주간 연구보고서

작성자	김한호	작성일자	2020.04.26.
연구기간	2020.04.20 ~ 2020.04.26		

1. 주간 연구 요약

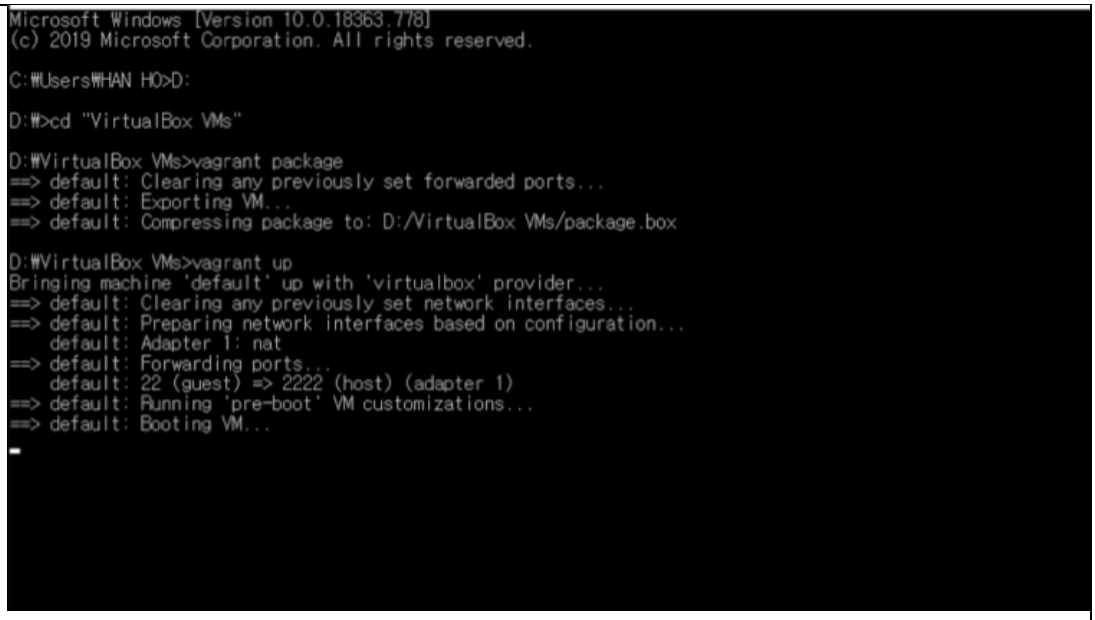
hyperledger 부동산 프로그램 작성

- 백업(backup) 방안 마련
- Token과 연관된 이체 구현

참여자(Participant)간 토큰이체
부동산(Realestate)거래시 금액에 대해 토큰 이체
거래금액(Token) 부족
거래대상(IsTrading)이 아닌 부동산 거래 안됨 구현

2. 주간 연구 상세

저번 주에 작성한 예제 프로그램이 일정정도 수준에 저장할 상태를 이룬 것 같아 백업을 하는 과정을 진행했습니다. 처음에는 윈도우 10에 IIS를 통해 Git상태로 저장을 하는 것이 낫다고 판단을 해서 설치와 설정을 진행했습니다. git Server는 bonobo서버로 진행을 했습니다. 윈도우에서 git를 다루고 IIS를 설정하는 과정에서 처음해서 그런지 생각하는 데로 잘 처리가 되지 않았습니다. git으로 서버 상태를 저장하는 것은 코드를 저장한다는 방향과 맞지 않는 것 같아서 다른 방법을 강구해보기로 했습니다.

