SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Zadanie 3

Hyperledger Fabric smart systém

Adam Valach, Anna Yuová

Predmet: Digitálne meny a blockchain

Akademický rok: 2020/2021

Semester: letný

Obsah

Cieľ projektu

Cieľom tohto zadania bolo vytvoriť chaincode a následne ho nasadiť pomocou Hyperledger Fabric. V rámci toho sme implementovali biznis sieť medzi aerolíniami a cestovnou kanceláriou.

Navrhnutá architektúra a jej prepojenia

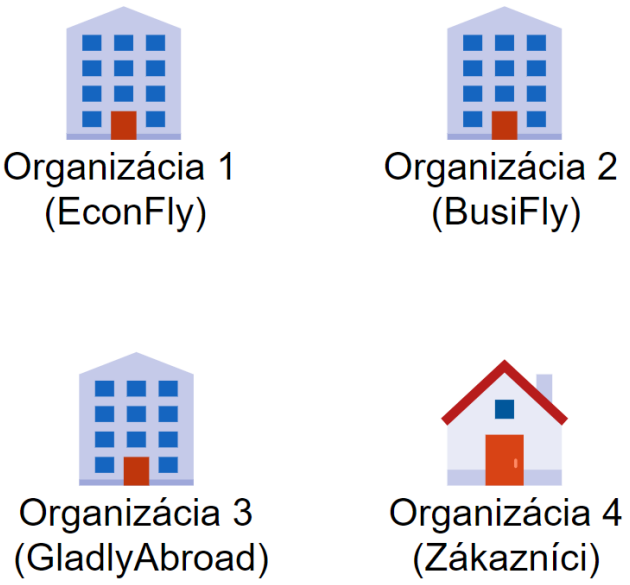
Architektúra nášho chaincodu prepája, aktualizuje a umožňuje komunikáciu medzi všetkými organizáciami v rámci siete.

Náš chaincode sa viaže na kanál *mychannel.* V rámci nášho lokálneho fabric-samples/test-network sme si vytvorili jeden channel (“*mychannel”),* kde sme si spustili čistú verziu našej testovacej siete. Keďže Org1 a Org2 sú tam defaultne, stačilo nám pridať už len ďalšiu organizáciu príkazom addOrg3.

V našom chaincode teda pracujeme s 3 organizáciami - organizácia 1 je aerolínia EconFly, organizácia 2 je aerolínia BusiFly a organizácia 3 je cestovná kancelária. Na kontrolu toho, čo môžu jednotlivé organizácie vykonávať, sme ošetrili podmienkami vo funkciách *createFlight( )*, *reserveSeats( )*, *bookSeats*( ) a *checkIn*( ), kde sme na začiatku skontrolovali či sú dané funkcie volané očakávanými organizáciami, inak by funkcia vrátila false.

Chaincode je spúšťaný na peeroch s cieľom vytvárať transakcie. Sú to účastníci v rámci každej organizácie, ktorí boli automaticky vytvorení pri pridaní organizácie. Na účely testovania stačil jeden peer v každej organizácii.

V rámci testovania sme pracovali iba s 3 organizáciami, avšak chaincode podporuje aj prítomnosť 4. organizácie reprezentujúcej zákazníka (podmienky sú dobre nastavené, akurát sme nevytvárali kópiu addOrg3 a neprerábali ju na addOrg4).



Obrázok č.1: Naše organizácie

Implementované časti kódu

Pri implementovaní sme vytvorili nový Gradle projekt, ktorému sme museli nastaviť build.gradle súbor. Tento postup popisuje aj článok na webe Medium [1].

```
plugins {
    id 'checkstyle'
    id 'java-library-distribution'
}

group 'org.example'
version '1.0'

repositories {
    mavenCentral()
}

dependencies {
    compileOnly 'org.hyperledger.fabric-chaincode-java:fabric-chaincode-shim:2.0.+'
    implementation 'com.owlike:genson:1.5'
    testImplementation 'org.hyperledger.fabric-chaincode-java:fabric-chaincode-shim:2.0.+'
    testImplementation 'org.junit.jupiter:junit-jupiter:5.4.2'
    testImplementation 'org.assertj:assertj-core:3.11.1'
    testImplementation 'org.mockito:mockito-core:2.+'
}
```

