Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/25BE14Markovic

РАЗВОЈ АПЛИКАЦИЈЕ СТУДЕНТСКЕ СЛУЖБЕ ПРИМЕНОМ ДЕЦЕНТРАЛИЗОВАНИХ И ДИСТРИБУИРАНИХ ТЕХНОЛОГИЈА И ПРОТОКОЛА

DEVELOPMENT OF STUDENT SERVICE APPLICATION USING DECENTRALIZED AND DISTRIBUTED TECHNOLOGIES AND PROTOCOLS

Петар Марковић, *Факултет техничких наука, Нови Сад*

Област – РАЧУНАРСТВО И АУТОМАТИКА

Кратак садржај – *У овом раду је имплементирана децентрализована апликација за рад са студентском службом. У оквиру овог рада је описана целокупна архитектура апликације коју чине паметни уговор постављен на Ethereum мрежи, сервер репрезентован као Express апликација и клијентска React апликација. Такође, објашњене су делатности свих горепоменутих компоненти и начин сарадње.*

Кључне речи: *блокчејн, Ethereum, паметни уговор, IPFS*

Abstract – *This paper presents an implementation for a decentralized student services app. The architecture of this application consists of three components, a smart contract that is deployed on an Ethereum testnet, a server layer represented as an Express application and a React client application. Furthermore, this paper provides an in depth perspective on the responsobilities of each said component and how these components interract.*

Keywords: *blockchain, Ethereum, smart contract, IPFS,*

1. УВОД

Централизовани систем се може дефинисати као структура где постоји једна управљачка компонента која има потпуну контролу. Оваква структура је представљала норму због своје лакоће развоја и приступачности одржавања, међутим тешко је занемарити и њене недостатке, конкретно лоше перформансе за удаљене кориснике и склоност грешкама. Како би се ублажиле ове нуспојаве, користе се алтернативе у виду децентрализованих и дистрибуираних система.

За разлику од ауторитативног централизованог система, децентрализовани систем има више власника при чему сваки складишти копију ресурса којима корисник има приступ. Дистрибуиран систем, иако сличан децентрализованом, разликује се по томе што сваки корисник доноси своје одлуке и коначно понашање система представља збир одлука ових корисника.

Овај рад описује једну потпуно децентрализовану апликацију студентске службе.

НАПОМЕНА:
Овај рад проистекао је из мастер рада чији ментор је био др Горан Сладић, ред. проф.

Сама апликација је вишеслојна и сваки слој имплементира адекватне дистрибуиране и децентрализоване технологије које су детаљније описане у наставку рада.

2. ТЕОРИЈСКЕ ОСНОВЕ

2.1. Блокчејн

Блокчејн се неретко дефинише као јавно доступна дистрибуирана главна књига, која је подељена између чворова рачунарске мреже [1]. Његова форма је облика ланац блокова који по структури подсећа на једноструко спрегнуту листу. Као технологија је први пут описана 1991. године. Оригинална идеја је била да временски означи дигиталне документе и да самим тим смањи ризик антидатирања или злонамерног манипулисања документима [2], али је њена прва практична употреба била 2009. у виду платформе познате као Bitcoin [3].

Идеја блокчејна јесте да омогући чување и дистрибуцију информација, али не и њену измену. Блок представља градивну јединицу блокчејна и сваки блок садржи одређени скуп података, сопствени хеш и хеш претходног блока. Неки од најбитнијих података који се налазе у сваком блоку су трансакције, временска одредница блока, величина блока и енкриптовани број (*nonce*).

2.2. Консензус алгоритми

Консензус алгоритам је процедура уз помоћ које сви учесници блокчејн мреже могу да дођу до заједничког закључка о стању самог блокчејна у реалном времену. Овај механизам омогућава мрежи да постигне поузданост тако што формира ниво поверења између различитих чворова док истовремено осигурава безбедност у окружењу. Суштински, консензус алгоритам обезбеђује да сваки нови блок који се дода на блокчејн буде једина права истинита верзија која је одобрена од стране чворова блокчејна.

2.3. *Ethereum*

Ethereum је децентрализована глобална платформа заснована на блокчејн технологији која омогућава свим својим корисницима да креирају безбедну дигиталну технологију. *Ethereum* такође има своју валуту, познату као *Ether* (*ETH*) [4], као и програмски језик *Solidity* [5].

