|  |  |  |
| --- | --- | --- |
|  | УНИВЕРЗИТЕТ У НОВОМ САДУ  **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**  **У НОВОМ САДУ** |  |

Петар Марковић

**Развој апликације студентске службе применом блокчејн технологије**

Дипломски рад

- Основне академске студије -

Нови Сад, 2022

|  |  |  |
| --- | --- | --- |
|  | УНИВЕРЗИТЕТ У НОВОМ САДУ  **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**  21000 НОВИ САД, Трг Доситеја Обрадовића 6 | Датум: |
|  |
| **ЗАДАТАК ЗА ИЗРАДУ ДИПЛОМСКОГ (BACHELOR) РАДА** | Лист: |
| 1/1 |

*(Податке уноси предметни наставник - ментор)*

| Врста студија: | **Основне академске студије** |
| --- | --- |
| Студијски програм: | **Софтверско инжењерство и информационе технологије** |
| Руководилац студијског програма: | **проф. др Мирослав Зарић** |

|  |  |  |  |
| --- | --- | --- | --- |
| Студент: | **Петар Марковић** | Број индекса: | **SW 73/2018** |
| Област: | **Електротехничко и рачунарско инжењерство** | | |
| Ментор: | **др Горан Сладић, редовни професор** | | |
| НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА ДИПЛОМСКИ РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА:   * проблем – тема рада; * начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна; * литература | | | |

**НАСЛОВ ДИПЛОМСКОГ (BACHELOR) РАДА:**

|  |
| --- |
| **Развој апликације студентске службе применом блокчејн технологије** |

**ТЕКСТ ЗАДАТКА:**

|  |
| --- |
| Упознати се са децентрализованим системима са посебним освртом на блокчејн системе. Специфицирати и имплементирати прототип децентрализоване апликације за студентску службу базиране на блокчејн технологији. Документовати решење. |

|  |  |
| --- | --- |
| Руководилац студијског програма: | Ментор рада: |
|  |  |

|  |
| --- |
| Primerak za:  - Studenta;  - Mentora |

# КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

|  |  |
| --- | --- |
| Редни број, **РБР**: |  |
| Идентификациони број, **ИБР**: |  |
| Тип документације, **ТД**: | монографска публикација |
| Тип записа, **ТЗ**: | текстуални штампани документ |
| Врста рада, **ВР**: | дипломски рад |
| Аутор, **АУ**: | Петар Марковић |
| Ментор, **МН**: | др Горан Сладић, редовни професор |
| Наслов рада, **НР**: | Развој апликације студентске службе употребом блокчејн технологије |
| Језик публикације, **ЈП**: | српски |
| Језик извода, **ЈИ**: | српски / енглески |
| Земља публиковања, **ЗП**: | Србија |
| Уже географско подручје, **УГП**: | Војводина |
| Година, **ГО**: | 2022 |
| Издавач, **ИЗ**: | ауторски репринт |
| Место и адреса, **МА**: | Нови Сад, Факултет техничких наука, Трг Доситеја Обрадовића 6 |
| Физички опис рада, **ФО**: | 9 / 51 / 16 / 3 / 17 / 0 / 0 |
| Научна област, **НО**: | Електротехничко и рачунарско инжењерство |
| Научна дисциплина, **НД**: | Безбедност у системима електронског пословања |
| Предметна одредница /  кључне речи, **ПО**: | Блокчејн, Ethereum, паметни уговори |
| **УДК** |  |
| Чува се, **ЧУ**: | Библиотека Факултета техничких наука, Трг Доситеја Обрадовића 6, Нови Сад |
| Важна напомена, **ВН**: |  |
| Извод, **ИЗ**: | У раду је приказана имплементација децентрализоване апликације за студентску службу. За ову сврху, на бекенду је употребљена Hardhat развојна платформа за рад са паметним уговорима у Solidity језику.За фронтенд је коришћена *React* библиотека у комбинацији са *Moralis* платформом за комуникацију са паметним уговором. |
| Датум прихватања теме, **ДП**: |  |
| Датум одбране, **ДО**: |  |
| Чланови комисије, **КО**: |  |
| председник | др Гордана Милосављевић, ред. проф, ФТН Нови Сад |
| члан | др Мирослав Зарић, доцент, ФТН Нови Сад |
| ментор | др Горан Сладић, ванр. проф, ФТН Нови Сад |
| Потпис ментора | |

# KEY WORDS DOCUMENTATION

|  |  |
| --- | --- |
| Accession number, **ANO**: |  |
| Identification number, **INO**: |  |
| Document type, **DT**: | monographic publication |
| Type of record, **TR**: | textual material |
| Contents code, **CC**: | bachelor thesis |
| Author, **AU**: | Petar Marković |
| Mentor, **MN**: | Goran Sladić, regular professor, PhD |
| Title, **TI**: | Development of students service application using blockchain technology |
| Language of text, **LT**: | Serbian |
| Language of abstract, **LA**: | Serbian / English |
| Country of publication, **CP**: | Serbia |
| Locality of publication, **LP**: | Vojvodina |
| Publication year, **PY**: | 2022 |
| Publisher, **PB**: | author’s reprint |
| Publication place, **PP**: | Novi Sad, Faculty of Technical Sciences, Trg Dositeja Obradovića 6 |
| Physical description, **PD**: | 9 / 51 / 16 / 3 / 17 / 0 / 0 |
| Scientific field, **SF**: | Software Engineering and Information Technologies |
| Scientific discipline, **SD**: | E-Business Systems Security |
| Subject / Keywords, **S/KW**: | Blockchain, Etherium, smart contracts |
| **UDC** |  |
| Holding data, **HD**: | Library of the Faculty of Technical Sciences, Trg Dositeja Obradovića 6, Novi Sad |
| Note, **N**: |  |
| Abstract, **AB**: | The paper presents the implementation of a decentralized application for student services. For this purpose, the Hardhat development platform for working with smart contracts in the Solidity language was used on the backend. The React library was used for the frontend in combination with the Moralis platform for communication with the smart contract. |
| Accepted by sci. Board on, **ASB**: |  |
| Defended on, **DE**: |  |
| Defense board, **DB**: |  |
| president | Gordana Milosavljević, PhD, full prof, FTN Novi Sad |
| member | Miroslav Zarić, PhD, assist. prof, FTN Novi Sad |
| mentor | Goran Sladić, PhD, assoc. prof, FTN Novi Sad |
| Mentor's signature | |

**САДРЖАЈ**

[КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА 5](#_Toc113415530)

[KEY WORDS DOCUMENTATION 7](#_Toc113415531)

[1 Увод 9](#_Toc113415532)

[2 Централизовани и децентрализовани систем 11](#_Toc113415533)

[3 Теоријске основе 11](#_Toc113415534)

[3.1 Дистрибуирани системи 12](#_Toc113415535)

[3.2 Блокчејн 12](#_Toc113415536)

[3.3 Консензус алгоритми 13](#_Toc113415537)

[3.3.1 Доказ о раду 14](#_Toc113415538)

[3.3.2 Доказ о улогу 14](#_Toc113415539)

[3.4 Ethereum 15](#_Toc113415540)

[3.5 Светска мрежа и њене итерације 16](#_Toc113415541)

[4 Модел система 17](#_Toc113415542)

[4.1 Класни дијаграм 17](#_Toc113415543)

[4.2 Дијаграм случајева коришћења 19](#_Toc113415544)

[4.3 Дијаграм активности 20](#_Toc113415545)

[5 Имплементација система 21](#_Toc113415546)

[5.1 Иницијална поставка 22](#_Toc113415547)

[5.2 Бекенд део апликације 24](#_Toc113415548)

[5.3 Фронтенд део апликације 37](#_Toc113415549)

[5.4 Финална апликација 40](#_Toc113415550)

[6 Закључак 45](#_Toc113415551)

[7 Литература 47](#_Toc113415552)

[8 Биографија 49](#_Toc113415553)

# Увод

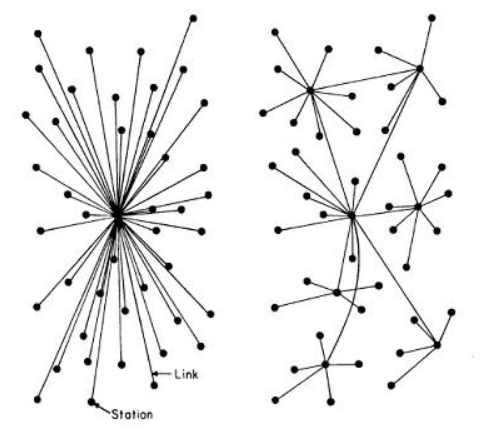
Посматрајући модерне системе, може се приметити да је велики број њих централизоване природе. Сам појам централизованог система се може дефинисати као структура где постоји једно владајуће тело које има потпуну контролу над подацима и функцијама. Дуго времена је оваква структура представљала норму због своје лакоће развоја и приступачности одржавања, међутим тешко је занемарити и њене недостатке, као што су потенцијално лоше перформансе за кориснике који су удаљени од система и склоност грешкама. Како би се ублажиле ове нуспојаве, користе се алтернативе у виду децентрализованих и дистрибуираних система. За разлику од централизованог система који садржи једног централног власника, децентрализовани систем има више власника при чему сваки власник складишти копију ресурса којима корисник има приступ. Дистрибуиран систем, иако сличан децентрализованом по томе што нема једно ауторитативно тело, разликује се по томе што сваки корисник доноси своје одлуке и коначно понашање система представља збир одлука ових корисника.

Иако се сви горепоменути системи могу користити индивидуално, неретко се користе и комбинације истих. Један овакав пример јесте блокчејн технологија. У зависности од имплементације, блокчејн може бити јавни (децентрализовани и дистрибуирани) или приватни (централизовани и дистрибуирани). Ова технологија пружа сасвим нов приступ пословању због чега све већи број грана индустрије пружа приоритет њеном усвајању. Карактеристике блокчејна које су довеле до њене нагле популаризације јесу транспарентност, следљивост, али и приватност, безбедност и непроменљивост података.

У наставку овог рада следи детаљнија спецификација централизованих и децентрализованих система, њихове примене и предности и мане. Наредно поглавље обрађује теоријске основе, тачније, опширно описује дистрибуиране системе, али и саму блокчејн технологију, као и њене битне концепте и елементе. Следеће поглавље приказује сам модел система уз помоћ *Unified Modeling Language (UML)* дијаграма. Затим је представљена сама имплементација система, где je објашњено како је реализовано само решење, које технологије су коришћене приликом развоја, и истакнуте су најбитније компоненте и делови кода. Последње поглавље јесте закључак, где се налази кулминација и резиме целог рада, као и потенцијална унапређења.

# Централизовани и децентрализовани систем

За сваку организацију је приликом развоја система од велике важности избор структуре. Наравно, овај избор зависи од различитог броја фактора, као што су потребе, могућности и ресурси организације, али и потенцијалних корисника система. Сходно томе, избор се наjчешће сведе на 2 опције: централизована или децентрализована структура. Главна разлика између ова 2 приступа јесте у самом нивоу контроле. Код централизованог система, контролу над подацима има само један владајући ентитет (индивидуа, корпорација, сервер итд.), док код децентрализованог система не постоји јединствено владајуће тело, него је контрола једнако расподељена између више независних јединки [1]. На слици 2.1 приказана је илустрација централизованог и децентрализованог система



Слика 2.1. централизован и децентрализован систем [40]

Велика предност централизованих система јесте његова практичност. Пошто је локацијски ограничен на једно место, он се може лако подесити и брзо се развија, такође, цена одржавања је јефтина. Међутим, не могу се занемарити очигледне безбедносне бриге које су настале због чињенице да једно тело складишти и управља подацима корисника. Поврх тога, овакав систем је склон отказу, јер уколико се деси да је централни сервер компромитован, читав систем престаје са радом [2, 3].

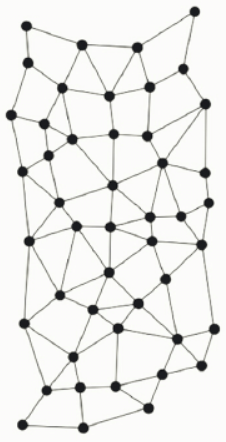
За разлику од централизованих система, децентрализовани системи су отпорнији на грешке због постојања више хардверских компоненти. Ово позитивно утиче и на перформансе, зато што је ове компоненте могуће распоредити географски на местима где је активност корисника висока. Ипак, очигледна мана овог система јесте висока цена одржавања и поставке [3,4].

# Теоријске основе

Узимајући у обзир да блокчејн представља основу овог рада, од великог је значаја разумевање концепта дистрибуираних система, јер је сам блокчејн заправо један вид дистрибуираног система.

## Дистрибуирани системи

Дистрибуирани систем се може дефинисати као колекција независних компоненти које међусобно комуницирају ради постизања заједничких циљева. Као такав, овај систем ће наизглед попримити форму једног интерфејса или јединственог рачунара крајњем кориснику. Сама идеја је таква да се заједничким радом може на максимално ефикасан начин искористити сви ресурси и информације. Такође, саме грешке су ретко скупе, јер уколико дође до отказа на једном уређају, ово не би требало да утиче на доступност сервиса. Још једна значајна особина дистрибуираних система јесте то да су изузетно флексибилни, јер се са лакоћом могу додати компоненте или ојачати већ постојеће [5, 6]. На слици 3.1.1 се може видети илустрација дистрибуираног система.



