Sustav za pohranu podataka s tehničkog pregleda vozila korištenjem pametnih ugovora na blockchainu

Turbeki, Dalibor

VELEUČILIŠTE U BJELOVARU
PREDDIPLOMSKI STRUČNI STUDIJ RAČUNARSTVO

Sustav za pohranu podataka s tehničkog pregleda vozila korištenjem pametnih ugovora na blockchainu

Završni rad br. 10/RAČ/2022

Dalibor Turbeki

Bjelovar, listopad 2022.

obrazac ZR - 001

Veleučilište u Bjelovaru

Trg E. Kvaternika 4, Bjelovar

1. DEFINIRANJE TEME ZAVRŠNOG RADA I POVJERENSTVA

Student: **Dalibor Turbeki** JMBAG: **0314020673**

Naslov rada (tema): **Sustav za pohranu podataka s tehničkog pregleda vozila korištenjem pametnih ugovora na blockchainu**

Područje: **Tehničke znanosti** Polje: **Računarstvo**

Grana: **Programsko inženjerstvo**

Mentor: **Ivan Sekovanić, mag.ing.inf.et comm.techn.** zvanje: **predavač**

Članovi Povjerenstva za ocjenjivanje i obranu završnog rada:

1. **Tomislav Adamović, mag. ing. el., predsjednik**
2. **Ivan Sekovanić, mag.ing.inf.et comm.techn., mentor**
3. **dr.sc. Zoran Vrhovski, član**

2. ZADATAK ZAVRŠNOG RADA BROJ: 10/RAČ/2022

U sklopu završnog rada potrebno je:

1. Analizirati i opisati mogućnost pohrane podataka na blockchain korištenjem pametnih ugovora
2. Izraditi i opisati klijentsku aplikaciju za pohranu i provjeru informacija o vozilu
3. Predložiti i opisati sadržaj pametnih ugovora korištenih za potrebe sustava za pohranu podataka s tehničkog pregleda vozila
4. Izraditi pozadinsku (engl. backend) programsku podršku za pohranu i provjeru informacija o vozilu korištenjem pametnih ugovora na blockchainu

Datum: 29.08.2022. godine

Mentor: **Ivan Sekovanić, mag.ing.inf.et comm.techn.**

REPUBLIKA HRVATSKA 2 BJELOVAR VELEUČILIŠTE U BJELOVARU

Sadržaj

1. UVOD

Transakcije na internetu su se dosada odvijale samo preko posrednika. Posrednik je bio centraliziran sustav koji je imao punu kontrolu nad svim transakcijama. Centraliziranost čini sustav nepovjerljivim za obavljanje osjetljivih transakcija zbog mogućeg utjecaja drugih aktera. Najbolji primjer su bankovne transakcije. Trenutno, nitko ne može tehnički zaustaviti banku da promijeni iznose stanja bankovnih računa pojedinaca, obriše obavljene transakcije ili nešto treće. Naravno, to bi dovelo do legalnih posljedica, ali i dalje ostaje ta mogućnost.

Krajem 2008. objavljen je članak *Bitcoin: A Peer-to-Peer Electronic Cash System* koji postaje bijeli papir za Bitcoin *blockchain*. Napisao ga je "otac *blockchain* tehnologije" pod pseudonimom Satoshi Nakamoto. U članku [1] je predstavio rješenje kako izbaciti posrednika u novčanim transakcijama. Predložio je spajanje postojećih tehnologija kao što su kriptografija i *peer-to-peer* protokol s novim konsenzus protokolom dokaz o radu (engl. *proof-of-work*). Nove transakcije se vremenski označavaju po vremenu njihovog odobrenja konsenzus protokolom te se zatim stavljaju na kraj postojećeg lanca transakcija. Ulančane transakcije čine javnu knjigu transakcija (engl. *ledger*). *Blockchain* je distribuiran pa su zato potrebni čvorovi (engl. *node*) koji obavljaju potrebne operacije i spremaju povijest transakcija. Također je i decentraliziran što znači da ni jedan čvor nema mogućnost individualnog utjecaja na cijeli *blockchain*.

Satoshijev Nakamotov Bitcoin eksperiment je usmjerio pozornost programera na dio koji definira distribuirani konsenzus. *Blockchain* tehnologija proširuje svoju originalnu primjenu te danas postaje sve atraktivniji izbor za izradu aplikacija. Jedna od najpopularnijih takvih *blockchain* mreža je Ethereum. Ethereum nudi mogućnost pohrane programskog koda koji se nalazi i izvršava na čvorovima mreže. Taj programski kod se naziva pametnim ugovorom (engl. *smart contract*).

U ovome radu će biti napravljen pametni ugovor koji će predstavljati sustav za pohranu podataka s tehničkog pregleda vozila. Sustav će služiti kao siguran i pouzdan izvor informacija o vozilima. Jednom zapisana informacija se ne može izbrisati iz povijesti. Time se stvara sustav u kojemu povjerenje prema čovjeku nije bitno jer to povjerenje garantira sam sustav.

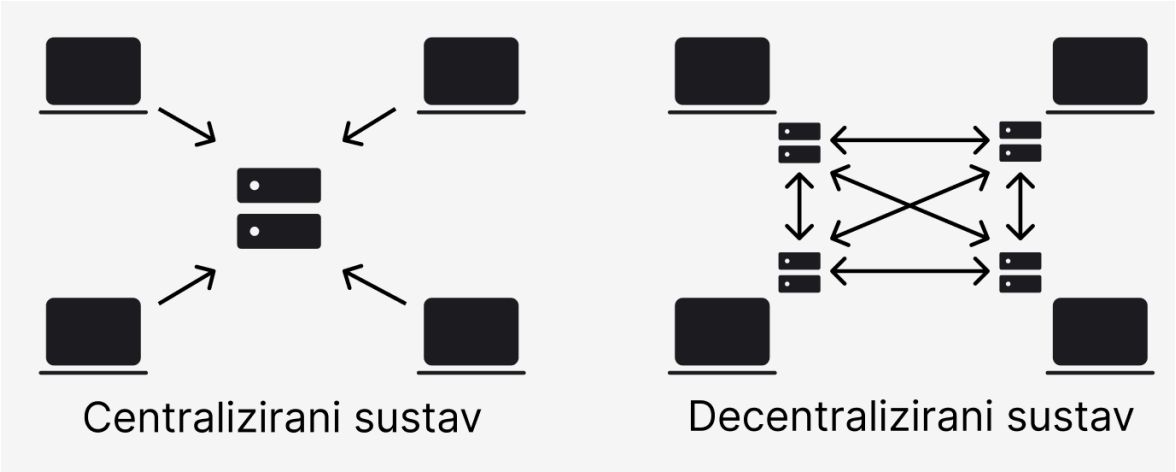
Blockchain tehnologija će biti opisana u sljedećem 2. poglavlju. U 3. poglavlju detaljno će se opisati *blockchain* mrežu Ethereum. U 4. poglavlju će biti predstavljen programski jezik Solidity koji se koristi za razvoj pametnih ugovora. Opisati će se i razvojno okruženje Remix IDE. Kako bi se aplikacija upotpunila, u 5. poglavlju će biti opisan Angular u kojemu je napravljeno korisničko sučelje. Sam projekt će biti opisan u 6. poglavlju. Kratak zaključak dan je u poglavlju 7.

2. BLOCKCHAIN

Blockchain je javan, distribuiran i decentraliziran sustav pohrane podataka u obliku transakcija. Sustav se naziva javnom knjigom transakcija na koju se podaci zapisuju u stvarnom vremenu preko konsenzus protokola koji se pokreće na čvorovima mreže. Nad transakcijama se pravi hash vremenskim pečatom te se kodiraju u Merkleovo stablo kao što je detaljnije objašnjeno u članku [2].

2.1. Decentraliziranost

Blockchain nije centralno kontroliran. Nepostojanje centralnog autoriteta stvara sustav u kojemu povjerenje nije bitno. Povjerenje je zamijenjeno konsenzus protokolima kao što su dokaz o radu ili dokaz o udjelu (engl. *proof of stake*). Protokoli preko rudara (engl. *miner*) ili validatora (engl. *validator*) provjeravaju transakcije, ukoliko su transakcije označene kao validne zapisuju se u javnu knjigu transakcija. Na slici 2.1 je prikazana razlika između centraliziranog i decentraliziranog sustava. Padom centralnog servera svi korisnici gube mogućnost pristupa podacima. Decentralizirani sustav i padom nekoliko čvorova nastavlja dalje normalno raditi.



Slika 2.1: Razlika između centraliziranog i decentraliziranog sustava

2.2. Distribuiranost

Blockchain sustav se ne nalazi na jednom serveru već je raspršen po čvorovima. Distribuirana baza podataka se nalazi u stanju za koje su se čvorovi sporazumno odlučili. Podaci moraju biti sinkronizirani kroz cijelu mrežu kako bi svi imali pristup ispravnim podacima. Čvorovi mreže su računala raspoređena diljem svijeta koji pokreću *blockchain* softver. Omogućuju sustavu da radi konstantno 24 sata dnevno, ubrzavaju i automatiziraju izvršavanje operacija. Distribuiranost čini sustav sigurnim zato što bi napadač morao istovremeno napadati veliki broj čvorova kako bi "zaobišao" konsenzus protokol.

2.3. Javna knjiga transakcija

Javna knjiga transakcija je sustav sličan bazi podataka. Bitna razlika je da je distribuiran. To daje sustav u kojemu povjerenje nije bitno jer "svi vide sve". Blokovi transakcija se zapisuju jedni iza drugih što čini lanac blokova. Blok koji je jednom "dobro" ulančan se ne može promijeniti ili obrisati iz sustava.

2.4. Transakcije

Transakcija je niz kriptografski potpisanih instrukcija. Klijent sastavlja transakciju te ju potpisuje sa svojim privatnim ključem. Mreži se objavljuje postojanje transakcije te mreža vraća identifikacijsku oznaku transakcije (engl. *txid*). Cilj transakcije je promijeniti stanje *blockchain* mreže što znači da ju rudari ili validatori moraju odobriti. Klijent zbog toga mora platiti pristojbu koju na kraju zarađuje rudar ili validator. Pristojba se računa po formuli 2.1.

$$pristojba = cijenaGoriva * količinaGoriva \tag{2.1}$$

Gorivo (engl. *gas*) prikazuje trošak provedbe transakcije. Što je transakcija kompleksnija to će gorivo biti veće. Cijena goriva se određuje ovisno o ponudi i potražnji za računalnom moći čvorova mreže. Pošiljatelj transakcije mora poslati dovoljno goriva kako bi se transakcija mogla do kraja izvršiti. Ako tijekom provedbe transakcije nestane

goriva, transakcija se poništava i *blockchain* se vraća u prethodno stanje. Klijentu se potrošeno gorivo ne može vratiti.

2.5. Novčanici

Novčanik *blockchain* mreže je digitalni novčanik u koji korisnici mogu spremati kriptovalute. Omogućuje upravljanje istima, stvaranjem i zapisivanjem transakcija te njihovim pretvaranjem u fiat valute. Novčanikom se upravlja pomoću dva ključa, jednim javnim i jednim privatnim ključem. Kako bi se odobrila neka radnja u novčaniku, radnja se mora potpisati. Potpis se stvara asimetričnom kriptografijom. Javni ključ je vidljiv svima, njega pošiljatelj koristi kao metu transakcije. Primatelj sa svojim tajnim privatnim ključem može potvrditi da je on vlasnik javnog ključa te da njemu poslani iznos sada pripada.