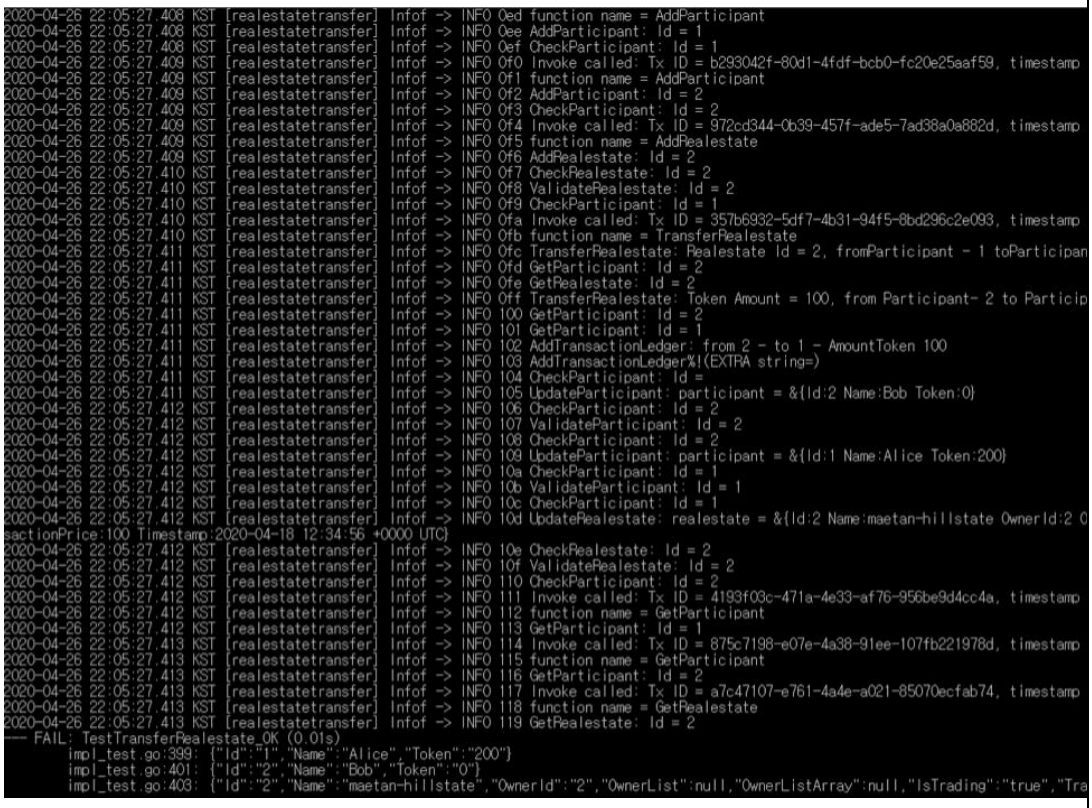


vagrant에서 제공하는 package를 통해서 현재 서버 상태를 저장했습니다.

현재 개발을 진행하고 있는 운영체제는 C드라이브에 있기 때문에 혹시나 발생할 하드디스크 문제가 생길 때 대응 할 수 있도록 D드라이브에 진행 상황이 저장할 만큼 발전이 되면 지속적으로 이미지 상태를 저장해 예측하지 못한 상황이 발생하면 대응할 수 있도록 할 생각입니다. 차후에는 이동디스크에 저장을 하는 매체를 늘려서 혹시나 발생할 위험에 대비할 수 있도록 진행하겠습니다.

