SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Iva Udovčić

SUČELJA (INTERFACES) U SOLIDITYJU

PROJEKT

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Iva Udovčić

Matični broj: 0016148057

Studij: Organizacija poslovnih sustava

SUČELJA U SOLIDITYJU

PROJEKT

Mentorica:

mag. inf. Snježana Križanić

Varaždin, lipanj 2024.

Iva Udovčić

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Tema rada su sučelja u Solidityu. Sukladno temi u radu je ukratko prikazan Solidity te njegove prednosti i mane. Osim toga objašnjen je i blockchain te pametni ugovori koji bez blockchaina nemaju tehnologiju koja bi ih podržala. U radu se spominju i razvojna okruženja za izvršavanje pametnih ugovora te su navedeni primjeri sučelja. Ukupno je navedeno četiri primjera od kojih svaki prikazuje neku bitnu karakteristiku sučelja. Neke karakteristike koje se prikazuju su mogućnost nasljeđivanja sučelja te mogućnost da ugovor naslijedi sučelje pomoću ključne riječi is. Osim toga, prikazana su i dva načina pisanja sučelja, a to su odvojeno od ugovora te u istoj datoteci kao i ugovor. Na kraju rada se nalazi zaključak.

Ključne riječi: Solidity, blockchain, pametni ugovor, sučelje, razvojno okruženje

Sadržaj

1. Uvod

Ovaj projekt bavi se temom sučelja u programskom jeziku Solidity. Rad na početku govori nešto o samom Solidityju te o blockchainu i pametnim ugovorima kao neizostavnim dijelovima tog programskog jezika. Ukratko je objašnjeno što je to blockchain te kako funkcionira i čemu služi. Nakon toga objašnjen je Solidity te su navedeni neki kratki primjeri koda koji prikazuju varijable, funkcije i ugovore. Solidity je jezik u kojem se stvaraju pametni ugovori koji su relativno novi pojam te postaju sve više korišteni. Pametni ugovori imaju široku primjenu i imaju mogućnosti trajnog čuvanja podataka, no tu se javlja pitanje GDPR-a.

Ethereum je mreža pametnih ugovora, odnosno platforma za izvršavanje tih ugovora. EVM je najpoznatiji dio te platforme koji izvršava pametne ugovore i obrađuje transakcije. Remix je razvojno okruženje koje omogućava programerima da lokalno provjere svoje ugovore prije prebacivanja na Ethereum blockchain. Remixu se može pristupiti preko preglednika.

Sučelja u Solidityju omogućavaju veću interoperabilnost te modularnost. Povećavaju interakciju između ugovora te smanjuju potrebu za stalnim deklariranjem istih funkcija. Karakteristično za sučelja je da ne sadrže implementacije funkcija te na taj način olakšavaju testiranja.

U radu su navedena četiri primjera sučelja, u primjerima se koriste *enum* i *struct* u sučelju te se prikazuje kako izgleda kad ugovor nasljeđuje sučelje te kad sučelje nasljeđuje drugo sučelje. Prikazan je i primjer gdje se sučelje i ugovor nalaze u istoj datoteci te primjeri gdje su u odvojenim datotekama. Drugi način se više prakticira.

2. Blockchain

Predstavlja knjigu transakcija koja omogućuje praćenje imovine u poslovnoj mreži. Moguće je pratiti materijalnu, ali i nematerijalnu imovinu odnosno sve što ima vrijednost. Blockchain smanjuje rizike i troškove za sve uključene. Posebno je važan kod razmjene informacija budući da se svaki posao temelji na informacijama te njihova razmjena treba biti što brža i što točnija. Blockchain knjizi mogu pristupiti samo ovlašteni članovi mreže. Osim praćenja informacija moguće je pratiti i narudžbe, plaćanja, račune, proizvodnju i slično. Ključni dijelovi Blockchaina su tehnologija, zapisi i pametni ugovori. Kad se spominje tehnologija misli se na tehnologiju distribuirane knjige kojoj svi sudionici mogu pristupiti. U zajedničkoj distribuiranoj knjizi svi zapisi se bilježe samo jednom. Zapisi u toj knjizi su nepromjenjivi što zapravo znači da niti jedan sudionik koji ima pravo pristupa knjizi ne može mijenjati transakciju koja je zapisana. Ukoliko se dogodi da zapis ima grešku dodaje se nova transakcija koja je ispravna, znači obje transakcije su vidljive samo što novo dodana sadržava ispravne podatke. Radi bržeg obavljanja transakcija koriste se pametni ugovori koji predstavljaju pravila po kojima se upravlja blockchainom. Neka od pravila mogu biti i načini prijenosa korporativnih obveznica poput uvjeta plaćanja putnog osiguranja i slično (IBM, b.d).

Način funkcioniranja blockchaina se može sažeti u tri glavna koraka. Prvi korak je bilježenje transakcija kao blokova. U blok se spremaju informacije koje su od značaja za tvrtku te se informacije mogu razlikovati prema različitim potrebama. Informacije mogu biti vezane uz transakcije materijalne ili nematerijalne imovine. Informacije koje se bilježe mogu odgovarati na pitanje tko, što, kada, gdje, koliko. Primjer informacije je temperatura pošiljke hrane. Još jedan razlog zbog kojeg blockchain sadržava riječ lanac u sebi je taj da blokovi čine lanac. Blokovi su povezani jedni s drugima i tvore lanac kako se imovina kreće s mjesta na mjesto. Blokovi prikazuju točno vrijeme i redoslijed transakcija, a povezani su kako bi se spriječila promjena nekog bloka ili čak umetanje blokova između postojećih. Dodavanjem novih blokova se ustvari jača lanac budući da svaki sljedeći provjerava prethodni. To omogućava nepromjenjivost, sprječava neovlašteni pristup i mijenjanje od strane zlonamjernih softvera (IBM, b.d).

Neke od prednosti blockchaina su veće povjerenje u podatke i informacije koje prima član mreže te sigurnost da podaci neće procuriti van mreže. Osim toga, sigurnost blockchaina leži i u tome da svi članovi mreže moraju potvrditi točnost podataka te u tome što su sve transakcije nepromjenjive, čak ih ni administrator sustava ne može izbrisati. Blockchain povećava učinkovitost jer nema potrebe za usklađivanjem zapisa te se koriste pametni ugovori za brže i automatizirano izvođenje transakcija (IBM, b.d).

Postoji nekoliko vrsta blockchain mreža to jest izgradnje mreže. Prva mreža koja će biti ukratko objašnjenja je javna mreža. Najbolji primjer te mreže je Bitcoin budući da se svatko može pridružiti. Nedostatci takve mreže su privatnost transakcije, sigurnost i potreba za računalnom snagom. Privatna mreža je nešto sigurnija od javne te je decentralizirana peer-to-peer mreža. Ovom mrežom upravlja jedna organizacija te to može značiti jače povjerenje sudionika jer organizacija daje pristup sudionicima te održava knjigu transakcija. Privatni blockchain može biti pokrenut iza korporativnog vatrozoida. Još jedna vrsta blockchain mreža je dopuštena blockchain mreža. Ova mreža se postavlja uz privatnu mrežu budući da postavlja dopuštenja, no može se postaviti i na javnu mrežu. Sudionici dobivaju pozivnicu ili dozvolu za pridruživanje. Konzorcij blockchains se koristi kako bi više sudionika bilo odgovorno za blockchain, ali i kako bi imali dopuštenja. Najčešće se odgovornosti dijeli među organizacijama (IBM, b.d).

3. Solidity

Relativno novi programski jezik koji se brzo razvija. Razvio ga je Ethereum 2015. godine. Neke bitne karakteristike su da je to programski jezik visoke razine te da je dizajniran za kreiranje pametnih ugovora, objektno je orijentiran te statički tipiziran. Nastao je prema Pythonu, C++ i JavaScriptu. Podržava korisnički definirano programiranje, biblioteke i nasljeđivanje. Koristi se za glasanje, aukcije na slijepo, grupno financiranje i novčanike s više potpisa (GeeksforGeeks, 2023).