Obrázok č. 2: Nastavenie pluginov v build.gradle

Ďalej sme vytvorili triedy *Let*, *Rezervácia* a *FlyNetContract*:

Class Rezervacia:
Predstavuje dátový typ, ktorý bude na ledgeri reprezentovať rezerváciu. Obsahuje údaje ako:

- reservationNr (ID rezervácie, začíname od 1 a majú prefix R)
- customerNames (Mená zákazníkov)
- customerEmail (E-mail zákazníka)
- flightNr (ID letu)
- nrOfSeats (Počet rezervovaných sedadiel)
- status (Stav objednávky)

Tieto premenné majú nastavenú anotáciu @Property, pretože predstavujú atribúty tohto dátového typu. Tiež sme vytvorili im zodpovedajúce gettery, settery a konštruktor - napr. *getCustomerNames( )*, *setFlightNr( )* a pod.

Class Let:
Predstavuje dátový typ, ktorý bude na ledgeri reprezentovať let.
Obsahuje údaje ako:

- flightNr (ID letu, začíname od 1 a majú prefixy EC alebo BS)
- flyFrom (Odkiaľ)
- flyTo (Kam)
- dateTime (Kedy)
- availablePlaces (Počet voľných miest)

Tieto premenné majú tiež nastavenú anotáciu @Property a vytvorili sme im aj príslušné gettery, settery a konštruktor - napr. *getFlightNr( )*, *setAvailablePlaces( ) a pod*.

Class FlyNetContract:
Implementuje hlavnú logiku nášho chaincodu a obsahuje hlavné funkcie:

- createFlight()
- getAllFlights()
- getFlight()
- reserveSeats()
- bookSeats()
- checkIn()

Všetky funkcie majú anotáciu @Transaction, pretože definujú transakcie v našom smart kontrakte.

createFlight( ): Na začiatku kontrolujeme či je funkcia volaná 3. organizáciou (cestovnou kanceláriou), pretože táto funkcia nesmie byť volaná aerolíniami alebo zákazníkom. Vytvorili sme novú inštanciu triedy *Let* s potrebnými parametrami - *flightNr*, *flyFrom*, *flyTo*, *dateTime*, a *availablePlaces*. Na odlíšenie letov, ktoré poskytujú aerolínie sme si vytvorili ID. ID je buď "*EC*" alebo "*BS*" podľa toho, či je let poskytovaný 1. organizáciou (EconFly) alebo 2. organizáciou (BusiFly). Postupne pridáme k daným identifikátorom čísla v poradí, v akom sú vytvárané (napr. EC1, EC2, ...). Ak sa let podarilo úspešne vytvoriť, funkcia vracia true. Počty jednotlivých letov sú uložené v ledgeri na miestach s kľúčmi "EC" a "BS".

```
Let let = new Let(id, flyFrom, flyTo, dateTime, Integer.parseInt(seats));
ctx.getStub().putState(id, let.toJSONString().getBytes(UTF_8));

if(ctx.getClientIdentity().getMSPID().equals("Org1MSP")) {
    if(new String(ctx.getStub().getState( key: "EC"), UTF_8).length() == 0) {
        ctx.getStub().putState("EC", "1".getBytes(UTF_8));
    }
    else {
        int count = Integer.parseInt(new String(ctx.getStub().getState( key: "EC"), UTF_8)) + 1;
        ctx.getStub().putState("EC", Integer.toString(count).getBytes(UTF_8));
    }
}
```

