UNIVERZITET U NOVOM SADU

FAKULTET TEHNIČKIH NAUKA U NOVOM SADU

Održavanje i kontrola kvaliteta softvera u infrastrukturnim sistemima

- PREDMETNI PROJEKAT -

Projektna dokumentacija

https://github.com/nikolaJov2605/checkers

Nikola Jovićević, E5 8/2022

Checkers (“mice”) je društvena igra u kojoj takmičari pomeraju žetone po tabli, pri čemu se trude da preskakanjem uklone protivničke žetone. Takmičar čije figure poslednje ostanu na talonu odnosi pobedu. Ova igrica je za potrebe projekta implementirana je na blockchain platformi, upotrebom CosmosSDK framework-a, i pisana je u Go programskom jeziku.

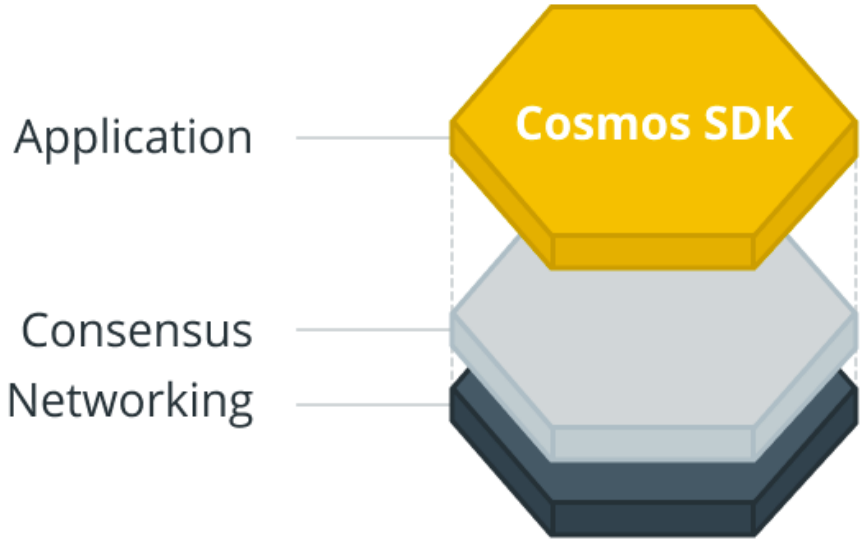
CosmosSDK

CosmosSDK (Cosmos Software Development Kit) predstavlja open-source framework za kreiranje višenamenskih, application-specific Proof-of-Stake (PoS), kao i Proof-of-Authority (PoA) blockchain-ova. CosmosSDK je kao alat veoma koristan jer dizajn na kom funckioniše omogućava raslojavanje, skalabilnost, kao i interoperabilnost sa drugim blockchain-ovima.

Svaka CosmosSDK aplikacija sastoji se iz tri fiksna sloja:

1) Networking layer;
2) Consensus layer;
3) State machine layer (aplikacija),

pri čemu je za prva dva sloja zadužena komponenta po imenu CometBFT, dok se izrada same aplikacije koja se oslanja na networking i consensus slojeve, sprovodi u okviru CosmosSDK-a.



Slika 1: Raslojavanje Blockchain aplikacije

Što se tiče komunikacije između CosmosSDK-a i CometBFT-a, značajnu ulogu ima ABCI (Application Blockchain Interface), od skoro unapređen u ABCI++. ABCI služi da prenosi poruke sa consensus sloja aplikativnom nivou, pri čemu dolazi do iniciranja odgovarajućih ponašanja sa strane aplikacije. Najvažnije poruke koje se prenose su:

- CheckTx – kada transakcija prispe iz CometBFT sloja, ona se prosleđuje aplikaciji na osnovne provere validnosti transakcije i popunjenih polja. Ova poruka uglavnom služi da bi se mempool (baza transakcija koje čekaju na izvršenje) zaštitio od nevalidnih transakcija i spam-a. Ako je transakcija validna, prosleđuje se mempool-u, a zatim i peer node-ovima (čvorovima u mreži).
- DeliverTx – nakon što je validan blok prispeo od strane CometBFT-a, svaka transakcija iz bloka dolazi do aplikacije kako bi bila izvršena. Pre izvršenja, svaka poruka se ponovo proverava, a zatim se handle-uje na odgovarajući način.
- BeginBlock/EndBlock – Ove poruke se prosleđuju na početku i kraju svakog bloka, bez obzira na to da li blok sadrži neku transakciju ili ne. Pri upotrebi ovih poruka, mora se voditi računa o kompleksnosti njihovog procesuiranja, jer skupe operacije mogu usporiti blockchain, ili ga čak i zaustaviti ukoliko dođe do mrtve petlje.

