Pametni ugovori pomoću blockchain tehnologije

Maček, Filip

Pametni ugovori pomoću blockchain tehnologije

Maček, Filip

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET
MATEMATIČKI ODSJEK

Filip Maček

PAMETNI UGOVORI POMOĆU BLOCKCHAIN TEHNOLOGIJE

Diplomski rad

Voditelj rada:
prof.dr.sc. Luka Grubišić

Zagreb, kolovoz 2020.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Zahvaljujem svojoj majci Deani, ocu Vladimiru, sestrama Lani i Dori, djedu Ivanu , baki Ankici i svim svojim prijateljima za svu potporu i pomoć koju su mi pružili. Također posebne zahvale mentoru prof.dr.sc. Luki Grubišiću što me dao priliku izraditi ovaj diplomski rad.

Sadržaj

Uvod

Pametni ugovori naziv su za bilo kakav kompjuterski program ili transakcijski protokol koji automatski izvršava i kontrolira njemu specificirane programske logike i njegove dijelove. Time je omogućeno da se od različitog spektra današnjih tradicionalnih ugovora može kreirati digitalna manifestacija tog ugovora i njegovih uvjeta. Glavna ideja pametnih ugovora je da se oni izvršavaju na decentraliziranoj arhitekturi kako bi se izbjegao problem centralnog izvršavanja ili posrednika. Tu se prometnula tehnologija *blockchain* kao jedan od mogućih medija za decentralizirano izvršavanje pametnih ugovora, koji ćemo i mi koristiti.

U ovom diplomskog radu pokušat ćemo pokazati jednu od mogućih implementacija i korisnost pametnih ugovora u stvarnom svijetu. Demonstrirat ćemo kako možemo komunicirati sa pametnim ugovorom na *blockchainu* pomoću posebnog tipa web aplikacija imena *DApp*, aludirajući da aplikacija za svoju bazu podataka i komunikaciju koristi pametni ugovor kreiran na decentraliziranoj mreži *blockchain*. Što je u suprotnosti sa današnjim modelom web aplikacija gdje se komunicira pomoću informacijskog sučelja *API-ja* sa svojom bazom podataka koja je privatno održavana tj. centralno postavljena.

Također cilj ovog diplomskog rada je pokazati kako takve pametne ugovore možemo spojiti sa vanjskim svijetom i na siguran način dostaviti informacije izvan *blockchaina* na naš pametni ugovor. To je inače problem jer *blockchain* kao zatvorena cjelina sa svojim konsenzus mehanizmom ne može na siguran i decentraliziran način dohvatiti te podatke bez da ne kompromitira svoj protokol.

Taj problem je moguće riješiti uz pomoć *Chainlink Oracle* mreže koja je zadužena za dotok vanjskih informacija na blockchain i izvršavanje bilo kakvih radnji i operacija koje pametni ugovor zahtijeva u vanjskom svijetu izvan *blockchaina*. Pomoću dva naša *Chainlink* čvora pokazati ćemo kako je naš pametni ugovor spojen sa vanjskim svijetom te na koji način pametni ugovor dobiva potrebne informacije pomoću *Chainlink* protokola.
Osmislili smo i mali ekosustav i pripadajući infrastrukturu kako bi opisali jedan pametni ugovor koji može služiti kao decentraliziran sustav dostave pošiljaka ili bilo kakve druga funkcionalnost koja zahtijeva prelazak od početne do krajnje točke, a izvršavaju ju registrirani korisnici.

Da bi to pametni ugovor na *blockchainu* mogao provjeriti je li korisnik stvarno prešao taj put, potrebne su mu lokacijski podaci prilikom izvršavanja te rute. To smo riješili tako da smo napravili i mobilnu Android aplikaciju koja će registriranim korisnicima služiti za prelazak ruta, a pametnom ugovoru kao izvor lokacijski podataka koje će se slati na obradu na *Chainlink* čvorove.

Poglavlje 1

Uvod u blockchain

1.1 Uvod

Kroz ovo poglavlje ukratko ćemo opisati glavne karakteristike i dizajn *blockchaina* kao i njegovu povijest. Također značajna će nam biti i njegova motivacija i kakva dobra svojstva *blockchain* donosi.

1.2 Povijest

Nakon izuma računala 1980-tih ljudi su se počeli pitati kakve su sve još inovacije moguće koje bi ima donijele veću produktivnost i efikasnost. Tada su još to bila izolirana računala koja su slabo bila povezana i nisu znala kako međusobno komunicirati. Ali i već tada su neki istaknuti stručnjaci počeli uviđati kako bi se ta računala mogla povezati u jedan koherentan sustav ili mrežu. *David Chaum* je bio pionir u tom području, i on je već 1982. u svojoj doktorskoj disertaciji [1] počeo razmišljati kako organizacije i kompanije, koje međusobno ne vjeruju jedna drugoj, mogu zajedničkim snagama izgraditi i održavati siguran računalni sistem koji svi sudionici mogu koristiti vjerovati. Uvidio je veliku potrebu u privatnom i javnom sektoru za takve sisteme. Uvidio je da bi kriptografskim tehnikama bilo bi moguće takav sistem ostvariti i učiniti praktičnim, gdje bi se spremljeni podaci u opticaju mogli zaštititi mehanizmom sefa (*eng. vault*).

Kasnije, 1991. *Stuart Haber i W. Scot. Sornetta* iznose prvi dizajn kriptografski osiguranog lanca blokova u kojem se ne može manipulirati vremenskim oznakama. Godinu dana nakon u svoj dizajn unose novu inovaciju binarno hash stablo (*eng. Merkle Tree).* Ono je omogućilo jednostavnu i sigurnu verifikaciju sadržaja velikih struktura podataka. Također je omogućilo da više certifikata dokumenata budu uključeni u blokove.

Bitcoin

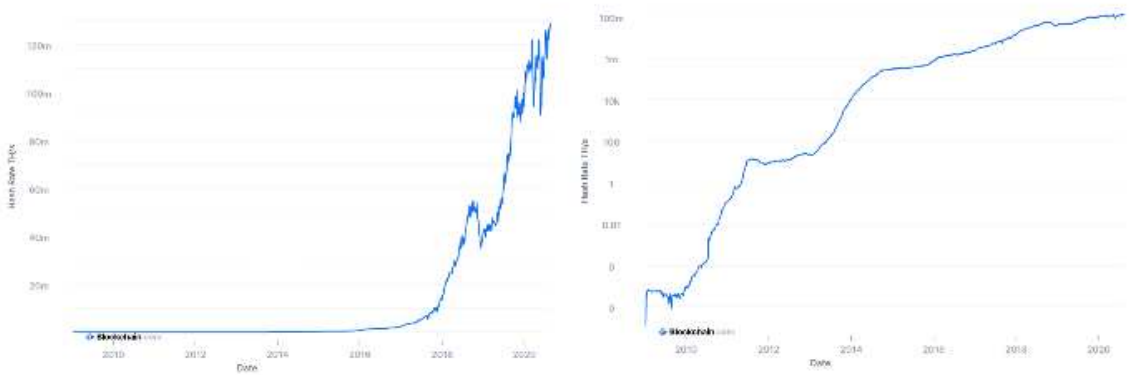
Iako je bilo mnoštvo akademski radove i napretka u kriptografiji do 2008. nije postojala nikakva veća i značajnija implementacija *blockchain* u svijetu. Tada je u je financijske krize skupina ljudi ili jedan čovjek (što do danas nije poznato) pod pseudonimom Satoshi Nakamoto objavljuje članak
eng. Bitcoin: A Peer-to-Peer Electronic Cash System. [2] U njemu iznosi osnovne teze i potrebe za neovisnim i decentraliziranim novčanim sustavom:

- Potpuno neovisan elektronički novac bi omogućio direktno slanje novca od jednog sudionika do drugog bez prisustva neke financijske institucija.
- Takav sustav koji se temelji na povjerenju u jedan centralni entitet ima svoje nedostatke. Neke od njih su mog
- Mogućnost poništavanja transakcija i blokiranja računa sudionika te time izgon iz financijskog sustava i nemogućnosti sudjelovanja u financijskim uslugama.
- Transakcijski troškovi su povećani prisustvom posredovatelja

U tom članku je preporučeno elektronički naplatni sustav koji se temelji na kriptografiji, a ne na povjerenju. Dopuštajući da bilo koja dva sudionika sudjeluju u financijskoj transakciji bez potrebe da vjeruju jedan drugome.

Problem dvostruke potrošnje (*eng. double-spending problem*) već potrošenog novca riješen je na način da se umrežena računala na Bitcoin mreži slože oko toga da postoji jedan opći vremenski lanac gdje bi se generirao kriptografski dokaz o kronološkom poretku transakcija.

Takav sustav je siguran dok većina pošteni čvorova na *Bitcoin* mreži kontrolira više procesorske moći nego napadačka skupina čvorova. Tvorac Satoshi Nakamoto je svoj sistem konstruirao na idejama jednostavnost i efikasnost da bi se postigla funkcionalnost koja mu je namijenjena. Komentirao je svojom poznatom rečenicom da *je takav sustav robustan u svojoj nestrukturiranoj jednostavnost* .



Slika 1.1: Hash rate *Bitcoin* mreže kroz vrijeme (linearna, logaritamska skala)

Iako u početku odbačen i ne hvaljen od strane akademika i financijskih institucija koji su u njemu vidjeli pokušaj promijene starih vrijednosti, mreža je aktivna i rastuća, te bez prestanka niti bilo kakvih smetnji pouzdano izvršava svoju glavnu zadaću. Popularnost i snagu mreže najviše možemo vidjeti na slici 1.1. gdje je prikazana *eng. hash rate* što predstavlja koliko puta cijela mreža može primijeniti SHA256 funkciju ma blok da bi dobila zadanu hash vrijednost. Ta mjera zapravo označava količinu procesorske moći Bitcoin protokola i svih računala na njemu. Ona je u konstantom porastu što govori o tome da se nova računala stalno priključuju na mreži i time ju čine čvršćom i sigurnijom.

Šest godina kasnije, 2014. godine izlazi članak i dizajn novog tipa *blockchaina* po imenu ***Ethereum***. On je naslijedio opću funkcionalnost slanja novčane valute od *Bitcoin* protokola, ali je u svojoj implementaciji dodao još jednu jako važno novinu. To je da čvorovi na *Ethereum* mreži još mogu izvršavati posebno napisanu programsku logiku u računalnom jeziku nazvanom *Solidity*. Time je otvoren put implementaciji pametnih ugovora te njihovoj širokoj upotrebi. Zapravo je i svaki takav programski isječak na *Ethereumu* i prozvan *eng. Smart Contract* .

Mi ćemo budućim poglavljima ćemo pobliže opisati unutrašnji dizajn Ethreumu, osnovne značajke programskog jezika *Solidity* te kako je moguće da se programski kod paralelno izvršava na svim računalima *Ethereum* mreže.

1.3 Osnovne karakteristike

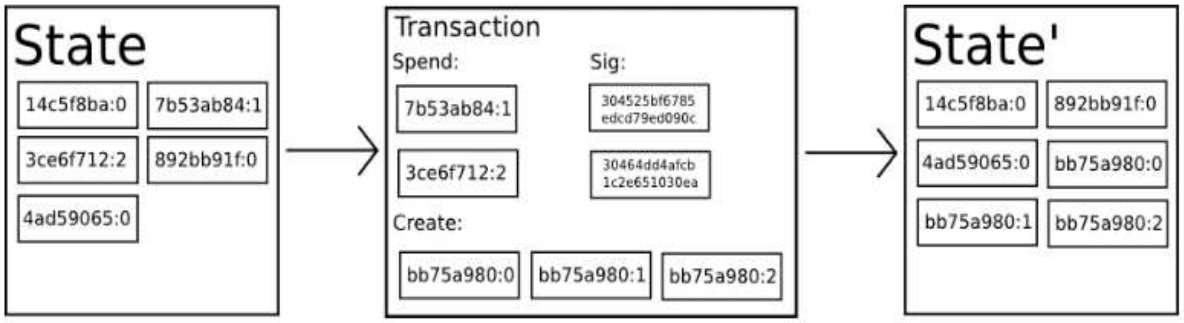
U ovom poglavlju ćemo ukratko opisati glavne dijelove i karakteristike *blockchaina* fokusirajući se na inicijalni dizajn *Bitcoin* protokola.

Njegov doprinos i značaj je ogroman jer je istovremeno riješio dva velika problema:

1. Omogućio je jednostavan i efektivan konsenzus algoritam, dajući računalima *(eng. nodes)* na mreži svojstvo da se kolektivno slože oko trenutačnog stanje *Bitcoinove* knjige salda *(eng. ledger)* te mehanizma kako će se ona mijenjati.
2. Riješio je veliki implementacijski problem tko odlučuje o konsenzusu na mreži, i time dao mogućnost jednostavnog ulaska i sudjelovanja novih računala na mreži.

Sa tehničkog stajališta, knjiga salda elektronske valute Bitcoin , je zapravo sistem promijene stanja *(eng. „state transistion system")* gdje se stanje vlasništvo na elektronskom valutom te funkcija promijene stanja *(eng. „state transition function")* prijenos prava na Bitcoin valutu sa jednog sudionika na drugog

.



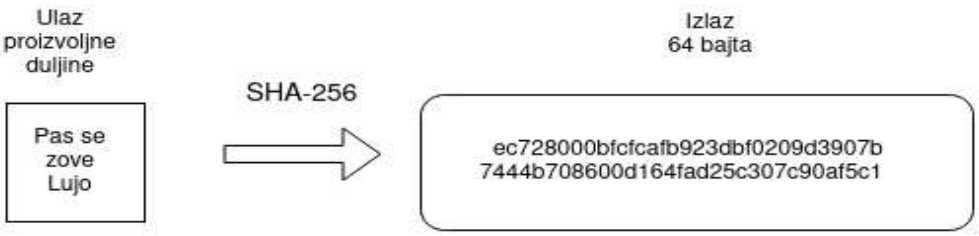
Slika 1.2: *Blockchain* kao sustav promijene stanja

Ovakav sustav ima puno različitih i kompleksnih dijelova, ali mi ćemo se fokusirati na one najbitnije i ukratko opisati njihovu svrhu i povezanost da ostalim dijelovima.

Kriptografska hash funkcija

U pozadini *blockchaina* je posebna klasa kriptografske funkcije koji ima svojstvo da je ulazni nezavisna varijabla proizvoljne veličine, a rezultat te funkcije je fiksne veličine. Takva funkcija je jednosmjerna te joj se ne može izračunati pripadajuća inverzna funkcija.

Time je jedini mogući algoritam za pronalaska ulaznog parametra za zadanu vrijednost funkciji , onaj u kojem se *eng. brute-force* algoritmom provjeravaju sve vrijednosti. U ovom slučaju koristi je posebna hash funkcija imenom SHA256 koja vraća 256 bitni broj koji je prikazan kao heksadecimalni broj sa 64 znamenke rade lakše čitljivosti.



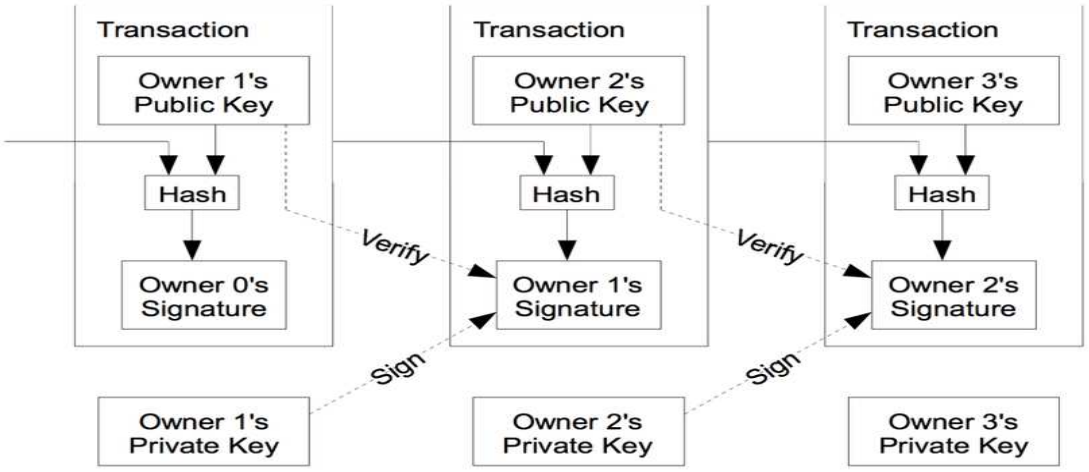
Slika 1.3 SHA-256 hash funkcija

Transakcije

Na *blockchainu* elektronska valuta se definira kao lanac digitalnih kriptografski potpisa. Svaki sudionika transferira željenu količinu digitalnog novca koju on posjeduje tako da pomoću svog privatnog ključa potpisuje skup koji sadrži sljedeće informacije:

- Hash posljednje vlastite transakcije
- Javnih ključ primatelja te digitalne transakcije

Transakcije nisu privatne, već javne. Time je omogućeno da se svaka transakcija vidi i provjeri je li ispravna uz pomoć jednostavne verifikacije javnim ključem.



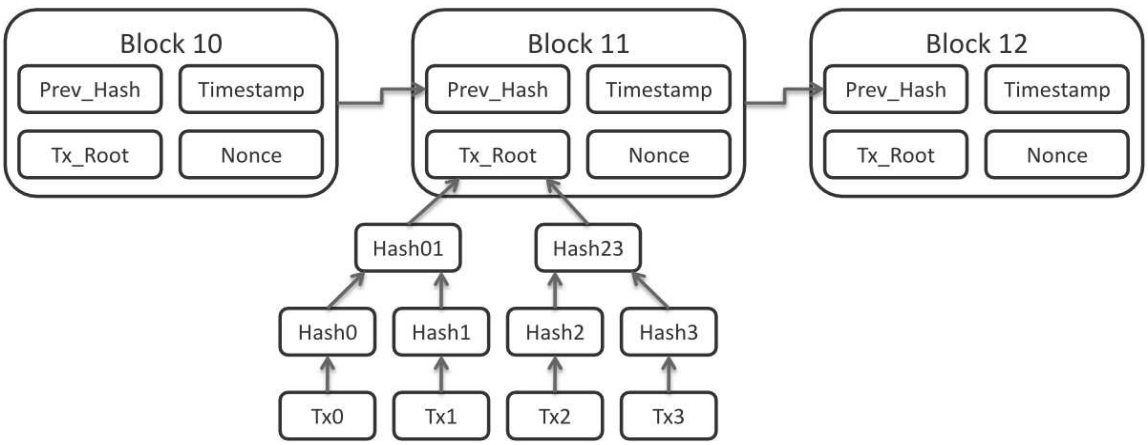
Slika 1.4: Dizajn Bitcoin transakcije

Ovim dizajnom se postiže da digitalna valuta nije fiksni iznos koji svaki sudionika ima (vrijednost na računu), već se sistem temelji na prijenosu prava sa prijašnjeg vlasnika na budućeg koji onda ima pravo potpisati i prenijeti ta prava gdje god on želi.

Blokovi

U prijašnjem objašnjenu o transakcijama na blockchainu, postoji problem dvostruke potrošnje *(eng. double-spending)* koji govori da primatelj digitalne valute ne može sa apsolutnom sigurnošću ustvrditi da pošiljatelj već negdje druge nije potrošio tu količinu valute. Jedan od rješenja tog problema bi bio stvaranjem centralnog entiteta kojem bi svi vjerovali, te bi on provjeravao da transakciju već nisu negdje drugdje potrošene. Ali time bi se potkopala cijela ideja decentralizacije i neovisnosti.

Umjesto toga implementiran je dizajn lanca blokova gdje se na sve elemente bloka primjenjuje hash funkcija te se vremenski obilježe te se propagiraju kroz cijelu mrežu.



Slika 1.5: Povezivanje blokova u lanac

Najbitniji dijelovi bloka su hash prethodnog bloka, vremenska oznaka *(engl. timestamp)* koja predstavlja vrijeme kada je blok dodan u lanac, korijen binarnog stabla *(engl. Tx_Root)* svih nepotvrđenih transakcija te broj imena *eng. nonce* pomoću kojeg se rješava algoritam za uključivanje bloka u lanac.

