



УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У
НОВОМ САДУ



Никола Маленчић

**Примена Ethereum блокчејн платформе
за развој децентрализоване апликације
за гласање**

Мастер рад

Нови Сад, 2020.



УНИВЕРЗИТЕТ У НОВОМ САДУ • ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
21000 НОВИ САД, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА


Редни број, РБР:	
Идентификациони број, ИБР:	
Тип документације, ТД:	Монографска документација
Тип записа, ТЗ:	Текстуални штампани материјал
Врста рада, ВР:	Мастер рад
Аутор, АУ:	Никола Маленчић
Ментор, МН:	др Душан Гајић, доцент
Наслов рада, НР:	Примена Ethereum блокчејн платформе за развој децентрализоване апликације за гласање
Језик публикације, ЈП:	Српски / ћирилица
Језик извода, ЈИ:	Српски/енглески
Земља публикаовања, ЗП:	Република Србија
Уже географско подручје, УГП:	Војводина
Година, ГО:	2020.
Издавач, ИЗ:	Ауторски репринт
Место и адреса, МА:	Нови Сад, трг Доситеја Обрадовића 6
Физички опис рада, ФО: (поглавља/страна/ цитата/табела/слика/графика/прилога)	
Научна област, НО:	Електротехника и рачунарство
Научна дисциплина, НД:	Примењене рачунарске науке и информатика
Предметна одредница / кључне речи, ПО:	Базе података, дистрибуирани информациони системи, <i>блокчејн</i>
УДК	
Чува се, ЧУ:	У библиотеци Факултета техничких наука, Нови Сад
Важна напомена, ВН:	
Извод, ИЗ:	У овом раду је описано софтверско решење које представља децентрализовану апликацију за гласање. Као платформа за подршку овој апликацији је изабрана <i>Ethereum блокчејн</i> платформа. Представљено решење користи блокчејн како би понудило нове могућности за гласачке системе. У раду су представљене и основе дистрибуираних система и блокчејн технологије.
Датум прихватања теме, ДП:	
Датум одбране, ДО:	
Чланови комисије, КО:	Председник: др Јованка Пантовић, редовни професор
	Члан: др Драган Дину, доцент
	Члан, ментор: др Душан Гајић, доцент
	Потпис ментора



UNIVERSITY OF NOVI SAD • FACULTY OF TECHNICAL SCIENCES
21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Accession number, ANO :			
Identification number, INO :			
Document type, DT :	Monographic publication		
Type of record, TR :	Textual printed material		
Contents code, CC :	Master Thesis		
Author, AU :	Nikola Malenčić		
Mentor, MN :	Dušan Gajić, Ph. D		
Title, TI :	Application of the Ethereum blockchain platform for the development of a decentralized voting system		
Language of text, LT :	Serbian		
Language of abstract, LA :	Serbian/English		
Country of publication, CP :	Republic of Serbia		
Locality of publication, LP :	Vojvodina		
Publication year, PY :	2020.		
Publisher, PB :	Author's reprint		
Publication place, PP :	Novi Sad, Dositeja Obradovića sq. 6		
Physical description, PD : (chapters/pages/ref./tables/pictures/graphs/appendixes)	5/64/0/2/36/0/1		
Scientific field, SF :	Electrical and computer engineering		
Scientific discipline, SD :	Applied computer science and informatics		
Subject/Key words, S/KW :	Databases, distributed information systems, blockchain		
UC			
Holding data, HD :	The Library of Faculty of Technical Sciences, Novi Sad		
Note, N :			
Abstract, AB :	This thesis presents a software solution which implements a decentralized voting application. The Ethereum blockchain has been chosen as the platform on which application runs due to its popularity and ease of development. The solution leverages the advantages of the blockchain technology to offer new value in voting systems. The fundamentals of distributed systems and blockchain technology are also discussed.		
Accepted by the Scientific Board on, ASB :			
Defended on, DE :			
Defended Board, DB :	President:	Dr. Jovanka Pantović, Full Professor	
	Member:	Dr. Dragan Dinu, Assistant Professor	Menthor's sign
	Member, Mentor:	Dr. Dušan Gajić, Assistant Professor	

	УНИВЕРЗИТЕТ У НОВОМ САДУ • ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА 21000 НОВИ САД, Трг Доситеја Обрадовића 6	Датум:
	ЗАДАТАК ЗА МАСТЕР РАДА	Лист/Листова:
		4/64

(Податке уноси предметни наставник - ментор)

Студијски програм:	Рачунарство и аутоматика
Руководилац студијског програма:	др Милан Видаковић, ред. проф.

Студент:	Никола Маленчић	Број индекса:	E2 54/2019
Област:	Електротехничко и рачунарско инжењерство		
Ментор:	др Душан Гајић, доцент		

НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА МАСТЕР РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА:

- проблем – тема рада;
- начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна;
- литература

НАСЛОВ МАСТЕР РАДА:

<p align="center">Примена Ethereum блокчејн платформе за развој децентрализоване апликације за гласање</p>

ТЕКСТ ЗАДАТКА:

<p>➤ Проучити основне концепте и механизме <i>блокчејн</i> технологије</p> <p>➤ Упознати се са <i>Ethereum</i> платформом која представља конкретну имплементацију <i>блокчејн</i> технологије</p> <p>➤ Упознати се са алатима неопходним за развијање апликација на Ethereum платформи</p> <p>➤ Специфицирати модел <i>блокчејн</i> мреже која подржава рад система за гласање</p> <p>➤ На основу специфицираног модела имплементирати одговарајуће софтверско решење</p>	
Руководилац студијског програма:	Ментор рада:
др Милан Видаковић, ред. проф.	др Душан Гајић, доцент

Примерак за: <input type="checkbox"/> - Студента; <input type="checkbox"/> - Ментора

Садржај

1. УВОД.....	1
2. Основе DLT-а и блокчејна.....	2
2.1. Централизовани и децентрализовани системи	2
2.2. Дистрибуирани системи	3
2.2.1 Колекција аутономних рачунских елемената	3
2.2.2 Један кохерентни систем.....	5
2.2.3. Middleware	5
2.3. Дистрибуирана главна књига	6
2.4. Блокчејн.....	7
2.4.1. Криптографија	9
2.4.2. Консензус механизам	10
3. Ethereum и Dapp-ови.....	12
3.1. Паметни уговори.....	12
3.2. Гас.....	13
3.3. Децентрализоване апликације (Dapp-ови)	13
3.4. Опис коришћених алата и технологија	14
3.4.1. Ganache	14
3.4.2. Truffle	16
3.4.3. Web3.js	17
3.4.4. MetaMask	18
4. ОПИС СОФТВЕРСКОГ РЕШЕЊА.....	20
4.1. Решење и начин функционисања	20
4.1.1. Блокчејн	20
4.1.2. Бекенд.....	20
4.1.3. Фронтенд.....	28
5. ЗАКЉУЧАК	38
ЛИТЕРАТУРА	39
ДОДАТАК А	40
СПИСАК КОРИШЋЕНИХ СКРАЋЕНИЦА	40
СПИСАК СЛИКА	41
БИОГРАФИЈА	43

1. УВОД

У овом раду су представљене могућности примене блокчејн технологије за подршку системима за гласање. Описан је развој једног таквог система у виду апликације на Ethereum блокчејн платформи. Апликација је имплементирана на дистрибуиран и децентрализован начин, како би се испитале предности и мане софтверског решења овог типа.

У другом поглављу су описане теоријске основе неопходне за схватање једног оваквог система. Ради се о такозваним технологијама дистрибуиране главне књиге (енг. Distributed Ledger Technology) и блокчејну (енг. blockchain). Апликација је у потпуности базирана на овим технологијама, тако да су оне од круцијалног значаја са развој оваквих система.

У трећем поглављу су описане конкретне технологије искоришћене за имплементацију децентрализоване апликације. Ради се о Ethereum блокчејн платформи, као и о централној бази њене инфраструктуре – Dapp-овима (енг. Decentralized application). Поред Ethereum-а, као саме инфраструктуре за систем, коришћене су још неке помоћне технологије. То су Ganache – алат за симулацију дистрибуиране главне књиге, Truffle – скуп библиотека за рад са паметним уговорима, Web3.js – скуп JavaScript библиотека за рад са Ethereum-ом и MetaMask – дигитални новчаник за рад са Ethereum-ом.

Четврто поглавље се бави самим описом софтверског решења. У овом поглављу су описане све функционалности апликације, као и начин постизања датих функционалности.

На крају самог рада се налази закључак, у ком је дат релевантни сажетак рада, као и неке мисли како рад може да се даље унапреди у будућности.

2. Основе DLT-а и блокчејна

2.1. Централизовани и децентрализовани системи

Осврнућемо се на неке типичне карактеристике централизованих и децентрализованих система.

Централизовани системи су типично једноставнији за имплементацију него децентрализовани. Софтвер код децентрализованих система је веома сложен због бројних проблема конзистентности система. Децентрализовани систем мора крајњем кориснику да одаје утисак да интерагује са једним кохерентним системом. Потребни су протоколи који ће сваком члану система (чвору) дати јединствену спрегу ка систему. Такође, у децентрализованом окружењу, више чворова могу истовремено да приступају једном ресурсу, што значи да протоколи морају да обезбеде и конзистентност података.

Децентрализовани системи се добро скалирају. Најразличитији уређаји који имплементирају протоколе система могу лако да се придруже, док код централизованих система то не мора увек бити случај. Уређаји који учествују у систему могу бити расути по целом свету али комуницирају са локалним уређајима како не би било претераног чекања. Такође, код централизованих система повећање броја корисника може довести до тога да централни сервери не могу да опслуже све клијенте, док је код децентрализованих система овај проблем непостојан.

Са стране перформанси појединачних уређаја, код децентрализованих система се може јавити слабије искоришћење самих хардверских јединица. Ово је зато што се типично код оваквих система јавља много комуникације између чворова која није везана за само решавање проблема. Ова комуникација служи како би се постигла конзистентност и кохерентност система.

Системи са централизованом архитектуром су подложни отказима. Код оваквих архитектура је довољан мали квар да парализује рад система. У дистрибуираном окружењу систем наставља да ради и ако неки чворови откажу.

Безбедност система је занимљива тема јер не постоји очигледна предност централизованог или децентрализованог начина рада. С једне стране, централизоване системе је лакше осигурати јер је потребно водити рачуна само о једном критичном делу система, док је код других потребно водити рачуна о понашању свих чворова система. Са најновијим протоколима нека децентрализована решења успевају да нуде велику безбедност, по неким чак и бољу него што нуде централизовани системи.

Једна од најактуелнијих тема у модерном добу је приватност. Популарност децентрализованих решења у великој мери проистиче као реакција на монопол и неетичке поступке великих технолошких компанија у 21. веку. Људи губе поверење у ове компаније јер сматрају да им подаци нису адекватно заштићени и да се ти подаци користе против њихових интереса. Због тога на светску сцену наступају велике платформе које су базиране на дистрибуираним технологијама које гарантују сигурност и приватност свих корисника који делују на њима. Дакле, може се рећи да

децентрализовани системи нуде већу заштиту личних података него централизовани системи.

2.2. Дистрибуирани системи

Постоје многе дефиниције дистрибуираних система. Већина тих дефиниција су међусобно контрадикторне, тако да ћемо користити један генерални опис како бисмо дефинисали ове системе.

Дистрибуирани систем је колекција аутономних рачунских елемената која се својим корисницима приказује као један кохерентни систем.

