Glass-IPFS Documentation

Authors:

Ben McKenzie 40536785

Conor King 40532386

Gabor Uzonyi 40429825

Ross Hunter 40478443

Stephen Park 40534021

Valentin Kisimov 40439132

Contents

What is GLASS

EU research project which aims to establish a common infrastructure at the European level to provide shared public storage for documentation, files and data, and hosting services to cross-sector organizations.

GLASS aims to design and deploy a blockchain-based distributed environment and deliver the operational framework for sharing common services including:

- The definition of the resource requirements
- the incorporation of the Interplanetary File System (IPFS) in the GLASS architecture
- the design, development, and deployment of the distributed ledger
- seamless identity management

What is a Distributed Ledger?

A distributed ledger is essentially a database that several different peers or nodes agree to host and synchronise between themselves. To solve disputes about the state of content on the ledger the nodes will agree upon the state which most nodes currently have, but this can be configured within Hyperledger Fabric. One of the main benefits this system being distributed is that nodes are redundant, and the network does not require all nodes to be online to function properly.

What is Hyperledger Fabric?

The Hyperledger foundation is an organisation focused on the development of blockchain projects and bringing these together into a cohesive ecosystem. One of these projects is Hyperledger Fabric, this is an open-source distributed ledger supported by the Linux Foundation and other contributors such as IBM. It offers a core architecture for building permissioned blockchains focused on the needs of private businesses while also remaining modular so it can adapt to specific edge case needs. It has been successfully deployed in many multinational companies including Honeywell, Walmart, and Splunk Technology. (Hyperledger, n.d.)

Unique Features of Hyperledger Fabric.

Unlike popular distributed ledgers such as Ethereum and Solana, Fabric is a permissioned blockchain that runs on an organisation's network/s this offers greater control over the data and the service configuration while also ensures that trust and transparency are maintained.

Like common permissionless ledgers Fabric supports the use of smart contracts, referred to a as Chaincode within the ecosystem. Chaincode is a program that is hosted on the ledger and can be called using a transaction invoke, this will perform the desired task with the arguments provided while also logging the entire interaction to ensure accountability. Furthermore, Chaincode can be written in several common languages including JavaScript, Java and Go-Lang offering more flexibility.

A core feature of Fabric is how it can manage private data. Ledgers can be completely isolated from un-authorised peers using separate channels i.e., Channel A cannot interact or view the ledger of

Channel B. Furthermore, if an organisation requires access to some of the ledger or requires some restriction on interaction this can be achieved through private data collections which allows grouping of authorised peers and organisations and their relevant permissions through a configuration file.

Authentication is achieved through the use of Public Key Infrastructure; each organisation has its own certificate authority that validates the identity of the organisation peers upon and chaincode invoke and query ensuring that man in the middle attacks are very difficult to achieve.

Our Aims with Hyperledger Fabric:

Our core aim is to use Hyperledger Fabric provide a distributed ledger that hosts a unique identification for the citizens of the European Union along with a Content ID that can be used in conjunction with IPFS to host personal details. Fabric will help ensure that this data is immutable and any interaction with the ledger through the Chaincode will be logged by Fabric for accountability and transparency.