Algoritam konsenzusa

Da bi se uspio implementirati ovakav distribuiran vremenski lanac blokova na većem broju računala *(eng. peer-to-peer)*, potrebno je utvrditi mehanizam kako će se sva računala koja održavaju mrežu složiti oko toga koji je glavni lanac i kako se on povezuje. Implementiran je algoritam imena eng. **Proof-of-Work.**

On je prenio moć odlučivanje sa formalnog demokratskog odlučivanja, gdje je jedinstveno registriran entitet ima pravo na jedan glas (*eng. one-IP-address-one-vote*) na ekonomsku barijeru, gdje je moć jednog računala na mreži u direktnoj proporciji na procesorsku snagu koju on donosi na mrežu (*eng. one-CPU-one-vote).*

Time je riješeno pitanje odabira glavnog lanca oko kojeg će se većina računala na mreži složiti, a to je onaj koji je najduži. Na njegovo stvaranje je potrošeno najviše procesorske moći, i uz njegov odabir kao glavnog lanca postiže se i najveća sigurnost protokola. Ako je većinom procesorske snage upravljaju poštena računala na mreži, tada će taj njihov lanac rasti najviše i premašiti ostale po veličini.

Taj algoritam je implementiran tako da se uvećava mjera *engl. nonce* u zaglavlju bloka dok hash tog bloka ne bude manji od unaprijed zadane složenosti. Time se želio postići da je prosječno vrijeme izrade bloka *(eng. mining)* oko 10 minuta.

Kako se mijenja procesorska moći cijele mreže, tako se i mijenja razina složenosti da bi se postiglo prosječno vrijeme izrade bloka. Mreža svakih 2016 blokova provjerava koliko je bilo prosječno vrijeme izrade blokova. Ako je manje od ciljanje 10-minutne, onda se složenost povećava tj. hash bloka će morati biti još manje tj. morat će imati više vodećih nula.

Scripting

U pozadini cijelog protokola je i jednostavan programski jezik na stogu *(eng. stack)* koji služi kao „scripting" sustav za transakcije. „Script" je zapravo lista instrukcija upisana u svaku transakciju koji opisuje kako će sljedeća osoba dobiti pristup svojem elektroničkom novcu. Provjera i izvršavanje ovog jednostavnog jezika je povjerena čvorovima na mreži, koji je izvršavaju i upisuju ispravne transakcije u novi blok, tj. transakcije u kojima se skripta uspješno izvršila. Koliko god ovaj sistem bio jednostavan i efikasan u održavanju Bitcoin mreže i protokola, on ima i neke svoje nedostatke:

- **Izostanak Turingove konačnosti**

  Iako sadrži veliki skup operacija koje se mogu izvršiti, taj programski jezik ne podržava petlje. To je implementirano sa svrhom da se izbjegnu beskonačno izvršavanje petlji dok se verificiraju transakcije.

- **Nemogućnost pamćenja vrijednosti**

  Ne postoji mogućnost da se tim transakcijskim skriptama da veća kontrola nad isplatama i iznosima transakcija. Time se efektivno ne mogu izvršavati nikakve složenije transakcije koji imaju neke uvjete.

- **Nepostojanje stanja**

  Sve nepotrošene transakcijske vrijednosti (*eng UTXO – Unspent transaction output)* su ili potrošeni ili nepotrošeni. Nema nikakve šanse da one ulaze u neke višestruke ugovore ili skripte koji čuvaju unutrašnje stanje.
- **Blockchain sljepoća**

  UTXO ostaju slijepe na ostale podatke u *blockchainu* kao što su *nonce*, vrijeme ili prijašnjih blok hash. Time je scripting programski jezik višestruko limitiran i nedostaju mu važni izvori slučajnosti.

1.3 Ethereum

Zbog sve već navedenih nedostataka, pojavila se potreba za drugačijom i sofisticiranijim tipom *blockchaina* koji bi uspijevao izvršavati i kompliciraniju programsku logiku nego samo slanje elektroničke valute. U tom cilju razvio se *Ethereum* [3] koji je imao cilj sagraditi alternativni tip *blockchaina* koji ne kompromitira ekonomski model i sigurnost koju nudi *blockchain*, a nudi mogućnost izvršavanja kompleksne programske logike obuhvaćene pametnim ugovorom.

Implementiran je na ovim idejama:

- **Jednostavnost**

  Ethereumov protokol mora biti što jednostavniji za korištenje i interakciju sa njim. Svaka optimizacija koja dodaje neku količinu kompleksnosti trebala bi biti izbjegnuta jedino ako daje vrlo veliku korisnost.
- **Univerzalnost**

  Fundamentalni dizajn *Ethereuma* je da nudi potpunu slobodu pri izradi aplikacija time što nudi interni Turing-konačan jezik imena *Solidity* što nudi programeru opću slobodu u izradi svakog tipa pametnog ugovora i svih operacija koje je moguće matematički izvršiti.
- **Modularnost**

  Dijelove i implementacije *Ethereum* protokola trebali bi biti modularni i neovisni te bi morali biti implementirani kao zasebni moduli koje i drugi protokoli mogu koristiti.
- **Svestranost**

  Detalji protokola nisu fiksirani, i svako poboljšanje koje pridonosi efikasnijem i sigurnijem radu su dobrodošli i bit će detaljno razmotreni.
- **Bez cenzure i diskriminacije**

  Protokol ne bi trebao nikako cenzurirati ili diskriminirati specifične kategorije upotrebe ili sudionike

Tip računa

Na *Ethereum* protokolu stanje sustava je prikazano uz pomoć objekata pod imenom *eng. accounts*. Svaki od njih ima svoju osobnu 20-bajtnu adresu, a funkcija promijene stanja *(eng. state transition function)* između njih je direktan transfer vrijednosti i informacija. Digitalna valuta na Ethereumu je Ether(ETH).

Postoje dva tipa računa na Ethereum protokolu

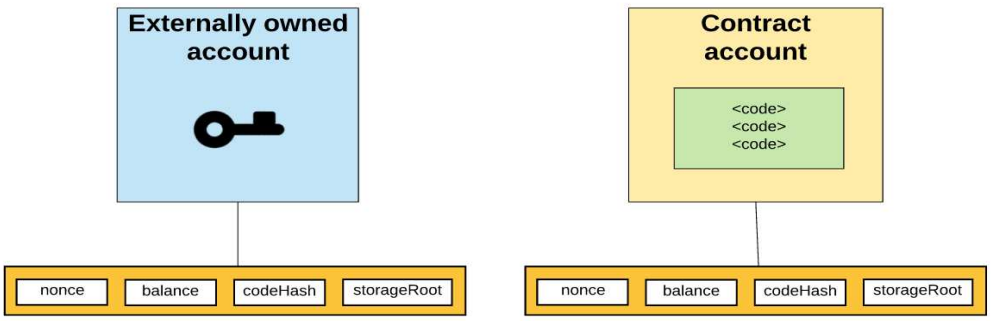
1. **Externally owned accounts** – kontrolirani uz privatni ključ
2. **Contract Accounts** – kontroliran od strane koda pametnog ugovora

Također svaki račun na Etheremu sadrži 4 važna polja:

- **Nonce** : brojač, koji služi da transakcije budu procesirana samo jednom
- **Stanje računa**: količina digitalne valute *Ether* koju račun posjeduje
- **Kod programa**: ako postoji
- **Memorija računa**: početna vrijednost je prazna

Razlika u ta dva tipa računa je također u tome što *externally owned account* nema nikakvog programskog koda. S toga računa se mogu slati i primati poruke potpisivanjem transakcija. S druge strane, *contract account* ima drugačija svojstva te on na svaku primljenu poruku aktivira određeni dio koda, koja onda izvršava sve potrebne operacije npr. zapisivanja u internu memoriju, slanje poruka ostalim računima ili kreiranje novih pametnih ugovora.

Takvim dizajnom se postiže to da pametni ugovora na *Ethereumu* ne bi trebali biti viđeni kao fiksne strukture koje mora biti izvršena ili s kojom je potrebno surađivati. Već više kao autonomne agente koji žive unutar *Ethereum* protokola i čekaju na poruke ili transakcije, te imaju direktnu kontrolu nad stanjem računa i svim unutrašnjim varijablama.



Slika 1.7: Tip računa na *Ethereum* protokolu

Transakcije i poruke

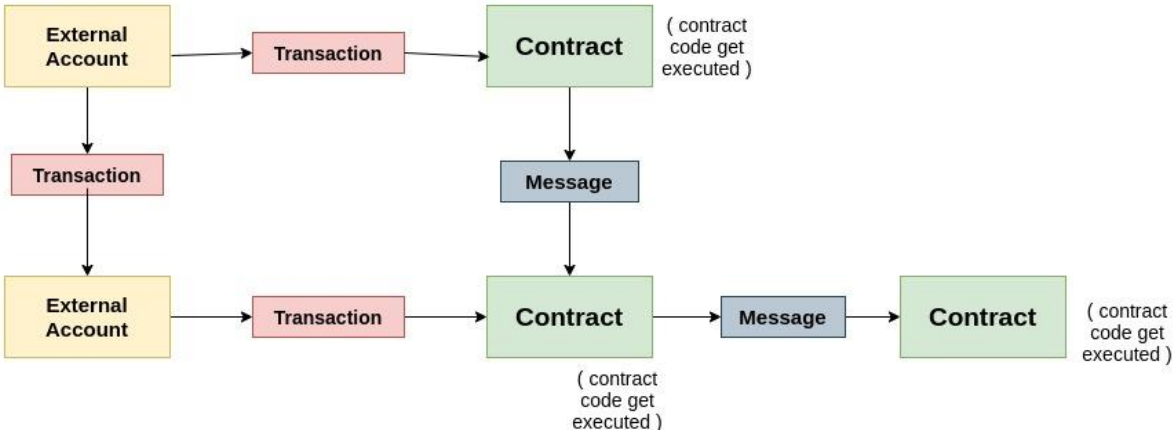
Uz različite tipova računa, na *Ethereum blockchainu* postoje i dva tipa poruku koje sudionici ili pametni ugovori mogu koristiti. To su:

1. **Transakcije**

   Skup podataka potpisan od strane vanjskog agenta (*eng. Exernal Actor*). Može reprezentirati poruku ili novi autonomni objekt, tj. novostvoreni pametni ugovor. Transakcije su eksplicitno zapisane na *blockchain*.

2. **Poruke (eng. Messages)**

   Pametni ugovori imaju mogućnost slati poruke jedan drugome. Poruke su u tom slučaju virtualni objekti koji nisu direktno upisani u blockchain već postoje jedino u *Ethreumovom* unutrašnjem okruženju *(eng.Ethereum execution environment)* i tamo se izvršavaju. Ukratko poruka je kao transakcija, ali je napravljena od strane pametnog ugovora, a ne vanjskog sudionika.



Slika 1.8: Poruke i transakcije

Svaka transakcije upisana u blok sadrži poruku, ali poruke koje šalju pametni ugovori nisu upisani u blok kao podaci, već su implicitno uključene u izvršavanje pametnog ugovora.

Formalno, kod pametnih ugovora je napisan u program jeziku niže razine te pretvoren u *eng. bytecode* koji onda *Ethereumov* virtualni kompjuter *(eng. Ethereum virtual machine -EVM)* izvršava dio po dio uz pomoć stoga.

Value	Mnemonic	δ	α	Description
0x00	STOP	0	0	Halts execution.
0x01	ADD	2	1	Addition operation. $\mu'_s[0] \equiv \mu_s[0] + \mu_s[1]$
0x02	MUL	2	1	Multiplication operation. $\mu'_s[0] \equiv \mu_s[0] \times \mu_s[1]$
0x03	SUB	2	1	Subtraction operation. $\mu'_s[0] \equiv \mu_s[0] - \mu_s[1]$
0x04	DIV	2	1	Integer division operation. $\mu'_s[0] \equiv \begin{cases} 0 & \text{if } \mu_s[1] = 0 \\ \lfloor \mu_s[0] \div \mu_s[1] \rfloor & \text{otherwise} \end{cases}$

Slika 1.9: Primjer nekih operacija i njihovih kodnih vrijednost

Taj programski jezik niže razine i sve njegove operacije pridonose raznolikosti operacija koje je moguće uključiti u svoj pametni ugovor.

Transakcijski troškovi

Osnovni jedinični računalni korak naziva se *eng. GAS* te svaka *bytdecode* operacija na EVM sadrži neku količinu koraka za provođenje, i oni su izraženi u količini *GAS* mjerne jedinice potrebnih za njegovo izvršavanje. U svakoj transakciji postoje dva polja koja pobliže označavaju koliko najviše računalnih koraka je pošiljatelj transakcije spreman platiti i po kojoj jediničnoj cijeni za osnovnu operaciju izraženu u GAS mjernoj jedinici:

1. **STARTGAS**

   Najveći broj računalnih koraka kojih je moguće izvršiti

2. **GASPRICE**

   Naknada koju pošiljatelj transakcije plaća po računalnom koraku *(eng. GAS)*

Ovim ekonomski mehanizmom postiže se korisnost koji računala koja održavaju *Ethereum blockchainu* dobivaju za održavanje mreže i protokola.

U analogiji s automobilom, GAS predstavlja količinu goriva koja je potrebna, a GASPRICE jediničnu cijenu po litri, a umnožak oba je zapravo ukupni trošak.

Glavni problem pri izvršavanju transakcija i uključivanju u blokove jer u tome što računala na *Ethereum* mreži koja rudare blokove ne mogu unaprijed, prije izvršavanje koda, znati hoće li ih biti u mogućnosti izvršiti za zadane resurse koji su primili u transakcije. To je poznati problem u računalnoj znanosti (*eng. halting problem*), i on se može riješiti. Računala koje izvršavaju tu transakciju će ukoliko ne uspiju izvršiti sve njene dijelove uz zadane resurse koji su primili od pošiljatelja ove transakcije, tu transakciju prekinuti te uzeti svu količinu valute *Ether* kao naplatu za izvršavanje, te podnijeti poseban tip iznimke imena *Out of Gas Exception.*

Poglavlje 2

Pametni ugovori

2.1 Uvod

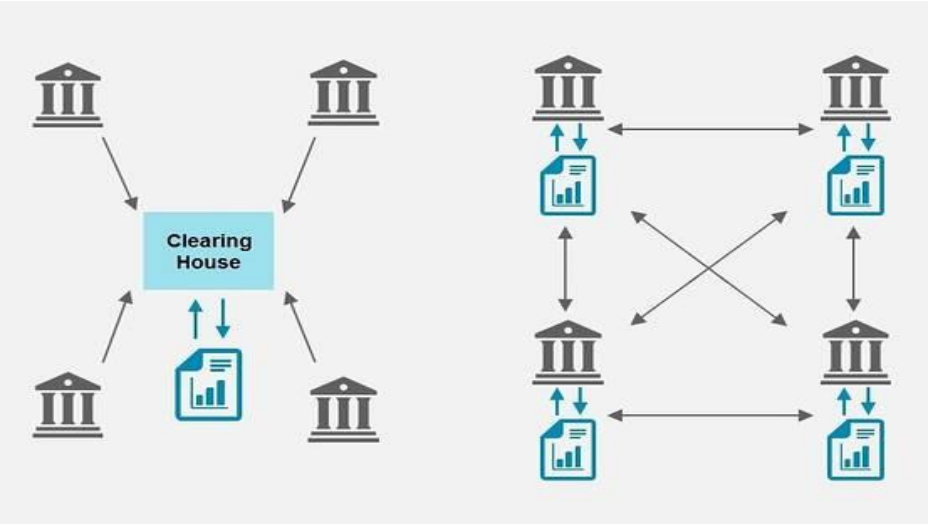
Pametni ugovori naziv su za bilo kakav kompjutorski program ili transakcijski protokol koji automatski izvršava i kontrolira izvršavanje njemu specificirane programske logike. Time je omogućeno da se od različitog spektra današnjih tradicionalnih ugovora može kreirati digitalna manifestacija tog ugovora i njegovih uvjeta. Ideja je da su uvjeti ugovora kao i njegova etape izvršavanja potpuno prepušteni kompjutorskom kodu.

Glavna korisnost pametnih ugovora je u tome što se oni ne ovise o nikakvim sigurnim posrednicima ili arbitražama. Također ne uvjetuju i nikakve troškove koje takvo posredstvo nosi. Svi ostali problemi i prevare koji mogu nastati u izvršavanju ugovora također su malo vjerojatni jer se podrazumijeva da je kod tog ugovora napravljen u cilju obuhvaćanja svih mogućih događaja u stvarnom svijetu koje se tiču tog ugovora. To podrazumijeva jako dobro planiranje tog ugovora, njegovu implementaciju i testiranje.

Još jedan važna odluka kod pametnih ugovora jest gdje će se taj program tj. ugovor izvršavati. Postoje više solucija ovog problema:

1. Pametni ugovor se izvršava na računalnoj infrastrukturi jednog od sudionika (centralizirani model)
2. Neovisan treći promatrač i njegova infrastruktura je zadužena za izvršavanje pametnog ugovora (centraliziran model)
3. Pametni ugovor se izvršava na decentraliziranoj infrastrukturi, najčešće *blockchain* (decentralizirani model)

Blockchain ,posebno *Ethereum*, se nametnuo kao idealno mjesto za decentralizirano izvršavanje pametnih ugovora radi svoje sigurnost i neovisnost. Nijedan sudionik ne može prekinuti ili zabraniti izvršavanje tog ugovora.



Slika 2.1: Centralizirani i decentralizirani model pametnih ugovora

Takvi pametni ugovori koji se izvršavaju na *blockchainu* nude mnoge druge pogodnosti nego današnji digitalni ugovori. Neki su od njih:

- **Sigurnost**

  Izvršavanje pametnog ugovora na decentraliziranoj infrastrukturi osigurava da ne ovisimo samo o jednom računalu i njegovom stanju prilikom izvršavanja ugovora. Također nema nikakvog centralnog entiteta koji će odlučivati što koji sudionik ili pametni ugovor smije raditi.

- **Pouzdanost**

  Procesiranje i izvršavanje našeg pametnog ugovora povjereno je mnoštvu računala na mreži sto daje veliku sigurnost da će se naš ugovor sigurno izvršiti.

- **Ravnopravnost**

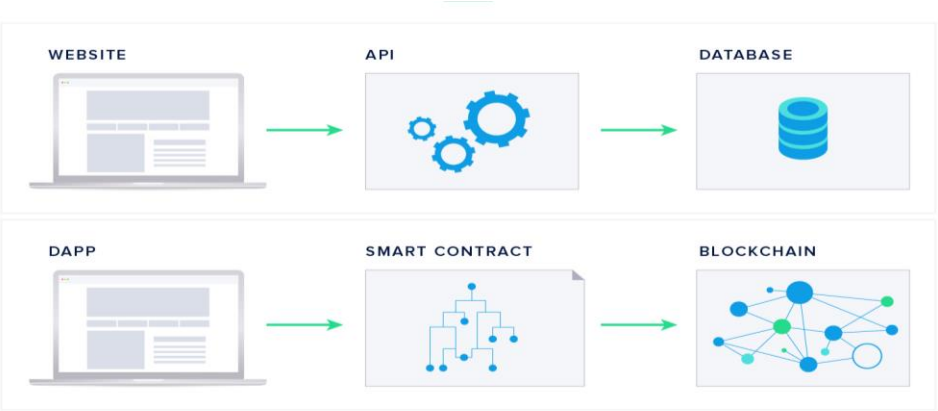
  Koristeći decentralizirano mrežu ravnopravnih partnera svi sudionici imaju jednaka prava i smanje se mogućnost manipulacije centralnih profitnih posrednika ili zlonamjernih sudionika.

- **Efikasnost**

  Automatizacijom procesiranja ugovora u pozadini i izostajanje papirologije pridonosimo većoj efikasnosti gdje nijedan sudionik ne mora čekati ni ručno unositi podatke.

Pametni ugovor na *blockchainu* bismo mogli usporediti sa tradicionalnom web arhitekturom gdje aplikacija komunicira sa svojim centralnim serverom i bazom podataka pomoću API-ja *(engl. Application Programming Interface)*. Vlasnik te infrastrukture ima potpuno kontrolu i privatnost, nitko ne autoriziran ne može pristupiti niti promijeniti podatke.S druge strane, aplikacije na *blockchainu* imaju svoj API u obliku pametnog ugovora, te *blockchain* kao bazu podataka. Time su te aplikacije decentralizirane te imaju svoj specijalni naziv imena ***DApp*** da bi se razlikovale od tradicionalne web arhitekture.

Tada se ta web aplikacija spaja preko *blockchain* klijenta sa pametnim ugovorom, i sva komunikacija ide preko njega.



