Ira A. Fulton Schools of Engineering
Arizona State University

Course Project2 - Fall B 2022
CSE 598: Engineering Blockchain Applications

Project2: Creating Smart contract for Permissioned Blockchain Systems

This document contains a description of the tasks required to complete the PROJECT 2 assignment. This project will help you familiarize yourself with a private blockchain ecosystem by understanding, examining, writing, and executing smart contracts (Chaincodes) for a simple-use case of product records management.

For this project, you will be working with the Hyperledger Fabric blockchain framework. Hyperledger is an open-source community focused on developing a suite of stable frameworks, tools, and libraries for enterprise-grade blockchain deployments. Hyperledger was established under the Linux Foundation. It serves as a neutral home for various distributed ledger frameworks including Hyperledger Fabric. Hyperledger Fabric is an open-source enterprise-grade private distributed ledger technology(DLT)platform, designed for use in enterprise contexts, that delivers some key differentiating capabilities over other popular distributed ledgeror blockchain platforms. Read more about Hyperledger Fabric on this link (https://hyperledger-fabric.readthedocs.io/en/release-1.4/whatis.html)

➔ Required Software, if you want to create Blockchain Network locally and test smart contract on local system.

- NodeJS
- NPM
- Hyperledger FabricSetup

Smart Contract

Smart contracts are mediums that are used to manage digital assets, store information, make decisions, and interact with other smart contracts. Hyperledger Fabric smart contracts usually manipulate JSON-like digital assets (arrays of key-value pairs) of any complexity. For every digital asset we want to store on a Hyperledger Fabric blockchain, there must be a smart contract in place for its management (writing data on the blockchain, updating, reading,etc.).

In Hyperledger Fabric smart contracts are packaged into chaincodes and the chaincode is deployed on the Hyperledger Fabric blockchain. Chaincode is a term local to the Hyperledger Fabric framework and, for now, you can think of chaincode and smart contract as synonyms. To read more about chaincode in Hyperledger Fabric, visit the link.

Writing smart contacts on Hyperledger Fabric network requires three classes: State class, Contract Class, Statelist class.

- ➢ State class: Used to represent the asset on which the smart contract will be applied
- ➢ Contract class: Used to define methods that are set up in the contract
- ➢ State List class: Used to interact with the blockchain network. In this class, we define methods to add, get and update data on the blockchain network.

To learn more about writing smart contracts using Hyperledger Fabric visit the link(https://hyperledger-fabric.readthedocs.io/en/release-1.4/chaincode4ade.html)

In this project, you will be writing a smart contract to manage patient records over the Hyperledger Fabric. For this project, you will receive a code base that needs to be completed so that the contract is fully functional. You will be in charge of complementing several smart contract functions.

1. A function that creates patient records on the Hyperledger Fabric blockchain network.
2. A function that updates one attribute of a patient record.
3. Several functions allow reading/accessing the information about the patient using CouchDB-enabled data indexing and querying.

In the provided code base you are provided with a NodeJS smart contract and you only need to complete functions in the following javascript files.

1. **patientrecord.js**: Reflects the blueprint of a single patient record and consists of the following attributes:
   - ◦ username
   - ◦ name
   - ◦ dob
   - ◦ gender
   - ◦ blood_type
2. **patientrecordcontract.js**: Responsible for accessing/updating/adding to the state database of Hyperledger Fabric.

Now that you are familiar with the concepts behind Hyperledger Fabric and smart contracts, we have examined the use-case behind this project, established what type of digital record our smart contract is supposed to manage (and what attributes it holds), and continue with executing the following tasks.

You need to complete the code base mentioned as TASK-X in the files.CTX IS ALWAYS THE FIRST PARAMETER IN A FUNCTION – DO NOT REMOVE IT.

✔ **Task 1 –** Complete the getPatientByKey function

Instructions

The getPatientByKey receives a patient's username and name. These two attributes together make a composite key for this record, and by this key the record is searchable in the blockchain. The first line of this function creates such a key for you to use in the rest of the function. Complete the code of the getPatientByKey function so that it returns the record of the patient for the created composite key. Use a function from the PatientRecordList class, which can be referenced from inside the getPatientByKey by calling ctx.patientRecordList.[functionName], such as ctx.patientRecordList.addPRecord.

✔ **Task 2 –** Complete the getter and setter methods for lastCheckupDate field of the PatientRecord.

Instructions

Write code to add lastcheckupDate field to the PatientRecord. Complete these tlastCheckupDate function to set the lastCheckupDate field of the patientRecord. This function takes date as input and assigns it to the lastCheckupDate field. Complete the get lastCheckupDate function to return the lastCheckupDate field of thePatientRecord.

✔ **Task 3 –** Complete updateCheckupDate function

Instructions

updateCheckupDate function receives the patient's username, name, transaction context and checkupDate. To update the patient's last checkup date first retrieve the patient record by calling the ctx.patientRecordList.getPRecord. Update the lastCheckupDate to the PatientRecord using the function implemented in task 2. Update the PatientRecord on the ledger by calling ctx.patientRecordList.[Function Name]

Sample Code

```
Precord = await transaction_context.patientRecordList.getPRecord(precordKey);

Precord.[Setter Function Defined in Task 2](last_checkup_date);

Await ctx.patientRecordList.updatePRecord(precord);
```

➔ **Task 4 – 6** PREREQUISITES (CouchDB-enabled indexed querying)

For these tasks you will

• Build the indexes on the top of the attributes that are defined in the PatientRecordclass.

• Write the query processing transactions functions that make use of the indexes defined. For writing indexes follow the path:

Project_foldername>META-INF>statedb>couchdb>indexes

The indexes folder contains genderIndex.json, you need to complete/create

1. blood_typeIndex.json

After defining the indexes, complete the query functions in the PatientRecordContract class.

- ✔ **Task 4 –** Complete queryByGender function

Instructions

This function takes transaction context and gender as input. Construct the JSON string object for the genderIndex. Use helper functions from the PatientRecordContract class to query the database and return the list of records with that gender.

- ✔ **Task 5 –** Complete querybyBlood_Type function.

Instructions

This function receives transaction context and blood_type as input. Construct the JSON string object for the blood_typeIndex. Use helper functions from the PatientRecordContract class to query the database and return the list of records with the given blood_type.

- ✔ **Task 6 –** Complete querybyBlood_Type_Dual function.

Instructions

This function takes the transaction context and two blood types as input. Construct the JSON string object for the index of two blood_types. To make sure the query will actually use the index that you have created you must specify the use_index attribute inside the queryString. Once the queryString is built, pass it to the queryWithQueryString. Use the helper functions of the PatientRecordContract class to query the database and return a list of records with the given blood_types.

- ✔ **Task 7 –** Complete the unknownTransaction function

Instructions

In smart contracts, it is possible to get afunction's name wrong when calling the contract. In this case, the smart contract usually returns an error.A good practice is to implement a certain default function that will, instead, execute every time a function is invoked that does not exist in the smart contract. This default function is called unknown Transaction and it receives the transaction context only. The purpose of this function is to throw an error when a function called doesn't exist in the contract. Complete the function to return a string message ["Function namemissing"]

To read more about transactions handlers refer to:

https://hyperledger-fabric.readthedocs.io/en/release-1.4/developapps/transactionhandler.html