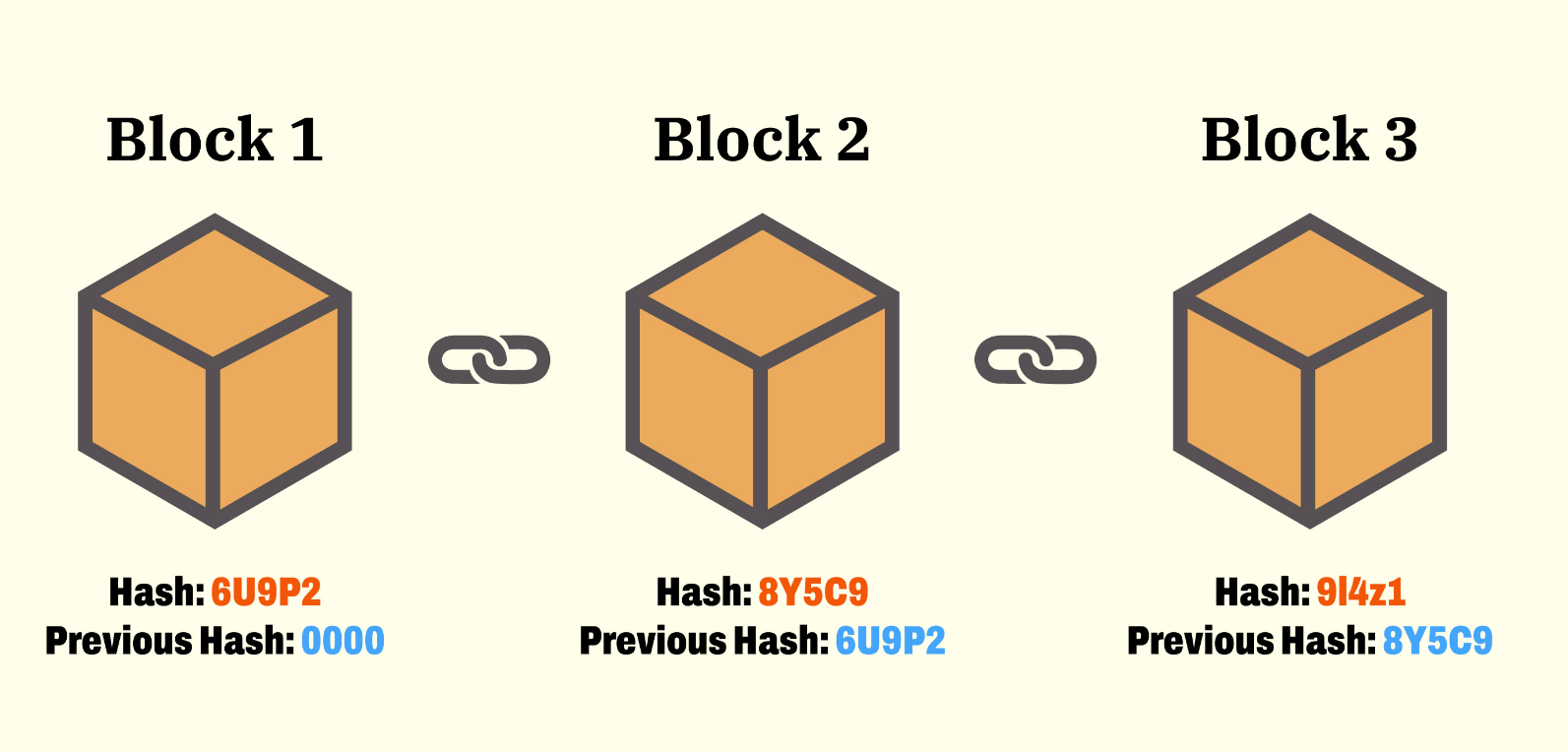
Слика 3.1.1 дистрибуирани систем [40]

Неки од значајних бенефита употребе дистрибуираних система су свакако лакоћа скалирања (како хоризонталног тако и вертикалног), конкурентност, толеранција на грешке и транспарентност. Нажалост, имплементација једног оваквог система долази и са изазовима, као што је повећана комплексност, сложенија поставка безбедности и потешкоће синхронизовања процеса [6].

## Блокчејн

Блокчејн се неретко дефинише као јавно доступна дистрибуирана главна књига, која је подељена између чворова компјутерске мреже [7]. Његова форма је облика ланац блокова који по структури подсећа на једноструко спрегнуту листу. Као технологија је први пут описана 1991. године чија је оригинална идеја била да временски означи дигиталне документе и да самим тим смањи ризик антидатирања или злонамерног манипулисања њиме [8], али је њена прва практична употреба била 2009. у виду платформе познате као *Bitcoin* [9].

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*). Сваки блок може да прими ограничену количину информација, и уколико се превазиђе капацитет блока, формира се нови [10, 11]. Хеш блока представља његов јединствени идентификатор. Сам хеш се рачуна током креирања блока, и он се приликом измене самог блока мења. Хеш претходног блока се складишти како би се формирао ланац блокова и додатно осигурао блокчејн. Наравно, први блок у ланцу може да садржи само свој хеш и скуп података и он се назива иницијални блок (genesis block). На слици 3.2.1 је приказан једноставни пример блокчејна.



слика 3.2.1 блокчејн [41]

## Консензус алгоритми

До популаризације блокчејна је у великој мери дошло због имутабилности, транспарентности, приватности и безбедности коју ова технологија пружа [12]. Иако не постоји један централни ауторитет, свака трансакција се сматра верификованом и обезбеђеном. Ово је могуће због употребе консензус алгоритама.

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. Сваки алгоритам тежи ка више циљева, као што су постизање заједничког споразума, пружање економског подстицаја, омогућавање поштеног и равноправног учешћа и максимизација отпорности на грешке [13, 14, 15]. Иако тренутно постоји велики број алгоритама, сваки са својим случајем коришћења, рад ће се фокусирати на 2 најпознатија консензус алгоритма: доказ о раду (*Proof of Work – PoW)* и доказ о улогу (*Proof of Stake – PoS)*.

### Доказ о раду

Доказ о раду је најстарији консензус механизам коришћен у блокчејн технологији и тренутно га користе *Bitcoin* и *Ethereum [17]*. Идеја иза овог механизма јесте да се чворови учесници, боље познати као рудари, међусобно такмиче у решавању комплексних математичких проблема употребом моћних рачунара. Рудар који први реши проблем и нађе одговарајући хеш стиче право да формира нови блок и потврди трансакције. Он је такође награђен са унапред дефинисаном количином криптовалута [15, 16].

Иако се генерално сматра најпоузданијим од свих механизама консензуса, свакако постоје потенцијални проблеми о којима се мора водити рачуна. Један од њих је могућност да централни ентитет поседује 51% или више чворова у мрежи и самим тим корумпира блокчејн тако што ће добити већину мреже. Ово је познато и као ризик 51%. Такође, ово захтева велику количину компјутерске моћи што не само што је изузетно скупо, него толика електрична потрошња може имати изузетно лош утицај на животну средину [14].

### Доказ о улогу

Као алтернатива *PoW* се најчешће користи *PoS*. Разлог популаризације овог консензус алгоритма јесте што не захтева скупе хардверске компоненте за решавање комплексних математичких загонетки. Учесници овог механизма се називају валидатори и они стичу право да потврде блокове који би могли да се додају на блокчејн. Међутим, да би се стекао статус валидатора, неопходно је приложити одређену количину криптовалута као улог. У зависности од конкретних блокова који се додају на блокчејн. Валидација блокова се врши тако што се валидатори кладе на откривене блокове који би се потенцијално могли додати у блокчејн. У зависности од додатих блокова, сви валидатори добијају награду у складу са њиховом уложеном опкладом и вредност њиховог улога складно расте. На крају *PoS* за генерисање блока се бира валидатор чија је вредност улога највећа на мрежи [14, 16].

Упркос његовој ефикасности и одрживости, овај приступ је наишао на критику због могућности да доведе до централизације ланца. Разлог томе је што валидатори са већом количином новца могу стећи већу контролу над системом, и на тај начин нарушити децентрализовану природу система [15].

## Ethereum

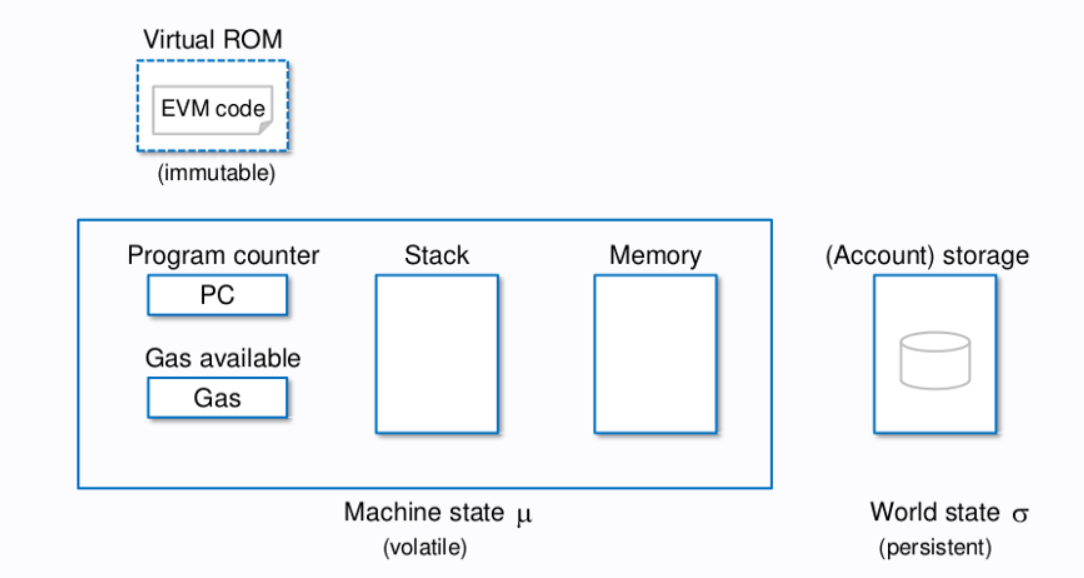
На самом почетку поглавља је споменут *Bitcoin* као прва апликативна примена блокчејна. Иако револуционарна технологија, сам *Bitcoin* је по дизајну туринг непотпун [18]. Другим речима, он не може да покрене комплексне програме или да реши сложене задатке. Међутим, *Bitcoin* је намерно имплементиран на овај начин, како би се избегло коришћење превише ресурса за чворове и на тај начин повећала једноставност, али и безбедност платформе. Иако је овакав дизајн био погодан за потребе *Bitcoin-a*, свакако је идеја туринг потпуне блокчејн платформе била примамљива. Један пример такве платформе јесте *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*) [35], као и програмски језик *Solidity [34]*.

Велика предност *Ethereum-a* јесте његова туринг потпуна природа. Баш због те карактеристике је постала јако погодна платформа за програмере који желе да развију своју децентрализовану апликацију. Развој ових децентрализованих апликација је омогућен захваљујући алату који је познат као паметни уговор.

Најједноставније дефинисано, паметни уговор је обичан програм 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 интерагују са паметним уговором, али ниједан од њих нема контролу над њим, чак ни учесник који је поставио паметни уговор на блокчејн. Ове интеракције се постижу подношењем трансакција које затим извршавају функције дефинисане у паметном уговору. Свака трансакција захтева и одређену количину гаса. Гас је вид додатне таксе неопходне за извршавање кода у паметном уговору. Количина гаса у великој мери зависи од тренутне понуде и потражње за рачунарском снагом која је потребна за обраду паметних уговора и трансакција. Паметни уговори дозвољавају обављање трансакција између анонимних страна без потребе за централним органом. Свака трансакција извршена у оквиру паметног уговора је следљива, транспарентна и иреверзибилна [20, 21, 22].

Темељ читаве оперативне структуре *Ethereum-*a је *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 [23, 24, 25]. На слици 3.4.1 се може видети архитектура *EVM*.



Слика 3.4.1 архитектура Етеријумове виртуелне машине [42]

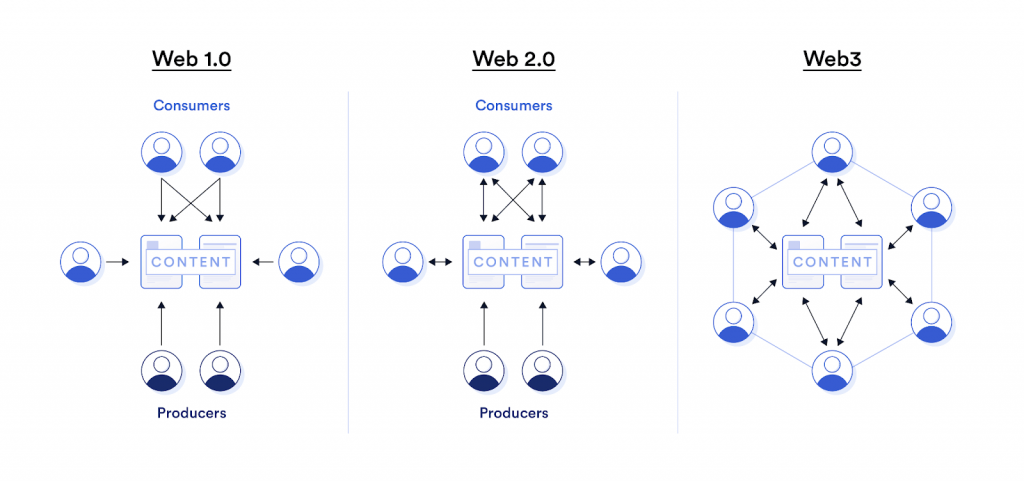
## Светска мрежа и њене итерације

Од свог оснивања, светска мрежа је изузетно напредовала и данас се сматра да постоји више верзија веба. Тренутан став је тај да је веб прошао кроз 2 итерације, а да ће основу наредне итерације чинити блокчејн и паметни уговори.