Slika 2.2: Usporedba tradicionalna web aplikacije i aplikacija na blockchainu

2.2 Oracle problem

Pametni ugovori *blockchainu* imaju prilike revolucionizirati mnoge sektore gdje bi zamijenili današnju potrebu za tradicionalnim legalnim ugovorima i centralnom kontrolom takvih ugovora. Ali zbog njihovog konsenzus mehanizma, *blockchain* ne nudi podršku za jednostavno i prirodno komuniciranje sa vanjskim svijetom već je ograničen na svoj unutrašnji sustav. Ta izolacija *blockchaina* je upravo ono što ga čini sigurnim i pouzdanim medijem gdje se pametnim ugovori mogu izvršavati. Ali kako bi ti pametni ugovori mogli iskoristiti sav svoj potencijal, oni nekako moraju biti povezani sa vanjskim svijetom.

Slika 2.2: Problem povezanosti blockchaina sa vanjskim svijetom

Neki od primjera takvih financijskih ugovora su:

- Financijski pametni ugovori koji trebaju podatke o cijenama financijskih instrumenta kako bi namirili ugovor i odredili količinu naplate.
- Pametni ugovori o osiguranju koje trebaju podatke iz senzora i udaljenih baza podataka da bi odlučili o stanju osiguravajuće police
- Trgovinski pametni ugovori koje traže različite trgovinske podatke i digitalne potpise, te se mnogi takvi ugovori žele naplatiti na tradicionalnim bankarskim putem

Danas, solucija za rješavanje ovog problema i povezivanje sa vanjskim svijetom se nudi u obliku entiteta po imenom eng. *„Oracle"* , aludirajući na mudru i sveznajući proročicu iz antičke Grčke.

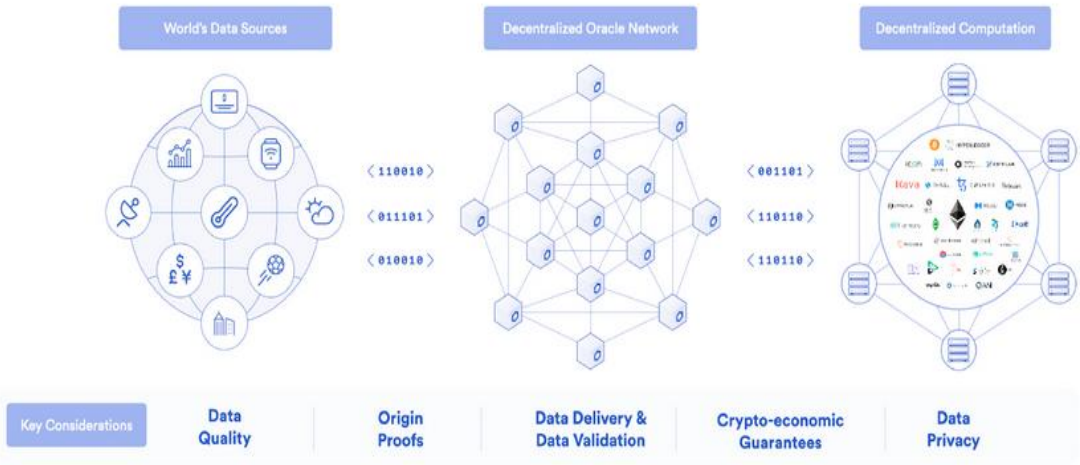
Takav entitet bi trebao biti u mogućnosti riješiti dva tipa problema povezana interakcijom sa vanjskim svijetom:

- Nemogućnost validacije i sigurnog dotoka podataka iz vanjskog svijeta
- Nemogućnost iznosa podataka u druge vanjske sistem niti interakcija sa njima

Blockchain *oracle* je sigurnosni međuprogram *(eng. middleware)* koji olakšava komunikaciju između *blockchaina* i svakog drugog vanjskog sistema koji uključuju podatkovne dobavljače, web API-je , udaljene podatke sa oblaka ili privatnih baza podataka, IoT *(eng. Internet of Things)* senzora, naplatnih sustava ili drugih *blockchaina*.

Funkcije *oracle* entiteta su sljedeće:

- Sluša *blockchain* za potencijalne zahtjeve o vanjskim podacima od strane sudionika ili pametnih ugovora
- Povezanost za više sustava i API-ja
- Formatiranje podataka iz vanjskog svijeta na pripadni *blockchain* ekvivalent
- Procesiranje na dobivenim podacima iz vanjskog svijeta ili pokretanje dugoročnih procesorskih zadataka koje su potrebne pametnih ugovorima
- Validacija različitim metodama kako bi se postigla veća sigurnost obrade podataka
- Slanje podataka ili zahtjeva izvršavanjem nekog zadataka na druge sustave



Slika 2.3: Model decentralizirane *oracle* mreže i njihovih izvora podataka

Da bi *oracle* entitet mogao zadovoljiti svu ovu funkcionalnost mora operirati na blockchainu *(eng. On-Chain)* i izvan njega *(eng. Off-Chain)*:

1. **On-Chain**

   Uspostavlja vezu sa pametnim ugovorom na blockchainu gdje će slušati na njegove zahtjeve. Slati će parametre sa blockchaina o dobivenim zadacima i upisivati rezultate i potvrde o izvršenom poslu.

2. **Off-Chain**

   Procesiranje zahtjeva, dohvat i obrada vanjskih podataka, slanje podataka na druge sustave itd.

Projekt **Chainlink** se postavio kao vodeće rješenje i standard za rješavanje problema vanjskih podataka na pametnim ugovorima *(eng. oracle problem)*. On je u svojem članku [4] predložio osnovne ideje i teze kako bi se ovaj problem riješio pomoću decentralizirane *oracle* mreže koja bi nudila mnoštvo različitih jamstva i solucije koje se mogu kombinirati kako bi se prilagodili korisniku i pametnom ugovoru koji bi koristi takav sustav.

Projekt *Chainlink* je ponudio inovativna i efikasna rješenja kroz ove solucije:

- **Decentralizacija**

  Korištenje modela decentralizacije na razini Chainlink čvorova te na razini višestrukog izvora podataka. Tako pametni ugovori ne bi ovisili samo o jednom čvoru i jednom izvoru podataka.

- **Politika „otvorenog koda” *(engl. open-source)***

  Objavljivanjem programskog koda cijelog projekta se postigla transparentnost i mogućnost da svi sudionici neovisno provjere sigurnost i pouzdanost tehnologije.

- **Obvezujući Ugovor o uslugama (eng. Binding Service Agreement)**

  Oracle *Chainlink* čvorovi stupaju u obvezujući ugovor o dotoku podataka kojeg su dužni izvršavati po unaprijed upisanim uvjetima. Svako neizvršavanje poslova specificiranih u ugovoru, utjecat će na reputaciju tog čvora te će mu smanjiti šanse za dobivanje novih poslova.

- **Potpisivanje podataka**

  Čvorovi kriptografski potpisuju podatke koje su dostavili pametnim ugovorima što omogućuje korisnicima da vide koji čvor je poslao podatke i provjere povijest izvršavanja zadataka tog čvora kako bi otkrili njegovu pouzdanost.

- **Reputacijski sloj**

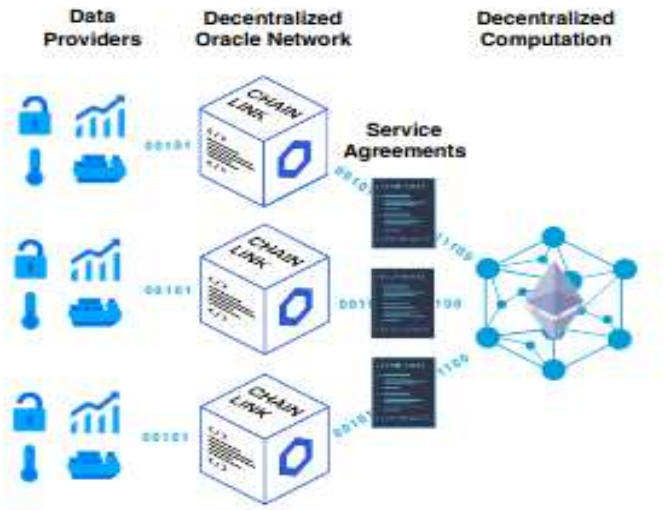
  Izvršavanjem poslova i potpisivanjem dobivenih podataka, čvorovi prolazi kroz reputacijski sloj gdje je zabilježena količina uspješno izvršenih poslova, zahtjeva, klijenata koje je taj čvor posluživao.

- **Certificirane usluge**

  Dopuštenje čvorovima da povećaju svoju sigurnost kroz različite certifikate i garancije

- **Napredna kriptografija i hardver**

  Čvorovi imaju fleksibilnost u odabiru naprednih kriptografskih metoda i hardvera koje mogu koristiti pri izvršavanju poslova koje im mogu date veću sigurnost i bolju performansu.



Slika 2.4: *Chainlink Oracle model*

LINK digitalna valuta

Chainlink oracle mreža koristi LINK digitalnu valutu kao svoje osnovno sredstvo plaćanja čvorovima za izvršavanje poslova specificiranih u „Ugovoru o uslugama". Time je i svaki posao izražen u količini LINK valute koju čvor tražio da bi izvršio taj posao. Također ta digitalna valuta služio kao koletaral u posebnom sistemu koje je *Chainlink* projekt osmislio i implementirao gdje se svako neizvršavanje poslova ili dovod krivih podataka, kažnjava gubitkom tog kolaterala. Tim se osigurava da čvorovi imaju i ekonomsku motivaciju da se ponašaju pošteno i pouzdano.

Da bi pametni ugovori ,poput oni kreiranih na Ethereum *blockchainu*, mogli koristiti *Chainlink* čvorove potrebno je da postoji poseban pametni ugovor koji označava postojanje LINK digitalne valute i sve njegove funkcionalnosti.

2.3 Opis projekta

Motivacija

U ovom poglavlju ćemo opisati projekt imena **„ Movement"** koji je smišljen za ovaj diplomski rad te sve njegove dijelove. Kako je pametni ugovor i njegova povezanost sa vanjskim svijetom glavna tematika ovog diplomskog rada, autor je htio osmisliti jedan mali ekosistem gdje će središnju riječ imati pametni ugovor koji će biti izvršavan na blockchainu *Ethereum*. Ostali dijelovi imati će također važnu ulogu i služit će pametnim ugovoru kao konekcija za vanjskim svijetom.

Cilj je bio napraviti neku vrstu decentraliziranog platforme koja će koristiti lokacijske podatke sa android telefona za neku svoju namjenu. Htjeli smo da pametni ugovor, koji nema informacije o događajima iz vanjskog svijeta, sazna i provjeri je li neki put od početne do krajnje točke napravljen. Namjena ovog projekta može biti raznolika, jer se prelaženje nekog puta sa lokacijskim podacima može se koristiti u svakakve svrhe kao npr, dostava pošiljaka,hrane ili prijevoza ljudi te bilo čega drugog.

Također htjeli smo pokazati kako svi dijelovi međusobno komuniciraju i odrađuju svoju funkcija u ovom sustavu. Time osiguravamo da je ovaj sustav , kojim upravlja pametni ugovor, autonoman i siguran te može služiti kao alternativa sadašnjim tradicionalnim centralnim modelima.

Pregled

Cijeli projekt ima nekoliko različitih i distinktnih cjelina koje možemo razlikovati po korisnosti i funkcionalnosti koju pružaju. Prvo ćemo opisati dijelove infrastrukture koji postoji te njihovu namjenu, a potom entitete i sudionike koje će i taj sustav koristiti.

Opišimo prvo dijelove infrastrukture:

1. **Pametni ugovor**

   Pametni ugovor imena „Movement" bit će kreiran na testnom Ethereum blockchainu Kovan te će služiti kao glavni izvor komunikacija sa *Dapp* web aplikacijom te Android aplikacijom. On će služiti kao glavna baza podataka te ćemo u njega upisivati sve relevantne informacije o korisnicima, rutama i njihovom statusu.

2. **Web aplikacija (Dapp)**

   Frontend web aplikacija napisana uz pomoć *React* biblioteke. Služi za interakciju sa pametnim ugovorom na *Ethreumu.* Registrira korisnike koji onda imaju pristup slobodnim rutama na mobilnoj aplikaciji. Također vodi računa i statusu ruta i događajima koje one povlače.

3. **Mobilna android aplikacija**

   Android mobilna aplikacija napisana u *Kotlinu.* Služit će registriranim korisnicima za prelaženje i status ruta te kao GPS senzor za lokacijske podatke koje će *Chainlink* čvorovi primati kako bi mogli vratiti točne podatke o stanju rute i njihovom izvršavanju na pametni ugovor.

Slika 2.4: Dijelovi projekta „Movement" za diplomski rad

Uz sagrađenu infrastrukturu vežemo i sudionike i entitete koji će ju koristi i čije će međusobne interakcije sačinjavati ovaj projekt (uključili smo engleska imena jer će ona biti prisutna u aplikacijama).

1. **Ruta (eng. Route)**

   Rute će biti zadane udaljenosti od početne do krajnje lokacije koji će korisnici moći preći. Rute će se zadavati preko *Dapp* web aplikacije te će sadržavati i naplatu u digitalnoj valuti Ether. Pametni ugovor će spremati rute, te brinuti za njihov status

2. **Korisnik (eng. User)**

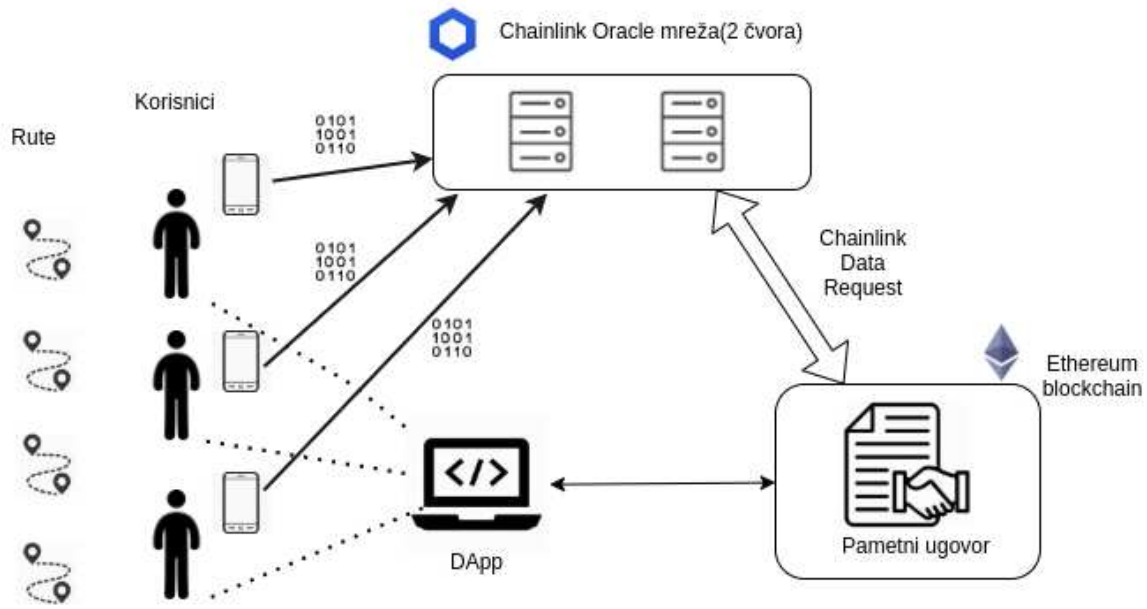
   Korisnici će biti sudionice ekosistema koji će izvršavati rute. Oni će se registrirati preko Dapp web aplikacije da bi imali pristup mobilnoj aplikacija koja će im služiti za obavljanje ruta i dobivanje informacije o njihovom napretku

3. **Čvor (eng. Nodes)**

   Čvorovi će biti računalni serveri koja se sadržavati više procesa složenih u *Docker* kontejnere među kojima su najvažniji **Data Adapter** (služi za obradu i spremanje lokacijskih podataka koji korisnici naprave na android aplikaciji) i **Chainlink node** koji će se služiti svojim *data adapterom* kako bi mogao točno i sigurno provjeriti je li korisnik stvarno napravio tu rutu. On će kao naplatu za izvršavanje svojih poslova (eng. Job) biti naplaćen u digitalnoj valuti **LINK** koja predstavlja *Chanlink-ovo* sredstvo plaćanja na *blockchainu*.

Slika 2.4: Sudionici i entiteti u projektu

Izradili smo jednostavan dijagram na slici 2.5 koji pobliže opisuje naš projekt i sve njegove dijelove. Dijagram ne sadržava sve pojedinosti i interakcije između dijelove, ali na najbolji mogući način opisuje ideju ovog projekta. U budućim poglavljima ćemo detaljno opisati svaki dio i njegovu funkcija te kako on pridonosi.



Slika 2.5: Pregled projekta i svih njegovih dijelova

2.4 Implementacija pametnog ugovora

Uvod

U ovom poglavlju predstaviti ćemo naš pametni ugovor koji će biti izvršavan na testnom *Ethereum blockchainu* Kovan. Ukratko ćemo opisati njegov proces izrade i alate, njegove dijelove te funkcionalnost.

Solidity

Pametni ugovor *„Movement"* napisan je u objektno orijentiranom programskom jeziku više vrste **Solidity** koji služi za izradu pametnih ugovora na Ethereumu. Taj programski jezik je naslijedio mnoga svojstva od strane *C++, Pythona i Javascripta* te je namijenjen da služi pisanju i implementiranju pametnih ugovora na bilo kakvom blockchainu koji koristi EVM (eng. Ethreum Virutal Machine).

Solidity je statički pisan programski jezik koji u sebi posjeduje mogućnost nasljeđivanja, rad sa ostalim modulima, kompleksno definirane tipove podataka i ostale stvari.

Također svaki pametni ugovor posjeduje *eng. Application Binary Interface (ABI)* koji opisuje funkcije, varijable i tipove podataka koji su univerzalni za pametne ugovore. ABI uvelike olakšava i povezanost pametnog ugovora sa drugim programskim jezicima kroz

različite biblioteke. Specifično mi ćemo koristi Java paket *web3j* za našu mobilnu aplikaciju u Kotlinu, a *JavaScript* paket *web3js* za našu web *Dapp* aplikaciju.

Proces izrade pametnog ugovora je takav da se pametni ugovor napisan u *Solidity-ju* prvo pretvara u *bytecode* koji će se moći izvršavati na *EVM (eng. Ethreum Virtual Machine)*, a tek onda šalji na *blockchain* u novoj transakciji.

Alati

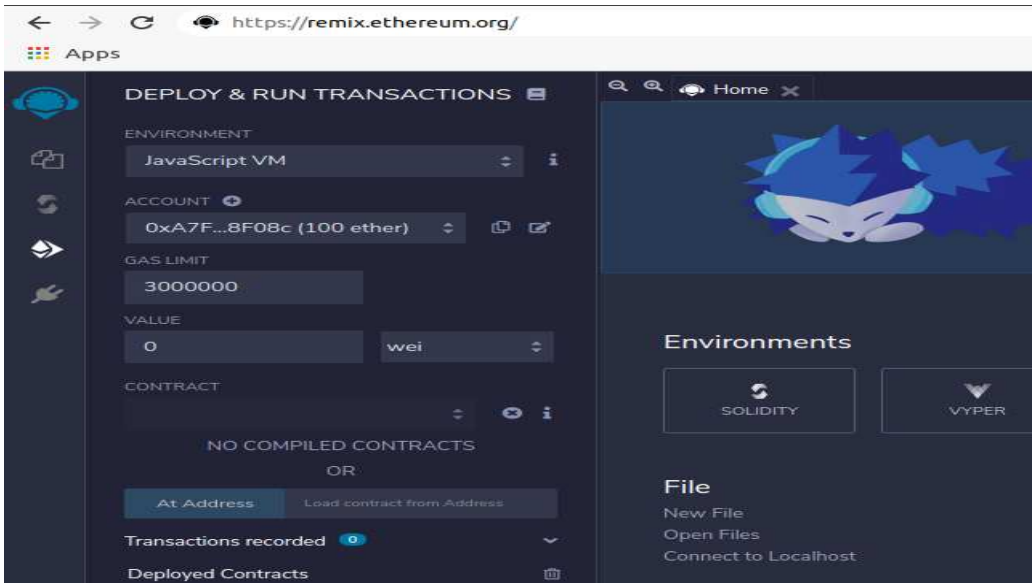
Nekoliko alata će nam biti potpuno neophodna za interakciju sa pametnim ugovorom i njihovu izradu. Ukratko ćemo ih nabrojiti i opisati njihovu funkcionalnost:

- **Remix IDE**

  Glavni online alat za pisanje i *debugiranje* pametnih ugovora u jeziku *Solidity*, koji onda i automatski kreira zadani pametni ugovor na *blockchainu* te izvršava funkcije ili čita stanje pojedinih varijabli.

