Call identifier: H2020-ICT-2016 - **Grant agreement no**: **732907**

Topic: ICT-18-2016 - Big data PPP: privacy-preserving big data technologies

Deliverable 6.8

Blockchain Analytics (1)

Due date of delivery: October 24th, 2017

Actual submission date: December 7th, 2017

Start of the project: 1st November 2016
Ending Date: 31st October 2019

Partner responsible for this deliverable: Gnúbila

Version: 6.0

Document Classification

Title	Blockchain Analytics (1)
Deliverable	D6.8
Reporting Period	M1-M12
Authors	Alexandre Flament, Mirko Koscina
Work Package	WP6
Security	Public
Nature	Report
Keyword(s)	Blockchain, analytics

Document History

Name	**Remark**	**Version**	**Date**
Alexandre Flament	First Version	1.0	October 4th 2017
Mirko Koscina	Second Version	2.0	October 30th, 2017
Jérôme Revillard	Third version	3.0	November 3rd, 2017
Mirko Koscina	Forth version	4.0	November 22th, 2017
Anna Rizzo	General review version	5.0	December 5th, 2017
Edwin Morley-Fletcher	Final reviewed version prior to submission	6.0	December 6th, 2017

List of Contributors

Name	**Affiliation**
Alexandre Flament	Gnúbila
Mirko Koscina	Almerys

List of reviewers

Name	**Affiliation**
Anna Rizzo	Lynkeus
Edwin Morley-Fletcher	Lynkeus

Table of contents

Table of figures

1 Scope of this document

This document corresponds to the *Deliverable 6.8 Blockchain Analytics*, based on the work done in T6.2 and T6.3. D6.8 will illustrate the blockchain usage thanks to illustrative and eye-catching analytics. It will serve the purpose of feeding the project website and marketing materials.

T6.2 Blockchain Assessment, Prototyping and Integration consists in the task of assessing existing blockchain technologies and select an appropriate candidate for implementation. Particular emphasis will be brought to Ethereum and the Hyperledger technologies. Scalability will be a key point for the proposed infrastructure and several data sharing, data indexing and (in memory) parallel processing techniques will be considered.

On the other hand, *T6.3 Blockchain Infrastructure Deployment and Test* consists in the task of deploying the technology selected in T6.2 over the physical infrastructure of the participating centres. More specifically, hospitals such as OPBG, DHZB, GOSH will be installed and parameterized as pillar nodes of the system. Others will join at a later stage as the network will propagate in the clinical community. A software package will be released periodically, allowing external centres to join in. It will comprise the blockchain mining service, the API, the Data Catalogue (PID indexing) and the core libraries from WP4. Once the blockchain infrastructure is deployed, scale-up tests will be operated with synthetic data also from WP4, to validate the overall infrastructure robustness, performance, and compliance with the initially identified requirements.

This document is organized in six sections. Section 1 corresponds to the scope of the document and Section 2 is an introduction about what is blockchain and how we can obtain data from it. In Section 3 we describe the Hyperledger components involved in transaction flow and data storage. In Section 4 we introduce the concept of chaincode to manage and develop applications on the blockchain. Subsequently, , in Section 5 we introduce a description of our implementation to feed the data analytics module. Finally, in Section 6 we present a mock-up of the data analytics platform for MHMD.

2 Introduction

A blockchain system is a distributed ledger where all the transactions which have taken place in the network are stored in blocks. These blocks are sorted and added to the network one by one, forming a chain of blocks. The system is decentralized because the ledger is replicated within the nodes participating in the processing services. The security properties of this decentralized system are based on cryptographic techniques that make this chain of blocks immutable. The new blocks generation and the ledger replication process within the node of the network is governed by the consensus algorithm defined in the network. The consensus algorithm uses cryptographic functions and business rules to decide which node will add the next block and the replication of the last blockchain state to the rest of the nodes.

In any system it is crucial to know the status of the infrastructure and how this is being used (operations, users, queries, among others) to measure the performance of the service that is running on it. In the case of decentralized system, this can be more tricky than in a central system because the status will depend on the nodes and there are several characteristics that made it more complex. In the case of blockchain, measuring the different status of the ledger will depend on multiples components that give life to the decentralized system.

A simple query to get information from a block implies that we must connect to a specific channel (ledger) and find the block. In the case of transactions, these operations can be more complex because they will depend on the status. If we would like to know the status of a simulated transaction, endorsed transaction or the committed transaction, we would need to take into account different modules of Hyperledger Fabric, like: consensus (responsible of the transactional confirmation process and block generation), state of the database, status and membership service, in order to know who is validly authorized to perform specific instructions.

Hyperledger Fabric provides a framework with standard instructions to operate over the network called Chaincode. This is one of the key elements of the blockchain implementation, because it gives us a simple interface to communicate with different nodes and channel in the decentralized system. In addition, we have Hyperledger Composer that is a suite to manage and develop new applications for the blockchain in a friendly user environment.

In the following sections we will describe the concepts introduced above and how all the components can interact by using Chaincode and Hyperledger Composer.

3 Hyperledger Fabric Blockchain

The Hyperledger project was founded in 2015 by the Linux Foundation to advance the blockchain technologies for multiple industries. The main goal is to facilitate and encourage the development of blockchain implementation beyond the widely known cryptocurrencies.