Solidity je i drugo najveće tržište kripto valuta. Temeljni je jezik na Ethereumu. Podržava uobičajene tipove podataka poput: boolean, integer (uint), string, modifer (prije izvršenja pametnog ugovora provjerava je li neki uvjet racionalan), array. U Solidityju je moguće mapiranje s enumima, operatorima i hash vrijednostima kako bi se vratile vrijednosti pohranjene na određenim mjestima za pohranu (Simplilearn, 2023).

Ono s čime započinje svaka .sol datoteka je (Simplilearn, 2023):

```
pragma solidity >=0.4.16 <0.7.0;
```

Pragma zapravo označava verziju prevoditelja, a time se nastoji spriječiti nekompatibilnost koda s budućim verzijama prevoditelja.

Ugovor započinje riječju *contract* (Simplilearn, 2023):

```
Contract Primjer{
//Kod
}
```

Varijable se deklariraju kako je prikazano u retku niže. Varijable stanja se trajno čuvaju (Simplilearn, 2023).

```
uint public var1;
uint public var2;
uint public sum;
```

Sve navedene varijable su tipa integer.

Funkcija koja je navedena niže zove se set, a modifikator pristupa je public. Varijable a i b su tipa integer i predstavljaju parametre funkcije (Simplilearn, 2023).

```
function set(uint a, uint b) public
```

Kod u Solidityju se može izvršiti izvanmrežno i online. U izvanmrežnom načinu rada potrebno je instalirati node.js, Truffle i ganache-cli. Nakon instalacije potrebno je napraviti

truffle projekt, razviti pametni ugovor te s njim komunicirati preko Truffle konzole. U online načinu rada koristi se Remix IDE (Simplilearn, 2023).

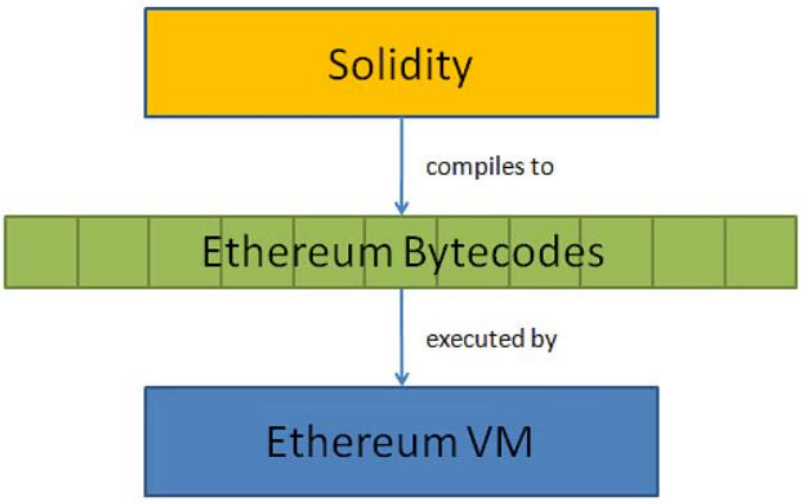
Prednosti Solidity programiranja: dopušta složene tipove podataka i varijable članova, omogućava ABI (eng, *Application binary interface*) odnosno aplikacijsko binarno sučelje koje generira pogrešku ukoliko kompajler otkrije nepodudarnost tipa podataka (Simplilearn, 2023).

3.1. Ethereum

Decentralizirana platforma koja izvršava kod koji se zove pametni ugovor. Koristi valutu Ether koju korisnik troši kad izvršava transakcije. Koristi se za kreiranje decentraliziranih aplikacija koje se još zovu i dapps. Osim toga koristi se pokretanje pametnih ugovora. Ethereum je platforma za više kripto valuta (GeeksforGeeks, 2023).

Ethereum je trenutno najvrijednija mreža pametnih ugovora na svijetu. Krajem lipnja prošle godine Ethereumov dapps vrijedio je 24,63 milijarde USD, dok drugoplasirani Tron ima vrijednost 5 milijardi. Takva razlika prikazuje zašto se Ethereum toliko koristi. Novonastali kripto projekti uključuju kompatibilnost s EVM-om te tako omogućuju jednostavno implementiranje pametnih ugovora. Neki lanci koji su kompatibilni s EVM-om su: BNB Chain, Polygon PoS, Optimism i mnogi drugi (Lepcha, 2023).

Ethereum Virtual Machine (EVM) je okruženje za izvođenje pametnih ugovora. Usredotočeno je na sigurnost i omogućuje izvršavanje nepouzdanog koda pomoću mreže javnih čvorova. Sprječava Denial-of-Service napade te osigurava da programi nemaju pristup međusobnom stanju (GeeksforGeeks, 2023).



Slika 1: Ethereum (Simplileran, 2023)

EVM je glavni dio Ethereuma jer izvršava pametne ugovore i obrađuje transakcije. To je virtualni stroj koji ima memoriju i izvršava kod kao i računalo. Zapravo, EVM je više računala diljem svijeta. Glavne funkcionalnosti su: upravljanje stanjem Ethereum blockchaina, pohranjivanje podataka, izvršavanje pametnih ugovora i obračun naknada za gas (Lepcha, 2023).

3.2. Kako funkcionira EVM

Koristi jezik Solidity u kojem se pišu pametni ugovori. Kod kojim je napisan pametni ugovor se prevodi u bajt kod koji EVM razumije. Prilikom prevođenja koda iz Solidityja u bajt kod prvo je potrebno razdijeliti taj kod na operacijske kodove, a to znači da se svaki redak koda pretvori u operativni kod. Budući da svaka transakcija troši gorivo potrebno je razumjeti odnos operacijskih kodova i naknada za gorivo jer kad se plaća naknada za gorivo ustvari se plaćaju operacijski kodovi koje izvršava EVM. Više operacijskih kodova znači veće troškove. Ne troše sve transakcije istu količinu goriva, transakcija koja šalje ETH s jednog na drugi račun troši manje nego transakcija koja stvara pametni ugovor (Lepcha, 2023).

3.3. Značajke EVM-a i arhitektura

EVM je ustvari Turingov kompletan virtualni stroj, a potpunost se odnosi na činjenicu da može riješiti bilo kakav problem ako ima dovoljno resursa (vremena, energije i slično). Kod Ethereuma to se odnosi na sposobnost mreže da implementira pametne ugovore. Još jedna značajka je da se pametni ugovori izvode deterministički, a to ustvari znači da za iste ulaze daju iste izlaze bez obzire gdje se izvode i tko ih pokreće. Osim toga pruža izolaciju jer se ugovori izvode u sigurnom okruženju (Lepcha, 2023).

Što se tiče arhitekture tu su: EVM kod, stanje, transakcija, prostor i gorivo. Svaki od ovih elemenata arhitekture će u nastavku ukratko biti objašnjen (Lepcha, 2023).

- EVM kod: bajt kod koji EVM izvršava,
- Stanje: označava stanje računa i salda koji se mijenjaju od bloka do bloka budući da se konstantno obrađuju ulazi,
- Transakcije: kriptografske upute korisnika, mogu biti:
    - Pozivi putem poruka: poput slanja ETH s jednog računa na drugi
    - Stvaranje ugovora: novi pametni ugovor,
- Prostor:
    - Stack - privremeno skladište, kapacitet 1024 predmeta. Tu se obavljaju sve operacije.

    - Memory – privremena memorija dostupna tijekom izvođenja pametnih ugovora
    - Storage – trajna memorija, veće naknade za gorivo,
- Gorivo: računalni napor za izvršenje operacija.

3.4. Remix IDE

To je Ethereum IDE otvorenog koda koji se odnosi na Web3, dApps i razvoj blockchaina. Osim što je razvojno okruženje to je i desktop aplikacija s bogatim skupom podatak te intuitivnim GUI-em. Koristi se za razvoj pametnih ugovora (moralis, 2021).