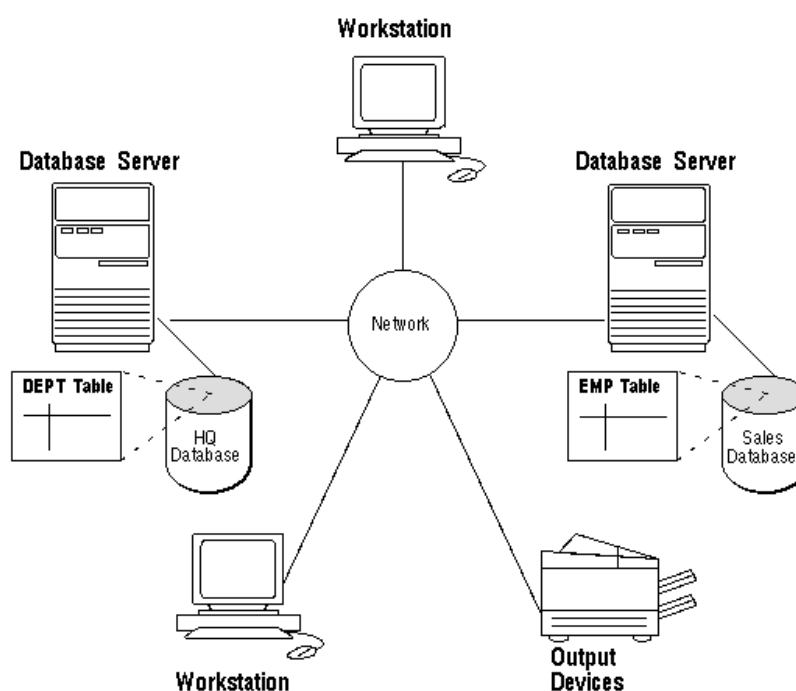
Ова дефиниција се односи на две карактеристике сваког дистрибуираног система.

2.2.1 Колекција аутономних рачунских елемената

Прва је то што је дистрибуирани систем скуп рачунских елемената који су способни да се понашају независно у односу једни на друге. Сваки појединачни рачунски елемент се назива чвор. Чворови могу бити или хардверске јединице или софтверски процеси. Модерни дистрибуирани системи се често састоје од најразличитијих врста чворова, од рачунара високих перформанси до најситнијих сензор уређаја. Основни принцип је да су чворови у могућности да делују независно, али уколико се они игноришу међусобно онда нема смисла да се постављају у исти дистрибуирани систем. У пракси су чворови испрограмирани да сарађују једни са другима што се реализује размењивањем поруке или са неком врстом дељене меморије.

Важно је приметити да се као последица међусобног деловања разних независних чворова јавља одсуство заједничке временске референце. Наиме, не може се претпоставити да постоји нешто као што је глобални сат. Овај недостатак заједничког времена доводи до великих проблема што се тиче синхронизације и координације између чворова.

Такође се јавља проблем припадања одређеној групи. Чињеница да се ради о колекцији чворова значи да мора постојати механизам који води рачуна о припадању тој колекцији. Могу постојати отворене и затворене групе. Отворена група је онај дистрибуирани систем коме се слободно придружује било који чвор и након придруживања може да шаље поруке другим чворовима. Затворен систем је онај где само одређени чворови могу да комуницирају међусобно. Ово такође подразумева постојање неког механизма који води рачуна о прихватању и избацивању чворова у и из групе.



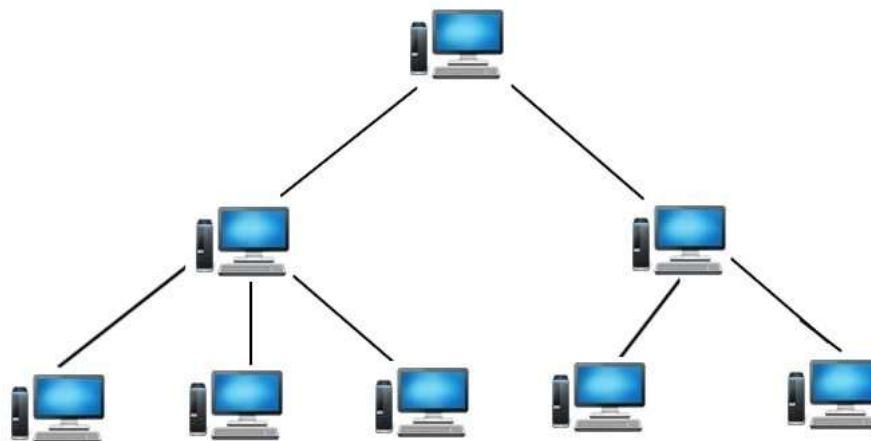
Слика 1 Дистрибуирани систем (преузето са docs.oracle.com)

Што се тиче организације колекције, у пракси се показало да је већина дистрибуираних система организовано као прекривна мрежа. У овом случају, један чвор је типично софтверски процес који поседује листу других чворова коме може да шаље одређене поруке. Постоје случајеви где чворови не садрже листе већ морају прво да се информишу о својим суседима. Прекривне мреже се углавном деле у два типа:

Структурирана: Сваки чвор има јасно дефинисан скуп суседа са којима може да комуницира. Пример: чворови су организовани у стабло.

Неструктурирана: Сваки чвор има неки број референци ка насумично одабраним чворовима.

Принципијално би свака прекривна мрежа требала бити потпуно повезана. Ово значи да између свака два чвора постоји комуникациони пут који омогућава овим чворовима да рутирају поруке један до другог.



Слика 2 Структурирана прекривна мрежа (преузето са jharaphula.com)

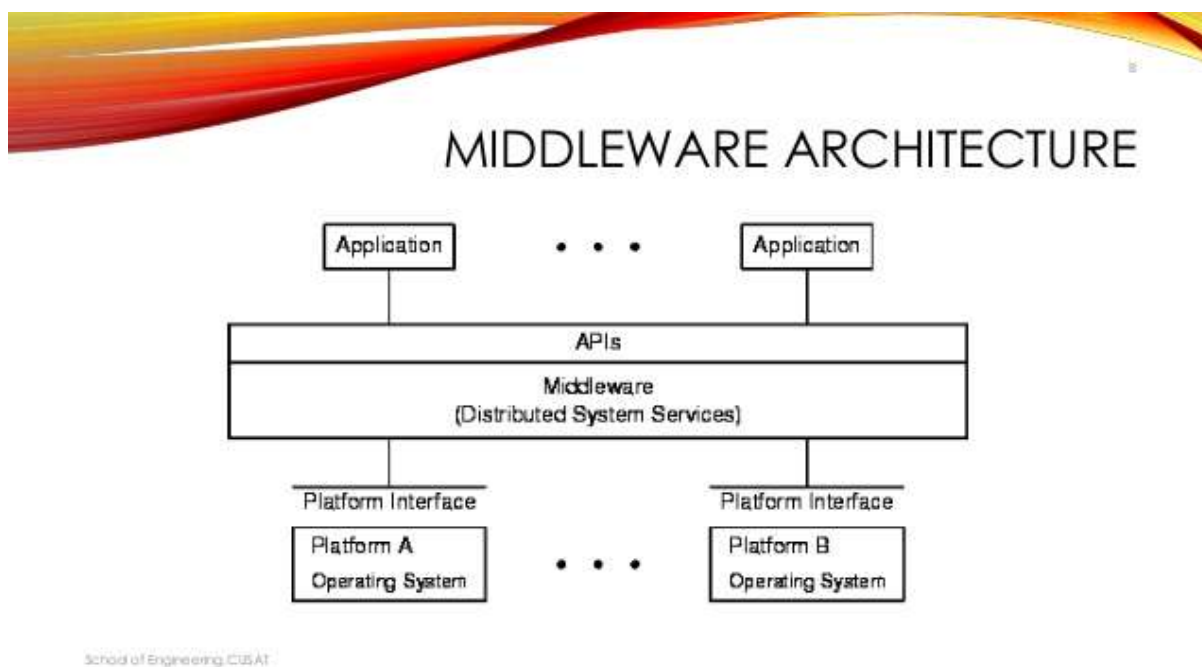
2.2.2 Један кохерентни систем

Дистрибуирани систем би требао да се приказује корисницима као јединствен и кохерентан систем. Неки кажу да крајњи корисници уопште не би требало да примећују да раде са процесима и подацима који су распрострањени по читавој дистрибуираној мрежи. Ово се углавном сматра претеривањем, и неки стабилни концензус је на томе да би систем требао да се понаша онако како корисници очекују да се понаша. Колекција чворова би требала да ради на исти начин каква год била интеракција између корисника и система.

Ово значи да крајњи корисник не би могао да зна на ком рачунару се неки процес тренутно обавља као ни информацију где се тренутно налазе подаци које користи и да ли су ти подаци репликовани. Ово се зове транспарентност дистрибуције. Овај приступ је сличан као онај у Unix оперативним системима, где се ресурсима приступа преко јединственог система датотека у ком су скривене разлике између датотека, уређаја, меморија и мрежа. Овакав приступ је изузетно комплексан за имплементацију. Чињеница да је систем у употреби дистрибуиран је нешто што се тешко сакрива. Дакле, непредвиђено понашање је нешто што се често јавља у дистрибуираним система и потребни су нам механизми који ће се суочавати са овим проблемом.

2.2.3. Middleware

Дистрибуирани системи су углавном организовани тако да имају посебан софтверски слој који је логички постављен изнад оперативног система појединачног рачунара који је део система. Овај слој се популарно назива *middleware*.



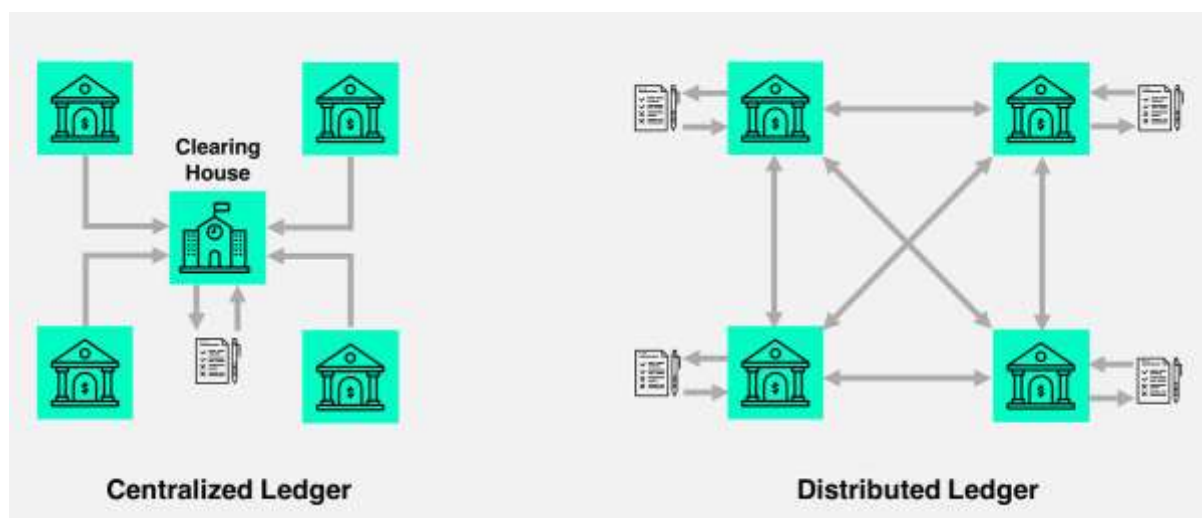
Слика 3 Middleware (преузето са www.slideshare.net/Rishikese)

На слици 2-3 можемо видети два рачунара који су део једног дистрибуираног система. На сваком рачунару се налазе две апликације и заједнички средњи слој. Рачунари такође имају другачије оперативне системе и хардверске структуре. Middleware омогућава да исте апликације међусобно комуницирају преко заједничке спреге, али такође омогућава да и различите апликације комуницирају на исти начин.