Svaka transakcija, handle-uje se na sledeći način:

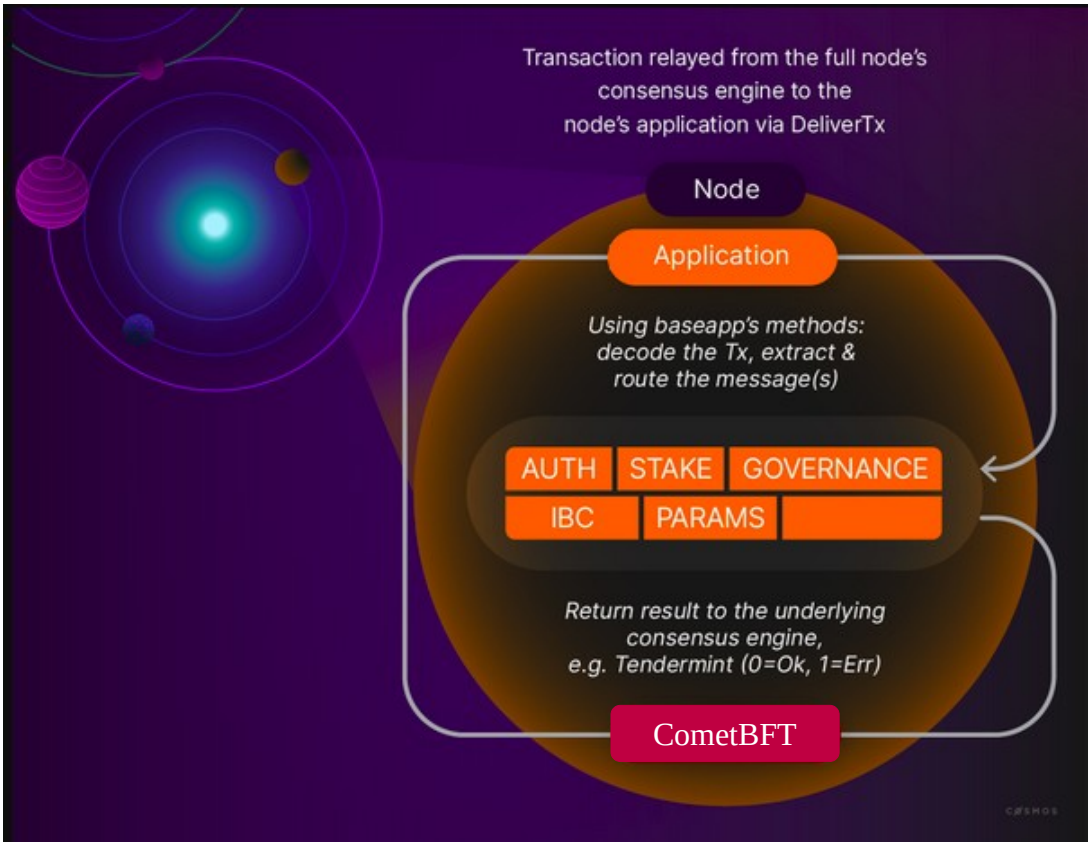
- Za početak se mora dekodirati jer CometBFT radi jedino sa nizovima bajtova;
- Zatim se transakcije raspakuju i rade se osnovne validacije;
- Svaka poruka se rutira do odgovarajućeg modula u okviru aplikacije;
- Commit-uju se promene stanja.

Za prva tri koraka zadužena je **baseapp** komponenta u kojoj su implementirane sve pojedinosti vezane za interakciju sa CometBFT komponentom. Dalje uloge u aplikaciji preuzimaju **moduli** – odvojene komponente od kojih svaka ima svoju ulogu, a koji mogu međusobno razmenjivati podatke. Rute između modula definišu se takođe u okviru **baseapp** komponente.

Neki od najvažnijih CosmosSDK modula su:

- **x/auth** – služi za rukovanje nalozima i potpisima;
- **x/bank** – omogućava upotrebu tokena, kao i njihov transfer;
- **x/staking** – koristi se za staking u Proof-of-Stake chain-ovima;
- **x/slashing** – za kažnjavanje nesavesnih validatora u Proof-of-Stake algoritmu;
- **x/gov** – omogućava on-chain governance, odnosno mogućnost korisnika da glasaju za određene predloge izmena u funkcionisanju mreže.

Važno je naglasiti da je grupisanje po modulima omogućilo da se jednom napisani moduli mogu koristiti u nekim budućim rešenjima, i da pisanjem CosmosSDK aplikacije, mi zapravo kreiramo nove module, koje po potrebi možemo uvezati sa ostalima i na taj način kreirati skladnu celinu.