Паметни уговор је обичан програм написан у *Solidity* програмском језику који се покреће на *Ethereum*-овом блокчејну. Он садржи код и податке који су

ускладиштени на посебној адреси на самом блокчејну. Сви учесници могу да обављају интеракције са паметним уговором, али ниједан од њих нема контролу над њим. Ове интеракције се постижу покретањем трансакција које затим извршавају функције дефинисане у паметном уговору. Свака трансакција захтева и одређену количину ”гаса”, што је вид додатне таксе – наплате за извршавање кода у паметном уговору.

Темељ читаве оперативне структуре *Ethereum*-а јесте *Ethereum Virtual Machine (EVM)*. *EVM* је задужен за извршавање и постављање паметних уговора уз помоћ њене мреже дистрибуираних јавних чворова. Како би се постигао консензус, сваки *Ethereum*-ов чвор ради на *EVM*. Виртуелна машина је потпуно изолована и самим тим код унутар *EVM* нема приступ мрежи, фајл системима или другим процесима [6].

2.4. *Interplanetary File System*

Interplanetary File System (*IPFS*) се може дефинисати као протокол дистрибуираног складишта датотека који омогућава рачунарима да складиште и услужују датотеке, али и фолдере, као део једне велике *Peer-to-Peer* (*P2P*) мреже. Свакој датотеци која је отпремљена на *IPFS* окружењу је могуће приступити и преузети је преко било којег уређаја који је повезан на IPFS [7].

Важан концепт *Interplanetary File System*-а јесте такозвани *Content Identifier (CID),* односно идентификатор садржаја. Свакој датотеци која се отпреми на овом протоколу се додели јединствена адреса која је изведена из хеша њеног садржаја. CID који се додели ресурсу је јединствен и непромењив (имутабилан), што значи да ни једно треће лице не може да измени његову првобитну вредност. Сами подаци се заправо складиште и добављају преко његовог хешираног идентификатора садржаја. Ово значи да је *IPFS* мрежа заправо систем базиран на адресирање садржајем што је у потпуној супротности са типичном централизованом мрежом, где корисници зависе од одређеног поузданог ауторитета да рукује подацима употребом неког вида система базираног на локацији као што је *Uniform Resource Locator (URL).*

3. МОДЕЛ СИСТЕМА

У склопу ове децентрализоване апликације постоје три примарна ентитета: *User, Exam* и *ExamResult. User* представља корисника ове апликације. Сваки корисник се јединствено идентификује путем адресе његовог новчаника. Поред адресе, за корисника се још складишти име и презиме, електронска пошта и рола. Рола може бити професор или студент, и сходно овој роли, он има могућност и приступ различитим функционалностима, конкретно професор може да оцени пријављени испит који он држи, док студент има право да пријави надолазећи испит. Заједничка функционалност за обе улоге јесте то што имају приступ листи испита и пријава, односно професор може да види све своје оцењене и неоцењене испите, док студент има преглед свих својих оцењених испита и испита које може да пријави.

Следећи ентитет јесте *Exam,* односно испит. У склопу овог ентитета чува се јединствени идентификатор испита, назив предмета повезаног са тим испитом, адреса *Ethereum* дигиталног новчаника професора који је надлежан за тај испит и датум и време одржавања испита.

Последњи ентитет јесте *ExamResult*, тачније резултат или пријава испита. Од атрибута он пре свега садржи јединствени идентификатор пријаве, адресу *Ethereum* дигиталног новчаника студента који је поднео пријаву, јединствени идентификатор асоцираног испита за који је направљена пријава и додељену оцену. Када се првобитно пријави испит, његова оцена је 0, док након оцењивања испита ова вредност може бити између 5 и 10.

4. ИМПЛЕМЕНТАЦИЈА СИСТЕМА

5.1. Иницијална поставка

Након поставке новчаника, наредни корак је креирање налога на *Alchemy-y [8]*. *Alchemy* представља блокчејн платформу за програмере која пружа низ алата за развој. Ова платформа замењује стандардне локалне чворове са чворовима који имају бржу и скалабилнију децентрализовану архитектуру. За потребе овог пројекта, креиран је *Alchemy* пројекат за *Ethereum Sepolia* тест мрежу. Након креирања апликације, неопходно је извући *Application Programming Interface* (*API*) кључ како би се успешно аутентификовао корисников чвор и омогућило слање захтева ка блокчејну.