Hyperledger Fabric is a blockchain framework that allows to implement smart contracts inside its private and permissioned ledger. The smart contracts are used to provide controlled access to the ledger. Using smart contracts, it is possible to encapsulate information and store it across the network, and define business rules that can execute transactions automatically. The permissioned access to the ledger is reached by using an enrolment service called Membership Service Provider (MSP). This facilitates the control over the network and gives flexibilities for the consensus algorithm because of the lack of need to prove the honesty of the nodes by using expensive techniques like Proof-of-Work. The identity manager administrates the user's IDs and authenticates all the participant member of the network. This service permits to parametrize different layers of permission of specific networks operations, making it possible to allow or block some users to invoke or develop new operations within the network. Additionally, Hyperledger Fabric offers privacy and confidentiality in the network by using private channels. These are restricted messaging paths that can be used to provide privacy for a specific group of users within the network.

The smart contracts triggered in the network are represented as intangibles assets by the decentralized system. These assets are managed as a collection of key-values pairs, with state changes recorded as transactions on a channel ledger. The operations, controls or modifications of these assets, are governed by the chaincode. The chaincode corresponds to small programs that run specific instructions in the ledger. With these we can manage network rules, configuration parameters, and query the ledger, among other things.

3.1 Hyperledger Fabric Components

Hyperledger Fabric is a comprehensive framework to implement customizable business blockchain networks. The Fabric model is based on six components: *Assets, Chaincode, Ledger, Channels, Security and Membership Service,* and *Consensus* [1]. All these components together make the full blockchain solution.

3.1.1 Assets

Hyperledger Fabric assets can range from intangibles like contracts and intellectual property to tangible like real estates or any goods. These assets can be modified by using special chaincode transactions. In Fabric, the assets are defined as a collection of key-value pairs with state changes recorded as transaction on the ledger.

3.1.2 Chaincode

Chaincode is an application or code that allows the administrator of the business blockchain define the assets and the operations (transaction instructions) to modify them. Chaincode functions runs over the current state database and are initiated through a transaction. As a result of the Chaincode execution, we obtain a set of key value writes to submit to the network and to be applied to the ledger on all the nodes.

3.1.3 Ledger

The blockchain ledger is a sequence of tamper-resistant record of all state transitions in the system. The state transitions are produced by transactions (invocations) submitted by members of the network. There is one ledger per channel and each node store a copy of the ledger of which they are participating.

Within the ledger we can find the following features [1]:

- Query and update ledger using key-based lookups, range queries, and composite key queries
- Read-only queries using a rich query language (if using CouchDB as state database)
- Read-only history queries - Query ledger history for a key, enabling data provenance scenarios
- Transactions consist of the versions of keys/values that were read in chaincode (read set) and keys/values that were written in chaincode (write set)
- Transactions contain signatures of every endorsing peer and are submitted to ordering service
- Transactions are ordered into blocks and are “delivered” from an ordering service to peers on a channel
- Peers validate transactions against endorsement policies and enforce the policies
- Prior to appending a block, a versioning check is performed to ensure that states for assets that were read have not changed since chaincode execution time
- There is immutability once a transaction is validated and committed
- A channel’s ledger contains a configuration block defining policies, access control lists, and other pertinent information
- Channel’s contain Membership Service Provider instances allowing for crypto materials to be derived from different certificate authorities

3.1.4 Channels

Channels are used to maintain subnets in the blockchain implementation. Each channel will have one ledger with specific business rules. Fabric permits to configure the system according to the business rules of the service, allowing to set one common shared ledger for all the members or more than one with restricted access to a selected group of members.

The implementation of multiples channels is uses to isolate the transactions and the ledgers. This can be done by installing chaincode only on peers that need to access the asset state to read and/or write into the private ledger. Additionally, it is possible to encrypt the data by using common encryption schemes with their correspondent secret keys.

3.1.5 Security and Membership Service

Hyperledger Fabric is a blockchain implementation where all the members are known participants. In order to enrol and validate members, a Certificate Authority (CA) is implemented based on a Public Key Infrastructure (PKI) environment. The CA is responsible of the digital certificate issuance process for each member of the network. The certificates tied organization, network components, and end users or client application. With this scheme is possible to govern the access to the entire network and on channel level by authenticating members according their digital certificate.

3.1.6 Consensus

The consensus algorithm is one of the backbones of any blockchain implementation, due to its role in the addition of the new blocks and in the chain replication. In the case of Hyperledger Fabric, the consensus algorithm is involved in the entire transaction flow from the proposal and endorsement, to the ordering, validation, and commitment. The consensus is achieved when the transactions have been ordered and passed through a series of explicit policy criteria checks. These checks are part of the transaction lifecycle and include endorsement policies, to establish which members must endorse the transaction (according to their type), as well as system chain codes, to ensure the policies enforcement. Additionally, system chain codes are used to validate that the transaction has been properly endorsed prior to the commitment. Moreover, there is a versioning check where the current state of the ledger is agreed upon, before adding a new block to the chain. Finally, there are identity verifications during all the transactions flow.