На неки начин, middleware је дистрибуираном систему исто што и оперативни систем једном рачунару. То је софтверски слој који делује као управљач ресурсима који нуди своје апликације како би се ти ресурси ефикасно поделили и искористили у мрежи. Поред управљања ресурсима, може да нуди и друге сервисе који су типични за оперативне системе. Ово може укључивати сервисе за безбедност, сервисе за контролу налога, сервисе за опоравак од отказа, сервисе за међуапликацијску комуникацију итд. Разлика између ових сервиса и сервиса оперативног система је то што се ови сервиси нуде у мрежном окружењу.

2.3. Дистрибуирана главна књига

Дистрибуирана база података је врста базе података код које се подаци чувају на више чворова, тј. дистрибуирани су на више локација. Главна књига (енг. Ledger) је књига која служи за бележење и сумирање економских трансакција, са дебитима и кредитима у посебним колонама, такође са почетним и крајњим стањем два рачуна. Дакле, дистрибуирана главна књига, или технологија дистрибуиране главне књиге (енг. Distributed Ledger Technology – DLT) је врста дистрибуиране базе података у којој се претпоставља присуство малициозних корисника тј. чворова.



Слика 4 Централизована и дистрибуирана главна књига (преузето са tradeix.com)

Постоје три главне карактеристике технологија дистрибуираних главних књига:

Језик трансакција: Мора да постоји одређени формат по коме ће чворови да траже промену стања главне књиге, тј. формалан начин на који ће иницирати трансакције

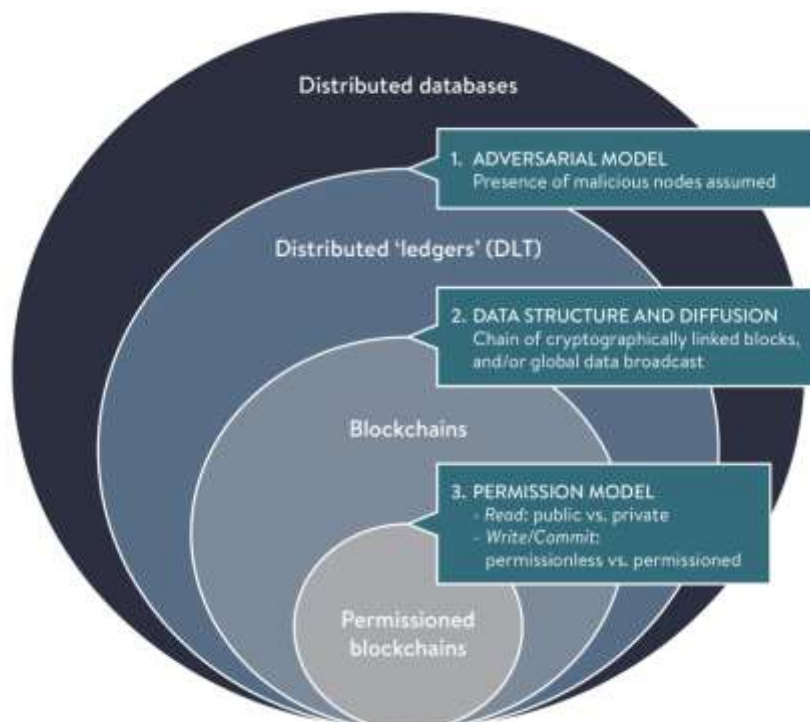
Протокол: Морају постојати правила по којима ће се међу учесницима у дистрибуираном систему постићи консензус о томе које ће се трансакције прихватити и у ком редоследу ће се уписати у главну књигу

Модел података: Тренутно стање главне књиге мора бити дефинисано одређеним моделом података који је одабран у време техничке изведбе система

Концепт технологија дистрибуиране главне књиге постоји већ дуго. Још 1982. године су Лампорт, Шостак и Пис описали "проблем Византијских генерала" и поставили питање како ће се рачунарски систем носити са супротстављеним информацијама у непријатељском окружењу. 1999. године истраживања доводе до првог алгоритма који на практичан начин решава претходно поменути проблем. Ова технологија је основа за нову генерацију апликација на бази трансакција које успостављају транспарентност, одговорност и поверење у рачунарским системима, а при том рационализују пословне процесе и правна ограничења кроз аутоматизацију.

2.4. Блокчејн

Блокчејн представља једну врсту дистрибуиране базе података у којој се складиште дигиталне трансакције. Ова технологија представља подскуп технологија дистрибуиране главне књиге. Са појавом Bitcoin-а је блокчејн довео до револуције у дистрибуираним системима и начину на коју људи гледају финансијско пословање, ланце испоруке, приватност итд.



Слика 5 Однос технологија (преузето са www.eu.com)

Постоје две основне идеје које разликују блокчејн од осталих технологија дистрибуиране главне књиге.

Прва је идеја да блокчејн буде пројектован тако да би се постигао поуздан и конзистентан договор између независних чворова о запису догађаја. Ово се постиже механизмом консензуса који омогућава да за сваког учесника у мрежи буде исти поглед на дељењу базу података. Дакле, учесници у блокчејн мрежи постижу консензус о променама у дељеној бази података без потребе да се проверава интегритет било ког чвора у мрежи.

Друга идеја је специфична структура података која омогућава овој технологији да реши проблем двоструке потрошње, тј. онемогућава корисницима да исти дигитални фајл копирају и преносе више пута. Блокчејн се из ових разлога може користити за размену валуте, вредности или других података без потребе за централним ауторитетом.

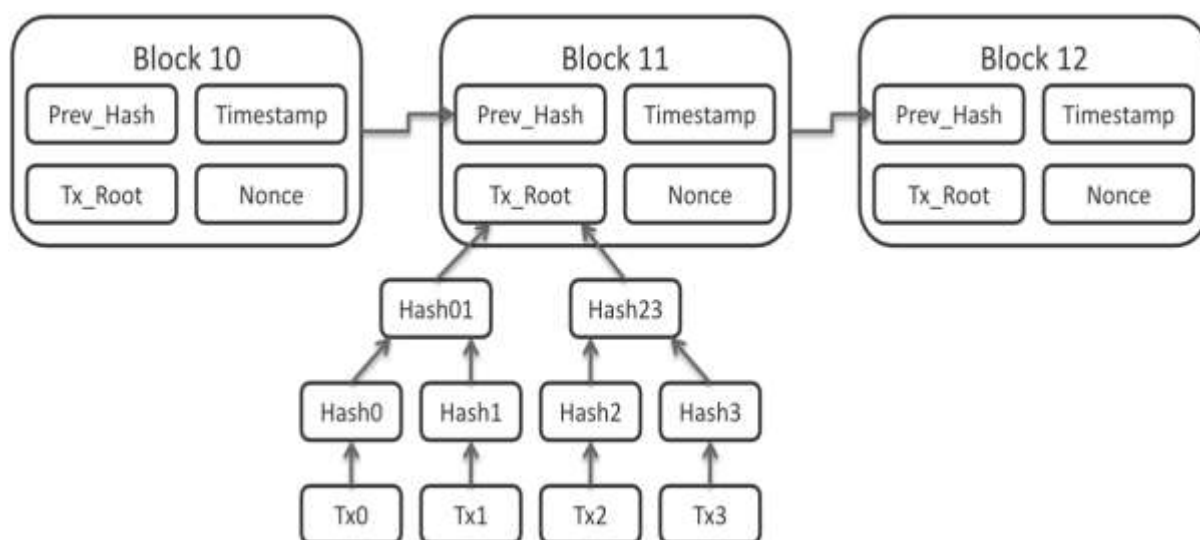
Основна подела блокчејнова би била на јавне и приватне. Јавним системима може да приступи било ко, док приватним могу да приступе само ауторизовани корисници. Такође је битно увидети да могу постојати разне контроле приступа. Тако на неким блокчејн системима може било ко и да чита и да пише на главну књигу (Bitcoin, Ethereum), на некима било ко може да чита, а само ауторизовани учесници могу да пишу (Sovrin), а на некима постоји контрола приступа и за операције читања и писања.

Осврнућемо се на најбитније компоненте блокчејн система.

2.4.1. Криптографија

Блокчејн користи разне криптографске технике за различите сврхе.

Првенствено, главна књига у блокчејн система представља ланац криптографски повезаних блокова. Блок је скуп одређеног броја трансакција које се истовремено додају у ланац. У заглављу сваког блока се налази референца на претходни блок у виду криптографског хеша тј. резултата хеш функције. Хеш функције су једносмерне (рачунање инверзне функције је математички захтевно) математичке функције које на основу неког скупа података дају хеш, тј. вредност фиксне дужине која је добијена на основу тог скупа података. Сврха овога је да се онемогући измена блокова након додавања блокова у ланац. Уколико неки учесник покуша да измени неки блок, остали учесници ће то лако приметити јер се хеш тог измењеног блока неће поклапати са хешом који је садржан у заглављу новог блока.



Слика 6 Структура блокова (преузето са bitcoin.stackexchange.com)

Осим криптографског хеша се у заглављу сваког блока налази и корен Мерклеовог стабла. Мерклеово стабло или хеш стабло је структура података које представља стабло коме је сваки терминални чвор означен хешом блока података, а сваки нетерминални чвор је означен хешом ознака његових потомака. Оваква структура омогућава поуздану и ефикасну претрагу и верификацију великих структура података.

Свако заглавље такође садржи временски отисак и попсе вредност. Временски отисак представља тренутак у времену када је блок додат у блокчејн. Nonce вредност је случајан број који се користи само једанпут, и има улогу у процесу валидације блокова.

У блокчејну се такође користи инфраструктура јавних кључева. На блокчејну постоје јавне адресе којима су придружене одређени ресурси (у примеру Bitcoin-а би

то била нека количина Bitcoin-ова). Идеја је да сваки корисник има свој приватан кључ, на основу којег он може да генерише јавне кључеве или адресе. Уз помоћ овог приватног кључа, корисник може да докаже поседовање неке јавне адресе, тј. адресе која је изведена из тог кључа. Дакле, сви ресурси се налазе у главној књизи, корисници путем својих кључева приступају главној књизи и добијају могућност да манипулишу ресурсима који су придружени њиховим јавним адресама.