Последњи важни предуслов пре почетка развоја апликације јесте упознавање са *Etherscan* платформом [9]. *Etherscan* се дефинише као блокчејн претраживач за *Ethereum*. Преко ове платформе могуће је пронаћи важне податке сачуване на ланцу. Поред тога, *Etherscan* се може користити и за верификацију паметних уговора. Ова верификација је нужна, јер преко ње се обезбеђује поверење и континуитет у извршењу пословних процеса паметног уговора. Како би се програмеру омогућило да верификује свој паметни уговор приликом поставке на блокчејн, неопходно је направити налог на *Etherscan* платформи и затим креирати свој *API* кључ.

5.2. Серверски део апликације

Solidity као програмски језик је Тјуринг комплетан. Самим тим, фактички је могуће да се све информације и операције које би биле од значаја за једну студентску службу ускладиште на паметном уговору, и да се затим њима управља по потреби. Међутим, треба се узети у обзир чињеница да *Ethereum* имплементира *Proof of Work* консензус механизам и да свака трансакција захтева одређену количину таксе која мора да се исплати рударима како би она била верификова. Ова количина таксе расте сразмерно са тежином рачунарске операције коју та трансакција захтева, али и са величином и комплексношћу података који се складиште на самом паметном уговору. Осим тога, *Solidity* ради на нижем ниво апстракције у односу на друге често коришћене програмске језике што негативно утиче на читљивост самог кода и лакоћу писања истог.

Због свега наведеног, *Solidity* паметни уговор је складиштио минималну потребну количину података. Овај уговор је попуњен подацима приликом његове поставке на *Ethereum* блокчејн мрежу и имплементирао је функционалности регистровања и оцењивања

испита на што је могуће рачунски незахтеван начин. Сам паметни уговор служи и као сервис за плаћање, пошто је приликом пријаве испита неопходно приложити одређену количину средстава која се након успешне трансакције чувају у уговору.

Наравно, било је неопходно имплементирати сервис који би имао детаљне податке о свим ентитетима једног студентског система. Приликом избора и развоја сервера, од круцијалног значаја је било да се обезбеди максимално децентрализовано окружење. Из овог разлога је одлучено да се са подацима рукује у склопу *OrbitDB* базе података. *OrbitDB* представља дистрибуирано *P2P* складиште података реализовано на *IPFS*-u које је дизајнирано за рад са децентрализованим апликацијама. Како *OrbitDB* пружа *JavaScript API*, одлучено је да сервис буде имплементиран у том језику.

5.2.1. Паметни уговор

За потребе развоја блокчејн стране рада, коришћен је радни оквир *Foundry* [10]. *Foundry* је окружење за развој децентрализованих апликација засновано на *Ethereum* блокчејну. Јединствена карактеристика овог радног оквира јесте његов скуп уграђених алата командне линије.

У склопу ове платформе развијен је *Solidity* паметни уговор назван *StudentService*. На самом почетку паметног уговора дефинишемо скуп варијабли стања. Свака од дефинисаних променљивих има приватни модификатор приступа и самим тим видљива је само у оквиру уговора. Прва варијабла EXAM_FEE је константа типа неозначени број од 256 бита и она представља цену пријаве једног испита. Наредна варијабла *exams* представља низ идентификатора свих испита. Последње три променљиве су типа *mapping*. По структури *mapping* је налик хеш табели и користи се за складиштење парова кључ-вредност. Прва од ове три променљиве названа *users* представља скуп свих корисника у систему. Кључ је адреса новчаника корисника, а вредност је типа *boolean* и верификује да ли корисник са том адресом постоји у студентској служби. Последње две варијабле *studentExamResults* и *professorExamResults* складиште информације о резултатима испита. Тачније, прва варијабла као кључ чува адресу новчаника студента, а као вредност чува низ идентификатора свих испита које је пријавио. Кључ друге варијабле је адреса професора, а вредност представља низ идентификатора пријављених испита за које је одговоран.