  Naš pametni ugovor je na lokalnoj memoriji računala i da bi bio povezan s web alatom Remix IDE -om potrebno mi je dati dopuštenje da iščitava sadržaj datoteke u kojem se nalazi naš pametni ugovor.

  Također da bi ga se moglo spojiti sa *blockchainom* i pokretati funkcije pametnog ugovora potreban je web novčanik MetaMask koji će onda služiti kao poveznica kroz koji će ići sve upisne transakcije koje zahtijevaju digitalnu valutu Ether.



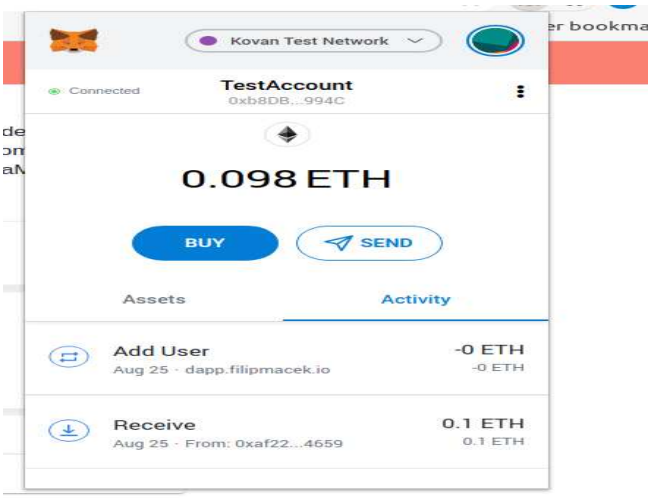
Slika 2.4: Remix IDE za rad sa pametnim ugovorima na Ethreumu

- **MetaMask novčanik**

  Metamask je *web Browser* ekstenzija koja služi kao digitalni novčanik, a ima i funkcionalnost da komunicira sa pametnim ugovorima ili *Dapp* web aplikacijama. Kako svaka transakcija i upis podataka u pametni ugovor treba neku količinu digitalne valute *Ether* ovo je neophodan alat za izradu pametnih ugovora na *blockchainu.*

  On je nam biti glavna veza sa *blockchainom* jer je potreban i za rad sa Remix IDE -om te za konekciju *Dapp* web aplikacija sa *blockchainom* jer *Metamask* unosi u DOM (eng. Document Object Model) svake stranice, web3 API koji će nam biti potpuno nužan za našu web aplikaicju.

  Web3 API je zapravo protokol s kojim komuniciramo s *Ethereum* klijentom održavanim od strane *MetaMaska*. Time nam se pojednostavljuje izrada *Dapp* web aplikacija jer ne moramo brinuti o održavanju *Ethereum* klijenta. Dok kod ostalih dijelova infrastrukture, točnije mobilne aplikacija i *Chainlink* čvora, morati ćemo sami osigurati *Ethereum* klijenta. To nas dovodi do sljedećeg alata.



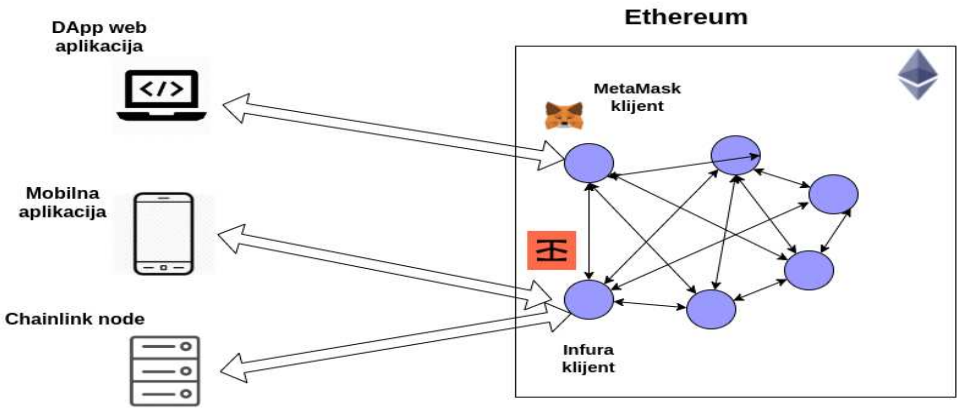
Slika 2.5: Metamask novčanik

- **Ethereum Infura klijent**

  Infura je *blockchain* kompanija i platforma koja nudi profesionalni usluge povezivanja sa *Ethereum-om* kroz svoje klijente. Oni nude besplatnog klijenta do 100,000 zahtijeva *(eng. requests)*, što će nam biti dovoljnu za ovu demo aplikaciju.
  Ethereum klijent će nam biti potreban kod povezivanja *Chainlink* čvora i mobilne aplikacije. Razlikovat ćemo i različite tehnologije povezivanja sa *blockchain* klijentom.
  Chainlink čvor će zahtijevati neprekidnu povezanost sa *blockchainom* da bi mogao osluškivati zatražene poslove *(eng. Jobs)* od pametnog ugovora. To je postignuto

uz pomoć *eng. WebSocket* tehnologije, dok će mobilna aplikacija zahtijevati *HTTP* protokol, jer neće biti potrebno da je osluškuje *Ethereum blockchain* neprekidno već samo kada hoće upisati neke podatke.

Dijagram svih konekcija infrastrukture sa Ethereum blockchainom izgleda ovako:



Slika 2.4: Dijagram povezanosti infrastrukture sa svojim Ethereum klijentima

Implementacija

Pametni ugovori napisani u *Solidity* se po tehnici pisanja ne razlikuju mnogo od objektnog programiranja i sam pametni ugovor možemo gledati kao jednu veliku klasu koji sadrži svoje varijable i metode/funkcije te može naslijediti varijable i funkcije iz drugih klasa. Svaki pametni ugovor deklaracije ovih mogućih tipova:

- **Varijable**
- **Funkcije**
- **Funkcijskih modifikatora** - određuju uz koje uvjete se funkcija izvršava
- **Događaji**
- **Struct tipovi** – kompleksni tipovi podataka koji sadrže vise različitih varijabli

Nasljedstvo

Naš pametni ugovor ***Movement*** nasljeđuje od druga dva pametna ugovora njihovu funkcionalnost. To su:

1. **ChainlinkClient**

   To je modul koji mu daje funkcionalnost da komunicira sa *Chainlink* čvorom pomoću različitih funkcija i koja mu olakšavaju slanje *Chainlink* zahtijeva.

2. **Ownable**

   Olakšava implementaciju vlasništva pametnog ugovora i osigurava autorizacijski sloj pametnog ugovora.

```
contract Movement is ChainlinkClient,Ownable{
```

Slika 2.5: Nasljedstvo pametnog ugovora

Struct tipovi podataka

Opišimo prvo sve kompleksne tipove podatke (*eng. struct*) koje ćemo koristi i koje ćemo spremati u niz da bi pametni ugovor imao uvid u sve događaje koji su mu bitni:

- **Korisnik**

```
struct User {
    uint userId;
    string username;
    string password;
    address addr;
    bool isExist;
    uint routesStarted;
    uint routesCanceled;
    uint routesFinished;
    uint routesCompleted;
}
// mapping from user address to userId
mapping(string =>bool) userExistsByUsername;
mapping(address => bool) userExistsByAddress;
mapping(string => uint) userIndexByUsername;
// Users array
User[] public users;
```

Slika 2.6*: User struct*

- **Ruta**

```
struct Route {
    uint routeId;
    address maker;
    string taker;
    string startLocation;
    string endLocation;
    bool isStarted;
    bool isFinished;
    bool isCompleted;
    string description;
}
 // Routes array
Route[] public routes;
```

Slika 2.7*: Route struct*

- **Node (Chainlink čvor)**

```
struct Node {
    uint nodeId;
    string nodeName;
    string ip;
    string data_endpoint;
    address oracleContractAddress;
    uint routesChecked;
    string jobIdDistance;
    string jobIdTime;
    string jobIdStatus;
    }
// Nodes array
Node[] public nodes;
```

Slika 2.8: *Node struct*

- **RouteStartEvent**

  Služio kao podatak da je korisnik prihvatio i započeo izvršavanje rute preko mobilne aplikacije.

```
struct RouteStartEvent{
    uint routeStartId;
    uint routeId;  // which route
    string username;    // who started this route
    uint timestamp;   // when the event has been published
    bool node1Status;
    bool node2Status;
}
RouteStartEvent[] public routeStartEvents;
```

Slika 2.9*: RouteStartEvent struct*

- **RouteEndEvent**

  Upisuje se kad korisnik završio rutu na mobilnoj aplikaciji i prije nego što *Chainlink* čvor upiše na *blockchain* stanje rute, tj. je li ona završena ili nije. Također uključena je informacija je li korisnik završio rutu ili ju se prekinuo.

```
struct RouteEndEvent{
    uint routeEndId;
    uint routeId;
    string username;
    uint timestamp;     // when the event has been published
    uint dataPoints;    // how many coordinate data points the app recorded
    uint node1DataPoints;
    uint node2DataPoints;
    uint userStatus;  // state in which route finished - did user cancel route, or sumbited it
}
RouteEndEvent[] public routeEndEvents;
```

Slika 2.10: *RouteEndEvent struct*

- **CheckStatusEvent**

  Sadrži sve informacije o izvršavanju zahtjeva *Chainlink* čvorova i njihovih odgovora o stanju rute, tj. je li korisnik završio rutu i u kojem vremenu i distanci.

```
struct CheckStatusEvent{
    uint checkStatusId;
    uint routeId;
    string username; // for which user we are checking if he completed the route;
    uint timestamp;
    uint node1Distance;
    uint node1Time;
    bool node1Status;
    uint node2Distance;
    uint node2Time;
    bool node2Status;
}
CheckStatusEvent[] public checkStatusEvents;
```

Slika 2.11: *CheckStatusEvent struct*

Događaji

Događaji (*eng. events*) su još jedan važan član pametnih ugovora na *Ethereumu*. Oni su apstrakcija evidentiranja *(eng. logging)* na *blockchainu*, gdje svaki događaj kada bude emitiran biva upisan u transakcijski dnevnik svog pripadajućeg pametnog ugovora, te ih različiti klijenti mogu iščitati ili se pretplatiti na novonastale događaje o kojima će biti obaviješteni.

S obzirom na pokrenute funkcije pametnog ugovora, odašiljat će se i različiti događaji. Ovo je njihov popis:

```
//============= Events ======================
  event UserCreated(uint userIndex);
  event RouteCreated(uint routeIndex);
  event StartRouteEvent(uint routeStartId);
  event EndRouteEvent(uint routeEndId);
  event StatusCheckEvent(uint checkStatusId);
  event StatusCheckFulfilled(uint checkStatusId);
  event CompletedRouteEvent(uint routeCompletedId);
```

Slika 2.12: Događaji pametnog ugovora

Globalne varijable

- **Adresa agenta**

  Agent je entitet koji bilježi i snima korisnikove poteze i događaje na mobilnoj aplikacija gdje taj korisnik izvršava zadane rute. To nam je bitno kako bismo upisali sve korisnikove odluke na blockchain. Agent nije zadužen za provjeru je li korisnik napravio rutu već samo motri ono što mu je zadano.

- **Adresa LINK digitalne valute**

  Chainlink čvorovi i njegovi pripadajući pametni ugovori koje koristi pri izvršavanju zadataka trebaju znati točnu adresu *LINK* digitalne valute koja im je standardizirana valuta naplate.

- **Količina naplate po izvršenom poslu**

  Za koliku količinu *LINK* digitalne valute će *Chainlink* čvorovi biti naplaćeni po izvršenom poslu.

- **Konstante** vezane za stanje je li korisnik prekinuo rutu tokom izvršavanja (ili zatvorio aplikaciju iako nije dovršio rutu) ili je korisnik uredno slijedio proces i dovršio rutu do kraja.

```
address private agent;

uint constant private ORACLE_PAYMENT = 1*LINK;

address constant LINK_TOKEN_ADDRESS =0xa36085F69e2889c224210F603D836748e7dC0088;

// Constants related to states that can happen with started route
 uint constant USER_CANCELED = 1;
 uint constant USER_SUBMITED_ROUTE = 2;
```

Slika 2.13: Globalne varijable

Funkcije

Funkcije ili metode pametnog ugovora su osnova za njegov rad i izvršavanje različitih potrebnih funkcionalnosti. Opisat ćemo sve najbitnije funkcije za rad našeg pametnog ugovora. One mogu biti pozvane od strane mobilne android aplikacije ili Dapp web aplikacija ovisno o namjeni.

- **addUser**

  Pametni ugovor će pomoću ove funkcije registrirati novog korisnika i dati mu pristup mobilnoj aplikacija gdje će se korisnik pomoću korisničkog imena i zaporke moći se ulogirati.

```
function addUser(string memory _username, string memory _password) public {
    users.push(User(users.length+1,_username,_password,msg.sender,true,0,0,0,0));
    userExistsByUsername[_username]=true;
    userExistsByAddress[msg.sender]=true;
    userIndexByUsername[_username]=users.length-1;
    emit UserCreated(users.length-1);
}
```

Slika 2.14: Funkcija *addUser*

- **addRoute**

  Ova funkcija registrira novu rutu koja je neki sudionik zadao na *Dapp* web aplikaciji. Da bi ruta bila zadana i upisana potrebno joj je startna lokacija i krajnja lokacija te kratki opis rute. Svaka lokacija mora sadržavati geografsku širinu *(eng,. latitude)* i dužinu *(engl. longitude)* u formatu „45.2322, 15.32112".

```
function addRoute(string memory _startLocation,string memory _endLocation,string memory _description)public {
    string memory taker = '';
    routes.push(Route(routes.length+1,msg.sender,taker,_startLocation,_endLocation,false,false,false,_description));
    emit RouteCreated(routes.length-1);
}
```

Slika 2.15: Funkcija *addRoute*

- **addNode**

  Registrira novog *Chainlink* čvora te upisuje sve relevantne podatke, kao što su njegova IP adresa i eng. „Data Endpoint" koja služe da bi mobilna android aplikacija znala na koju adresu da šalje lokacije podatke. (Npr. format „ip:endpoint" – *53.21.123.22/api).*

  Također sadrži identifikacijsku oznaku za *Chainlink* čvor, da bi znao koji posao *(eng. job)* dohvata podatka na *blockchain* treba izvršiti.

```
function addNode(string memory _name,string memory _ip,string memory _data_endpoint,
                 address _oracleContractAddress,
                 string memory _jobIdDistance,
                 string memory _jobIdTime,
                 string memory _jobIdStatus) private {
    nodes.push(Node(nodes.length+1,_name,_ip,_data_endpoint,_oracleContractAddress,0,_jobIdDistance,_jobIdTime,_jobIdStatus));
}
```

Slika 2.16: Funkcija *addNode*

- **startRouteEvent**

  Pokreće se kada korisnik započne izvršavanje rute na mobilnoj aplikaciji. Ovom funkcijom mijenjamo status rute i tu informaciju upisujemo na *blockchain*. Također funkciju sadrži *modifier „onlyAgent"* što znači da ju jedino može izvršiti adresa agenta.

```
// Agents functions
function startRouteEvent(uint _routeId,string memory _username,bool _node1Status,bool _node2Status) public onlyAgent{
    // require that user is registerd and exists
    require(userExistsByUsername[_username],"User doesnt exists");

    // Change route status
    // Dont forget to add -1 because its index of array
    routes[_routeId-1].isStarted = true;
    routes[_routeId-1].taker =_username;
    users[userIndexByUsername[_username]].routesStarted+=1;


    // Add info to routesStartEvents
    routeStartEvents.push(RouteStartEvent(routeStartEvents.length+1,_routeId,_username,now,_node1Status,_node2Status));

    emit StartRouteEvent(routeStartEvents.length-1);


}
function getRouteStartEventCount() public view returns(uint){
    return routeStartEvents.length;
}
```

Slika 2.17: Funkcija *startRouteEvent*

- **endRouteEvent**

  Izvršava se kada korisnik prekine ili završi izvršavanje rute na mobilnoj aplikacija. Moramo uračunati oba slučaja kada korisnik prekine izvršavanje rute ili zatvori aplikaciju) te dovrši rutu do kraja.

  Upisujemo još neke meta podatke koje je mobilna aplikacija zabilježila, kao što su količina lokacijskih podataka koji su postojali te koliko su ih *Chainlink* čvorovi primili.

```
function endRouteEvent(uint _routeId,string memory _username,uint user_event,uint _dataPoints,uint _node1DataPoints,uint _node2DataPoints)public onlyAgent{
    uint len = routeEndEvents.length;
    if(user_event == USER_CANCELED){
        // User canceled so we dont care about any other data
        // we init both dataPoints and node data to 0

        routeEndEvents.push(RouteEndEvent(len+1,_routeId,_username,now,0,0,0,USER_CANCELED));
        routes[_routeId-1].isStarted = false;
        routes[_routeId-1].taker ='';
        users[userIndexByUsername[_username]].routesCanceled+=1;
        emit EndRouteEvent(routeEndEvents.length-1);

    }else if(user_event == USER_SUBMITED_ROUTE){
        // User sumbited route
        routeEndEvents.push(RouteEndEvent(len+1,_routeId,_username,now,_dataPoints,_node1DataPoints,_node2DataPoints,USER_SUBMITED_ROUTE));
       routes[_routeId-1].isFinished =true;
       users[userIndexByUsername[_username]].routesFinished+=1;


       // User submited route - call requestRouteStatus to ask chainlink nodes about route status
       requestRouteStatus(_routeId,_username);


       emit EndRouteEvent(routeEndEvents.length-1);

    }
}
```

Slika 2.18: Funkcija endRouteEvent

- **requestRouteStatus**

  To je privatna funkcija koje će biti pozvana iz funkcije **endRouteEvent** ukoliko korisnik pravilno završi rute. Time će je efektivno agent pozvati i on će snositi transakcijske troškove te funkcije.

  Funkcija na registriranu adresu *(eng. „oracle address")* s kojom su registrirani *Chainlink* čvorovi šalje poseban tip zahtjeva eng. „*ChainlinkRequest*" svim upisanim čvorovima na *blockchain* da izvrše svoje poslove i dohvate joj tražene podatke.