Iako se Remix IDE i Remix koriste kao sinonimi to nije isto. Remix IDE je dio Remix Projecta koji služi za razvojne alate odnosno to je platforma za razvojne alate koja radi na temelju arhitekture dodataka. Remix Project osim Remix IDE uključuje i Remix Plugin Engine i Remix Libs (moralis, 2021).

Remix IDE omogućuje kodiranje unutar preglednika, ali postoji i verzija za desktop. Napisan je u JavaScriptu. Sadržava tri modula, a to su modul za testiranje, otklanjanje grešaka te implementaciju pametnih ugovora. Remix ima nekoliko biblioteka (moralis, 2021):

- **Remix Analyzer:** za statičku analizu pametnih ugovora, provjerava sigurnosne ranjivosti,
- **Remix ASTWalker**: za čitanje AST ugovora,
- **Remix Debug:** omogućava dodavanje značajki za otklanjanje pogrešaka za pametne ugovore,
- **Remix Solidity:** omot oko Solidity kompajlera,
- **Remix Lib:** mjesto za biblioteke koje se koriste u više modula,
- **Remix testovi**: dodaje jedinično testiranje,
- **Remix URL Resolvers:** pomoćnici za rješavanje sadržaja s vanjskih URL-ova.

Razlika između EVM-a i Remixa je ta što je EVM stvarni izvršni stroj koji osigurava izvršavanje ugovora u distribuiranom i sigurnom okruženju, a Remix je razvojno okruženje koje omogućava da se ugovori pišu i testiraju lokalno prije nego se implementiraju na pravi Ethereum blockchain (moralis, 2021).

4. Pametni ugovori

Još se zovu i kripto ugovori, provjeravaju vjerodostojnost transakcije bez trećih strana. Prema njima se određuju pravila za prijenos digitalne imovine. Pametni ugovori nalaze se u blockchainu koji se pokreće nakon ispunjenja uvjeta te nakon provedenih provjera. Ovlašteni sudionici vide rezultate, a nakon transakcije ako neka strana ne obavi svoj dio pogodbe novac se šalje natrag (PLURALSIGHT, 2023).

Prednosti pametnih ugovora su: brzina, točnost i učinkovitost budući da se izvršavaju odmah nakon ispunjenja uvjeta. Digitalni su i automatizirani pa je manja mogućnost pogreške. Uključeni sudionici su jedini koji imaju uvid u transakcije što znači da treće strane ne mogu mijenjati podatke. Svi zapisi su šifrirani i nepromjenjivi, ukoliko haker upadne u sustav mora promijeniti cijeli lanac budući da su svi blokovi povezani. Osim toga, jeftini su jer nema potrebe za trećim stranama niti plaćanja naknada (PLURALSIGHT, 2023).

Nedostatak pametnih ugovora je nemogućnost popravljanja greške, iako je to prije navedeno kao prednost to ustvari ima i svoje mane. Jedna od mana te karakteristike pametnih ugovora je ta što popravljanje grešaka može biti skupo i dugotrajno. Osim toga, ugovori ne zadovoljavaju zakone svih zemalja te je teško jamčiti da će ih se poštovati i na globalnoj razini. Jedan od nedostataka pametnih ugovora je prikupljanje podataka, za tvrtke koje imaju kvantificirane podatke ovo nije problem, no za tvrtke čiji podaci nisu lako mjerljivi ovo može predstavljati prepreku. Iako su pametni ugovori vrlo sigurni, ne može se reći da su sigurni kao i GDPR. Naime, prema GDPR-u svaki građanin ima pravo biti zaboravljen, a pametni ugovori se ne mogu mijenjati što znači da podaci o nekom pojedincu trajno ostaju sačuvani. Stvaranje pametnih ugovora je posao za stručnjaka, no pronaći stručnjaka koji poznaje netradicionalne programske jezike kao što je Solidity je vrlo izazovno. Još jedan od nedostataka je skalabilnost budući da Ethereum u sekundi obradi samo 30 transakcija (BasuMallick, 2023).

Primjena pametnih ugovora je prilično velika neke od primjena su: u transportu lijekova gdje se prate podaci vezani uz temperaturu na kojoj se lijekovi nalaze i slično. Pametni ugovori se primjenjuju i u odnosima između trgovca i dobavljača, u međunarodnoj trgovini i slično (IBM; b.d).

Ideja pametnih ugovora rodila se 1998. godine kada je Nick Szabo razvio Bit Gold (virtualna valuta). U svom radu je predstavio pametne ugovore, no u to vrijeme nije postojala tehnologija koja bi podržala njihov razvoj. Pametni ugovori su ostali slovo na papiru do 2008. godine kada je Satoshi Nakamoto predstavio blockchain tehnologiju. Svoj značajniji razvoj pametni ugovori su doživjeli pet godina kasnije pojavom Ethereum blockchain platforme (BasuMallick, 2023).

4.1. Kako funkcioniraju pametni ugovori

Pametni ugovor slijedi upute napisane na blockchainu poput if, if else i slično. Nakon što je transakcija izvršena blockchain se ažurira te se dodaje novi blok koji svjedoči da se transakcija dogodila, naravno taj blok se ne može mijenjati. Rezultate transakcije vide jedino sudionici koji imaju dopuštenje. Broj i vrsta uvjeta za izvršenje pametnog ugovora ovisi o sudionicima i podacima koji su im potrebni. Osim toga, sudionici definiraju i kako su transakcije predstavljene na blockchainu. Posljednji korak je programiranje pametnog ugovora iako sve više organizacija koristi već postojeće predloške (IBM, b.d).

Funkcioniranje pametnih ugovora može se sažeti u nekoliko koraka. Prvi korak je dogovor između strana koje sklapaju ugovor, stvari oko kojih se trebaju dogovoriti su kako će pametni ugovor funkcionirati i koji uvjeti trebaju biti ispunjeni da bi ugovor bio ispunjen. Nakon dogovora slijedi korak u kojem se ugovor kreira, strane to mogu napraviti same ili uz pomoć neke treće strane. U ovom koraku se posebna pažnja posvećuje sigurnosti. Treći korak je objavljivanje ugovora odnosno učitavanja u blockchain. Učitavanje je isto kao i kod učitavanja kripto valuta. Kad je učitavanje gotovo odnosno transakcija je gotova ugovor se ne može izmijeniti niti poništit. Pametni ugovor prati blockchain te se okidači mogu digitalno potvrditi, a to je ujedno i četvrti korak. U petom koraku kada se okine neki okidač ugovor se izvrši, a šesti korak je ustvari bilježenje rezultata izvršenja ugovora na blockchainu. Blockchain nakon toga provjerava izvršenje te pohranjuje ugovor (BasuMallick, 2023).

4.2. Vrste pametnih ugovora

Postoje tri kategorije pametnih ugovora, a to su: pravni, decentralizirana autonomna organizacija (DAO) i logički ugovor. U nastavku je svaki ukratko objašnjen (BasuMallick, 2023):

Pametni pravni ugovor: zajamčen je zakonom te nudi veću transparentnost od tradicionalnih ugovora. Ugovor se sklapa digitalnim potpisom te se može izvršavati samostalno uz prethodno zadovoljenje uvjeta.

Decentralizirane autonomne organizacije (DAO): organizacija kojom upravlja pametni ugovor i nema predsjednika. Dakle, sva načela organizacije ugrađena su u ugovor te se na taj način reguliraju sredstva i tako organizacija funkcionira.

Logički ugovor (ALC): ovaj ugovor se potpisuje između strojeva i drugih ugovora, a omogućuje interakciju uređaja (IoT).