У наставку програма налазе се компоненте за обраду догађаја (енгл. *event*). Уз помоћ *event* компоненти, могуће је успоставити комуникацију са клијентском апликацијом и обавестити је да је дошло до промене на блокчејну након одређене трансакције. Најчешће им се придодају параметри који представљају важне информације о трансакцији. У склопу ове апликације постоје 2 догађаја: *examRegistered* и *examGraded*.

Solidity садржи конструкторе. У оквиру ове посебне функције може се писати код и ова функција ће се извршити само једном након креирања уговора. Конструктор се користио како би се поставиле иницијалне вредности у оквиру горе наведених варијабли стања.

Последњи део паметног уговора састоји се из функција. Функције су подељене у три групе: *view* функције коришћене за приступ вредностима варијабле стања, *pure* функције које представљају помоћне функције валидације и главне функције у склопу којих су имплементиране функционалности.

У паметном уговору постоје две главне функције: *registerExam* и *gradeExam*. Прослеђени параметри ових функција се прво валидирају употребом горепоменутих *view* и *pure* функција. Уколико су параметри исправни, функција наставља са извршавањем и на крају се окида адекватан догађај.

У *Foundry* окружењу је могуће написати и тестове. Ови тестови спадају у један вид паметног уговора где је у самој декларацији уговора неопходно додати *is Test* како би скуп алата знао да ову датотеку третира као тестну. Узимајући у обзир релативну једноставност самог паметног уговора, за потребе овог пројекта написани су базични тестови који тестирају *view* и *pure* функције.

Након финализације процеса развоја и тестирања, може се прећи на поставку и верификацију уговора. Ово се обавља путем уноса инструкција у командну линију коју *Foundry* и конкретно њен пакет *forge* подржавају. Како би се поставио уговор на мрежу, неопходно је унети команду *forge create*. Поред ове команде неопходно је и проследити низ параметара, конкретно приватни кључ новчаника који ће извршити поставку уговора, *HTTPS* крајњу тачку са *API* кључем дефинисаним у *Alchemy* платформи, параметре конструктора и *API* кључ *Etherscan* налога.

5.2.2. Серверски слој

За развој серверској слоја коришћен је *Express.js*, популарни радни оквир за развој веб апликација у *JavaScript* језику. Овај радни оквир је првенствено изабран пошто за *IPFS* и децентрализовану *OrbitDB* базу података постоји обимно документована *JavaScript* имплементација

Приликом покретања апликације се подиже и *OrbitDB* база података. Да би се база података конфигурисала, прво је неопходно направити инстанцу *IPFS*-а коју ће *Orbit* користити као комуникациони слој. Приликом дефинисања *IPFS* чвора, неопходно је навести и конкретну локацију, односно репозиторијум, где ће подаци овог чвора бити ускладиштени.

У склопу овог пројекта, компоненте су модуларно подељене у зависности од њихове одговорности. *model* директоријум садржи *JavaScript* класе које репрезентују шеме ентитета који се складиште у табелама базе података. Укупно постоје три модела и то: *Exam*, односно испит, *ExamResult*, односно резултат или пријава испита и *User*, односно корисник. За сваки од ових модела постоји одвојена база података.

За приступ и манипулацију подацима користе се датотеке у *dao* директоријуму. Сам *dao* је акроним за *Data Access Object*, слој веб апликације задужен за рад са базом података. Како бисмо могли да извршавамо операције над *OrbitDB*, потребно је сваки пут програмски отворити базу података и дефинисати табелу којој желимо да приступимо. Када се изврше неопходне операције над базом података, добра

пракса је да се она одмах затвори како би се ослободили ресурси.
Следећа компонента архитектуре која ће бити описана јесте сервисни слој. Сврха овог слоја јесте да енкапсулира пословну логику саме апликације и да употребом функционалности из претходно описаног *dao* директоријума руководи ентитетима који су сачувани у децентрализованој бази података.
Последња значајна компонента серверске архитектуре јесте *controllers*. Овај слој садржи низ контролера чији примарни циљ јесте да буде медиатор између крајњег корисника и серверске апликације и омогући корисницима да приступе и управљају функционалностима које су раније описане.