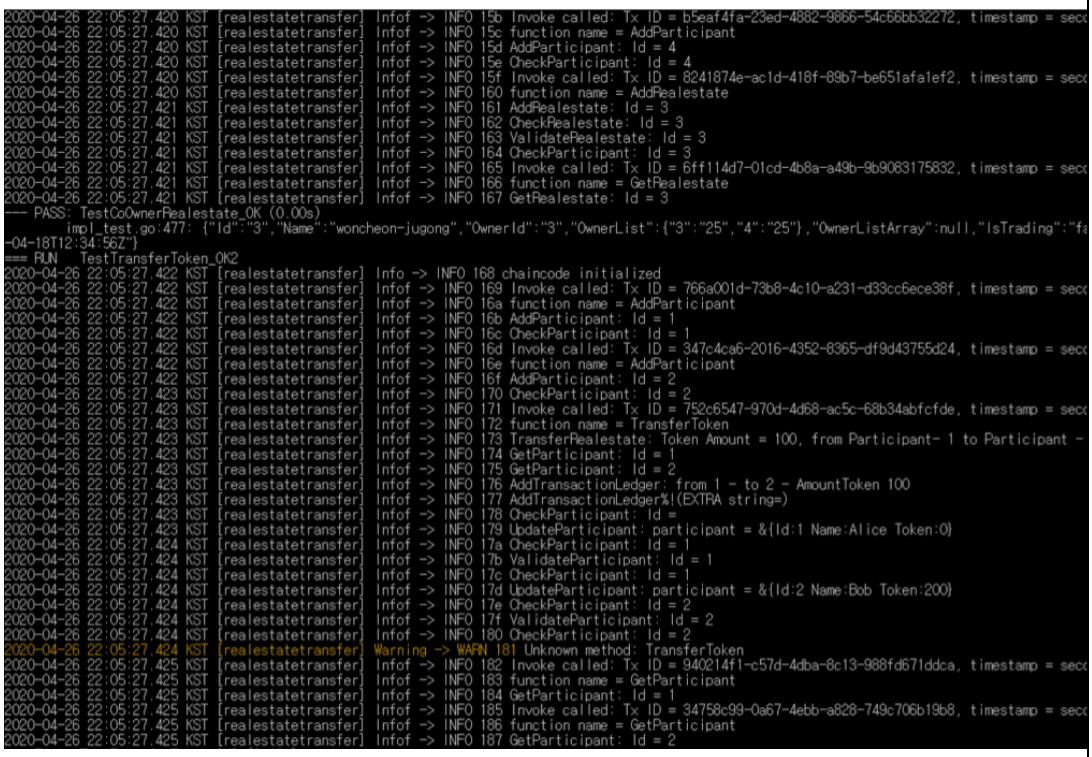
```
package chaincode_test

import (
	"sort"
	"fmt"
	"encoding/json"
	"realestatetransfer/chaincode"
	"github.com/hyperledger/fabric/common/util"
	"github.com/hyperledger/fabric/core/chaincode/shim"
	pb "github.com/hyperledger/fabric/protos/peer"
	"github.com/stretchr/testify/assert"
	"testing"
)

const (
	alice = `{"Id":"1","Name":"Alice","Token":"100"}`
	alice_zero = `{"Id":"1","Name":"Alice","Token":"0"}`
	bob = `{"Id":"2","Name":"Bob","Token":"100"}`
	bob_200 = `{"Id":"2","Name":"Bob","Token":"200"}`
	carol = `{"Id":"3","Name":"carol","Token":"100"}`
	dave = `{"Id":"4","Name":"dave","Token":"100"}`

	emptyParticipant = "[]"
	oneParticipant = "["+alice+"]"
	twoParticipants = "["+alice+","+bob+"]"

	timestamp = `"2020-04-18T12:34:56Z"`

	realestate1 = `{"Id":"1","Name":"eileen-garden","OwnerId":"1","IsTrading":"true","TransactionPrice":"200","Timestamp":`+timestamp+`}`
	realestate1b = `{"Id":"1","Name":"eileen-garden","OwnerId":"2","IsTrading":"true","TransactionPrice":"200","Timestamp":`+timestamp+`}`
	realestate2 = `{"Id":"2","Name":"maetan-hillstate","OwnerId":"1","IsTrading":"true","TransactionPrice":"100","Timestamp":`+timestamp+`}`
	realestate2b = `{"Id":"2","Name":"maetan-hillstate","OwnerId":"2","IsTrading":"true","TransactionPrice":"100","Timestamp":`+timestamp+`}`
	realestate3 = `{"Id":"3","Name":"woncheon-jugong","OwnerId":"3","OwnerList":{"3":"0.5","4":"0.5"},"IsTrading":"false","TransactionPrice":"50","Timestamp":`+timestamp+`}`
	realestate4 = `{"Id":"4","Name":"maetan-jugong","OwnerId":"2","IsTrading":"true","TransactionPrice":"10","Timestamp":`+timestamp+`}`

	oneRealestate = "["+realestate1+"]"
	twoRealestates = "["+realestate1+","+realestate2+"]"

	one = "1"
	two = "2"
	three = "3"
	hundred = "100"
	twoHundred = "200"
)
```

