National Research University Higher School of Economics

Faculty of Computer Science

HSE and University of London Double Degree Programme in Data Science and Business Analytics

Review of Programming Project (Research project)

Completed by a Student in Year 02, group 193 of the degree programme "Data Science and Business Analytics" at the HSE University Faculty of Computer Science

Baminiwatte Arachchiralalage Ranga Nayanakantha,

Full name

on the topic:

Corporate Settlements mechanism based on Smart Contracts

No.	**Evaluation Criteria (only assess those applicable to student's work)**	**Score (on a 10-point scale)**
1.	Preciseness and accuracy in formulating of the project's aims and goals	10
2.	Full use of information sources (books, articles, HSE University electronic resources, web-sources, etc.)	10
3.	Complexity and volume of the completed work	10
4.	Complexity and/or capacity of software realisation / offered technological decision / investigation of research problem	10
5.	Accomplishment of the project's aims and goals	10
6.	Quality of the composed text	10

Plagiarism evaluation from the Antiplagiat System (cannot be higher than 20%), admission to Project Defence – 5%

General commentary on the work and scores (filled by Supervisor)

During the course of the project, Ranga Baminiwatte demonstrated a good general methodological background and specialized skills in software development. The project tasks are fully completed - the test blockchain network demonstrates the possibility of integrating the deal management process with the implementation of virtual payments. Positions that require further detailed study with the bank's business units have been identified.The software part of the project (code development) was performed directly in the code base of the current R-Chain adapter. The Account smart-contract has been developed and the Deal smart-contract has been emproved. According to the presented templates, the functions for processing blockchain platform events (Accept Payment, PaymentExecuted, SetBalance) and performing operations (AddAccount, UpdateBalance, AddPayments, ApprovePayment) were developed. When coding, the recommended architecture of the program code, standard operation templates and design rules were fully observed.The project materials are protected by the NDA.

Score for the quality of code development - 10

FINAL SCORE OF THE SUPERVISOR excellent/well/acceptable/unacceptable. **(10_ out of 10)**

Supervisor/Reviewer Pavel D. Bolotov, Senior SwA, Raiffeisen Bank

Position, academic degree, department/place of work Full name Signature

Болотов 28.05.2021

NATIONAL RESEARCH UNIVERSITY
HIGHER SCHOOL OF ECONOMICS

Faculty of Computer Science
Bachelor's Programme 'HSE University and University of London Double Degree Programme in Data Science and Business Analytics'

UDC _____

Research Project Report (Final)

on the topic Corporate settlements mechanism based on smart contracts.

Fulfilled by the Student:

group #БПАД 193 _____ Baminiwatte A.R.N

Signature — Surname, First name, Patronymic, if any

28.05.2021

Date

Checked by the Project Supervisor:

Pavel Bolotov P.

Surname, First name, Patronymic (if any), Academic title (if any)

Senior SwA, Digital Innovations Department

Job

Raiffeisen Bank Russia

Place of Work (Company or HSE Department)

Date 29.05 2021 — 10 — _____

Grade according to 10-point scale — Signature

Moscow 2021

Contents: **Pg. num**

1. Introduction

1.1 Abstract

Blockchain technology is generally acknowledged the future of financial infrastructure and banking systems. As most leaders of Banking world are now customizing their applications to perform particular tasks through Blockchain based smart contracts- a self-executing program (a code) which runs on a Blockchain network which would automatically implement the terms of agreement or a deal between parties. This is a critical step forward, streamlining networks that are spread across multiple ERP systems and DLT databases.

Virtual cooperate settlements is a unique type of smart contract which is used to plausibly make and trading settlements faster and more flexible. It could transcend existing intermediaries to speed up a settlement process. This technology opens a window for time sensitive investors to complete settlements earlier by paying additional transaction fees known as gas.

Usage of blockchain technology can lead to a rapid and a much flexible settlement process. The major challenge of settlements based on blockchains is how to mitigate the risk of settlement. The system has to allocate proper block sizes and block runtimes to make sure that “forking” (a system attack that undoes a true transaction record) is economically infeasible.

As world is undergoing the Covid-19 pandemic currently, with all the health measures and social distancing protocols, there is rising concerns of the public that cash may transmit the Covid-19 virus and new government to public payment schemes have further speed up the shift towards digital currencies. Hence, the relevance of this project to contemporary industries is extremely high.

Main objective of this project is to develop and implement a cooperate settlement mechanism for a group of companies based on R-Chain abstract business objects.

1.2 Terminology and definitions.

Since Blockchain technology and related services are still quite unknown to the most it is important to specify special keywords used in the subject area.

Blockchain – Blockchain technology is a distributed virtual ledger that records the origin of a digital asset. It follows a decentralized architecture which makes records in it hard to change.

Blocks- A block is a permanent container of logs and records.

Smart Contract - A collection of agreements that exist in a digital format, including the various agreements followed by the parties in performing these agreements.

Mining – The process of securing and verifying bitcoin transactions.

Genesis block- First block(starting) of a Blockchain.

Gas- The amount of currency required to do a transaction in the network.

Bitcoin – The most popular digital cryptocurrency in the world introduced in January 2009.

Ether – The type of cryptocurrency used in Ethereum platforms.

R- Chain -A novel blockchain platform based on the formal framework of decentralized computation which leverages software development through producing massive scale concurrent blockchains.

Geth - Go-Ethereum is command-line interface (CLI) tool that interact with the Ethereum network and links your device and the rest of the Ethereum nodes.

Private Ethereum network – A separated Blockchain that is remotely functioning from the Main Ethereum network. These Private Networks are mainly developed and used by organizations to regulate the reading accessibility of the Blockchain.

D-Apps - A computer application that runs on a distributed computing system.

Ethereum wallet - API or mobile applications that allow you to interact with Ethereum accounts.

Solidity - Solidity is an OOP(object-oriented programming language) mainly used in creating smart contracts. It is used in implementing Ethereum based blockchain platforms as the main developing tool.

Truffle - Truffle is an all-in-one development platform used to test frameworks and pipeline assets. It is based on Ethereum Blockchain and is used to develop flawless and efficiently functioning D-Apps.

Proof-of-work or Proof-of-stake - A type of consensus protocol which use a certain number of computational resources to prove other parties' work, or we can think it as an arbitrary

mathematical puzzle.

Ethereum – An open-source blockchain which uses a decentralized method of software architecture and uses the cryptocurrency unit ether.

DLT - Distributed Ledger Technologies are P2P networks that allow several participants to constantly update and share their ledgers. DLTs provide their members possibilities to securely deal with their transactions.

ERP - Enterprise resource planning

Quorum - Quorum Blockchain was built by JPMorgan using Ethereum network. The absence of gas cost is an important feature of Quorum.

Swarm – A storage platform which allows participants to efficiently combine their storage capacity and network bandwidths in order to provide a base layer native service of the Ethereum web 3 stack.