Прва итерација је позната као *Web 1.0*. То је мрежа отвореног кода без пермисија која је пружала стриктно статички садржај. Ова верзија веба је дизајнирана тако да помогне корисницима да боље пронађу информације. Међутим, ова иницијална фаза мреже се сматра доста примитивном јер му недостају битни динамички елементи као што су форме, визуелни елементи и интерактивност [26].

Наредна етапа је позната као *Web 2.0*. *Web 2.0* описује тренутно стање светске мреже, које пружа динамички садржај и бољу употребљивост за крајње кориснике у поређењу са *Web 1.0*. Ова нова верзија веба је омогућила виши ниво размене информација, али и повећану међусобну повезаност између корисника [26, 27].

Иако се позитиван утицај *Web 2.0* не може занемарити, ова етапа је имала и мане. Порастом учесника веба, Одређен број врхунских компанија је остварило монопол над интернетом и тако су стекли велику моћ у одлучивању шта је дозвољено, а шта није. Узимајући у обзир ову чињеницу, као и нагли развој блокчејна, полако је настала идеја о потенцијално новој итерацији мреже која би се звала *Web 3.0*. Концепт је формирање потпуно децентрализоване мреже без пермисија са динамичким садржајем и економијом која је базирана на криптовалутама и токенима. Велики бенефит који *Web 3.0* пружа је приватност података уз помоћ уграђене псеудонимности, али и поновна дистрибуција власништва над интернетом [26, 27, 28]. На слици 3.5.1 се налази илустрација горепоменутих итерација веба.



Слика 3.5.1 итерације веба [43]

# Модел система

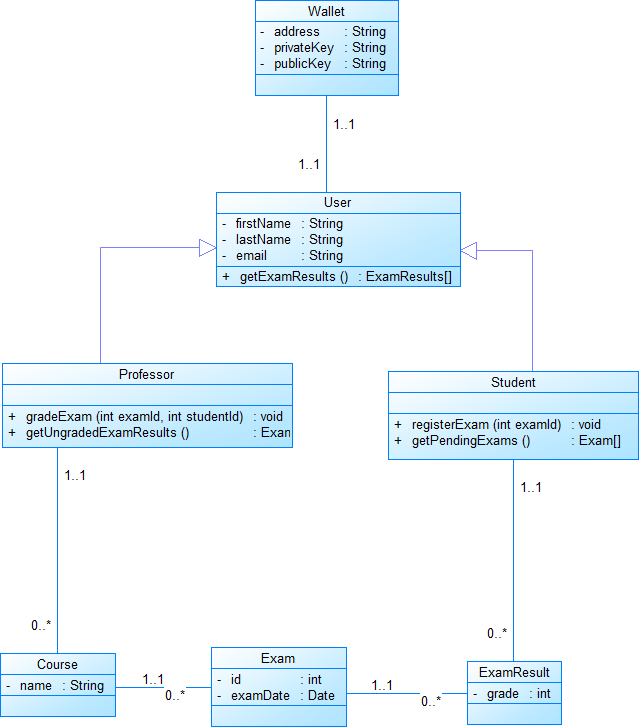
Како би се најбоље разумео концепт и начин пословања децентрализоване апликације студентске службе, неопходно је прво описати њен модел. За ову сврху, апликацију можемо представити путем дијаграма који репрезентују систем на различитим нивоима апстракције. У наставку поглавља биће приказана 3 дијаграма: класни дијаграм, дијаграм случајева коришћена и дијаграм активности.

## Класни дијаграм

Класни дијаграм је тип дијаграма статичке структуре коришћен зарад описа система преко класа, њених атрибута и метода, али и односа између објеката.

На слици 4.1.1 приказан је класни дијаграм. Класе у овом дијаграму су:

* *Wallet* (енгл. новчаник) – новчаник складишти криптовалуте корисника и преко њега се обављају све трансакције. Први атрибут *Wallet*-a је адреса. Адреса новчаника је хексадецимална вредност изведена из јавног кључа новчаника и представља јавни идентификатор сваког новчаника преко којег могу да се уплате средства на налог. Поред овога сваки новчаник садржи и 2 кључа: приватни кључ и јавни кључ. Јавни кључ се користи како би могле да се пошаљу криптовалуте новчанику. Са друге стране, улога приватног кључа је налик шифри и користи се зарад верификације трансакција и доказивања власништва над адресом. Последњи атрибут је салдо и то је децимални број који представља количину *ETH* на новчанику.
* *Course* (енгл. курс) – као атрибуте има назив курса и конкретног предавача.
* *Exam* (енгл. Испит) – као атрибуте има шифру испита, датум одржавања испита и асоцирани предмет.
* *ЕxamResult* (енгл. резултат испита) – као атрибуте има студента који је полагао предмет, оцену и конкретан испит.
* *User* (енгл. корисник) – као атрибуте садржи новчаник, име и презиме и имејл адресу. Такође, корисник има методу преко које може да прегледа резултате испита. наслеђују је класе:
  + *Student* (енгл. студент) – не поседује додатне атрибуте, али има операцију преко које може да пријави испит и добије преглед непријављених испита.
  + *Professor* (енгл. професор) – професор као атрибут још садржи и низ курсева на којима предаје. Такође, професор има методу за добављање неоцењених испита и оцењивање испита.



Слика 4.1.1 дијаграм класе

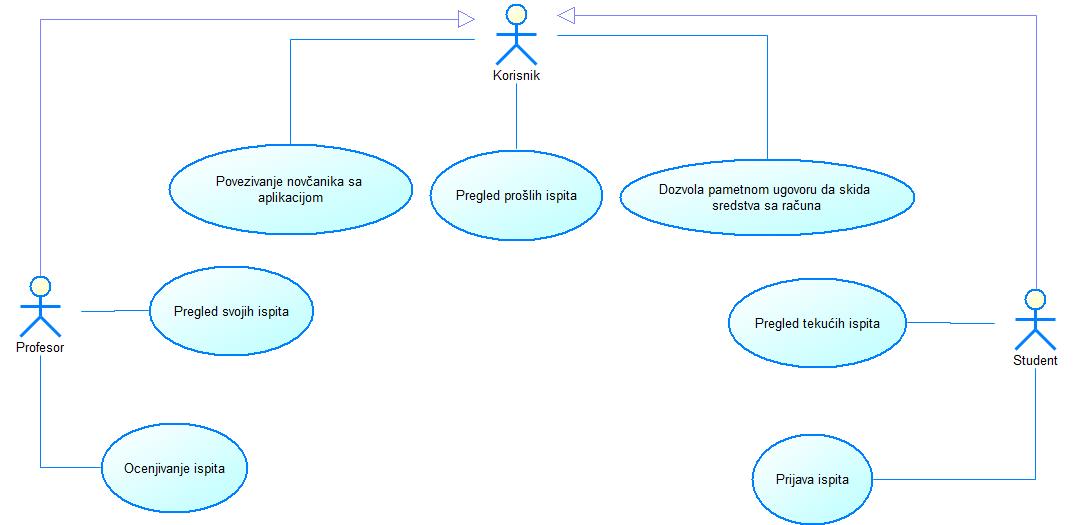
## Дијаграм случајева коришћења

На слици 4.2.1 приказан је дијаграм случајева коришћења. Улога дијаграма случајева коришћења јесте да демонстрира различите начине на које учесници или актери система могу интераговати са системом. Идеја јесте да се направи апстрактна депикција релација измећу корисника, случајева коришћења и система.

У случају описане апликације, постоје 2 врсте актера: студент и професор. Заједничко за обе врсте корисника је то што могу да повежу свој новчаник са апликацијом, као и да дозволе паметном уговору да скида средства са рачуна приликом извршавања трансакција. Осим тога они могу да прегледају резултате прошлих испита: у случају студента он има резултате свих испита на које је изашао, док професор има приказ свих испита које је оценио. Ово су једине заједничке функционалности за обе роле.

Поред прегледа оцењених испита, професор такође има могућност да види списак свих пријављених испита које он држи, а који још увек нису оцењени. Такође, професор може да оцени неки од испита.

Са друге стране, студент има преглед свих текућих непријављених испита за које испуњава услове за пријаву. Oсим тога, уколико студент задовољава услове, он може да пријави испит студентској служби.



Слика 4.2.1 дијаграм случајева коришћења

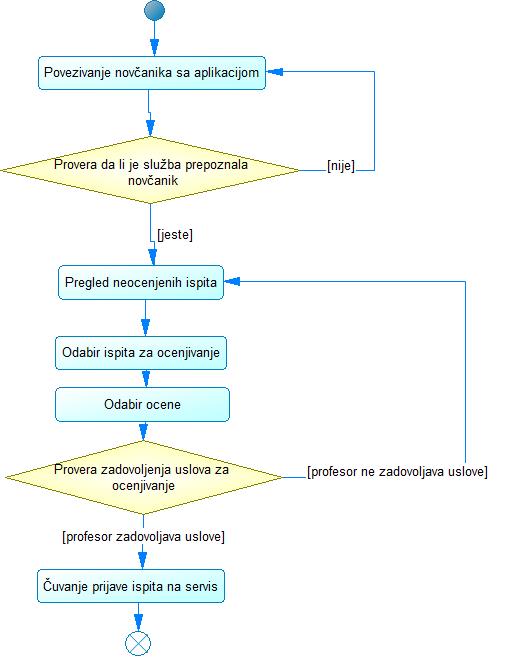
## Дијаграм активности

Дијаграм активности се користи ради описа како су активности у оквиру једног система координисане. Сваки догађај је сачињен од скупа операција где су наведене операције најчешће корелиране или у јакој релацији и самим тим наредна операција може у већој или мањој мери зависити од претходне.

Ток пријаве испита изгледа овако:

1. студент се пријављује на службу повезивањем свог новчаника.
2. уколико је служба препознала новчаник студента, њему се приказује преглед свих испита за које је испунио услове полагања.
3. студент бира један од испита који жели да пријави.
4. Пре него што прође пријава на систем, неопходно је проверити да ли студент задовољава све критеријуме прво (да ли има довољно средства на рачуну, да ли је испит пријављен више од 3 дана пре почетка одржавања итд.).
5. Уколико су ови критеријуми задовољени, пријава ће бити успешно сачувана.

На слици 4.2.1 приказан је дијаграм активности за студента.

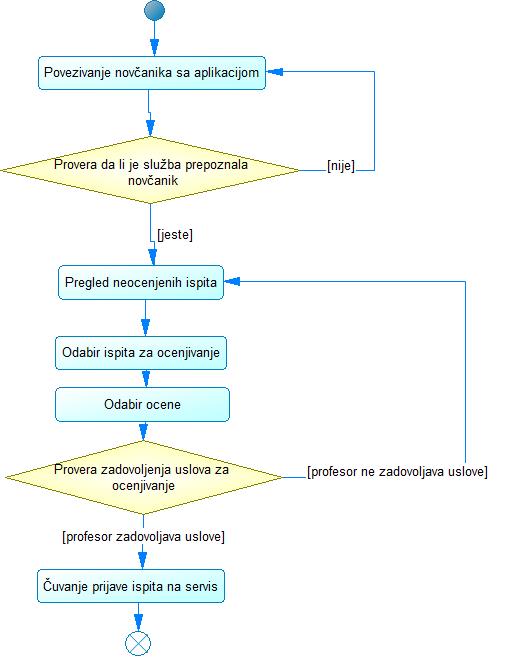


Слика 4.2.1 дијаграм активности за студента

Ток оцењивања испита изгледа овако:

1. професор се пријављује на службу повезивањем свог новчаника.
2. уколико је служба препознала новчаник професора, њему се приказује преглед свих неоцењених испита које он држи.
3. Након одабира испита и оцене, уколико има довољно средства на рачуну да плати гас, испит се архивира на служби.

На слици 4.2.2 приказан је дијаграм активности за професора.



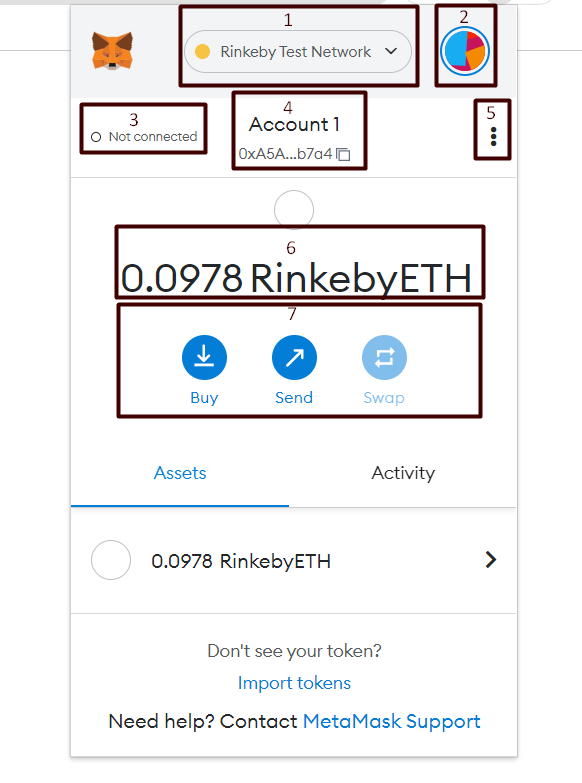
Слика 4.2.2 дијаграм активности за професора

# Имплементација система

Наредно поглавље ће бити фокусирано на саму имплементацију система, процес развоја и тестирања, као и финално решење. Ради погодности, ово поглавље ће бити подељено на 3 сегмента. Први део ће се фокусирати на неке основне концепте и компоненте које је неопходно разумети и подесити пре развоја и употребе програма. Други део ће се фокусирати на бекенд аспект решења, тачније на писање и поставку паметног уговора, док ће последњи део анализирати фронтенд апликације, конкретније самог клијента, али и његову комуникацију са паметним уговором.

