembly allows to read entire words of 32 bytes in a single operation.
  String comparisons for example can become very fast because you can compare the m in chunks of 32 bytes instead of byte by byte.
  (This wasn't necessary for this assignment.)

- **More exposure of operations.**

  Some operations aren't exposed in Solidity. For instance, the keccak opcode takes a byte range in memory to hash, while the Solidity function takes a string.
  Hashing parts of a string in Solidity would require costly string copying operations.
  With inline assembly, only the needed bytes can be passed to hash.

- **Faster**

  Because the opcodes are optimized and entire words can be processed at once, the execution of a function will be faster than equivalent Solidity code.

- **Educational**

  You really need to think about how Solidity works internally.
  The layout of storage variables and their types are important.
  EVM uses more operations to store elements smaller than 32 bytes.
  To optimize this, the order of storage
  variables needs to be such that they can be packed tightly. For example: uint128, uint128, uint256 needs two slots of storage while uint128, uint256, uint128 needs 3.

Groep 9
Bart Viaene
Guy Van Hoe
Nick Vandewalle

Inline assembly in this assignment:

We used inline assembly for 3 functions:

- ***getAmountOfDocumentsFromAuthor***
    - Input: bytes32 authorName
    - Output: uint256 nrOfDocuments

  The use of assembly isn't really necessary here.
  It was done for demonstrational purposes.

- ***registerDocumentAsm***
    - Input: bytes32 docHash, bytes32 title, bytes32 releaseDate, bytes32 authorName, bytes32 authorMail
    - Output: bool succes

  Inline assembly would mostly benefit here.
  The use of storage variables is costly.
  For this assignment it was needed to use 2 structs and 2 mappings. Inline assembly all owed to access those storage variables in a more efficient manner.
  We didn't really see a significant reduced gas cost.
  This is probably because of the CMC setup.

- ***getDocumentListFromAuthorAsm***
    - Input: bytes32 authorName, uint256 startIndex, uint256 endIndex
    - Output: bytes32 authorMail, bytes32[pageSize] memory titles, bytes32[pageSize] memory releaseDates

  Because this function is constant, gas reduction isn't needed.
  Inline assembly does execute this function faster.

  Note: Dynamic arrays can't be passed between contracts. So fixed sized arrays are used.

Inline assembly in this assignment – explanation:

We have following storage variables:

struct *Author* { bytes32 name; bytes32 mail; }
struct *Document* { bytes32 title; bytes32 releaseDate; Author author; bool isEntity; }
Mappings:

documentsOfAuthor[authorName] = array of docHashes
documents[docHash] = Document

To get the location of documentsOfAuthor[key] in storage:

- Get first free memory slot with mload(0x40).
- Store the key into that location, which is 32 bytes.
- Store the slot of storage it uses in the next 32 bytes.
- Keccak (SHA3) those 64 bytes. Output is 32 bytes.
- The length of the dynamic array is stored at that location.

To get element documentsOfAuthor[authorName][i]:

- For element 0: Keccak the first 32 bytes of previous output.
- Add i to the new output of 32 bytes and load value stored at that location

It's the same principle for documents[docHash]:

- Store 32 byte docHash in memory.
- Store slot of storage in next 32 bytes.
- Keccak those 64 bytes.
- That location contains the struct. To access

a variable in the struct, just add its position to the calculated output.

Frontend

html

The index.html is a basic html file to create the input fields, the logic itself is a combination of JavaScript files.

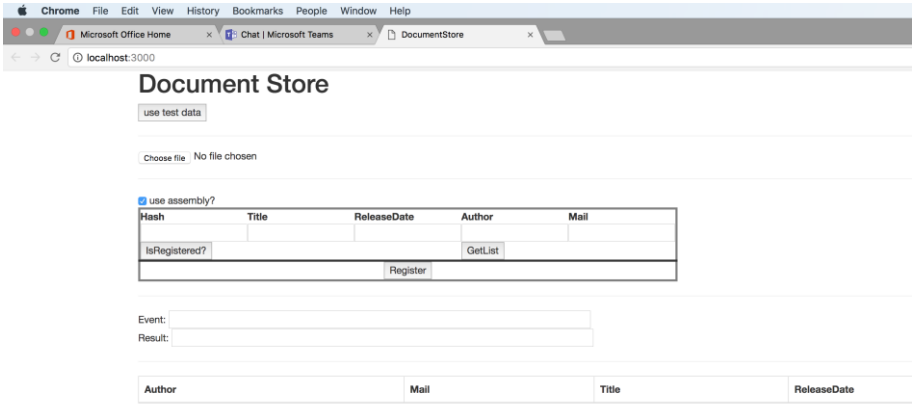
GUI

The page has 3 options:

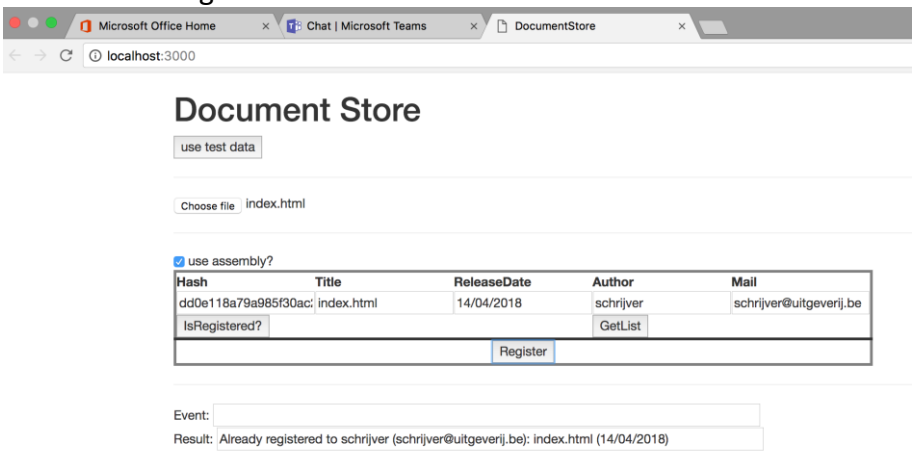
Groep 9
Bart Viaene
Guy Van Hoe
Nick Vandewalle

Choose file

This is the main functionality, with this button you can select a file and the hash of that file is calculated, the metadata has to be filled in. Once this is done you click on “Register”, if the file isn’t registered yet the Hash + metadata will be uploaded in the blockchain.

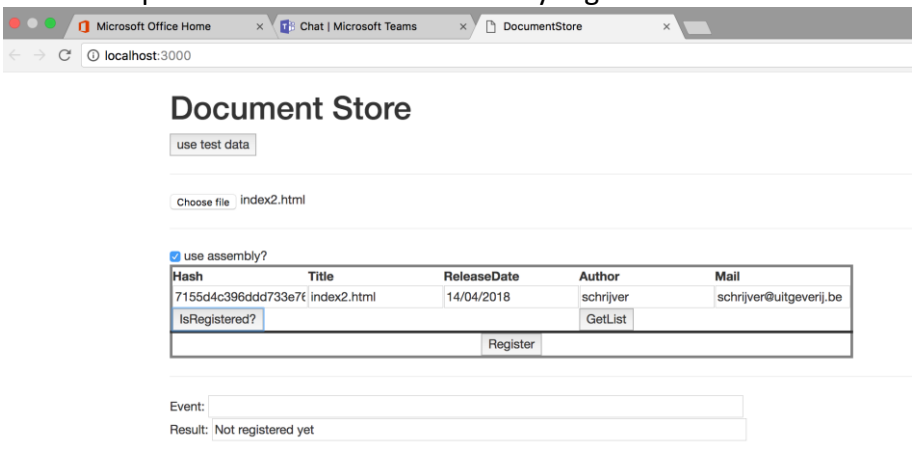


If you want to upload an already existing hash, you will receive a message that this hash is already registered including the author linked to the hash:



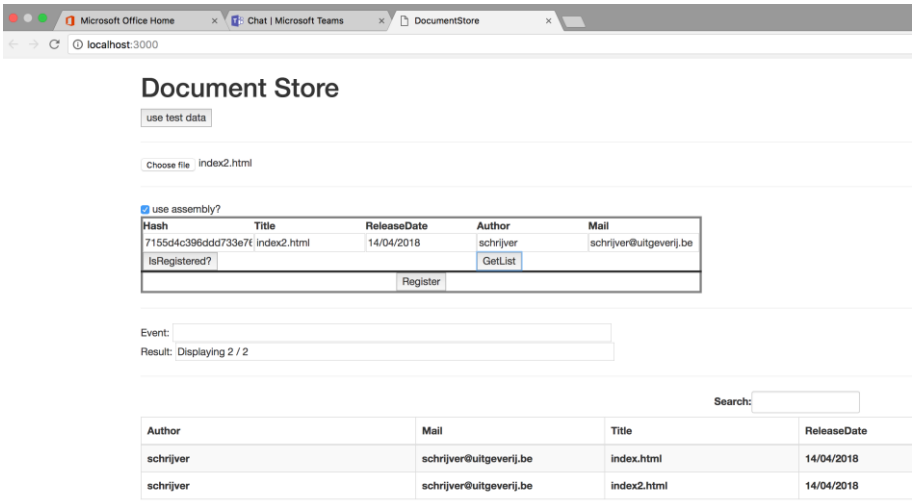
IsRegistered

This option checks if a hash is already registered in the chain



GetList

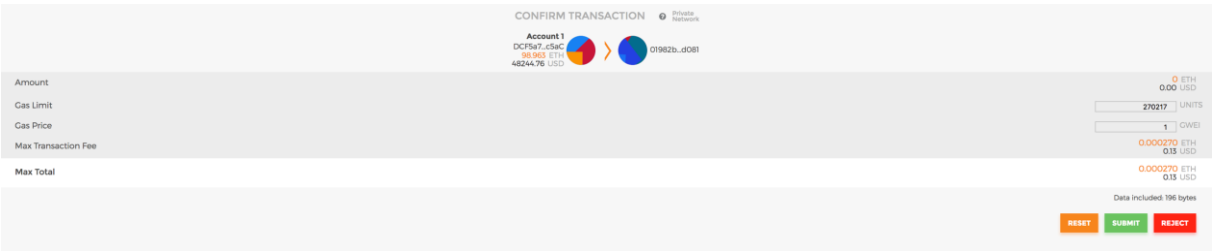
This option returns the documents registered on a specific author, the author has to be filled in in the author field.