Slika 2: Životni vek transakcije u CosmosSDK aplikaciji

Checkers modul

Za potrebe implementacije ove igrice, kreiran je checkers modul. Prilikom kreiranja modula, potrebno je uvezati ga sa svim ostalim komponentama sistema. Isto to je potrebno i prilikom kreiranja novih transakcija, upita, tipova i event-ova. Činjenica je da je samo za dodavanje jedne nove vrste poruke potrebno izmeniti nekoliko fajlova. U tome je od velike pomoći alat po imenu Ignite, a koji uz svega par komandi kreira nove tipove koji nam trebaju. Uz pomoć Ignite-a kreirani su svi tipovi podataka, sve poruke i upiti u aplikaciji.

Princip rada

Kada iz CometBFT sloja pristigne poruka, ona se prevede i rutira do x/checkers modula, koji tu poruku obradi i prosledi rezultate nazad consensus engine-u. **Poruke** koje su dostupne x/checkers modulu na obradu, nalaze se u proto/checkers/tx.proto fajlu, i to su: MsgCreateGame, MsgCreateGameResponse, MsgPlayMove, MsgPlayMoveResponse, MsgRejectGame i MsgRejectGameResponse. Za svaku pristiglu poruku, postoji i struktura odgovora modula.

Primer najosnovnijeg flow-a objasnićemo primerom MsgPlayMove poruke, čijim se pristizanjem iniciraju sve neophodne provere deinisane u fajlu x/checkers/types/message_play_move.go. Odatle se inicira poziv metode iz keeper komponente koja jedina ima pravo da menja stanje svog modula, ukoliko nije drugačije

definisano u baseapp-u. U keeper komponenti definisano je na koji način je handle-ovana spomenuta poruka (x/checkers/keeper/msg_server_play_move.go). Na samom kraju metode emituju se neophodni event-i, menjaju se neophodna stanja u store-ovima (memoriji modula kojoj mogu pristupati samo keeper-i), obračunava se trošak transakcije (gas) i odgovor se plasira nazad consensus sloju.

Posmatrajući aplikaciju kao širu celinu sa aspekta blockchain-a, može se zaključiti da se zapravo u store-ovima čuvaju podaci koji predstavljaju podatke iz blokova. Na osnovu svih store-ova u svim modulima aplikacije, formira se trenutno stanje aplikacije, koje se upisujuđe u lanac.

Kada su u pitanju **upiti**, za njih je važno naglasiti da ne menjaju stanje aplikacije, ne zahtevaju postizanje koncenzusa, već samo čitaju iz potrebne informacije iz stanja i vraćaju odgovore, za šta im ne treba ništa više od jednog dostupnog node-a koji te podatke može da obezbedi. Upiti se obrađuju na sličan način kao poruke, zahtev stiže do baseapp-a, koji ga potom rutira do odgovarajućeg modula koji taj upit obrađuje i vraća rezultat.

Upite dostupne našem modulu možemo naći izlistane u fajlu proto/checkers/query.proto. Tu se takođe može videti da za svaki QueryRequest postoji i odgovarajući QueryResponse. Za primer možemo uzeti upit QueryCanPlayMoveRequest koji inicira čitanje potrebnih podataka iz store-a od strane odgovarajućeg keeper-a (x/checkers/keeper/grpc_query_can_play_move.go). Nakon izvršenih provera i popunjavanja strukture za odgovor, rezultat je vraćen iz pozvanog upita.

Frontend

Frontend aplikacije preuzet je sa Github repo-a: https://github.com/nablsi14/react-checkers. Pisan je u React-u, a prilagođen blockchain backend-u korišćenjem CosmJS frameworka uz uputstva sa Interchain akademije. Tipovi i osnovne strukture podataka, kao i poruke i upiti, prilagođeni su komunikaciji sa backendom. Takođe, na frontend-u je integrisan i Keplr wallet koji će koristiti za potpisivanje transakcija.

Scenario rada aplikacije

Kada se korisnici konektuju sa svojim walletima, mogu inicirati početak partije slanjem zahteva na navedenu adresu. Korisnik na specificiranoj adresi ima predviđen vremenski rok za koji može da odgovori na zahtev za partiju. Ukoliko ne odgovori, zahtev će isteći, a ukoliko odbije, partija takođe neće započeti. Ako je korisnik ipak prihvatio partiju, ona će započeti pri čemu će svaki od igrača ostaviti neki ulog u native valuti aplikacije, a u pitanju je takozvana STAKE valuta. To se ostvaruje uz pomoć bank keeper-a koji transferuje sredstva na račun x/checkers modula. Od samog početka svaki potez će inicirati transakciju koja mora biti potpisana od strane korisnika i koja će kao naknadu skinuti određenu količinu tokena. Na kraju igre, igrač koji odnese pobedu,

osvaja oba zaloga koji se do tog momenta čuvaju na x/checkers modulu, a koji se ponovo uz pomoć bank keeper-a, isplaćuju sa računa modula na korisnički.