  Bit će nam potrebna tri tipa podatka od svakog čvora. A to su: informacija je li korisnik završio rutu, potrebno vrijeme u sekundama i udaljenost u metrima koju je prošao.

```
// Request route status by user on dapp
function requestRouteStatus(uint _routeId,string memory _username) private {

    string memory _routeIdString = uintToString(_routeId);


    // Init CheckStatusEvent data -- we will write to this variable all our results
    checkStatusEvents.push(CheckStatusEvent(checkStatusEvents.length+1,_routeId,_username,now,0,0,false,0,0,false));
    emit StatusCheckEvent(checkStatusEvents.length);


    //--------------Node 1 ----------------------
        // Distance Data Request
        Chainlink.Request memory req1 = buildChainlinkRequest(stringToBytes32(nodes[0].jobIdDistance),address(this),this.fulfillDistanceRequest1.selector);
        req1.add("extPath",_routeIdString);
        sendChainlinkRequestTo(nodes[0].oracleContractAddress,req1,ORACLE_PAYMENT);

        // // Time Data Request
         Chainlink.Request memory req2 = buildChainlinkRequest(stringToBytes32(nodes[0].jobIdTime),address(this),this.fulfillTimeRequest1.selector);
        req2.add("extPath",_routeIdString);
        sendChainlinkRequestTo(nodes[0].oracleContractAddress,req2,ORACLE_PAYMENT);

        // Status Data Request
         Chainlink.Request memory req3 = buildChainlinkRequest(stringToBytes32(nodes[0].jobIdStatus),address(this),this.fulfillStatusRequest1.selector);
        req3.add("extPath",_routeIdString);
        sendChainlinkRequestTo(nodes[0].oracleContractAddress,req3,ORACLE_PAYMENT);


    // -------------------- Node 2 -------------------------------
       // Distance Data Request
        Chainlink.Request memory req4 = buildChainlinkRequest(stringToBytes32(nodes[1].jobIdDistance),address(this),this.fulfillDistanceRequest2.selector);
        req4.add("extPath",_routeIdString);
        sendChainlinkRequestTo(nodes[1].oracleContractAddress,req4,ORACLE_PAYMENT);

        // // Time Data Request
         Chainlink.Request memory req5 = buildChainlinkRequest(stringToBytes32(nodes[1].jobIdTime),address(this),this.fulfillTimeRequest2.selector);
        req5.add("extPath",_routeIdString);
        sendChainlinkRequestTo(nodes[1].oracleContractAddress,req5,ORACLE_PAYMENT);

        //Status Data Request
         Chainlink.Request memory req6 = buildChainlinkRequest(stringToBytes32(nodes[1].jobIdStatus),address(this),this.fulfillStatusRequest2.selector);
        req6.add("extPath",_routeIdString);
        sendChainlinkRequestTo(nodes[1].oracleContractAddress,req6,ORACLE_PAYMENT);

        emit StatusCheckFulfilled(checkStatusEvents.length);
}
```

Slika 2.19: Funkcija *requestRouteStatus*

- ***Fulfill* skup funkcija**

  To su poseban tip funkcija koje se pozivaju kada *Chainlink* čvorovi izvrše zadane poslove i prime potvrdu da im je uplaćena *LINK* digitalne valuta kao naplata. Oni tada upisuju sve zatražene podatke u *blockchain.*

```
// Status Callbacks for node1 && node2

 function fulfillStatusRequest1(bytes32 _requestId,bool _status)public recordChainlinkFulfillment(_requestId){
    checkStatusEvents[checkStatusEvents.length-1].node1Status = _status;

}
    function fulfillStatusRequest2(bytes32 _requestId,bool _status)public recordChainlinkFulfillment(_requestId){
    checkStatusEvents[checkStatusEvents.length-1].node2Status = _status;

}
```

Slika 2.20: Primjer dvije *callback* funkcije oba *Chainlink* čvora o stanju rute

- ***GetCount* skup funkcija**

  Ovaj skup funkcija samo vraća duljina svih inicijaliziranih nizova za već definirane tipove podataka. Potreban nam je kod poziva sa *Dapp* web aplikacije da bismo znali koliko API poziva je potrebno napraviti da dostavimo podatke sa *blockchaina* na web aplikaciju.

```
// Get Count funkcije

function getUsersCount() public view returns(uint) {
    return users.length;
}

function getRoutesCount() public view returns(uint) {
    return routes.length;
}

function getNodesCount()public view returns(uint){
    return nodes.length;

}
function getRouteStartEventCount() public view returns(uint){
    return routeStartEvents.length;
}
function getRouteEndEventCount() public view returns(uint){
    return routeEndEvents.length;
}
function getCheckStatusEventCount () public view returns(uint){
    return checkStatusEvents.length;
}
```

Slika 2.21: *GetCount* tip funkcija

Poglavlje 3

Chainlink čvor

3.1 Uvod

U ovom poglavlju ćemo opisati sve dijelove *Chainlink* čvora. Kao važna napomena je da kada kažemo *Chainlink* čvor (eng. Node) mislimo i na ostale pomoćne procese koje su mu potrebni da bi izvršavao svoju funkcija (npr. *DataAdapter, Mongo* baza podataka itd).

3.2 Opis tehnologija i procesa

Odlučili smo se u ovom projektu koristiti dva *Chainlink* čvora, tj, koristi ćemo dva *Linux* servera uz pomoć Amazon web usluga *(eng. Amazon Web Services)* .Ti Linux serveri će imati potpuno istu konstituciju, iste procese i istu funkciju. Time će i oba čvora vraćati iste podatke na *blockchain* o stanju ruta koje je korisnik napravio kada budu zatražene od pametnog ugovora. Iako će čvorovi vraćati iste podatke i napraviti isti zaključak o tome je li korisnik prešao rutu, cilj je ipak bio pokazati kako pametni ugovor može decentralizirati svoj izvor podataka i ne ovisiti samo o jednom čvoru i njegovom odluci.

Name	Instance ID	Instan...	Ins...	Stat	Avai	Availab...	Pub	Public IPv4 ...
node1	i-08ae2fa30e3fbc87e	Running	t2.micro		+	us-east-1e	e...	100.26.234.183
node2	i-0d9ff6d6b006f5980	Running	t2.micro		+	us-east-1e	e...	52.91.148.208

Slika 4.1: Linux serveri na *AWS* oblaku koji služe kao *Chainlink* čvorovi

Docker

Kao glavnu tehnologiju za povezivanje svih procesa koristili smo *Docker*. Ideja je da izoliramo procese u jednostavne kontejnere koji se lako mogu pokrenuti i zaustaviti. *Docker* koristi mogućnost virtualizacije u operacijskom sustavu da bi zapakirao te procese te uvelike smanjio količinu potrebnih resursa nego tradicionalnije virtualnih mašine *(eng. Virtual Machine).* Koristili smo i pomoćnu konfiguraciju *docker-compose* koje nam omogućava da lako i jednostavno spojimo više *Docker* procesa u jednu mrežu. Ti kontejneri se slažu tako da se prvotno specificira početna slika (tj. operacijski sustav) od kojeg će taj kontejner biti složen, a onda instaliraju pomoćni paketni i svi ostali alati. Na kraju kopiramo naš programski kod te mu se daje informacija kako da se pokrene taj kontejner i koje pristupne točke *(eng. ports)* su otvorene za komunikaciju sa drugim procesima.

PostgresSQL i MongoDB

Koristit ćemo relacijsku(eng. SQL) bazu podataka *Postgres* koja je potrebna *Chainlink* čvoru kako bi upisao sve njemu važne i relevantne podatke (npr. poslove koje izvršava, njihove konfiguracije itd.).

Za potrebe spremanja i čitanja lokacijskih podataka od strane našeg Data Adapter, koristili smo nerelacijsku (eng. NoSQL) *MongoDB* bazu podataka.

Redis

Redis je posebna baza podataka u koja sprema tražene strukture podataka u memoriju i nama služi kao red (*eng. queue*) u koji upisujemo lokacijski podataka koji je pristigao, da ga u pozadini može obraditi radnik (*eng. worker*) napisan uz pomoć *Javascript* paketa „bull”.

Data Adapter

Data Adapter je web server napisan u jeziku *Javascript* uz pomoć paketa *express* i ”mongoose” koji ga spaja sa *MongoDB* bazom podataka. Cilj ovog procesa je da ima otvorene rute i prihvaća podatke sa mobilne aplikacije kada korisnik započne izvršavanje rute. Kada ovaj proces primi lokacijski podatak, ta informacija se upisuje u *redis* i naš radnik započinje obradu tog lokacijskog podataka. Time naš server ne blokira, već obrađuje podatke u pozadini.

Nginx

Kako bi osigurali veću sigurnost našeg servera, otvorit ćemo samo *port 80* kroz koji će prolaziti svi naši lokacijski podaci. Time nam web server *nginx* služi kao *eng. reverse proxy* koji samo prosljeđuje dobivene podatke na ostale procese po unaprijed zadanim pravilima.

Chainlink čvor

Sam *Chainlink* čvor napisan je u jeziku *Golangu*, i služi zapravo kao eng. *middleware* između blockchaina i vanjskog svijeta. Cilj je da su svi čvorovi na decentraliziranoj *Chainlink oracle* mreži standardizirani i koriste istu strukturu specifikacija za zadane poslove. Neka od najvažnijih svojstva ovog programa su:

- jednostavna povezanost sa pametnim ugovorima na *blockchainu* sa bilo kojim vanjskim sustavom ili API-jem različite metode dohvata i obrade podataka za potrebe pametnog ugovora
- automatska briga o troškovima upisa na blockchain
- translacija različitih tipova podataka u pripadajuće tipove na *Ethereum* blockchainu
- stvaranje notifikacija prilikom svake promjene stanje na pametnom ugovoru, koji prati pomoću transakcijskog dnevnika na *Ethereum*

Chainlink čvor prilikom inicijalizacije stvara svoj privatnih i javnih ključ kako bi imao vlastitu adresu na blockchainu te bi pomoću nje mogao upisivati dohvaćene tražene podatke kada oni budu potrebni.

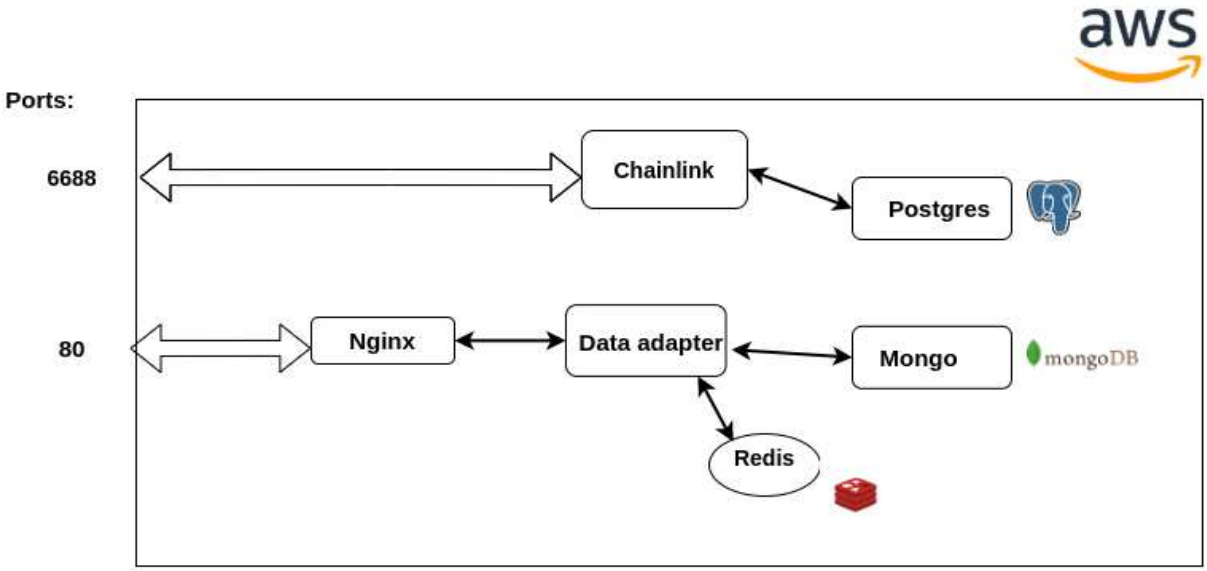
Komunicirat ćemo sa Chainlink čvorom pomoću njegovog web sučelja koji se otvara na *portu* 6688.

Popis svih procesa u komandnoj liniji može se vidjeti na slici 4.2

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
fcb6abf7c8fd	dataadapter_data_adapter	"docker-entrypoint.s…"	5 days ago	Up 5 days	0.0.0.0:3000->3000/tcp	data_adapter
8db3fd25b97b	mongo:latest	"docker-entrypoint.s…"	5 days ago	Up 5 days	0.0.0.0:27017->27017/tcp	mongo
13971e8e3ee4	redis:latest	"docker-entrypoint.s…"	7 days ago	Up 7 days	0.0.0.0:6379->6379/tcp	redis
9dd9f9d39a99	dataadapter_nginx	"nginx"	7 days ago	Up 6 days	0.0.0.0:80->80/tcp	nginx
f336faccb75e	smartcontract/chainlink	"chainlink local n"	7 days ago	Up 7 days		chainlink
2eaabc40a66a	postgres	"docker-entrypoint.s…"	7 days ago	Up 7 days	0.0.0.0:5432->5432/tcp	postgres

Slika 4.2: Svi *Docker* procesi na serveru od *Chainlink* čvora

Prikažimo cijeli server sa svim njegovim dijelovima u jednom dijagramu kako bi lakše mogli razumjeti kako je sve povezano.



Slika 4.3: Dijagram Chainlink čvora i njegovih dijelova

3.3 Implementacija

Nakon što smo opisali sve procese i dijelove koji nam omogućavaju da naš *Chainlink* čvor izvršava svoju funkcionalnost ,pogledajmo u ovoj cjelini kako sistem funkcionira sa tehničkog gledišta. Prikazat ćemo sve najbitnije dijelovi da bi Chainlink čvor uspješno izvršavao svoje poslove te potrebna konfiguracija za te poslove.

Prvo što je potrebno da bi pokrenuli *Chainlink* čvor je inicijalizacija početnih varijabli *(eng. environment variables)*

```
ROOT=/chainlink
LOG_LEVEL=debug
ETH_CHAIN_ID=42
MIN_OUTGOING_CONFIRMATIONS=2
LINK_CONTRACT_ADDRESS=0xa36085F69e2889c224210F603D836748e7dC0088
CHAINLINK_TLS_PORT=0
SECURE_COOKIES=false
GAS_UPDATER_ENABLED=true
ALLOW_ORIGINS=*
ETH_URL=wss://kovan.infura.io/ws/v3/42f42a255e264be7a6bc8373c4308e96
DATABASE_URL=postgresql://node:pass@localhost:5432/chainlink_db?sslmode=disable
```

Slika 4.4: Početne varijable *Chainlink* čvora

Ukratko opišimo najbitnije varijable:

- ETH_CHAIN_ID

  Označava identifikacijski broj Kovan testnog *Ethereum blockchaina.*
- LINK_CONTRACT_ADDRESS

  Adresa LINK digitalne valute koje *Chainlink* čvorovi koriste kao sredstvo naplate za svoje usluge.
- DATABASE_URL

  Konfiguracijski tekst koji sadrži korisničko ime, zaporku i lokaciju za *Postgres* bazu podataka.
- ETH_URL

  WebSocket adresa *Infura Ethereum* klijenta koji mu je potreban da bi osluškivao *blockchain* za moguće tražene poslove koje je spreman izvršiti.

Prilikom inicijalizacije *Chainlink* čvora, smo također morali upisati email adresu i zaporku. Pomoći njih ćemo se se spojiti na web sučelje *Chainlink* čvora ili drugog naziva **Operatora** koji se paralelno koristi. Pomoću tog web sučelja mi ćemo imati uvid u sve izvršene poslove i upisane podatke te specifikaciju svih poslova.

Poslovi

Funkcija dohvata vanjskih podataka ili izvršavanje nekih zadataka izvan *blockchaina*, izvršavaju se kroz entitet poslova *(eng. Jobs).*

Njihova specifikacija sadrži sekvencijalno izvršavane zadatke *(eng. tasks)*, koje *Chainlink* čvor mora izvršiti ukoliko želji proizvesti zadani rezultat. Ta specifikacija je napisana u JSON formatu. Njeni dijelovi su:

1. Pokretač (*eng. Initiators)*

   Pokretač je entitet koji određuje kako će se posao početi pokretati. Postoje više vrsta pokretača, a neki od njih su:
   - ***Cron***

     - pokreće ponavljajuće poslove s obzirom na dan vremenski parametar
   - ***EthLog***

     -sluša na blockchainu specifične događaja na temelju kojih onda pokreće posao
   - ***RunLog***

     -najjednostavniji i najkorišteniji pokretač (kojeg ćemo i mi koristiti). Kao EthLog sluša na odgovarajuće događaje koje Oracle pametni ugovor ispušta za njega za kreirani identifikacijski broj posla *(eng. JobID).* Kada izvrši posao, prijavljuje i upisuje odgovarajući rezultat na blockchain.

2. **Zadaci (eng. Tasks)**

   Zadaci su individualni koraci koje *Chainlink* čvor prati da bi obradio podatke. Svaki zadatak se sastoji od adaptera i njegovih parametara.

   Adaptera i njihovih parametara ima mnoštvo,ali nabrojimo samo najvažnije i one koje ćemo mi koristiti:

   - **HttpGet**

     -vraća rezultat uspješnog GET zahtjeva, mi koristimo identičnu inačicu, ali sa pristupom lokalnoj mreži, jer radimo GET zahtjev na lokalnu adresu (*eng. localhost) --httpgetwithunrestrictednetworkaccess*

     -parametri:

     - **get** - adresa url GET zahtjeva
     - **headers** - zaglavlje request, uzima vrijednost i ključ
     - **queryParams** - parametri url GET zahtjeva
     - **extPath** - dodaje na kraj url zahtjeva dodatan tekst

   - **JsonParse**

     - uzima podatak u JSON formatu te uzima specifičan put u tom podatku te uzima njegovu vrijednost

     - parametri:

     - **path** - niz tekstova koji označavaju JSON put

   - **Multiply**

     -pretvara ulazni podataka u decimalni oblik i množi ga odgovarajućim parametrom

     - **times** - s koliko ćemo pomnožiti decimalni broj

   - **EthBool**

     - prima logičku *(eng. boolean)* vrijednost i pretvara je u odgovarajuću

   - **EthTx**

     -najčešće na kraju svakog posla da upiše dobiveni rezultat na *blockchain*, ulazni podatak stavlja kao u podatkovni parametar u transakciju

Nama su potrebni u ovom projektu 3 važna podataka koja nas pametni ugovor *Movement* očekuje da mu ih *Chainlink* čvor dostavi:

1. **Stanje rute**

   Hoćemo znati je li korisnik posjetio početnu lokaciju, a potom otišao do krajnje lokacije. Time podrazumijevamo da je ruta napravljena i da je status pozitivan.

2. **Prijeđena udaljenosti**

   Pređena udaljenost između krajnje i početne točke.

3. **Vrijeme**

   Mjereno vrijeme pri prolasku rute.