1.3 Project team, roles, and supervisor:

The project is being done by myself and Jintao Xu a 2nd year student from Business and Data Analytics program at HSE. Although there are no exact roles allocated, Jingtao is focusing on development of the programming part whereas I am focusing on analyzing, maintaining database and testing aspect of the project. However, during the project there were instances where the roles have been swapped.

We are supervised by Mr. Paul Bolotov, senior software architect at IT department Raiffeisen Bank Russia.

2. Project Review and Requirements

2.1 Project description and overview

Before we started the project our supervisor from Raiffeisen bank was already working on "abstract deals" concept and mechanism. Having analyzed the possibilities and consequences of a R-Chain blockchain platform approach in previous projects they had done in 2016-17, they had come to the conclusion that it is necessary to introduce an adapter of unified business objects – another intermediate between the business-backend and "technical components". This technique refers to standard program architecture templates, which still does not eliminate its effectiveness. As a result, they got the following program architecture of the Node of the Platform (Shown in figure 1). According to the proposed architecture, Business app works not directly with abstractions of DLT-components, but with a certain "universal business process" (further – Process) – creates a Process, changes Process state. "Business process mirror" is a set of components, which converts universal Process data structure and operations into the data and operations, used in a DLT-client. This "Mirror" also conducts a reverse conversion of the information, received from DLT-client, and filters Processes information, not related to the Node owner Member. To recapitulate, in every Node there is maintained a synchronous Database of relevance to this Node business processes. Furthermore, the format of the database is the most convenient to the Business app. Business app interacts with this Business processes Database, receiving information about Processes state and sending operations on the state changes.

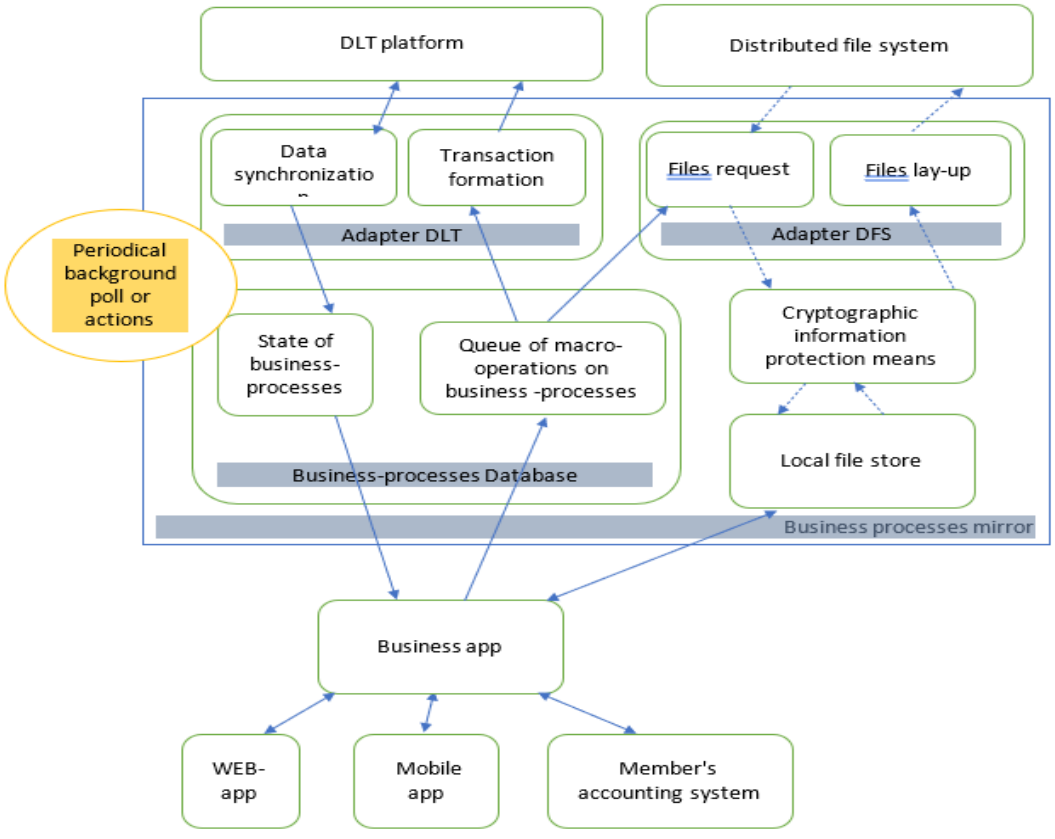


Figure 1:Components of a common R-chain platform.

In related to this specific project of virtual settlements he first task given to us was to establish private blockchain node in personal computers in order to get familiar with the attributes of a decentralized network, and do simple such creating accounts, making simple transactions, and writing basic smart contracts etc.

In the second half of project, we were to improve the existing mechanism on R-chain platform and modify the smart contract and adapter program code in terms of settlements. This was core part of the project which included smart contract writing database management and adapter development.

Another task which we undertook is to design and analysis of different options of the settlement mechanism. Mainly we focus on two approaches. In first method we develop a mechanism which Virtual contracts executed through central account smart contract. In second, we try to create a dedicated node or an oracle node which would execute smart contract deals by itself.

The official project objectives and requirements given are mentioned below.

The developed mechanism should suffice the following functional and non-functional requirements:

2.1 Functional requirements and Non- functional requirements:

Functional requirements:

- To establish settlements within the blockchain network using virtual accounts and deals.
- Settlements must be confirmed by the payer
- Settlements are made when a certain deal status occurs

Non- functional requirements:

- Performance: up to 100 deals by 1 hour, up to 500 status changes by 1 hour, up to 1000 executed payments by 1 hour.
- Number of participants for 1 deal - up to 6
- Each participant should have an opportunity to work with multiple accounts.

2.3 Project Toolkit

- **C++**: A powerful object-oriented programming language which was used to develop the adapter DLT.
- **Solidity**: Another object-oriented programming language which was used to write the required smart contracts
- **Citrix Workspace**: A remote desktop application, which was mainly used to access Raiffeisen bank servers remotely. All the code development and testing was done in that environment.
- **PgAdmin4:** An open-source GUI database management tool, which is mainly used maintain databases and used as an input API of our mechanism.
- **Putty:** open-source terminal emulator used to access UNIX or LINUX system remotely
- **Ethereum_RPC:** An API dedicated to interact with blockchain network, A comprehensive explanation of its usage is provided below.
- **GitHub**: Used to upload codes and maintain repositories.