5. Sučelja (eng. Interfaces) u Solidityju

U programskom jeziku Solidity moguće je komunicirati s drugim ugovorima bez njihovog koda tako da se koristi njihovo sučelje. Sučelja omogućuju suradnju unutar ekosustava Ethereum. Ono ustvari predstavlja skicu deklaracija funkcija kojih se ugovori pridržavaju te na taj način omogućuju suradnju bez napora i potiču interoperabilnost (Sharedeum, 2023).

Sučelje se kreira koristeći riječ *interface*, te se nakon te ključne riječi definira naziv tog sučelja. Nakon toga deklariraju se funkcije i događaji koje drugi ugovori trebaju imati te se navode njihovi potpisi. Kad je sučelje implementirano mogu ga koristiti drugi ugovori tako da sadržavaju riječ *is* pa naziv sučelja. Osim *is* sadržavaju i tijelo funkcije. Na taj način je omogućena interakcija te suradanja. U sučelju se izostavlja kod te se funkcije deklariraju kao vanjske (Sharedeum, 2023).

Okrenutost sučelja prema van čini ih dostupnim vanjskim entitetima za interakciju s ugovorima te se potiče kompatibilnost ugovora i olakšava se suradnja te se promovira modularni pristup (Sharedeum, 2023).

Sučelja u Solidityju su od krucijalne važnosti za pametne ugovore te potiču interoperabilnost između ugovora. Omogućuju interakciju ugovora bez izlaganja internih detalja, omogućuju pridržavanje zajedničkog jezika, olakšavaju kompatibilnost ugovora i integraciju unutra složenih sustava (Sharedeum, 2023).

Razlika između nasljeđivanja i sučelja je u tome što nasljeđivanje omogućava ponovnu upotrebu koda i produljenje ugovora, a sučelje omogućava standardizaciju komunikacije i interoperabilnost ugovora (Sharedeum, 2023).

Karakteristike sučelja su (Alchemy, 2022):

- Sučelja u Solidityju mogu naslijediti druga sučelja,
- Ugovori mogu nasljeđivati sučelja, obrnuto ne vrijedi,
- Funkcija sučelja se može nadjačati,
- Iz drugih ugovora se može pristupiti tipovima podataka u sučelju.

Razlika apstraktnih ugovora i sučelja je ta što sučelja ne mogu imati implementiranu funkciju te su ograničena na ono što ABI može predstavljati, a apstraktni ugovori imaju barem jednu funkciju koja nije implementirana te od njih drugi ugovori mogu nasljeđivati (Alchemy, 2022).

Već je spomenuto kako sučelje ne može imati implementirane funkcije, no to nije jedini zahtjev sučelja. Naime, funkcije sučelja mogu biti samo *external*, sučelje ne može deklarirati konstruktor i varijable stanja. Sučelje može imati *enum* i *struct* kojima se može pristupiti koristeći točku pa ime sučelja (Alchemy, 2022).

Prednost sučelja je i ta što omogućuje lakše testiranje ugovora. Testiranje se može lakše provesti jer se testni slučajevi pišu tako da pozivaju funkcije iz sučelja. Na osnovu rezultata može se zaključiti je li sve kako treba, a da se pritom ne poznaje implementacija ugovora (Kacar, 2023).

Sučelja u Solidityju su nešto drugačija nego kod drugih programskih jezika poput Jave ili C#. U Solidityju sučelja predstavljaju samo popis funkcija te ne postoji nasljeđivanje ili polimorfizam te se ne mogu stvoriti instance sučelja. Kod ugovora sučelja se prevode u konačni bajt, a to znači da ako se poziva funkcija definirana u sučelju iz drugog ugovora upotrebljava se notacija *interface.imeFunkcije* (Kacar, 2023).

5.1. Primjer 1

Ovaj primjer prikazuje sučelje IPrisutnostStudent koje ima tri funkcije, a to su dodajStudenta, azurirajPrisustvo i dohvatiStudenta. Sve funkcije su *external* kako i mora biti kada se radi o sučelju. Sučelje je implementirano u zasebnoj datoteci od ugovora. Preporuke su da se sučelje i ugovor nalaze u posebnim datotekama, no to ne mora uvijek biti tako. Funkcija dodajStudenta prima studentov jmbag, imePrezime i prisutnost za parametre, funkcija azurirajStudenta na osnovu studentovog jmbag-a ažurira prisutnost. Funkcija dohvatiStudenta na osnovu studentovog jmbag-a prikazuje podatke o studentu. Prisutnost je tipa bool pa tako može biti false ili true.

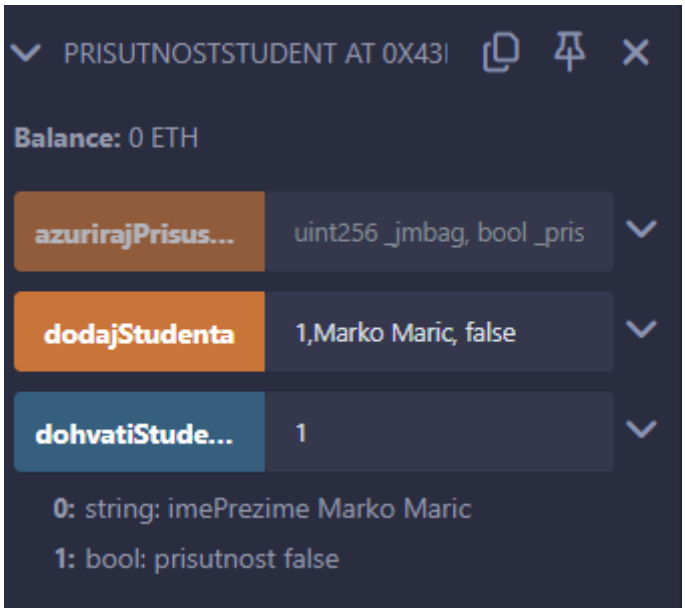
```
pragma solidity >=0.7.0 <0.9.0;
interface IPrisutnostStudent {
function dodajStudenta
(uint _jmbag, string calldata _imePrezime, bool _prisutnost) external;
function azurirajPrisustvo
     (uint _jmbag, bool _prisutnost) external;
function dohvatiStudenta(uint _jmbag) external view returns (string
memory imePrezime, bool prisutnost);
}
```

U ugovoru se nasljeđuje sučelje pomoću ključne riječi *is*. U ugovoru se također nalazi i *struct* te implementacija funkcija koje su u sučelju samo deklarirane. Pomoću riječi *import* označavamo koje sučelje se treba koristiti. U funkciji dodajStudenta linija

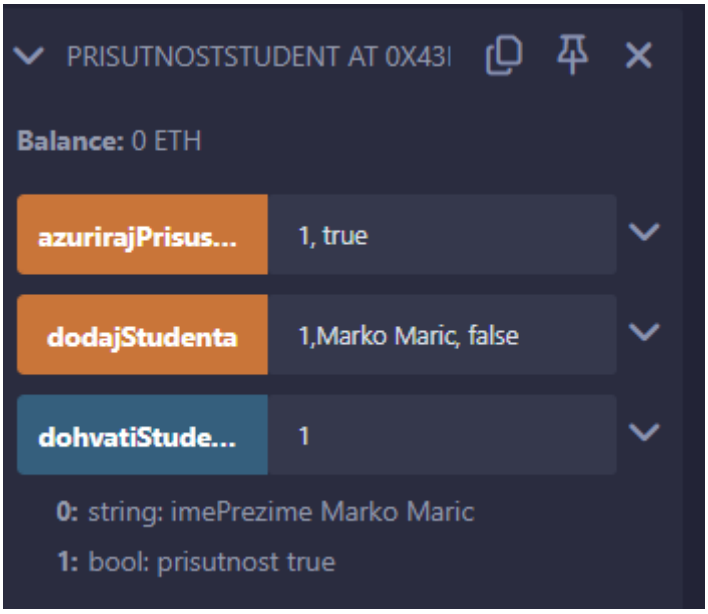
require(bytes(studenti[_jmbag].imePrezime).length == 0, "Student vec postoji"); osigurava da se ne doda dva puta student s istim jmbag-om. Linija koda *require(bytes(studenti[_jmbag].imePrezime).length != 0, "Student ne postoji");* osigurava da se ažuriranje i dohvaćanje izvrše nad jmbag-om koji postoji.

```
pragma solidity >=0.7.0 <0.9.0;
import "./pr1_sbool_sucelje.sol";
contract PrisutnostStudent is IPrisutnostStudent {
    struct Student {
        string imePrezime;
        bool prisutnost;
    }
    mapping(uint => Student) private studenti;

    function dodajStudenta(uint _jmbag, string calldata _imePrezime,
bool _prisutnost) external override {
        require(bytes(studenti[_jmbag].imePrezime).length == 0, "Student
vec postoji");
        studenti[_jmbag] = Student(_imePrezime, _prisutnost);
    }

    function azurirajPrisustvo(uint _jmbag, bool _prisutnost) external
override {
        require(bytes(studenti[_jmbag].imePrezime).length != 0, "Student
ne postoji");
        studenti[_jmbag].prisutnost = _prisutnost;
    }

    function dohvatiStudenta(uint _jmbag) external view override returns
(string memory imePrezime, bool prisutnost) {
        require(bytes(studenti[_jmbag].imePrezime).length != 0, "Student
ne postoji");
        Student memory student = studenti[_jmbag];
        return (student.imePrezime, student.prisutnost);
    }
}
```