Novčanici se prema povezivosti dijele na tople (engl. *hot*) i hladne (engl. *cold*). Topli novčanici su povezani s internetom što ih čini rizičnijim, ali korisniku pristupačnijim odabirom. Koriste se za svakodnevne transakcije. Hladni novčanici ne zahtijevaju internet i uglavnom služe za pohranu veće količine kriptovaluta. Hladni novčanici se dijele na: fizičke i papirnate novčanike.
Fizički novčanik je uređaj koji izgleda kao USB memorija koji se može povezati na računalo ili mobitel. Transakcije se mogu potpisati bez pristupa internetu, no tek nakon povezivanja, transakcija će biti zapisana na *blockchainu*. Nisu lagani za korištenje, ali su jedna od najsigurnijih opcija za čuvanje i rad s kriptovalutama.
Papirnati novčanik je fizički papir/kartica na kojemu su privatni ključevi ispisani. Nemoguće ih je hakirati, no lako se može izgubiti ili nenamjerno oštetiti.
Topli novčanici se dijele na: desktop, web i mobilne novčanike.
Desktop novčanik je aplikacija koja se instalira na računalo. Koristi enkripciju za spremanje privatnih ključeva na računalo. Jednostavni su za korištenje pa je vjerojatno najbolji izbor za topli novčanik standardnom korisniku. Preporučuje se korištenje antivirusnih programa, sigurnosno kopiranje podataka i korištenje lozinke za pristup računu na računalu.
Novčanik na mobitelu je novčanik instaliran na mobitelima ili tabletima u obliku aplikacije. Veoma su popularni jer korisnici moraju imati samo internet i mobitel koji se većinu vremena nalazi uz njih. Slični su desktop novčanicima no manje sigurni zbog toga što su mobiteli izloženiji sigurnosnim napadima.

Web novčanik je novčanik na webu. Korisnik nema kontrolu nad privatnim ključevima te su skloni čestim napadima. Ovo je ujedno najjednostavniji i najnesigurniji novčanik. Preporuka ga je koristiti samo za sitne transakcije.

2.6. MetaMask

MetaMask je jedan od najpopularnijih web novčanika za spremanje kriptovaluta. Služi kao novčanik na Ethereum mreži. MetaMask dolazi kao proširenje web preglednika. Na njega je moguće spremati i slati Ether i sve ERC-20 žetone (engl. *token*).

Instaliranje je veoma jednostavno. Instalira se kao i svako drugo proširenje. Nakon završene instalacije klikom na oznaku proširenja se otvara prozor ne kojemu počinje proces kreiranja novčanika. Potrebno je unijeti lozinku, preporuča se korištenje malih i velikih slova, brojeva i znakova. Nakon prihvaćanja uvjeta korištenja MetaMask će prikazati frazu od 12 riječi koji će služiti kao sigurnosna kopija. Frazu je potrebno negdje zapisati i spremiti ju na sigurno mjesto. Na sljedećem zaslonu MetaMask će tražiti upis te iste fraze kako bi se uvjerio da je korisnik zapisao tu frazu. Fraza se koristi u slučaju oporavka novčanika nakon gubljenja pristupa računalu. Bitno je znati da svatko tko posjeduje frazu može pristupiti novčaniku i svim mogućnostima.

Pomoću MetaMaska, moguće je spojiti se na web decentralizirane aplikacije. Prilikom otvaranja web stranice pojaviti će se MetaMask prozor u kojemu će aplikacija tražiti dozvolu za povezivanje. Nakon povezivanja moguće je koristiti sve mogućnosti koje aplikacija nudi.

MetaMask je popularan i veoma jednostavan za instalaciju i korištenje. No, kao i svaki web novčanik, privatne ključeve pohranjuje u spremnik internet preglednika što predstavlja sigurnosni rizik kojeg treba biti svjestan.

2.7. Konsenzus protokoli

Konsenzus protokol je efikasan i pravedan mehanizam kojim se dolazi do dogovora između distribuiranih čvorova o legitimnosti transakcije u stvarnom vremenu. Mehanizam je otporan na greške pojedinačnih čvorova. Primjeri konsenzus protokola su dokaz o radu, dokaz o udjelu, dokaz o autoritetu (engl. *proof of state*) i dokaz o prostoru i vremenu (engl. *proof of spacetime*).

Dokaz o radu zahtjeva rješavanje kompleksnih matematičkih problema prije ulančavanja određenog bloka. Probleme rješavaju rudari, a proces se naziva rudarenje (engl. *mining*). Prije nego što se blok ubaci u lanac, rudari pokušavaju izračunati blok određene težine. Kako bi se *blockchain* zaštitio, nad zadnjim ubačenim blokom se radi hash, nastaje niz brojeva koji postaju zaglavlje sljedećeg bloka. U zaglavlju se nalazi nasumičan jedinstveni broj koji rudari žele izračunati, a zove se *nonce* (broj korišten samo jednom). Nonce i hash zadnjeg bloka se uparuju te se zatim provjerava da li se novi hash nalazi „ispod“ ciljne vrijednosti (engl. *target value*). Ciljna vrijednost je prag određen težinom rudarenja. Prvi rudar koji izračuna *nonce* može objaviti blok na *blockchain* te za taj rad dobiva naknadu. Glavni problemi dokaza o radu su velika količina energije koju troši, skalabilnost i slaba performansa ukoliko gledamo broj obrađenih transakcija po sekundi.

Dokaz o udjelu koristi uložene domaće (engl. *native*) žetone pojedinog *blockchaina* za provjeru transakcija i njihovo dodavanje na taj *blockchain*. Validator se određuje slučajnim odabirom, ali šanse se povećavaju ovisno o količini udjela i vremenu držanja udjela žetona pojedinog validatora. Ukoliko je validator izabran dobiva naknadu. Ovakav tip provjere transakcija troši manje računalnih resursa što je veliki problem dokaza o radu. Nisu potrebna moćna računala što dovodi do većeg broja validatora i bolje decentraliziranosti sustava. Postoji opasnost od napada “51%” gdje bi većinski vlasnik žetona pokušao postati centralna točka kontrole *blockchaina*. Time bi narušio i vrijednost svojih žetona što bi potencijalnog napadača trebalo odvratiti od napada.

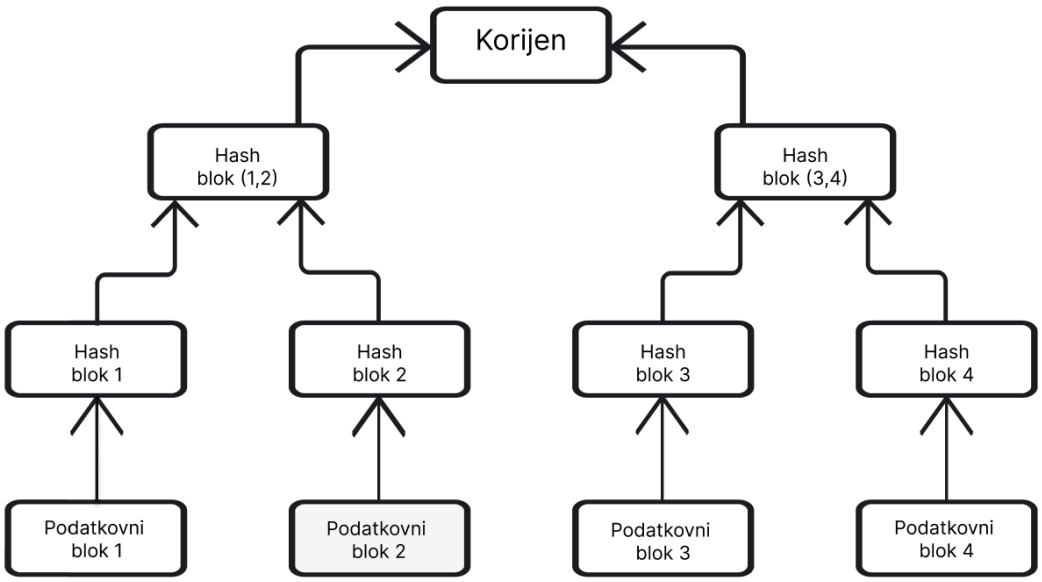
Dokaz o autoritetu koristi identitet odnosno reputaciju čvora koji želi napraviti promjenu na *blockchainu*. Za validaciju blokova nije potrebna računska moć ili novac. Sustav se pouzdaje u ograničen broj predefiniranih validatora blokova da će održavati sustav točnim. Validatori moraju biti pouzdani, što znači, postupak da se postane jedan je težak. Potencijalni validator mora biti spreman uložiti novac i dugoročno biti dostupan. Ograničen broj validatora povećava skalabilnost, no smanjuje decentraliziranost. Brži su izbor od dokaza o radu i dokaza o udjelu.

Dokaz o prostoru i vremenu se bazira na slobodnom prostoru koji rudar može pružiti za pohranu podataka i o duljini vremena koliko taj podatak pohranjuje. Ukoliko korisnik aplikacije zatraži određene podatke, aplikacija provjerava kod rudara da li i dalje pohranjuje te podatke. Ukoliko ih pohranjuje, korisnik dobiva svoje tražene podatke dok rudar dobiva naknadu. Kako bi postao rudar potreban je slobodan prostor za pohranu, brzina dohvata i interneta. Problem kod ovog protokola je taj što se ti podaci nalaze na

centraliziranim serverima. Ukoliko jedan rudar drži veliku količinu podataka, može manipulirati s njima i time ugroziti točnost sustava.

2.8. Merkleovo stablo

Merkleovo stablo je binarno stablo koje se koristi za sigurnu enkripciju podataka na *blockchainu*. Stablo se sastoji od hasheva raznih blokova podataka koji prikazuju sve transakcije u bloku. Grade se iz listova koji su zapravo blokovi podataka. Primjer Merkleovog stabla je prikazan na slici 2.2. Viša razina stabla su hashirani blokovi podataka. Nakon toga se uzimaju parovi blokova nad kojima je napravljen hash te se nad parovima tih blokova radi hash. Taj proces se obavlja sve dok se ne dođe do zadnjeg para koji se hashira u korijen Merkleovog stabla. Korijen Merkleovog stabla se sprema u zaglavlje bloka uz ostale atribute kao što su vrijeme, *nonce* i hash prošlog bloka.



Slika 2.2: Primjer Merkleovog stabla

3. ETHEREUM

Godine 2014. Vitalik Buterin objavljuje članak u kojem predstavlja Ethereum *blockchain* mrežu. Cilj Ethereuma je stvoriti tehnologiju za razvoj decentraliziranih aplikacija. Razvoj mora biti brz, aplikacije moraju biti sigurne i moraju moći međusobno komunicirati. Dizajniran je da bude skalabilan, siguran i decentraliziran. Trenutno je najpopularniji izbor za razvoj decentraliziranih aplikacija na *blockchainu*.

3.1. Ether

Ether pokreće Ethereum mrežu. Koristi se kao gorivo koje služi kao nagrada validatorima. Ether je zapravo ono što motivira validatore da se drže konsenzus protokola i da validiraju blokove. Svaki validator posjeduje neku količinu Ethera što je uvjet za decentraliziranost sustava.

Ether je također kriptovaluta kao Bitcoinom pa se s njim može trgovati. Ether je žeton koji se može podijeliti na manje jedinice. Najmanja je wei, a najveća Ether. Preračunavanje Ethera na manje jedinice je prikazano u tablici 3.1.