저번주에는 단순하게 부동산을 옮기기만 했습니다. 이번 주에는 부동산에 측정된 금액을 통해서 상대방에게 부동산의 소유권을 넘겨주고 금액을 받아오는 프로세스를 모델링했습니다. 위에 Realestate3, 4를 새로 정의해서 거래가능함(IsTrading)상태에 따라서 거래가 가능한지 혹은 불가능하면 에러를 발생시키는 것을 체인코드에서 구현했습니다.



```
2020-04-26 22:05:27.408 KST [realestatetransfer] Infof -> INFO 0ed function name = AddParticipant
2020-04-26 22:05:27.408 KST [realestatetransfer] Infof -> INFO 0ee AddParticipant: Id = 1
2020-04-26 22:05:27.408 KST [realestatetransfer] Infof -> INFO 0ef CheckParticipant: Id = 1
2020-04-26 22:05:27.409 KST [realestatetransfer] Infof -> INFO 0f0 Invoke called: Tx ID = b293042f-80d1-4fdf-bcb0-fc20e25aaf59, timestamp
2020-04-26 22:05:27.409 KST [realestatetransfer] Infof -> INFO 0f1 function name = AddParticipant
2020-04-26 22:05:27.409 KST [realestatetransfer] Infof -> INFO 0f2 AddParticipant: Id = 2
2020-04-26 22:05:27.409 KST [realestatetransfer] Infof -> INFO 0f3 CheckParticipant: Id = 2
2020-04-26 22:05:27.409 KST [realestatetransfer] Infof -> INFO 0f4 Invoke called: Tx ID = 972cd344-0b39-457f-ade5-7ad38a0a882d, timestamp
2020-04-26 22:05:27.409 KST [realestatetransfer] Infof -> INFO 0f5 function name = AddRealestate
2020-04-26 22:05:27.409 KST [realestatetransfer] Infof -> INFO 0f6 AddRealestate: Id = 2
2020-04-26 22:05:27.410 KST [realestatetransfer] Infof -> INFO 0f7 CheckRealestate: Id = 2
2020-04-26 22:05:27.410 KST [realestatetransfer] Infof -> INFO 0f8 ValidateRealestate: Id = 2
2020-04-26 22:05:27.410 KST [realestatetransfer] Infof -> INFO 0f9 CheckParticipant: Id = 1
2020-04-26 22:05:27.410 KST [realestatetransfer] Infof -> INFO 0fa Invoke called: Tx ID = 357b6932-5df7-4b31-94f5-8bd296c2e093, timestamp
2020-04-26 22:05:27.410 KST [realestatetransfer] Infof -> INFO 0fb function name = TransferRealestate
2020-04-26 22:05:27.411 KST [realestatetransfer] Infof -> INFO 0fc TransferRealestate: Realestate Id = 2, fromParticipant - 1 toParticipan
2020-04-26 22:05:27.411 KST [realestatetransfer] Infof -> INFO 0fd GetParticipant: Id = 2
2020-04-26 22:05:27.411 KST [realestatetransfer] Infof -> INFO 0fe GetRealestate: Id = 2
2020-04-26 22:05:27.411 KST [realestatetransfer] Infof -> INFO 0ff TransferRealestate: Token Amount = 100, from Participant- 2 to Particip
2020-04-26 22:05:27.411 KST [realestatetransfer] Infof -> INFO 100 GetParticipant: Id = 2
2020-04-26 22:05:27.411 KST [realestatetransfer] Infof -> INFO 101 GetParticipant: Id = 1
2020-04-26 22:05:27.411 KST [realestatetransfer] Infof -> INFO 102 AddTransactionLedger: from 2 - to 1 - AmountToken 100
2020-04-26 22:05:27.411 KST [realestatetransfer] Infof -> INFO 103 AddTransactionLedger%!(EXTRA string=)
2020-04-26 22:05:27.411 KST [realestatetransfer] Infof -> INFO 104 CheckParticipant: Id =
2020-04-26 22:05:27.411 KST [realestatetransfer] Infof -> INFO 105 UpdateParticipant: participant = &{Id:2 Name:Bob Token:0}
2020-04-26 22:05:27.412 KST [realestatetransfer] Infof -> INFO 106 CheckParticipant: Id = 2
2020-04-26 22:05:27.412 KST [realestatetransfer] Infof -> INFO 107 ValidateParticipant: Id = 2
2020-04-26 22:05:27.412 KST [realestatetransfer] Infof -> INFO 108 CheckParticipant: Id = 2
2020-04-26 22:05:27.412 KST [realestatetransfer] Infof -> INFO 109 UpdateParticipant: participant = &{Id:1 Name:Alice Token:200}
2020-04-26 22:05:27.412 KST [realestatetransfer] Infof -> INFO 10a CheckParticipant: Id = 1
2020-04-26 22:05:27.412 KST [realestatetransfer] Infof -> INFO 10b ValidateParticipant: Id = 1
2020-04-26 22:05:27.412 KST [realestatetransfer] Infof -> INFO 10c CheckParticipant: Id = 1
2020-04-26 22:05:27.412 KST [realestatetransfer] Infof -> INFO 10d UpdateRealestate: realestate = &{Id:2 Name:maetan-hillstate OwnerId:2 O
sactionPrice:100 Timestamp:2020-04-18 12:34:56 +0000 UTC}
2020-04-26 22:05:27.412 KST [realestatetransfer] Infof -> INFO 10e CheckRealestate: Id = 2
2020-04-26 22:05:27.412 KST [realestatetransfer] Infof -> INFO 10f ValidateRealestate: Id = 2
2020-04-26 22:05:27.412 KST [realestatetransfer] Infof -> INFO 110 CheckParticipant: Id = 2
2020-04-26 22:05:27.412 KST [realestatetransfer] Infof -> INFO 111 Invoke called: Tx ID = 4193f03c-471a-4e33-af76-956be9d4cc4a, timestamp
2020-04-26 22:05:27.412 KST [realestatetransfer] Infof -> INFO 112 function name = GetParticipant
2020-04-26 22:05:27.412 KST [realestatetransfer] Infof -> INFO 113 GetParticipant: Id = 1
2020-04-26 22:05:27.413 KST [realestatetransfer] Infof -> INFO 114 Invoke called: Tx ID = 875c7198-e07e-4a38-91ee-107fb221978d, timestamp
2020-04-26 22:05:27.413 KST [realestatetransfer] Infof -> INFO 115 function name = GetParticipant
2020-04-26 22:05:27.413 KST [realestatetransfer] Infof -> INFO 116 GetParticipant: Id = 2
2020-04-26 22:05:27.413 KST [realestatetransfer] Infof -> INFO 117 Invoke called: Tx ID = a7c47107-e761-4a4e-a021-85070ecfab74, timestamp
2020-04-26 22:05:27.413 KST [realestatetransfer] Infof -> INFO 118 function name = GetRealestate
2020-04-26 22:05:27.413 KST [realestatetransfer] Infof -> INFO 119 GetRealestate: Id = 2
--- FAIL: TestTransferRealestate_OK (0.01s)
        impl_test.go:399: {"Id":"1","Name":"Alice","Token":"200"}
        impl_test.go:401: {"Id":"2","Name":"Bob","Token":"0"}
        impl_test.go:403: {"Id":"2","Name":"maetan-hillstate","OwnerId":"2","OwnerList":null,"OwnerListArray":null,"IsTrading":"true","Tra
```