Slika 2 prikazuje dodavanje studenta s jmbag-om 1, ime i prezime studenta je Marko Maric i prisutnost je *false*, nakon što se klikne na dodajStudenta može se upisati 1 pored dohvatiStudenta ta onda kliknuti na gumb dohvatiStudenta kako bi se dohvatila prethodno upisana vrijednost. Na slici 3 vidi se ažuriranje prisutnosti na true te se ponovno dohvati student.



Slika 2: Dodavanje i dohvaćanje studenta Primjer 1 (samostalna izrada)



Slika 3: Ažuriranje studenta Primjer 1 (samostalna izrada)

5.2. Primjer 2

U drugom primjeru prikazano je kako izgleda kada sučelje nasljeđuje sučelje te se sučelje i ugovor nalaze u istoj datoteci. Dodano je još jedno sučelje koje se zove INasljedujePrisutnostStudent koje ima deklariranu funkciju obrisiStudenta koja na osnovu jmbag-a studenta obriše studenta. Sučelje INasljedujePrisutnostStudent pomoću ključne riječi *is* nasljeđuje sučelje IPrisutnostStudent. Ugovor PrisutnostStudent nasljeđuje sučelje

INasljedujePrisutnostStudent. Kao i u prethodnom primjeru koristi se linija koda *require(bytes(studenti[_jmbag].imePrezime).length != 0, "Ne postoji");* koja ima istu svrhu kao i u prethodnom primjeru.

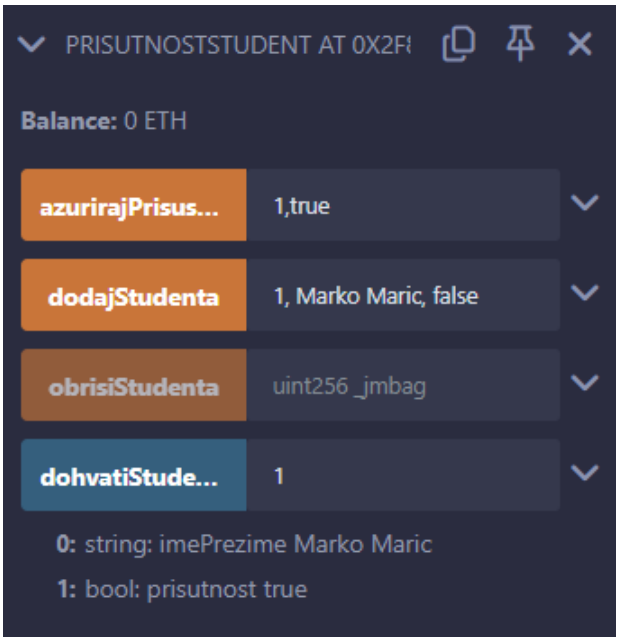
```
pragma solidity >=0.7.0 <0.9.0;


interface IPrisutnostStudent {
    function dodajStudenta(uint _jmbag, string calldata _imePrezime,
bool _prisutnost) external;
    function azurirajPrisustvo(uint _jmbag, bool _prisutnost) external;
    function dohvatiStudenta(uint _jmbag) external view returns (string
memory imePrezime, bool prisutnost);
}


interface INasljedujePrisutnostStudent is IPrisutnostStudent {
    function obrisiStudenta(uint _jmbag) external;
}


contract PrisutnostStudent is INasljedujePrisutnostStudent {
    struct Student {
        string imePrezime;
        bool prisutnost;
    }


    mapping(uint => Student) private studenti;


    function dodajStudenta(uint _jmbag, string calldata _imePrezime,
bool _prisutnost) external override {
        studenti[_jmbag] = Student(_imePrezime, _prisutnost);
    }


    function azurirajPrisustvo(uint _jmbag, bool _prisutnost) external
override {
        require(bytes(studenti[_jmbag].imePrezime).length != 0, "Ne
postoji");
        studenti[_jmbag].prisutnost = _prisutnost;
    }


    function dohvatiStudenta(uint _jmbag) external view override returns
(string memory imePrezime, bool prisutnost) {
        require(bytes(studenti[_jmbag].imePrezime).length != 0, "Ne
postoji");
```

```
        Student memory student = studenti[_jmbag];
        return (student.imePrezime, student.prisutnost);
    }

    function obrisiStudenta(uint _jmbag) external override {
        require(bytes(studenti[_jmbag].imePrezime).length != 0, "Ne
postoji");
        delete studenti[_jmbag];
    }
}
```

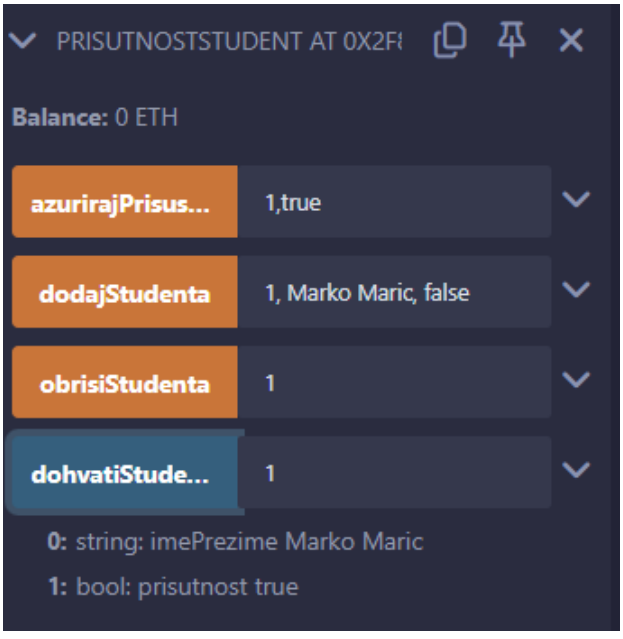
Slika 4 prikazuje isto što i slika 2 samo što se na ovoj slici nalazi još i gumb za brisanje studenta.

Slika 4: Dodavanje i dohvaćanje studenta Primjer 2 (samostalna izrada)

Slika 5 prikazuje ažuriranje prisutnosti studenta koji je na slici 4 dodan. Slika 6 prikazuje brisanje studenta s jmbag-om 1, a slika 7 prikazuje poruku koja se dobije ukoliko se pokuša dohvatiti student koji ne postoji.