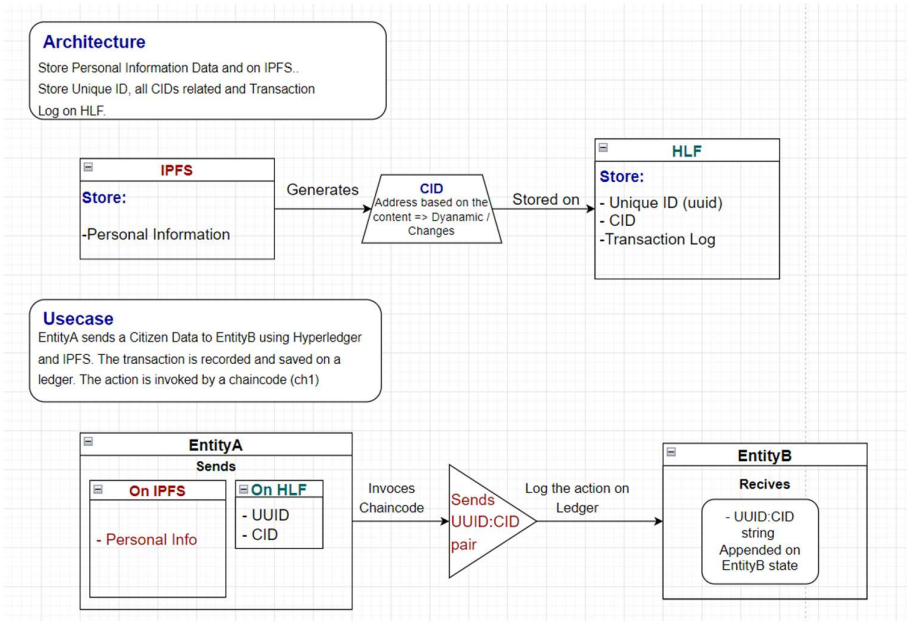

Figure 1: The flow chart represents our program flow

Smart Contracts

To support the consistent update of information and to enable a whole host of ledger functions (transacting, querying, etc.) a blockchain network uses smart contracts to provide controlled access to the ledger. Smart contracts to interact with the channel ledger. Smart contracts contain the business logic that governs assets on the blockchain ledger. Applications run by members of the network can invoke smart contracts to create assets on the ledger, as well as change and transfer those assets. Applications also query smart contracts to read data on the ledger. Hyperledger Fabric users often use the terms smart contract and Chaincode interchangeably. Chaincode can be implemented in several programming languages. Currently, Go, Node.js, and Java Chaincode are supported. In general, a smart contract defines the transaction logic that controls the lifecycle of a business object contained in the world state. It is then packaged into a Chaincode which is then deployed to a blockchain network. Think of smart contracts as governing transactions, whereas

Chaincode governs how smart contracts are packaged for deployment. *(*Consensus *Smart Contracts and Chaincode — hyperledger-fabricdocs)*

The process of keeping the ledger transactions synchronized across the network to ensure that ledgers update only when transactions are approved by the appropriate participants, and that when ledgers do update, they update with the same transactions in the same order is called consensus. Transactions must be written to the ledger in the order in which they occur. For this to happen, the order of transactions must be established and a method for rejecting bad transactions that have been inserted into the ledger in error (or maliciously) must be put into place. Hyperledger Fabric has been designed to allow network starters to choose a consensus mechanism that best represents the relationships that exist between participants. *(Introduction — hyperledger-fabricdocs main documentation)*

What is Self-sovereign identity SSI

Self-sovereign identity (SSI) is a term used to describe the digital movement that recognizes an individual should own and control their identity without the intervening administrative authorities. SSI allows people to interact in the digital world with the same freedom and capacity for trust as they do in the offline world. With SSI, the power to control personal data resides with the individual, and not an administrative third party granting or tracking access to these credentials. The SSI identity system gives you the ability to use your digital wallet and authenticate your own identity using the credentials you have been issued. You no longer must give up control of personal information to dozens of databases each time you want to access new goods and services, with the risk of your identity being stolen by hackers. This is called “self-sovereign” identity because each person is now in control of their own identity, they are their own sovereign nation. People can control their own information and relationships. A person’s digital existence is now independent of any organization: no-one can take their identity away. *(What is self-sovereign Identity? - Sovrin)*

What are Distributed Ledger Technologies (DLT)

Digital database which every member can supplement the data stored there. The data is stored locally on each machine not on a centralized cloud (decentral peer-to-peer network). Mining verifies the data and makes DLT transparent, safe, and decentral. Every blockchain is a form of DLT but not every DLT is blockchain. Nodes of the DLT are located on different locations.

Chaincode Implementation:

Chaincode development was based on “*simple*” chaincode provided by the minifab tool. The decision was motivated by the ease of use and simplicity of the chaincode. “*Marbles*” chaincode would possibly be a better option, but several issues were encountered that were not straightforward to solve, which was not the case with working with “simple” chaincode. As the “simple “chaincode is originally working with integer values, there were some modifications necessary, which led to a new chaincode that accepts strings as parameters and appends them successfully to the ledger. Multiple approaches were tried, but the final implementation was done by string manipulation, using the build-in functionality of Golang. Working with maps was hard as maps cannot be easily converted to a byte array. The idea behind the chaincode developed is that each entity has a “payload” in string format which have multiple key: Value pairs separated by white space.