Obrázok č. 3: Vytvorenie inštancie Let

getAllFlights( ): Funkcia vracia všetky dostupné lety - do stringu "*response"* si ukladáme postupne najprv všetky lety EC v poradí ako idú za sebou a za ne zapíšeme všetky lety BS. Jednotlivé lety sú medzi sebou oddelené čiarkou (podľa oficálneho JSON formátu).
Ukážka výsledného stringu response:
{flights[(dateTime: "2020", flightNr: "EC1", flyFrom: "NR", availablePlaces: "20", flyTo: "BA"), (dateTime: "2020", flightNr: "BS1", flyFrom: "TN", availablePlaces: "15", flyTo: "TT"),...]}

```
String response = "{flights: [";
Let let;
if(ec != 0) {
    for(int i = 1; i <= ec; i++) {
        response += new String(ctx.getStub().getState( key: "EC" + i),UTF_8);
        if(!(bs == 0 && i == ec)) {
            response += ", ";
        }
    }
}
if(bs != 0) {
    for(int i = 1; i <= bs; i++) {
        response += new String(ctx.getStub().getState( key: "BS" + i),UTF_8);
        if(i != bs) {
            response += ", ";
        }
    }
}
response += "]}";
return response;
```

Obrázok č. 4: Postupné ukladanie všetkých letov za seba do stringu

reserveSeats( ): Na začiatku kontrolujeme, či je táto funkcia volaná 3. organizáciou (cestovná kancelária), pretože funkcia môže byť volaná jedine cestovnou kanceláriou. Rezervácii nastavíme *reservationNr* ako ID (R + poradové číslo), *flightNr*, *nrOfSeats*, *customerNames*, *customerEmail* a stav na “Pending”. Ak do funkcie pošleme správne argumenty, napríklad [“BS2”, “2”, “[Adam,Anna]”, “email@gmail.com”] tak nám funkcia vráti true. Počty jednotlivých rezervácii sú uložené v ledgeri na mieste s kľúčom “R”.

bookSeats( ): Na začiatku kontrolujeme či je táto funkcia volaná 1. alebo 2. organizáciou (EconFly alebo BusiFly), pretože funkcia môže byť volaná len aerolíniami. Skontrolujeme, či je dostatok voľných miest v lietadle (*getAvailablePlaces( )*), pre taký počet aký chceme rezervovať - teda, či je počet voľných miest väčší alebo rovný počtu miest, ktoré sme sa rozhodli rezervovať. Ak je táto podmienka splnená, tak zmeníme stav na “Completed” a aktualizujeme počet voľných miest (*počet voľných miest - počet miest, ktoré sme rezervovali*).

```
if(let.getAvailablePlaces() >= rezervacia.getNrOfSeats()) {
    rezervacia.setStatus("Completed");
    let.setAvailablePlaces(let.getAvailablePlaces() - rezervacia.getNrOfSeats());
    ctx.getStub().putState(reservationNr, rezervacia.toJSONString().getBytes(UTF_8));
    ctx.getStub().putState(let.getFlightNr(), let.toJSONString().getBytes(UTF_8));
}
```

Obrázok č. 5: Kontrola a aktualizácia dostupných miest

checkIn( ): Na začiatku kontrolujeme, či je daná funkcia volaná 3. alebo 4. organizáciou (cestovnou kanceláriou alebo zákazníkom), pretože táto funkcia nemôže byť volaná leteckými spoločnosťami. Funkcia *checkIn( )* porovnáva mená cestujúcich z rezervácie (*customerNames*) s menami získanými pri kontrole

osobných dokladov (*passportIDs*). Ak mená a počty cestujúcich sedia, tak nám funkcia vráti true (premenná *valid* je true) a zároveň nastaví stav rezervácie z “Completed” na “Checked-in”. Ak sa mená nezhodujú, jedno z mien nenašlo alebo je cestujúcich viac ako v rezervácii, tak sa premenná *valid* nastaví na false a tým pádom sa sedadlá nepridelia.

```
if(valid) {
    for(int i = 0; i < passportIDs.length; i++) {
        for(int j = 0; j < customerNames.length; j++) {
            if(j == customerNames.length - 1 && !passportIDs[i].equals(customerNames[j]))
                valid = false;
                break;
            }
            if(passportIDs[i].equals(customerNames[j])) {
                break;
            }
        }
        if(!valid) {
            break;
        }
```

Obrázok č. 6: Kontrola, či sa počty mien zhodujú so sedadlami

Implementačné prostredie

Chaincode sme sa rozhodli implementovať v programovacom jazyku Java. Kód sme písali v IntelliJ IDE (verzia 2021.1.1), kde sme vytvorili nový Gradle projekt. Pre nasadenie projektu do siete Hyperledger a všetko, čo súvisí s nasadením chaincode-u sme robili v Terminali operačného systému Ubuntu 20.04.2 LTS.

