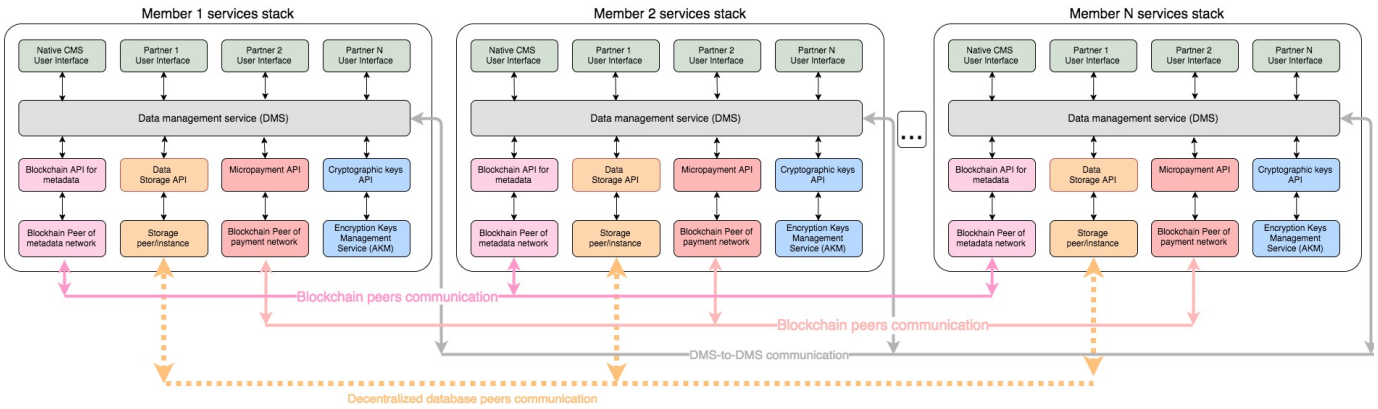
Decentralized Data Management Fabric (DDMF)

Functional Components Schema



The main goal of Decentralized Content Management System is to unite community members, who do not fully trust each other, with a purpose of mutual data sharing in a secure way. The system should be based on Blockchain as decentralized communication, metadata sharing, accounting and routing component. Sensitive data of each member should be stored in encrypted data storage. Only the member, who owns the data, can decrypt it and provide to other members on specific request.

Functional Components Description

Member
A set of system components (services) belonging to specific organization/owner

Blockchain peer
Blockchain peer is one (or a set) of Blockchain platform nodes specific to chosen Blockchain platform. The peer of each member is responsible for shared Blockchain database maintenance and supporting consensus algorithms to keep shared database state consistent across all the peers.
Tech Stack: Enterprise Blockchain platforms: Hyperledger Fabric, Ethereum Quorum, etc

Smart Contract
Smart Contract is program logic, which copies are kept and run by each Blockchain peer, who validates transactions. Smart Contract is immutable due to their decentralized nature and uniquely identified in Blockchain. There is no member, who can remove it or force changes of its internal logic. Smart Contracts execution is isolated from any system resources except Blockchain database itself.
Tech Stack: GoLang (Hyperledger Fabric), Solidity (Ethereum Quorum), etc

Data management service (DMS)
The purpose of DMS is to be central communication service. It has a central connecting role for other services: Blockchain API, Data Storage API, Key Management API, Micropayment APIs as well as DMS APIs of other Members.

Our intention is to build Members services stack pluggable. So, basically, any service or component in the stack can be replaced with alternative (some of the services even can be absent in the system).

DMS is the only service, which communicates with User Interfaces directly. Its RESTful API Endpoints for User Interfaces and DMS-to-DMS connectivity represent data in unified independently of other services in the stack.

In terms of data management DMS functions are sensitive data checks, encryption and decryption, Data Storage update and query, etc.
DMS provides onboarding functions (in future releases) to add/remove new system Members by majority Members decision.
DMS should be designed in a way to provide switching between different types of Blockchain platforms, Data Storages, Key Management, Payment Systems in an easy way.
Tech Stack: Node.js

Data Storage
Data Storage, in general, can be any type of centralized, cloud or decentralized database or even file system. Some of options are Openstack Swift, Key-Value DB, RDBMS, IPFS File System, etc. Data storage is meant to be a pluggable component to provide flexibility for data storage choice.