2.4.2. Консензус механизам

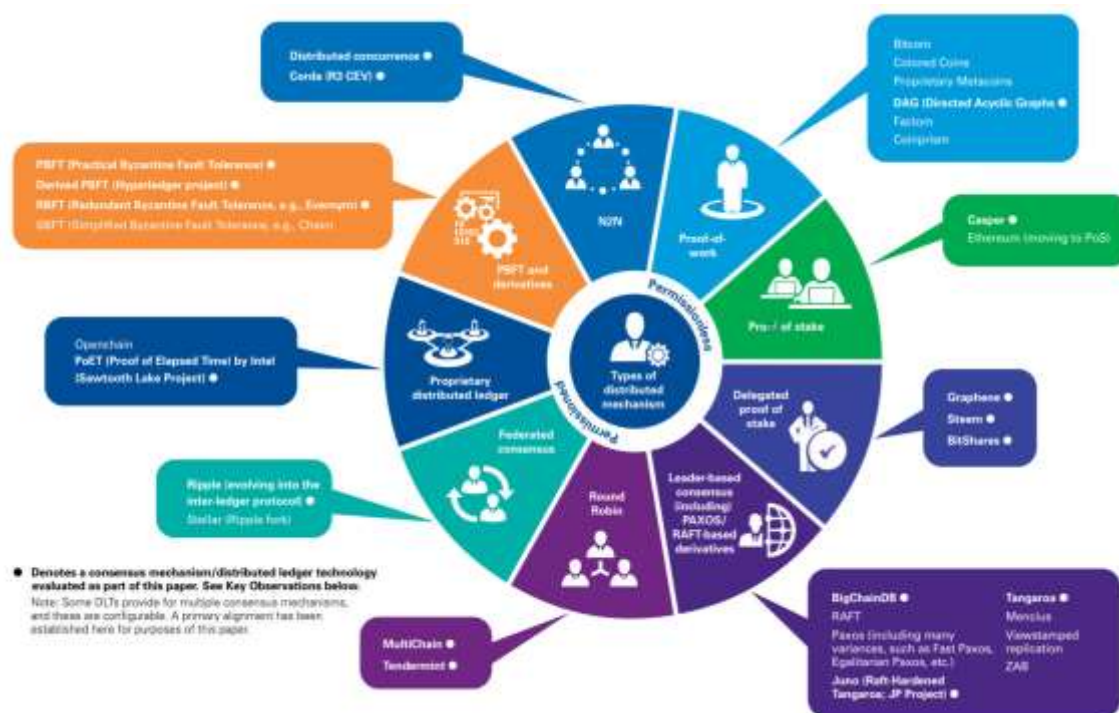
Постизање договора око новог стања главне књиге је један од највећих проблема дистрибуираних система. Консензус је процес постизања договора између свих чворова у мрежи о исправном стању података и циљ је да сви чворови деле исте податке. Ово се постиже консензус алгоритмима. Ови алгоритми осигуравају да су уписани подаци у главну књигу исти за све чворове у мрежи и тако спречавају малициозне учеснике да манипулишу подацима.

Осврнућемо се на неке најпопуларније консензус алгоритме.

Доказ посла (енг. Proof of work) је консензус алгоритам базиран на лутрији у коме се подразумева решавање математичке загонетке која је захтевна за израчунавање како би се креирао нови блок у систему. Најпознатији систем базиран на доказу посла је хешкеш (Адам Бек 1997.) и служи за лимитирање спама и спречавање denial-of-service напада. Једини начин да се пронађе решење математичке загонетке су brute-force алгоритми који мање више погађају тачно решење. Провера решења је једноставна и не захтева комплексна израчунавања. Процес израчунавања се назива рударењем, а чворови у мрежи који покушавају да дођу до решења се називају рудари. Рудари имају економски подстицај јер проналазак хеша новог блока доводи до стварања нових ресурса који припадају рудару (у неким блокчејн система) и добијају провизију од трансакција које су валидиране у том блоку.

Доказ улога (енг. Proof of stake) је консензус алгоритам који је такође базиран на лутрији и представља генерализацију доказа посла. У овом случају су чворови познати као валидатори и они валидирају трансакције како би зарадили провизију од истих (овде нема стварања новог ресурса). Чворови се случајно бирају за валидаторе блокова, а ова вероватноћа зависи од величине улога неког чвора. Уколико чвор А поседује два ресурса, а чвор Б један ресурс, чвор А има већу шансу да буде постављен као валидатор. Кључни аспект је случајност, како би се избегла ситуација где најбогатији чворови стално добијају улогу валидатора.

Доказ протеклог времена (енг. Proof of elapsed time) је консензус алгоритам који је базиран на лутрији. Овде се проблем консензуса решава извршавањем програма у окружењу са поверењем. Насумично се бирају чворови који треба да изврше одређене захтеве у неком времену. Ти чворови узоркују случајну променљиву са експоненцијалном дистрибуцијом и чекају онолико времена колико је одређено тим узорком. Варање се спречава верификацијом идентитета, применом окружења од поверења (нпр. Intel SGX) и додатним полисама.



Слика 7 Типови консензус механизма (преузето са *devopedia.org*)

Поједностављена Византијска толеранција отказа (енг. Simplified Byzantine fault tolerance) је алгоритам базиран на гласању и имплементира прилагођену верзију практичне Византијске толеранције отказа. Идеја је да постоји јединствени валидатор који слаже предложене трансакције и формира нови блок. У овом случају је валидатор позната страна пошто је алгоритам намењен за блокчејн систем са контролом приступа. Консензус се постиже тако што неки минимални број чворова у мрежи потврди блок. Како би имао Византијску толеранцију, минимални број чворова за консензус је $2X+1$ у систему са $3X+1$ чворова, где је X број отказа у систему.

3. Ethereum и Dapp-ови

Ethereum је једна од најпознатијих платформи базираних на блокчејн технологији. Ова платформа је отвореног кода, јавна и омогућује развој и рад разних пројеката који желе да искористе предности децентрализације. За разлику од Bitcoin-а, што је платформа која искључиво служи за размену дигиталне валуте, на Ethereum-у могу да се имплементирају најразличитије функционалности.

Ране блокчејн апликације као што су Bitcoin су једино дозвољавале корисницима ограничен скуп предефинисаних операција. За разлику од оваквих пројеката, Ethereum нуди корисницима да праве своје операције. Ово нас доводи до концепта Ethereum Виртуалне Машине (EVM). EVM је Ethereumово окружење за извршавање, и служи за извршавање паметних уговора. Пошто сваки Ethereum чвор извршава Ethereum Виртуалну Машину, апликације изграђене на овој бази добијају погодности децентрализације без потребе да користе неки посебно дизајниран блокчејн.

Слично као и у Bitcoin-у, Ethereum има свој концепт рударења и своју криптовалуту Етар. Ethereum тренутно користи доказ посла као консензус алгоритам (као и Bitcoin), али је у плану прелазак на доказ улога. До тада Етар ствара платформа и исплаћује га рударима који израчунају хеш новог блока трансакција.

3.1. Паметни уговори

Паметни уговори су програмски кодови који на високом нивоу описују интеракције између корисника и дистрибуиране главне књиге. Ови кодови могу да се компајлирају у EVM бајткод и да се извршавају на Ethereum Виртуалној Машини. Као што Bitcoin има свој BitcoinScript, тако и Ethereum има свој језик за писање паметних уговора, Solidity. Разлика између ова два је то што је Solidity Тјуринг комплетан и нуди много више могућности, док BitcoinScript подржава само финансијске трансакције.

Паметни уговори се постављају на Ethereum блокчејн где добијају своје јавне адресе. Ово заправо значи да ће код ових паметних уговора извршавати машине-чворови од рудара. Корисници могу приступати овом уговору путем те јавне адресе и вршити интеракцију са њим. Појединости интеракције зависе од функционалности самог паметног уговора.

Нпр. Алиса и Боб желе да се опкладе у 100 етара у исход фудбалске утакмице. Они се договоре око поверљиве веб странице за резултате, и свако пошаље по 100 етара на јавну адресу паметног уговора. Након што се утакмица заврши, паметни уговор ће проверити резултат утакмице на основу поверљиве веб странице. У зависности од резултата, паметни уговор ће аутоматски исплатити 200 етара победнику опкладе.

3.2. Гас

Као што смо већ напоменули, Solidity, језик у коме се пишу Ethereum паметни уговори, је Тјуринг комплетан. Ово доводи до неких очигледних проблема. Злонамеран учесник би могао да напише паметни уговор у коме постоји бесконачна петља. Оваква ситуација би довела до престанка рада целе мреже. Да би се суочили са овим проблемом, људи који развијају Ethereum су измислили концепт гаса. Гас је јединица која служи за мерење колико кошта извршавање једне операције на Ethereum Виртуалној Машини. Нпр. ирачунавање збира два броја кошта 3 гаса. Дакле, свако ко жели да направи нову трансакцију на Ethereum блокчејну ће морати да плати гас за њу. Уз сваку трансакцију се мора послати количина гаса довољна да покрије трошкове извршавање трансакције. Уколико се пошаље мање него што је потребно, трансакција ће бити одбијена. Уколико се пошаље више него што је било потребно, трансакција ће бити прихваћена и вишак гаса ће бити враћен кориснику. Гас има своју одређену цену у односу на вредност Етара.

3.3. Децентрализоване апликације (Дарр-ови)

Оригинална замисао Ethereuma је да он делује као „светски рачунар“. Чворови мреже би сви имали EVM као мидлвер слој, који би давао утисак да су сви рачунари део једног кохерентног система.

Традиционална веб апликација би имала три дела: фронтенд, бекенд и базу података. Код децентрализованих апликација постоји сличан приступ.

Фронтенд део, који представља предњи део апликације са којим корисник интерагује, може бити хостован на неком централном серверу, а може бити и хостован на неким децентрализованим платформама за складиштење као што су Svarm. Овако би се успоставила потпуна децентрализација.

Бекенд део, тј. серверски део који представља логику и спој између података и фронтенда, би се имплементирао као паметни уговор на некој блокчејн мрежи. Овако се тај бекенд не би извршавао на неком централизованом серверу, већ би га извршавали сви чворови мреже на потпуно дистрибуиран начин.

База података би у овом случају била сама дистрибуирана главна књига блокчејн система. Замисао оснивача Ethereum-а, Виталик Бутерина, је од почетка била да Ethereum има улогу светског рачунара где чворови извршавају децентрализоване апликације. Идеја је да децентрализација нуди бројне предности у финансијској, производној, здравственој и информативној индустрији.

3.4. Опис коришћених алата и технологија

Као подршка развоју окружења за бављење блокчејн системом је искоришћен Truffle Suite. Truffle Suite обухвата неколико алата за олакшан рад са системима на бази Ethereum.

3.4.1. Ganache



Слика 8 Ganache (преузето са www.trufflesuite.com)

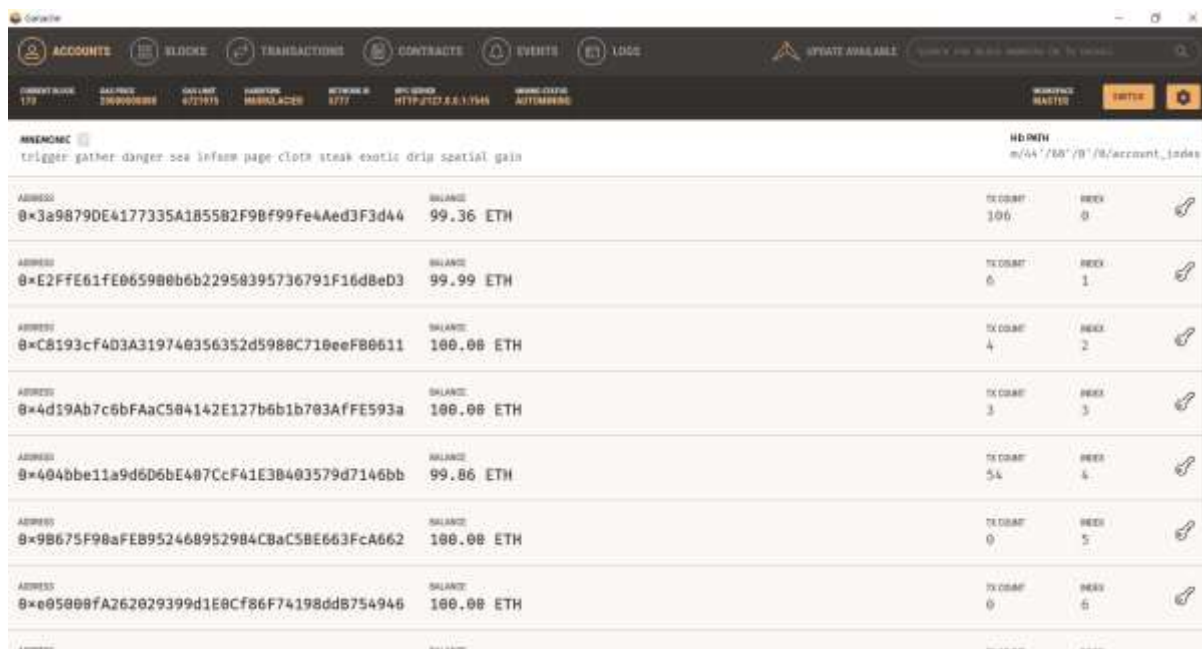
Ganache је персонални блокчејн намењен за брзи развој Ethereum и Corda дистрибуираних апликација. Може се користити током целог циклуса развоја, омогућавајући развој, покретање и тестирање децентрализованих апликација у безбедном и детерминистичком окружењу.