Návod na spustenie

Inštalácia Hyperledger Fabric
curl -sSL https://bit.ly/2ysbOFE | bash -s -- 2.3.2 1.5.0

Vytvorenie “*mychannel*”
./network.sh up createChannel -c mychannel -ca

Pridanie 3. organizácie *(pred pridaním treba upraviť v priečinku ./addOrg3/docker verzie z 2 na 3.5)*
cd ./addOrg3/
./addOrg3.sh up
cd ..

Nasadenie chaincode *(pribalený chaincode treba vložiť do priečinka fabric-samples)*
./network.sh deployCC -ccn dmblock3 -ccp ../dmblock3 -ccl java

Nastavenie premennej path pre príkaz peer
export PATH=${PWD}/../bin:$PATH
export FABRIC_CFG_PATH=$PWD/../config/
peer version

Prepínanie medzi organizáciami:
export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_LOCALMSPID=”Org1MSP”
export
CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
export
CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganization/org1.example.com/users/Admin@org1.example.com/msp
export CORE_PEER_ADDRESS=localhost:7051

export CORE_PEER_LOCALMSPID=”Org2MSP”
export
CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt
export
CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganization/org2.example.com/users/Admin@org2.example.com/msp
export CORE_PEER_ADDRESS=localhost:9051

export CORE_PEER_LOCALMSPID=”Org3MSP”
export
CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org3.example.com/peers/peer0.org3.example.com/tls/ca.crt
export
CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganization/org3.example.com/users/Admin@org3.example.com/msp
export CORE_PEER_ADDRESS=localhost:11051