User Interface
User Interface communicates all the services through DMS service APIs. Current version of default Web UI based on INSPINIA - Responsive Admin Theme

Tech Stack: HTML5, CSS3, Bootstrap, Angular 1.x, Angular 2.0, etc.

Data Object

Data Object is logical data unit related to some specific abstract or physical entity. It can be healthcare information of some person, credit history of a customer, etc. More than one Member can have the same, partially overlapping or not overlapping information of the same Data Object.
Data Object fields locations are:

1. Data Object ID - *Blockchain*
2. Data Object Info - *Blockchain*
3. Data Object Creation Time - *Blockchain*
4. Data Object Update Time - *Blockchain*
5. Data Object Value - *Data Storage*

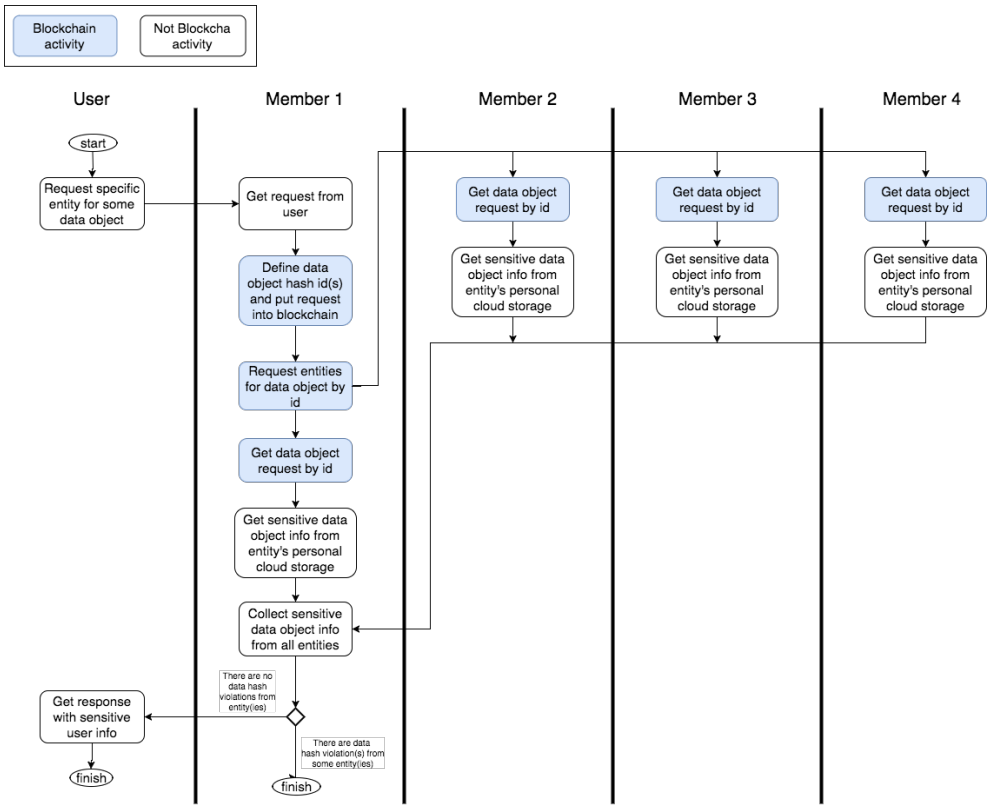
Blockchain data purpose is collaboration of Members on data storage and distribution. A Member share this data to all the Members through decentralized Blockchain database.

Data Storage data is maintained by a Member. This is stored locally, in cloud or data storage in encrypted form. A Member is the only company, which has free access to its data. A Member can provide specific Data Object Value to other Members only if the object matches that Member request and requestor fulfilled all request requirements.