## Иницијална поставка

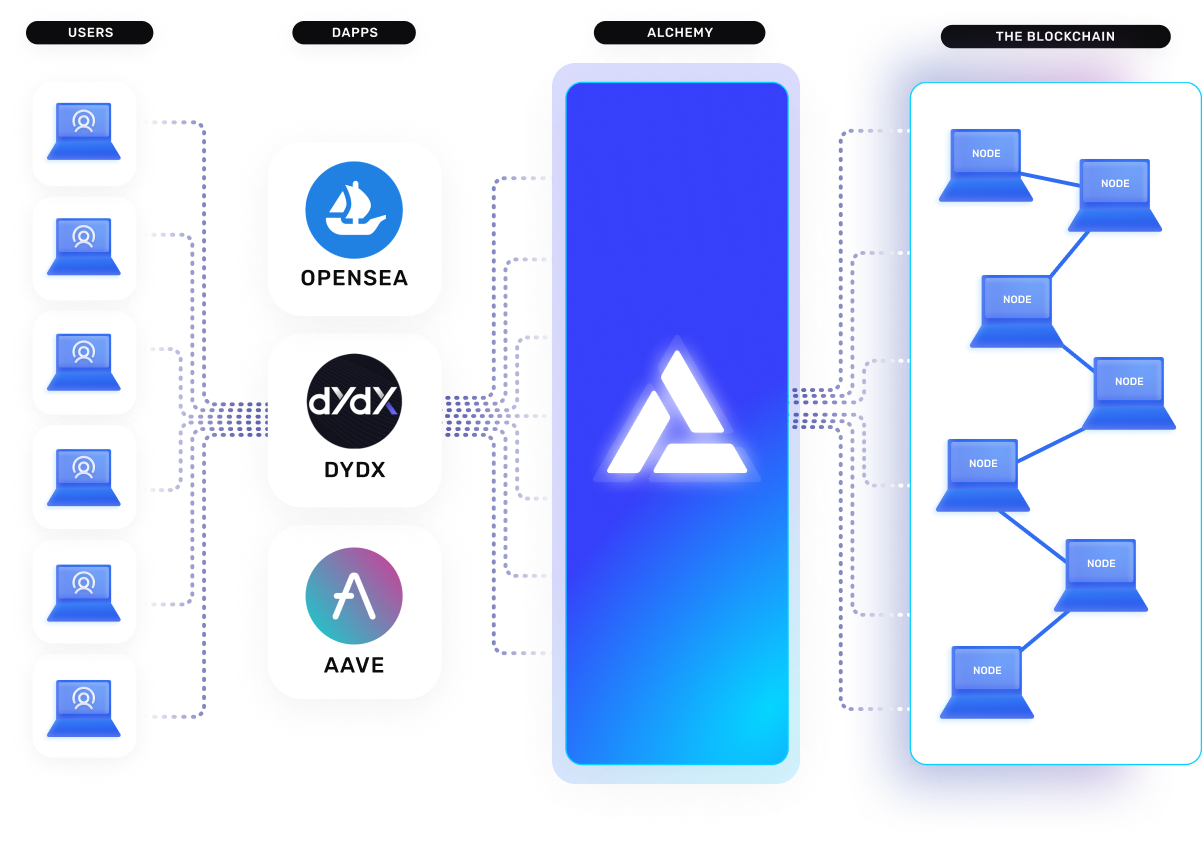
Како би уопште могла да се постигне комуникација са децентрализованим апликацијама, али и могућност развоја истих, прво је неопходно да се подеси лични новчаник криптовалута. Сврха ових новчаника јесте да омогући корисницима да шаљу и примају криптовалуте, али и да извршавају трансакције над паметним уговорима. Постоји велики број ових новчаника и сваки од њих има своје предности и мане, међутим за сврху овог рада коришћен је *MetaMask* [29]. Овај новчаник је изабран из простог разлога што је он *Ethereum-*ов новчаник, а коначно решење овог рада је развијено на *Ethereum* платформи. Сама инсталација *MetaMask*-a је једноставна и интуитивна, која захтева само инсталирање екстензије претраживача и постављање шифре.



Слика 5.1.1 *MetaMask* новчаник

На слици 5.1.1 је приказан интерфејс *MetaMask* новчаника. Под бројем 1 је означен падајући мени уз помоћ којег се може одабрати блокчејн мрежа на коју корисник жели да се повеже. Подразумевано се као опције налазе *Ethereum* главна мрежа, али и тестне мреже *Ropsten*, *Kovan*, *Rinkeby* и *Goerli*, а корисник по потреби може додати више мрежа [30]. Ради олакшаног развоја и како програмери не би користили права средства приликом развоја апликације, неретко се користе неке од претходно наведених тестних мрежа. За сврху овог пројекта је коришћена *Rinkeby* мрежа. Дугме са бројем 2 омогућава кориснику рад са налозима. Након што кликне ово дугме, корисник може да се пребаци на други налог, али такође може да креира или импортује нов налог. Број 3 представља индикатор који показује да ли је новчаник повезан са апликацијом или страницом. Број 4 представља назив и адресу тренутног налога. Број пет је падајући мени који пружа кориснику функционалности као могућност да прошири приказ интерфејса, погледа детаље налога и добије приказ свих повезаних страница. Број 6 представља конкретну валуту за изабрану мрежу као и колику количину те валуте корисник има у свом новчанику.

Након поставке новчаника, наредни корак је креирање налога на *Alchemy-у [31]*. *Alchemy* представља моћну блокчејн платформу за програмере која пружа низ алата за развој. Ова платформа замењује стандардне локалне чворове са чворовима који имају бржу, скалабилнију и децентрализованију архитектуру. Поред овога, он пружа алате за мониторинг, аналитику, евидентирање и дебаговање грешака за блокчејн базиран софтвер. За потребе овог пројекта, креиран је *Alchemy* пројекат за *Ethereum* *Rinkeby* тест мрежу. Након креирањa апликације, неопходно је извући *Application Programming Interface* (*API*) кључ како би се успешно аутентификовао корисников чвор и омогућило слање захтева ка блокчејну. На слици 5.1.2 је приказана илустрација *Alchemy* платформе.



Слика 5.1.2 *Alchemy [31]*

Последњи важни предуслов пре почетка развоја апликације јесте упознавање са *Etherscan* платформом [32]. *Etherscan* се дефинише као блокчејн претраживач за *Ethereum*. Преко ове платформе могуће је пронаћи важне податке сачуване на ланцу, као што су трансакције, блокови, као и адресе новчаника и паметних уговора. Уз помоћ овог алата корисник може боље разумети како се постиже интеракција са компонентама блокчејна. Поред ових функционалности, *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.

## Бекенд део апликације

За потребе развоја бекенд стране рада, коришћен је радни оквир *Hardhat [33]*. *Hardhat* је окружење за тестирање, компајлирање, постављање и дебаговање децентрализованих апликација засновано на *Ethereum* блокчејну. Захваљујући *Hardhat-у*, програмери могу да организују задатке неопходне за развој децентрализованих апликација и паметних уговора. Поред тога, *Hardhat* омогућује аутоматизацију одређених задатака и пружа девелоперима мноштво корисних функција. Такође, велика предност овог радног оквира јесте њена уграђена локална *Ethereum* мрежа која је фокусирана на дебаговање *Solidity* кода, тако што пружа поруке грешке приликом компајлирања, извршавања кода, постављања итд.

Како би се инкорпорирао *Hardhat* у решење, прво је неопходно креирати пројекат. За ову потребу се може искористити било који менаџер пројекта или пакета, али је за рад коришћен *Yarn [36]*. У празан директоријум се прво унесе команда приказана на слици 5.2.1.



Слика 5.2.1 иницијализација *Yarn-а*

Након покретање ове команде, у празном директоријуму је креирана *package.json* датотека. Овај фајл је задужен за складиштење метаподатака пројекта, али се овде налазе и зависности, скрипте, верзија и друге важне информације о самом пројекту. Да би укључили *Hardhat* у пројекат неопходно је унети команду као на слици 5.2.2.



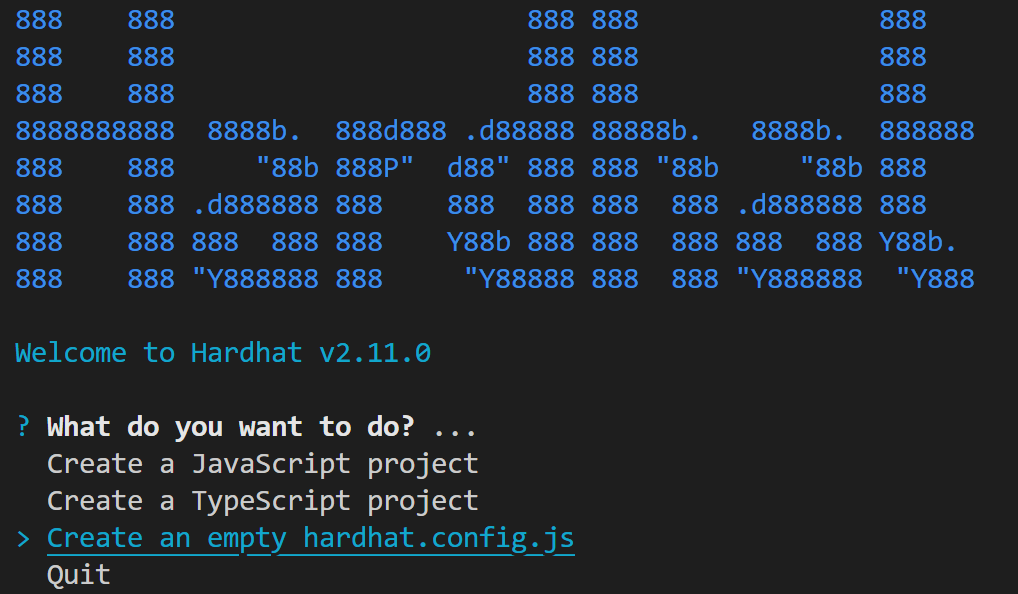
Слика 5.2.2 поставка *Hardhat-a* у пројекат

Покретањем ове команде, као зависност је сачуван *Hardhat*. Да бисмо започели рад са овим окружењем, потребно је терминалу проследити унос као на слици 5.2.3.



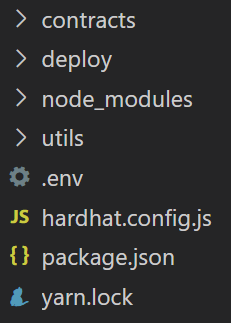
Слика 5.2.3 иницијализација *Hardhat-a*

Након ове команде, у оквиру терминала се приказује *Hardhat-ова* форма. Ова форма нуди кориснику наредне опције: креирање *JavaScript* пројекта, креирање *TypeScript* пројекта, креирање празног пројекта који садржи само *hardhat.config.jsоn* конфигурациони фајл или одустанак од креирања пројекта. За потребе пројекта изабрана је опција генерисања празног пројекта. Изглед форме је представљен на слици 5.2.4.



Слика 5.2.4 *Hardhat-a* форма за генерисање пројекта

Након свих горе наведених корака, *Hardhat* окружење ће бити успешно имплементирано у оквиру пројекта. Следећи поступак јесте организација пројекта. У оквиру слике 5.2.5 може се видети хијерархија фолдера и фајлова.

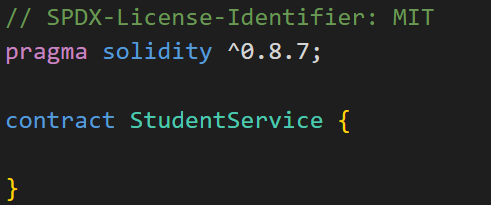


Слика 5.2.5 хијерархија пројекта

Како би се боље разумео пројекат, неопходно је објаснити кључне директоријуме и датотеке из претходне слике:

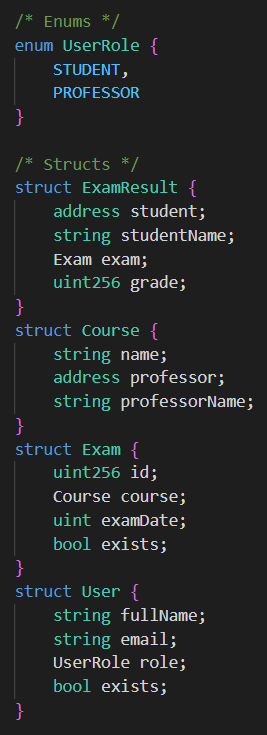
* *contracts -* Садржи паметни уговор написан у *Solidity* језику.
* *deploy –* Садржи две скрипте: једну везано за постављање паметног уговора на блокчејн и другу за прослеђивање важне информације о самом паметном уговору клијенту.
* *utils –* Садржи помоћну класу за верификацију паметног уговора.
* *.*env – Датотека која складишти осетљиве променљиве средине. Овде се чува приватни кључ корисника који поставља паметни уговор на мрежу, али и његови *API* кључеви креирани на *Alchemy* и *Etherscan* платформи.

У оквиру *contracts* директоријума налази се датотека *StudentService.sol.* Ова датотека представља основу самог бекенда јер је у оквиру ње написан паметни уговор у *Solidity* језику. *Solidity* је објектно оријентисан програмски језик високог нивоа који се користи за имплементацију паметних уговора. Он је статички типизиран језик витичастих заграда и подржава концепте као што су наслеђивање, употреба библиотека и комплексних типова. Основа сваког *Solidity* програма јесте паметни уговор. У оквиру *Solidity-а*, паметни уговор се дефинише кључном речју *contract*, након које иде назив уговора и пар витичастих заграда. Овај уговор је концептуално веома сличан класама у објектно оријентисаним језицима, јер у оквиру уговора се налазе компоненте и концепти као што су методе, атрибути, модификатори приступа, конструктор, наслеђивање итд. Међутим, пре дефинисања уговора, неопходно је резервисати 2 линије кода за посебне сврхе. У првој линији кода се специфицира лиценца коју уговор користи. У другој линији кода је описана верзија компајлера коришћена у оквиру фајла. На слици 5.2.6 је приказан иницијални изглед *StudentService.sol* фајла.