Ganache има своју десктоп апликацију која је искоришћена у склопу овог рада да се симулира понашање Ethereum блокчејна. Развијати децентрализоване апликације на примарном Ethereum блокчејну није практично из више разлога. Првенствено, трансакције на Ethereum-у коштају. Због овога Ethereum има своју тест мрежу, где Етар нема вредност, али ни она није практична за коришћење. Ово је због тога што је рачунарски захтевно бити пуноправни чвор на Ethereum мрежи. Зато је коришћен Ganache, који практично симулира понашање правог Ethereum блокчејн система.



Слика 9 Почетни екран апликације

Ganache нам омогућава истовремено подешавање и одржавање различитих блокчејнова тј. простора за рад (енг. workspace). Притиском на дугме quickstart нам Ganache може направити блокчејн са подразумеваним параметрима док дугметом new workspace можемо ручно подесити подешавање блокчејна по нашем нахођењу.



Слика 10 Главни екран апликације

Ganache нуди комплетан мониторинг целог система. На почетном екрану, осим информација од тренутном блоку, цене гаса, лимит гаса итд., имамо све „налоге“ који су активни на блокчејн-у. Ову нису прави налози у смислу речи, већ само одређене адресе којима су придодати Ethereum токени за сврхе тестирања. У Ganache-у можемо да водимо рачуна и да примамо информације о свим налозима, о блоковима, о трансакцијама, о паметним уговорима, о догађајима и о неким посебним логовима, дефинисаним од стране корисника.

3.4.2. Truffle



Слика 11 Truffle (преузето са www.trufflesuite.com)

Truffle је окружење за рад са блокчејновима који раде у склопу Ethereum виртуелне машине базирано на NodeJS. NodeJS је окружење отвореног кода за извршавање JavaScript кода ван претраживача. Truffle олакшава развој и тестирање Ethereum апликација. Неке ствари које Truffle нуди су:

- Уграђени механизми за компајлирање паметних уговора и линковање бинарних датотека
- Могућност писања аутоматских тестова за брзи развој
- Писање скрипти за подешавање уговора и миграција
- Управљање са становишта мрежа
- Управљање пакетима и библиотекама
- Интерактивна конзола

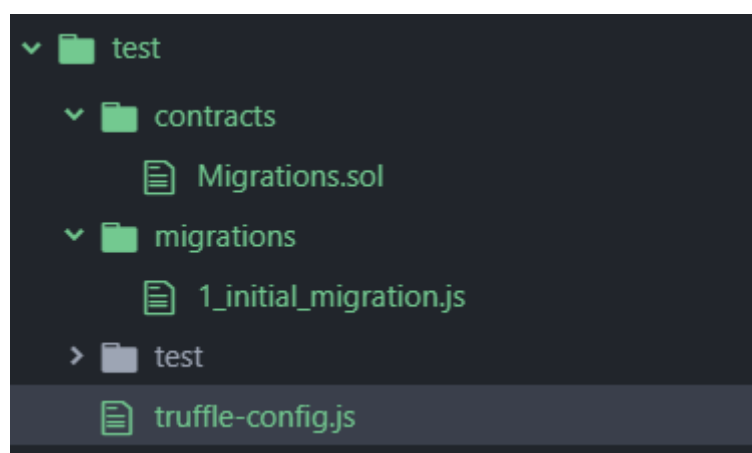
Truffle се скида и инсталира путем NPM-a (Node Packet Manager). Након подешавања самог оквира, можемо започети коришћење Truffle-a.

```
Nikola@DESKTOP-A77496V MINGW64 ~/Documents/fakultet/Master/res/test (master)
$ truffle init

Starting unbox...
=====
```

Слика 12 Креирање новог пројекта

Командом `truffle init` конструишемо нови Truffle пројекат у директоријуму. Ова команда ће направити генерични, празан пројекат. Truffle нуди могућност да се пројекат иницијализује са различитим подешавањима, параметрима, датотекама итд. у зависности од потребе инжењера.



Слика 13 Структура пројекта

На слици изнад можемо видети почетну структуру пројекта. У директоријуму `contracts` се налазе све Solidity датотеке везане за паметне уговоре. `Migrations.sol` је посебна датотека која садржи подешавања за миграцију паметних уговора на блокчејн. У директоријуму `migrations` се налазе скрипте које су одговорне за правилно преношење паметних уговора са локала на блокчејн мрежу. У директоријуму `test` би стојале Solidity или JavaScript датотеке за тестирање. Датотека `truffle-config.js` садржи сва подешавања за Truffle пројекат.

3.4.3. Web3.js

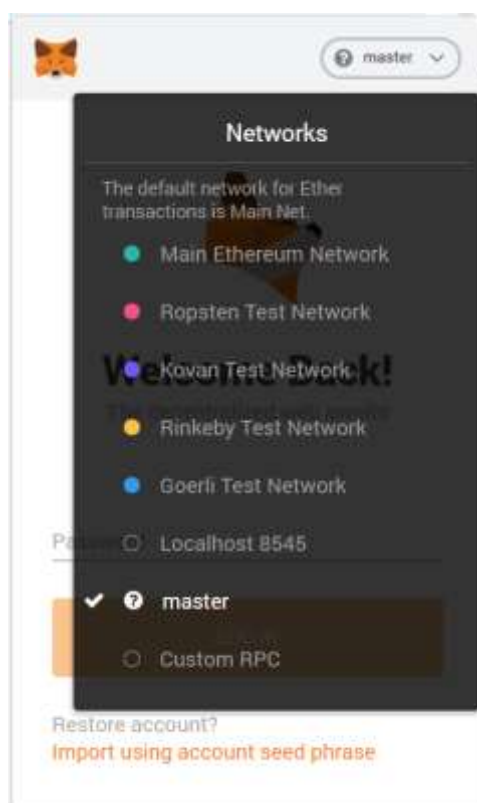
Web3.js је JavaScript библиотека која омогућава рад са Ethereum чворовима. Сваки Ethereum чвор има RPC API и методе из овог API-ја је могуће позивати путем метода из Web3.js библиотеке.

3.4.4. MetaMask



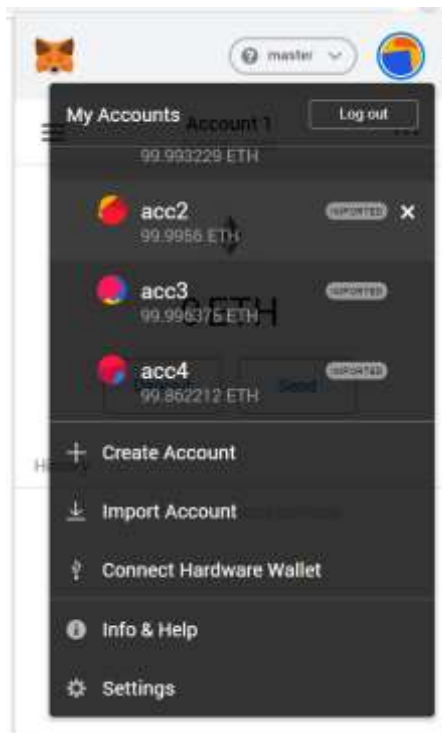
Слика 14 MetaMask (преузето са metamask.io)

MetaMask је новчаник за криптовалуте и спрега претраживача и блокчејна. Долази у форми плагина за претраживач (Mozilla, Chrome..). Омогућава паметно управљање Ethereum адресама и кључевима из претраживача. Помоћу MetaMask-а, фронтенд апликација може да прослеђује релевантне корисничке податке ка блокчејну.



Слика 15 Одабир блокчејн мреже

Поред заштите података у форми логовања, MetaMask нам нуди да бирамо на коју блокчејн мрежу желимо да се повежемо. Подразумевано су понуђене неке најкоришћеније мреже, а са опцијом Custom RPC можемо додати још неку мрежу.



Слика 16 Одабир "налога" тј. јавне адресе

MetaMask нам нуди да користимо различите „налоге“, тачније, јавне (и приватне) адресе на некој мрежи. Можемо да правимо нове налоге, додајемо постојеће, чак и да преузмемо налог са хардверског новчаника.

Такође, MetaMask нуди све остале потребне опције за интеракцију са блокчејном. Када фронтенд апликација жели да обави неку трансакцију са корисничким налогом, MetaMask ће тражити од корисника потврду да жели да изврши трансакцију. Ова потврда ће такође обавестити корисника о цени трансакције, као и о другим појединостима.

4. ОПИС СОФТВЕРСКОГ РЕШЕЊА

У овом поглављу ће бити дат комплетан опис и начин функционисања самог софтверског решења. Замисао овог рада је да се имплементира децентрализована апликација преко које је могуће правити децентрализоване изборе. Дакле, апликација нуди следеће могућности: креирање нових избора, додавање нових кандидата (уколико је корисник власник избора), додавање нових учесника (уколико је корисник власник избора), гласање на изборима (уколико је корисник учесник у неким изборима) и приказивање резултата избора.

4.1. Решење и начин функционисања

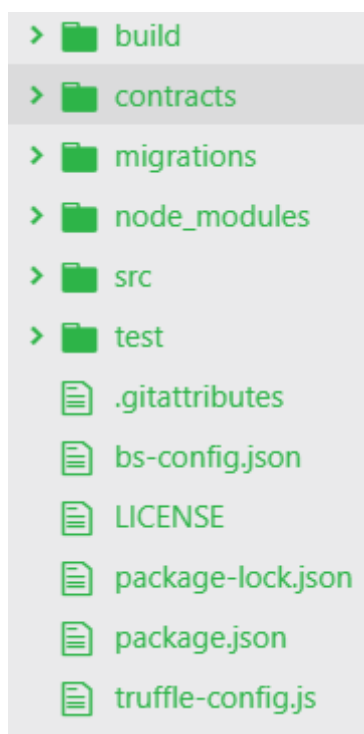
У овом делу рада ће бити описано само софтверско решење. Софтверско решење се састоји од три функционалне целине: дистрибуирана база података (тј. блокчејн), паметног уговора (фактички бекенд) и фронтенд апликација. Сваки део ће бити описан понаособ.