3.2 Chain

The chain corresponds to a transaction log where each transaction is recorded in a block [2]. These blocks contain N transactions and are sorted in sequences. Each block is linked to their predecessor by adding the transaction hash value of the previous block in the header. In addition, the hash value of the transaction recorded in the block is also stored in the header. This allows to maintain the chain ordered in sequence and linked between consecutive blocks by using cryptographic functions.

The chain structure mentioned above guarantees the ledger immutability property. This means that the hash link implies that any change in a recorded transaction will affect the block hash value. This would force to reconstruct the entire chain from the block where the change was made it until the last block.

3.3 State Database

The state database is an indexed view of the transaction log and can be recovered at any time or automatically upon the started up peers [2].

The current state database corresponds to the latest values for all keys ever included in the transaction log. It is also known as World State because it represents all the latest key values known to the channel. The current state database is stored in the state database to make more efficient the operations from the chaincode. Within the supported databases, there are levelDB and couchDB.

3.4 Transaction Flow

The transaction flow starts when an application client sends a transaction proposal to an endorsing peer. The endorsing peer verifies the digital signature of the client, and then simulates a transaction by executing a chaincode function. From this process we get a set of key/value versions that were read in the chaincode (read set) and a set of key/value that were written in the chaincode (write set). The transaction proposal response is sent back to the client with the endorsement signature of the endorsing peer. Then, the client broadcasts a transaction payload with the endorsement to an ordering service. This service is responsible to broadcast the ordered transactions to all the peers on the channel.

Once the peers have received the transactions they will validate it according to the endorsing policy of the channel, and authenticate the signatures with respect to the transaction payload. This process ensures that the correct allotment of specific peers have signed the result of the transaction proposal. In addition, peers check the versioning of the read set to ensure the data integrity and protect against double-spend attack. Finally, for each valid transaction the write sets are committed to the current state database and the block is appended to the chain [2].

In Figure 1 we can see the transaction flow of Hyperledger Fabric [3].

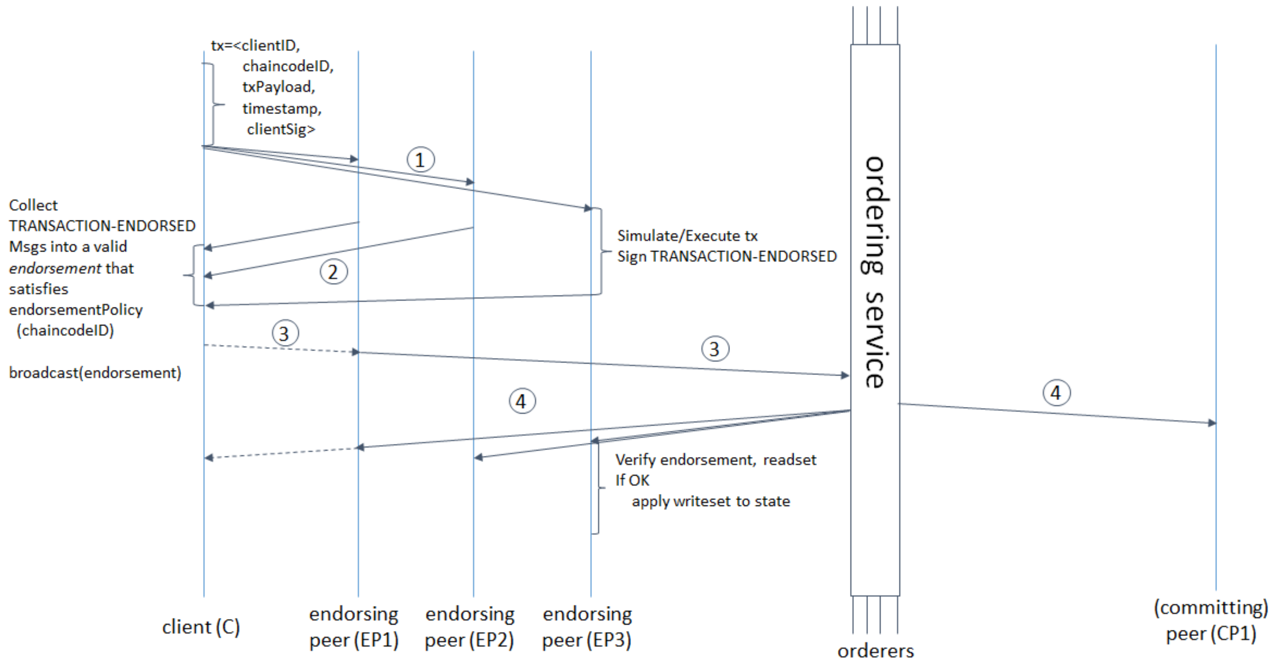

tx=<clientID, chaincodeID, txPayload, timestamp, clientSig>
1
Collect TRANSACTION-ENDORSED Msgs into a valid endorsement that satisfies endorsementPolicy (chaincodeID)
2
Simulate/Execute tx
Sign TRANSACTION-ENDORSED
3
broadcast(endorsement)
3
ordering service
4
4
Verify endorsement, readset
If OK
apply writeset to state
client (C)
endorsing peer (EP1)
endorsing peer (EP2)
endorsing peer (EP3)
orderers
(committing) peer (CP1)

Figure 1: Hyperledger Fabric Transaction Flow (copyright Linux Foundation – Hyperledger Project)

4 Chaincode

Chaincode is an application that manages the business logic of the network [4]. From the practical point of view, chaincode can be considered as a smart contract because it is a business logic agreed upon by the network members. The chaincode runs in a secured Docker container isolated from the endorsing peer process, while initializing and managing the ledger state through the transactions sent by the applications.

