CSE598 Engineering Blockchain Applications

Project 1: Hyperledger Fabric Private Blockchain and Smart Contracts.

This document contains description of tasks required to complete the PROJECT 1 assignment. This project will help you familiarize with a private blockchain ecosystem by understanding, examining, writing and executing smart contracts for a simple-use case of patient 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 permissioned distributed ledger technology (DLT) platform, designed for use in enterprise contexts, that delivers some key differentiating capabilities over other popular distributed ledger or blockchain platforms. Read more about Hyperledger Fabric on this link (https://hyperledger-fabric.readthedocs.io/en/release-1.4/whatis.html).

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 chaincodes in Hyperledger Fabric, visit the link (https://hyperledger-fabric.readthedocs.io/en/release-1.4/chaincode.html).

In this project, you will be writing a smart contract that manages patient record digital assets. For this smart contract, you will receive a ode base that needs to be updated on certain places so that the contract is fully functional. You will be in charge of complementing several smart contract functions:

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

To learn more about the concept and development of Hyperledger Fabric smart contracts visit this link[1]. To see how a typical Hyperledger Fabric smart contract looks like (we will use the same template) visit this link[2]. Try to grasp and understand the code of this smart contract and its functions before proceeding forward. It will make a good exercise for successful completion of this project.

[1] https://hyperledger-fabric.readthedocs.io/en/latest/developapps/smartcontract.html

[2] https://github.com/hyperledger/fabric-samples/tree/master/commercial-paper/organization/magnetocorp/contract

In this project you are provided with semi-written project code (codebase). In some code blocks, you will find the following text: *"GRADED FUNCTION: Function Name"*. These functions are half-done: you will notice that they are missing some key lines of code for them to work when deployed. Your overall task is to update and finish the code for these functions and submit you work to the Grading Service.

Once again, the use-case behind this project is manage patient records. The smart contract must be able to write patient records on the blockchain and execute various queries based on attributes of patient records. In the provided code base, please examine the **patientrecord.js** file that defines how a patientrecord looks like. From this file we can conclude that a patient record consists of the following attributes:

- Username - text that uniquely identifies a patient
- Name - full name (first name and surname)
- Dob - date of birth of the patient
- Gender - self explanatory
- blood_type - text representing patient's blood type (A, AB-, etc.)

In the provided code base, specifically the patientrecordcontract.js file containing the code of our smart contract, you will notice that every function has one regular input parameter: **ctx**. This parameter refers to the transaction context within the execution of the function. At the beginning of the PatientRecordContract file, you can see the definition of the **PatientRecordContext** and its constructor, where **PatientRecordList** class is initialized. This is done so that the **PatientRecordList** object could be referenced in every function without initializing it every time, but simply callint **ctx. patientRecordList.[function]**. Inside the PatientRecordContract class there is a default createContext function that makes the connection between the smart contract and the context, enabling us to use code like **ctx. patientRecordList.[function]**. You can read more about the transaction context, motivation and use-cases behind it on this link[3]. **CTX IS ALWAYS THE FIRST PARAMETER IN A FUNCTION – DO NOT REMOVE IT.**

Now that you are familiar with the concepts behind Hyperledger Fabric and smart contracts, that 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), continue with executing the following tasks.

Task 0 – Complete the createPatientRecord() function in the PatientRecordContract class.

Get/retrieve the current PRecord by calling function in the PatientRecordList.js

Note: Remove the throw new error() from the createPatientRecord() function

Task 1 – Complete the getPatientByKey function (patientrecordcontract.js)

Instructions

The getPatientByKey function 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

[3] https://hyperledger-fabric.readthedocs.io/en/release-2.2/developapps/transactioncontext.html

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. (patientrecord.js)

Instructions

In this task you need to write code to add a new attribute to a patient record, the lastcheckupDate attribute representing the most recent date the patient had a physical exam (checkup). Complete the **setlastCheckupDate** function to set the lastCheckupDate field of the patientRecord. This function takes date as input and assigns it to the lastCheckupDate field. Complete the **getlastCheckupDate** function to return the lastCheckupDate field of the PatientRecord. The dates are text fields in the US date format. See get/set methods for other fields and apply the same model for **getlastCheckupDate and setlastCheckupDate.**

Task 3 – Complete the updateCheckupDate function (patientrecordcontract.js)

Instructions

The **updateCheckupDate** function receives the transaction context, patient's username, name and checkupDate. To update the patient's last checkup date first retrieve the patient record by calling the ctx.patientRecordList.getPRecord. The **getPRecord function** receives the composite key as an input parameter (the key is made from the username and name fields). Update the lastCheckupDate on the PatientRecord using the function implemented in task 2. Update the PatientRecord on the ledger by calling ctx.patientRecordList.[functionName]

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

For tasks 4-6 you will need to perform a crucial preparatory step, otherwise the tasks are not going to be graded – you have to physically build the index files on the top of the attributes that are defined in the **PatientRecord** class. For writing indexes follow the path: *Project_foldername>META-INF>statedb>couchdb>indexes.* Inside this folder you will find one file *genderIndex.json*. This folder structure and this file are telling the Hyperledger Fabric framework that, when deploying the smart contract, you also want to create indexes for certain attribute names for the records your smart contract is going to manage. Indexes enable the CouchDB to perform faster searches on all records given a certain query string. Our smart contract manages patient records, so we want to index two fields: (1) the blood_type and (2) gender. As the *genderIndex.json* already exist, create the **blood_typeIndex.json** file in the indexes folder with the same structure as *genderIndex.json*, but referencing another attribute.

Task 4 – Complete queryByGender function (patientrecordcontract.js).

Instructions

This function takes transaction context and gender as input. Your task is to construct the JSON CouchDB selector query-string object (*queryString*) that uses the genderIndex. See what CouchDB selector queries are and a few examples on this link[4]. 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 build, pass it to the *queryWithQueryString*. This function will return a list of records that correspond to the gender that is passed, and you need to return this list from the **queryByGender** function.

[4] https://docs.couchdb.org/en/stable/api/database/find.html#selector-syntax

Task 5 – Complete querybyBlood_Type function (patientrecordcontract.js).

Instructions

This function takes transaction context and blood_type as input. Your task is to construct the JSON CouchDB selector query-string object(*queryString*) that uses the blood_typeIndex. To make sure the query will use the index you have created you must specify the *use_index* attribute inside the *queryString*. Once the *queryString* is build, pass it to the *queryWithQueryString*. This function will return a list of records that correspond to the blood_type that is passed as the input and you need to return this list from the **querybyBlood_Type** function.

Task 6 – Complete querybyBlood_Type_Dual function (patientrecordcontract.js).

Instructions

This function takes the transaction context and 2 blood types as input. Your task is to construct the JSON CouchDB selector query-string object that uses the 2 blood_type indexes. 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 build, pass it to the *queryWithQueryString*. This function will return a list of records that correspond to the gender that is passed, and you need to return this list from the **queryByGender** function.

Use the helper functions of the PatientRecordContract class to query the query the database and return list of records with the given blood_types.

Task 7 – Complete the unknownTransaction function

Instructions

In smart contracts, it is possible to get a function'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 *unknownTransaction* 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 Name Missing"]

To read more about unknown transaction refer:

https://hyperledger.github.io/fabric-chaincode-node/master/api/fabric-contract-api.Contract.htm l