부동산 거래에 따라서 토큰이 옮겨가고 부동산 소유가 넘어가는 것을 확인 할 수 있습니다.



공동소유를 하는 부동산과 토큰만 주고받는 기능을 테스트를 하는 것도 볼 수 있습니다. 거래 대상이 아닌 것을 거래가 가능하지 않게 하고 에러가 나오게 하는 것과 거래금액이 부족하면 거래금액이 충분하지 않다는 에러를 내면서 거래가 되지 않게 하는 등의 에러 처리도 되도록 했습니다.

3. 연구에서 미진했던 점

공동 소유를 처리를 데이터를 받아들이는 것까지는 시간이 별로 걸리지 않았습니다. 시간이 많이 들어간 부분은 단독소유의 부동산을 다수의 공동소유를 하려는 사람에게 팔려는 거래, 다수의 공동소유를 하고 있는 부동산을 단독 혹은 다수에게 팔려는 거래를 구현하려고 하는데 시간이 많이 들어갔습니다. 구현에 문제인지 아니면 논리의 문제인지 시간을 많이 들어갔지만 M대 N거래는 앞으로 계속 부딪칠 문제라고 생각이 들기 때문에 시간을 많이 들이더라도 구현을 할 수 있도록 시간을 더 많이 쏟도록 하겠습니다.