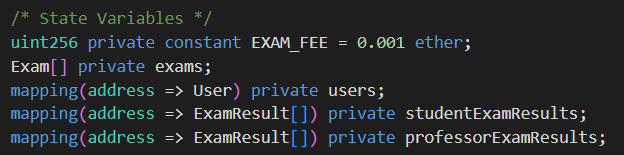
Слика 5.2.6 *StudentService.sol*

У оквиру витичастих заграда ће бити написан сав програмски код. Прва ствар која ће се дефинисати је модел апликације. Модел у овом уговору је конзистентан са класним дијаграмом описаним у претходном поглављу и имплементиран је употребом структура. Структуре у *Solidity* језику су у великој мери сличне структурама у другим програмским језицима. То су комплексни типови података који садрже више једноставних и комплексних атрибута. Структуре су дефинисане кључном речју *struct*. Атрибути се дефинишу у структи тако што се наводи прво тип, а након њега назив атрибута. Пре структура је дефинисана и једна енумерација која одређује улогу корисника и садржи две вредности: студент и професор. На слици 5.2.7 је приказана енумерација и све структуре.



Слика 5.2.7 модел паметног уговора

Након модела, наредни блок кода представља скуп променљивих. Конкретно, ове променљиве се користе ради чувања стања паметног уговора и играју сличну улогу као база података, јер складиште важне информације. Када се варијабла стања дефинише, прво се наводи тип података варијабле, након чега иде модификатор приступа и на крају назив варијабле. Пре назива променљиве, може се додати модификатор *constant* или *immutable.* Просто речено, први модификатор сигнализира да се варијабли додељује вредност само приликом њене дефиниције, док други модификатор омогућава да се вредност променљиве дефинише једном и само једном у оквиру паметног уговора. На слици 5.2.8 су приказане све варијабле стања.

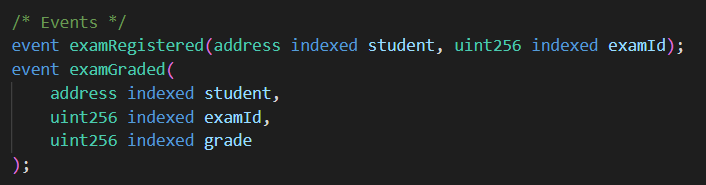


Слика 5.2.8 варијабле стања

Свака варијабла на слици 5.2.8 има приватни модификатор приступа и самим тим видљива је само у оквиру уговора. Прва варијабла је константа типа неозначени број од 256 бита и он представља цену пријаве једног испита. Наредна варијабла представља низ свих испита. Како би се разумеле последње три променљиве, важно је разумети *mapping* тип. По структури *mapping* је налик хеш табели и користи се за складиштење парова кључ-вредност. Прва од ове три променљиве представља скуп свих корисника у систему. Кључ је адреса новчаника корисника, а вредност садржи податке о кориснику са том адресом. Последње две варијабле складиште информације о резултатима испита. Тачније, прва варијабла као кључ чува адресу новчаника студента, а као вредност чува низ свих испита на које је изашао. Кључ друге варијабле је адреса професора, а вредност представља низ испита које је оценио.

У наставку програма налазе се догађаји или *event* компоненте. 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. Ови параметри могу садржати кључну реч *indexed* која олакшава претрагу догађаја преко тих параметара. *На* слици 5.2.8 налазе се 2 догађаја:

* *examRegistered* – догађај који се емитује након регистрације испита. Као параметре прима адресу студента и шифру испита.
* *examGraded* – догађај који се емитује након оцењивања испита. Као параметре прима адресу студента, шифру испита и уписану оцену.



Слика 5.2.8 догађаји

Следећи сегмент уговора чине ставке које су карактеристичне за *Solidity* и називају се модификатори. Улога модификаторе јесте да промени понашање функције на коју се примени. Генерално се користи за проверу испуњења услова пре или после извршења функције. На овај начин смањује се редундантност кода и омогућује поновна употреба у различитим деловима кода. Сваки модификатор се дефинише кључном речју *modifier* која је праћена називом модификатора и његовим телом где се дефинише код. За потребе пројекта, ови модификатори се користе за руковање са изузецима.



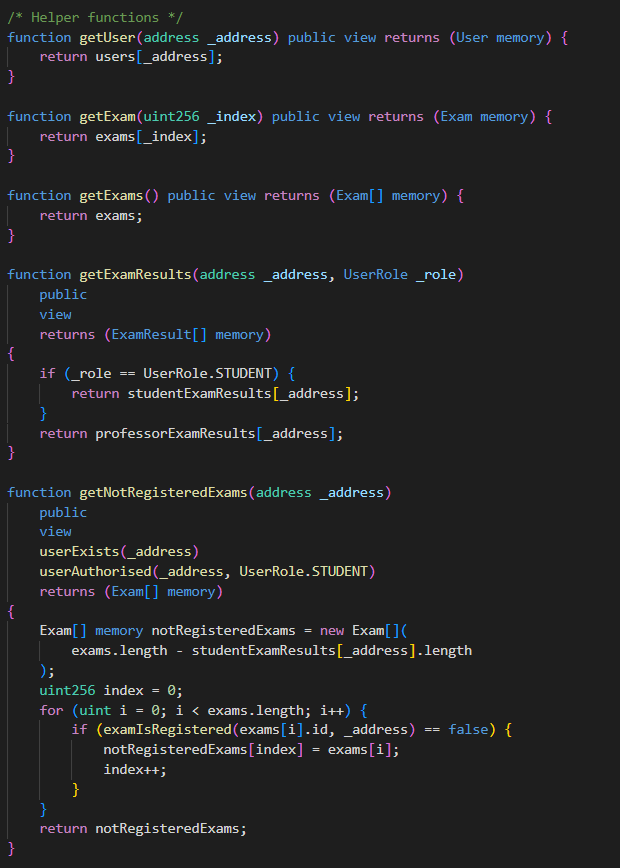
Слика 5.2.9 модификатори

Сваки модификатор на крају тела садржи симбол \_;. Ово само означава да желимо да се програмски код модификатора изврши пре функције. Улога модификатора на слици 5.2.9 је:

* *userExists* – проверава да ли корисник са датом адресом постоји.
* *examExists* – проверава да ли испит са датом шифром постоји.
* *userAuthorised* – проверава да ли је корисник ауторизован за извршавање функције.
* *sufficientETH* – Проверава да ли је прослеђено довољно средства за пријаву испита.
* *examRegisteredLate* – Проверава да ли је испит пријављен мање од 2 дана пре одржавања.
* *examHistoryExists –* проверава да ли је студент са датом адресом већ пријавио испит са датом шифром.

Као и други језици са објектно оријентисаном парадигмом, и *Solidity* садржи конструкторе. У оквиру ове посебне функције може се писати код и ова функција ће се извршити само једном након креирања уговора. За потребе пројекта, конструктор се користио како би се поставиле иницијалне вредности у оквиру горе наведених варијабли стања.

Последњи део уговора чини скуп функција. Функције у *Solidity* језику су описане са кључном речју *function* након чега иде назив функције и низ параметара. Поред овога, може се дефинисати модификатор приступа функције, повратна вредност функције, али и низ *modifier-a*. Ради прегледности, функције су подељене у 2 групе: помоћне и главне функције. На слици 5.2.10 се налазе помоћне функције за добављање података.

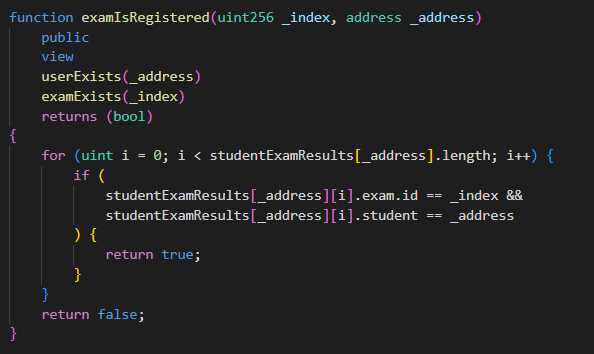


Слика 5.2.10 помоћне функције за добављање података

Специфично за ове тзв. *getter* функције јесте атрибут *view* дефинисан након кључне речи *public.* Овај атрибут се користи уколико функција не мења стање уговора, него се користи само за читање вредности варијабли стања. Самим тим, пошто променљиве стања остају нетакнуте, позивање ових функција неће захтевати гас. Такође, у оквиру повратне вредности функције, поред типа, налази се и кључна реч *memory.* Ова реч указује да се та вредност складишти само привремено и након извршавања се брише. Улога функција са слике 5.2.10 је:

* *getUser* – добавља корисника са датом адресом.
* *getExam* – добавља испит са датом адресом.
* *getExamResults* – добавља резултате испита за корисника са датом адресом и улогом.
* *getNotRegisteredExams* – добавља све испите које студент са датом адресом није пријавио.

Поред ових функција, налази се још једна помоћна функција чија улога јесте да провери да ли је корисник са одређеном адресом пријавио испит са датом шифром. Програмски код ове функције се налази на слици 5.2.11.



Слика 5.2.11 помоћна функција за проверу пријаве испита

У оквиру паметног уговора налазе се 2 главне функције:

* *registerExam* – функција за пријаву испита. Уколико су прошле све провере *modifier-a,* проналази се вредност *studentExamResults* за студента који је започео пријаву и вредност *professorExamResults* за професора који држи пријављени испит. Овим вредностима се додаје *ExamResult* објекат са оценом 0 што указује на неоцењени испит. На крају функције емитује се догађај *examRegistered.*
* *gradeExam* – функција за оцењивање испита. Уколико су прошле све провере *modifier*-a, проналази се испит са прослеђеном шифром и адресом студента који је пријавио испит. Након што професор упише оцену, ажурира се резултат тог испита у *studentExamResults* за студента који је пријавио испит као и резултат испита у *professorExamResults* за професора који је оценио испит. На крају функције емитује се догађај *examGraded.*

Ове 2 функције су приказане на слици 5.2.12.



Слика 5.2.12 главне функције

Након што је написан паметни уговор, неопходно је подесити конфигурацију за постављање истог на блокчејн мрежу. Сва нужна конфигурација се налази у *hardhat.config.js* датотеци. Податке које је неопходно дефинисати овде су:

* *defaultNetwork* – Подразумевана мрежа за постављање уговора.
* *networks* – низ мрежа на које се може поставити уговор, свака мрежа дефинисана у овој ставци мора садржати:
  + *chainId* – идентификатор ланца који се користи за валидацију повезане мреже.
  + *url –* линк ка коришћеном чвору.
  + *accounts* – скуп налога који *Hardhat* користи.
* *etherscan* – подаци о *Etherscan-у*, мора садржати бар атрибут:
  + *apiKey* – *API* кључ *Etherscan* налога.
* *solidity* – верзија *Solidity* компајлера.
* *namedAccounts* – омогућава асоцирање имена са адресама и конфигурисање по ланцу користи се најчешће за дефинисање деплојера и учесника.

На слици 5.2.13 налази се подешен конфигурациони фајл за сврху решења.



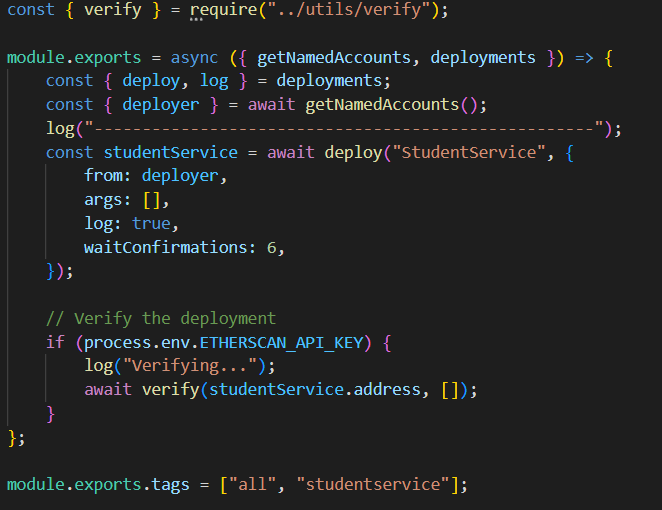
Слика 5.2.13 *hardhat.config.js*

Променљиве *ALCHEMY\_API\_KEY, ETHERSCAN\_API\_KEY* и *PRIVATE\_KEY* су извучене из *.env* датотеке.

Након подешавања конфигурационог фајла, могу се написати скрипте за постављање уговора у оквиру *deploy* директоријума. Срећом, *Hardhat* нуди функцију за постављање паметног уговора на мрежу која такође користи битне информације дефинисане у конфигурационом фајлу, програмер једино мора да проследи 2 аргумента:

* назив паметног уговора.
* *објекат* који садржи следеће атрибуте:
  + *from -*  налог који је задужен за постављање уговора.
  + *args –* аргументи паметног уговора.
  + *log –* да ли корисник жели испис логова приликом постављања.
  + *waitConfirmations –* број потврда које треба сачекати након које се паметни уговор укључи у ланац.

Целокупна логика за постављање уговора налази се у *deploy-student-service.js* датотеци. Садржај ове датотеке приказан је на слици 5.2.14.



Слика 5.2.14 *deploy-student-service.js*

Након постављања уговора, врши се и верификација уговора на *Etherscan-у*. Направљена је *verify.js* скрипта у *utils* директоријуму. У оквиру ове скрипте користи се *Hardhat* уграђена функција за верификацију уговора на коју је придодато руковање изузецима. Ова скрипта је приказана на слици 5.2.15.