The chaincode has two perspectives, one for developers and other for operators. The Chaincode for Developers is focused on the application development to add or control functionalities customized by the implementer. On the other hand, the Chaincode for Operators is oriented to manage the blockchain network.

4.1 For Developers

The chaincode programs are developed in Go, Java or Javascript languages, and must implement the chaincode interface [5]. These interfaces are methods called in response to received transactions. In the case of the Init method, this is called when a chaincode receives an Instantiate or Upgrade transaction. With this instruction the chain code performs any necessary initialization (including initialization of the application state).

On the other hand, the Invoke method is called in response to an invoke transaction to process the transaction proposal. In addition, there is a chaincode interface called ChaincodeStubInterface, which is used to access and modify the ledger, and to perform invocations between chain codes.

4.2 For Operators

The chaincode for operations allows to administrate the system by using an API that can be accessed by command line. The commands to manages the chaincode lifecycle are: package, install, initiate and upgrade. Once the chaincode is installed and initiated, the chaincode remains active to process transaction via invoke.

Additionally, the system chaincode runs within the peer process instead of run in an isolated container like normal chaincode. This chaincode are used to implement a number of system behaviours that can be modified or replaced by the system administrator. Currently, we have available the following system operations [6]:

- LSCC: Lifecycle system chaincode handles lifecycle requests described above.
- CSCC: Configuration system chaincode handles channel configuration on the peer side.
- QSCC: Query system chaincode provides ledger query APIs such as getting blocks and transactions.
- ESCC: Endorsement system chaincode handles endorsement by signing the transaction proposal response.
- VSCC: Validation system chaincode handles the transaction validation, including checking endorsement policy and multiversioning concurrency control.

For example, In Figure 2 we can see the chaincode swim-lane [7] of a transaction process in Hyperledger Fabric.

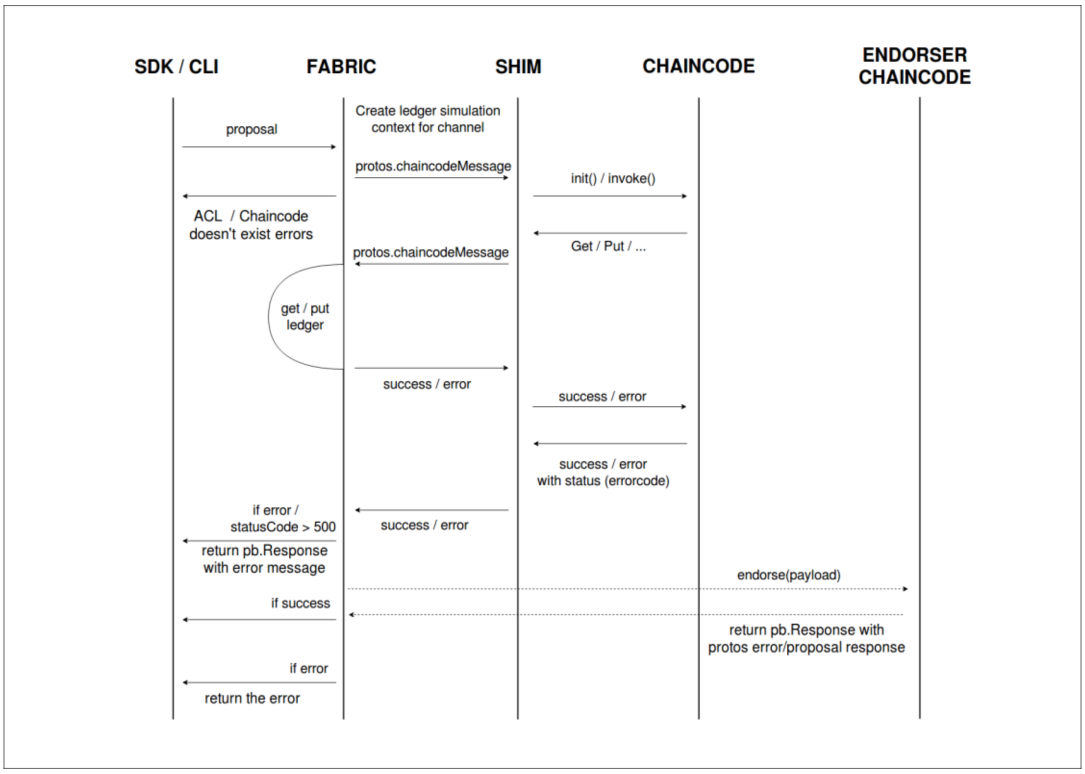


Figure 2: Hyperledger Fabric Chaincode Swimlane (copyright Linux Foundation – Hyperledger Project)

