**Project 1 Report**

**Title**: Hyperledger Fabric Private Blockchain and Smart Contracts

**Student Name:** Tiriac Ioana-Raluca

**ASU ID:** 1217060520

**Abstract:**

This project revolves around writing a smart contract for managing patient record digital assets on the blockchain implemented using the Hyperledger Fabric framework, by updating and completing a codebase provided by the teaching staff of the course. A method had to be written that creates patient record assets on the Hyperledger Fabric blockchain network, one method had to be updated that updates one attribute of a patient’s record and several functions that allow reading/accessing/querying the informations about a patient had to be completed by using data indexing and querying in the lightweight NoSQL database CouchDB used by the smart contract.

**Keywords:**

Hyperledger Fabric, smart contract, chaincode, permissioned blockchain, patient record, digital assets

**Introduction:**

The last couple of years have brought increase in popularity of blockchain technology with numerous projects being implemented by private and public entities. It is however an emerging technology and quickly evolving technology too. By creating a decentralized system, it removes the indulgence of central servers and provides peer-to-peer interaction. It can create a fully transparent and open to all database, which could bring transparency to the governance and elections.

**Terminology:**

term 1:

“Hyperledger Fabric is a modular [blockchain](https://www.investopedia.com/terms/b/blockchain.asp) framework that acts as a foundation for developing blockchain-based products, solutions, and applications using plug-and-play components that are aimed for use within private enterprises.”[[1]](#footnote-1)

term 2:

A smart contract is according to Wikipedia “a [computer program](https://en.wikipedia.org/wiki/Computer_program) or a [transaction protocol](https://en.wikipedia.org/wiki/Transaction_Protocol_Data_Unit) which is intended to automatically execute, control or document legally relevant events and actions according to the terms of a [contract](https://en.wikipedia.org/wiki/Contract) or an agreement. The objectives of smart contracts are the reduction of need in trusted intermediators, arbitrations and enforcement costs, fraud losses, as well as the reduction of malicious and accidental exceptions.”[[2]](#footnote-2)

term 3:

Chaincode is an important piece of the Hyperledger Fabric’s architecture and represents the program that runs on top of the blockchain and executes the business logic of how applications interact with the ledger.[[3]](#footnote-3) “Chaincode is a program, written in [Go](https://golang.org/), [node.js](https://nodejs.org/), or [Java](https://java.com/en/) that implements a prescribed interface. Chaincode runs in a secured Docker container isolated from the endorsing peer process. Chaincode initializes and manages ledger state through transactions submitted by applications”.[[4]](#footnote-4)

term 4:

A permissioned blockchain is a private blockchain that requires permission to access.

term 5:

Digital asset = anything available in digital format that comes with a right to use [[5]](#footnote-5)

**Goal Description**:

**Phase I:**

1. The first step of the project involved carefully reading up on the assignment tasks and researching the linked materials about smart contracts to understand the code base.

**Phase II:**

1. Started working on completing the given project, by completing the *getPatientByKey()* function in the patientrecordcontract.js, that receives a patient’s username and name and retrieves the patient’s record by searching after this composite key.
2. In the next step, the setter and getter for the *lastCheckupDate* field had to be implemented in the patientrecord.js, a class that represents a patient’s record.
3. In the smart contract(*patientrecordcontract.js*), the *updateCheckupDate()* method had to be completed, by receiving a new checkup date, retrieving a patient’s record by inputing her username and name and storing the patient record again with the updated date.
4. The next step was a preparatory one necessary to successfully complete the next tasks and involved the creation of a new index for the attribute “blood\_type” for the lightweight NoSQL database “CouchDB” from Apache used by the smart contract, the new index was created in order for the database to perform faster searches involving query strings related to the attribute “blood\_type”.
5. The next task was about completing the *queryByGender()* function of the smart contract, by using CouchDB’s documentation on querying using indexes.
6. The next step required completing another similar query function – querybyBlood\_Type() function, that like the previous task required database querying by using our previously created index on blood\_type.
7. The next challenge involved writing the *queryByBloodType\_Dual()* function, that took as input the transaction context and 2 blood types and required constructing the CouchDB selector query-string object that uses the 2 blood\_type indexes.
8. The last step of the project required implementing the *unknownTransaction()* function of the smart contract, a default function executed every time an unknown, non-existing function is invoked on the smart contract.

**Description of proposed solution:**

The first phase of the project involved understanding and reading about what smart contracts are and the particularities for the used technology - Hyperledger Fabric. The next parts of the project were accomplished by completing the code base function by function and submitting and debugging at every step to check the correctness of the implementation.

Phase I:

Hyperledger Fabric is an open-source enterprise-grade permissioned distributed ledger technology. In Hyperledger Fabric smart contracts manipulate and manage JSON like digital assets, meaning they offer “the contract” – reading, updating, writing data – to the given digital asset. Each digital asset needs to have a smart contract in place if it’s to be stored on the blockchain. In Hyperledger Fabric smart contracts are packaged into chaincodes and chaincode is deployed on the Hyperledger Fabric blockchain, chaincode being a term specific to the “Hyperledger Fabric” technology.