2.4 Theoretical part and algorithms:

There are no specific theories related to this project as this is mostly an applied procedure but theory of Blockchain is quite important to understand.
Blockchain technology is peer-to-peer network which uses a DLT (Distributed Ledger Technology). A peer-to-peer network is a system of computing devices connected with each other without centralized server following a distribution application architecture. A DLT is a technology that keeps a ledger of all records of input and output process such that any record could be accessed immediately. A basic DLT would consist of 3 components; **Protocol** (Used in building consensus-an agreement which all users of the blockchain), **Language of transactions** (changes the state of ledger) and **Data Model** (current state of ledger). In this technology consensus are quite important because it's the most complicated part(theoretically) in the blockchain technology. Since we consider about Ethereum in this project, Ethereum uses a mechanism called proof of stake algorithm to validate transactions.
Proof of stake -

Figure 2:proof of stake (source: https://www.ledger.com/academy/blockchain/what-is-proof-of-stake)

A blockchain is a chain of blocks interconnected with each other. A single block would contain, A time stamp, a nonce (a unique number that can be only used once), Merkel tree Root, and cryptographic hash of the previous block. In the proof of work/stake miners would calculate the nonce. The blocks are immutable due to its cryptographic nature. But it doesn't imply that it's not possible to change its structure. It's extremely difficult to change its data and any change is easily detectable making blockchain system a very secure mean of information containers. A Merkle tree is binary tree consisting of hash pointers which would allow efficient and secure verification of content in huge data collection. The advantages of using Merkle trees include **O(logn)** steps of proving a block is a member.

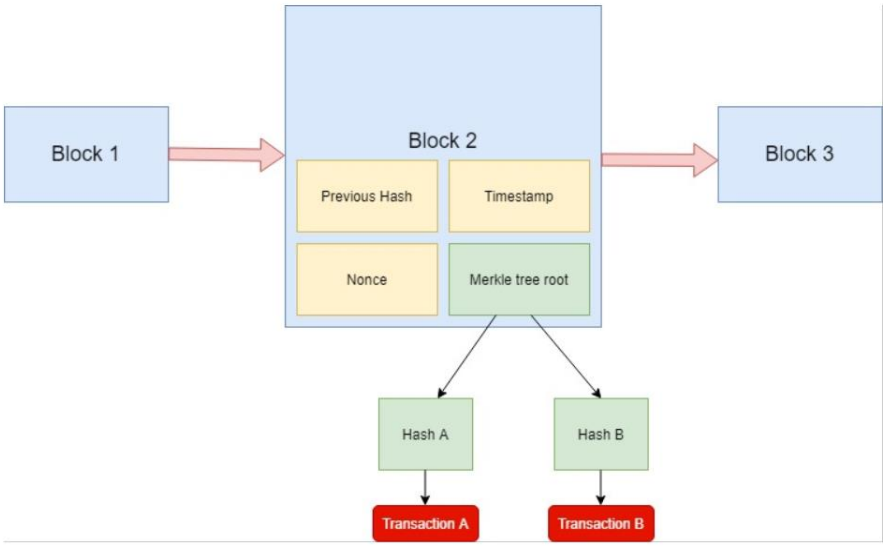


Figure 3:A block in a blockchain

Quorum – It is important to consider about Quorum blockchain protocol as we mainly use it as the blockchain protocol in this project. Quorum is a special private blockchain network designed to conduct more closed transactions, in which there is either a single member owning all nodes, or a consortium blockchain network, in which multiple members each own a part of the network. Quorum is developed from Ethereum by modifying the Geth clients. Advantages of Quorum includes,

- Possibility of private transactions: members of a Quorum network can send private transactions that are addressed to a subset of nodes, so that the contents of the transaction are not exposed to unauthorized members.
- Another important advantage of Quorum is flexible consensus: As, Quorum supports valid consensus options Raft and Istanbul BFT. Both support transaction finality (which means chain forking is lacking) and offer shorter block intervals than proof-of-work.
- A Quorum network can be run in a permission configuration mode such that all nodes must be listed explicitly in an access control list enforced by all nodes preventing foreign

nodes from tampering into the network and duplicating blocks like in permissionless networks.

3. Project Implementation

In this section we'll be looking at all aspects of implementing the project.

3.1 Development of Smart Contracts (SC)

Smart contracts are like the blueprints for the whole mechanism. We mainly developed 2 smart contracts. **Account.sol** and **Deal_free.sol**. (Full codes can be found in the following GitHub repository (https://github.com/JingtaoXu/VirtualSettlement)
Account.sol is dedicated to perform all attributes of an actual account. Deal_free.sol is related to payments execution. A comprehensive descriptions of all operations of both smart contracts are provided below.
Their basic structures were based on flow diagrams provided to us by our supervisor, are shown below.

Figure 3 shows the relationship between both deal and account smart contracts. Account smart contract could be dedicated either to Payer or the beneficiary. When a participant node in blockchain sends a list of expected payments into Deal SC it will redirected to payers Account SC which would emit a payment confirmation request to payer's node in the blockchain. The payment is finally completed when it goes through a two-step verification process, that is to confirming payment from both payer's node and Payers Account SC.

Another feature in this mechanism is sending a set of payment triggering statue, which is stored in map-ping data structure (payment states maps to payment address). This method would avoid checking state of each payment individually. This method would reduce the time complexity of payment execution greatly.

Figure 4 illustrates the workflow of Deal SC.