Time ćemo imati potpuni uvid o korisnikovom izvršavanju rute i sve potrebne informacije bit će upisane na *blockchain*. Tri naša posla i njihove pripadajuće specifikacije u *JSON* formatu označene su rednim brojevima i glase:

```
{  "initiators": [    {
          "type": "runlog"}  ],
   "tasks": [
          {"type": "httpgetwithunrestrictednetworkaccess",
               "params": {
                    "get": "http://localhost:3000/api/route/status"      }},
          {"type": "jsonparse",
               "params": {
                    "path": [ "distance"] }},
          {"type": "multiply",
                    "params": {       "times": "1"      }},
          {"type": "ethuint256"},
          {"type":"ethtx"} ]
}
```

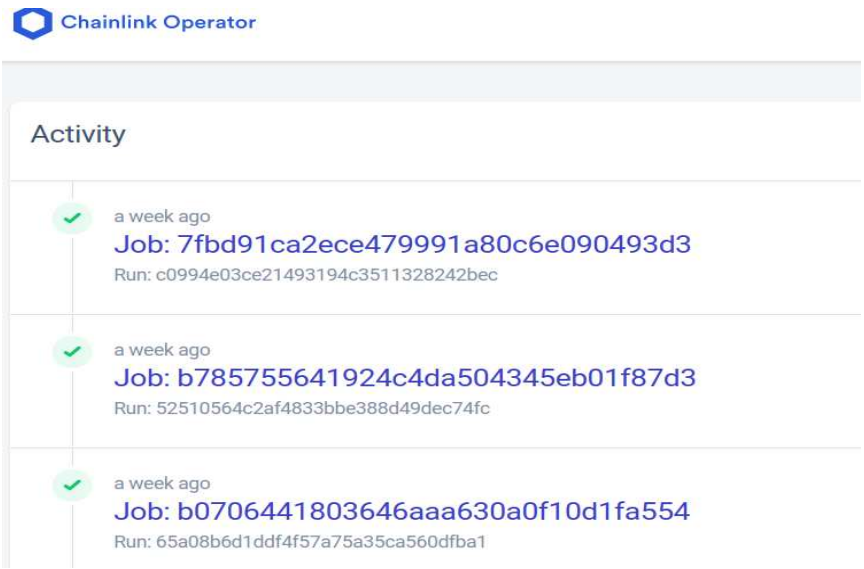
Slika 4.5: Specifikacija posla za izračuna prijeđene udaljenosti

```
{  "initiators": [    {
          "type": "runlog",        }],
   "tasks": [
          {"type":     "httpgetwithunrestrictednetworkaccess",
               "params": {
                    "get": "http://localhost:3000/api/route/status"      }},
           {"type": "jsonparse",
                    "params":{
                              "path": ["time"]}},
          {"type": "multiply",
                    "params": {"times": "1"}},
          {"type": "ethuint256"},
          {"type":"ethtx"}  ]
}
```

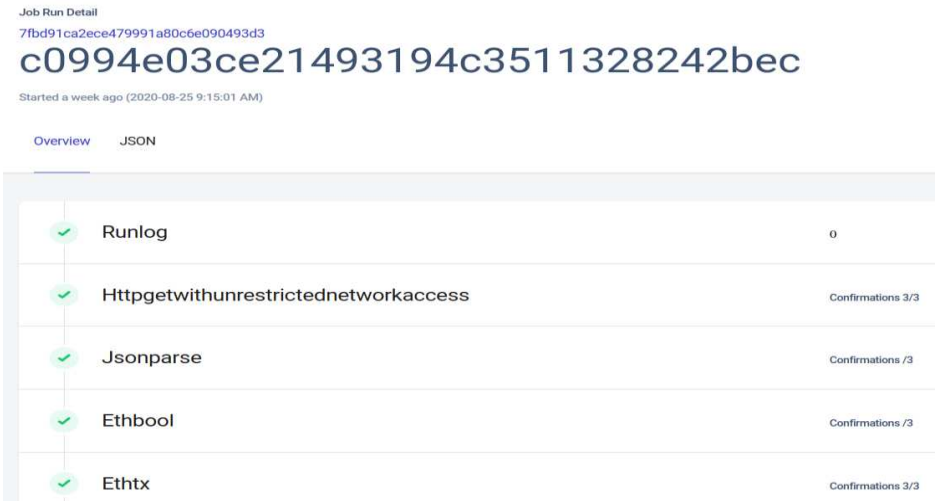
Slika 4.6: Specifikacija posla za izračun potrebnog vremena

```
{  "initiators": [   {
          "type": "runlog"    } ],
   "tasks": [
          {"type": "httpgetwithunrestrictednetworkaccess",
                   "params": {
                   "get": "http://localhost:3000/api/route/status",
                   "extPath": "1"      }    },
          {"type": "jsonparse",
                   "params": {
                            "path": ["routeFinished"] }},
          {"type": "ethbool"},
          {"type": "ethtx"}],
}
```

Slika 4.7: Specifikacija posla o stanju rute



Slika 4.8: Primjer izvršenih poslova na web sučelju *Chainlink* čvora



Slika 4.9:Detalji izvršavanje *Chainlink* posla i njegovi zadaci na web sučelju

Oracle pametni ugovor

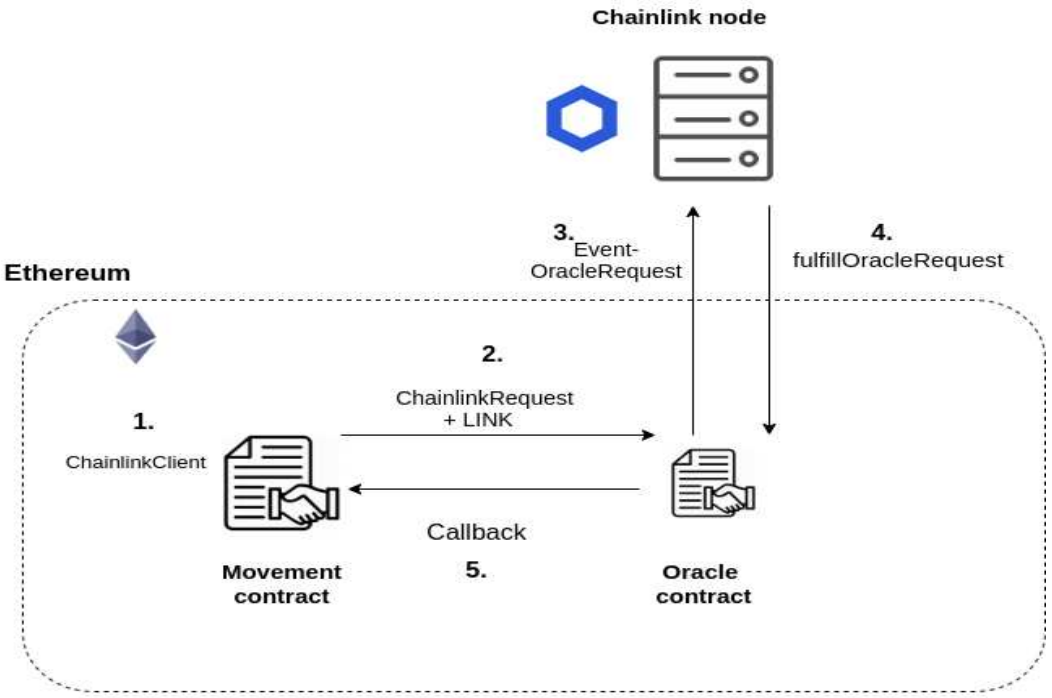
Kako imamo specifikacije poslova sa pripadajućim zadacima, te inicijaliziran *Chainlink* čvor sve je spremno da ga spojimo sa *blockchainom* i našim pametnim ugovorom. To ćemo postići pomoću posebno tipa pametnog ugovora na *blockchainu* naziva **Oracle.** Taj ugovor povezuje naš pametni ugovor sa *Chainlink* čvorom i preko njega ide sva komunikacija te zahtjevi za izvršavanjem unaprijed određenih poslova *(eng.jobs)* . Mi registriramo naš *Chainlink* čvor (tj. njegovu adresu) na taj ugovor, koji nakon registracije čeka na izvršavanje poslova koje je unaprijed specificirao. On se dan od strane *Chainlink* asocijacije i stoji na njihovom *Github* računu.

Opišimo ukratko postupak i etapa izvršavanje *Chainlink* poslova i našeg pametnog ugovora:

1. Naš pametni ugovor je naslijedio svojstva od drugog ugovora imena ***ChainlinkClient*** što mu omogućuje slanje posebno tipa zahtjeva.
2. Kada našem pametnom ugovoru zatrebaju podaci, on šalje ***ChainlinkRequest*** zahtjev na adresu „Oracle" pametnog ugovora u kojem se nalazi svi relevantni podaci (identifikacijski broj posla koje želimo izvršiti i svi njegovi parametri).Također šaljemo mu i identifikator posebne ***Callback*** koja se izvršava na našem pametnom ugovoru kada *Chainlink* čvor ispuni svoj zadatak. Tom funkcijom mi najčešće upisujemo dobivene podatke u svoje varijable. U

transakcijskom pozivu je uključena i LINK digitalna valuta kao naplata za izvršen posao.

3. ***Oracle*** pametni ugovor vidi zahtjev i transferiranu *LINK* digitalnu valutu. On tada emitira događaj *(eng. event)* koji obavještava *Chainlink* čvor izvan blockchaina da mu je zahtjev za posao upućen i on kreće sa izvršavanjem tog posla.
4. Kada *Chainlink* čvor uspješno izvrši sve zadatke iz specifikacije posla koji mu je bio namijenjen, tada zove posebnu funkciju **fulfillOracleRequest** sa rezultatom izvršenog posla.
5. Nakon toga „Oracle" pametni ugovor uzima taj rezultat te poziva **Callback** funkciju na našem pametnom ugovoru koje je bila prije specificirana u početnom **ChainlinkRequest-u.** Tada mi imamo taj podataka i upisujemo ga u svoj pametni ugovor.



Slika 4.6: Dijagram izvršavanje zahtjeva pomoću *Chainlink* protokola

Poglavlje 4

Mobilna i web aplikacija

4.1 Uvod

U ovom poglavlju ćemo opisati druge dvije važne komponente našeg projekta *Movement*. To su mobilna android aplikacija te web *Dapp* frontend aplikacija. Obje aplikacije komuniciraju sa pametnim ugovorom na *Ethereumu*.
Kako implementacija ovih aplikacija i njihovih tehnologija nije tema ovog diplomskog rada nego pametni ugovori i njihova povezanost sa vanjskim svijetom, nećemo stavljati isječke programskog koda tih aplikacije, već ćemo samo opisati njihovu funkcionalnost.
Njihov kod će se moći pronaći na *Github* računu autora ovog diplomskog rada (https://github.com/filipmacek)

4.2 Tehnički opis aplikacija

Opišimo ukratko za svaku aplikaciju pojedinačno njihove tehnologije, najvažnije korištene pakete te ideju i motivaciju za njihov dizajn i funkcionalnost.

Web Dapp aplikacija

Kako bi web aplikacije koje komuniciraju sa blockchainom (tj. pametnim ugovorom kreiranim na blockchainu) bile interaktivne i jednostavnije za izradu programerima, najčešći se koristi popularna *JavaScript* biblioteka *ReactJS*.

ReactJS je biblioteka za kreiranje korisničkog sučelja čije su aplikacije građene od puno malih cjelina koje se zovu komponente. Cijela aplikacija je jedna komponenta koja sadrži puno manji podkomponenti koje zajedno čine kompleksno korisničko sučelje. Za dizajn su najčešće korištene gotove stilizirane komponente iz biblioteka *react-bootstrap* ili *rimble-ui*.

Najvažnija biblioteka koja nam služi za spajanje sa pametnim ugovorom na *blockchainu* je *„web3js"* koja će biti povezana sa *Ethereum-om* pomoću *MetaMask* novčanika.

Sav *JavaScript* programski kod se na kraju skuplja i grupira pomoću alata *webpack* u jednu veliku datoteku koja se onda samo priključi kao skripta u odgovarajuću HTML datoteku. Također aplikacija je upakirana onda u *Docker* kontejner kako bi se lakše moglo s njom upravljati.

Android mobilna aplikacija

Mobilna aplikacija služio korisniku kao glavno pomagalo pri izvršavanju ruta. Možemo ju gledati kao „digitalnog blizanca" u vanjskom svijetu koji predstavlja korisnika i njegove lokacije koji se dobivaju pomoću GPS senzora iz mobilnog uređaja, a pametnom ugovori služio kao izvor informacija i svojevrsni promatrač.

Aplikacija *Movement* je rađena za *Android* operacijski sustav u programskom jeziku *Kotlin* uz pomoć Android Studio kao svog programskog alata. Mobilna aplikacija je napisana u modernom stilu koristeći puno ideja iz funkcionalnog i reaktivnog programiranja kao i pripadnih biblioteka.

Također prati preporučenu MVVM *(eng. „Model,View,ViewModel)* gdje model sadrži kako podaci izgledaju, pogled (*eng. view*) reprezentira korisničko sučelje koje sluša na podatkovne promijene od strane entiteta *eng. ViewModel.* On služi kao poveznica između modela i pogleda, gdje se podaci transformiraju i obrađuju. Pogledi su povezani u fragmente i navigacijom između njih.

Za kreiranje završne pokretačke datoteke i cijele aplikacije koristimo sistem *Gradle.*

4.3 Pregled Dapp web aplikacije

U ovom poglavlju dat ćemo pregled i sve funkcionalnosti naše *Dapp* web aplikacije. Ona će nam biti glavna poveznica sa našim pametnim ugovorom.

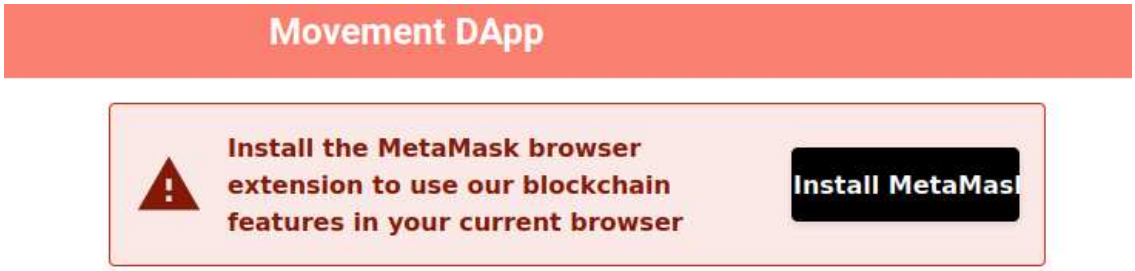
Prilikom prvom otvaranju ove web aplikacije očekujemo dvije stvari da su osigurane i namještene:

1. **MetaMask web novčanika**

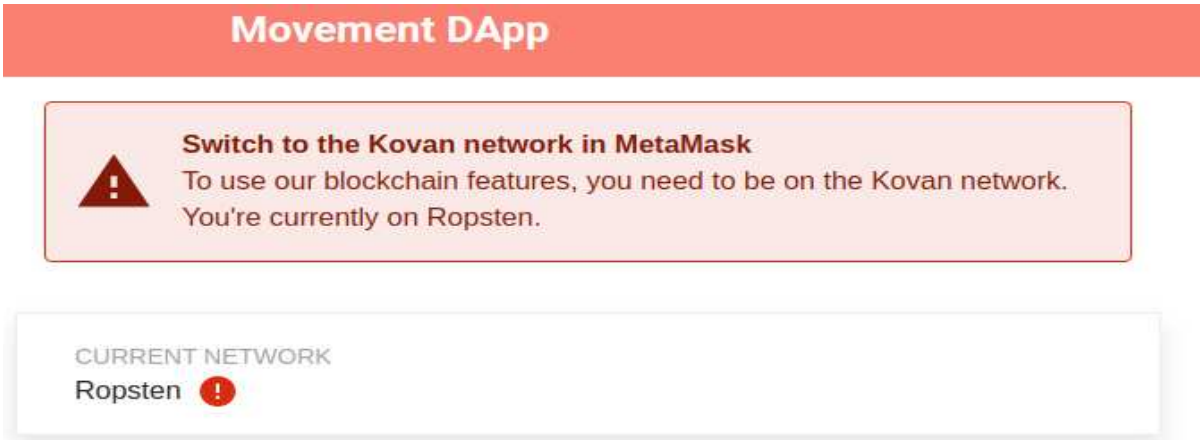
   Potreban za komuniciranje sa Ethereum-om i potpisivanjem transakcija.

2. **Kovan testni Ethereum blockchain**

   Nakon što je *MetaMask* web ekstenzija uspješno instalirana, potrebno se prebaciti na točan blockchain gdje je pametni ugovor kreiran, a to je *Kovan*.

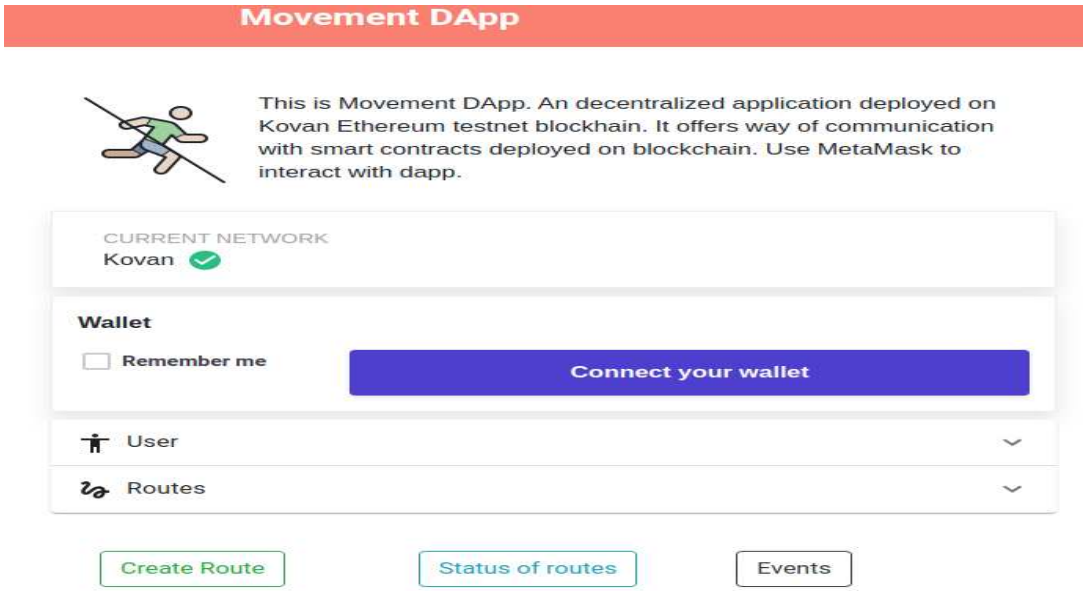


Slika 5.1: Upozorenje web aplikacije da MetaMask novčanik nije instaliran



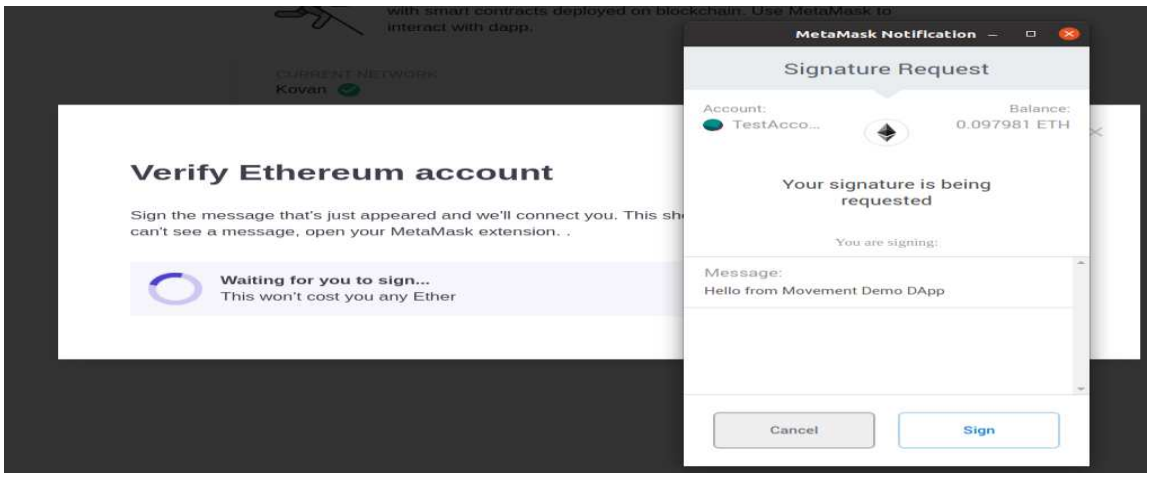
Slika 5.2: Upozorenje web aplikacije da smo na krivom *blockchainu*

Ako su zadovoljena ta dva uvjeta (da imamo *MetaMask* web novčanik i da smo ga inicijalizirali za Kovan *blockchain*) upozorenja se neće pokazati i moći ćemo koristi svu funkcionalnost ove *Dapp* web aplikacije, a time i pametnog ugovora s kojima je ona povezana. Aplikacije će u tom slučaju izgledati ovako:

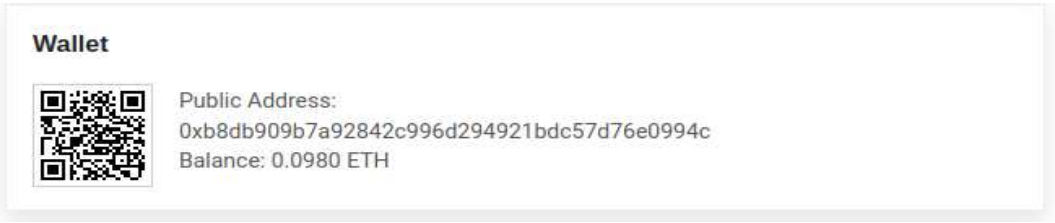


Slika 5.3: *Dapp* web aplikacija

Da bi se *Dapp* web aplikacija u potpunosti povezala sa novčanikom potrebnu ju je direktno spojiti sa *Ethereum* adresom u novčaniku. Inače naša *Dapp* web aplikacija ne zna s kojeg *MetaMask* računa se korisnik spaja (*MetaMask* novčanik može sadržavati više Ethereum računa tj. adresa).Da bismo točno potvrdili s koje adrese se korisnik povezao s našom aplikacijom te da ima vlasništvo nad privatnim ključem te adrese, bit će proslijeđen na potpisivanje jedne proizvoljne poruke. Kada poruka bude potpisana, unijet ćemo adresu u našu aplikaciju i *React* komponenta *Wallet* će promijeniti svoj izgled te će ispisati stanje zapamćenog računa, QR kod i adresu.