Phase II:

2. In *patientrecordcontract.js* (the smart contract) the *getPatientByKey* function was implemented, that receives a patient’s username and name as parameters. This was done by using the transaction context (ctx) of the smart contract, that has access to the patientRecordList. The patient record was retrieved by invoking *getPRecord()* and using the composite key – username + name on the *ctx.patientRecordList* and then awaiting the result – because *ctx.patientRecordList.getPatientByKey()* processes the result asynchronously and we need to await the result in order to return it.

3. In the next step, a new attribute was added to *patientrecord.js* (a class representative of a patient’s record)), the *lastCheckupDate* attribute representing the most recent date a patient had a physical exam.

The setter and getter for the lastCheckupDate attribute were implemented in the *patientrecord.js,* similar to other fields.

4. The *updateCheckupDate()* function was implemented, by receiving transaction context “ctx”, the patient’s username, name and checkup date. Again the asynchronous *getPRecord* function of the *ctx.patientRecordList* receives the composite key – username and name and by “awaiting” it we get the patient record, we update the checkup date of this record and save the patient record again – by calling the *ctx.patientRecordList.updatePRecord()* and “awaiting” for it to finish processing again.

5. For this step a physical index “blood\_typeIndex.json” was created under the path Project folder>META-INF>statedb>couchdb>indexes as a JSON file, similar to genderIndex.json . The fields of this index are “blood\_type”, and the “ddoc” and “name” were given values according to the name of the field. The creating indexes enable the NoSQL database Couchdb to perform faster searches on query strings related to fields that have been indexed.

6. The *queryByGender()* function in the smart contract takes transaction context and gender as input. Then the query-string object is built by using CouchDB documentation, it uses the genderIndex.json and this is specified by the key value *“use\_index”: “genderIndexDoc”* pair (genderIndexDoc being a reference to the genderIndex.json), and the selector {gender: genderParameter}. Afterwards the query-string object is stringified and passed to the more general *queryWithQueryString(ctx, query)* method of the smart contract. This function will return a list of records that correspond to the gender that is passed, and this list is also returned from the queryByGender() function.

7. The *queryByBlood\_Type()* function was implemented similarly toqueryByGender() function by taking as input the transaction context ctx and a string representing the blood type. It then builds up a query string by using as a selector the blood type and using the previously created blood\_typeIndex.json by referencing it with the attribute *use\_index: ”blood\_typeIndexDoc”* inside the query string. Once the query string is built it’s passed to the *queryWIthQueryString(ctx, query)* method, which will return a list of records that correspond to the blood type passed as the input, and the same list will be returned from the *queryByBlood\_Type()* function as well.

8. The next query function – *queryByBlood\_Type\_Dual()* function takes the transaction context and two blood types as input. The key to completing this function is in writing a query string that uses an appropriate selector for the field “blood\_type”, my solution involved using the “or” operator to combine selectors :

let query = {  
 selector: {  
 "$or": [  
 {blood\_type: blood\_type1},  
 {blood\_type: blood\_type2}  
 ]  
 },  
 use\_index: 'blood\_typeIndexDoc'  
};

and of course by referencing the previously created *blood\_TypeIndex.json*. The **$or operator** simply states the *blood\_type1* or *blood\_type2* input parameters should be selected. The *queryString* is again passed to the *queryWithQueryString(ctx, query)* method and the returned list of records corresponds to either *blood\_type1* or *blood\_type2* passed as input.

1. The *unknownTransaction(ctx)* function inside the smart contract (patientrecordcontract.js) takes the transaction context as input and is called every time an unknown/non-existing function is invoked that doesn’t exist in the smart contract. This default function throws an error with the default error message “Function name missing”.

**Issues Faced and Methods used to resolve them:**

I had to submit multiple times to get to the correct implementation, learned something new each time and managed to build the right solution by following the debug messages of the grader tool. The existing documentation on CouchDB and the detailed description of the assignment helped.

**Conclusions**:

Independently of the blockchain technology used, smart contracts are supposed to be lightweight, they represent the business applications written on the blockchain and establish a digital contract or interface for the digital assets they manage. Their purpose is to eliminate 3rd party intermediators, enforcements costs and fraud losses.

**Bibliography**:

<https://hyperledger-fabric.readthedocs.io/en/release-1.4/whatis.html>

<https://www.ibm.com/blogs/blockchain/2018/07/what-are-smart-contracts-on-blockchain/>

<https://github.com/hyperledger/fabric-samples/tree/master/commercial-paper/organization/magnetocorp/contract>

<https://docs.couchdb.org/en/stable/api/database/find.html#selector-syntax>

project1 – assignment-instructions on Coursera

1. <https://www.investopedia.com/terms/h/hyperledger-fabric.asp> [↑](#footnote-ref-1)
2. <https://en.wikipedia.org/wiki/Smart_contract> [↑](#footnote-ref-2)
3. <https://medium.com/kokster/understanding-hyperledger-fabric-chaincode-e7767d50f0b4> [↑](#footnote-ref-3)
4. <https://hyperledger-fabric.readthedocs.io/en/release-2.0/chaincode.html#:~:text=Chaincode%20is%20a%20program%2C%20written,through%20transactions%20submitted%20by%20applications>. [↑](#footnote-ref-4)
5. <https://en.wikipedia.org/wiki/Digital_asset> [↑](#footnote-ref-5)