Tablica 3.1: Mjerne jedinice za iskazivanje Ethera

Naziv	wei	Vrijednost u Etheru
wei	1	10^-18
kwei	1,000	10^-15
mwei	1,000,000	10^-12
gwei	1,000,000,000	10^-9
microether	1,000,000,000,000	10^-6
miliether	1,000,000,000,000,000	10^-3
ether	1,000,000,000,000,000,000	1

3.2. Izvršavanje koda

Ethereum je izgrađen na temeljima Ethereum virtualnog stroja (engl. *Ethereum virtual machine*, EVM). EVM je lagana virtualna mašina koja omogućuje *runtime*

okruženje za izvršavanje koda. Pokreće se na svim čvorovima mreže. Ponaša se kao jedan procesor i izvršava bajt-kod (engl. *bytecode*) kao i običan procesor iako je distribuiran. Pošto je EVM virtualan, nema svoje resurse odnosno hardware već za to koristi čvorove. Kako bi to sve funkcioniralo postoji softver koji svi čvorovi pokreću. Taj softver zapravo implementira Ethereum protokol. U sebi sadrži EVM u kojemu se može izvršavati bajt-kod.

EVM je baziran na stog arhitekturi dubine 1024 sloja gdje je svaki sloj veličine 256 bitova. Okruženje u kojemu je svaki hardware i operacijski sustav kompatibilan. EVM stvara sloj apstrakcije između koda i stroja koja izvodi kod. On je zapravo samostalan *sandbox* koji nema pristup mreži, sustavu datoteka i ostalim procesima što stvara sigurniji sustav. EVM je također Turing potpun što znači da može izračunati (gotovo) svaki zadatak uz dovoljno resursa.

Solidity kod se ne može izvršiti od strane EVM u izvornom obliku. Kod se prvo mora pretvoriti u set instrukcija pod imenom opkod koji se dalje pretvara u bajt-kod, kod koji EVM može pročitati. Opkodovi su čitljivi ljudima, svaki opkod je predstavljen jednostavnom engleskom riječju koja ga opisuje. Jedan opkod se može zapisati u jedan bajt.

3.3. Lokacije pohrane podataka

U EVM-u postoji nekoliko lokacija pohrane podataka sa svojim karakteristikama i primjenama. Lokacije se mogu podijeliti na dvije vrste: nevolatilne (kod i spremnik) i volatilne (slog, argumenti i memorija). Volatilne i nevolatilne lokacije su prikazane na slici 3.1.

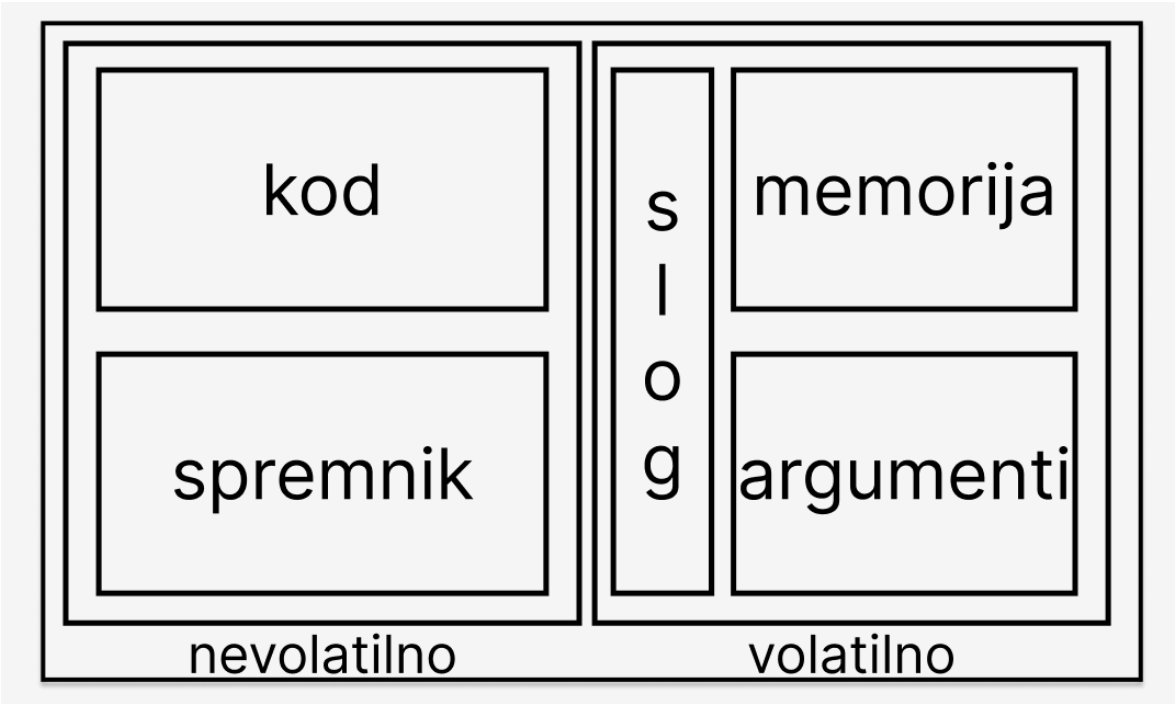
Kod (engl. *code*) se koristi za pohranu binarnog izvornog koda pametnog ugovora. Kod se sprema samo jednom i to tijekom objavljivanja pametnog ugovora.

Spremnik (engl. *storage*) se koristi za trajnu pohranu podataka. Podaci se mogu čitati, pisati i mijenjati. Pisanje u spremnik je puno skuplje od čitanja. Troškovi su veliki jer su to podaci koji se trajno pohranjuju na *blockchain* te su dostupni svima u mreži. U pozadini, spremnik je mapa s 2^256 polja veličine 32 bajta. Na spremnik se može gledati kao tvrdi disk.

Slog (engl. *stack*) se koristi za spremanje ulaznih i izlaznih podataka EVM instrukcija. Dubina sloga je 1024 polja veličine 32 bajta, no samo gornjih 16 je dostupno.

Argumenti (engl. *calldata*) se koristi za spremanje parametara funkcija. Podaci u ovoj lokaciji se mogu samo čitati.

Memorija (engl. *memory*) se koristi za pohranu podataka tijekom izvršavanja funkcija. Nakon izvršavanja funkcije podaci se gube. Memorija se može koristiti za pohranu podataka koji se ne moraju sačuvati. Na nju se može gledati kao RAM računala.



Slika 3.1: Volatilne i nevolatilne lokacije

3.4. Ethereum računi

Postoje dvije vrste Ethereum računa. Račun u vanjskom vlasništvu (engl. *Externally owned account*, EOA) i račun ugovora (engl. *Contract account*, CA).
EOA se sastoji od javnog ključa, privatnog ključ i jedinstvene adrese. Računom se upravlja pomoću privatnog ključa. Koristi se za držanje, slanje i primanje žetona. Može stvoriti interakciju s pametnim ugovorima, ali ne veže na sebe programski kod. Dva EOA međusobno mogu samo slati ili primati žetone.

Stvaranjem CA se mijenja mreža pa se zato plaća pristojba za razliku od stvaranja EOA. Uz sebe drži programski kod napisan u Solidity programskom jeziku. Nema ni javni ni privatni ključ, ali ima svoju jedinstvenu adresu. Zbog jedinstvene adrese drugi CA ili EOA mogu komunicirati s njim, pozivati funkcije koje ugovor sadrži te slati žetone. CA je

kontroliran pomoću programskog koda koji se automatski pokreće nakon ispunjavanja određenih uvjeta ugovora.

3.5. Decentralizirane aplikacije

Decentralizirane aplikacije (DAPPS) su aplikacije koje se izvršavaju na *blockchain* mreži. Aplikacija jednom objavljena na mreži je nepromjenjiva i njezin programski kod je javan. DAPPS imaju svoje prednosti i mane.

Programski kod, odnosno funkcionalnost aplikacije je javna. Prije korištenja, svatko može pogledati programski kod i uvjeriti se da program radi ono što su programeri obećali. Prednosti su te da svi mogu koristiti aplikaciju bez potrebe da korisnik objavljuje osobne informacije. Također, vlasnik pametnog ugovora ne može nikome zabraniti korištenje aplikacije što čini sustav bez cenzure.

DAPPS su novost u programskom svijetu pa se problemi i nedostaci otkrivaju kroz razvoj i korištenje aplikacija. Veliki problem su sigurnosne ranjivosti koje su do sada napadači iskoristili i ukrali više od 10 milijardi kuna.

3.6. Primjena decentraliziranih aplikacija

Primjena decentraliziranih aplikacija je velika, najpopularnije vrste su DeFi, aukcijske aplikacije, kockarske igre, grupno financiranje i DAO.

DeFi ili decentralizirane financije je naziv za sve financijske aplikacije na *blockchainu*. DeFi omogućava tržište bez centralnog autoriteta i tržište koje je stalno otvoreno za sve. Nastao je kao proizvod Bitcoina. Rješava dosta problema dosadašnjih tradicionalnih financijskih procesa. Prednosti DeFi-a nad tradicionalnim financijama su: korisnik drži svoj novac i ima punu kontrolu nad njim, tržište je stalno otvoreno i dostupno svima, transakcije su transparentne i obrađuju se puno brže zbog veće automatiziranosti.

Aukcijske aplikacije su još jedan odličan primjer rješavanja problema tradicionalnih stranica. Aukcije na *blockchainu* su u potpunosti transparentne što rješava problem ilegalnog podizanja cijene od strane prodavatelja. Sve aukcije su automatizirane, dostupne svima te se vlasništvo prebacuje puno brže nego u tradicionalnim aukcijama.

Za kockarske igre se je uvijek pričalo da su namještene od strane kockarnica. Uvođenjem *blockchaina* i pametnih ugovora dovodi se transparentnost. Korisnik može vidjeti kolike su mu šanse za dobitak bez potrebe za povjerenjem u ono što kockarnica predstavlja.

Grupno financiranje je odličan primjer automatiziranosti pametnih ugovora. Cilj grupnog financiranja je sakupiti određeni iznos novaca. Ukoliko se iznos sakupi, on se koristi za ono što je predviđeno pametnim ugovorom. U slučaju da se iznos ne sakupi, članovi grupe svoje novce dobivaju nazad.

Decentralizirana autonomna organizacija (DAO) je način upravljanja nad organizacijom koja nema centralni autoritet. Član organizacije je osoba koja posjeduje žetone. Žetoni se koriste za glasanje prilikom donošenja odluka vezanih za organizaciju. Svi glasovi se objavljuju na *blockchain* mreži pa su zato javno dostupni. Članovi mogu posjedovati različit broj žetona i time imati veću glasačku moć prilikom donošenja odluka. Sustav i s tom glasačkom razlikom ostaje pravedan, jer je u interesu svih da organizacija što bolje posluje.

3.7. Pametni ugovori

Pametni ugovori su temeljni blokovi Ethereum aplikacija. Pametni ugovor je set dogovorenih instrukcija koji se automatski izvršavaju nakon ispunjavanja uvjeta. Uvjeti su nepromjenjivi i javni što čini sustav pravednim i iskrenim. Ethereum daje mogućnost programerima da razvijaju aplikacije koje se drže svojih pravila i principa. Pametni ugovori se u Ethereumu pišu u programskom jeziku Solidity.

Pametni ugovor je Ethereum račun ugovor. Ima svoju javnu adresu i može slati, posjedovati i primati Ether.