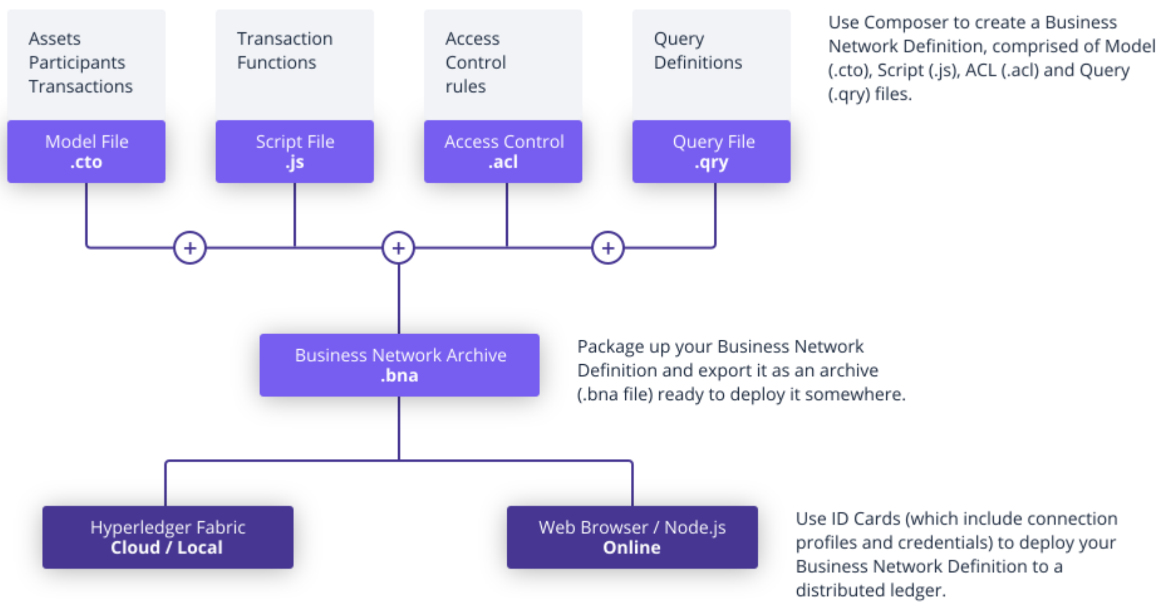
5 Implementation

The implementation of the module for data analysis for MHMD is based on an integration between the different components of the ledger. The assets represent the information that we have recorded in the ledger, the channel is ledger to query, the database is the status of the chain, the chaincode are the functions that allow to query the ledger and the transactions are the triggers to execute some operations on the chain.

To get access to the different components of the ledger and create a new business logic for this purpose, it is necessary to develop a platform capable of simplifying the management and development on the blockchain. The tool selected to implement the programs to feed the data analytics module is Hyperledger Composer. With this tool we will operate the ledger and develop the smart contracts and the business logic to feed the analytics module.

5.1 Hyperledger Composer

Hyperledger Composer is a comprehensive toolset to develop blockchain applications [8]. This suite let us to develop an API that run some business rules to get specific data to feed it to the data analytics module. Also, is managed by Composer the access to different queries to maintain the restricted the operations on the private channel (ledger). In Figure 3 we can see how composer interact with the Hyperledger Blockchain.

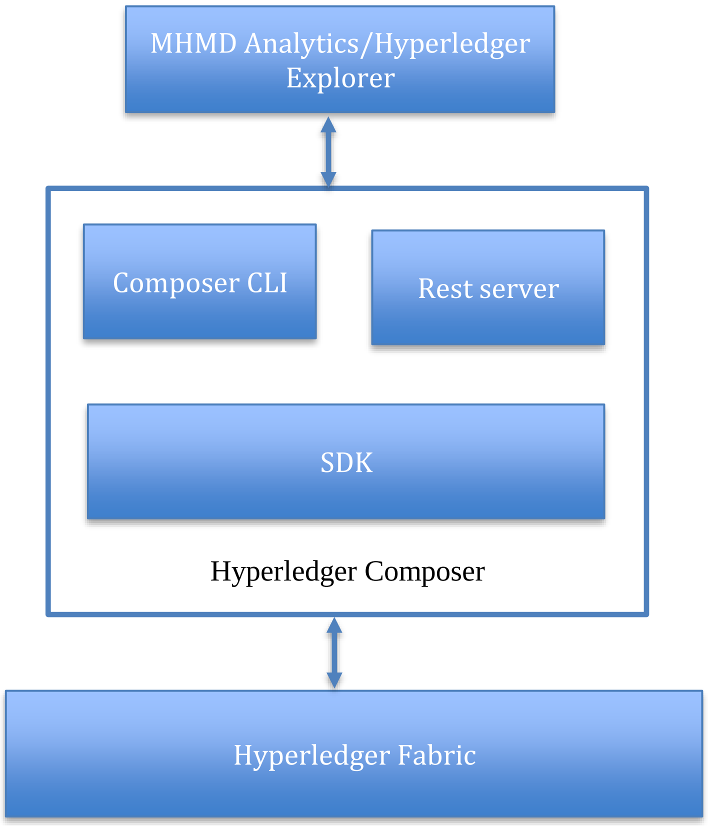


Assets Participants Transactions
Transaction Functions
Access Control rules
Query Definitions
Use Composer to create a Business Network Definition, comprised of Model (.cto), Script (.js), ACL (.acl) and Query (.qry) files.
Model File .cto
Script File .js
Access Control .acl
Query File .qry
Business Network Archive .bna
Package up your Business Network Definition and export it as an archive (.bna file) ready to deploy it somewhere.
Hyperledger Fabric Cloud / Local
Web Browser / Node.js Online
Use ID Cards (which include connection profiles and credentials) to deploy your Business Network Definition to a distributed ledger.