- key – Unique ID generated using google uuid tool (https://github.com/google/uuid)

- cid which is generated by the IPFS when a file is uploaded.

Each string format is :“uuid:cid uuid:cid uuid:cid” - each uuid and cid are unique and each pair is separated by a single blank space.

Initialisation

Firstly, the chaincode must be installed, approved, and committed. Then we must initialise entities with Key: Value pairs for example purposes. This is done by the command:

Command:

- *minifab initialize -p '"init","entityA","uniqueid1:cid1 uniqueid2:cid2","entityB","uniqueid3:cid3 uniqueid4:cid4"'*

-p specifies a method to be called, in this case - “init”

The method accepts 4 arguments – [0], [1], [2], [3]

[0] – 1st Entity.

[1] – Payload of 1st Entity (a string containing Key: Value pairs in the format specified above).

[2] – 2nd Entity.

[3] – Payload of 2nd Entity (a string containing key: Value pairs in the format specified above) This creates two entities.

The action is recorded on the ledger as an initialisation action.

Transaction (Asset Transfer)

Command:

- *minifab invoke -p '"invoke","a","b","uniqueid2"'*

Invoke accepts 3 arguments – [0], [1], [2]

[0] – Sender Entity (or Add or Delete functionality).

[1] – Receiver Entity.

[2] – Unique id of the Key: Value pair.

See line 134 in chaincode Strings (chaincodes/strings/main.go).

The function gets the state of the Sender and Receiver entity. The state is received in a byte array which is converted into a string. The string is split into a slice on “ “ a delimiter. The element of the slice is looped and searched for a prefix specified in argument [2] unique id. When the prefix matches the index of the element is saved and the element is appended to the string of the receiver entity. The element specified in the argument [2] is removed using the “*RemoveAtIndex*” function. The slice is converted to a string which is converted to a byte array and the state is of sender entity is updated. The transaction is logged as a transaction on the ledger.

Associate an asset to an Entity (Add)

https://pkg.go.dev/github.com/hyperledger/fabric-chaincode-go/shim

Command:

- *minifab invoke -p '"invoke","add","a","uuid:cid"'*

Invoke accepts 3 arguments – [0], [1], [2]

[0] – add.

[1] – Receiver Entity.

[2] – Key: Value asset pair.

The method used is invoke is not the most elegant solution, as the action is recorded as a transaction and not as an addition, but it works fine. The decision was partly forced as the ledger is configured in a way that accepts only query, invoke and delete methods, specified in the "simple" chaincode which "strings' chaincode is based on. The "simple" chaincode is an extension of this chaincode "https://pkg.go.dev/github.com/hyperledger/fabric-chaincode-go/shim", which cannot be modified. Lack of documentation and explanation of how to modify and create methods which are accepted as valid from the ledger led to the decision to use the same method with different arguments and flow control. The main downside is the lack of customisation of the action logged on the ledger and ease of use.

With a simple flow-control statement, the invoke method checks if the user specifies as argument [0] an *entity*, *add* or *delete*. In this case, "***add***" is supplied as argument [0].

Firstly, the state of receiver entity is recorded, converted to string and the Key: Value asset specified is appended to it. Then the string is converted back to byte array and the state of the entity is updated. The action is recorded as a transaction on the ledger.

Delete an asset:pair from an entity (delete)

Command:

- *minifab invoke -p '"invoke","delete","a","uuid:cid"'*

Explanation:

Invoke accepts 3 arguments – [0], [1], [2]

[0] – delete.

[1] – Receiver Entity.

[2] – uuid.

When argument [0] is **"delete"** we get the state only of the receiver entity, convert it to a string, then split it to a slice and loop it through searching for a uuid prefix specified in argument [2]. When the uuid matches the index is recoded and the function remove Index which accepts two methods slice index, which simply removes the element on the specified index is executed. After execution, the slice is converted back to string, the string to a byte array and the byte array is the new state of the ledger. The action is recoded as a transaction on the Ledger Log.