4.1.1. Блокчејн

Решење треба да користи Ethereum блокчејн платформу као своју дистрибуирану базу података, као и Ethereum виртуелну машину за своје извршавање. Пошто је развој на самом Ethereum блокчејну веома скуп, користи се Ganache апликација за симулацију блокчејна. Ganache нам нуди произвољан број налога за сврхе тестирања.

4.1.2. Бекенд

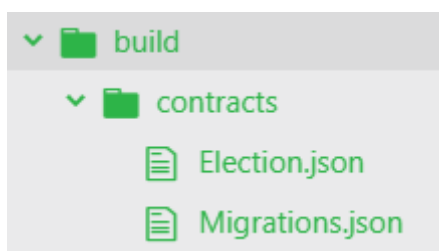
Цела апликација је развијена као Truffle пројекат. На наредној слици је приказана структура пројекта.



Слика 17 Структура пројекта

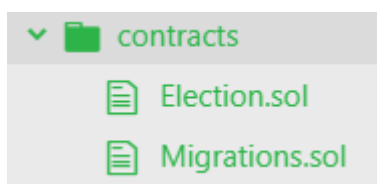
У директоријуму *src* се налазе све датотеке везане за фронтенд апликацију, док се све остало фактички може третирати као бекенд.

Директоријум *build* садржи датотеке које представљају компајлиране паметне уговоре. Ове датотеке су у JSON формату. Оне се у овом формату шаљу на блокчејн, и блокчејн он њих ствара функционалне паметне уговоре.



Слика 18 Build директоријум

У директоријуму *contracts* се налазе паметни уговори који су писани у Solidity програмског језику.



Слика 19 Contracts директоријум

С обзиром да је ажурирање верзије паметног уговора на Ethereum блокчејну мало компликованије, уз сваки паметни уговор постоји још један уговор који се аутоматски генерише и најчешће се назива Migrations.sol. Овај уговор садржи неке метаподатке о примарном паметном уговору и води рачуна о промени верзија уговора.

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity >=0.4.21 <0.7.0;
3
4 contract Migrations {
5     address public owner;
6     uint public last_completed_migration;
7
8     constructor() public {
9         owner = msg.sender;
10    }
11
12    modifier restricted() {
13        if (msg.sender == owner) _;
14    }
15
16    function setCompleted(uint completed) public restricted {
17        last_completed_migration = completed;
18    }
19 }
20
```

Слика 20 Migrations.sol

Променљива owner представља адресу власника паметног уговора, тј. адресу са које је послата трансакција-захтев за стварањем паметног уговора. Променљива last_completed_migration је број верзије уговора. Ова датотека се аутоматски генерише и најчешће се не мења уопште. Кључна реч msg представља трансакцију која је довела до позивања функција у којој се кључна реч спомиње (у Ethereum-у не постоје „нормални“ позиви функција већ се све врши преко трансакција). Тако конструкција msg.sender означава адресу пошиљаоца те трансакције.

У датотеци Election.sol се налази комплетна имплементација паметног уговора. Сада ће се дати анализа кода паметног уговора писаног у Solidity језику.

```
6      struct Participant {
7          uint id;
8          address addr;
9          string name;
10     }
11
12     struct Owner {
13         uint id;
14         address addr;
15         string name;
16     }
17
18     struct Candidate {
19         uint id;
20         string name;
21         uint voteCount;
22     }
23
```

Слика 21 Структуре података

На слици изнад су дате три структуре података, које представљају различите врсте корисника у систему. Корисник може бити учесник (енг. Participant, може да учествује у изборима), власник (енг. Owner, креатор неких избора) или кандидат (енг. Candidate, кандидат на изборима). Структуре Participant и Owner су исте и садрже три поља: id, addr и name. Поље id је типа uint (неозначени цео број) и представља јединствени идентификатор корисника. Поље addr је типа address (Ethereum јавна адреса) и представља јавну адресу корисника. Поље name је типа string (низ карактера) и представља име корисника. Структура Candidate се разликује од осталих. Она садржи следећа поља: id (јединствени идентификатор), name (име кандидата) и voteCount (број гласова које је кандидат освојио).

```
24     struct SingleElection {
25         uint id;
26         mapping(uint => Owner) owner;
27         mapping(uint => Candidate) candidates;
28         uint candidatesCount;
29         mapping(uint => Participant) election_participants;
30         mapping(address => bool) voted;
31         uint participantsCount;
32     }
```

Слика 22 Структура једних избора

Наредна структура је SingleElection. Ова структура представља једну инстанцу избора. Сада ћемо се осврнути на сва поља ове структуре. Прво поље је id, што представља јединствени идентификатор избора. У овом тренутку ћемо напоменути да у Solidity програмског језику постоји одређени тип података који се зове mapping. Овај тип представља хеш мапу које се у Solidity-у користе уместо низова. Ова мапа садржи парове кључ-вредност. Дакле, наредна два поља су типа mapping: owner и candidates. Owner поље садржи само један пар кључ-вредност, и то је кључ број 0, а вредност је Owner структура власника избора. Разлог зашто је овде искоришћен тип mapping је што је у Solidity-у немогуће слати структуре које у себи садрже структуре (што ће бити релевантно касније). Поље candidates представља парове кључ-вредност где су кључеви јединствени идентификатори кандидата а вредности одговарајуће Candidate структуре. Постоје још два mapping поља: election_participants и voted. Election_participants поље је хеш мапа која памти све учеснике у изборима на основу њихових јединствених идентификатора. Поље voted води рачуна о томе да ли је неки корисник са неке адресе гласао већ или не. Када се прими глас са неке јавне адресе, вредност за тај кључ (адресу) ће бити постављена на true (истинито). Сада ћемо се осврнути на поља самог паметног уговора.

```
34     uint public singleElectionCount;
35     mapping(address => SingleElection) public owners;
36     mapping(address => SingleElection) public participants;
37     mapping(uint => SingleElection) public singleElections;
```

Слика 23 Поља паметног уговора

Постоје четири поља паметног уговора (могу се сматрати глобалним променљивама). Поље singleElectionCount је бројач чија вредност представља број тренутно актуелних избора. Хеш мапа owners мапира јединствене јавне адресе власника избора на саме структуре избора. Слична је улога и поља participants, које

мапира јавне адресе учесника у изборима на структуру избора у коме је предвиђено да учествују. Поље `singleElections` служи као низ свих структура избора и намењено је да се кроз овај низ итерира уз помоћ поља `singleElectionCount`.

У наредном делу ће бити описане све функције које су садржане у `Election.sol` паметном уговору.

```
114     function newSingleElection(string memory _owner_name) public {
115         require(owners[msg.sender].id==0);
116         singleElectionCount ++;
117         Owner memory t_owner;
118         t_owner = Owner(singleElectionCount, msg.sender, _owner_name);
119         singleElections[singleElectionCount] = SingleElection(singleElectionCount,0,0);
120         singleElections[singleElectionCount].owner[0] = t_owner;
121         owners[msg.sender] = singleElections[singleElectionCount];
122     }
```

Слика 24 Функција за креирање нових избора

Функција `newSingleElection` служи за креирање нових избора. Једини параметар који се прослеђује овој функцији је `_owner_name` и представља име власника избора. Идентификатор `memory` се мора нагласити за типове података као што су низови, и означава у ком меморијском простору ће се променљива сачувати. У 115-ој линији можемо видети услов за улазак у ову функцију. Овај услов проверава да ли је корисник који покушава да направи изборе већ власник неких других избора. Уколико није, извршавање функције се наставља и број појединачних избора се повећава у следећој линији. У наредне две линије (117, 118) се конструише инстанца структуре `Owner` са одговарајућим параметрима. Након тога, конструише се нова структура `SingleElection` и додаје се у мапу осталих избора са правилним кључем (`singleElectionCount`). Притом се у линији 120 тој структури додаје одговарајућа структура власника. На крају, у линији 121, се новокреирани избори повезују са адресом власника путем `owners` хеш мапе.

```
124     function getStatus() public view returns (uint, uint){
125         if(owners[msg.sender].id!=0){
126             return (0, owners[msg.sender].id);
127         }
128         if(participants[msg.sender].id!=0){
129             uint electionId = participants[msg.sender].id;
130             if(!singleElections[electionId].voted[msg.sender]){
131                 return (1, participants[msg.sender].id);
132             } else {
133                 return (2, participants[msg.sender].id);
134             }
135         }
136         return (3, 0);
137     }
```

Слика 25 Функција која враћа статус корисника

Када неки корисник приступа децентрализованој апликацији, паметни уговор мора да провери која је његова улога (да ли је учесник, власник итд.). Када год се освежи фронтенд апликација, она тражи од паметног уговора информацију о улози корисника путем функције `getStatus`. Ова функција не прима никакве параметре, али враћа два неозначена цела броја. Први број представља код улоге, а други број представља идентификатор избора. Дакле, уколико постоје избори у мапи `owners` за дату адресу (`msg.sender`) и њихов идентификатор није једнак 0, знамо да је корисник власник неких избора. Уколико је ово случај, шаљемо назад код 0 (корисник је власник) и идентификатор избора чији је власник. Ако корисник није власник, преостају три могућности: корисник је учесник и није још гласао (код 1), корисник је учесник и гласао је (код 2), корисник није ни власник ни учесник (код 3). Кључна реч `view` у називу функције означава да ова функција ни на који начин не мења стање блокчејн-а.

```
51     function addCandidate(string memory _name) public {
52         SingleElection storage election = owners[msg.sender];
53         uint electionId = election.id;
54         SingleElection storage t = singleElections[electionId];
55         election.candidatesCount++;
56         t.candidatesCount++;
57         uint id = election.candidatesCount;
58         election.candidates[id] = Candidate(id, _name, 0);
59         t.candidates[id] = Candidate(id, _name, 0);
60     }
```

Слика 26 Функција за додавање кандидата

Уколико је корисник препознат као власник избора, он има могућност да додаје кандидате на изборима. Томе управо служи функција `addCandidate` која прихвата један параметар `_name`, тј. име кандидата. Да бисмо додали новог кандидата, прво морамо да добавимо одговарајуће инстанце структура избора. Ово се ради са два места, јер у суштини имамо дуплиране инстанце избора, једна структура се налази као вредност у мапи `owners`, а друга се налази као вредност у мапи `singleElections` (ово мора овако да се имплементира, у `Solidity`-у различити кључеви мапа не могу као вредност да имају исте структуре). Дакле, прво у линији 52 добављамо одговарајућу инстанцу структуре из `owners` мапе на основу адресе пошиљаоца, а потом у 54-ој линији добављамо инстанцу из `singleElections` мапе на основу идентификатора. Сада, када смо добавили структуре, можемо да додамо кандидата. У обе структуре се инкрементује број кандидата (`candidatesCount`), па се потом у обе структуре уписује нови кандидат са одговарајућим идентификатором.