Figure 3: Hyperledger Composer Diagram (copyright Linux Foundation – Hyperledger Project)

5.2 Implementation for MHMD

The analytics module is based on Hyperledger Explorer that runs on Hyperledger Composer. The MHMD Analytics will interact with the Composer framework (Rest Server-CLI and SDK) to periodically query the ledger regarding the number of transactions, the number of blocks and the number of nodes operating within the network.



5.3 MHMD Hyperledger Explorer

5.3.1 Transactions Query

The amount of transactions can be queried by reading the HistorianRecords of the channel. This record contains the information about the historical transactions and can query the full container as well as just specific transactions. The implementation for MHMD is based on the query of the full HistorianRecord to get the total amount of transactions performed in the network.

The implementation of the transaction query is presented below:

```
    .then(() => {
        return MHMDNetworkConnection.getHistorian();
    }).then((historian) => {
        return historian.getAll();
    }).then((historianRecords) => {
        console.log(prettyoutput(historianRecords));
    })
```

5.3.2 Block Count Query

The number of blocks will provide the information about the growth rate of the chain and allows to calculate the performance of the consensus algorithm. With this the time that the system takes to to confirm a new transaction, by adding it into a new block, can be measured.

An implementation example of the block count query is presented below:

```
type BlockchainInfo struct {
     Height            uint64 `protobuf:"varint,1,opt,name=height"
json:"height,omitempty"`
     CurrentBlockHash  []byte
`protobuf:"bytes,2,opt,name=currentBlockHash,proto3"
json:"currentBlockHash,omitempty"`
     PreviousBlockHash []byte
`protobuf:"bytes,3,opt,name=previousBlockHash,proto3"
json:"previousBlockHash,omitempty"`
}
```

5.3.3 Participants Count Query

The participants count gives us the status of the network according the number of participants. This information can be presented as a total number of participant and/or according the role into the network.

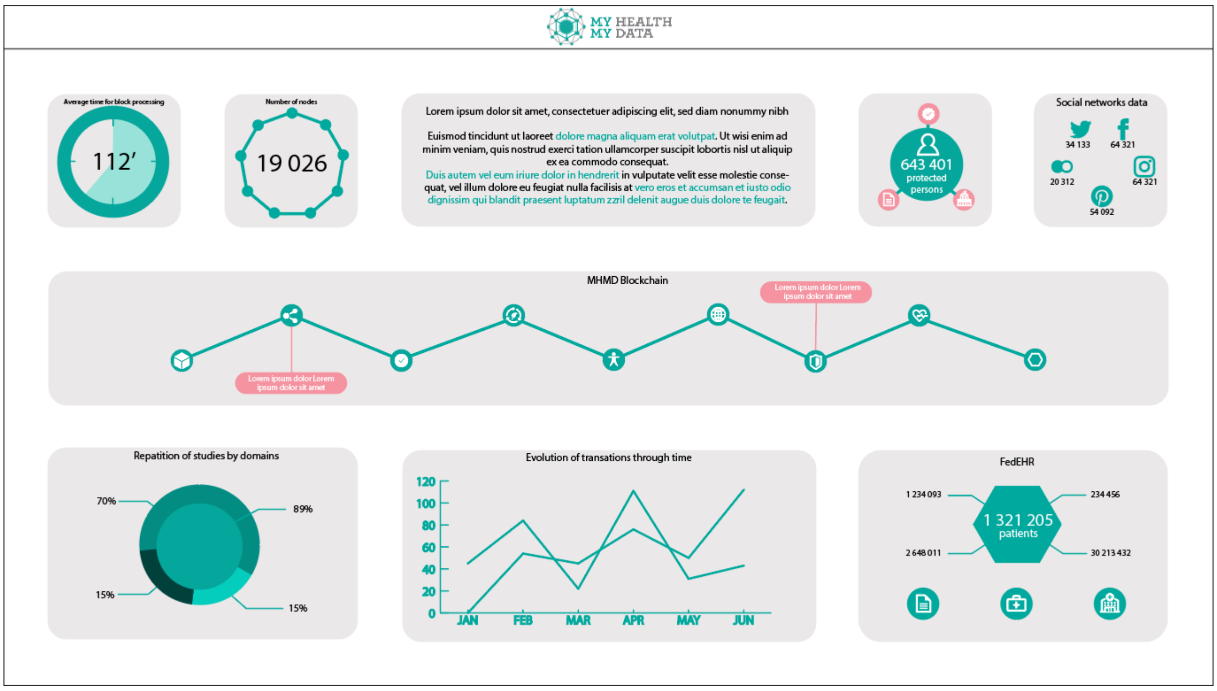
The implementation of the nodes count query is presented below:

```
  const MHMDNetworkConnection = require('composer-
client').MHMDNetworkConnection;
  let MHMDNetworkConnection = new MHMDNetworkConnection();
  return MHMDNetworkConnection.connect('hlfv1', 'digitalproperty-network',
'admin', 'adminpw')
      .then(() => {
          return MHMDNetworkConnection.getIdentityRegistry();
      })
      .then((identityRegistry) => {
          return identityRegistry.getAll();
      })
      .then((identities) => {
          identities.forEach((identity) => {
            console.log(`identityId = ${identity.identityId}, name =
${identity.name}, state = ${identity.state}`);
          });
          return MHMDNetworkConnection.disconnect();
      })
      .catch((error) => {
          console.error(error);
          process.exit(1);
      });
```

6 MHMD Analytics Mock-up

The dashboard is meant to display, in an efficient and eye-catching way, every relevant data related to the blockchain. It must be appealing and yet insightful, using charts and graphical components to illustrate the data gathered by MHMMD software. It is meant to be a powerful and highly interactive interface, with multiple metrics to look at.