The log of actions can be examined by using the Hyperledger explorer web UI or the query method from strings chaincode.

To bring up the explorer:

- *“Minifab explorerup”*

minifab invoke -p '"query", "entity"' - to query the state of a specified entity.

IPFS Data Encryption

An encryption function has been implemented in the program. The main goal of the function is to store encrypted data on the IPFS network to protect PPI and meet standards. The encryption function uses the asymmetric encryption method, AES. Moreover, a decryption function is implemented as well. Symmetric key-based encryption might not be the best for the project, however, in a future version, asymmetric key encryption can be implemented with a rolling key.

Logging

Logging in the Hyperledger fabric uses the “common/flogging” package and is usable from the “peer” and “orderer”. The package can control the logging based on the severity of the message or on the software logger generating the message. You can change how it prints the information in pretty-printing options. (*Logging Control*)

Example:

```
2018-11-01 15:32:38.268 UTC [ledgermgmt] initialize -> INFO 002 Initializing ledger mgmt
2018-11-01 15:32:38.268 UTC [kvledger] NewProvider -> INFO 003 Initializing ledger provider
2018-11-01 15:32:38.342 UTC [kvledger] NewProvider -> INFO 004 ledger provider Initialized
2018-11-01 15:32:38.357 UTC [ledgermgmt] initialize -> INFO 005 ledger mgmt initialized
2018-11-01 15:32:38.357 UTC [peer] func1 -> INFO 006 Auto-detected peer address: 172.24.0.3:7051
2018-11-01 15:32:38.357 UTC [peer] func1 -> INFO 007 Returning peer0.org1.example.com:7051
```

Figure 2 – An example of the output from the logger.

The logging levels of “peer” and “orderer” are controlled by the logging specification “FABRIC_LOGGING_SPEC”. Logging severity is specified by using a case-insensitive string from below.

FATAL | PANIC | ERROR| WARNING | INFO | DEBUG

Chaincode logging can be implemented by the chaincode developer. When developing the chaincode it is possible to have outputs on stdout/stderr. This can be very useful for development of the system. Normally this is disabled and will need to be activated by “CORE_VM_DOCKER_ATTACHSTDOUT=true” config option. This is done on a per-peer basis.

Once enabled each chaincode will receive its own logging channel with a key of the container-id. Any stdout or stderr will be integrated to the peer’s log on a per-line basis. This is not recommended for using in production.

“docker logs <chaincode_container_id>”

Moreover, it would be Ideal to tunnel these loges into a SIEM (Security information and event management). Splunk has an add-on that can help analyze the data. The name of the add-on is:

Splunk App for Hyperledger Fabric. This can help monitor and stop Denial of Service (DoS) attacks, Key Theft attacks, Network Partitioning attacks, Consensus Manipulation, and Blockchain Integrity Attacks.

Access Control

Access Control is essential in protecting the private data of EU citizens unfortunately we did not meet this requirement in our development phase, but the following will describe how we would implement it.

Private Data Collections

Private data collection is a method of restricting access to specific organisations on within the same channel. Consider a Heath department and Tax department within France, both organisations handle extremely important information that very rarely need to interact. With a data collection implemented each organisation could hold a private copy of this data on a personal ledger shared peer to peer using the Gossip protocol while a hash of this data is put on the public ledger. The organisations privy to this sensitive data can be defined in a private collection JSON file using signature definition, it is also possible to define what organisations can write to this data collection in the definition affording more fine control over access.

Chain code level access

It is possible to define read and write access within the chain code itself, this can be done by using the GetMSPID() for organisation level control or GetCreator() function for user level control. This within the chaincode could be used in a switch statement to check if submitter meets the allowed organisations or user. Ideally this would be defined in an external JSON file for more central control.

Implementing both methods would offer some level of redundancy in checks meaning if one was by-passed another check would still be performed. Although this would come with the drawback of making maintenance of the configuration files more difficult. A possible solution for this would be making use of the same private data collection configuration JSON but parsing this data would be more difficult to implement.