3.8. Sigurnost pametnih ugovora

Pametni ugovori često rade sa osjetljivim podacima i novcem. Kao takvi su skloni čestim pokušajima napada. Ukoliko pametni ugovor ima propust, on će kad tad biti iskorišten. Dosadašnji napadi su služili kao motivacija za stvaranje dodatnih mehanizama i standarda kod razvoja pametnih ugovora. Jednom objavljen pametni ugovor se ne može promijeniti te je zato potrebna potpuna ispravnost i funkcionalnost. Trenutno postoje mnoge tvrtke koje se bave samo pregledom i testiranjem pametnih ugovora. Pametni

ugovor se nakon razvoja šalje trećoj stranki koja nakon pregleda vraća izvještaj. U izvještaj se unose pronađene greške i potencijalne ranjivosti. Nakon toga slijedi proces ispravljanja grešaka te potencijalno ponovno slanje pametnog ugovora na pregled. Korištenje trećih stranki služi kao dokaz klijentima da je pametni ugovor siguran jednom kada se sve pronađene greške isprave. U izvještaj se osim sigurnosnih ranjivosti često unose i savjeti za razvoj ugovora kako bi se pratili postojeći standardi.

Napad ponovnog upada (engl. *reentrancy attack*) je jedan od najgorih napada na pametni ugovor i sve njegove korisnike. Rizik se javlja kod podizanja žetona (engl. *withdraw*). Do ranjivosti dolazi ako pametni ugovor pošalje žetone prije nego što ažurira stanje na računu zlonamjernog ugovora. Napad se pokreće slanjem žetona na zlonamjerni pametni ugovor. Taj ugovor sadrži zamjensku funkciju koja ponovno traži podizanje žetona. Pošto stanje računa nije ažurirano, zlonamjerni ugovor može uzeti sve žetone koje pametni ugovor posjeduje. Ozbiljnost ove ranjivosti naglašava 60 milijuna dolara koji su ukradeni 2016. godine napadom ponovnog upada. Preporučuje se korištenje *Checks-Effects-Interaction* uzorka i stvaranje zaštitnog mehanizma od napada ponovnog upada. Zaštitni mehanizam se naziva *ReentrencyGuard*.

Pametni ugovori mogu primati i držati Ether. Ukoliko se prilikom razvoja zaboravi dodati opcija za podizanje Ethera, taj Ether može ostati zauvijek zaključan. Kako se to ne bi dogodilo potrebno je dodati funkciju za podizanje Ethera.

Ne korištenje uređivača funkcija i funkcija zahtjeva može dovesti do toga da osobe koje ne bi smjele imati pristup pozivaju funkcije. Pomoću uređivača funkcija i funkcija zahtjeva se također provjerava ispravnost poslanih podataka, bili oni namjerno poslani ili ne. Primjer je slučajno dodavanje minus znaka prije broja. Standard je i pisanje poruka greške kako bi se obavijestila osoba zašto je funkcija prekinula s izvođenjem.

Korištenje takozvane volatilne *pragma*-e može dovesti do neočekivanog izvođenja jednom objavljenog pametnog ugovora. Pragma bi trebala biti „zaključana“ od početka razvoja i testiranja do samog objavljivanja na mrežu. Korištenjem volatilnih ili različitih *pragma*-i prilikom testiranja i objavljivanja može dovesti do potencijalnih sigurnosnih rizika. Također se preporučuje korištenje stabilnih i novih verzija.

3.9. Testne mreže

Osim glavne *blockchain* mreže zvane *mainnet* postoje i testne mreže se koriste tijekom razvoja pametnih ugovora. Testne mreže se još nazivaju i *testnet*. Testne mreže su veoma slične glavnoj mreži. Najpopularnije su Ropsten, Kovan i Goerli. Na njih se spaja pomoću RPC-a (engl. *Remote Procedure Call*). U RPC se unosi link željene testne mreže. Mnogi novčanici kao što je MetaMask također nude opciju spajanja i na testne mreže. Moguće je stvoriti vlastiti lokalni *blockchain* i na njega se povezati. Ganache je primjer lokalnog *blockchaina* s grafičkim sučeljem. Moguć je uvid u sve adrese, transakcije i blokove na takvom *blockchainu*.

4. SOLIDITY

Solidity je objektno orijentirani programski jezik za razvoj pametnih ugovora. Solidity je inspiriran popularnim jezicima kao što su C++, Python i JavaScript. Podržava biblioteke, nasljeđivanje, događaje i ostale dodatne funkcionalnosti. Ekstenzija Solidity datoteke je.`sol`.

4.1. Definiranje Solidity programskog koda

Solidity prevoditelj (engl. *compiler*) potiče definiranje autorskih prava. Prevoditelj zapisuje licencu u metapodatke bajt koda. Pozicija licence u kodu nije bitna, ali je standard da se piše u prvoj liniji koda. Licenca se zapisuje u obliku komentara prikazanim u programskom kodu 4.1.

Programski kod 4.1: Definiranje autorskih prava programskog koda

```
// SPDX-License-Identifier: Unlicensed
```

Na sljedećoj liniji koda se definira *pragma* u kojoj se piše Solidity pametni ugovor (programski kod 4.2). Svakom novom *pragmom* se dodaju promijene koje utječu na pisanje koda. Ova linija ne mijenja prevoditelj koji se koristi već govori kojim prevoditeljem bi se kod trebao prevoditi. Ukoliko se verzije ne podudaraju, prevoditelj će vratiti grešku. Verzija bi trebala biti što novija, preporučuju se verzije nakon 0.8.0 i preporučuje se definiranje fiksne verzije.

Programski kod 4.2: Definiranje pragma-e

```
pragma solidity 0.8.4;
```

Moguće je odabrati ABI koder i dekoder. Postoje dvije verzije; verzija 1 i verzija 2. Verzija 1 je bila standardna verzija do Solidity 0.6.0 verzije. Nakon 0.6.0 Solidity-a moguće je bilo koristiti tada eksperimentalnu verziju 2. U članku [7] o promjenama s dolaskom nove verzijom je naznačeno da u 0.8.0 Solidity-u verzija 2 postaje standard. Sljedećim programskim kodom 4.3 se može promijeniti ili eksplicitno definirati verzija:

Programski kod 4.3: Definiranje ABI kodera i dekodera

```
pragma abicoder v2;
```

Solidity programske datoteke se mogu umetati jedna u drugu (programski kod 4.4). Time se stvara modularnost prilikom izrade pametnih ugovora. Nudi mogućnost pisanja funkcionalnosti na jednom mjestu i njezinog korištenja u drugim datoteka.

Programski kod 4.4: Primjer umetanja datoteke

```
import „./primjer.sol“;
```

Ugovor unutar Solidity datoteke se definira ključnom riječi contract i nazivom imena. Nakon toga se stavljaju vitičaste zagrade unutar kojih se razvija funkcionalnost ugovora kao što je prikazano u programskom kodu 4.5.

Programski kod 4.5: Primjer definiranja ugovora

```
contract VehicleInspection {
     /* funkcionalnost ugovora */
}
```

4.2. Vidljivost podataka i funkcija

Solidity definira 4 vrste vidljivosti podataka i funkcija, a to su: privatno (engl. *private*), unutarnje (engl. *internal*), vanjsko (engl. *external*) i javno (engl. *public*).
Privatni podaci i funkcije su vidljivi samo unutar ugovora u kojemu se nalaze. Privatna vidljivost je zadana vidljivost za varijable stanja. Unutarnji podaci i funkcije su dostupne unutar ugovora i ugovora koji su naslijedili ugovor u kojemu se nalaze podaci i funkcije. Vanjski podaci i funkcije su dostupni samo vanjskim ugovorima ili EOA računima preko transakcija. Ova vidljivost nije dostupna varijablama stanja. Javni podaci i funkcije se mogu pozivati iznutra i izvana. Varijablama definiranim kao javnima se automatski dodaje *getter*.

4.3. Vrste varijable

Solidity koristi statično pisanje varijabli. Svaka deklarirana varijabla mora imati svoj tip i vrijednost. Ukoliko se varijabla ne inicijalizira eksplicitno, ona i dalje ima svoju početnu vrijednost ovisno o tipu varijable. Solidity nudi 3 vrste varijabli: varijable stanja, lokalne varijable i globalne varijable.

Varijable stanja (engl. *state variables*) su varijable čije vrijednosti se trajno pohranjuju u spremnik. Deklariraju se u kodu pametnog ugovora, ali izvan funkcija. Memorija za varijable stanja se alocira prije objavljivanja pametnog ugovora te se ne može naknadno mijenjati. Korištenje varijabli stanja troši gorivo zbog korištenja spremnika pametnog ugovora koji se trajno pohranjuje na mreži *blockchaina*. Varijable stanja mogu imati javnu, unutarnju i privatnu vidljivost.

Lokalne varijable (engl. *local variable*) su varijable čije vrijednosti su dostupne samo tijekom izvršavanja funkcije. Nakon izvršavanja funkcije vrijednosti lokalnih varijabli se gube. Koriste se za držanje privremenih vrijednosti potrebnih za izvršavanje nekog zadatka. Pohranjuju se u memoriju pa ne troše gorivo jer ne mijenjaju stanje mreže.

Globalne varijable (engl. *global variables*) su posebne varijable koje se koriste za dohvaćanje podataka o transakciji i *blockchainu*. Globalne varijable se ne deklariraju već su pružene od strane Solidity-a. U tablici 4.1 se može vidjeti lista često korištenih globalnih varijabli dok se u programskom kodu 4.6 nalazi primjer definiranja varijabli.

Tablica 4.1: Tablica često korištenih globalnih varijabli

Naziv	Funkcionalnost
blockhash	Vraća hash bloka
block.timestamp	Vraća vrijeme trenutnog bloka
msg.data	Vraća potpuni calldata
msg.sender	Vraća pozivatelja funkcije
msg.value	Vraća količinu poslanog wei
tx.origin	Vraća pošiljatelja transakcije

Programski kod 4.6: Primjer definiranja varijabli

```
contract Ugovor {
      int varijablaStanja;

      function test() public pure {
            int lokalnaVarijabla = 10;
            uint trenutnoVrijemeBloka = block.timestamp;
      }
}
```

4.4. Tipovi podataka

Zbog Solidity-jevog statičnog pisanja varijabli, tip varijable se mora definirati tijekom deklaracije. Postoje tri podjele varijabli: vrijednosne varijable, referentne varijable i mapirane varijable.

Vrijednosne varijable se prenose pomoću vrijednosti (engl. *passed by value*), uvijek se kopiraju kada se koriste kao argumenti ili objekti. Primjer takvih tipova varijabli su: boolean, integer, adresa, ugovor, bajtovi i tako dalje.

Referentni tipovi se za razliku od vrijednosnih ne kopiraju već drže referencu (adresu) do objekta. Referentni tipovi Soliditya su strukture, polja i mape. Kako bi se moglo raditi s njima mora se definirati gdje je varijabla spremljena: *memory, storage* ili *calldata*.

Mapirane varijable rade na principu ključ – vrijednost. Ključ i vrijednost mogu biti bilo kojeg gore navedenog tipa. Mogu se jedino nalaziti na lokaciji storage i ne mogu biti korištene kao parametri ili tip podatka koji funkcija vraća.

4.5. Funkcije

Funkcija je blok koda koji se može reciklirati i ponovno koristiti u programu. Funkcija je temelj modularnosti razvoja programa. Smanjuju vrijeme razvoja i potrebne memorije za pohranu programa. Funkcijama se mogu proslijediti ulazni parametri te funkcija na kraju izvođenja može vratiti neku vrijednost. Ulaznim parametrima se također definira lokacija u memoriji ovisno o korištenom tipu parametra.

U Solidityu, funkcije se mogu podijeliti ovisno o tome na koji način koriste stanje ugovora. Mogu ga mijenjati, čitati ili ga uopće ne koristiti.