The mock-up of the full dashboard is presented below:



MY HEALTH MY DATA
Average time for block processing
112'
Number of nodes
19 026
643 401 protected persons
Social networks data
34 133
64 321
20 312
64 321
54 092
MHMD Blockchain
Lorem ipsum dolor Lorem ipsum dolor sit amet
Lorem ipsum dolor Lorem ipsum dolor sit amet
Repatition of studies by domains
70%
89%
15%
15%
Evolution of transations through time
120
100
80
60
40
20
0
JAN
FEB
MAR
APR
MAY
JUN
FedEHR
1 234 093
234 456
1 321 205 patients
2 648 011
30 213 432

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh

Euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait.

Figure 4: MHMD analytics Dashboard

1.1 Block processing

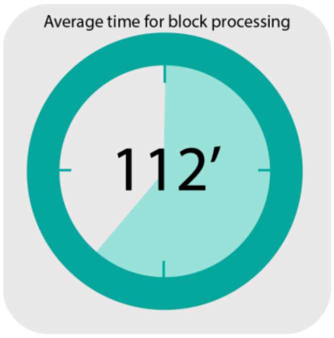


Figure 5:Clock-like visualization for block processing time

This component displays the average time for block processing in a clock-like visualization. It is measured in blocks per minutes

1.2 Number of nodes

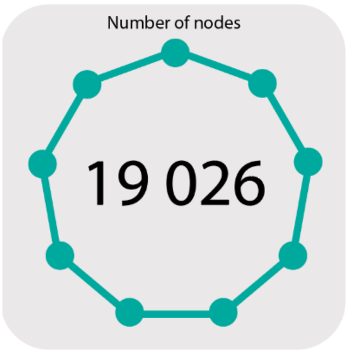
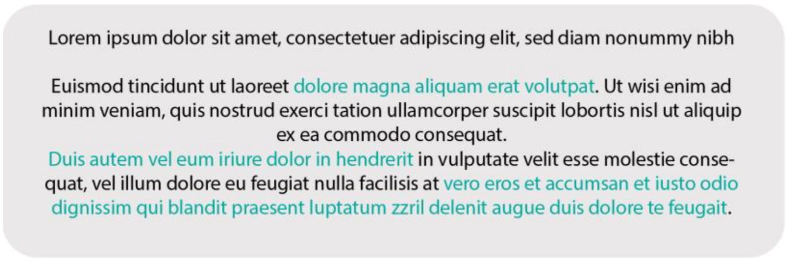


Figure 6: Number of nodes visualization

This component displays the total number of nodes in the system. The visualization is made with circles linked to each other to represent the nodes and the blockchain.

1.3 Textual space



Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh

Euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.
Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait.

Figure 7: Information text box

A block for displaying information that's not related to any type of data, or presenting how the blockchain works.

1.4 GDPR compliance

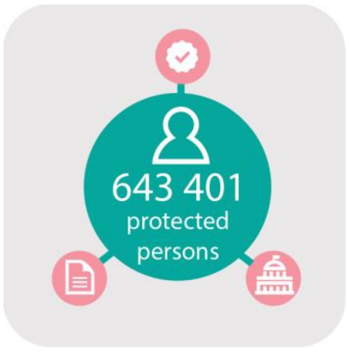


Figure 8: GDPR information box

The total number of protected persons in the blockchain. Also, a reminder of the GDPR norms, and how MHMD is complying with these laws. Complementary can be displayed by hovering the circles. The three main axis are:

- Compliance journey
- Right to be forgotten
- Transparency client

1.5 Social networks data

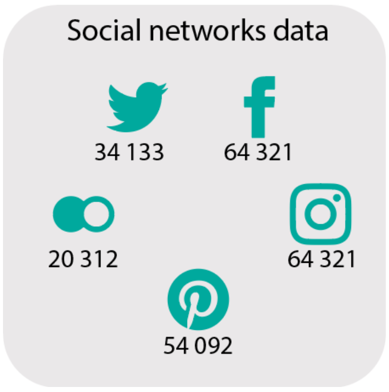


Figure 9: Social network data box

This block displays the data gathered by digi.me. It could be images, text or metadata from Twitter, Facebook, Flickr, Pinterest and Instagram.