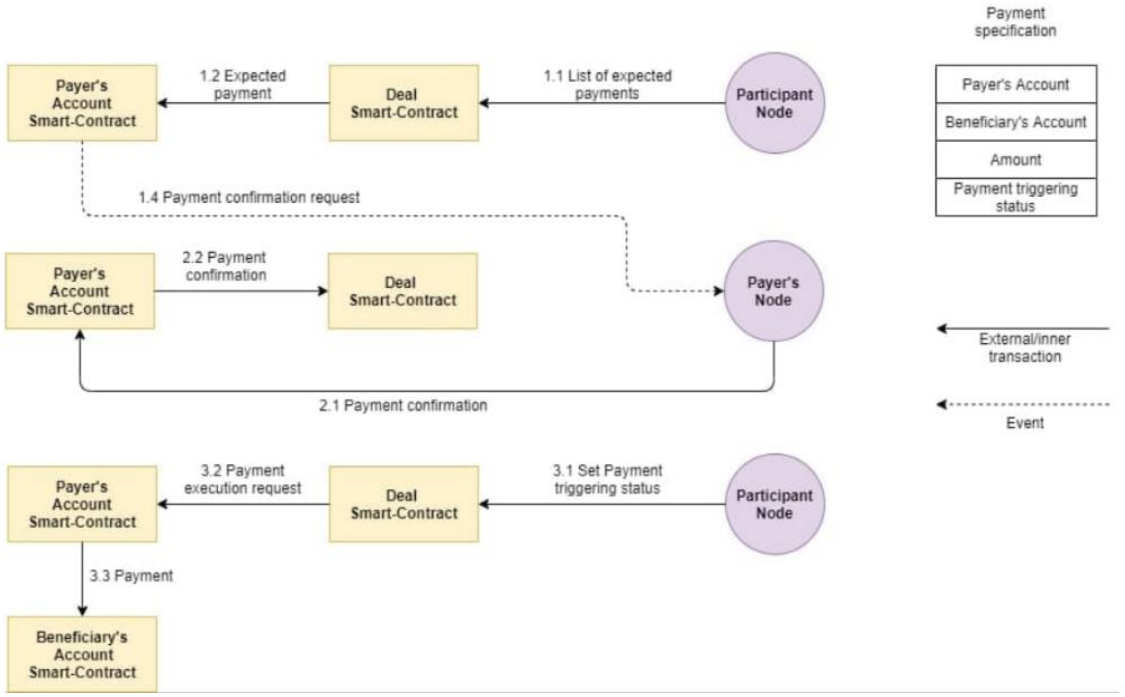


Figure 4:structure of Virtual settlement mechanism

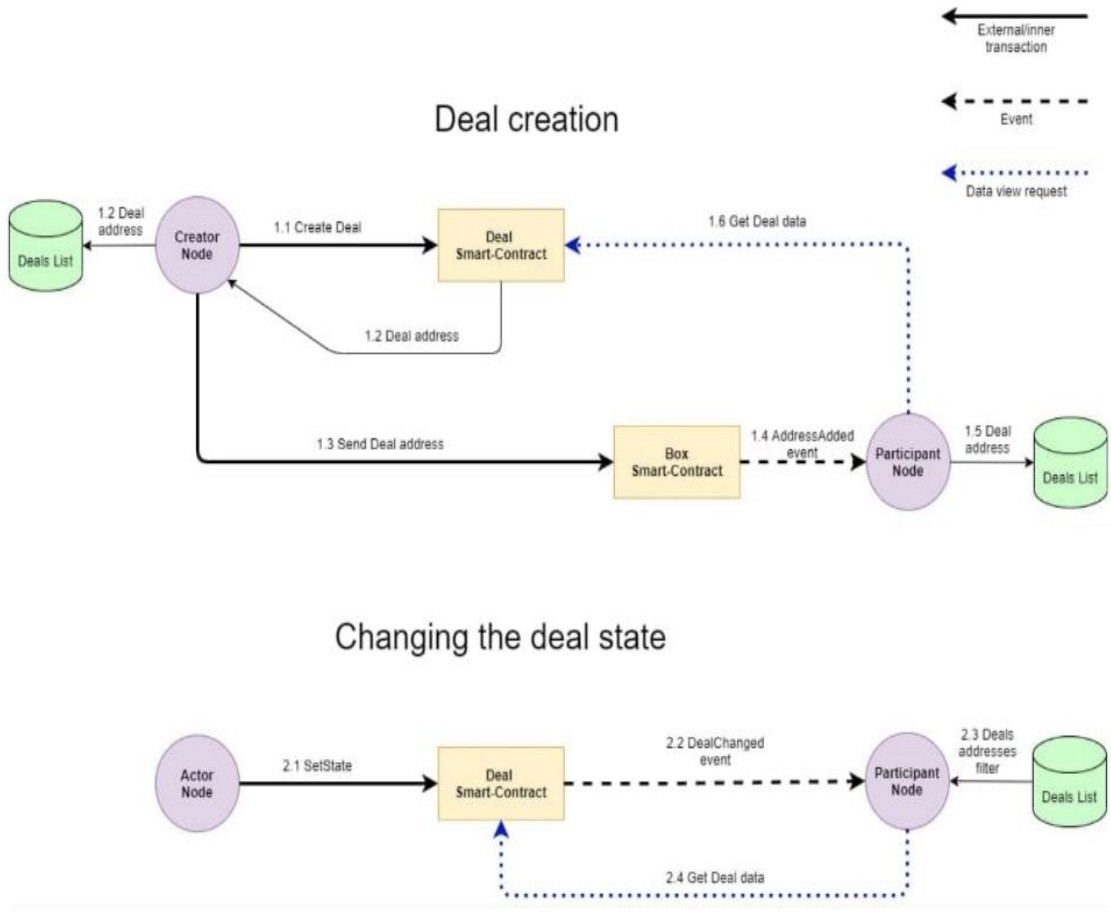


Figure 5: Deal.sol flow chart

Functions and operations of Account.sol and Deal.sol

Table 1: Account.sol

Function	Description
PutPayment()	It forms a payment and emits AcceptPayment event
PaymentAccept()	It is called for setting status = 'Y' (ONLY by owner)
GetPayment()	It gives full information of payment by passing ID and ADDRESS.
GetBalance()	It returns the current balance of the caller.
ExecutePayment()	After checking all necessary conditions, it does the transaction and emits SetBalance and PaymentExecuted
Income()	It increases balance by the given AMOUNT

Table 2: Deal_Free.sol

Function	Description
SetStatus()	It's used to set the status of deal(New__, Foamed__, Accepted__, Rejected__,Paid)
PutPayment()	It inputs a single payemnt and push into a new array PaymentsTriggers if state of payment is execute_state_.
LoadPayments()	It's used to iterate through an array of Payemnts and Calls PutPayment function on each array element.
ConfirmPayment()	It's used to set Payments[_id].confirmed="Y"

3.2 R-chain Adapter

Another important part of the mechanism is the R-chain Adapter, which acts as intermediate between the smart-contracts and data base. Basically, adapter is the backend program which contains functions and operations which corresponds to functions of our smart contracts and interpret them to data base and do vice versa. Whenever data is modified in the data base, R-chain adapter would respond to that and update the change with a new block in the blockchain with new time stamp. The entire code in the adapter was not developed by us as it was already a work in progress. As part of our project tasks, we were assigned to design the following functions.

int **EMIR_dl_action_AddDeal** (Dl_action *action, SQL_link *db, char *error)
int **EMIR_dl_action_SetStatus** (Dl_action *action, SQL_link *db, char *error)

int **EMIR_dl_action_AddPayments** (Dl_action *action, SQL_link *db, char *error)
int **EMIR_dl_action_AddAccount** (Dl_action *action, SQL_link *db, char *error)
int **EMIR_dl_action_UpdateBalance** (Dl_action *action, SQL_link *db, char *error)
int **EMIR_dl_action_GetPayment** (Dl_action *action, SQL_link *db, char *error)
long long **EMIRi_dl_GetBalance** (char *contract, char *error)
int **EMIRi_dl_GetDealId** (Dl_action *action)