Pregledne (engl. *view*) funkcije su funkcije koje samo čitaju stanja na *blockchainu*. Obećavaju da neće ništa mijenjati. Definiraju se ključnom riječju *view* i definiraju koji tip podatka vraćaju ključnom riječju *returns* (programski kod 4.7).

Programski kod 4.7: Primjer pregledne funkcije

```
int varijablaStanja = 0;

function zbrojiSVarijablomStanja(int a) public view returns(int) {
     return a + varijablaStanja;
}
```

Čiste (engl. *pure*) funkcije obećavaju da neće ni čitati ni pisati u stanje. Definiraju se ključnom riječju *pure* i definiraju koji tip podatka vraćaju ključnom riječju *returns* (programski kod 4.8).

Programski kod 4.8: Primjer čiste funkcije

```
function zbroji(int a, int b) public pure returns(int) {
     return a + b;
}
```

Zamjenska (engl. *fallback*) funkcija je specijalna funkcija koja se izvršava ukoliko netko pokušava pristupiti funkciji koja ne postoji u ugovoru. Mora biti deklarirana kao external. Da bi mogla primati Ether mora se označiti kao payable. Primjer zamjenske funkcije je prikazan u programskom kodu 4.9.

Programski kod 4.9: Primjer zamjenske funkcije

```
fallback() external payable {}
```

Funkcija primanja (engl. *recieve*) je specijalna funkcija koja se izvršava kada se pozove ugovor bez podataka. Označuje se kao external i payable, koristi se za primanje Ethera. Funkcija primanja je prikazana u programskom kodu 4.10.

Programski kod 4.10: Primjer funkcije primanja

```
recieve() external payable {}
```

Funkcija zahtjeva (engl. *require*) je funkcija koja zahtjeva da su neki uvjeti zadovoljeni. Ukoliko uvjeti nisu zadovoljeni, izvršavanje funkcije se prekida i stanje *blockchaina* se ne mijenja. U parametre funkcije se stavljaju uvjet i poruka koja će se prikazati ukoliko uvjet nije zadovoljen. U programskom kodu 4.11 je prikazana funkcija zahtjeva koja provjerava da li je broj pozitivan.

Programski kod 4.11: Primjer funkcije zahtjeva

```
require(pozitivanBroj > 0, „Broj nije pozitivan!“);
```

Funkcije u Solidity-u mogu biti *overloadane*, ugovor može sadržavati više funkcija istog imena, ali s različitim parametrima.

4.6. Uređivači funkcija

Uređivači funkcija (engl. *modifier*) se koriste za postavljanje uvjeta prije njenog izvršavanja. Definiraju se ključnom riječju modifier i mogu primati ulazne parametre. U uređivaču se prvo definira uvjet pomoću funkcije zahtjeva ili uvjetnog bloka. Nakon uvjeta se dodaje specijalni simbol _ (programski kod 4.12).

Programski kod 4.12: Primjer uređivača funkcije

```
modifier samoPozitivanBroj(int pozitivanBroj) {
      require(pozitivanBroj > 0, „Broj nije pozitivan!“);
      _;
}
```

4.7. Događaji

Događaji su dio Ethereumovog zapisničkog (engl. *logging*) protokola. Emitiranjem događaja se spremaju podaci u zapisnik transakcija. Ovi zapisi se zapisuju na *blockchain*.

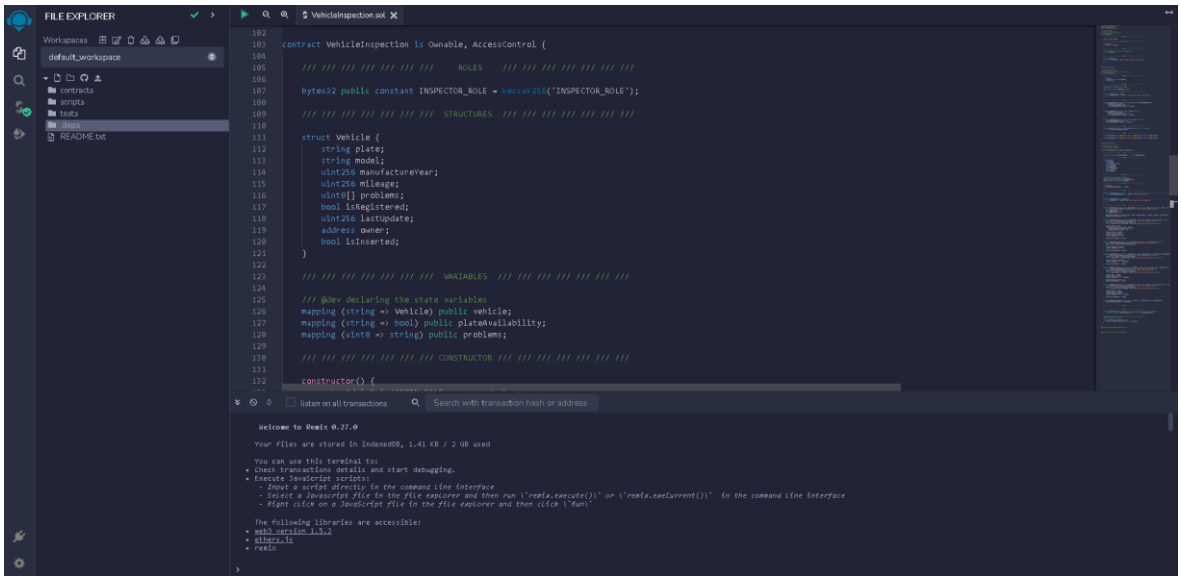
Događaji se koriste kako bi se klijentu ili web 3.0 aplikaciji javilo da je došlo do promjene na *blockchainu*. Primjer događaja koji emitira obavijest se nalazi u programskom kodu 4.13.

Programski kod 4.13: Primjer događaja i njegovog emitiranja

```
event Obavijesti(address pošiljatelj, string obacijest);

function objaviObavijest(string memory obavijest) public {
      /* ostala funkcionalnost */
      emit Obavijesti(msg.sender, obavijest);
}
```

4.8. Remix IDE

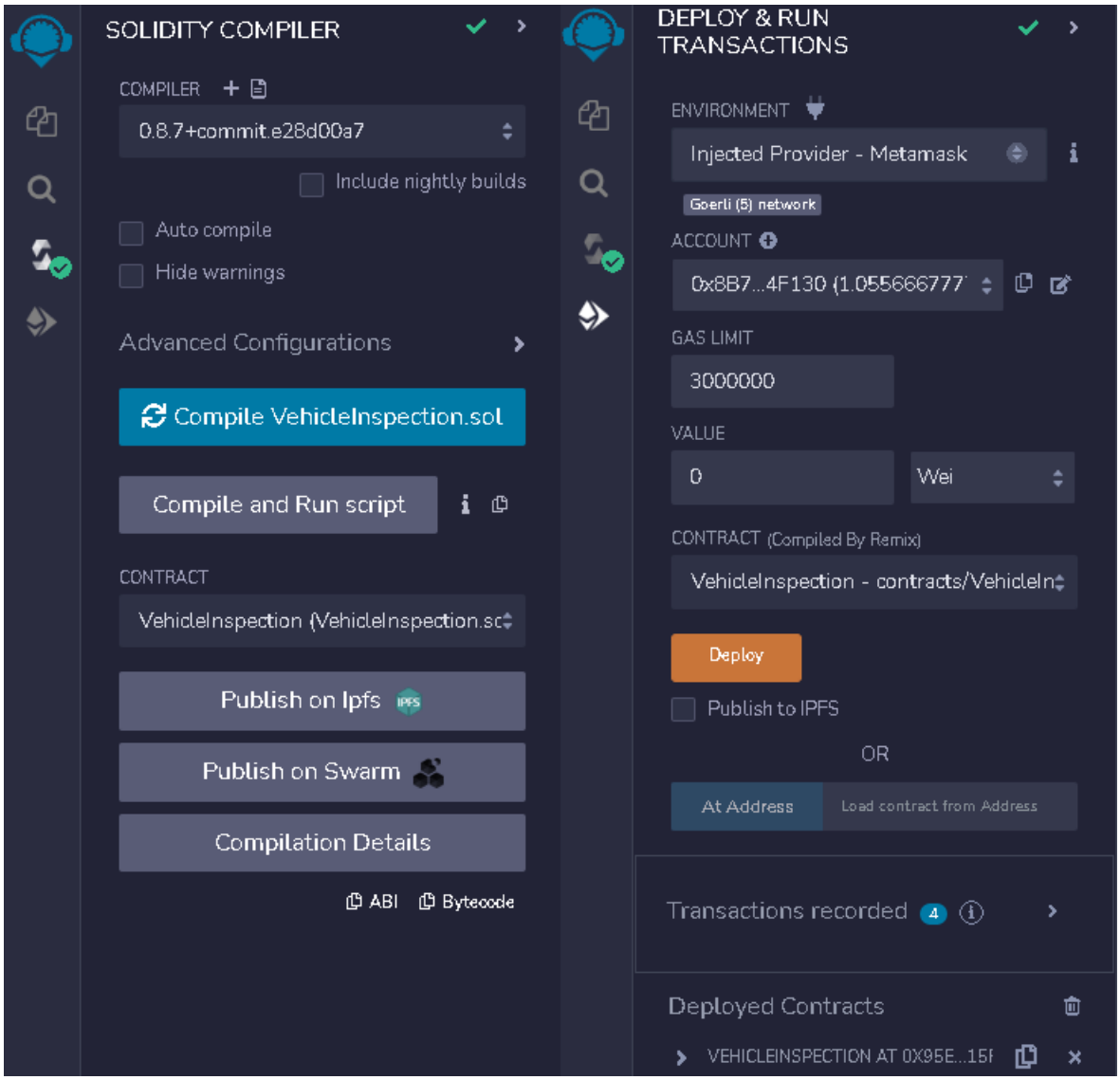
Remix IDE je *open source* aplikacija Remix Project platforme za razvoj pametnih ugovora. Koristi se kao razvojno okruženje, pokriva cijeli proces od pisanja do objavljivanja pametnog ugovora. Ima ugrađeni prevoditelj i debugger te se kod može objaviti na ugrađenu lokalnu mrežu. Ugrađena lokalna mreža ubrzava razvoj i testiranje. Nakon svake promjene nije potrebno ponovno objavljivati pametni ugovor na online *blockchain* mrežu i time trošiti žetone. Nakon završetka razvijanja pametnog ugovora, Remix IDE nudi i mogućnost povezivanja na online *blockchain* mreže. Jednom objavljeni pametni ugovor moguće je naknadno koristiti unutar Remix IDE okruženja. Na slici 4.1 prikazan je Solidity programski kod razvijen u Remix IDE okruženju.



Slika 4.1: RemixIDE

Remix IDE nudi opciju prevođenja Solidity programskog koda. Programer odabire prevoditelj koji želi koristiti u padajućem izborniku „Compiler“. U padajućem izborniku „Contract“ odabire koji pametni ugovor želi prevoditi. Pritiskom na plavi gumb Compile, pokreće se prevoditelj. Ukoliko je sve uredu pojavljuje se zelena kvačica. Na dnu se nalaze opcije za kopiranje Bytecode-a i ABI-a koji se koristi za daljnji razvoj web aplikacije. Na slici 4.2 s lijeve strane je prikazana kartica „Solidity compiler“.