Слика 5.2.15 *verify.js*

У оквиру *deploy* директоријума налази се и скрипта *deploy-setup-frontend.js* која се користи за прослеђивање *Application Binary Interface* (*ABI) [36]* и адресе паметног уговора фронтенд делу апликације. Ова скрипта је приказана на слици 5.2.16.



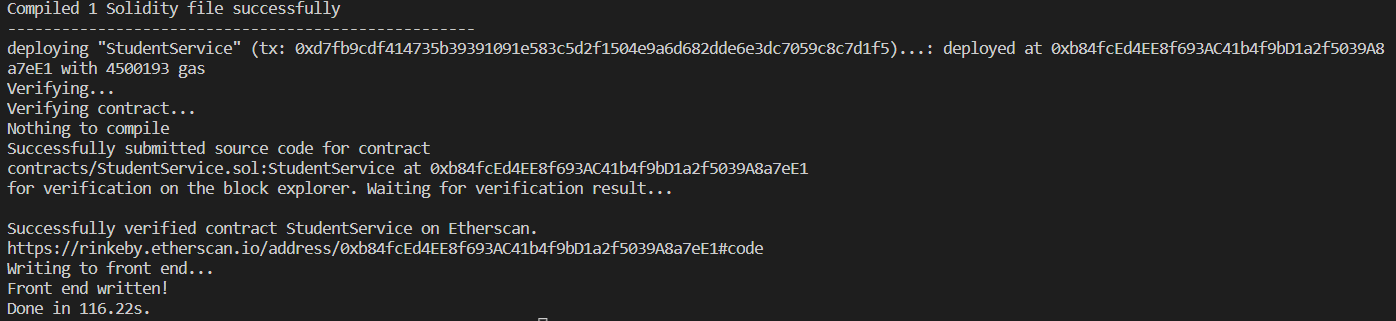
Слика 5.2.16 *deploy-setup-frontend.js*

Како бисмо урадили компајлирање, постављање и верификацију паметног уговора, довољно је унети команду као на слици 5.2.17.



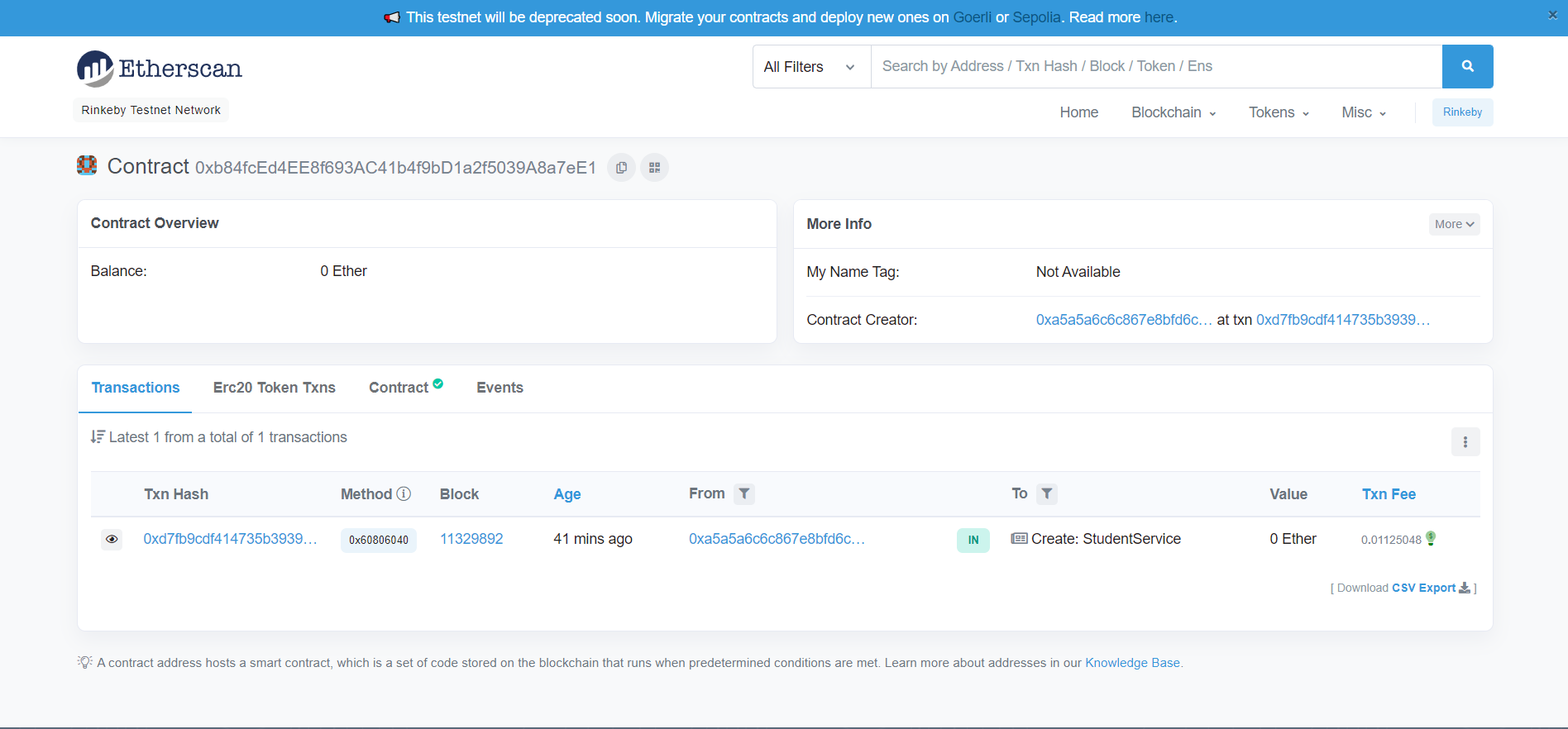
Слика 5.2.17 команда за постављање

Уколико не дође до грешке, на конзоли ће бити испит представљен на слици 5.2.18.



Слика 5.2.18 конзолни испис након постављања

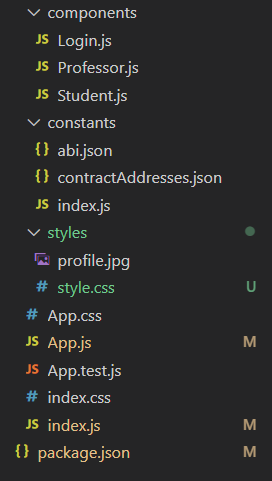
Након постављања, паметни уговор би требао бити доступан на *Etherscan* платформи. Приказ уговора на *Etherscan-у* је представљен на слици 5.2.19.



Слика 5.2.19 паметни уговор на *Etherscan* платформи

## Фронтенд део апликације

За развој фронтенд дела апликације, коришћена је једна од најпознатији *JavaScript* библиотека: *React [37].* Ова библиотека је по природи декларативна, ефикасна и флексибилна чија је главна употреба за развој корисничког интерфејса. Структура *React* пројекта је представљена на слици 5.3.1.



Слика 5.3.1 структура *React* пројекта

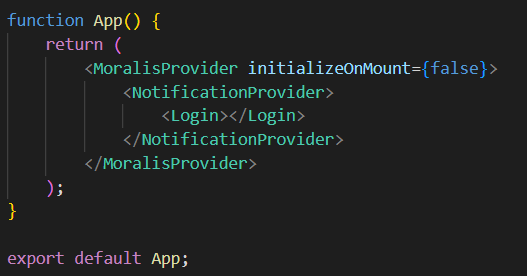
Важни елементи пројекта су:

* *components* – директоријум у којем се налазе кључни елементи сваког *React* програма звани компоненте чија је улога изградња корисничког интерфејса.
* *constants* – директоријум у којем се налазе *ABI* и адреса паметног уговора, ове информације су добијене из *deploy-setup-frontend.js* која је описана у претходном одељку.
* *styles –* директоријум у којем се налазе датотеке коришћене за стилизовање компоненти.
* *App.js –* главна компонента, компонента која се налази на врху хијерархије.
* *index.js* – улазна тачка апликације*.*

Нажалост, *React* као библиотека нема уграђену подршку за рад са блокчејном и паметним уговорима, због чега је било неопходно инсталирати наредне зависности:

* *react-moralis [38]* – библиотека која олакшава развој децентрализованих апликација. Пружа функционалисти уз помоћу којих могу да се позивају функције постављеног паметног уговора, добави стање повезаног новчаника са страницом итд. За потребе пројекта коришћене су наредне функционалности ове библиотеке:
  + *useMoralis* – функционалност која добавља информације о повезаном налогу као што су адреса налога, да ли је налог аутентификован итд.
  + useWeb3Contract – функционалност која омогућава комуникацију са паметним уговором.
* *web3uikit [39]* – библиотека која пружа стилизиране и лагане компоненте корисничког интерфејса за развој *Web 3.0* апликација. За потребе пројекта коришћене су наредне функционалности ове библиотеке:
  + *ConnectButton* – прилагођена компонента која има имплементирану логику за повезивање криптоновчаника са страницом.
  + *useNotification* – компонента обавештења која се приказује након успешно или неуспешне операције*.*

Како би *React* апликација могла да користи функционалности *react-moralis* библиотеке, неопходно је садржај *App.js* датотеке умотати *MoralisProvider* компонентом. Такође, како би могле да се користе нотификације које пружа *web3uikit* библиотека, потребно је додатно умотати коренску компоненту *NotificationProvider* компонентом. Структура *App.js* фајла је приказана на слици 5.3.2.

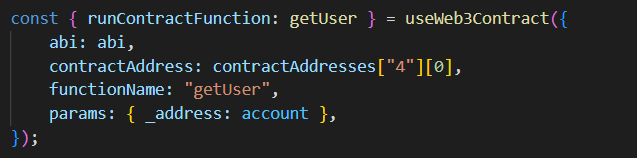


Слика 5.3.2 *App.js*

Како би се позвале функције постављеног паметног уговора из прошлог одељка, користи се *useWeb3Contract* функционалност коју пружа *react-moralis* библиотека. За коришћење ове функционалности, неопходно је дефинисати жељени назив функције у оквиру компоненте и проследили наредне параметре:

* *abi* – представља *ABI* паметног уговора.
* *contractAddress* – представља адресу постављеног паметног уговора.
* *functionName* – представља назив функције која се позива из паметног уговора.
* *params* – представља параметре функције паметног уговора.
* *msgValue* – представља стринг вредност количине *WEI (*криптовалута која је 1018пута мања од *ETH*)која се шаље паметном уговору (опционо поље).

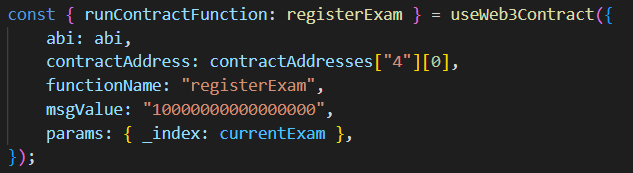
На слици 5.3.3 налази се изглед функције за добављање повезаног корисника која се налази у *Login.js* компоненти.



Слика 5.3.3 *getUser* функција

Вредности параметара *abi* и *contractAddress* су добављени из *constants* директоријума, док је вредност параметра ­\_addressпроменљива *account.* Ова променљива је извучена из *useMoralis*  функције и представља адресу тренутно повезаног новчаника.

*useWeb3Contract* је најчешће коришћена функционалност, јер омогућава позивање свих функција које су дефинисане у паметном уговору. Ради сажетости, неће бити приказани сви делови кода где је ова функционалност коришћена, него ће само бити представљена употреба useWeb3Contract за 2 главне функције, пријава испита и оцењивање испита, на сликама 5.3.4 и 5.3.5 респективно.



Слика 5.3.4 пријава испита



Слика 5.3.5 оцењивање испита

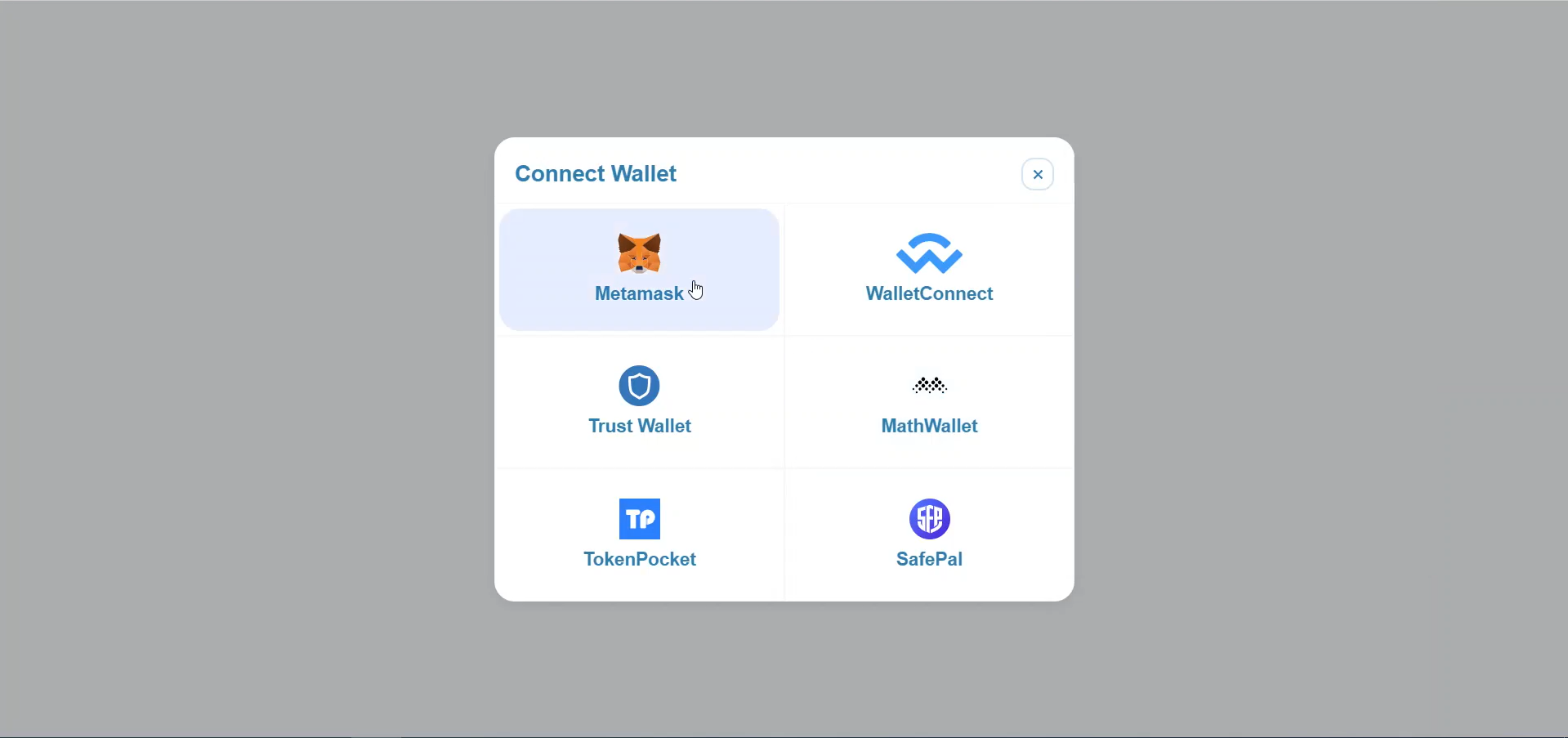
## Финална апликација

У овом поглављу биће описана апликација у њеној целости. Приликом првог уласка на апликацију, кориснику је приказана страница за пријаву. Ова страница је приказана на слици 5.4.1.