IPFS

A blockchain can store data, however, the distributed ledger technology is not designed to do so. The biggest problem with storing data in a blockchain is an issue with scalability and related transaction cost. In addition, query data from a blockchain can be slow due to the mentioned scalability issue. IPFS is an alternative secure and immutability storage solution, which was designed to store data. (Marx, 2018)

IPFS is a peer-to-peer (p2p) storage network. Content is accessible through peers located anywhere in the world, which might relay information, store it, or do both. The main idea of the IPFS is to have many nodes all over the world synchronize their data, therefore the storage solution is immutable.

CID

IPFS knows how to locate what you ask for utilizing its content address (CID) instead of the location of the data. CID is a label used to point to material in IPFS. The CIDs are based on the content's cryptographic hash function, therefore any difference in the content will produce a different CID.

The same content added to two different IPFS nodes using the same settings will produce the same CID. IPFS splits content into blocks and verifies them through directed acyclic graphs (DAGs), SHA file hashes will not match CIDs. However, the CID is still deterministic, uncorrelated, unique and a one-way function. (IPFS, n.d.)

Immutability

An immutable object is an object whose state cannot be altered or modified once created. Once a file is added to the IPFS network, the content of that file cannot be changed without altering the content identifier (CID) of the file (IPFS, n.d.). However, when it comes to content that needs to be altered or updated, immutability becomes a problem. Currently, the IPFS system has no version control (Jbenet, 2015).

Install IPFS:

This document will present how to set up a privet-IPFS network. The private IPFS (Inter Planetary File System) network does not connect to public IPFS nodes.

Automatic installation with the snap manger

- *"sudo snap install ipfs"*

Manual Installation

Download the Linux binary from dist.ipfs.io

- *"wget https://dist.ipfs.io/go-ipfs/v0.12.0/go-ipfs_v0.12.0_linux-amd64.tar.gz"*

Unzip the file:

- *"tar -xvzf go-ipfs_v0.12.0_linux-amd64.tar.gz"*

Move into the go-ipfs folder and run the install script:

- *"cd go-ipfs"*
- *"sudo bash install.sh"*

Test that IPFS has installed correctly:

- *"ipfs –version"*

Setup a private network with the swarm key

Create shared key

Swarm.key The key allows us to create a private network and tells the network node to only communicate with the node with the same key. Execute the following command on one node:

- *"go get -u github.com/Kubuxu/go-ipfs-swarm-key-gen/ipfs-swarm-key-gen ipfs-swarm-key-gen > ~/.ipfs/swarm.key"*

Remove the default bootstrap node

In order not to connect to the global IPFs network, you need to delete the node information of the default bootstrap.

- *"ipfs bootstrap rm –all"*

Private network node configuration

Add the bootstrap of the first node in each node

- *"ipfs bootstrap add /ip4/192.168.11.11/tcp/4001/ipfs/QmSyQFFm5KdB9zoTNQJhPnMs4LYsVmEpY427QYxH2CaFqL"*

QmSyQFFm5KdB9zoTNQJhPnMs4LYsVmEpY427QYxH2CaFqLbyipfs init

The node ID generated by the IPFS id View the ID of the current node.

We also need to set environment variables "LIBP2P FORCE PNET" To force our network into private mode

- *"export LIBP2P_FORCE_PNET=1"*

Add the IPFs process to the system process to start

Each IPFs boot is added to the system's daemons to start, add/etc/systemd/system/ipfs.service

The background daemons of IPFs can be started with the following command:

- *"systemctl daemon-reload"*
- *"systemctl enable ipfs"*
- *"systemctl start ipfs"*
- *"systemctl status ipfs"*

Installation

IPFs cluster consists of two components:

- *"ipfs-cluster-service"* - The daemons used to initialize the cluster peer and run it
- *"ipfs-cluster-ctl"* - Manage the nodes and data of the cluster

We will ipfs-cluster Clone to the Gopath, then compile and install make (the system has installed make)

- git clone https://github.com/ipfs/ipfs-cluster.git $GOPATH/src ipfs-cluster
  make install

Check to see if the installation is successful:

- *"ipfs-cluster-service –version"*
- *"ipfs-cluster-ctl –version"*

Set cluster key

Similar to the secret key of IPFs, generate a random key in the management node:

- *“od -vN 32 -An -tx1 /dev/urandom | tr -d ' \n' ”*

Add the generated random string to the environment variable, for example:

b55262c36de6f97bd50b5233f75866445ec51db74613bad78e906c4dc9ba1d30

Modify each node’s “~/.bashrc” Add to environment variable:

- *export “CLUSTER_SECRET=b55262c36de6f97bd50b5233f75866445ec51db74613bad78e906c4dc9ba1d30 “*

Do not forget to save it source ~/.bashrc

Initialize cluster

Each node executes the initialization command:

- *“ipfs-cluster-service init“*

Start the process at the management node:

- *“ipfs-cluster-service daemon“*

Other nodes start—bootstrap Add master node:

- *ipfs-cluster-service daemon --bootstrap /ip4/192.168.11.11/tcp/9096/ipfs/12D3KooWEGrD9d3n6UJNzAJDyhfTUZNQmQz4k56Hb6TrYEyxyW2F*

Notice that; “*12D3KooWEGrD9d3n6UJNzAJDyhfTUZNQmQz4k56Hb6TrYEyxyW2F”* it is the IPFs cluster node ID, not the IPFs node ID. You can use the ipfs-cluster-service id see.

The cluster node status can be viewed through the command:

- *“ipfs-cluster-ctl peers ls”*

Add the IPFs cluster node to the system process to start

- *“add/etc/systemd/system/ipfs-cluster.service:”*

The background daemons of IPFs cluster can be started with the following command:

- *“systemctl daemon-reload”*
- *“systemctl enable ipfs-cluster”*
- *“systemctl start ipfs-cluster”*
- *“systemctl status ipfs-cluster”*

Test cluster data replication

Add a file to one of the nodes:

- *“ipfs-cluster-ctl add test.txt”*

The file status can be viewed through the added file CID, and you can see the file and its status in all nodes PINNED

- *“ipfs-cluster-ctl status CID”*

Please note: Reprinted from notes of Ryan is a rookie LNMP technology stack (Ryan, 2021)

Security:

In the real-world application of this project, many nodes would be used. To have Geo-redundant storage. To establish a connection between nodes the internet would be used, which could open the system for cyber-attacks. Therefore, the system needs to be hardened, our recommendation is two uses of Defense-in-depth and the zero-trust model.

Firstly, IPFS has a private key system called swarm key to protect the network. Only nodes which the right swarm key can connect to the IPFS network, however, if the key leaks any bad actors can connect to the network. A possible idea to improve security could be change the key regularly in a secure way.

Secondly, nodes should have a firewall rule which blocks all incoming traffic unless the packet is originated from a known node IP address, furthermore, the same rule should be implemented in a network firewall. To implement a zero-trust type environment all outbound traffic needs to be filtered, only connections to known IPFS nodes should be allowed.

For a higher level of security, additional security systems are recommended such as IDS/IPS and an application-level firewall as well which should be monitored 24/7 in a SIEM. Moreover, the Physical security of the servers should be considered. Lastly, it might be a good idea to create a site-to-site VPN connection between nodes for additional security. However, the architecture for such a system needs to be considered carefully for high availability.