Nakon uspješnog prevođenja pametni ugovor je moguće objaviti na kartici „Deploy & run transactions“. Moguće je odabrati okruženje na koje se želi objaviti ugovor. Remix IDE također nudi svoje okruženje na kojemu se brzo i jednostavno mogu testirati pametni ugovori. Zatim se odabire pametni ugovor koji se želi objaviti. Nakon odabira potrebnih opcija klikom na narančasti gumb „Deploy“ se objavljuje pametni ugovor. Na dnu kartice pod „Deployed Contracts“ pojavljuje se objavljeni pametni ugovor. Na slici 4.2 s desne strane je prikazana kartica „Deploy & run transactions“.



Slika 4.2: RemixIDE kartice

5. ANGULAR

Angular je open-source platforma za razvoj klijentskih aplikacija napisana u TypeScriptu. Ovakva platforma olakšava i ubrzava razvoj web aplikacija. Programeri mogu iskoristiti postojeće temelje i samo graditi na njih. Angular aplikacija se gradi od komponenti. Komponente se zatim spajaju u NgModule koji omogućuju gore spomenutu skalabilnost razvoja. Aplikacije se razviju u HTML, CSS i TypeScript programskim jezicima.

5.1. NgModuli

Angular aplikacija mora sadržavati minimalno jedan modul koji se još naziva korijenskim modulom. On se stvara prilikom kreiranja Angular projekta pod nazivom *AppModule*. *AppModule* se koristi za pokretanje aplikacije. U njega se mogu ubaciti drugi moduli. Postoji veliki broj razvijenih modula raznih funkcionalnosti. Jedan od takvih je Browser modul koji se koristi za pokretanje web preglednika i Angular aplikacije u njemu. On je automatski dodan u Angular projekt prilikom kreiranja. Kod deklaracije modula se koristi dekorator @NgModule. U programskom kodu 5.1 je prikazan početni izgled korijenskog modula.

Programski kod 5.1: Korijenski modul AppModule

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

5.2. Komponente

Komponente su gradivni elementi korisničkog sučelja Angular aplikacije. Prilikom kreiranja Angular projekta se stvara korijenska komponenta pod nazivom *AppComponent*. Svaka sljedeća komponenta se ubacuje u korijensku komponentu. Razvojem više komponenti se stvara struktura projekta koja podsjeća na stablo. Komponenta se gradi od TypeScript skripte koja sadrži funkcionalnost komponente i HTML i CSS datoteka koje služe za razvoj izgleda komponente. Komponenta se deklarira deklaratorom @Component. U programskom kodu 5.2 je prikazan početni izgled korijenske komponente.

Programski kod 5.2: Korijenska komponenta AppComponent.

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'vehicle-inspection;
}
```

5.3. Angular CLI

Angular CLI je naredbeni redak koji služi kao pomoć prilikom kreiranja i razvoja Angular projekata. Dolazi kao dio @angular/cli paketa. Pomoću Angular CLI moguće je kreirati i pokrenuti Angular projekt naredbama iz programskog koda 5.3.

Programski kod 5.3: Kreiranje i pokretanje Angular projekta

```
ng new vehicle-inspection
cd vehicle-inspection
ng serve
```

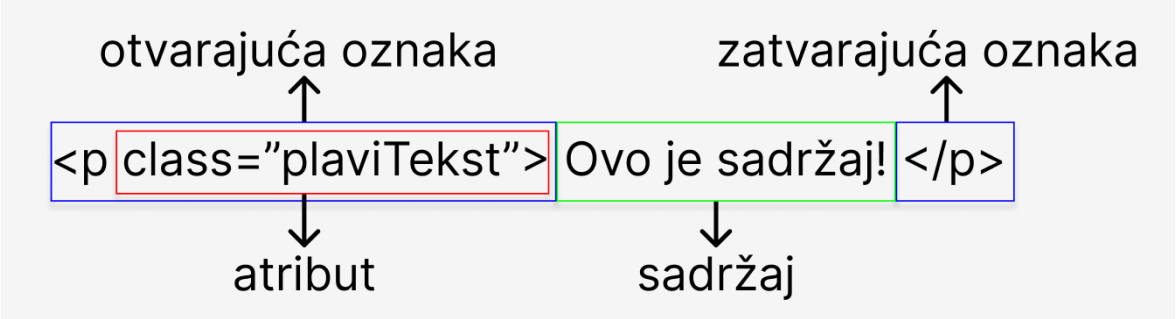
Kod razvoja Angular aplikacija programer će stvarati nove module i komponente. Programer mora sam stvoriti HTML, CSS i TypeScript datoteke. U njih ubacuje već standardan programski kod. Novokreirane dijelove je potrebno točno referencirati u

korijenskom modulu. Taj cijeli proces Angular CLI olakšava korištenjem *generate* naredbe.

5.4. HTML

HTML (engl. *Hyper Text Markup Language*) je uređivački jezik koji predstavlja strukturu web stranice. Sastoji se od niza elemenata i teksta. Svaki element ima svoje karakteristike i primjenu. U elemente se može unijeti sadržaj koji će se prikazati u web pregledniku. Izgled sadržaja se mijenja ovisno o elementu u kojemu se nalazi te stilovima primijenjenim na njega.

Elementi imaju svoju otvarajuću i zatvarajuću oznaku (engl. *opening and closing tag*). Otvarajuća oznaka se označava nazivom elementa unutar špicastih zagrada, a zatvarajuća sa prefiksom „/“ i nazivom unutar špicastih zagrada. U otvarajuću oznaku se mogu dodati atributi. HTML element ne mora sadržavati sadržaj. Na slici 5.1 je to prikazano grafički.



Slika 5.1: HTML element

U programskom kodu 5.4 je prikazan primjer elementa s atributom i sadržajem i primjer element bez sadržaja.

Programski kod 5.4: Primjer HTML elementa

```
<body>
      <p class=“plaviTekst“>Ovo je sadržaj!</p>
      <img src=“slika.jpg“ alt=“Slika“>
</body>
```

5.5. CSS

CSS (engl. *cascading style sheets*) je stilski jezik za uređivanje HTML elemenata. Dodatno opisuje kako će se elementi prikazati na web stranici. CSS se može direktno pisati u HTML element unutar otvarajuće oznake. No, standard je korištenje vanjskih CSS datoteka i njihovo umetanje u HTML stranice. Time se stvara čišći i pregledniji HTML kod. Vanjske CSS datoteke se također mogu iskoristiti unutar više HTML stranica čime se stvara standard koda. Izgled elemenata unutar CSS datoteka se može definirati na dva načina. Definiranjem izgleda pojedinih HTML elemenata ili korištenjem klasa koje se zatim mogu umetnuti u bilo koji HTML element. U programskom kodu 5.5 su prikazana oba primjera.

Programski kod 5.5: Primjer CSS datoteke

```
p {
      color: blue;
}
.plaviTekst {
      color: blue;
}
```

U prvom primjeru je definiran izgled teksta u svakom HTML elementu p (paragraf). To znači da će svaki paragraf u cijeloj HTML stranici imati plavi tekst. Ukoliko to nije željeni rezultat, mogu se koristiti klase kao u drugome primjeru. Klasa se dodaje samo u one paragrafe u kojima je potreban tekst plave boje. Također ta klasa se može dodati i na druge elemente čime se stvara fleksibilnost prilikom razvoja.

5.6. TypeScript

TypeScript je programski jezik izgrađen na temeljima JavaScripta. TypeScript nudi jednostavnost i snagu JavaScripta s dodatnim kontrolama što ga čini stabilnijim. Ono što ga čini drugačijim su tipovi podataka, sučelja i klase. Prevodi se u standardni JavaScript.

JavaScript koristi dinamičnu dodjelu tipova podataka. Tip podatka varijable se tek dodaje kada je program pokrenut što može dovesti do neočekivanih bugova u programu. Velika prednost TypeScripta je to što se varijablama mogu dodati tipovi podataka. Za

razliku od JavaScripta, TypeScript koristi statičnu provjeru tipa podataka. Prije samog prevođenja TypeScript upozorava na grešku u napisanom kodu. No, kako se TypeScript prevodi u JavaScript, kod će svejedno biti pokrenut. TypeScript služi kao dodatna sigurnost u razvoju aplikacija, dok će napisan kod biti jednako brzo izvršen kao da je napisan u JavaScriptu.

Programski kod 5.6: Primjer koda

```
let var = „Hello world!“;
var = 123;
console.log(var);
```

Programski kod 5.6 prikazuje primjer koda kojim će se vidjeti razlika između JavaScripta i TypeScripta. Prvo je izvršen kao JavaScript kod. Definira se u varijablu var kojoj se zadaje tip string. U sljedećoj liniji varijabli var se pridružuje vrijednost broja. Ovo je dopušteno jer je JavaScript dosta popustljiv oko redefiniranja varijabli. Na kraju izvršavanja koda ispisati će se broj 123 u konzoli. U drugom slučaju je kod izvršen kao TypeScript kod. Prije samog izvršavanja se prikazuje „*Type 'number' is not assignable to type 'string'*.“. To govori programeru da bi ovakav kod mogao dovesti do neželjenog ponašanja programa kod izvršavanja. Ukoliko se to zanemari kod će se izvršiti i ispisati u konzolu broj 123. U ovom specifičnom jednostavnom primjeru je dobiven željeni rezultat. U kompleksnijim programima ovo može dovesti do neželjenog ponašanja i bugova.

6. PROJEKT

Izrađena je aplikacija koja se sastoji od pametnog ugovora i frontenda razvijenog u Angularu. Aplikacija omogućava korisnicima pristup funkcijama pametnog ugovora preko korisničkog sučelja prikazanog na slici 6.1.

Slika 6.1: Početna stranica aplikacije

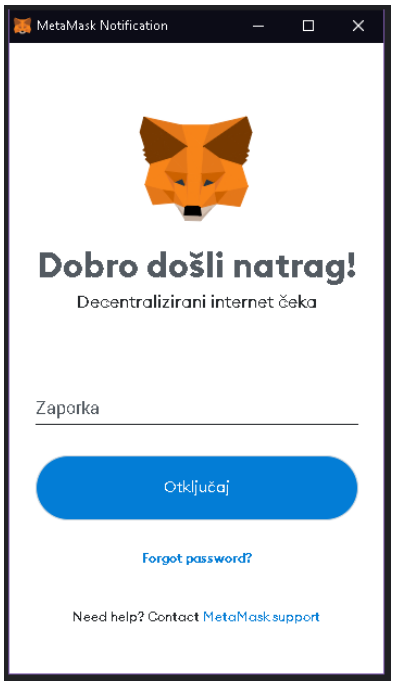
6.1. Ideja

Za posjedovanje vozila je potrebna izdašna količina novca. Od kupovine do održavanja. Vrijednost vozila služi kao motivacija kriminalcima za razne prijevare. Kriminalci su vraćali kilometražu, lažno se predstavljali kao vlasnici vozila i lagali o stanju vozila. Sustav izrađen na *blockchain* tehnologiji bi trebao umanjiti mogućnosti kriminalaca zbog osobina *blockchaina* i pametnih ugovora ranije navedenih kroz rad. Sustav za pohranu podataka s tehničkog pregleda vozila mogu koristiti centri za tehnički pregled vozila i obični ljudi za provjeru informacija o vozilu.

6.2. Prijava u aplikaciju

Korisnik se prijavljuje u aplikaciju pomoću svog računa u MetaMasku. Prilikom pokretanja web aplikacije u gornjem desnom kutu će iskočiti MetaMask prozor sa slike 6.2

u koji korisnik unosi svoje podatke. Nakon uspješne prijave odabire adresu s koje želi komunicirati sa pametnim ugovorom.



Slika 6.2: MetaMask prozor za prijavu u aplikaciju

U korijenskoj komponenti Angular aplikacije se pozivaju metode prikazane u programskom kodu 6.1 koje otvaraju prozor i spremaju adrese korisnika. Za spajanje na *blockchain* se koristi web3.js. Web3.js je skup biblioteka koji se koristi za komuniciranje s Ethereum *blockchainom*. Biblioteke omogućavaju dohvaćanje korisnikovih novčanika i adresa, slanje transakcija i pozivanje funkcija pametnih ugovora. Prilikom pokretanja aplikacije se instancira web3 objekt s parametrom koji predstavlja poslužitelja. U aplikaciji je korišten *ethereum* poslužitelj. Nakon instanciranja, pokušava se dohvatiti adresa novčanika. Ukoliko korisnik prvi put koristi aplikaciju, od njega se traži pristup pomoću ugrađene funkcije *ethereum.enable()*. U programskom kodu 6.1 se može vidjeti blok programskog koda zadužen za ranije opisani proces prijave. Pomoću metoda *getWallet* i *getAccounts* se dohvaćaju adrese računa dok se metoda *setWallet* poziva u korijenskoj komponenti Angulara.

Programski kod 6.1: Blok programskog koda za prijavu korisnika