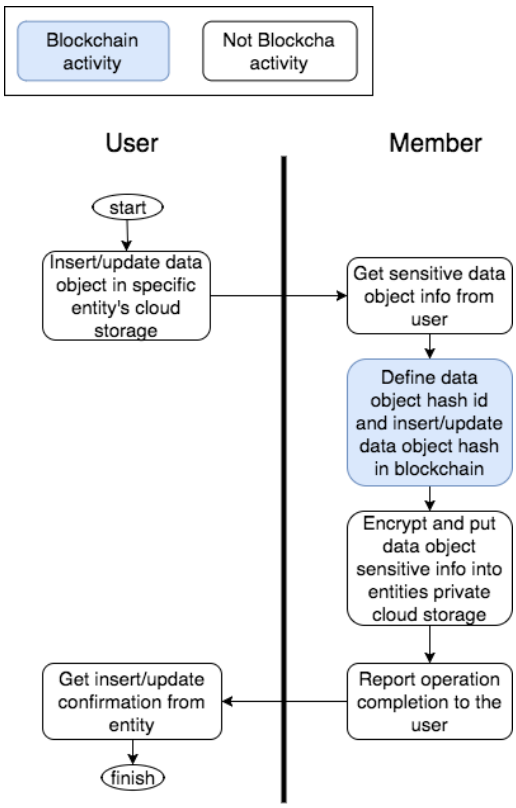
Blockchain Consensus

Blockchain consensus can be of different type: PBFT, Raft, Proof-of-Work, Proof-of-Stake. It depends on specific Blockchain platform choice and settings.
General purpose of any Blockchain consensus algorithm is to keep shared Blockchain database copies in consistent state.

Data Object Value Request Flow



Data Object Value Update Flow



Data Management Service (DMS)

User Interface REST API

Follow the link...

DMS-to-DMS REST API

Follow the link...

Data Storage API

Follow the link...

Blockchain API

Follow the link...

User Interface REST API

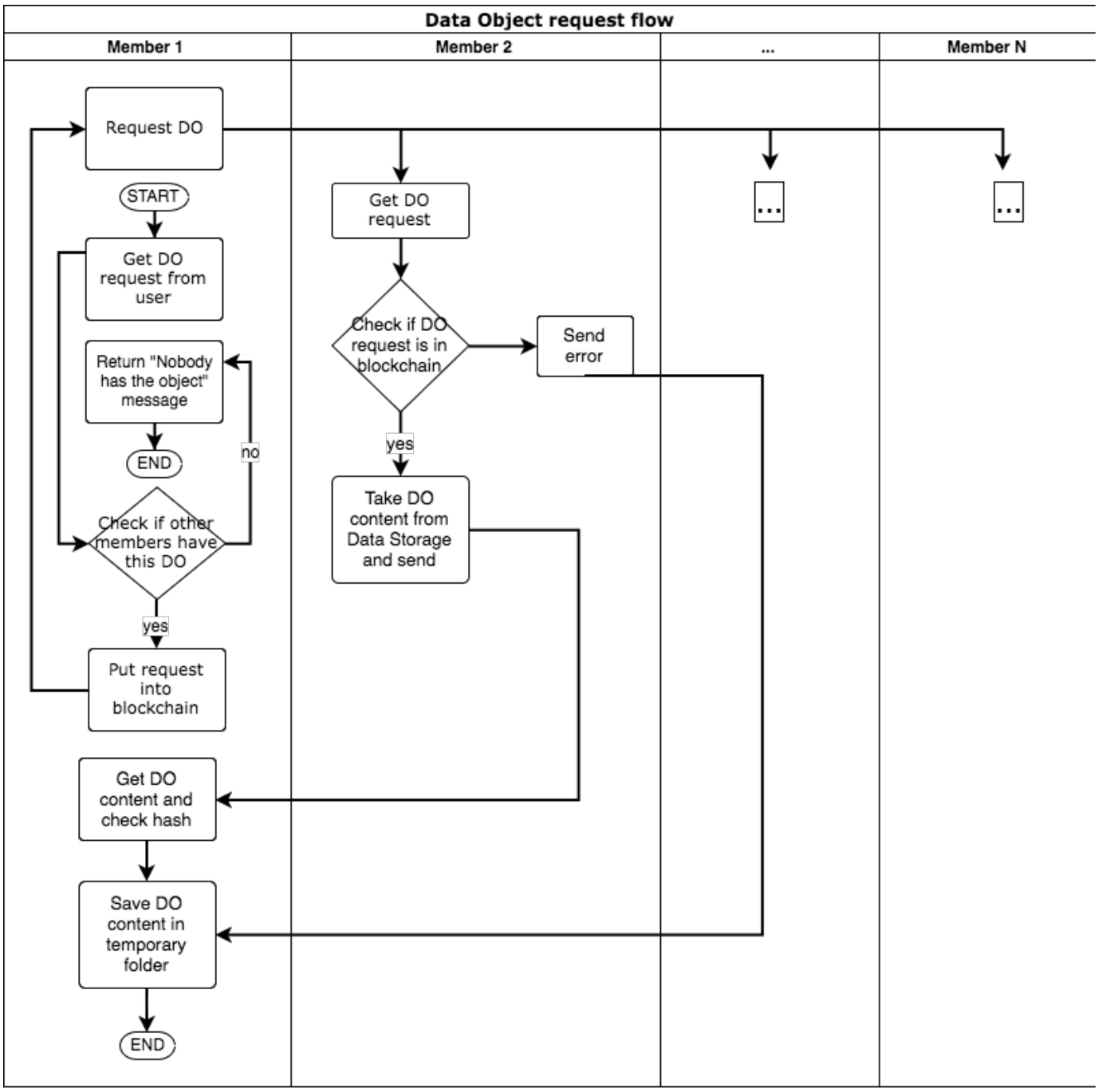
URI	HTTP Method	Payload	Responce	Description