Slika 5: Ažuriranje studenta Primjer 2 (samostalna izrada)



Slika 6: Brisanje studenta Primjer 2 (samostalna izrada)

```
call to PrisutnostStudent.dohvatiStudenta errored: Error occurred: revert.

revert
        The transaction has been reverted to the initial state.
Reason provided by the contract: "Ne postoji".
You may want to cautiously increase the gas limit if the transaction went out of gas.
```

Slika 7: Poruka nakon pokušaja dohvaćanja studenta koji je izbrisan Primjer 2 (samostalna izrada)

5.3. Primjer 3

Kako je već navedeno sučelje može sadržavat *struct*, pa sljedeći primjer prikazuje *struct* unutar sučelja. U ovom primjeru sučelje se nalazi u jednoj, a ugovor u drugoj datoteci. Sučelje je prikazano donjim kodom. Sve ostalo je isto kao i u prethodnom primjeru, linije koje provjeravaju da postoji student s određenim jmbag-om te linija koja ne dopušta unos studenta ako postoji student koji ima isti jmbag.

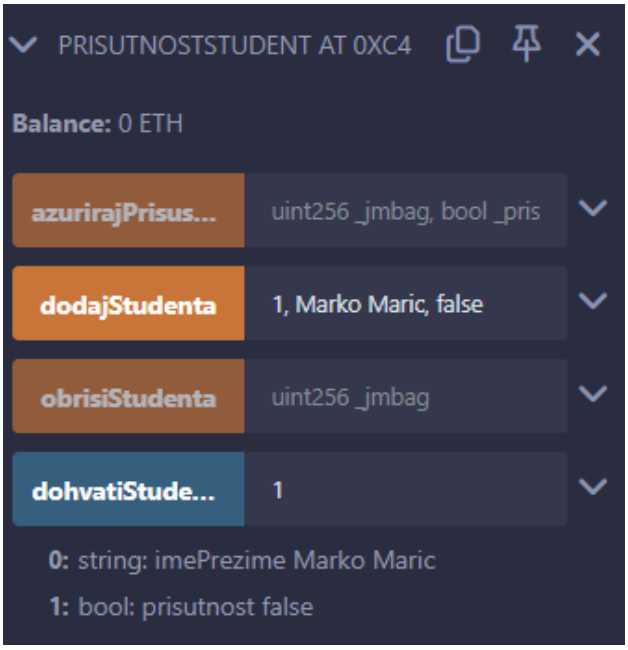
```
pragma solidity >=0.7.0 <0.9.0;

interface IPrisutnostStudent {
    struct Student {
        string imePrezime;
        bool prisutnost;
    }

    function dodajStudenta(uint _jmbag, string calldata _imePrezime,
bool _prisutnost) external;
    function azurirajPrisustvo(uint _jmbag, bool _prisutnost) external;
    function dohvatiStudenta(uint _jmbag) external view returns (string
memory imePrezime, bool prisutnost);
    function obrisiStudenta(uint _jmbag) external;
}
```

Donji kod prikazuje ugovor u kojem su implementirane funkcije iz sučelja.

```
pragma solidity >=0.7.0 <0.9.0;

import "./structsuceljebool.sol";

contract PrisutnostStudent is IPrisutnostStudent {
    mapping(uint => IPrisutnostStudent.Student) private studenti;

    function dodajStudenta(uint _jmbag, string calldata _imePrezime,
bool _prisutnost) external override {
        require(bytes(studenti[_jmbag].imePrezime).length == 0, "Student
vec postoji");
        studenti[_jmbag] = IPrisutnostStudent.Student(_imePrezime,
_prisutnost);
    }
```

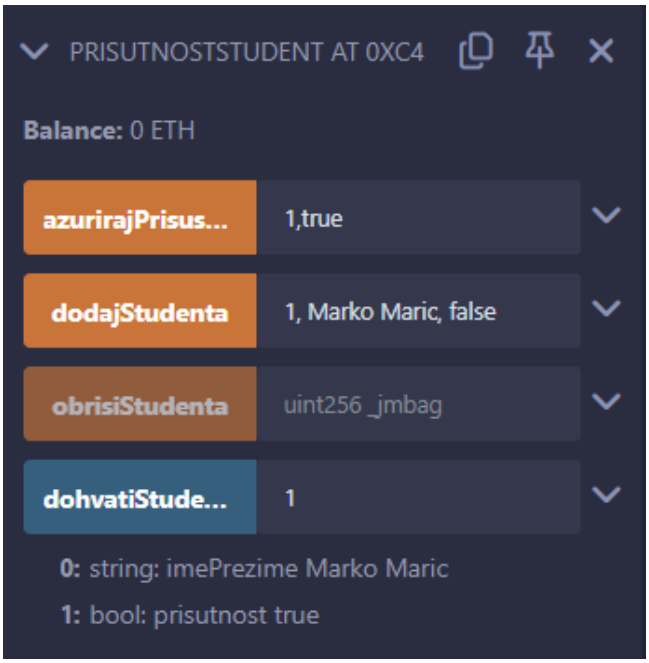
```
    function azurirajPrisustvo(uint _jmbag, bool _prisutnost) external
override {
        require(bytes(studenti[_jmbag].imePrezime).length != 0, "Student
ne postoji");
        studenti[_jmbag].prisutnost = _prisutnost;
    }

    function dohvatiStudenta(uint _jmbag) external view override returns
(string memory imePrezime, bool prisutnost) {
        require(bytes(studenti[_jmbag].imePrezime).length != 0, "Student
ne postoji");
        IPrisutnostStudent.Student memory student = studenti[_jmbag];
        return (student.imePrezime, student.prisutnost);
    }

    function obrisiStudenta(uint _jmbag) external override {
        require(bytes(studenti[_jmbag].imePrezime).length != 0, "Student
ne postoji");
        delete studenti[_jmbag];
    }
}
```

Slika 8 prikazuje dodavanje studenta te dohvaćanje studenta po njegovom jmbag-u.

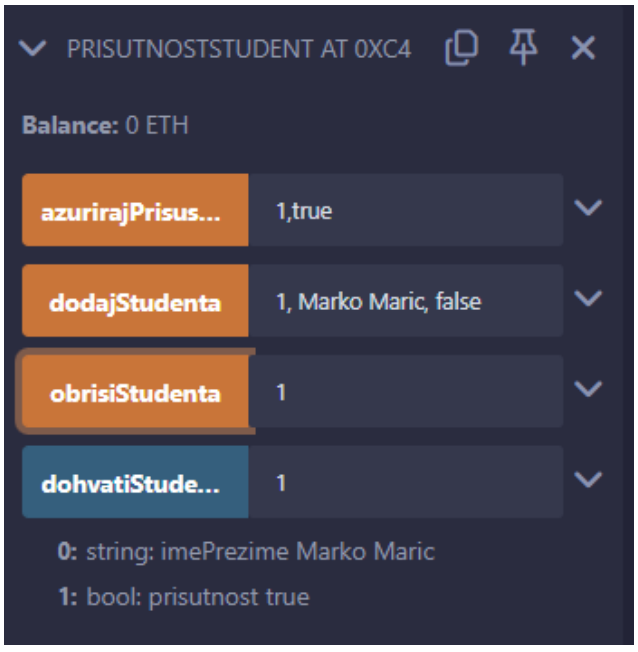


Slika 8: Dodavanje i dohvaćanje studenta Primjer 3 (samostalna izrada)

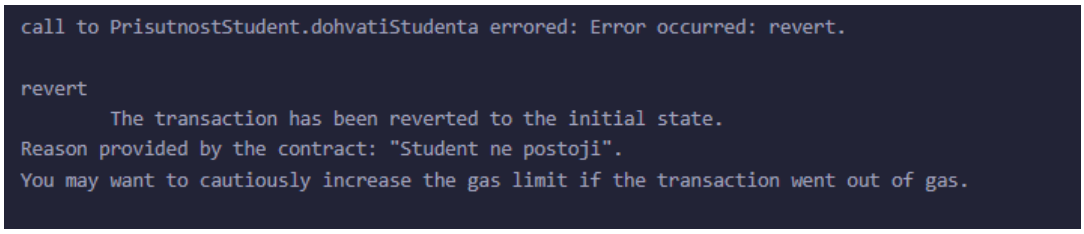
Slika 9 prikazuje ažuriranje prisutnosti studenta na true te dohvaćanje tog studenta. Slika 10 prikazuje brisanje studenta, a slika 11 prikazuje poruku koja se ispisuje u konzoli kad se pokuša dohvatiti student koje je prethodno obrisan.