Blockchain

For this assignment metamask linked to a ganache-cli blockchain was used.

```
Colruyts-MacBook-Pro-2:Task1 colruyt$ ganache-cli
Ganache CLI v6.1.0 (ganache-core: 2.1.0)

Available Accounts
==================
(0) 0xdcf5a71dd9c833ac791d49794a36d16b1cd7c5ac
(1) 0xce1e41d98ee807f463c7a179348c60388f652b01
(2) 0xa0a01ebcf08ff3e53f40388a7202a4f272bfb468
(3) 0x384a562805a5c0656bdf35c02823ce6f588ee612
(4) 0x3f489d5526109d92f1bfa8d8bd34653ce3f76e15
(5) 0x46c859d45deedcc3f644d3331d3747c52e2d9dd1
(6) 0x6a551a51b9f5ea526ace585a96becf48db4a0957
(7) 0xe1a3a1cb56f5c65c679b8c616173c26ba56eb593
(8) 0x34cbc651e6991e0732bd9b1f46eb041b6e090afe
(9) 0x4cd82da4bddb831066b568411561f908dc321a64

Private Keys
==================
(0) 93f108a3e1ebfd2d5512822a79af667dc79ab2c42f095d4cd672a814fbc9bf92
(1) 831aa2b7a695fe64cba085831bb317d56311c13172acb0cd56c8e7359bd046c2
(2) 2a4cc3f6bedc855ded977250281d294ac8d6fe10ba48e41840f2efc75158df13
(3) 3fe63fb100a35c635a9fa6e06e6c8c9f7a88f145c2e4181281ea6728eb1e7b05
(4) 299b50411eaefca0c2f9fe4ab63d9bf6671153545032a1830315ece5d77d9d6d
(5) 56a34b3bfe871941c92d81b8069d97b9559deed11fcd9ce514236be852e45e64
(6) 6a4fa919041d65dc15e4b7cf88a1332add573481a46229c2b1e65424858fcd01
(7) 4bc366f4c1bfec8042dcfdba84fbb7533878be9c926a56da886c123f77e98404
(8) ce249fcb9f6ad759b6d26715cd4dce8ceffd52f85aa89d55a9da4a5970c7001c
(9) e8a021678236cc5ba0b7f0631390ed43b4cf70964b0b8a2e030b89d42bf5d8e3

HD Wallet
==================
Mnemonic:      risk make isolate erupt future oxygen case always oven viable powder since
Base HD Path:  m/44'/60'/0'/0/{account_index}

Listening on localhost:8545
net_version
eth_accounts
eth_accounts
eth_accounts
net_version
net_version
eth_sendTransaction

  Transaction: 0x32808f9b6fa943ca4cd12f4128d067e64510f2f62b32f9a754275fe43146f61a
  Contract created: 0x999780d316a5557c18fff037d482d68798d35b1e
  Gas usage: 268535
  Block Number: 1
  Block Time: Thu Apr 12 2018 20:05:28 GMT+0200 (CEST)
```



Groep 9
Bart Viaene
Guy Van Hoe
Nick Vandewalle

Test scripts

As the truffle test files are developed in the 2 flavours, TestDocumentManager.js and TestDocumentManager.sol, both are executed if you run the “truffle test” command.

```
TestDocumentManager
  ✓ testDoug_ShouldBeCorrectlyConfigured (71ms)
  ✓ testRegistration_NewHash_ShouldSucceed (63ms)
  ✓ testRegistration_DuplicateHash_ShouldFail (53ms)
  ✓ testRegistration_NewHashAsm_ShouldSucceed (58ms)
  ✓ testIsDocumentRegistered_RegisteredDocument_ShouldReturnStoredData (76ms)
  ✓ testIsDocumentRegistered_RegisteredDocumentAsm_ShouldReturnStoredData (78ms)
  ✓ testIsDocumentRegistered_UnregisteredDocument_ShouldReturnNullData (71ms)
  ✓ testGetDocumentListFromAuthor_ExistingAuthor_ShouldReturnStoredDocumentHashes (404ms)
  ✓ testGetAmountOfDocumentsFromAuthor_ExistingAuthors_ShouldReturnAmountOfStoredDocumentHashes (78ms)

Contract: DocumentManager
  ✓ should have correctly configured Doug
  ✓ should be possible to register a document (70ms)
  ✓ should be possible to get a list of documents by author (207ms)


12 passing (2s)
```

Used versions

During development we noticed some different behaviour depending on the workstation it was running, it seems that this was caused by the fact that we didn’t run the same version of the underlying tooling. The project works on following version:

- Truffle: v4.1.5
- Ganache-cli: v6.1.0
- Solidity: v0.4.19 (fails on v0.4.18)
- Metamask: 4.5.5