```
public setWallet() {
    this.getWallet().then((address) => {
      this.wallet = address;
    });
```

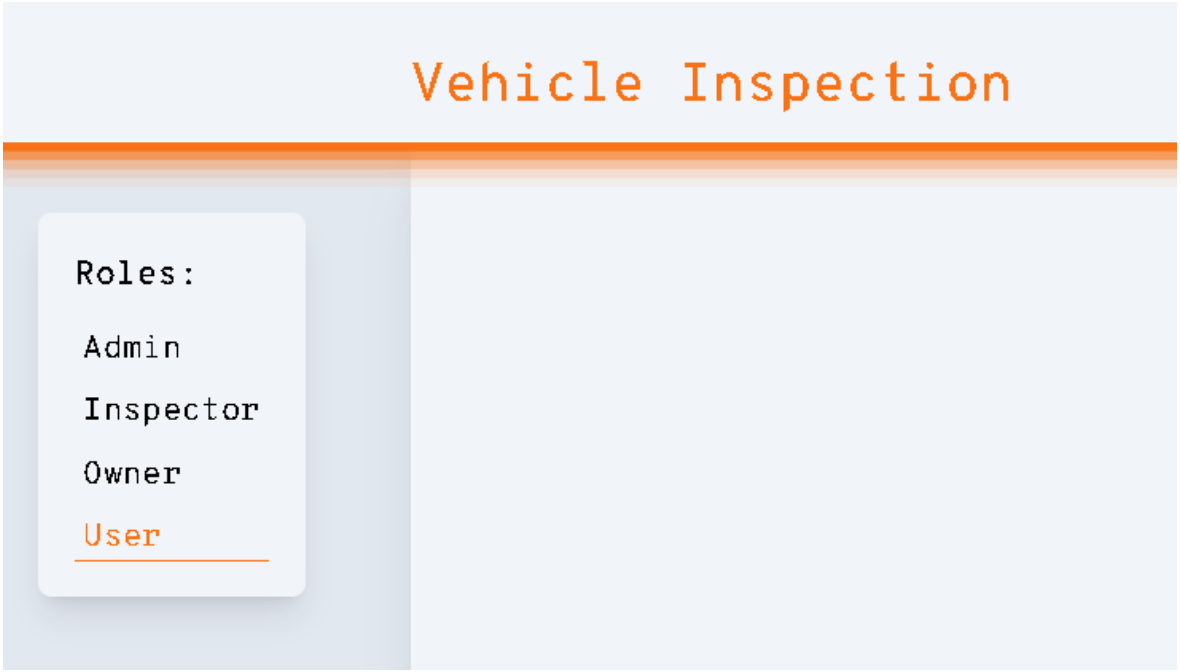
```
  }

private async getAccounts() {
     try {
          return await window.ethereum.request({
               method: 'eth_accounts'
          });
     } catch (e) { return []; }
}

private async getWallet() {
    window.web3 = new Web3(window.ethereum);
    let addresses = await this.getAccounts();

    if (!addresses.length) {
      try {
        addresses = await window.ethereum.enable();
      } catch (e) {
        return false;
      }
    }

    return addresses.length ? addresses[0] : null;
}
```

6.3. Korisnici

Aplikacija je podijeljena tako da ju mogu koristiti četiri vrste korisnika. Pošto su pametni ugovori javni, korisnik sam mora voditi računa koju ulogu posjeduje. Pametni ugovor sadrži informaciju o ulogama korisnika koji pozivaju funkcije. Poziv će biti poništen ako adresa korisnika nema potrebnu ulogu. Ukoliko svejedno pokuša pozvati metodu izgubiti će novac koji će biti potrošen na gorivo.

Četiri vrste korisnika su: vlasnik pametnog ugovora, administrator, inspektor i običan korisnik. Vlasnik pametnog ugovora kreira pametni ugovor i može prebaciti vlasništvo na drugu adresu. Administrator ima mogućnost dodjeljivanja i oduzimanja uloge inspektora. Inspektor unosi podatke o vozilu. Može unijeti novo vozilo u sustav ili ažurirati postojeće. Također može prebaciti vlasništvo vozila s jedne adrese na drugu. Običan korisnik ima mogućnost pregleda informacija o vozilima. Običan korisnik je također početna uloga svakoga tko želi komunicirati s aplikacijom.

Na slici 6.3 je prikazan dio korisničkog sučelja u gornjem lijevom kutu gdje korisnik odabire koju ulogu posjeduje.



Slika 6.3: Izbornik uloga

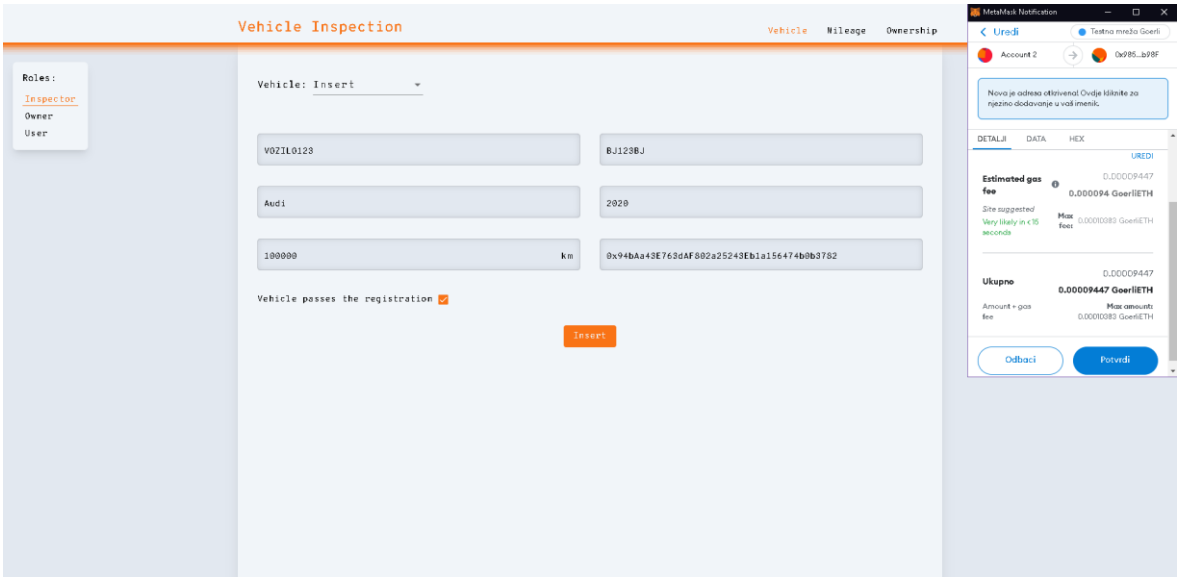
Uloge unutar pametnog ugovora su riješene pomoću dva pomoćna pametna ugovora Ownable i AccessControl. Oba ugovora su standard korištenja kod razvoja pametnog ugovora. Ownable ugovor postaje neizostavni dio prilikom pisanja pametnih ugovora dok se AccessControl koristi ovisno o potrebi za dodatnim ulogama. Ownable ugovor sadrži sve funkcionalnosti vezane za vlasnika pametnog ugovora. Konstruktor glavnog ugovora se inicijalizira tako da se postavlja vlasnik. Vlasnik je osoba koja objavljuje pametne ugovore. Korištenjem funkcija zahtjeva i uređivača funkcija se kontrolira tko smije pozvati zaštićene funkcije. U ovome projektu su to: prebacivanje vlasništva pametnog ugovora te davanje i oduzimanje uloge administratora. Same funkcije za davanje i oduzimanje uloge se nalaze u AccessControl ugovoru. Vlasnik automatski postaje i administrator kako bi mogao drugima dodijeliti istu ulogu. Administrator dodjeljuje i oduzima ulogu inspektora. U programskom kodu 6.2 su prikazane funkcije za dodjeljivanje i oduzimanje uloga. Funkcije su zaštićene uređivačem funkcije koja dozvoljava samo administratoru da ih izvršava.

Programski kod 6.2: Dio programskog koda AccessControl pametnog ugovora

```
struct Role {
      mapping(address => bool) members;
      bytes32 role;
}
mapping(bytes32 => Role) private roles;

function grantRole(bytes32 role, address account) public
onlyRole(ADMIN_ROLE) {
      if (!hasRole(role, account)) {
            roles[role].members[account] = true;
            emit RoleGranted(role, account, msg.sender);
            }
      }

function revokeRole(bytes32 role, address account) public
onlyRole(ADMIN_ROLE) {
      if (hasRole(role, account)) {
            roles[role].members[account] = false;
            emit RoleRevoked(role, account, msg.sender);
      }
}
```

6.4. Korištenje aplikacije

Korisnik nakon prijave u izborniku u gornjem desnom kutu može izabrati koju metodu pametnog ugovora želi pozvati. Ispred njega se prikazuje forma u koju unosi podatke. Nakon upisa podataka i provjere može kliknuti gumb za podnošenje forme. Otvara se MetaMask koji traži potvrdu za slanje transakcije. Nakon potvrde korisnik čeka neko vrijeme da se transakcija zapiše u *blockchain*. Kada korisnik u MetaMasku vidi potvrdu da je transakcija prošla može nastaviti s daljnjim radom. Na slici 6.4 vidljiv je primjer ispunjavanja forme. U ovome primjeru inspektor unosi novo vozilo u sustav sa svim potrebnim podacima.



Slika 6.4: Ispunjavanje forme

U programskom kodu 6.3 je prikazana funkcija pametnog ugovora za unos podataka vozila. Funkcija koristi uređivač funkcija koji dozvoljava samo inspektoru izvršavanje funkcije. Pomoću funkcije zahtjeva se provjera da li vozilo sa serijskim brojem već postoji u mapu *vehicles*. Kako se ne bi morao slati dodatan parametar koji predstavlja da li vozilo prolazi registraciju, provjerava se duljina polja parametra *_problems*. Problemi se zapisuju kao kodovi u obliku brojeva jer zauzimaju manje memorije od stringova. U mapi *problemCodes* su definirana značenja svakog koda. Veliki broj parametara može predstavljati problem prilikom prevođenja i izvršavanja jednom objavljenog pametnog ugovora. Zatim se unosi vozilo u *vehicles* sa serijskim brojem kao ključem, a ostalim podacima kao vrijednostima te mape.

Programski kod 6.3: Dio programskog koda za unos vozila