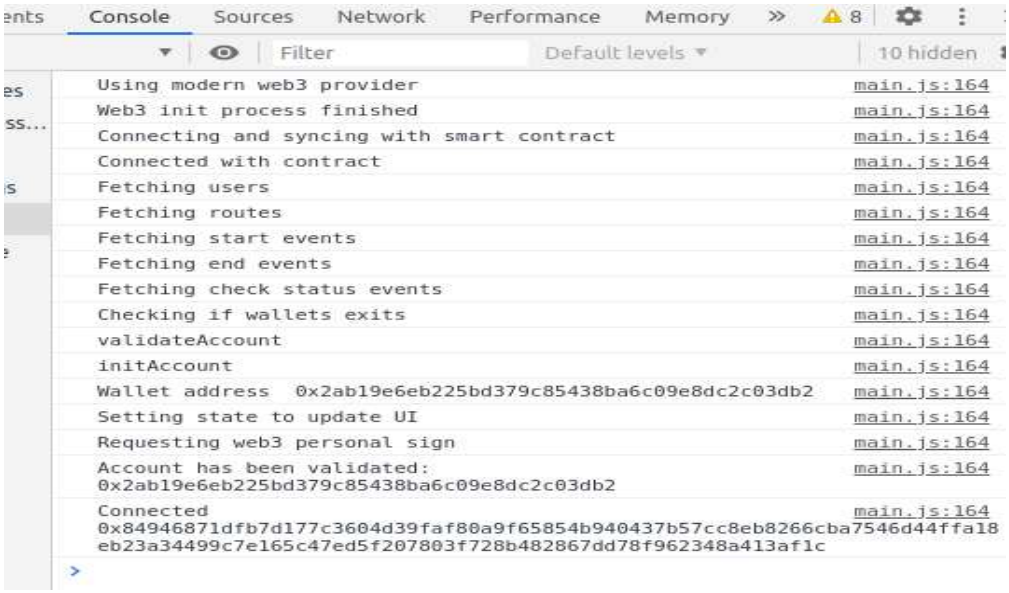
Slika 5.4: Povezivanje MetaMask računa sa aplikacijom



Slika 5.5: Verificiran *Ethereum* račun

Povezivanje sa pametnim ugovorom

U pozadini se događaju mnogi procesi i inicijalizacije kako bi naša aplikacija bila u potpunosti povezana sa pametnim ugovorom *Movement* na blockchainu. Aplikacija posjeduje adresu tog pametnog ugovora i sve njegove funkcije i specifikacije kroz *ABI (eng. Application Binary Interface)*. Cijeli proces se najbolje može vidjeti kroz zabilješke web preglednika.



Slika 5.6: Zabilješke iz Dapp web aplikacije

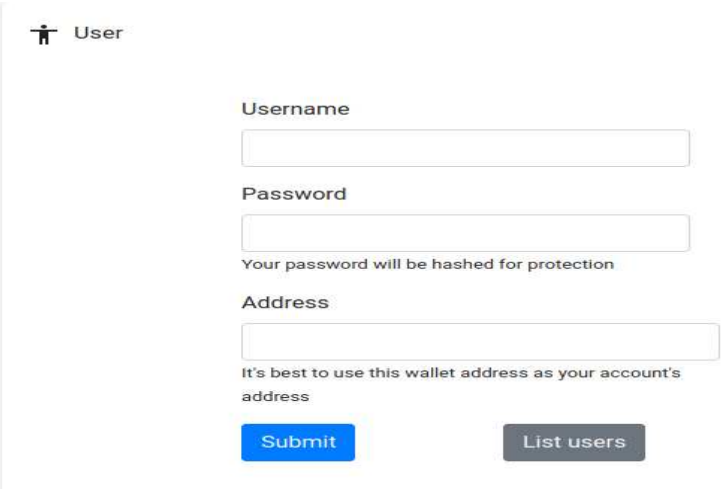
Time smo u potpunosti povezali našu *Dapp* web aplikacijom i sa *Ethereum* blockchainom i sa korisnikovim računom, s kojim ćemo i potpisivati sve potrebne transakcije.

Sva komunikacija sa pametnim ugovorom se onda izvršava pomoću biblioteke „*web3js*" koja ima sve informacije i pregled o varijablama i funkcijama našeg pametnog ugovora.

Prođimo sada ukratko sve ostale funkcionalnosti koje nam se otvaraju kada je aplikacija potpuno povezana i inicijalizirana.

1. **Registracija korisnika**

   Registracija korisnika izvršava se kroz jednostavno sučelje koje onda upisuje kroz pametni ugovor na *blockchain*. Na zaporku se dva put primjenjuje *SHA256 hash* funkcija radi sigurnosti jer su podaci na *blockchainu* javni. Ako je korisnik već registriran forma za registraciju se neće pokazati već ćemo vidjeti da je korisnik već registriran i da pametni ugovor posjeduje informacije o napravljenim rutama korisnika.

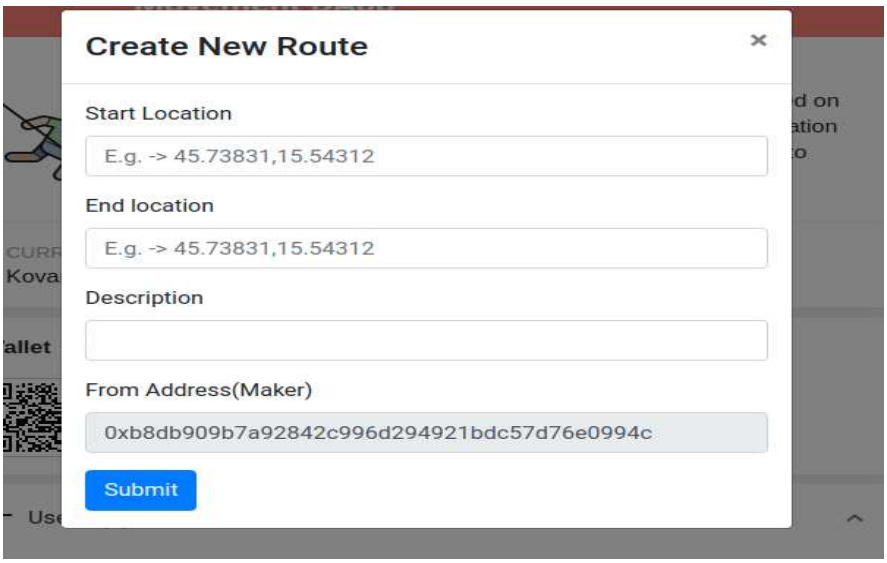


Slika 5.7: Dio korisničkog web sučelja za registraciju

Slika 5.8: Dio korisničkog web sučelja kada je korisnik već registriran

2. Kreiranje rute

Kreiranje ruta se provodi pritiskom na gumb *„Create Route"* nakon kojeg se otvara jednostavni *modal* u kojem se upisuje sve bitne informacije o ruti. Potrebni su nam početna lokacija i krajnja lokacija, a očekujemo da je lokacija prikazana u formatu geografske širine (*eng. latitude)* i dužine *(eng. longitude)* u jednom *string* tekstu odvojene zarezom. Također dodajemo jednostavan opis rute. Rute može svatko kreirati i upisati na blockchain ako posjeduje novčanik i digitalnu valutu Ether.



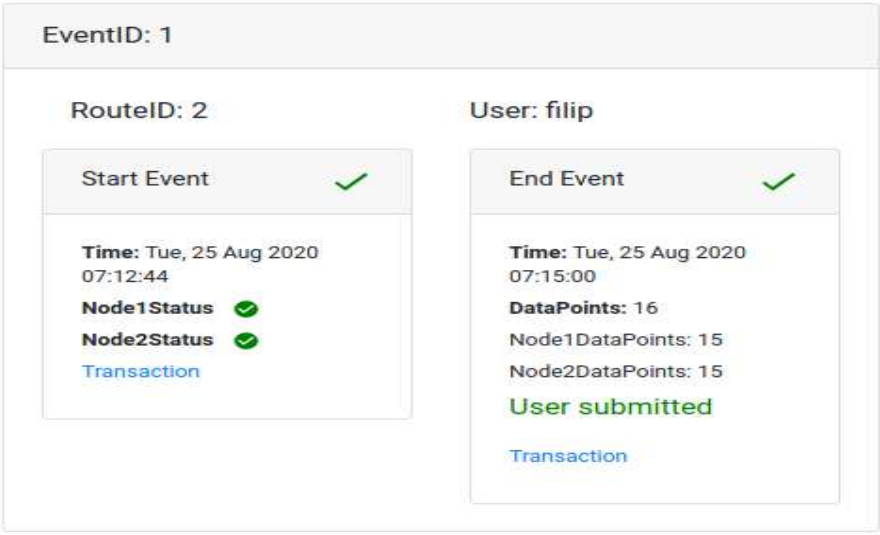
Slika 5.9: Web sučelje za kreiranje rute

3. Događaji

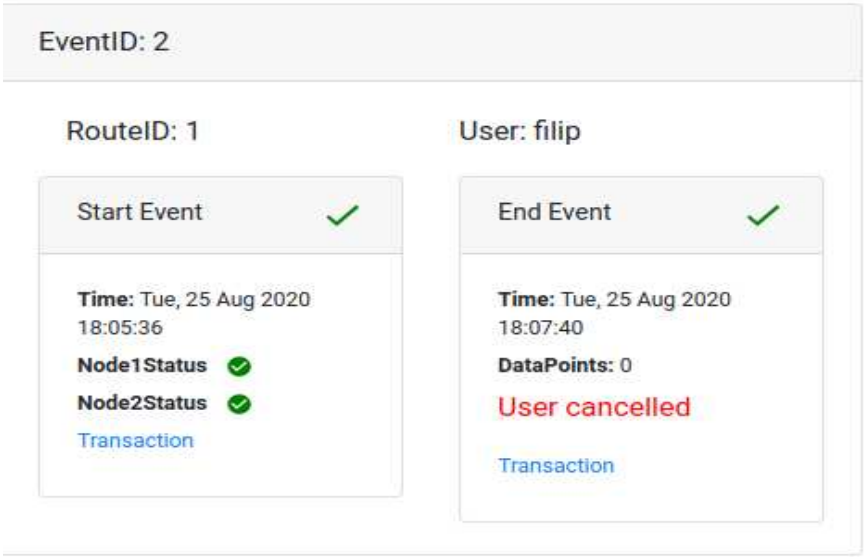
Kao što smo već objasnili u poglavlju o implementaciji našeg pametnog ugovora, postoje dva tip događaja od strane korisnik koje se tiču njegov izvršavanje rute. Prvi je korisnikovo prihvaćanje i pokretanje rute implementirano kao varijabla **StartRouteEvent**, a drugi je **EndRouteEvent**.

One služe našem pametnom ugovoru kako bi znao u kojem stanju je ruta i kako ju je korisnik izvršavao. Kada ju je započeo , je li ju je nakon prekinuo ili je dovšena do kraja. Ako ju se dovršio, pametni ugovor nastavlja sa etapama provjere i pokreće svoju funkciju **requestRouteStatus** koji onda već uz opisan postupak pokreće validaciju od strane dva Chainlink čvora.

Pritiskom na gumb *„Events"* otvara se *modal* za svih upisanim korisničkim događajima. Prikažimo slikama kada je izgledalo jedno uspješno izvršavanje rute, te jedno neuspješno gdje je korisnik prekinuo rutu iako ju nije dovršio.



Slika 5.10: Pregled uspješnog korisnikovog izvršavanje rute



Slika 5.11: Pregled korisnikovog prekidanja rute

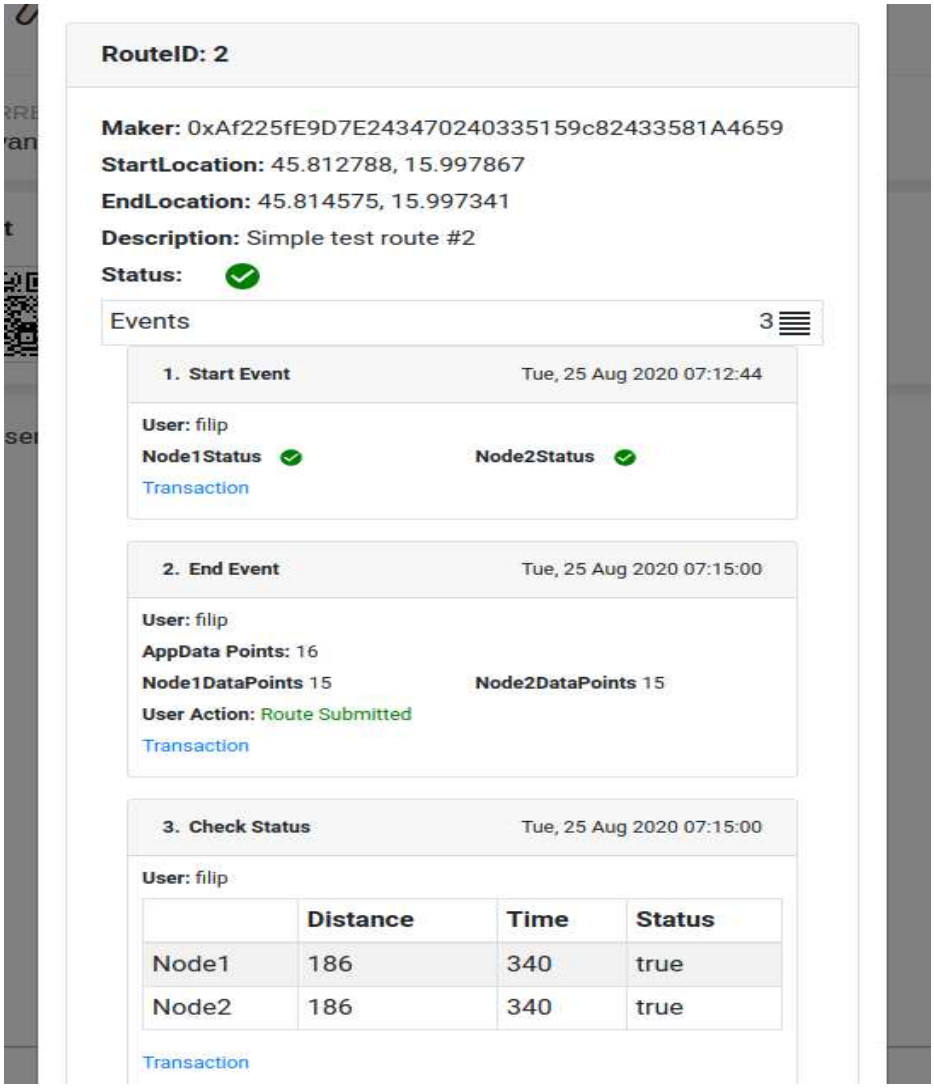
4. Status ruta

Pritiskom na gumb **„Status of routes"** otvara se *modal* sa popisom svih kreiranih ruta i njihovim stanjima: je li ruta izvršena i provjerena ili je slobodna za izvršavanje. Također ispod svake rute nalazi se popis već opisanih događaja za tu rutu te link na svaku transakciju koja se može pogledati u *eng „block exploreru"* za *Kovan* blockchain.

Ponovimo još jednom cijeli postupak sa slikom na primjeru završene rute i odgovora o statusu od strane *Chainlink* čvorova.

1. **Start Event** – korisnik je započeo izvršavanje rute, upisujemo također status dva *Chainlink* čvora, tj. njihovih *Data Adaptera*, jesu li bili aktivni i počeli spremati lokacijske podatke
2. **End Event** – korisnik je uspješno dovršio izvršavanje, upisujemo ukupnu količinu lokacijskih podataka, te koliko ih je svaki *Chainlink* čvor primio
3. **Check Status** – pametni ugovor je zatražio status rute i dodatne informacije o vremenu i udaljenosti od dva *Chainlink* čvora pomoću zahtjeva na blockchainu. Upisujemo što je svaki čvor odlučio (koji podataka je upisao na blockchain.

   Napomena : Oba *Chainlink* čvora ,iako različita, koriste isti programski kod za svoj *Data Adapter* pa će i prijaviti isti rezultat. To nije problem, jer je cilj bio pokazati kako pametni ugovor koristi dva izvora za informaciju iz vanjskog svijeta o stanju rute, i time je decentralizacija obrade podataka veća.

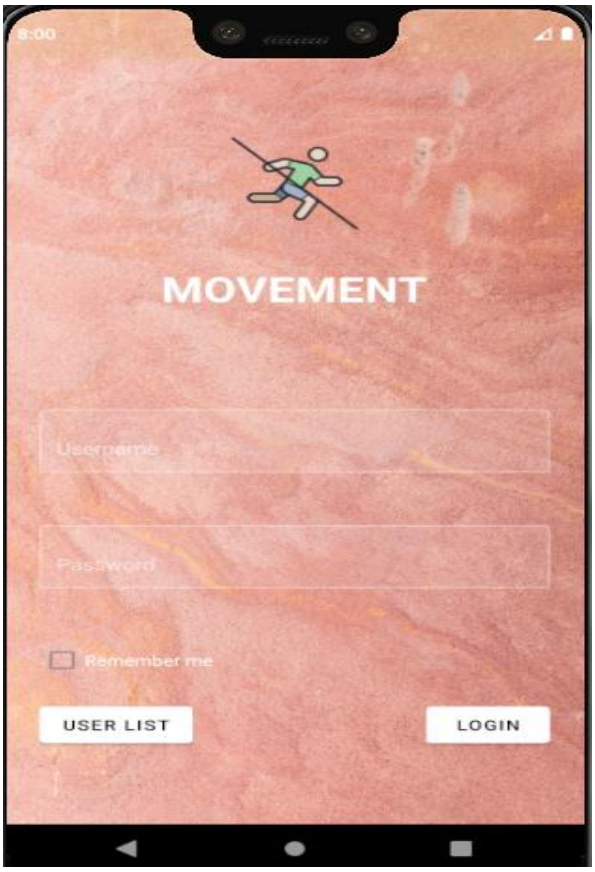


Slika 5.12: Web sučelje o statusu rute sa detaljima i pripadnim događajima

5.4 Pregled mobilne android aplikacije

Mobilna *Android* aplikacija služit će korisniku kako bi mogao izvršavati rute te kako bi mogao odašiljati lokacijske podatke prilikom prelaska rute.

Nakon što se korisnik registrirao preko *Dapp* web aplikacije, pametni ugovor na blockchainu ga je spremio u svoj niz spremljenih korisnika. Takav korisnik tada ima mogućnost se pristupiti u mobilnu aplikaciju pomoću korisničkog imena i zaporke koju je unio. Također da bi se aplikacija uspješno dobivala lokacijske podatke potrebno joj je dati dopuštanje. Taj zahtjev će se otvoriti prilikom prvog otvaranja aplikacije.



Slika 5.13: Početni prijavni ekran mobilne aplikacije

Aplikacija se prilikom pokretanje povezuje preko *Infura HTTP Ethereum* klijenta na *Kovan* testni *blockchain* i uzima sve potrebne podatke od pametnog ugovora i prima ih u unutrašnju memoriju.

Nakon što je korisnik upisao odgovarajuće korisničko ime i zaporku, program provjerava je li se podaci podudaraju s onim dobivenim iz pametnog ugovora. Ako se podudara korisničko ime i dva puta hashirana zaporka, aplikaciju potvrđuje identitet i uvodi korisnika u novi Android fragment koji služio korisniku kao nadzorna ploča *(eng. dashboard)*.