All the functions we developed correspond to a function in smart contracts, in which **EMIR_dl_action** prefix is added to indicate the adapter function. While the entire code itself is provided in the resources section the following code shows block by block breakdown of the function **EMIR_dl_action_AddAccount**(Dl_action *action, SQL_link *db, char *error)**.**

```
/*---------------------------------------------------- initialisation of SQL pointer */

                          *error=0 ;

     if(code==NULL)  code=(char *)calloc(1, _CODE_SIZE) ;

        status=EMIR_db_nodepars(db, error) ;
     if(status)  return(-1) ;



 /*-------------------------------------------- Verifying account & password */

                memset(account, 0, sizeof(account)) ;

       if(action->executor[0]!=0)
               strncpy(account,  action->executor, sizeof(account)-1) ;
       else    strncpy(account, __member_executor, sizeof(account)-1) ;

               password=strchr(account, ':') ;
            if(password==NULL) {
/                                    password="" ;
                                }
            else                {
                                    *password=0 ;
                                     password++ ;
                                }


        Cursor=db->LockCursor("EMIR_sc_action_AddAccount") ;
     if(Cursor==NULL) {
                           strcpy(error, db->error_text) ;
                                       __db_errors_cnt++ ;
                              return(-1) ;
                    }
```

```
  /*- - - - - - - - - - - - - - - - - - - - - - - - -  Check Account existance */
                      sprintf(text, "select \"Id\" "
                                    "from   %s "
                                    "where  \"Id\"='%s'",
                                     __db_table_deals_accounts, action->data) ;
         status=db->SelectOpen(Cursor, text, NULL, 0) ;
       if(status) {
                    strcpy(error, db->error_text) ;
                                  db->error_text[0]=0 ;
                                __db_errors_cnt++ ;
                                       break ;
                  }
         status=db->SelectFetch(Cursor) ;
  //   if(status==_SQL_NO_DATA)   break ;
       if(status) {
         if(status==_SQL_NO_DATA)  sprintf(error, "Get data: No record") ;
         else                      sprintf(error, "Get data: %s", db->error_text) ;
                                       db->error_text[0]=0 ;
                                     __db_errors_cnt++ ;
                    break ;
                  }

             strncpy(contract, (char *)Cursor->columns[0].value, sizeof(contract)-1) ;

                   db->SelectClose(Cursor) ;

  /*- - - - - - - - - - - - - - - - - -  Get txn id from SystemConfiguration */
            status=EMIR_db_syspar(db, "Account Template", txn, error) ;

         if(status)  break ;



/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - Sending transaction to the blockchain */
       do {
              status=EMIR_node_getcode(txn, code, _CODE_SIZE-1, error) ;  /*  retrieving code of contract */

           if(status)  break ;

              status=EMIR_node_checkgas(account,
                                        NULL, code, gas, error) ;
           if(status)  break ;

              status=EMIR_node_unlockaccount(account,
                                             password, reply, sizeof(reply)-1, error) ;
           if(status)  break ;

              status=EMIR_node_publcontract(account,               /* send transaction */
                                             code, gas, txn, error) ;
           if(status)  break ;

         } while(0) ;



/*- - - - - - - - - - - - - - - - - - - - - - -  Save contract address */
                  sprintf(text, "update %s "
                                "set    \"BlockChainId\"='%s'"
                                "where  \"Id\"=%s",
                                     __db_table_deals_accounts, contract, action->data) ;
       status=db->SqlExecute(Cursor, text, NULL, 0) ;
     if(status) {
                   strcpy(error, db->error_text) ;
                                 db->error_text[0]=0 ;
                               __db_errors_cnt++ ;
                            break ;
              }

         } while(0) ;
```

```
/*- - - - - - - - - - - - - - - - - - - - - - - -  Entering results to the DataBase */
               db->SelectClose(Cursor) ;

     if(status==0)  sprintf(text, "update %s "
                              |    "set    \"Status\"='WAIT', \"Reply\"='%s'"
                                   "where  \"Id\"=%s",
                                        __db_table_deals_actions, txn, action->id) ;
     else           sprintf(text, "update %s "
                                   "set    \"Error\"='%s'"
                                   "where  \"Id\"=%s",
                                        __db_table_deals_actions, error, action->id) ;
        status=db->SqlExecute(Cursor, text, NULL, 0) ;
     if(status) {
                     strcpy(error, db->error_text) ;
                               __db_errors_cnt++ ;
                          return(-1) ;
              }

                  db->Commit() ;
                  db->UnlockCursor(Cursor) ;
       }

/*---------------------------------------- waiting for confirmation */

   if(!stricmp(action->status, "WAIT"))  {

      do {

        Cursor=db->LockCursor("EMIR_sc_action_AddAccount") ;
     if(Cursor==NULL) {
                           strcpy(error, db->error_text) ;
                                     __db_errors_cnt++ ;
                              return(-1) ;
                      }


  /*- - - - - - - - - - - - - - - - - - - - - - -  receiving the receipt */
              status=EMIR_node_checktxn(action->reply, block, contract, error) ;
            if(status== 0)  return(0) ;
            if(status < 0)  break ;

                 strcpy(contract, contract+2) ;     |

              status=EMIR_node_identify(contract, reply, error) ;
            if(status)  break ;

            if(*reply==0) {
                                  strcpy(error, "Empty identify reply") ;
       /*- - - - - - - - - - - - - - - - - - - - - - -  send results to the DataBase */
            if(status==0)  sprintf(text, "update %s "
                                         "set    \"Status\"='DONE', \"Reply\"='%s'"
                                         "where  \"Id\"=%s",
                                              __db_table_deals_actions, contract, action->id) ;
            else           sprintf(text, "update %s "
                                         "set    \"Error\"='%s'"
                                         "where  \"Id\"=%s",
                                              __db_table_deals_actions, error, action->id) ;
              status=db->SqlExecute(Cursor, text, NULL, 0) ;
            if(status) {
                           strcpy(error, db->error_text) ;
                                     __db_errors_cnt++ ;
                                return(-1) ;
                      }

                          db->Commit() ;
                          db->UnlockCursor(Cursor) ;
       /*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*/
```

Here is how the code works:

In the database:

1. Once the Account smart-contract is deployed to Ethereum and txn id(transaction id) is received.
2. Add parameter 'Account Template' with txn id to SystemConfiguration
3. Replace "action->data" by value linked to keyword 'Account Template'

In the adapter:

1.SQL pointer to DB is initialized.
2.Then account and password are verified and existence of account in the database table deals_accounts is checked.
3. Then txn id is extracted from systemconfiguration.
4. After that corresponded transaction is sent to the blockchain, and code of contract is retrieved and saved.
5. The resulted code is entered into database.
6. The adapter also waits for confirmation receipt and that result is also added to data base.

3.3 Database Creation and Management.

Database is the third important component in our mechanism which acts as the API of data entry and data retrieval. SQL scripts are written to interact between Database and R-chain adapter. Raiffeisen bank uses the database management application PGadmin4 to maintain all their databases related to this project. The purpose of these tables is to keep track of accounts, deals and transactions in the system, with their credentials. The database would have several mirrors which are the participants of the process. These mirrors help us to do transactions between parties easily. For example, mirror 1 is the Bank itself and mirror 2 might be different company which is a client of the bank.