```
62     function addParticipant(address _addr, string memory _name) public {
63         SingleElection storage election = owners[msg.sender];
64         uint electionId = election.id;
65         SingleElection storage singleElection = singleElections[electionId];
66         election.participantsCount++;
67         singleElection.participantsCount++;
68         uint id = election.participantsCount;
69         election.election_participants[id] = Participant(id, _addr, _name);
70         participants[_addr] = election;
71         singleElection.election_participants[id] = election.election_participants[id];
72     }
```

Слика 27 Функција за додавање учесника

Уколико је корисник препознат као власник избора, он има могућност да додаје учеснике (гласаче) на изборима. Томе управо служи функција `addParticipant` која прихвата два параметра: `_name` и `_addr`. Параметар `_name` представља име учесника, док параметар `_addr` представља јавну адресу учесника, са које ће он гласати. Да бисмо додали новог учесника, прво морамо да добавимо одговарајуће инстанце структура избора. Ово се ради са два места, једна структура се налази као вредност у мапи `owners`, а друга се налази као вредност у мапи `singleElections`. Дакле, прво у линији 63 добављамо одговарајућу инстанцу структуре из `owners` мапе на основу адресе пошиљаоца, а потом у 65-ој линији добављамо инстанцу из `singleElections` мапе на основу идентификатора. . Сада, када смо добавили структуре, можемо да додамо кандидата. У обе структуре се инкрементује број учесника (`participantsCount`), па се потом у обе структуре уписује нови кандидат са одговарајућим идентификатором. Такође, одговарајуће изборе `election` додајемо у мапу `participants` где је адреса новог учесника кључ.


```
74     function vote (uint _candidateId, uint _electionId) public {
75         require(!singleElections[_electionId].voted[msg.sender]);
76
77         SingleElection storage election = singleElections[_electionId];
78
79         election.candidates[_candidateId].voteCount++;
80
81         election.voted[msg.sender] = true;
82     }
```

Слика 28 Функција за гласање

Функција `vote` служи за само гласање. Она прима два параметра: `_candidateId` и `_electionId`. На основу ова два параметра је познато ком кандидату у којим изборима треба проследити глас. Прва линија функције проверава да ли се са те адресе већ гласало у тим изборима. Уколико није, добавља се структура избора из мапе `singleElections` и инкрементира се број гласова за одговарајућег кандидата. Потом се у `voted` мапи поставља одговарајућа вредност на `true` како се са исте адресе не би могло гласати више пута.

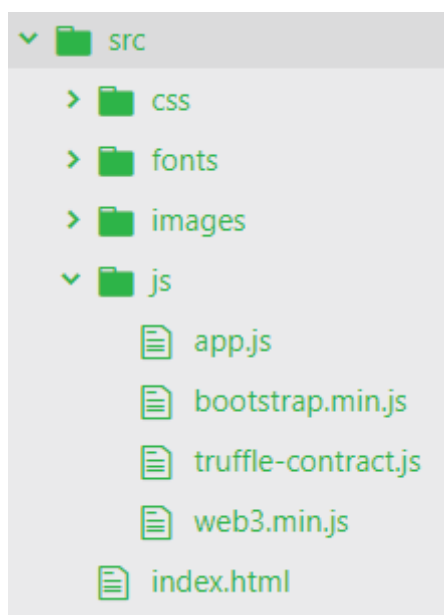
```
109     function getCandidate(uint electionId, uint key) public view returns (uint, string memory, uint){
110         Candidate storage t = singleElections[electionId].candidates[key];
111
112         uint id = t.id;
113         string memory name = t.name;
114         uint voteCount = t.voteCount;
115
116         return (id, name, voteCount);
117     }
118 }
```

Слика 29 Функција која враћа информације о кандидату

`getCandidate` је последња функција паметног уговора. Њу позива фронтенд апликација како би добила информацију о кандидатима. Параметри ове функције су `electionId` и `key`. `ElectionId` представља идентификатор релевантних избора, док је `key` позиција кандидата у листи кандидата. Дакле, фронтенд апликација ће итерирати по листи кандидата неких избора уз помоћ `key` параметра.

4.1.3. Фронтенд

Фронтенд апликација је типична NodeJS апликација писана у JavaScript-у.



Слика 30 Структура фронтенд апликације

У *css* директоријуму се налазе датотеке везане за стилове веб странице. У *fonts* директоријуму се налазе подаци везани за фонтове који се користе на веб страници. У *images* директоријуму се налазе све слике везане за веб страницу. У *js* директоријуму се налазе изворне датотеке писане у JavaScript језику. Датотека *index.html* је HTML основа за целу веб страницу и у њој је описан генерални изглед веб странице. За нас је од највећег значаја датотека *app.js* која садржи у себи функционалност фронтенд апликације.

```
1  App = {
2    web3Provider: null,
3    electionId: 0,
4    contracts: {},
5    account: '0x0',
6
7    init: function() {
8      return App.initWeb3();
9    },
```

Слика 31 Поља класе *App*

App је JavaScript класа која ће описивати функционалност фронтенд апликације. Апликација има четири поља: *web3Provider*, *electionId*, *contracts* и *account*. Поље *web3Provider* служи за смештање инстанце добављача који представља спрегу између апликације и *Ethereum* чвора. *ElectionId* служи за памћење идентификатора специфичних избора са којима апликација тренутно ради. У променљивој *contracts* ће се налазити сви паметни уговори с којима ће апликација комуницирати (у нашем

случају ће то бити само један уговор). Поље account служи за смештање јавне адресе тренутног корисника апликације.

Функција init се позива сваки пут када се учита веб страница.

```
270     $(function() {  
271         $(window).load(function() {  
272             App.init();  
273         });  
274     });
```

Слика 32 Освежавање странице

Након тога се позива initWeb3 где се инстанцира добављач уколико апликација нема већ један.

```
11     initWeb3: function() {  
12         if (typeof web3 !== 'undefined') {  
13             App.web3Provider = web3.currentProvider;  
14             web3 = new Web3(web3.currentProvider);  
15         } else {  
16             App.web3Provider = new Web3.providers.HttpProvider('http://localhost:7545');  
17             web3 = new Web3(App.web3Provider);  
18         }  
19         return App.initContract();  
20     },
```

Слика 33 Иницијализација web3 библиотеке

Уколико се web3 библиотека већ има инстанцу која је повезана са Ethereum чвором, апликација ће само преузети ту инстанцу (линије 12-14). Ако не, конструише се нови web3 добављач и повезује се са локалним Ethereum чвором. Ganache хостује локални чвор на порту 7545 (линије 15-17). Када смо успешно спојили апликацију са Ethereum чвором, следећи корак је иницијализација паметног уговора.

```
22     initContract: function() {
23
24         $.getJSON("Election.json", function(election) {
25             App.contracts.Election = TruffleContract(election);
26             App.contracts.Election.setProvider(App.web3Provider);
27             return App.render();
28         });
29     },
```

Слика 34 Иницијализација паметног уговора

Рекли смо раније да датотека Election.json настане компајлирањем паметног уговора Election.sol. Апликација мора да додави уговор путем JavaScript-ове getJSON методе. Након тога се инстанцира уговор (линија 25), па се уговору придружи одговарајући web3 добављач (линија 26). Након тога се позива најкомплекснија функција апликације, функција за исцртавање render.

```
31     render: function() {
32         var electionInstance;
33         var loader = $("#loader");
34         var content = $("#content");
35         var owner_content = $("#owner_content");
36         var no_content = $("#no_content");
37
38         loader.show();
39         content.hide();
40         owner_content.hide();
41         no_content.hide();
42     }
```

Слика 35 Почетак функције за исцртавање

На почетку функције се иницијализују променљиве. Променљива electionInstance ће садржати инстанцу избора у себи, док остале променљиве представљају делове странице које ће се исцртавати (eng. render). Loader је поглед који ће се приказати кориснику док се чека завршетак неке операције у позадини, content је поглед који се приказује учеснику у изборима, owner_content је поглед који ће се приказивати власнику избора док ће се no_content приказивати корисницима који ни на

који начин не учествују у неким изборима. На почетку се приказује loader, док се скривају остали погледи (линије 38-41).

```
43     window.ethereum.enable();
44     web3.eth.getCoinbase(function(err, account) {
45         if (err === null) {
46             App.account = account;
47             $("#accountAddress").html("Your Account: " + account);
48         }
49     });
```

Слика 36 Добављање активне јавне адресе

Линија 43 је посебна инструкција која омогућава рад MetaMask екстензије за претраживач у контексту апликације. Након тога можемо да добавимо јавну адресу корисника која му је тренутно активна у MetaMask.

```
51     App.contracts.Election.deployed().then(function(instance) { // get contract instance
52         electionInstance = instance;
53         return electionInstance.getStatus({from: App.account}).then(function(result){
54             App.electionId = result[1].c[0];
55             var code = result[0].c[0];
56             switch(code){
```

Слика 37 Добављање инстанце уговора и статус корисника

Да би апликација наставила даље са радом, мора да добави инстанцу паметног уговора (линије 51-52). Након тога, апликација од паметног уговора захтева информацију о статусу тренутног корисника на основу његове јавне адресе (линија 53). Паметни уговор ће вратити статусни код (0, 1, 2 или 3) и идентификатор избора у коме корисник учествује (линије 54-55). Потом апликација улази у switch исказ на основу примљеног статусног кода.

```
57         case 3: // user is not involved in any election
58             loader.hide();
59             no_content.show();
60             break;
```

Слика 38 Случај 3

Уколико је примљен код 3, тренутни корисник не учествује ни на једним изборима. Њему се приказује поглед no_content који му ставља до знања да не учествује на изборима и нуди му опцију да креира изборе.

Election

You are not a participant in any election

Would you like to create an election?

Enter your name here

Yes

Слика 39 Поглед корисника који не учествује у изборима

Следећи случај је код 0, што означава да је корисник власник неких избора.

```
61 case 0: // user is owner of an election
62 var candidatesResultsOwner = $("#candidatesResultsOwner");
63 candidatesResultsOwner.empty();
64 electionInstance.owners(App.account).then(function(singleElection){
65 for(i=1;i<=singleElection[1];i++){
66     electionInstance.getCandidate(singleElection[0], i, {from: App.account}).then(function(candidate) {
67         var id = candidate[0];
68         var name = candidate[1];
69         var votes = candidate[2];
70         var candidateTemplate = "<tr><th>" + id + "</th><td>" + name + "</td><td>" + votes + "</td></tr>"
71         candidatesResultsOwner.append(candidateTemplate);
72     });
73 }
74 });
75 loader.hide();
76 owner_content.show();
77 break;
```

Слика 40 Случај 0, када је корисник власник избора

Променљива `candidatesResultsOwner` представља табелу у којој се налазе сви кандидати и информације о њима. Апликација сваки пут кад испртава страницу ће испразнити ову табелу и добити најновије информације о кандидатима од паметног уговора (линије 63-69). Сваки кандидат ће бити додат у табелу (линије 70-71). Након тога се сакрива `loader` и приказује се `owner_content` (линије 75-76).

Election

#	Name	Votes
1	Marko Markovic	0
2	Jovan Jovanovic	0

Enter participant public address

Enter participant name

Add participant

Enter candidate name

Add candidate

Слика 41 Случај 0, поглед власника избора

Уколико је корисник учесник на изборима, који није још гласао, случај је код 1.