/members/me/dos/all	GET	-	**Success:** Status: 200 OK Body: [{"doid":[string],"dohash":[string],"info":{"type":[string],"name":[string],"price":[integer],"permissions":[[string]]},"createdate":[string],"updatedate":[string],"files":[{"name":[string],"url":[string],"type":[string]}]}] **Failure:** Status: 404 Not Found	Get information of all current member Data Objects.
/members/:id/dos/all	GET	-	**Success:** Status: 200 OK Body: [{"doid":[string],"dohash":[string],"info":{"type":[string],"name":[string],"price":[integer],"permissions":[[string]]},"createdate":[string],"updatedate":[string],"files":[{"name":[string],"url":[string],"type":[string]}],"memberid":[integer]}] **Failure:** Status: 404 Not Found	Get information of all specific member Data Objects.
/members/all/dos/all	GET	-	**Success:** Status: 200 OK Body: [{"doid":[string],"dohash":[string],"info":{"type":[string],"name":[string],"price":[integer],"permissions":[[string]]},"createdate":[string],"updatedate":[string],"files":[{"name":[string],"url":[string],"type":[string]}],"memberid":[integer]}] **Failure:** Status: 404 Not Found	Get information of all Data Objects of all Members.
/dos	POST	Content-type: multipart/form-data The parts of the payload are: Part 1: Content-type: application/json; name="dometadata" Body: {"name":[string], "price":[integer], "permissions":[string]} Part 2: Content-Type, text/plain, text/css, text/html, image types; name="objectcontent" Body: <<data object file>>	**Success:** Status: 200 OK Body: {"doid":[integer]} **Failure:** Status: 404 Not Found	Add new Data Object. The payload should be composite multipart/form-data type. If some parts on the payload or just specific JSON values are missing they will be replaced with default values.

/dos/:id	POST	Content-type: multipart/form-data The parts of the payload are: Part 1: Content-type: application/json; name="dometadata" Body: {"name":[string], "price":[integer], "permissions":[string]} Part 2: Content-Type, text/plain, text/css, text/html, image types; name="objectcontent" Body: <<data object file>>	**Success:** Status: 200 OK **Failure:** Status: 404 Not Found	Update Data Object. If some parts on the payload or just specific JSON values are missing they will not be updated.
/dorequests	GET	-	**Success:** Status: 200 OK Body: [{"requestid":[integer], "memberid":[string], "objectid":[string], "requestdate":[string]}] **Failure:** Status: 404 Not Found	Get Blockchain request history of all data objects
/dorequests	POST	Content-Type: application/json Body: {"doid":[integer]}	**Success:** Status: 200 OK Body: {"dorequestid":[integer]} **Failure:** Status: 404 Not Found	Put DO request into DataObjectRequestHistory in blockchain and request DO from other Members by DMS-to-DMS API Here is a flow of this process
/dorequests/:id	GET	-	**Success:** Status: 200 OK Body: [{"memberid":[integer], "dostatus":[string], "dohash":[string], "docreatedate":[string],"doupdatedate":[string],"dorequeststatus":[string],"dofilesystemlink":[string]}] **Failure:** Status: 404 Not Found	Show DO request details by request ID
/memberstatuses	GET	-	**Success:** Status: 200 OK Body: [{"memberid":[integer], "memberstatus":[string], "memberdetails":[string]}] **Failure:** Status: 404 Not Found	Get all members statuses in the network