5.2. Кориснички (*frontend)* део апликације

За развој корисничког дела апликације, коришћена је једна од најпознатијих *JavaScript* библиотека: *React*. У комбинацији са *React*-ом, употребљена је и једна популарна библиотека за развој и дизајнирање корисничког интерфејса звана *Chakra UI*.
Нажалост, *React* као библиотека нема уграђену подршку за рад са блокчејном и паметним уговорима, због чега је било неопходно инсталирати зависности *react-moralis* и *web3uikit*. *react-moralis* пружа функционалности уз помоћу којих могу да се позивају функције постављеног паметног уговора и да се добави стање повезаног новчаника са страницом, док *web3uikit* садржи стилизиране и лагане компоненте корисничког интерфејса за развој *Web 3.0* апликација.
Како би се позвале функције постављеног паметног уговора из прошлог одељка, користи се *useWeb3Contract* функционалност коју пружа *react-moralis* библиотека. За позивање ове функционалности, неопходно је дефинисати жељени назив функције у оквиру компоненте и проследити *ABI* паметног уговора, адресу постављеног паметног уговора, назив функције који се позива из паметног уговора, списак параметара паметног уговора и потенцијално количину *ETH* која се шаље паметном уговору.
Уколико позивом ове функције добијемо позитиван одговор од стране паметног уговора, наредни корак јесте да пошаљемо адекватан захтев *Express* серверу како бисмо управљали са подацима у њиховој целости. Ради олакшане комуникације са самим сервером, коришћен је популарни *JavaScript HTTP* клијент *axios*.

6. ЗАКЉУЧАК

У овом раду имплементирана је децентрализована апликација студентске службе. Како би се олакшао развој, тестирање, компајлирање и постављање паметног уговора који представља основу апликације, коришћена је *Foundry* развојна платформа. Узимајући у обзир колико може бити скупо складиштење велике количине података и вршење комплексних рачунских операција у *Solidity* уговору, конкретни паметни уговор је складиштио само идентификаторе ентитета од значаја и имплементирао плаћање криптовалутама, а детаљнији садржај и пословна логика је била имплементирана у одвојеном *Express* серверу. Овај сервер је складиштио и руковао подацима на децентрализован начин употребом *OrbitDB*, база података изграђена на *IPFS* протоколу. Како би се постигла комуникација са овим паметним уговором који је постављен на *Ethereum* блокчејну, на клијентској страни је било неопходно подесити *Moralis* платформу.
Правци даљег развоја апликације би били фокусирани на креирање компатабилне мобилне апликације. Поред тога, потенцијално би се могло додати још функционалности или проширити постојеће (отказивање испита, овера семестра, одбрана дипломског рада итд.). Такође, са становишта оптимизације, сам паметни уговор би се могао рефакторисати тако да извршавање операција над њим захтева још мање гаса.
На самом серверу се могу постићи унапређења, пре свега имплементацијом неког вида контроле приступа која је базирана на ролама.
Поред тога, овај сервер је написан у обичном *JavaScript* језику и самим тим је динамички типизиран језик, што може да има негативан утицај на перформансе и безбедност. Решење би било надоградити постојећи пројекат употребом *TypeScript*-а, или мигрирати базу кода сервера на статички типизиран програмски језик.

7. ЛИТЕРАТУРА

[1] https://www.investopedia.com/terms/b/blockchain.asp (приступљено у септембру 2023.)
[2] Haber, Stuart, and W. Scott Stornetta. How to time-stamp a digital document. Springer Berlin Heidelberg, 1991.
[3] Nakamoto, Satoshi. "Bitcoin: A peer-to-peer electronic cash system." Decentralized business review (2008).
[4] https://ethereum.org/en/eth/ (приступљено у септембру 2023.)
[5] https://soliditylang.org/ (приступљено у септембру 2023.)
[6] https://ethereum.org/en/developers/docs/evm/ (приступљено у септембру 2023.)
[7] https://docs.ipfs.tech/concepts/what-is-ipfs/ (приступљено у септембру 2023.)
[8] https://www.alchemy.com/ (приступљено у септембру 2023.)
[9] https://etherscan.io/ (приступљено у септембру 2023.)
[10] https://book.getfoundry.sh/ (приступљено у септембру 2023.)

Кратка биографија:

Петар Марковић рођен је у Нишу 1999. године. Мастер рад на Факултету техничких наука из области Софтверско инжењерство и информационе технологије – Електронско пословање одбранио је 2023. године

контакт: petarns99@yahoo.com