```

79 case 1: // user is a participant that has not voted in an election
80 var candidatesResults = $("#candidatesResults");
81 candidatesResults.empty();
82 var candidatesSelect = $("#candidatesSelect");
83 candidatesSelect.empty();
84 electionInstance.participants(App.account).then(function(singleElection){
85 for(i=1;i<singleElection[1];i++){
86     var candidate = electionInstance.getCandidate(singleElection[0], i, {from: App.account}).then(function(candidate){
87         var id = candidate[0];
88         var name = candidate[1];
89         var votes = candidate[2];
90         var candidateTemplate = "<tr><th>" + id + "</th><td>" + name + "</td><td>" + votes + "</td></tr>"
91         candidatesResults.append(candidateTemplate);
92         var candidateOption = "<option value='" + id + "'>" + name + "</option>";
93         candidatesSelect.append(candidateOption);
94     });
95 }
96 });
97 loader.hide();
98 content.show();
99 break;

```

Слика 42 Случај 1, корисник је учесник који није гласао

Случај 0 и 1 су доста слични, с тим да учесник осим приказа кандидата има на располагању и картицу где може да одабере кандидата за кога жели да гласа.

Election

#	Name	Votes
1	Marko Markovic	0
2	Jovan Jovanovic	0

Select candidate

Jovan Jovanovic

Vote

Your Account: 0xc8193cf4d3a319740356352d5980c710eefb0611

Слика 43 Случај 1, корисник учествује на изборима

Случај 1 и 2 су идентични, с тим да се у случају 2 (где је учесник већ гласао) сакрива картица са избором кандидата и дугме за гласање.

```
case 2: // user is a participant that has voted in an election
```

Слика 44 Случај 2 и случај 1 су малтене исти

Сада ће бити дат преглед осталих функција у апликацији App.

```
128     castVote: function() {
129         var candidateId = $('#candidatesSelect').val();
130         App.contracts.Election.deployed().then(function(instance){
131             return instance.vote(candidateId, App.electionId, {from: App.account});
132         }).then(function(result){
133             App.render();
134         }).catch(function(err){
135             console.error(err);
136         });
137     },
```

Слика 45 Функција за гласање

Функција `castVote` служи за гласање за појединачног кандидата. Позива се притиском на дугме `Vote` које се исцртава у случају 1 (корисник је учесник који није гласао). Апликација преузме вредност избора учесника (линија 129). Након тога се позива функција паметног уговора `vote` којој се прослеђују идентификатор кандидата као и идентификатор избора (линије 130-131). Потом се страница освежава (линија 133).

```
139     newElection: function() {
140         var ownerName = $("#ownerName").val();
141         App.contracts.Election.deployed().then(function(instance) {
142             return instance.newSingleElection(ownerName, {from: App.account});
143         }).then(function(res){
144             App.render();
145         }).catch(function(err){
146             console.error(err);
147         });
148     },
149     },
```

Слика 46 Функција за креирање нових избора

Функција `newElection` служи за креирање нових избора. Позива се притиском на дугме које приказује у случају 3 (корисник није учесник ни власник избора). Апликација преузме вредност имена власника избора (линија 140) па онда позива функцију паметног уговора `newSingleElection` (линија 142).

```
151     addCandidate: function(){
152         var candidateName = $("#addCandidateName").val();
153         App.contracts.Election.deployed().then(function(instance){
154             return instance.addCandidate(candidateName, {from: App.account});
155         }).then(function(res){
156             App.render();
157         }).catch(function(err){
158             console.error(err);
159         });
160     },
```

Слика 47 Функција за додавање новог кандидата

Функција `addCandidate` служи за додавање новог кандидата на изборима. Позива се притиском на дугме `Add Candidate` које се приказује у случају 0 (корисник је власник избора). Апликација чита вредност имена новог кандидата и позива функцију из паметног уговора `addCandidate`.

```
163     addParticipant: function(){
164         var participantName = $("#addParticipantName").val();
165         var participantAddr = $("#addParticipantAddr").val();
166         App.contracts.Election.deployed().then(function(instance){
167             return instance.addParticipant(participantAddr, participantName, {from: App.account});
168         }).then(function(res){
169             App.render();
170         }).catch(function(err){
171             console.error(err);
172         });
173     }
174 }
```

Слика 48 Функција за додавање новог учесника

Функција `addParticipant` служи за додавање новог учесника на изборима. Позива се притиском на дугме `Add Participant` које се приказује у случају 0 (корисник је власник избора). Апликација преузима вредности за име учесника и његову јавну адресу (линије 164-165). Потом позива функцију паметног уговора `addParticipant`.

5. ЗАКЉУЧАК

Као крајњи резултат истраживања представљеног у овом раду развијена је потпуно функционална децентрализована веб апликација која може да се извршава на Ethereum платформи.

Представљено решење се састоји из три дела: дистрибуиране базе података, бекенд апликације, и фронтенд апликације. Улогу дистрибуиране базе података врши сама Ethereum блокчејн платформа. Овој бази приступа бекенд апликација, у виду паметног уговора који регулише трансакције из и ка блокчејну. Фронтенд апликација, написана у JavaScript-у, комуницира са бекенд апликацијом у циљу исправног приказивања релевантних података кориснику.

Апликација подржава следеће могућности: прављење нових децентрализованих избора, додавање нових кандидата на изборе, додавање нових учесника (гласача) на изборе, гласање на изборима, приказивање резултата избора. Тренутна верзија апликације омогућава искључиво дељење информација о изборима релевантним корисницима проверама јавних адреса корисника. Ово ограничење се може превазићи у будућности додавањем сервиса за чланство (енг. membership service). Овако би корисници могли да учествују на више избора истовремено, а и могли би да прате резултате других избора на којима не учествују, уколико имају дозволу за то.

Предност ове децентрализоване апликације се огледа у томе да пружа много већу заштиту него традиционалне централизоване апликације. Овакав приступ омогућава безбедно и поуздано дељење информација међу корисницима. Малициозни учесници који имају за циљ да компромитују неке изборе имају значајно редуковане могућности у оваквом децентрализованом систему. С друге стране, овакав систем је много захтевнији за имплементацију, као и за одржавање, од традиционалних централизованих система.

Осим већ споменутог сервиса за чланство, прилике за даљи развој укључују додавање могућности за отказивање избора, као и побољшања у графичкој корисничкој спрези.

ЛИТЕРАТУРА

- [1] Др Душан Гајић, *Материјали са предмета Паралелни и дистрибуирани алгоритми и структуре података*, доступно на: <http://www.acs.uns.ac.rs/sr/node/237/4468699>, последњи приступ јул 2020.
- [2] Maarten van Steen, Andrew S. Tanenbaum., *Distributed Systems (third edition)*, 2018.
- [3] Don Tapscott, *Blockchain Revolution*, 2016.
- [4] *Oracle7 Server Distributed Systems Manual Vol.1*, https://docs.oracle.com/cd/A57673_01/DOC/server/doc/SD173/ch1.htm, последњи приступ март 2020.
- [5] *Various types of Network Topology in Networking with Diagram*, <https://jharaphula.com/types-of-network-topology-diagram/>, последњи приступ март 2020.
- [6] *Middleware and Middleware in distributed application*, <https://www.slideshare.net/Rishikese/middleware-and-middleware-in-distributed-application>, последњи приступ март 2020.
- [7] *The Difference Between Blockchain & Distributed Ledger Technology*, <https://tradeix.com/distributed-ledger-technology>, последњи приступ март 2020.
- [8] *Global Blockchain Benchmarking Study*, [https://www.ey.com/Publication/vwLUAssets/ey-global-blockchain-benchmarking-study-2017/\\$FILE/ey-global-blockchain-benchmarking-study-2017.pdf](https://www.ey.com/Publication/vwLUAssets/ey-global-blockchain-benchmarking-study-2017/$FILE/ey-global-blockchain-benchmarking-study-2017.pdf), последњи приступ март 2020.
- [9] *Blockchain Consensus*, <https://devopedia.org/blockchain-consensus>, последњи приступ март 2020.
- [10] *Truffle*, www.trufflesuite.com, последњи приступ јул 2020.
- [11] *Dapp University*, <https://www.dappuniversity.com/>, последњи приступ јул 2020.

ДОДАТАК А

СПИСАК КОРИШЋЕНИХ СКРАЋЕНИЦА

Скраћеница	Значење
DLT	Distributed Ledger Technology – Технологије дистрибуираних главних књига
EVM	Ethereum Virtual Machine – окружење за извршавање паметних уговора на блокчејн мрежи развијено од стране Ethereum-а
NPM	Node Packet Manager - Софтвер за управљање NodeJS пакетима
RPC	Remote Procedure Call – Позив удаљене функције
JSON	JavaScript Object Notation – Формат за слање порука
HTML	HyperText Markup Language – Језик за описивање изгледа веб страница
API	Application Programming Interface – спрега која дефинише начин позивања функција неке апликације са спољашњости, као и формат порука

СПИСАК СЛИКА

Слика 1 Дистрибуирани систем (преузето са docs.oracle.com).....	4
Слика 2 Структурирана прекривна мрежа (преузето са jharaphula.com)	5
Слика 3 Middleware (преузето са www.slideshare.net/Rishikese).....	6
Слика 4 Централизована и дистрибуирана главна књига (преузето са tradeix.com)	7
Слика 5 Однос технологија (преузето са www.eu.com)	8
Слика 6 Структура блокова (преузето са bitcoin.stackexchange.com).....	9
Слика 7 Типови консензус механизма (преузето са devopedia.org)	11
Слика 8 Ganache (преузето са www.trufflesuite.com)	14
Слика 9 Почетни екран апликације	15
Слика 10 Главни екран апликације	15
Слика 11 Truffle (преузето са www.trufflesuite.com)	16
Слика 12 Креирање новог пројекта	17
Слика 13 Структура пројекта	17
Слика 14 MetaMask (преузето са metamask.io).....	18
Слика 15 Одабир блокчејн мреже.....	18
Слика 16 Одабир "налога" тј. јавне адресе	19
Слика 17 Структура пројекта	21
Слика 18 Build директоријум	21
Слика 19 Contracts директоријум	21
Слика 20 Migrations.sol.....	22
Слика 21 Структуре података	23
Слика 22 Структура једних избора	24
Слика 23 Поља паметног уговора	24
Слика 24 Функција за креирање нових избора.....	25
Слика 25 Функција која враћа статус корисника	26
Слика 26 Функција за додавање кандидата	26
Слика 27 Функција за додавање учесника.....	27
Слика 28 Функција за гласање	28
Слика 29 Функција која враћа информације о кандидату	28
Слика 30 Структура фронтенд апликације	29
Слика 31 Поља класе App	29
Слика 32 Освежавање странице.....	30
Слика 33 Иницијализација web3 библиотеке	30
Слика 34 Иницијализација паметног уговора	31
Слика 35 Почетак функције за испртавање	31
Слика 36 Додављање активне јавне адресе	32
Слика 37 Додављање инстанце уговора и статус корисника.....	32
Слика 38 Случај 3	32
Слика 39 Поглед корисника који не учествује у изборима.....	33
Слика 40 Случај 0, када је корисник власник избора	33
Слика 41 Случај 0, поглед власника избора	34

Слика 42 Случај 1, корисник је учесник који није гласао	34
Слика 43 Случај 1, корисник учествује на изборима	35
Слика 44 Случај 2 и случај 1 су малтене исти	35
Слика 45 Функција за гласање	35
Слика 46 Функција за креирање нових избора	36
Слика 47 Функција за додавање новог кандидата	36
Слика 48 Функција за додавање новог учесника	37

БИОГРАФИЈА

Маленчић Никола рођен је 16. новембра 1996. у Новом Саду, Република Србија. Средње образовање је стекао у Гимназији „Јован Јовановић Змај“ у Новом Саду, на природно-математичком смеру. Након тога, 2015. године уписује основне академске студије на Факултету техничких наука у Новом Саду, на смеру Рачунарство и аутоматика. Дипломира на катедри за Рачунарску технику и рачунарске комуникације 2019. године. Исте године уписује мастер академске студије на Факултету техничких наука, смер Рачунарство и аутоматика, модул Електронско пословање.