/memberid	GET	-	**Success:** Status: 200 OK Body: {"currentmemberid":[integer]} **Failure:** Status: 404 Not Found	Get current memberid
/systemchecks	GET	-	**Success:** Status: 200 OK Body: [{"check":[string],"checkstatus":[string],"checkdescription":[string]}] **Failure:** Status: 404 Not Found	Get health statuses of current member components
/systemstatus	GET	-	**Success:** Status: 200 OK Body: {"systemstatus":[string]} **Failure:** Status: 404 Not Found	*Not implemented yet!* Get general system status: "OK", "Error". Return OK if all member components checks are success. Return "Error" if any of the checks fails.
/balance	GET		**Success:** Status: 200 OK Body: {"settledbalace":[integer], "pendingbalance":[integer]} **Failure:** Status: 404 Not Found	*Not implemented yet!* Get current member balance.

Data Object request process flow



DMS to DMS REST API

The main purpose of this component is to support secure communication and Data Object transfer to and from other Members DMS services.

URL	HTTP Method	Payload	Output	Description
/dos/:id	GET	-	**Success:** Status: 200 OK Content-Type: multipart/form-data **Failure:** Status: 404 Not Found	Returns content for specific Data Object

Blockchain API

Functions

Add data object record

Add new DO record to Blockchain database on behalf of Member.

Input	Output
• DO hash • DO id • Member id	• DO id • DO hash • Create Date • Update Date

Protobuf: `dcms.blockchain.Blockchain.AddDataObject`

Get my data objects

Get information of all DOs of current community Member.

Input	Output
• Member id	List of DOs For each DO: • Do id • Do hash • Create Date • Update Date

Protobuf: `dcms.blockchain.Blockchain.GetMyDataObjects`

Get DO network info

Get Specific DO info split by each community Member.

Input	Output

• DO id
• Member Ids
	Recordset with row per each community member:
	
	• Member ID,
	• DataObject, if it is owned by the member:
	  • DO id
	  • DO hash
	  • Create Date
	  • Update Date

Protobuf: `dcms.blockchain.Blockchain.GetDataObjectNetworkInfo`

Update DO record

Update DO record in DataObjectMemberOwnership, move old record values into DataObjectHistory.

Input	Output
• DO id • DO hash	• Do id • Updated Do hash • Updated Create Date • Updated Update Date

Protobuf: `dcms.blockchain.Blockchain.UpdateDataObject`

Request DO from network

Put request into blockchain on behalf of Member. Add record to into DataObjectRequestHistory table.

Input	Output
• DO id • Member id	• Request Id • Request origin member Id • DO id • Request Create Date

Protobuf: `dcms.blockchain.Blockchain.MakeRequest`

Check DO requests

Get all DO requests from all members.

Input	Output
—	List of DO Requests. For each DO Request: • Request ID • Member ID • Data Object ID • Request Creation Date

Protobuf: `dcms.blockchain.Blockchain.GetDataObjectRequests`

Update member information

Update member information

Input	Output
▪ Member Id ▪ Member Name ▪ Member Info*	Same as Input: • Member Id • Member Name • Member Info

Member Info is base64 encoded JSON.

Protobuf: `dcms.blockchain.Blockchain.UpdateMember`

Get member information

By Member Id

Input	Output
• Member Id	• Member Id • Member Name • Member Info

Member Info is base64 encoded JSON.

Protobuf: `dcms.blockchain.Blockchain.GetMemberInfo`

Get information on all members

Input	Output

—	List of members. For each member: • Member Id • Member Name • Member Info

Member Info is base64 encoded JSON.

Protobuf: `dcms.blockchain.Blockchain.GetAllMembers`

Smart Contract

Smart contract is decentralized programming logic, which is immutable and its copy is stored in every Blockchain peer. Each function call of this logic is executed by each validating peer, so all honest peers will have the same result to consent on function call outcome and Blockchain database state.