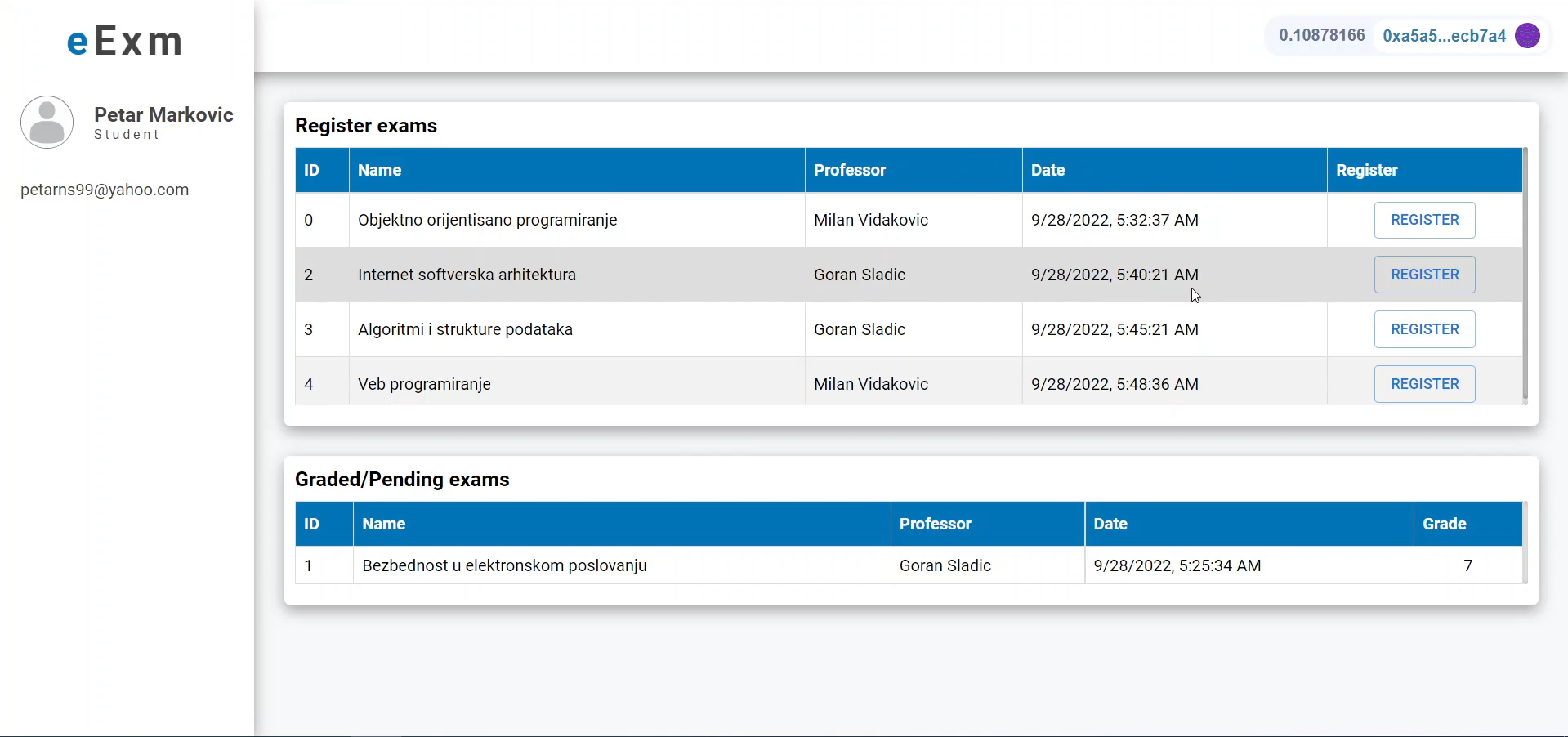
Слика 5.4.1 страница за пријаву

Притиском на дугме *Connect Wallet,* кориснику се приказује дијалог где може да изабере конкретну врсту криптоновчаника коју жели да повеже са апликацијом. Овај дијалог је представљен на слици 5.4.2.



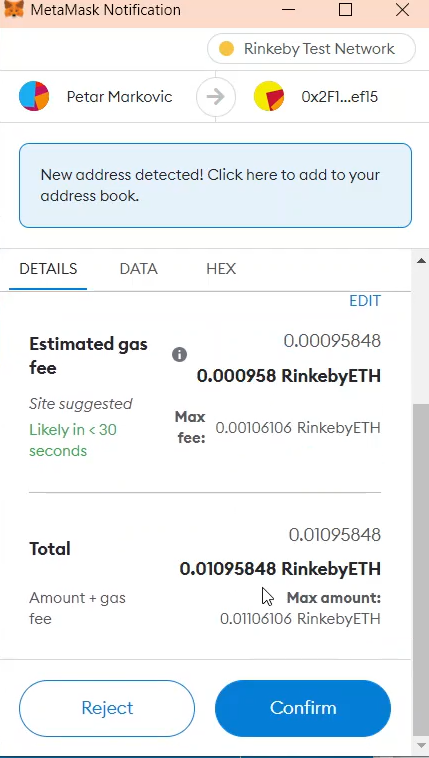
Слика 5.4.2 дијалог за избор новчаника

Након одабира новчаника, уколико су подаци о повезаном налогу сачувани у оквиру паметног уговора, прелази се на главну страницу. Ако је аутентификовани корисник студент, ова страница изгледа као на слици 5.4.3.



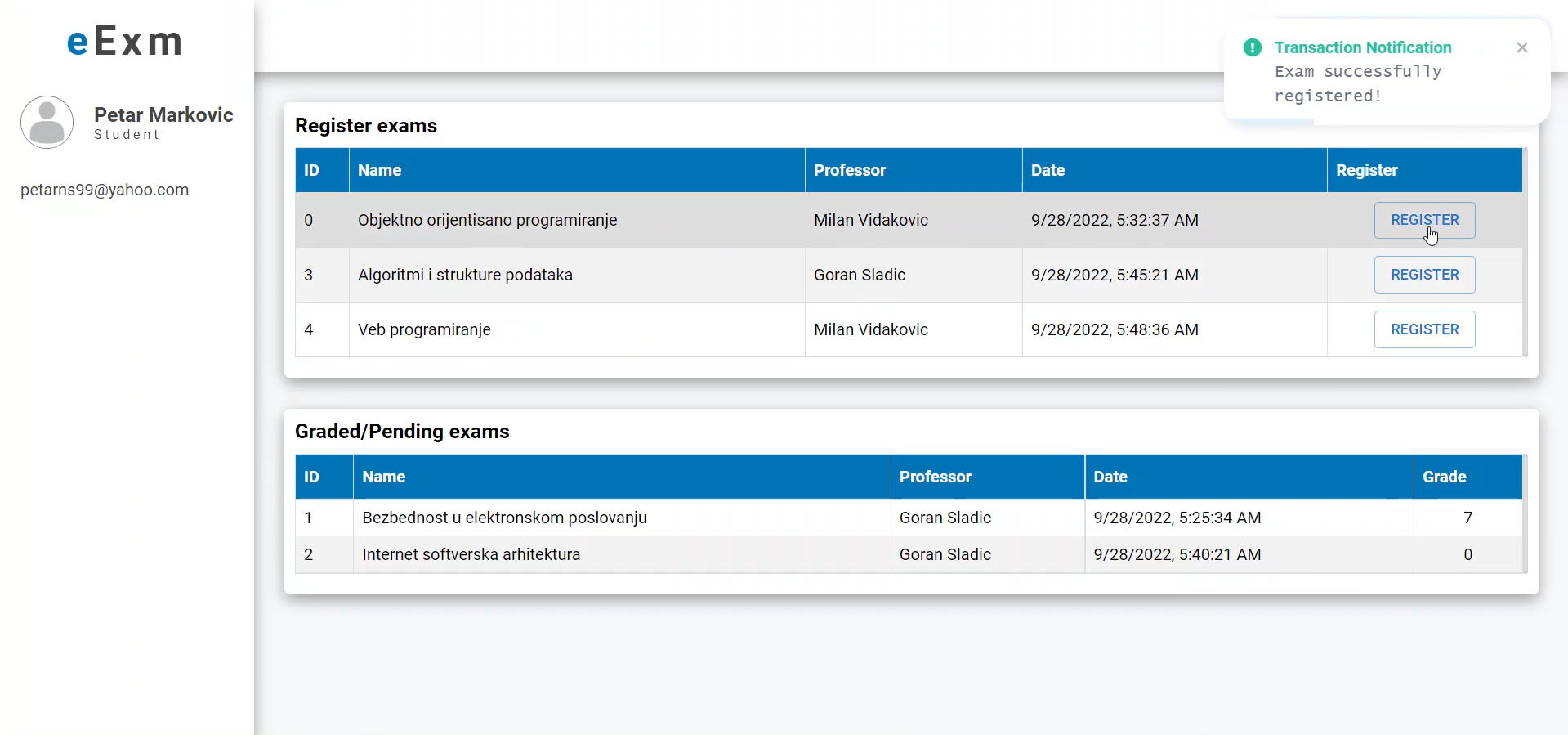
Слика 5.4.3 главна страница студента

У оквиру ове странице, горња табела представља испите које корисник може да пријави, док је доња табела скуп оцењених и пријављених испита. Такође, у горњем десном ћошку је дугме које садржи информације о салду и адреси новчаника. Студент може да пријави било који испит у горњој табели притиском на дугме *REGISTER* у оквиру колоне жељеног испита. Након притиска овог дугмета, отвара се обавештење у оквиру новчаника где корисник треба да потврди да ли жели да обави ову трансакцију у износу од 0.01 *ETH.* Ово обавештење је описано на слици 5.4.4.



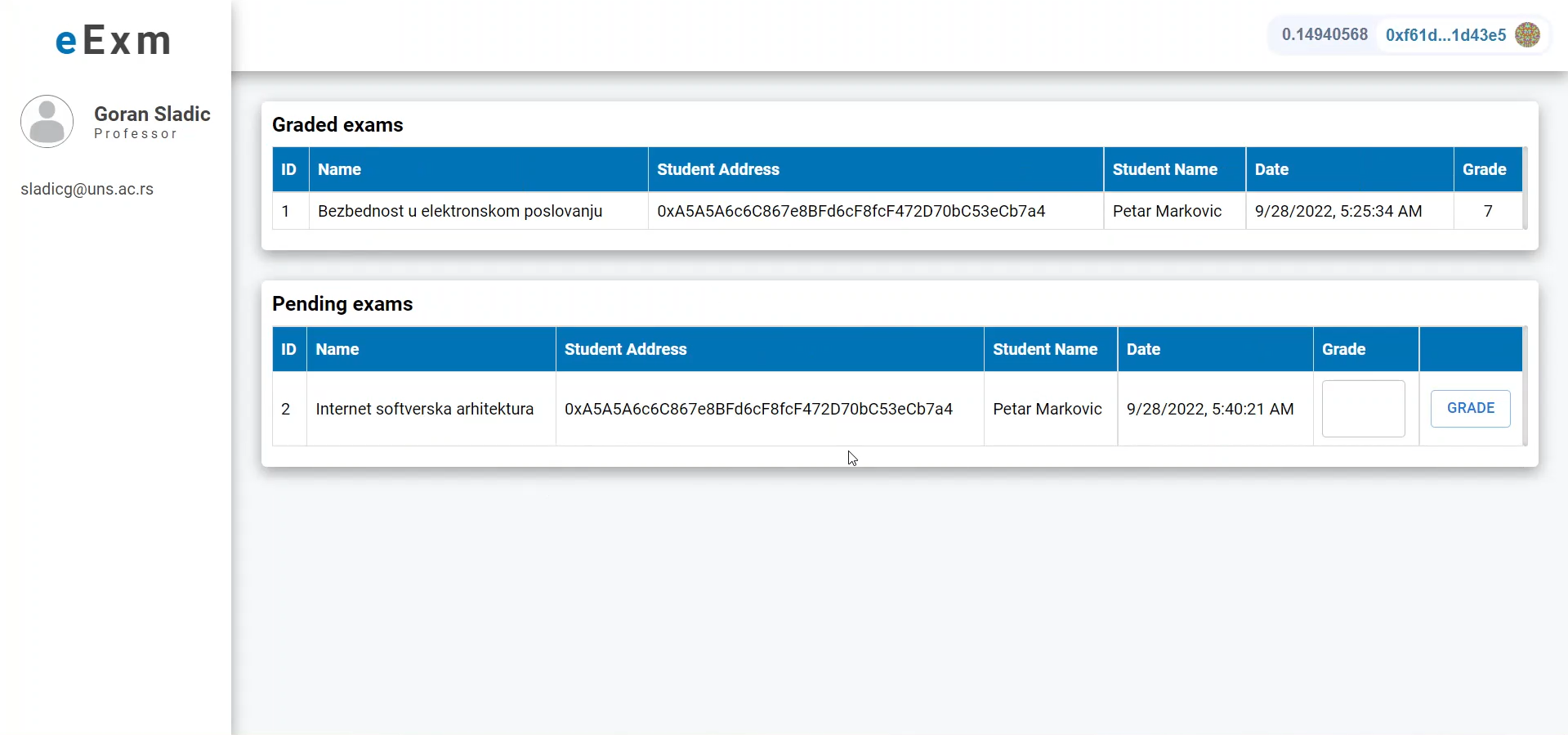
Слика 5.4.4 обавештење новчаника

Уколико студент притисне на дугме *Confirm*, мора сачекати одређено време да се ова трансакција потврди. Након што се ова пријава сачува на блокчејн, кориснику стиже нотификација. Овај ток се може видети на слици 5.4.5.