Testiranje

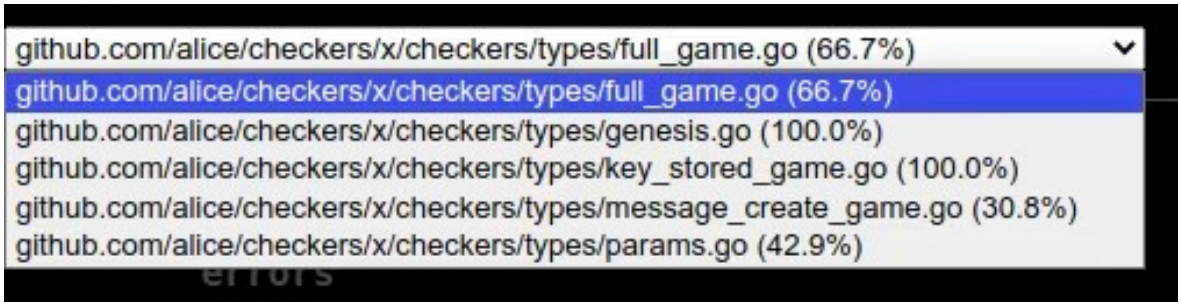
Kada je reč o testiranju, korišćen je osnovni pristup unit testiranja u go programskom jeziku. Testiranje se sprovodi na nivou paketa, pri čemu su fajlovi sa testnim slučajevima smešteni u iste pakete kao i izvršni kod.

```
nikola@nikola-ethernal:~/Ethernal/onboarding/Interchain/checkers$ go test /home/nikola/Ethernal/onboarding/Interchain/checkers/x/checkers/keeper
ok      github.com/alice/checkers/x/checkers/keeper     (cached)
nikola@nikola-ethernal:~/Ethernal/onboarding/Interchain/checkers$ go test /home/nikola/Ethernal/onboarding/Interchain/checkers/x/checkers/types
ok      github.com/alice/checkers/x/checkers/types      (cached)
nikola@nikola-ethernal:~/Ethernal/onboarding/Interchain/checkers$
```