4. 연구 진행시 생긴 의문점 혹은 아이디어

현재 구현이 앞으로 구현을 할 공동소유 채권/채무 관계를 무리 없이 구현을 할 수 있을지 고민을 하고 있습니다. 거래를 생각하게 되니 에스크로와 같은 중간 단계를 두어서 거래 안정성을 높이는 것이 좋을 것

같다는 생각이 들었습니다. 부동산에서 가장 큰 문제는 자금을 옮기고 받는 시간의 차이와 거래의 차이가 있기 때문에 필연적으로 발생하는것 아닌가 싶었습니다. 결국 기존에 거래의 시간차를 해결하는 방법은 안전 결제 시스템을 통해서 해결하는 것이 무난한 방법이라는 생각이 들었습니다.

앞으로 테스트를 하는 과정에서 이 자금은 어디서 흘러왔는가를 거래 시작을 통해서 흐름을 추상화할 수 있으면 좋겠다는 생각이 들었습니다. 참여자에 회사 혹은 금융기관을 통해서 자금을 이체받고 자금에 원천이 참여자에게 타서 거래를 모델링 할 수 있다면 상위 기관의 다른 거래를 확장시켜가는 과정의 지난함을 줄일 수 있지 않을까 하는 아이디어가 들었습니다. 처음에 자금의 과정을 모델링하기 위해 데이터 타입을 더 쪼개고 채권과 채무과정을 쉽게 더하게 하기 위해 다른 클래스 분리하기가 필요하다는 생각이 들었습니다.

```
type Participant struct {
        Id string
        Name string
        Token string
}

type TransactionLedger struct {
        Id string
        FromId string
        ToId string
        AmountToken string
}

type OwnerListLedger struct {
        OwnerId string
        TransactionAmount string
}

type Realestate struct {
        Id string
        Name string
        OwnerId string
        OwnerList map[string]string
        OwnerListArray []OwnerListLedger
        IsTrading string
        TransactionPrice string
        Timestamp time.Time
}

type RealestateTransfer interface {
        // Participant method interface
        AddParticipant(shim.ChaincodeStubInterface, *Participant) error
        GetParticipant(shim.ChaincodeStubInterface, string) (*Participant, error)
        UpdateParticipant(shim.ChaincodeStubInterface, *Participant) error
        CheckParticipant(shim.ChaincodeStubInterface, string) (bool, error)
        ValidateParticipant(shim.ChaincodeStubInterface, *Participant) (bool, error)
        ListParticipants(shim.ChaincodeStubInterface) ([]*Participant, error)

        TransferToken(shim.ChaincodeStubInterface, string, string, string) error

        // TransactionLedger method interface
        AddTransactionLedger(shim.ChaincodeStubInterface, *TransactionLedger) error
        AddTransactionLedgerbyAttr(shim.ChaincodeStubInterface, string, string, string) error

        // Realestate method interface
        AddRealestate(shim.ChaincodeStubInterface, *Realestate) error
        CheckRealestate(shim.ChaincodeStubInterface, string) (bool, error)
        ValidateRealestate(shim.ChaincodeStubInterface, *Realestate) (bool, error)
```

대략적으로 진행을 하는 청사진은 위에 과정을 통해서 돈의 흐름을 남기는 TransactionLedger라는 구조체를 생성해서 자금과 연관되는 모든 과정은 추적할 수 있도록 하게하는 것이 어떤가하고 생각을 하고 있습니다. 공동 소유도 위에 OwnerList라는 변수를 쓰는 것이 아니라 OwnerListLedger라는 구조체를 만들어서 Realestate구조체이 OwnerListArray라는 배열을 사용하는 것이 더 낫지 않을까 고민을 하고 있습니다. 공동 소유 거래를 구현하면서 코드의 복잡함이 구현의 빠르기를 넘어서는 순간이라는 생각이 들어서 다른 설계를 통해 극복하는 것이 어떠한지 고민중에 있습니다.