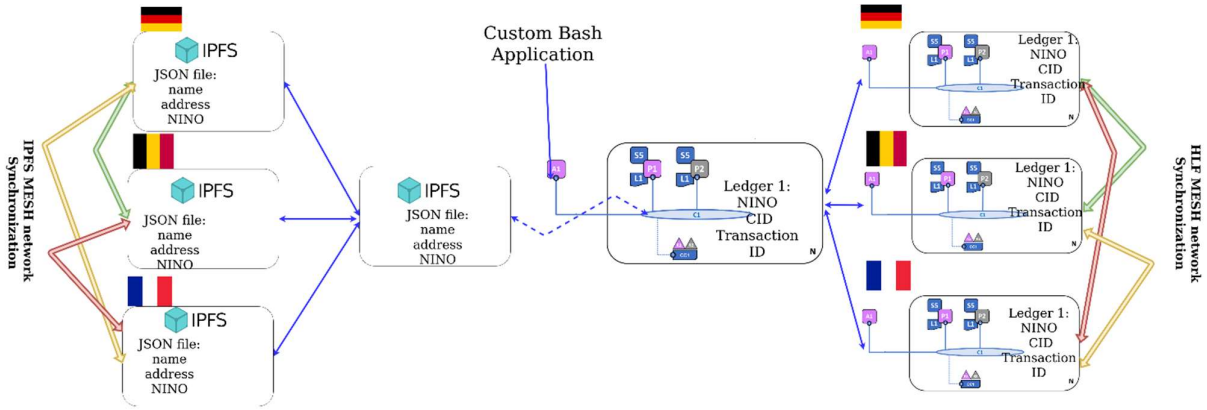

Figure 3: The above diagram is a visual representation of our architecture.

Limitation of the architecture:

The system is not truly a distributed system because if our main IPFS or our main HLF goes down the whole system is down. Moreover, the system has a careerist in subordinate network design. This can be improved in a further version. A possible improvement to create the system fully decentralized could be based on current DEFI technology. Secondly, an alternative solution could be using a redundant connection from our application to the network.

Acknowledgement

We would like to express our sincere gratitude to several individuals and organizations for supporting us throughout our project. First, we wish to express our sincere gratitude to our supervisor, Dr Nick Pitropakis, for his enthusiasm, patience, insightful comments, helpful information, and practical advice. Moreover, we would like to express our gratitude to Dr. Sarwar Sayeed and to Dr. Owen Lo. Lastly but not least, we would like to thank you to Dr Naghmeh Moradpoor Sheykhkanloo for sponsorship during the project.

References

Hyperledger. (n.d.). *Hyperledger Fabric Model*. Retrieved from Hyperledger Fabric Docs: https://hyperledger-fabric.readthedocs.io/en/release-2.2/fabric_model.html

Hyperledger. (n.d.). *Introduction*. Retrieved from Hyperledger Fabric Docs: https://hlf.readthedocs.io/en/latest/blockchain.html

Hyperledger. (n.d.). *Logging Control*. Retrieved from Hyperledger Fabric Docs: https://hyperledger-fabric.readthedocs.io/en/release-2.2/logging-control.html

Hyperledger. (n.d.). *Open, Proven, Enterprise-grade DLT*. Retrieved from https://www.hyperledger.org/wp-content/uploads/2020/03/hyperledger_fabric_whitepaper.pdf

Hyperledger. (n.d.). *Smart Contracts and Chaincode*. Retrieved from Hyperledger Fabric Docs: https://hyperledger-fabric.readthedocs.io/en/release-2.2/smartcontract/smartcontract.html

IPFS. (n.d.). *Command Line*. Retrieved from IPFS Docs: https://docs.ipfs.io/install/command-line/#official-distributions

IPFS. (n.d.). *Content addressing and CIDs*. Retrieved from IPFS Docs: https://docs.ipfs.io/concepts/content-addressing/#identifier-formats

IPFS. (n.d.). *Immutability*. Retrieved from IPFS Docs: https://docs.ipfs.io/concepts/immutability/

IPFS. (n.d.). *Run IPFS inside docker*. Retrieved from IPFS Docs: https://docs.ipfs.io/how-to/run-ipfs-inside-docker/#set-up

Jbenet. (2015, 06 02). *Versioning: Commit + Repo Datastructures #23*. Retrieved from Github: https://github.com/ipfs/notes/issues/23

Marx, L. (2018, 06 05). *Storing Data on the Blockchain: The Developers Guide*. Retrieved from Malcoded: https://malcoded.com/posts/storing-data-blockchain/

Ryan. (2021, 03 29). *Construction of IPFs private network cluster*. Retrieved from Develop PAPER: https://developpaper.com/construction-of-ipfs-private-network-cluster/

Sovrin. (2018, 12 06). *What is self-sovereign Identity*. Retrieved from Sovrin: https://sovrin.org/faq/what-is-self-sovereign-identity/