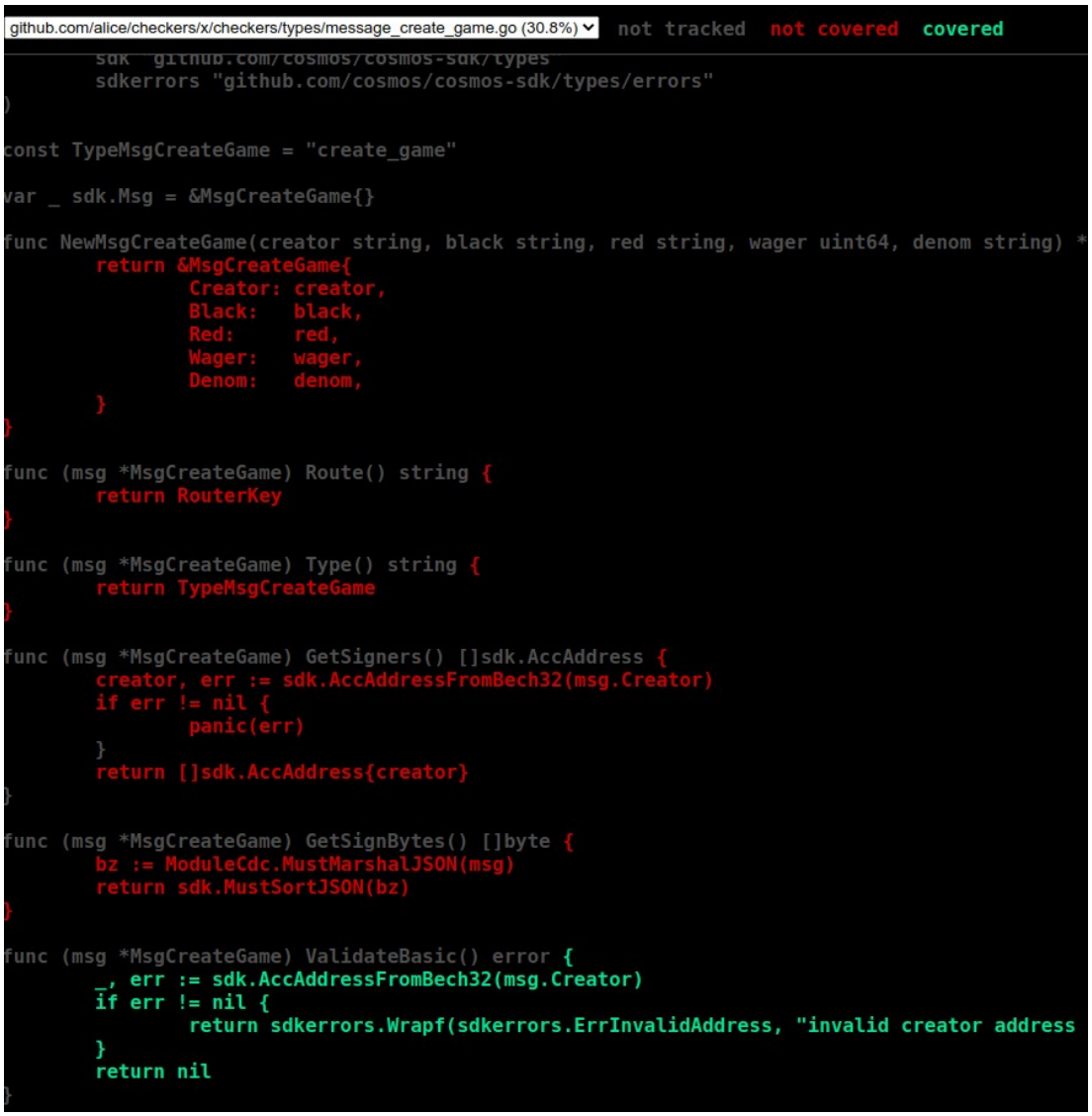
Što se tiče code coverage-a, otvara se pitanje koji scope projekta je relevantan za analizu coverage-a, budući da je veliki deo koda generisan od strane Ignite alata. Takođe, u velikoj meri su prisutni .pb.go i .pb.gw.go fajlovi koje generiše protobuf kompajler iz proto fajlova, tako da kada pokrenemo analizu coverage-a na dva paketa koja implementiraju skoro celu logiku projekta dobijemo sledeće rezultate:

```
nikola@nikola-ethernal:~/Ethernal/onboarding/Interchain/checkers$ go test -coverprofile keeperCoverage /home/nikola/Ethernal/onboarding/Interchain/checkers/
x/checkers/keeper
ok      github.com/alice/checkers/x/checkers/keeper     0.030s  coverage: 91.9% of statements
nikola@nikola-ethernal:~/Ethernal/onboarding/Interchain/checkers$ go test -coverprofile typesCoverage /home/nikola/Ethernal/onboarding/Interchain/checkers/x
/checkers/types
ok      github.com/alice/checkers/x/checkers/types      0.026s  coverage: 2.0% of statements
nikola@nikola-ethernal:~/Ethernal/onboarding/Interchain/checkers$
```

Nakon izbacivanja pomenutih fajlova, code coverage za types paket izgleda ovako:



Problem prave još fajlovi sa manjim coverage-om, koji su takođe parcijalno generisani od strane Ignite-a:



Semgrep

Projekat je takođe skeniran alatom za statičku analizu po imenu Semgrep, koji je uključen u CI na Github-u. Semgrep funkcioniše tako što koristi bazu šablona nekih ustaljenih greški koje se prave prilikom programiranja. Za potrebe projekta napisan je određeni set šablona (rules.yaml) koji su se proveravali pri commit-ima i u zakazanom terminu (može se videti u github akcijama). Fajlovi koji su ignorisani u proveri navedeni su u .semgrepignore fajlu. Takođe, moguće je koristiti i public registry sa njihovog oficijalnog sajta. Tamo su dostupni šabloni pisani za različite programske jezike, među kojima je i Golang.

Organizacija rešenja

Za implementaciju rešenja korišćeno je okruženje VisualStudio Code, programski jezik Go, zatim CosmosSDK i CosmJS framework, kao i React na frontend-u. Rešenje je raslojeno na dva repozitorijuma, pri čemu je UI predstavljen kao *submodule* glavnog repozitorijuma.

Mogućnost nadogradnje

Prikazani projekat dalje je moguće nadograditi dodatnim modulima. Primer dobrog koraka napred bio bi modul koji omogućava interchain komunikaciju i kreiranje neke vrste tabele rezultata koja bi bila dostupna korisnicima različitih chain-ova, kao i igranje partija između njih.