HelloWorld Transaction Processor

HelloWorld is a simple blockchain application written in JavaScript with Hyperledger Sawtooth. It provides the very basic functionality of writing a data to the blockchain and read that data from the blockchain.

The transaction processor defines the business logic for this application. It has two top-level components: Processor class and Handler class. The transaction processor receives the payload and:

- decodes it
- verifies if it is a valid action
- modifies state in a way that satisfies the action

The steps involved here are:

1. Create the handler class
2. Hash function
3. Constructor
4. Apply Method
5. Decode payload
6. Address generation
7. Write the function to store data to state
8. Create the processor class

Step 1 : Create the handler class

Create the handler class and it should extend the TransactionHandler class defined in the Sawtooth JavaScript SDK. So import the required class from sawtooth-sdk. To import the class we need to add a module name for the handler.

```
const { TransactionHandler } = require('sawtooth-sdk/processor/handler')
```

```
class HelloWorldHandler extends TransactionHandler {

}

module.exports = HelloWorldHandler;
```

Step 2 : Hash function

Define a function to find hash of a specific value. This hash will be used for addressing purposes by other functions. Here we use "crypto" module for hashing. SHA512 hash is created.

```
const crypto = require('crypto');
```

```
function hash(v) {
   return crypto.createHash('sha512').update(v).digest('hex');
}
```

Step 3 : Constructor

We should register our family name, family version and namespace(first 6 characters of hashed family_name) with the validator. This could be done in the constructor, where we can call the parent class constructor by passing family name, the version and namespace of the TP.

Information about Helloworld Transaction Family

FAMILY_NAME ='HelloWorld' ;

FAMILY_VERSION ='1.0';

NAMESPACE = First six characters of the hash of FAMILY_NAME

```
const FAMILY_NAME='HelloWorld';
const NAMESPACE = hash(FAMILY_NAME).substring(0, 6);
```

//Write the following code within **HelloWorldHandler** class

```
constructor(){
      super(FAMILY_NAME, ['1.0'], [NAMESPACE]);
   }
```

Step 4 : Apply Method

Apply is the single method that defines all the business logic for a transaction family. Apply method is called for each and every transaction. Apply method has two arguments, *transactionProcessRequest* and *context*. The transactionProcessRequest holds the data of valid transactions sent by the client and context stores information about the current state.

```
apply(transactionProcessRequest, context){

}
```

The transactionProcessRequest object is the transaction sent from client, which contains header, headerSignature and payload. The context object contains methods for accessing and modifying blockchain state.

Step 5 : Deserialize payload

Client creates the payload in a serialized format. So Transaction Processor will need to deserialize the payload to get the message, which should be added to state. Import the library for encoding and use decode method to decode the payload.

```
const {TextEncoder, TextDecoder} = require('text-encoding/lib/encoding')

var encoder = new TextEncoder('utf8')
var decoder = new TextDecoder('utf8')
```

//Write the following code within **apply** function

```
var msg = decoder.decode(transactionProcessRequest.payload);
```

Step 6 : Address generation

A state address in Sawtooth is 35 bytes long, typically expressed as 70 hexadecimal characters. By convention, the first six characters are reserved for a namespace for the transaction family and remaining 64 characters are up to each family to define.

Addressing used in HelloWorld application is as follows:

- The first 6 characters of the address are the first 6 characters of a sha512 hash of the FamilyName: “HelloWorld”
- The remaining 64 characters of the address are the first 64 characters of a sha512 hash of signerPublicKey present in the header of the transaction.

The *transactionProcessRequest* object contains *header*, *headerSignature* and *payload*. The *signerPublicKey* is obtained from the header.

//Write the following code within **apply** function

```
let header = transactionProcessRequest.header
this.publicKey = header.signerPublicKey
this.address = hash(FAMILY_NAME).substr(0, 6) +
hash(this.publicKey).substr(0,64);
```

Step 7 : Write the function to store data to state

Now write a function to store data to the state. The context (second parameter in the apply method) is the sawtooth data layer object which is used for setting a state value and also to retrieve any state value. The data is to be stored in the address mentioned in the previous step. The decoded payload will contain the message to be stored in state. So we create a function with these three parameters - context, address and message. The message is stored in encoded format, so we use encode function to encode the message. Create entries, where addresses are the keys and data is the value, to store in state. Finally call the setState of context with entries as parameter.

```
function writeToStore(context, address, msg){
   let msgBytes = encoder.encode(msg);
   let entries = {
      [address]: msgBytes
    }
   return context.setState(entries);
}
```

//Write the following code within **apply** function

```
return writeToStore(context, this.address, msg);
```

Step 8 : Create the processor class

The processor class routes transaction processing requests to a registered handler class. The processsor class creates the process that runs the Transaction processor.

Import the *TransactionProcessor* function from the library and create a new object by connecting to a specific validator (tcp://localhost:4004)

After that we have to add a handler to the transaction processor with addHandler method of the transaction processor object, here the handler is imported from the HelloWorldHandler.js file.

Finally start the transaction processor with start() method. Running the HelloWorldProcessor.js file will start the TP and keep listening at TCP 4004 port by default.

//Add the following code within **HelloWorldProcessor.js** file

```
const { TransactionProcessor } = require('sawtooth-sdk/processor');
const HelloWorldHandler = require('./HelloWorldHandler');

const address = 'tcp://validator:4004';
const transactionProcesssor = new TransactionProcessor(address);
transactionProcesssor.addHandler(new HelloWorldHandler());
transactionProcesssor.start();
```

Additional Lab Exercises

1. **Append the existing state store data with data received from client**

   Sample transaction processor application already covered overwrites state data(Eg: Name) with data received from client. Instead of overwriting existing state data, transaction processor should append existing state data with new data received from client

   Hint: to get the data from a state address in TP

```
context.getState([Address]).then(function(stateKeyValueAddress){
console.log("State Address value",JSON.stringify(stateKeyValueAddress));
var data = 0;
data = stateKeyValueAddress[Address];
```

```
    }
```

2. **Check duplicate data exists in state store**

   Modify the additional exercises 1 with following functionality. If new data is received from client ,transaction processor should check whether it is duplicate data or not. If duplicate data found, then return an error code otherwise append the data.