Slika 9: Ažuriranje studenata Primjer 3 (samostalna izrada)



Slika 10: Brisanje studenta Primjer 3 (samostalna izrada)



Slika 11: Poruka nakon pokušaja dohvaćanja studenta koji je izbrisan (samostalna izrada)

5.4. Primjer 4

Ovaj primjer prikazuje sučelje u kojem se nalaze *enum* i *struct* te prikazuje kako se jedno sučelje može koristiti u više ugovora što je zapravo i svrha samog sučelja. Naime, u ovom primjeru su: ugovor vezan za redovite studente te ugovor vezan za izvanredne studente. Prisutnost više nije *bool* već je *enum* koji sadržava tri vrijednosti, a to su Odsutan, Prisutan i Nepoznato. U ugovor vezan za redovite studente se unose podaci o redovitim studentima, a u ugovor o izvanrednim studentima se unose podaci o izvanrednim studentima. Oba ugovora koriste funkcije deklarirane u sučelju. Sučelje izgleda ovako:

```
pragma solidity >=0.7.0 <0.9.0;

interface IPrisutnostStudent {
    enum PrisutnostStatus { Odsutan, Prisutan, Nepoznato }

    struct Student {
        string imePrezime;
        PrisutnostStatus prisutnost;
    }

    function dodajStudenta(uint _jmbag, string calldata _imePrezime)
external;
    function azurirajPrisustvo(uint _jmbag, PrisutnostStatus
_prisutnost) external;
    function dohvatiStudenta(uint _jmbag) external view returns (string
memory imePrezime, PrisutnostStatus prisutnost);
    function obrisiStudenta(uint _jmbag) external;
}
```

Ugovor za izvanredne studente izgleda ovako:

```
pragma solidity >=0.7.0 <0.9.0;
```

```
import "./4_primjer_sucelje.sol";


contract IzvanredniStudenti is IPrisutnostStudent {


    mapping(uint => IPrisutnostStudent.Student) private studenti;


    function dodajStudenta(uint _jmbag, string calldata _imePrezime)
external override {
        require(bytes(studenti[_jmbag].imePrezime).length == 0, "Student
vec postoji");
        studenti[_jmbag] = IPrisutnostStudent.Student(_imePrezime,
IPrisutnostStudent.PrisutnostStatus.Nepoznato);
    }


    function azurirajPrisustvo(uint _jmbag,
IPrisutnostStudent.PrisutnostStatus _prisutnost) external override {
        require(bytes(studenti[_jmbag].imePrezime).length != 0, "Student
ne postoji");
        studenti[_jmbag].prisutnost = _prisutnost;
    }


    function dohvatiStudenta(uint _jmbag) external view override returns
(string memory imePrezime, IPrisutnostStudent.PrisutnostStatus
prisutnost) {
        require(bytes(studenti[_jmbag].imePrezime).length != 0, "Student
ne postoji");
        IPrisutnostStudent.Student memory student = studenti[_jmbag];
        return (student.imePrezime, student.prisutnost);
    }


    function obrisiStudenta(uint _jmbag) external override {
        require(bytes(studenti[_jmbag].imePrezime).length != 0, "Student
ne postoji");
        delete studenti[_jmbag];
    }
}
```

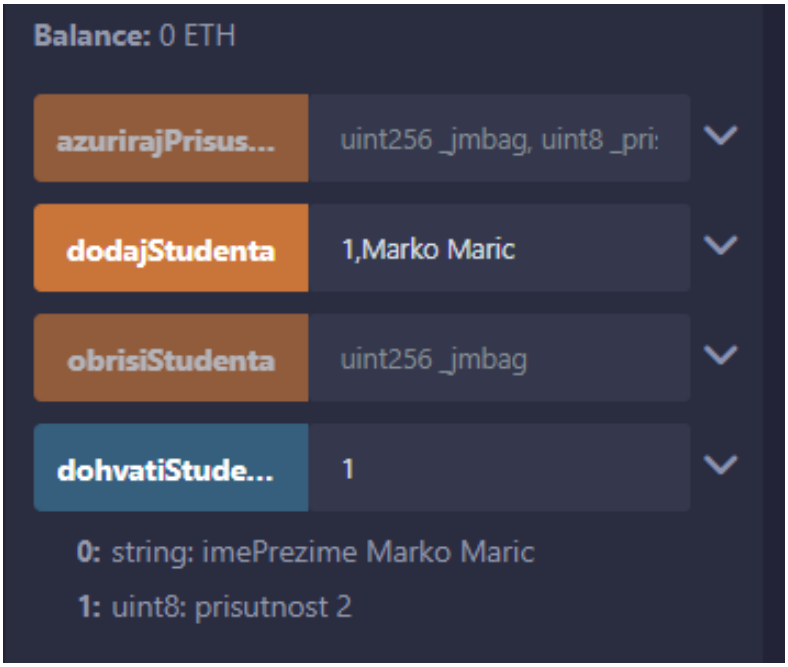
Ugovor za redovite studente izgleda ovako:

```
pragma solidity >=0.7.0 <0.9.0;


import "./4_primjer_sucelje.sol";
```

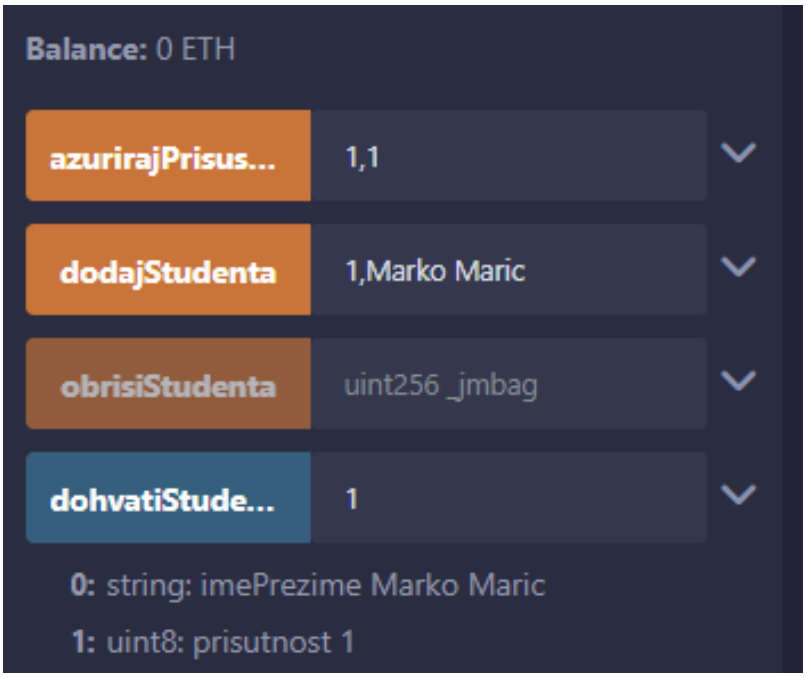
```
contract RedovniStudenti is IPrisutnostStudent {


    mapping(uint => IPrisutnostStudent.Student) private studenti;


    function dodajStudenta(uint _jmbag, string calldata _imePrezime)
external override {
        require(bytes(studenti[_jmbag].imePrezime).length == 0, "Student
vec postoji");
        studenti[_jmbag] = IPrisutnostStudent.Student(_imePrezime,
IPrisutnostStudent.PrisutnostStatus.Nepoznato);
    }


    function azurirajPrisustvo(uint _jmbag,
IPrisutnostStudent.PrisutnostStatus _prisutnost) external override {
        require(bytes(studenti[_jmbag].imePrezime).length != 0, "Student
ne postoji");
        studenti[_jmbag].prisutnost = _prisutnost;
    }


    function dohvatiStudenta(uint _jmbag) external view override returns
(string memory imePrezime, IPrisutnostStudent.PrisutnostStatus
prisutnost) {
        require(bytes(studenti[_jmbag].imePrezime).length != 0, "Student
ne postoji");
        IPrisutnostStudent.Student memory student = studenti[_jmbag];
        return (student.imePrezime, student.prisutnost);
    }


    function obrisiStudenta(uint _jmbag) external override {
        require(bytes(studenti[_jmbag].imePrezime).length != 0, "Student
ne postoji");
        delete studenti[_jmbag];
    }
}
```