As the first Blockchain platform option we will use Ethereum Quorum Blockchain. It uses Ethereum Virtual Machine (EMV) as an engine for Smart Contract execution. Solidity is most popular EMV programming language, which we will use to implement Smart Contract.

Smart contract functionality

Data Structures

Smart contract should shore data in a form similar to relational database tables and records. This can be achieved by using arrays of structs in Solidity Smart Contract language. See data model below.

Elementary CRUD operations

Smart contract should support these elementary operations for every Data Structure. This can be implemented as internal functions.

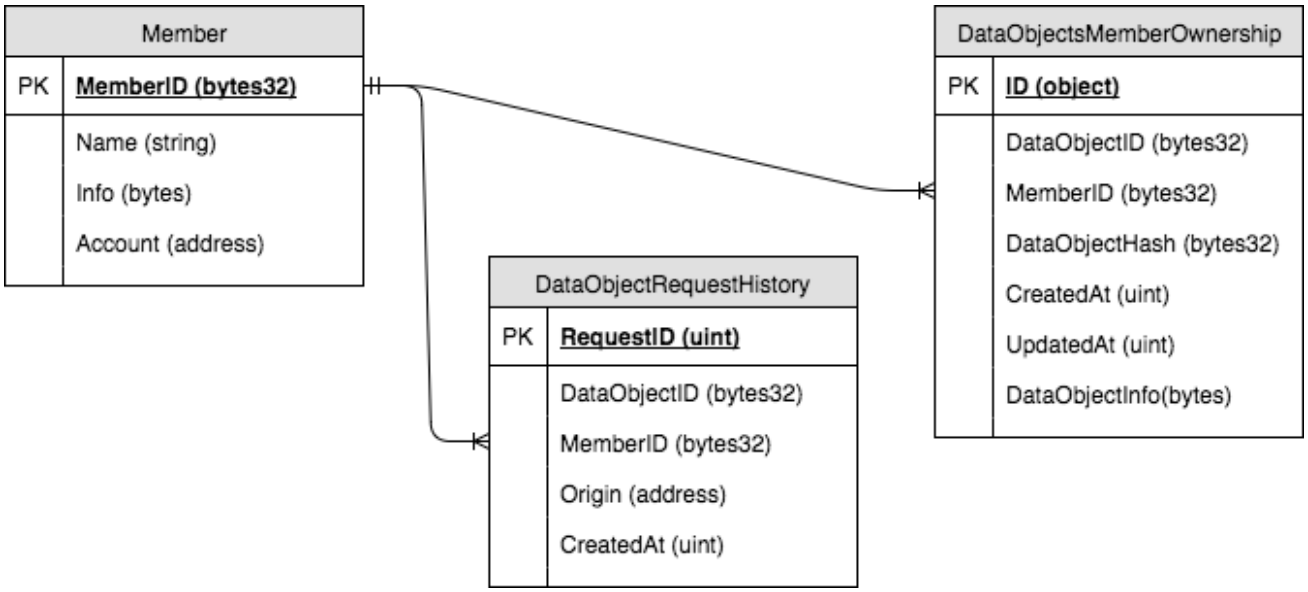
High Level Functions

Other components will make calls to these functions, so they should be available from outside of Smart Contract. These functions will use Smart Contract functionality components and functions, describe above, as lower level building blocks for its functionality.

The functions should support functionality of Blockchain & Smart Contract communication component.

Smart Contract Data Model

The model has nothing to do with regular relational database tables and constraints. It is depicted here in a form of logical data model just for visualization purpose.



Some of the fields in this data model have complex data structure and represent objects rather than simple values.

Here are descriptions of current structure of such objects in JSON form:

Member.Info

Description: Member metadata representing Member specific info.

```
{

"dmscertificate":[string]   // "DMS service
certificate of the member for secure DMS-to-DMS
communication, in base64",

"dmslocation":[string]   // "DMS service ip and
port of the member for DMS-to-DMS communication"

}
```

DataObjectMemberOwhership.DataObjectID

Description: Data object identity. Although internal structure of this object is simple and could be represented by atomic data field, we are likely to switch to more complex structure in the future

```
{

"id":[string]   // Data object id
}
```