1.6 Blockchain data

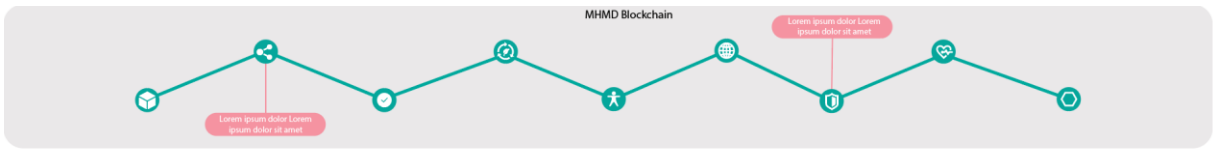

MHMD Blockchain
Lorem ipsum dolor Lorem ipsum dolor sit amet
Lorem ipsum dolor Lorem ipsum dolor sit amet

Figure 10: General blockchain data box

This block is used to displays data related to the blockchain. It could be specifications of the blockchain: advantages, costs, transparency, decentralization… I can also display some data like: Number of participants by types (hospitals, research labs, digi.me…), number of transactions, etc.

The nodes have different logos and reveals their information on hover, with a soft popup.

1.7 Repartition of studies

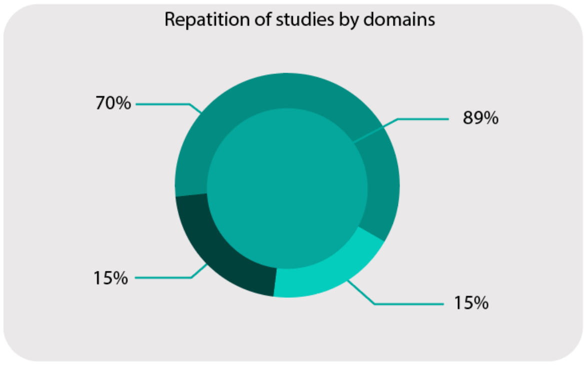


Figure 11: Studies distribution box

A detailed sunburst graph is used to display the repartition of studies per types: Hospitals, research labs… It's interactive and can be zoomed in or out.

1.8 Transactions

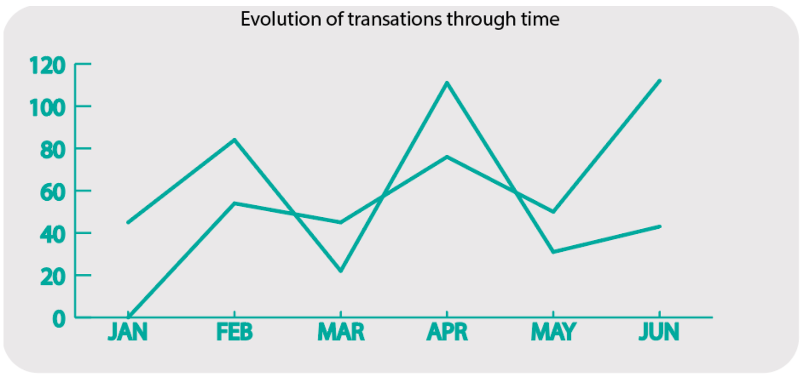


Figure 12: Transaction chart box

A detailed chart graph using blockchain data to display the evolution of transactions through time. It can be relative to any period of time.

1.9 FedEHR data

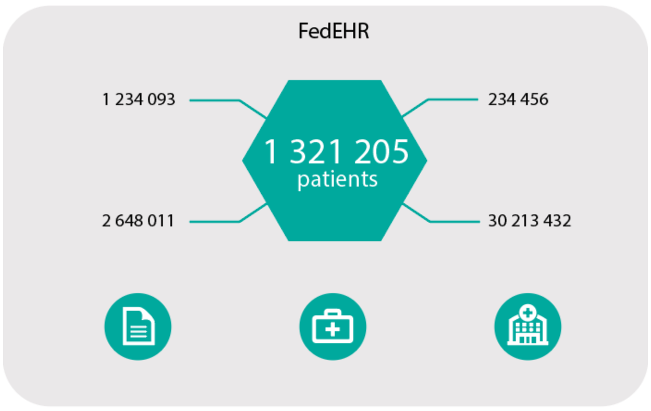


Figure 13: FedEHR data box

This block contains all the relevant data gathered in FedEHR like:

- Number of patients
- Number of clinical variables
- Number of medical events
- Number of hospitals

References

[1] Hyperledger-fabric.readthedocs.io. “Hyperledger Fabric Model”, Hyperledger-fabricdocs master documentation, 2017

[2] Hyperledger-fabric.readthedocs.io. “Ledger”, Hyperledger-fabricdocs master documentation, 2017

[3] Hyperledger-fabric.readthedocs.io. “Architecture Explained”, Hyperledger-fabricdocs master documentation, 2017

[4] Hyperledger-fabric.readthedocs.io. “Chaincode Tutorials”, Hyperledger-fabricdocs master documentation, 2017

[5] Hyperledger-fabric.readthedocs.io. “Chaincode for Developers”, Hyperledger-fabricdocs master documentation, 2017

[6] Hyperledger-fabric.readthedocs.io. “Chaincode for Operators”, Hyperledger-fabricdocs master documentation, 2017

[7] Hyperledger-fabric.readthedocs.io. “Chaincode Swimlanes”, Hyperledger-fabricdocs master documentation, 2017

[8] https://hyperledger.github.io. “Welcome to Hyperledger Composer”, Hyperledger Composer documentation, 2017