```
function insertVehicle(
      string calldata _serialNumber, string calldata _plate,
      string calldata _model, uint256 _manufactureYear,
      uint256 _mileage, uint8[] calldata _problems, address _owner)
      public onlyInspector {

            require(vehicles[_serialNumber].isInserted == false,
                  "Data: Vehicle already inserted.");
            require(plateAvailability[_plate] = true,
                  „Data: Plate is not available.“);
            bool _isRegistered;
            uint256 _registrationDate;
```

```
            if(_problems.length == 0) {
                  _isRegistered = true;
                  _registrationDate = block.timestamp;
            }
            vehicles[_serialNumber] = Vehicle(
                  _plate, _model, _manufactureYear, _mileage,
                  _problems, _isRegistered, _registrationDate,
                  _owner, true);
            plateAvailability[_plate] = false;

            emit VehicleRegistration(_serialNumber, _isRegistered)
}
```

Prilikom registracije vozila se unosi registracijska oznaka. U funkciji je potrebno provjeriti da li je ta registracijska oznaka dostupna. Solidity ne nudi svoje rješenje za uspoređivanje *stringova*. Potrebno je napisati vlastitu funkciju (programski kod 6.4). U Solidity-u se može napraviti hash *stringa* korištenjem keccak256 algoritma. Keccak256 je algoritam SHA-3 obitelji koji pravi hash ulaznog parametra u izlaz fiksne duljine.

Programski kod 6.4: Funkcija za usporedbu stringova

```
function compareStrings(string memory a, string memory b)
      internal pure returns(bool) {
            return (keccak256(abi.encodePacked(a)) ==
            keccak256(abi.encodePacked(b)));
}
```

Glavni pametni ugovor zapravo ne sadrži raspisane funkcije za dohvaćanje podataka. Sve varijable stanja koje spremaju podatke imaju javnu vidljivost te Solidity samostalno stvara *getter* funkcije.

Plaćanje tehničkog pregleda vozila bi se moglo obaviti i direktnim slanjem Ethera na adresu pametnog ugovora. Pametni ugovor sadrži funkcije za primanje i podizanje Ethera prikazane u programskom kodu 6.5 Funkciju podizanja Ethera može pozvati samo vlasnik ugovora.

Programski kod 6.5: Funkcije za primanje i podizanje Ethera

```
recieve() payable external { }
```

```
function withdraw() onlyOwner external {
      payable(msg.sender).transfer(address(this).balance);
}
```

Angular aplikacija je minimalističkog izgleda, navigiranje i unos podataka su jednostavni. Time se ostvaruje preglednost kod unosa osjetljivih podataka, svako polje forme sadrži validatore koji daju korisniku dodatnu informaciju da li su podaci ispravnog formata. Moguće greške su krivi unos godine gdje validator provjerava da godina proizvodnje nije veća od trenutna godine. Moguć je slučajni unos slova na mjesto gdje se upisuje broj ili slučajno upisivanje minusa ispred kilometraže. Sve te poruke će biti ispisane prije nego što korisnik pokuša pozvati pametni ugovor čime se štedi vrijeme i resursi. Primjer takve forme se nalazi u programskom kodu 6.6.

Programski kod 6.6: Angular forma za unos vozila

```
registerForm = this.fb.group({
      serialNumber: ['', Validators.required],
      plate: ['', Validators.required],
      model: ['', Validators.required],
      manufactureYear: ['', [
            Validators.required, Validators.max(this.maxYear),
            Validators.min(1900)]
      ],
      mileage: ['', [Validators.required, Validators.min(1)]],
      problems: ['', [
            Validators.required,
            Validators.maxLength(256)]
      ],
      isRegistered: [false],
      owner: ['', Validators.required],
});
```

7. ZAKLJUČAK

Blockchain postaje sve popularniji izbor kao rješenje za pohranu podataka iliti transakcija. Osobine kao što su javnost, distribuiranost i decentraliziranost čine sustav u koji se može vjerovati. Sustav radi neprekidno bez centralne kontrole te sve što se događa na mreži je vidljivo svima. Transakcije se zapisuju u javnu knjigu transakcija, a koja transakcija će se zapisati, odlučuje se jednim od mnogih konsenzus protokola. *Blockchain* mreže se brzo razvijaju te je njihova primjena sve veća. Organizacije sve više shvaćaju važnost korištenja ovakvog sustava. Može se pretpostaviti da će *blockchain* postati obavezna tehnologija u razvoju aplikacija koje koriste osjetljive podatke.

Vremenom su se razvijale razne *blockchain* mreže sa svojim primjenama i razlikama. 2014. godine nastaje Ethereum koji omogućava razvoj decentraliziranih aplikacija u programskom jeziku Solidity. Te aplikacije se popularnije nazivaju pametnim ugovorima. Kao i sve na *blockchainu*, programski kod pametnih ugovora je javan. Svi koji žele komunicirati s pametnim ugovorom su u mogućnosti vidjeti programski kod. Solidity je objektno orijentirani programski jezik sa svojim posebnim mogućnostima kako bi se prilagodio izvršavanju na *blockchainu*. Solidity je konstantno u razvoju te svakim ažuriranjem stvara jednostavnije i brže okruženje za razvoj.

U radu je izrađen sustav za pohranu podataka s tehničkog pregleda vozila. Korištena je Ethereum *blockchain* mreža i Solidity programski jezik za pisanje pametnog ugovora. Pametni ugovori su standard razvoja decentraliziranih aplikacija. Uz pametni ugovor je napravljena web aplikacija u Angularu. Web aplikacija služi kao primjer kako bi se funkcije pametnog ugovora mogle ukomponirati u jednostavno korisničko sučelje. Aplikacija se povezuje s pametnim ugovor i šalje podatke koji se ispravno pohranjuju u *blockchain*.

Svaki sustav teži dodatnom razvoju i poboljšanju. Pohrana velikog broja podataka može biti preskupa potencijalnim klijentima te bi dodatna optimizacija pametnog ugovora bila najbitniji korak prema poboljšanju pametnog ugovora. Većina sadašnjih aplikacija dolazi u nekoliko različitih inačica. Običnim korisnicima bi mobilna inačica ovog sustava vjerojatno bolje odgovarala nego web aplikacija jer im je bitan samo pregled podataka. Sustav u ovoj verziji ispunjava uvjete za korištenje u stvarnom svijetu, no ranije spomenuta poboljšanja bi imala veliki pozitivan utjecaj na potencijalne klijente.

8. LITERATURA

[1] Satoshi Nakamoto. Bitcoin: A Peer-toPeer Electronic Cash System. 2008. Dostupno na: https://bitcoin.org/bitcoin.pdf. (17.10.2022.)

[2] Vitalik Buterin. Ethereum Whitepaper. 2014. Dostupno na: https://ethereum.org/en/whitepaper. (17.10.2022.)

[3] Ethereum Virtual Machine. Dostupno na: https://ethereum.org/en/developers/docs/evm. (17.10.2022.)

[4] Gas and fees. Dostupno na: https://ethereum.org/en/developers/docs/gas. (17.10.2022.)

[5] Stefan Dziembowski, Sebastian Fraust, Vladimir Kolmogorov, Krzystof Pietrzak: Proofs of Space, 2013

[6] Solidity – Solidity 0.8.4 documentation. 2021. Dostupno na: https://docs.soliditylang.org/en/v0.8.4. (17.10.2022.)

[7] Solidity v0.8.0 Breaking Changes. 2021. Dostupno na: https://docs.soliditylang.org/en/v0.8.15/080-breaking-changes.html. (17.10.2022.)

[8] Angular – Introduction to the Angular Docs. Dostupno na: https://angular.io/docs. (17.10.2022.)

9. OZNAKE I KRATICE

CA – Contract account (račun ugovora)

DAO - Decentralizirana autonomna organizacija

DAPPS – Decentralized application (decentralizirana aplikacija)

DeFi – decentralizirane financije

EVM – Ethereum virtual machine (Ethereum virtualna mašina)

10. SAŽETAK

Naslov: Sustav za pohranu podataka s tehničkog pregleda vozila korištenjem pametnih ugovora na *blockchainu*

Blockchain je javna knjiga transakcija. *Blockchain* je javan, distribuiran i decentraliziran sustav za pohranu transakcija. Konsenzus protokol nudi pravedan način odlučivanja koja transakcija će biti zapisana kao sljedeća u lanac blokova. Osim pohrane čistih podataka, određene *blockchain* mreže kao što je Ethereum nude pohranu i izvršavanje programskog koda. Program na Ethereumu se naziva pametnim ugovorom. Pametni ugovori se pišu u objektno orijentiranom programskom jeziku Solidity. Ethereum virtualna mašina daje *runtime* okuženje za izvršavanje tog istog programskog koda. Pametni ugovori su vidljivi svim članovima mreže. Time se omogućava razvoj aplikacija u kojima povjerenje prema vlasniku aplikacije nije potrebno. To povjerenje zamjenjuje javnost programskog koda. Aplikacija kojoj takav sustav odgovara je aplikacija za pohranu podataka s tehničkog pregleda vozila. Kroz rad je opisana aplikacija od ideje, razvoja pa sve do konačnog proizvoda.

Ključne riječi: *blockchain*, Ethereum, pametni ugovor, Solidity.

11. ABSTRACT

Title: A system for storing data from the technical inspection of vehicles using smart contracts on the blockchain

Blockchain is a public book of transactions. Blockchain is a public, distributed and decentralized system for storing transactions. The consensus protocol offers a fair way of deciding which transaction will be stored next on the blockchain. In addition to storing data, certain blockchains such as Ethereum offer the storage and execution of code. The program run on Ethereum is called a smart contract. Smart contracts are written in the object-oriented programming language Solidity. The Ethereum virtual machine provides a runtime environment to execute code. Smart contracts are visible to all network members. This enables the development of applications in which trust towards the application owner is not necessary. That trust is replaced by the publicity of the program code. The application to which such a system corresponds is an application for data storage with a technical inspection of the vehicle. The paper describes the application from the idea, development to the final product.

Keywords: blockchain, Ethereum, smart contract, Solidity.

IZJAVA O AUTORSTVU ZAVRŠNOG RADA

Pod punom odgovornošću izjavljujem da sam ovaj rad izradio/la samostalno, poštujući načela akademske čestitosti, pravila struke te pravila i norme standardnog hrvatskog jezika. Rad je moje autorsko djelo i svi su preuzeti citati i parafraze u njemu primjereno označeni.

Mjesto i datum	Ime i prezime studenta/ice	Potpis studenta/ice
U Bjelovaru, 24.10.2022.	Dalibor Turbeki	Dturbeki

Prema Odluci Veleučilišta u Bjelovaru, a u skladu sa Zakonom o znanstvenoj djelatnosti i visokom obrazovanju, elektroničke inačice završnih radova studenata Veleučilišta u Bjelovaru bit će pohranjene i javno dostupne u internetskoj bazi Nacionalne i sveučilišne knjižnice u Zagrebu. Ukoliko ste suglasni da tekst Vašeg završnog rada u cijelosti bude javno objavljen, molimo Vas da to potvrdite potpisom.

Suglasnost za objavljivanje elektroničke inačice završnog rada u javno dostupnom nacionalnom repozitoriju

Dalibor Turbeki

ime i prezime studenta/ice

Dajem suglasnost da se radi promicanja otvorenog i slobodnog pristupa znanju i informacijama cjeloviti tekst mojeg završnog rada pohrani u repozitorij Nacionalne i sveučilišne knjižnice u Zagrebu i time učini javno dostupnim.

Svojim potpisom potvrđujem istovjetnost tiskane i elektroničke inačice završnog rada.

U Bjelovaru, 24.10.2022.

D Turbeki

potpis studenta/ice