Na donjoj slici prisutnost 2 znači da je Nepoznato budući da je prisutnost *enum*, a Nepoznato odgovara poziciji 2, jer je Odsutan na poziciji 0, a Prisutan na poziciji 1. Osim toga slika prikazuje dodavanje studenta pomoću funkcije dodajStudenta te dohvaćanje dodanog studenta pomoću funkcije dohvatiStudenta. Slika 13 prikazuje ažuriranje prisutnosti studenta tako da se prisutnost postavi na 1 odnosno na Prisutan. Slika 14 prikazuje brisanje studenta s jmbag-om 1, a slika 15 prikazuje koja se poruka ispiše ukoliko se pokuša dohvatiti izbrisani

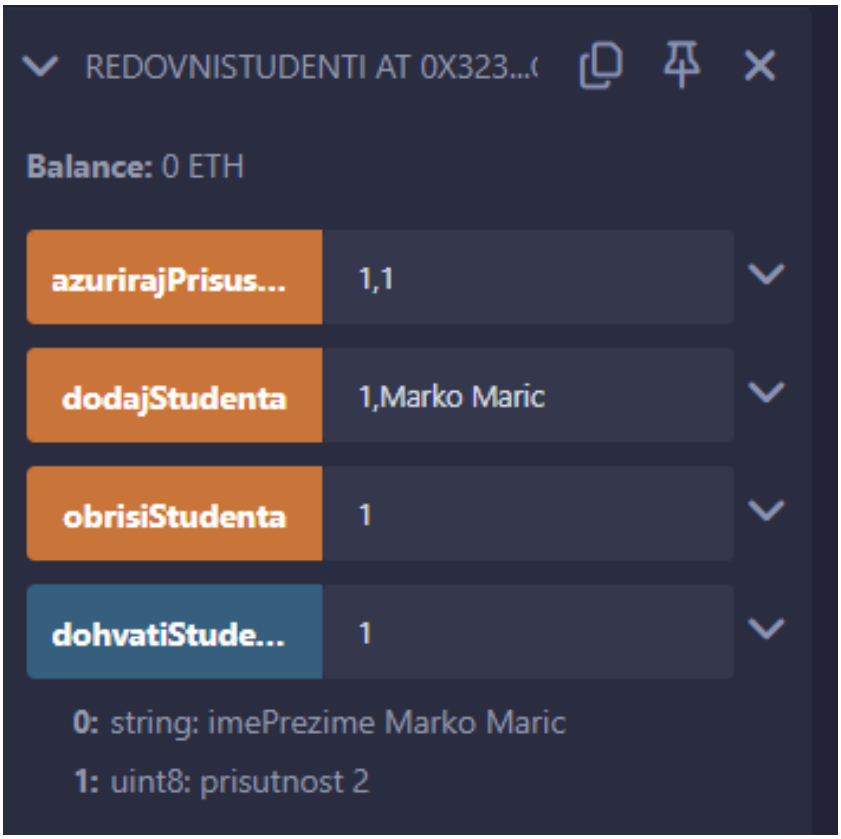
student. Na slikama je prikazano dodavanje, ažuriranje, brisanje i dohvaćanje redovnih studenata, isti je postupak i za redovne tako da nema potrebe za prikazivanjem istih.



Slika 12: Dodavanje i dohvaćanje studenta Primjer 4 (samostalna izrada)



Slika 13: Ažuriranje studenta Primjer 4 (samostalna izrada)



Slika 14: Brisanje studenta Primjer 4 (samostalna izrada)

```
call to RedovniStudenti.dohvatiStudenta errored: Error occurred: revert.

revert
        The transaction has been reverted to the initial state.
Reason provided by the contract: "Student ne postoji".
You may want to cautiously increase the gas limit if the transaction went out of gas.
```

Slika 15: Poruka nakon pokušaja dohvaćanja studenta koji je izbrisan Primjer 4 (samostalna izrada)

6. Zaključak

Rad obrađuje jednu od najbitnijih tehnologija danas, a to je blockchain te sve ono što se usko veže uz blockchain, a to su pametni ugovori te jezik za pisanje tih ugovora. Glavni dio rada odnosi se na sučelja u Solidityju koja su vrlo važna za interoperabilnost te komunikaciju. Sučelja omogućuju standardizaciju između ugovora i kao takva su vrlo važna za smanjenje vremena utrošenog na deklariranje funkcija koje se koriste više puta. Iako su vrlo slični, apstraktni ugovor i sučelja nisu isti pa se ponekad sučelja nazivaju apstraktnim ugovorima. Također, sučelja i nasljeđivanje nisu isti pojmovi budući da nasljeđivanje omogućuje produljivanje ugovora. Sa svojim karakteristikama sučelja se kao takva mogu pronaći samo u Solidityju.

Rad kroz četiri primjera nastoji pokriti sve mogućnosti sučelja te tako prikazati svrhu sučelja i njihovu primjenu kod pisanja pametnih ugovora. Kao što se moglo i zaključiti sučelja su neizostavan dio svakog ugovora te bez njih bi pisanje ugovora bilo kompliciranije. Sučelja olakšavaju interakciju između ugovora.

Popis literature

IBM (bez dat). *What is blockchain?,* Preuzeto 2.6.2024. s https://www.ibm.com/topics/blockchain

GeeksforGeeks (24.2.2023). Intoduction to Solidity, Preuzeto 2.6.2024. s https://www.geeksforgeeks.org/introduction-to-solidity/

Simplilearn (26.5.2024). *What is Solidity Progrmming: dana Types, Smart Contracts, and EVM?*, Preuzeto 2.6.2024. s https://www.simplilearn.com/tutorials/blockchain-tutorial/what-is-solidity-programming

Lepcha (21.11.2023). *Ethereum Virtual Machine (EVM),* Preuzeto 2.6.2024. s https://www.techopedia.com/definition/ethereum-virtual-machine-evm

PLURALSIGHT (10.1.2023). Solidity & Smart Contracts: A quick introduction, Preuzeto 2.6.2024. s https://www.pluralsight.com/blog/software-development/what-is-solidity-smart-contracts

IBM (bez dat). What are smart contracts on blockchain? Preuzeto 2.6.2024. s https://www.ibm.com/topics/smart-contracts

BasuMallick, C.,(20.11.2023). *What Are Smart Contracts? Types, Benefits, and Tools,* Preuzeto 2.6.2024. s https://www.spiceworks.com/tech/innovation/articles/what-are-smart-contracts/

Sharedeum (24.5.2023). What is Solidity interface? – Explained, Preuzeto 2.6.2024. s https://shardeum.org/blog/solidity-interfaces/#Solidity_Interface_Examples

Alchemy, (4.10.2022). *What is the Solidity contract interface?* Preuzeto 2.6.2024. s https://www.alchemy.com/overviews/solidity-interface

Kacar, K. (7.1.2023). *Interfaces in Solidity*, Preuzeto 2.6.2024. s https://medium.com/@kaankacar02/interfaces-in-solidity-333236f5dc12

moralis (20.8.2021) *Remix Explained – What is Remix?* Preuzeto 3.6.2024. s https://moralis.io/remix-explained-what-is-remix/

Popis slika