DataObjectMemberOwhership.DataObjectInfo

Description: Object metadata representing flexible object info. Any of this fields may be missing, in that case the field is treated by DMS as default value.

```
{

"name":[string],    // Default "". Object name

"type":[string],    // Default "storage". Object
type. "Storage" if stored in Member storage
system, "blockchain" if stored in blockchain only

"price":[integer]   // Default 0. Price for
object. This price will be used for micro
transactions and is measured in wei - minimal
fracture of native Ethereum network
cryptocurrency.
```

Web User Interface

Functions:

- My Network status
- Network Members Status
- My Data Objects
  - Add new Data Object
  - View/Update Data Object
- Request Data Object from Network
  - View requested Data Object
- Data Objects request history

```
"permissions":[array]   // Default []. A list of
Members allowing to request for the Data Object.
Examples: ["1","3"] - specific Members, ["all"] - all
                                    ed

                                  // Default []. Any
                                  n be stored here
```

My network checks

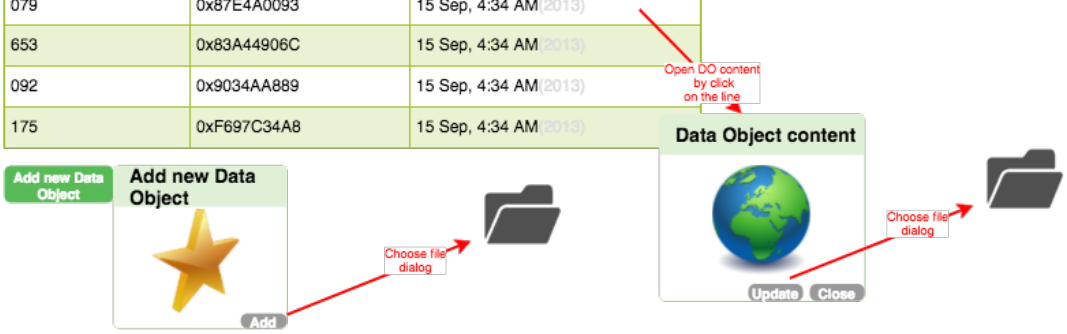
Functionality	Status
Blockchain	OK
Data Storage	FALED

Network Members Status

Member ID	Member Status	Member Details
1	Online	Member 1 details
2	Online	Member 2 details
3	Offline	Member 3 details
4	Online	Member 4 details

My Data Objects

Data Object ID	Data Object Hash	Data Object last update
079	0x87E4A0093	15 Sep, 4:34 AM(2013)
653	0x83A44906C	15 Sep, 4:34 AM(2013)
092	0x9034AA889	15 Sep, 4:34 AM(2013)
175	0xF697C34A8	15 Sep, 4:34 AM(2013)



Request Data Object from Network

Data Object ID 789 Request DO from network Request successful

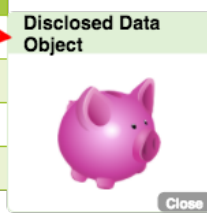
Data Objects request history

Request ID	Member ID	Data Object ID	Request DateS	Status
4	2 (Me)	097	15 Sep, 4:34 AM(2013)	In progress
3	3	345	15 Sep, 4:34 AM(2013)	Completed
2	1	874	15 Sep, 4:34 AM(2013)	Completed
1	2 (Me)	097	15 Sep, 4:34 AM(2013)	Completed

Open request details by click on the line

Data Object Request details

Member ID	Status	DO Hash	DO Last updated	DO Request Status
1	Exists	0x194AED5A49	15 Sep, 4:34 AM(2013)	Completed
2	Not Exists	0xA4D5AE9401	15 Sep, 4:34 AM(2013)	
3	Exists	0x148015DAA0	15 Sep, 4:34 AM(2013)	Pending
4	Exists	0xFFFF5BFF512	15 Sep, 4:34 AM(2013)	Completed



Data Storage API

At current project iteration Data Storage is file system. Data Storage API is not implemented as separate microservice. All its functionality is temporary included into DMS service. Data Objects will be stored in specific directory as separate files or folders.

Functionality (briefly):

- **Add DO**

Save Data Object as separate file in specific directory

Input: Data Object ID, Data Object content

Output: -

- **Update DO**

Update the file corresponding to specific Data Object

Input: Data Object ID, Data Object new content

Output: -

- **Get DO**

Get specific Data Object content

Input: Data Object ID

Output: Data Object content