PGadmin4 program provides a user-friendly interface to draw up queries and create tables. (figure 5)
The basic structure of a table and a part of SQL query code are shown below.

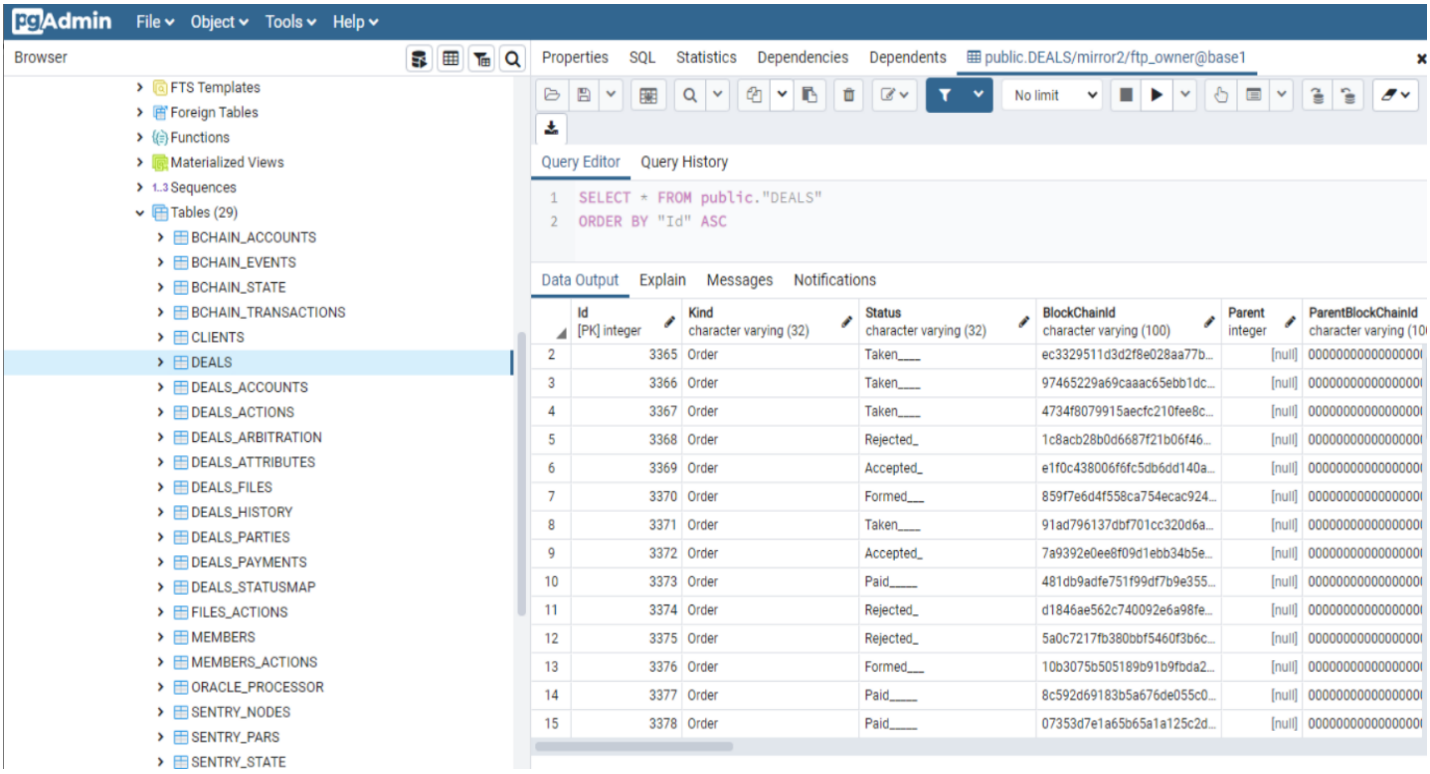


Figure 6:Interface of PGadmin4 and Deals table which contains records about deals in mirror 2.

Two of the main tables we had create were Deals_Payments(figure 6) and Deals_Accounts(figure 7) table. Their SQL scripts are shown below.

```
-- SEQUENCE: public."DEALS_PAYMENTS_Id_seq"

-- DROP SEQUENCE public."DEALS_PAYMENTS_Id_seq";

CREATE SEQUENCE public."DEALS_PAYMENTS_Id_seq"
    INCREMENT 1
    START 1
    MINVALUE 1
    MAXVALUE 9223372036854775807
    CACHE 1;

ALTER SEQUENCE public."DEALS_PAYMENTS_Id_seq"
    OWNER TO ftp_owner;

-- Table: public."DEALS_PAYMENTS"

-- DROP TABLE public."DEALS_PAYMENTS";

CREATE TABLE public."DEALS_PAYMENTS"
(
    "Id" bigint NOT NULL DEFAULT nextval('"DEALS_PAYMENTS_Id_seq"'::regclass),
    "DealId" bigint NOT NULL,
    "UID" character varying(32) COLLATE pg_catalog."default" NOT NULL,
    "Account_from" character varying(100) COLLATE pg_catalog."default" NOT NULL,
    "Account_to" character varying(100) COLLATE pg_catalog."default",
    "StateExecuted" character varying(2) COLLATE pg_catalog."default",
    "StatePlanned" character varying(2) COLLATE pg_catalog."default",
    "DataUpdate" character varying(2) COLLATE pg_catalog."default",
    "InsertTimestamp" timestamp without time zone,
    CONSTRAINT "DEALS_PAYMENTS_pkey" PRIMARY KEY ("Id")
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public."DEALS_PAYMENTS"
    OWNER to ftp_owner;
```

Figure 7: SQL script for Deals_Payments table.

```
-- SEQUENCE: public."DEALS_ACCOUNTS_Id_seq"

-- DROP SEQUENCE public."DEALS_ACCOUNTS_Id_seq";

CREATE SEQUENCE public."DEALS_ACCOUNTS_Id_seq"
    INCREMENT 1
    START 1
    MINVALUE 1
    MAXVALUE 2147483647
    CACHE 1;

ALTER SEQUENCE public."DEALS_ACCOUNTS_Id_seq"
    OWNER TO ftp_owner;

-- Table: public."DEALS_ACCOUNTS"

-- DROP TABLE public."DEALS_ACCOUNTS";

CREATE TABLE public."DEALS_ACCOUNTS"
(
    "Id" integer NOT NULL DEFAULT nextval('"DEALS_ACCOUNTS_Id_seq"'::regclass),
    "BlockChainId" character varying(100) COLLATE pg_catalog."default" NOT NULL,
    "BankAccount" character varying(100) COLLATE pg_catalog."default" NOT NULL,
    "BankBIC" character varying(20) COLLATE pg_catalog."default" NOT NULL,
    "Description" character varying(1000) COLLATE pg_catalog."default",
    "Balance" bigint,
    "Currency" character varying(20) COLLATE pg_catalog."default",
    CONSTRAINT "DEALS_ACCOUNTS_pkey" PRIMARY KEY ("Id")
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public."DEALS_ACCOUNTS"
    OWNER to ftp_owner;
```

Figure 8:SQL script for Deal_Accounts table.