Tamo se nalaze tri pogleda/fragmenta koje korisnik može pokretima prstom promijeniti:

1. **Routes**

   Popis svih ruta sa početnim i krajnjim lokacijama, njihovim stanjem te jednostavnim objašnjenjem. Ako korisnik želi vidjeti gdje se nalaze te lokacije, to može napraviti pritiskom na tipku *„INFO"* gdje mu se otvara *„Google Maps"* program sa označenim lokacijama. Ako želi započeti rutu pritisnut će tipku „ACCEPT"
2. **Nodes**

   Prikazuje registrirane *Chainlink* čvorove *(eng. nodes)* sa ostalim bitnim informacijama, kao što su IP adresa servera, *URL endpoint* (dodaje se na IP adresu kako bi znali di šaljemo lokacijske podatke), koliko su ruta provjerili te informaciju je li taj čvor/server aktivan (ako jest pokazat će se zelena boja, ako nije crvena)
3. **Account**

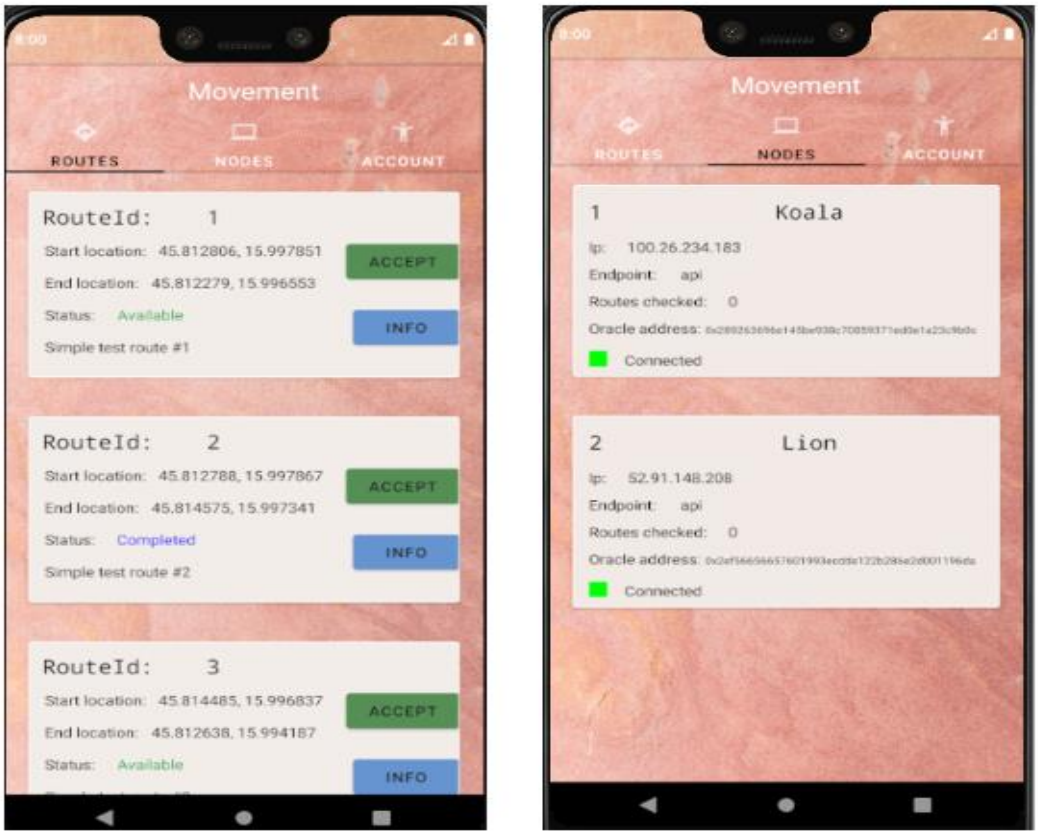
   Sadrži osobne informacije o korisniku i njegovim rutama.

Agent pametnog ugovora

Poseban entitet u mobilnoj web aplikaciji koji je zadužen za upisivanje korisničkih radnji na pametni ugovor. On upisuje već opisane **StartRouteEvent, EndRouteEvent** tipove podataka i pokreće **RequestRouteStatus** kako bi *Chainlink* čvorovi provjerili i u upisali rezultat obavljene rute od strane korisnika.

Pozivanje funkcija pametnog ugovora, a time i potpisivanje pripadajuće transakcija agent radi uz pomoć privatnog ključa kojeg smo upisali u posebnu datoteku koji biva uključena pri kompilaciji ove Android aplikacije da bi ostala tajna.

Da bi pametni ugovor dao dopuštenje agentu za izvršavanje tih funkcija upisali smo agentovu adresu u globalnu varijablu pametnog ugovora.



Slika 5.14: Korisničko sučelje nakon što se korisnik uspješno prijavio

Proces izvršavanja rute

Nakon što je korisnik odabrao rutu i prihvatio je, otvara se poseban *Android* fragment te se u pozadini pokreće lokacijski proces koji će našoj aplikaciji dostavljati promjene korisnikovih lokacija. Mi ćemo kod svake promijene, novu lokaciju poslati na *IP* adresu *Data Adaptera* od *Chainlink* čvora. Opišimo dijelove tog pogleda:

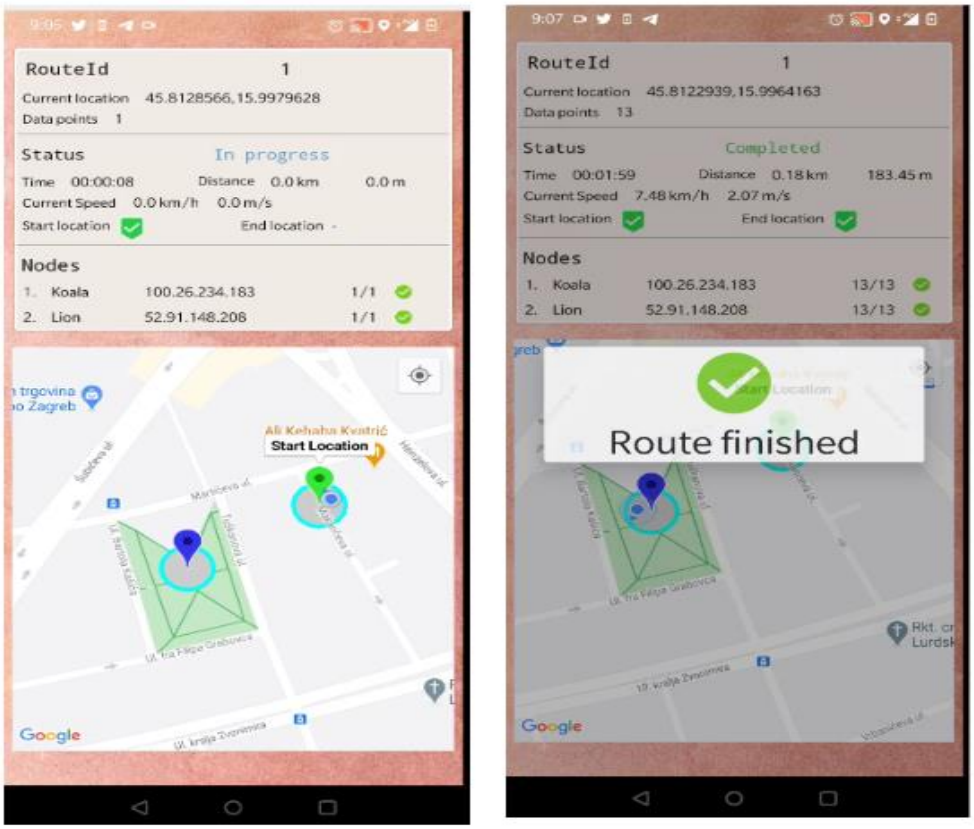
1. **Komandna ploča**

   Komanda ploča je glavni alat na pomoću kojeg će korisnik vidjeti trenutno stanje rute i ostale bitne informacije:

   - Trenutačnu lokaciju i broj lokacijskih promjena dosad
   - Stanje rute, vrijeme, prijeđenu distancu i trenutnu brzinu
   - Informaciju je li posjećena početna i krajnja lokacija
   - Informaciju jesu li *Chainlink* čvorovi aktivni, njihova imena, ip adrese, te koliko lokacijskih podataka su primili

2. **Mapa**

Google mapa koji sadrži početnu i krajnju točku te trenutnu lokaciju. Također oko svake lokacijske točke nalazi se obojana kružnica polumjera 25 metara što smo označili kao uvjet da je korisnik posjetio tu točku



Slika 5.15: a) početak izvršavanja rute b) kraj izvršavanje rute

Aplikacija će uz posjećivanje početne, a potom i krajnje lokacije, obavijestiti korisnik da je završio rutu, da su svi podaci poslani na Data Adaptere od *Chainlink* čvorova te će upisati **EndRouteEvent** na *blockchain* kao znak da je korisnik završio rutu do kraja. Pametni ugovor će onda po već opisanom postupku nastaviti sljedeće etape provjere stanja ruta

.

Poglavlje 5

Zaključak i komentar projekta

5.1 Uvod

U ovom poglavlju ćemo dati cjelokupni zaključak cijelog projekta. Komentirati ćemo ukratko ideju projekta, bavit ćemo se i sigurnosnim pitanjima i opaskama. Usporediti ćemo ga sa tradicionalnim centralnim modelom aplikacije i komentirati koje su prednosti i mane u usporedbi na našom implementacijom pametnog ugovora na *blockchainu*.

5.2 Sigurnosni aspekt

Kako je *blockchain* javan i otvoren svim sudionicima da vide sve transakcije i informacije o pametnim ugovorima, to uzrokuje mnoge opasnosti s kojima se tvorci pametnih ugovora mogu susretati. Time pametni ugovori moraju biti dobro smišljeni i dizajnirani u cilju maksimalne sigurnosti i pouzdanosti. To uključuje pomno planiranje i osmišljavanje implementacije. Završna faza je temeljito testiranje na testnom *blockchainu* i traženje sigurnosni rupa i vektora koje bi mogle kompromitirati cijeli pametni ugovor. Tek nakon toga možemo pustiti pametni ugovor u produkciju na glavni *blockchain* gdje se transakcija i interakcije sa pametnim ugovorom plaćaju pravom digitalnom valutom.

Također korisnici takvih pametnih ugovora su izloženi mnogim rizicima i opasnostima koje takva tehnologija ima u svojoj suštini. Primjeri prevara i krađa privatnih ključeva od računa, a time i digitalnih valuta su česte. Sve to uključuje da su aplikacije na *blockchainu*

trenutno namijenjene za uski krug ljudi koji su tehnološki vješti i dobro poznaju ovaj sistem.

Ne postoji previše primjera aplikacija na *blockchainu* koju koristi veća masa ljudi baš zbog svih nabrojenih teza. Time i ne možemo računati na to da će naša aplikaciju koristiti veći broj ljudi ako ju konstruiramo uz pomoć pametnog ugovora na *blockchainu*.

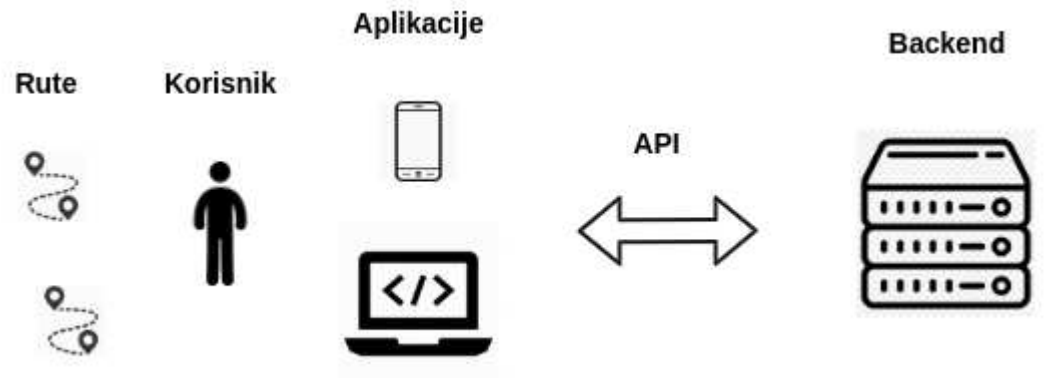
5.3 Alternativna implementacija projekta

Cijeli ovaj projekt složen je od puno različitih dijelova koji svaki imaju svoju specifičnu funkciju. Dizajnirati sve te dijelove i povezati ih u jednu koherentnu cjelinu je bio veliki izazov za autora ovog rada. Ali slaganjem i povezivanjem moglo se duboko prodrijeti u srž problema i svih pojedinosti koje ovakav projekt nosi.

U poglavlju **2.3** ***Opis projekta*** najbolje se može vidjeti kompleksnost ovog projekta i rad potreban kako bi se spojili sve ti dijelovi. U središtu dizajna ovog projekta je pametni ugovor na *Ethereumu*, koji je i glavna baza podataka te sve druge komponente zapravo komuniciraju s njim i njemu podnose krajnji rezultat.

Ideja je i bila prikazati ovako neku složenu arhitekturu pomoću novih tehnologija u kojima se promiče ideja decentralizacije i prepuštanja ovlasti za izvršavanje pametnom ugovoru na blockchainu koji me ne kontroliramo.

Pokušajmo sada zamisliti kako bi ovaj projekt mogao izgledati na drugačijim temeljima i dizajnu, imajući na umu tradicionalan model gdje bi u centru bio naš veliki server *(eng. backend)* koji bi komunicirao pomoću API-ja *(eng. Application Programming Interface)* sa mobilnom i web aplikacijom.



Slika 5.1: Alternativna implementacija projekta

Na *slici 5.1* najbolje se može vidjeti ideja takve tradicionalne implementacije koja je u suštinskoj razlici sa našom implementacijom vidljivom **na *slici 2.5* u poglavlju *Opis Projekta.***

Zapravo je ovo klasični model na kojem je temeljna skoro svaka aplikacija koje ne uključuje *blockchain* ,a takve su većina. Jer svaka kompanija mora i želi voditi brigu o svojim serverima i procesima. Naša implementacija ovog projekta je zapravo alternativni model koji još nije postigao svoju globalnu upotrebu i funkcija, a ni široku upotrebu.

Da bi bolje opisali kakva bi bila motivacija sudionika da napravi svoj projekt po jednom ili drugom modelu, opišimo sve pozitivne strane, a potom i negativne .

Pogodnosti i korisnost koju bi pružala implementacija aplikacija na centralnom modelu gdje mi upravljamo infrastrukturom su:

- ✓ **Veća kontrola**

  Sva kontrola i svi procesi i njihova implementacija su prepušteni nama. Mi moramo osigurati potreban hardver i infrastrukturu. Možemo to implementirati na oblaku ili na svojim lokalnim računalima.

- ✓ **Veća sigurnost za tvorce aplikacija**

  Mi ne prepuštamo kontrolu pametnom ugovoru na *blockchainu* koji se autonomno izvršava po zadanoj logici, već mi imamo apsolutno kontrolu i mi pokrećemo sve procese i radnje.

- ✓ **Privatnost**

  Naš programski kod i implementacija je naše privatno vlasništvo, i time se štitimo od mnogih rizika koje bi moglo nastati ukoliko bi naš programski kod bio javan kao u slučaju pametnih ugovora. Kako je *blockchain* je javan, svi mogu čitati njegove podatke i upisivati nove, također je i kod pametnog ugovora java pa se vidi i njegov dizajn Time otkrivamo cijelu implementaciju našeg projekta i izlažemo se mnogim rizicima.

- ✓ **Intelektualno vlasništvo**

  Kako ne otkrivamo programski kod ni implementaciju, štitimo se od konkurenata jer nas oni ne mogu tako lagano prekopirati.

Prebacimo se sada na gledište korisnika i nabrojimo sada neke nedostatke i zamjerke koje bi korisnik mogao imati u usporedbi da bi ta ista aplikacija bila napravljena pomoću pametnih ugovora pomoću decentralizirane *blockchain* tehnologije.

- ➢ **Centralna kontrola**

  Svu kontrolu prepuštamo centralnom serveru i kompaniji koja ga održava. To je u suprotnosti sa modelom koji decentralizirani pametni ugovori zagovaraju.

- **Nejednakost**

  Mi nemamo kontrolu na serverima i procesima, koji nisu upravljani od strane nepristranog agenta već su kontrolirani centralnim entitetom. Time nemamo ista prava ni ovlasti.

- **Mogućnost diskvalifikacije**

  Kako nemamo kontrolu na procesima, time se dovodimo u mogućnost i prekida suradnje ili blokiranje našeg računa ako centralni server odluči tako. Odluka može biti opravdana i neopravdana, dok će se izvršavanje pametnog ugovora nastaviti bez obzira na sve jer nitko nema kontrolu nad *blockchainom*.

- **Implicitno povjerenje**

  Mi implicitno moramo vjerovati centralnog serveru i kompaniji da će izvršiti ono što je dogovoreno.

5.4 Zaključak

U ovom poglavlju smo predstavili i drugačiji model kako je naš projekt mogao biti implementiran. Također smo opisali sve moguće opasnosti koje bi implementaciju pametnog ugovora na *blockchainu* mogla nositi. Mi se u ovom diplomskom radu ne pokušavamo prikloniti jednoj ili drugoj implementaciji, već sa objektivnog gledišta opisujemo i promatramo sve dobre i loše strane.

Decentralizacija ima svoju veliku korisnost, ali sa sobom nosi slabiju performansu takvih sustava koji ne mogu biti tako efikasni i primjenjivi za široku masu ljudi. Trenutačno je prosječno vrijeme za potvrdu transakciju za *Ethereum blockchainu* oko 15 sekundi, a može i više biti ako se *blockchain* bude posebno aktivan u raznim periodima dana. To je u čistoj suprotnosti koju današnji korisnici očekuju. Oni su navikli na trenutne aplikacije sa minimalnim zastojem u čekanja i procesiranju.

Također korisnik mora sam plaćati transakcijske troškove na *blockchainu* te se mora naučiti rukovati sa digitalnim novčanikom na kojemu su spremljene digitalne valute.

S druge strane pogodnosti koje *blockchain* arhitektura nudi su također primamljive i imaju svoju korisnost. Od toga da je sustav decentraliziran, ne ovisi o centralnom entitetu pa do toga da nitko ne može zabraniti ni blokirati izvršavanje transakcija i pametnih ugovora.

Za kraj vidimo da svaki tip dizajna aplikacija ima svoje dobre i loše strane, te je na kreatoru aplikacije da sam odlučio što mu je potrebno i čega se spreman odreći ako se odluči koristiti decentraliziranu arhitekturu kao što je *blockchain*.

Bibliografija

[1] David Lee Chaum, *Computer Systems Established, Maintained and Trusted by Mutually Suspicious Groups,* (1982.) University of California, Berkley

[2] Satoshi Nakamoto, Bitcoin: A Peer-to-Peer Electronic Cash System, (2008.) https://bitcoin.org/bitcoin.pdf

[3] Ethereum, whitepaper: https://ethereum.org/en/whitepaper/

[4] Chainlink, whitepaper: https://link.smartcontract.com/whitepaper

Sažetak

Pametni ugovori kao vrsta digitalnog i računalnog izvršavanje neke njemu specificirane programske logike mogu poslužiti u raznim primjenama gdje sudionici ne vjeruju jedan drugome i potreban im je posrednik ili decentralizirani medij kao blockchain gdje nitko neće imati kontrolu, a pametni ugovor će se svejedno moći izvršiti. Pametni ugovori imaju usku funkcionalnost ako se samo bave onim što se događa na blockchainu, ali ako ih želimo povezati na vanjskim svijetom potreban je entitet imena Oracle, koji služi kao poveznica sa informacijama iz vanjskog svijeta i operacijama koje se jedino mogu izvršavati izvan blockchaina (npr. slanje novca preko bankarskog sustava itd). Mi smo koristili Chainlink decentraliziranu Oracle mrežu i njihovu tehnologiju da bi povezali pametne ugovore sa vanjskim svijetom.

Kreirali smo mobilnu i web aplikaciju te ih povezali sa pametnim ugovorom kreiranim na testnom Kovan Ethereum blockchainu. Funkcionalnost tog pametnog ugovora je bila da registrirani korisnicu mogu prelaziti zadane rute te biti sigurni da će se njihovi lokacijski podaci nastali prilikom izvršavanje rute na siguran način obraditi i biti upisani na blockchain pomoću Chainlink čvorova tj. operatora koji su uz pomoć dodatnog programa (eng. Data Adaptera) procesirali i obradili te lokacijske podatke.

Summary

Smart contract is a computer program which is intended to automatically execute some form of programing logic which can be used by all sort of participants who dont trust each other and need some third-party entity or decentralized blockchain that can execute this type of contract. Unfortunately, because of their underlying consensus protocols , the blockchains on which smart contracts run cannot support native communication with external systems. Today, the solution to this problem is to intruduce a new functionality, called an oracle, that provides connectivity to the outside world. Existing oracles are centralized services. Any smart using such services has a single point of failure, making it no more secure than a tradiotional , centrally run digital agreement. We will be using Chainlink,a decentralized Oracle network with two Chainlink nodes as our data providers on blockchain.

Also we created a mobile and web application that are connected with our smart contract on Kovan Ethereum testnet blockchain. Our smart contract called „Movement" will be responsible for managing registered routes with start and end location that users will be able to solve. To get status of these routes, smart contract will have two Chainlink nodes which will process and store location data about users and routes that they will get from an Android mobile application.

Životopis

Filip Maček rođen je 19. srpnja 1994. godine u Zagrebu. Školovanje je započeo u Osnovnoj školi Dr. Ivan Merz te je nakon toga upisao srednju opću 2. Gimnaziju. Nakon završetka, 2013. godine upisuje sveučilišni preddiplomski studij matematike na Prirodoslovno-matematičkom fakultetu u Zagrebu. 2016. godine završava preddiplomski studij te upisuje Diplomski studij Financijske i poslovne matematike na istom fakultetu.

Područja interesa su mu geopolitika, makroekonomija, trgovanje i investiranje na financijskim tržištima, blockchain tehnologija, računala i programski jezici.