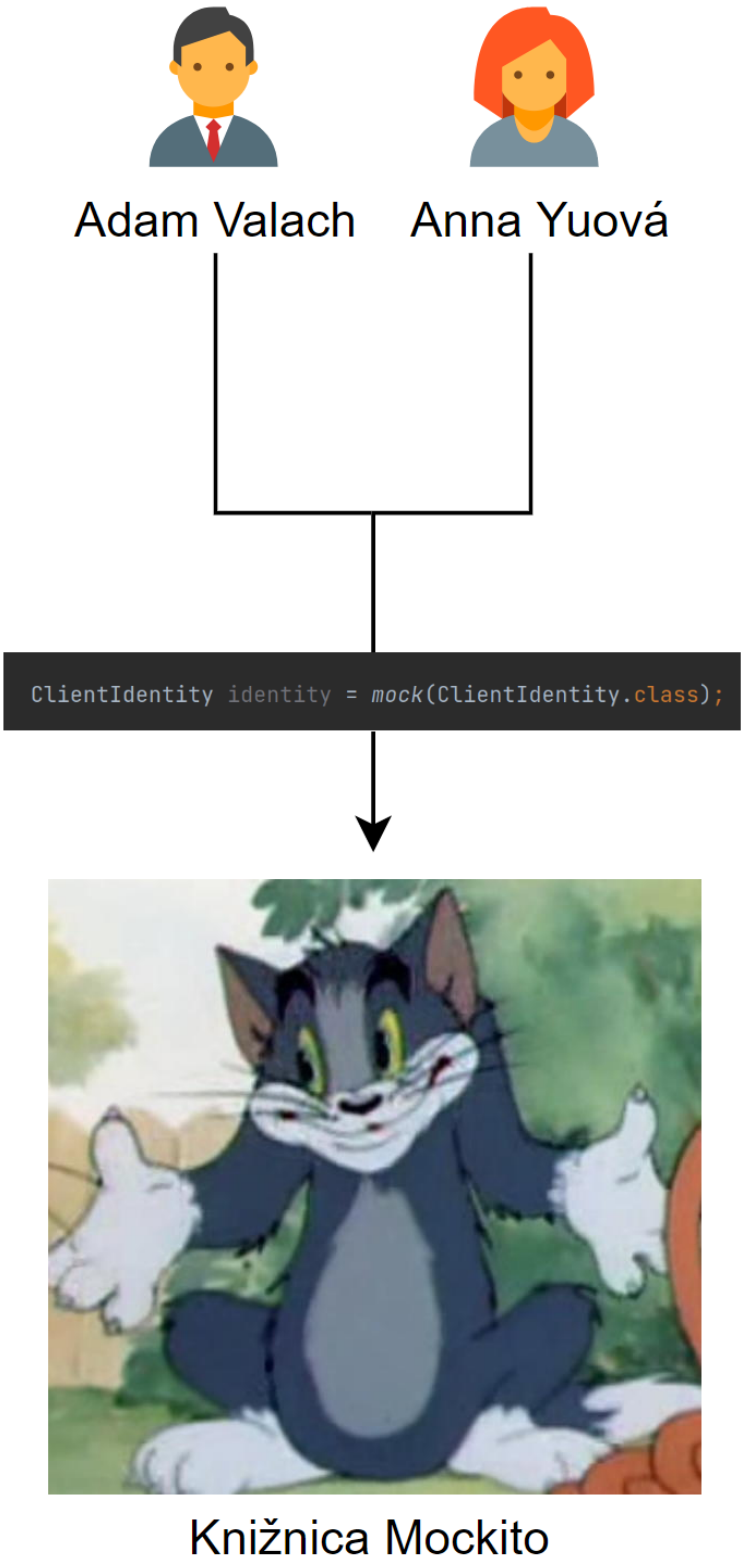
Volanie jednotlivých funkcií:
peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile ${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C mychannel -n dmblock3 --peerAddresses localhost:7051 --tlsRootCertFiles ${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt --peerAddresses localhost:9051 --tlsRootCertFiles ${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt -c '{"function":"nazov_funkcie","Args":["arg1","arg2","arg3","arg4"]}'

Testovanie

Testovanie prebiehalo za pomoci knižníc Mockito a JUnit v IntelliJ IDEA. Inšpirovali sme sa oficiálnym Github repozitárom Hyperledgeru [2] a jednou ďalšou implementáciou na Github-e [3]. Boli sme schopní otestovať iba 2 funkcie - tie, v ktorých nedochádza k overovaniu organizácie, nakoľko trieda ClientIdentity je typu final a tým pádom na ňu nevieme použiť funkciu *mock( )* z testovacej knižnice Mockito.

Nakoľko Mockito a JUnit boli v obidvoch nájdených implementáciách testov, dospeli sme k záveru, že Mockito a JUnit sú “oficiálnou cestou” pre testovanie, avšak kvôli nemožnosti volania funkcie *mock( )* a faktu, že Java Hyperledger tutoriálov je nedostatok (nenašli sme žiadne testy, kde by sa pracovalo s ClientIdentity) sme tiež usúdili, že testovanie klientských identít pravdepodobne zatiaľ nie je podporované.

Naše zistenie možno vidieť aj na diagrame interakcie s danými knižnicami na obrázku č. 7.



Obrázok č. 7: Diagram interakcie s testovacou knižnicou a triedou ClientIdentity

Testované boli funkcie *getAllFlights()* a *getFlight( )*.