4. Results and Conclusion

4.1 Testing Scenario

After developing R-chain adapter and Smart contracts, it was necessary test the complete process through a practical scenario.
Scenario 1- We consider 3 participants namely,

1. RBRU (bank)
2. STS (Company)
3. Dollarama(Store)

In the first stage, STS makes a payment of 500 to RBRU.
In the second stage RBRU makes a payment of 450 to DOLARAMA. Simultaneously STS must pay a bank free of 50 to RBRU.
Desired outcome – Balance of the bank - 100
Figure 9 shows two step process in the scenario.

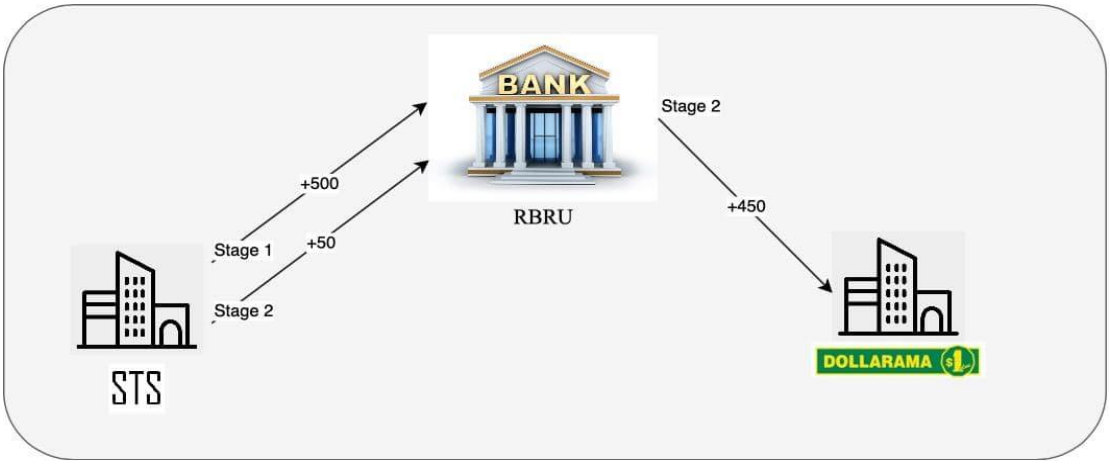


Figure 9:Testing scenario 1 diagram

Scenario 1 Test-
Here test the above scenario step by step.

The SQL code for whole testing scenario is provided in the following repository
https://github.com/JingtaoXu/VirtualSettlement/blob/main/test/Demo.sql

1. First, the smart contracts Account.sol and Deal_Free.sol are deployed and put into the blockchain network through Ethereum_RPC application of Raiffeisen bank. (figure 10)

2. Then we create account nodes for RBRU, STS, DOLARAMA.

3. Through the database we update balance in accounts of STS and DOLARAMA. We can add 1000 to both accounts to initialize.

4. Creating the deals of RBRU, STS and DOLARAMA on node STS.

5. Loading the following Payments to the deal:
   i. STS ⟶ RBRU +500 (status- “Stage 1”)
   ii. RBRU ⟶ DOLARAMA +450 (status- “Stage 1”)
   iii. STS⟶ RBRU +50 (status- “Stage 2”)
6. Setting state of deal to “Stage 1” from “New” in STS

7. Setting state of deal to “Stage 2” from “Stage 1” in DOLARAMA

8. Finally checking “Balance” field in the DEALS_ACCOUNTS table at each node to confirm the successful transaction. (figure 12)

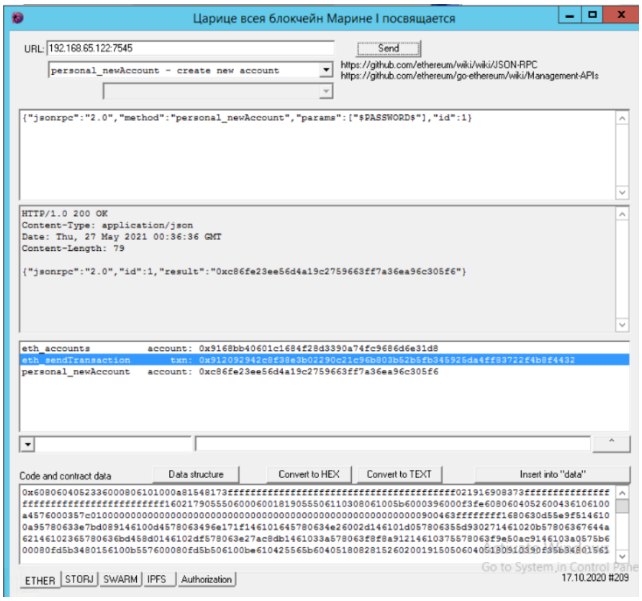


Figure 10:Ethreum_ RPC application. Deploying New account

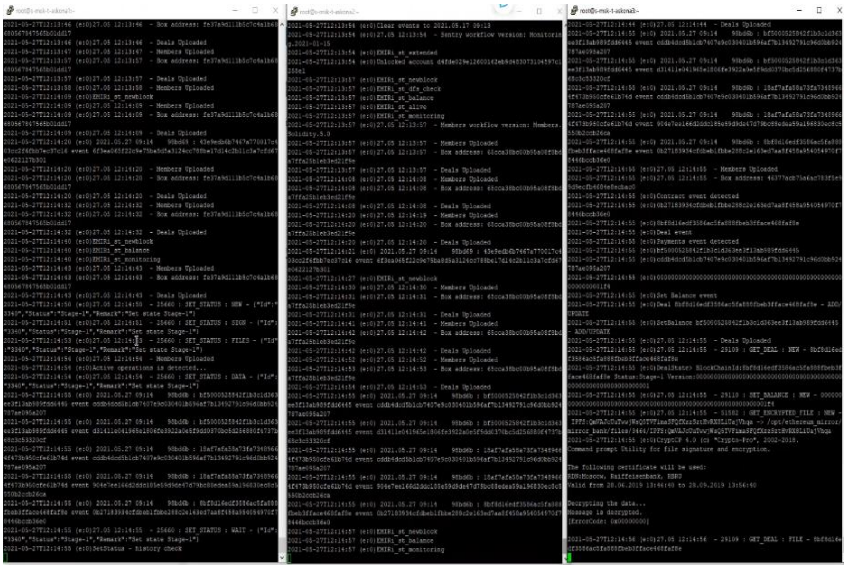
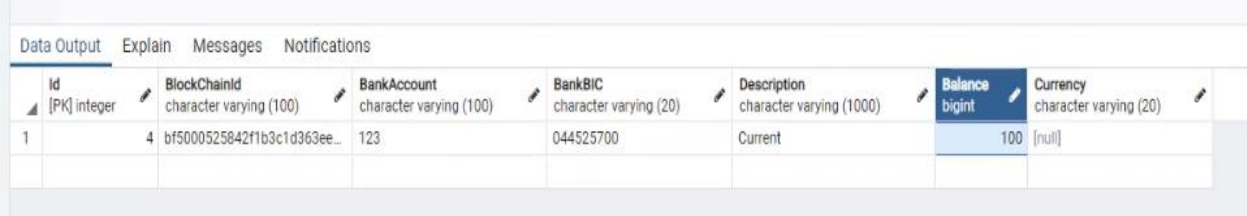


Figure 11: Blockchain running in the background for 3 nodes(participants).



	Id [PK] integer	BlockChainId character varying (100)	BankAccount character varying (100)	BankBIC character varying (20)	Description character varying (1000)	Balance bigint	Currency character varying (20)
1	4	bf5000525842f1b3c1d363ee...	123	044525700	Current	100	[null]

Figure 12:Bank nodes DEALS_ACCOUNTS table. The final balance should 100 as (500-450) +50 = 100. Hence the desired outcome

Scenario 2- In this case, we consider third parties apart from the 3 participants we considered earlier.

In the first stage, STS makes a payment of 500 to RBRU and DOLLARAMA makes a payment of 1100 to a third party.
In the second stage RBRU makes a payment of 450 to DOLARAMA. Simultaneously STS must pay a bank free of 50 to RBRU and third party to STS of 500.
Desired outcome – Balance of the bank – 100

Also in this case we test how accounts work when the balance is negative.

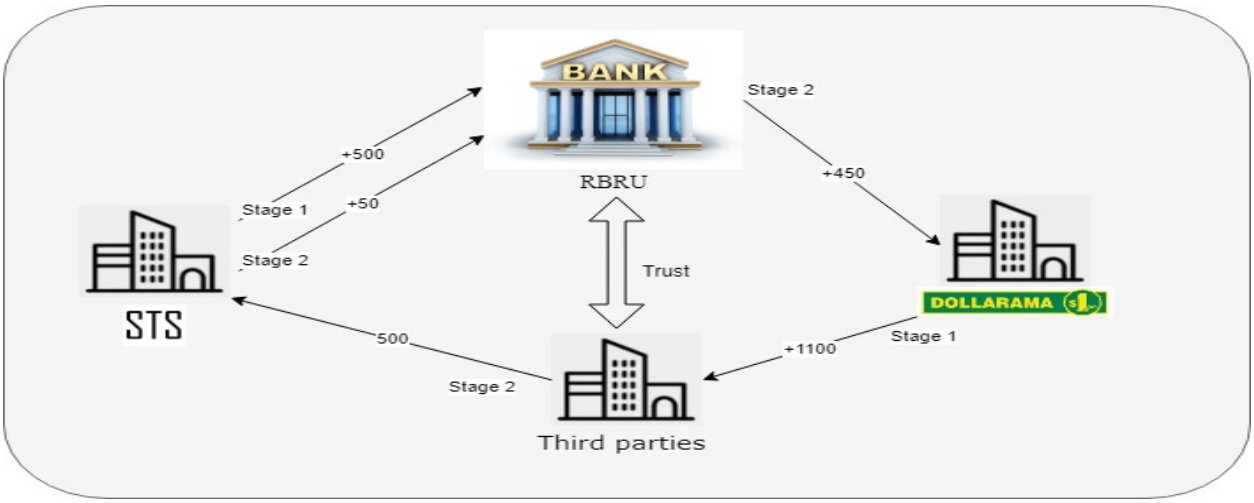


Figure 13:Testing scenario 2 diagram

The way scenario 2 test is conducted is similar to the test in scenario 1. The desired outcome is achived.

The only difference is after completion of "Stage 1" DOLLARAMA account would show a negative balance. In such cases a method called periodic clearing is used. That is, for the time bieng, payments are settled mutually and during such a settlement process, a negative account balances are allowed. It's important that the third party being member of the network so that bank could trust the third party despite the sender having a negative balance.

For classical cryptocurrencies such negative balances are considered as a bad practice, but for a virtual payment system like this, it's acceptable.

4.2 Conclusion:

After the successful test scenario of reciving desired results that are clearly up to expectation, we came up with the following conclusions.

Also a result of the project, in practice, the possibility of automatic execution of payments at the level of smart contracts and virtual accounts was confirmed when various deals conditions occurred.

At the same time, the points of the payment process that require more careful study from the point of view of the business process are identified:

- Control of the initial accrual of funds to the virtual accounts of participants;
- Handling situations where the payment has not been confirmed for execution;
- The ability to identify other participant's accounts not through a blockchain identifier, but through semantics notation (for example, by account name).
- The non-functional requirments of <<up to 100 deals by 1 hour, up to 500 status changes by 1 hour, up to 1000 executed payments by 1 hour>> and <<Up to 6 participants working with each other simultaneously>> are also met simply because the newly introduced characteristics to already established mechanism does not indicate a significant delay in process.

In general we could draw up conclusions about the whole D-app mechanism such as:

- The implementation smart contracts Account.sol and Deal.sol are essential for virtual settlement mechanism. Therefore, the developers must have a clear understanding about how smart-contracts work and how Solidity language .
- R-chain adapater can be optimized without making much alteration of the code for different smart contracts therefore adapter is a fixed componenet in the mechanism and it can be further improved to perform tasks such as supporting multiple smart contracts executed by different participants.It is alos possible to develop operations such as cancel payment, Delete Account etc.
- One of the disadavatges of decentralised applications and blockchains is higher possibilty of data loss and being prone to external attacks. However with private blockchain network like Quorum this risk could be considerably mitigated.
- D-app and blockchain uses are still a subject largely in reasearch stage with it’s true potential yet to be identified.

5. Links to reference materials

1. [1] https://github.com/project-ubin - Ubin project code.
2. [2] https://www.amazon.com/Building-Blockchain-Projects-decentralized-applications/dp/178712214X - Nrayin Prusty Building Blockchain Projects.
3. [3]https://github.com/JingtaoXu/VirtualSettlement - Entire code developed so far.
4. [4]https://remix-ide.readthedocs.io/en/latest/ - remix documentation
5. [5]https://docs.soliditylang.org/en/v0.7.4/ - solidity documentation
6. [6]PgAdmin documentation and user guide- https://www.pgadmin.org/docs/
7. [7]MySQL documentation- https://dev.mysql.com/doc/