Pre funkciu *getAllFlights( )* bolo testovaných 5 scenárov:

- **Keď neexistujú žiadne lety**: Program by mal vrátiť prázdne pole *flights* (“{flights: []}”)
- **Keď existuje 1 EC let a 0 BS letov**: Program by mal vrátiť korektný JSON formát reprezentácie pola flights s jedným prvkom (mal by sa vykonať iba prvý for cyklus).
- **Keď existuje 1 EC let a 1 BS let**: Program by mal vrátiť korektný JSON formát reprezentácie pola flights s dvoma prvkami (mali by sa vykonať obidva for cykly).
- **Keď existujú 2 EC lety a 1 BS let**: Program by mal vrátiť korektný JSON formát reprezentácie pola flights s tromi prvkami (mali by sa vykonať obidva for cykly).
- **Keď existujú 2 EC lety a 2 BS lety**: Program by mal vrátiť korektný JSON formát reprezentácie pola flights s tromi prvkami (mali by sa vykonať obidva for cykly, kde v druhom *for cykle*, by sa prvýkrát nemala vykonať *if* podmienka).

Pre funkciu *getFlight( )* boli testované 2 scenáre:

- **Hľadaný let neexistuje**: Funkcia by mala vrátiť prázdny String.
- **Hľadaný let existuje**: Funkcia by mala vrátiť reprezentáciu objektu v JSON formáte

Odpovede na otázky

Podľa vášho názoru, je takéto blockchain-based riešenie najlepšia možnosť na vyriešenie daných výziev?

V zadaní tohto projektu boli spomenuté nasledovné nevýhody starého centralizované systému:

- stratené rezervácie letov
- neplatné dáta pri letoch
- náhodné zmeny rezervácií zákazníkom

Tieto veci rieši využitie decentralizovanej účtovnej knihy, nakoľko decentralizované databázy je nemožné upravovať bez konsenzu väčšiny siete. Taktiež podľa stránky ***doyouneedablockchain.com*** by bolo možné využitie permissioned blockchainu, ako je to v tomto prípade.

V prípade, že by si dané spoločnosti medzi sebou dôverovali (s pribúdajúcim počtom spoločností je to však čoraz komplikovanejšie) by bolo centralizované riešenie postačujúce.

Aké sú výhody a nevýhody používania technológie distribuovanej účtovnej knihy (distributed ledger technology) v porovnaní s centralizovaným systémom na tomto konkrétnom prípade použitia?

Výhody:

- Dôvera aj v nedôveryhodných biznis prostrediach (účastník si nemôže prikrášliť svoj podiel zo zisku úpravou záznamov v databáze)
- V našej implementácii sa vyhnú náhodným úpravám/miznutiu dát

Nevýhody:

- Decentralizované blockchain siete sú pri veľkom návale transakcii často neefektívne, nakoľko každé volanie smart kontraktu musí byť vykonané každým uzlom v sieti
- Za predpokladu, že by sa jednalo o dôveryhodné prostredie, firmy by ušetrili nasadením centralizovaného systému, nakoľko blockchain je relatívne nová téma a počet špecialistov na decentralizované systémy je podstatne nižší ako na tie tradičné.

Záver

Pri tomto zadaní sme sa naučili implementovať a nasadiť chaincode v sieti Hyperledger Fabric. Zhodnotili sme, že Java asi nebola najvhodnejšou voľbou pre implementáciu, a to z dôvodu problémov pri testovaní a nízkeho počtu návodov/ukážok k testovaniu a vývoju na Hyperledgeri. Naučili sme sa ako funguje sieť Hyperledger Fabric a pochopili fungovanie a význam channelov, organizácií a peerov.

Percentuálne porovnanie podielu študentov: 50:50

Zdroje

[1] I. Alberquilla, “How to create a Java chaincode and deploy in a Hyperledger Fabric 2 network,” Medium, Aug. 21, 2020. https://medium.com/coinmonks/how-to-create-a-java-chaincode-and-deploy-in-a-hyperledger-fabric-2-network-65199e5f645d (accessed May 14, 2021).

[2] Hyperledger, “hyperledger/fabric-chaincode-java,” GitHub. https://github.com/hyperledger/fabric-chaincode-java (accessed May 14, 2021).

[3] I. Alberquilla, “AgreementRepositoryTest.java,” Gist. https://gist.github.com/ialberquilla/1472039b802b874174e876b15c6176b8 (accessed May 14, 2021).