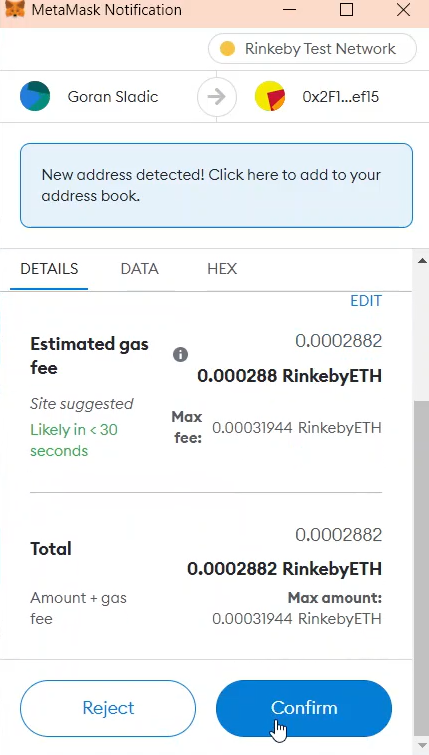
Слика 5.4.5 успешно пријављен испит

Ако се као корисник пријави професор, његова главна страна ће изгледати другачије и она је приказана на слици 5.4.6.



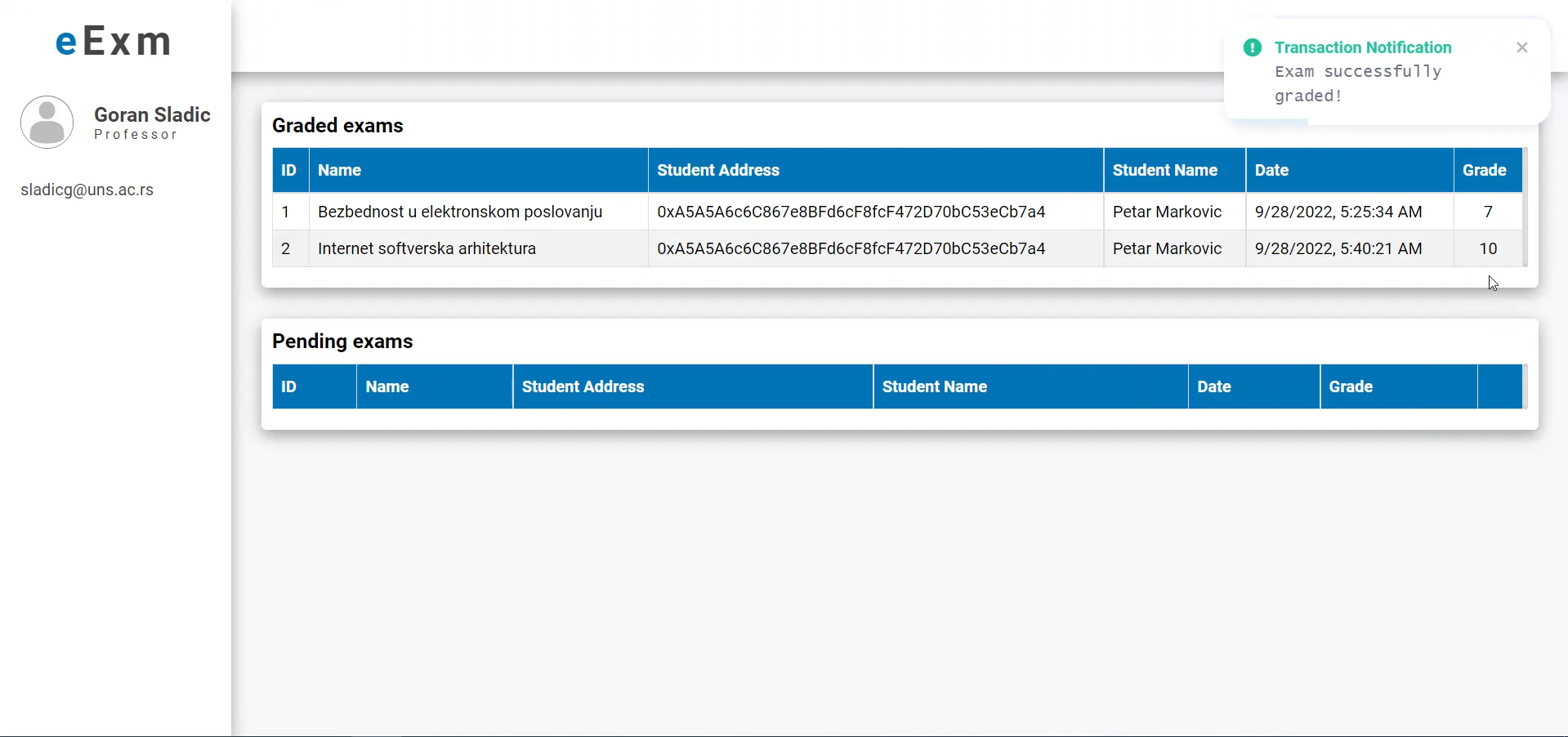
Слика 5.4.6 главна страница професора

У оквиру ове странице, горња табела представља испите које је корисник, док је доња табела скуп пријављених испита који нису оцењени. У доњој табели професор може да оцени одговарајући испит уносом оцене у претпоследњој колони и притиском на дугме *GRADE* у последњој колони*.* Након притиска овог дугмета, отвара се обавештење у оквиру новчаника где корисник треба да потврди да ли жели да обави ову трансакцију и плати само одређену количину гаса. Ово обавештење је описано на слици 5.4.7.



Слика 5.4.7 обавештење новчаника

Уколико професор притисне на дугме *Confirm*, мора сачекати одређено време да се ова трансакција потврди. Након што се овај оцењени испит сачува на блокчејн, кориснику стиже нотификација. Овај ток се може видети на слици 5.4.8.



Слика 5.4.8 успешно оцењен испит

# Закључак

У уводу рада су обрађене различите врсте система са великим фокусом на централизоване и децентрализоване системе. Направљено је директно поређење између централизованих и децентрализованих система. Закључак је да је главна разлика између ова 2 приступа у самом нивоу контроле, где централизовани систем има једно ауторитативно тело које управља свим ресурсима док у оквиру децентрализованог система не постоји централни власник, него сваки чвор складишти своје ресурсе. Поред ових структура, описан је и дистрибуирани систем. Дистрибуирани систем је дефинисан као рачунарска средина где су све компоненте расподељене по рачунарима у оквиру једне заједничке мреже.

Како је рад у великој мери имплементиран употребом блокчејн технологије, било је неопходно издвојити један сегмент који објашњава концепте и теоријске основе исте. Блокчејн је јавно доступна дистрибуирана главна књига. Предност блокчејн технологије јесте његова безбедност која је постигнута механизмом који се назива консензус алгоритам. Ово је поступак уз помоћ ког сви чворови блокчејн мреже могу да направе компромис о стању самог блокчејна. Први пут се блокчејн као технологија користи за сврхе платформе *Bitcoin* која је употребом криптографских доказа формирала систем плаћања криптовалутама. Ова блокчејн платформа је туринг непотпуна, јер осим вршења трансакција, не пружа могућност обављања комплекснијих задатака. Као алтернатива, креиране су платформе које су омогућиле уметање програмског кода и логике у сам блокчејн. Једна таква платформа је *Ethereum.* *Ethereum* технологија има свој посебан програмски језик по називу *Solidity.* *Solidity* се користи за писање паметних уговора. Паметни уговори су програми који се извршавају на *Ethereum* блокчејну.

У овом раду је имплементирана децентрализована апликација студентске службе. Како би се олакшао развој, тестирање, компајлирање и постављање паметног уговора који представља основу апликације, коришћена је *HardHat* развојна платформа. Како би се постигла комуникација са овим паметним уговором који је постављен на *Ethereum* блокчејн, 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 захтева мање гаса. Ово се може постићи реформатирањем кода или складиштењем података о корисницима и испитима ван самог ланца.

# Литература

1. Centralized vs Decentralized Logistic Distribution, [https://www.analyticssteps.com/blogs/centralized-vs-decentralized-logistic-distribution](https://www.analyticssteps.com/blogs/centralized-vs-decentralized-logistic-distribution%20)
2. Decentralized Vs. Centralized: A Detailed Comparison, [https://www.blockchain-council.org/blockchain/decentralized-vs-centralized/](https://www.blockchain-council.org/blockchain/decentralized-vs-centralized/%20)
3. What Are the Pros and Cons of Centralized and Decentralized Distribution, [https://parcelindustry.com/article-5707-What-Are-the-Pros-and-Cons-of-Centralized-and-Decentralized-Distribution.html](https://parcelindustry.com/article-5707-What-Are-the-Pros-and-Cons-of-Centralized-and-Decentralized-Distribution.html%20%20)
4. Decentralized vs Centralized: A Complete Comparison,

[https://101blockchains.com/decentralized-vs-centralized/](https://101blockchains.com/decentralized-vs-centralized/%20)

1. Distributed Systems - The Complete Guide,

<https://www.confluent.io/learn/distributed-systems/>

1. What Are Distributed Systems,

[https://www.splunk.com/en\_us/data-insider/what-are-distributed-systems.html](https://www.splunk.com/en_us/data-insider/what-are-distributed-systems.html%20)

1. Blockchain Facts: What Is It, How It Works, and How It Can Be Used,

[https://www.investopedia.com/terms/b/blockchain.asp](https://www.investopedia.com/terms/b/blockchain.asp%20)

1. Haber, S., Stornetta, W.S. How to time-stamp a digital document. J. Cryptology 3, 99–111 (1991),

<https://link.springer.com/content/pdf/10.1007/BF00196791.pdf>

1. Bitcoin A Peer-to-Peer Electronic Cash System,

<https://bitcoin.org/bitcoin.pdf>

1. What is blockchain technology,

[https://www.ibm.com/topics/what-is-blockchain](%20https:/www.ibm.com/topics/what-is-blockchain)

1. What Is Blockchain Technology How Does It Work,

[https://builtin.com/blockchain](%20https:/builtin.com/blockchain)

1. Benefits of blockchain,

<https://www.ibm.com/topics/benefits-of-blockchain>

1. Consensus Algorithm,

[https://www.techtarget.com/whatis/definition/consensus-algorithm](%20https:/www.techtarget.com/whatis/definition/consensus-algorithm)

1. Consensus Algorithms in Blockchain,

[https://www.geeksforgeeks.org/consensus-algorithms-in-blockchain/](%20https:/www.geeksforgeeks.org/consensus-algorithms-in-blockchain/)

1. Consensus Mechanism (Cryptocurrency), <https://www.investopedia.com/terms/c/consensus-mechanism-cryptocurrency.asp>
2. A Guide to Understand Blockchain Consensus Algorithms, <https://appinventiv.com/blog/blockchain-consensus-algorithms-guide/>
3. Ethereum,

<https://ethereum.org/en/>

1. Turing Complete,

<https://academy.binance.com/en/glossary/turing-complete>

1. What Is Ethereum,

<https://www.investopedia.com/terms/e/ethereum.asp>

1. Introduction to smart contracts, <https://ethereum.org/en/developers/docs/smart-contracts/>
2. Smart Contracts Definition,

<https://www.investopedia.com/terms/s/smart-contracts.asp>

1. What are smart contracts on blockchain,

<https://www.ibm.com/topics/smart-contracts>

1. Еthereum Virtual Machine,

<https://ethereum.org/en/developers/docs/evm/>

1. What Is an Ethereum Virtual Machine (EVM), <https://coinmarketcap.com/alexandria/glossary/ethereum-virtual-machine-evm>
2. Everything you need to know about Ethereum Virtual Machine (EVM), <https://www.analyticssteps.com/blogs/everything-you-need-know-about-ethereum-virtual-machine-evm>
3. What is Web 1.0, 2.0, and 3.0 and How They Compare, <https://www.simplilearn.com/what-is-web-1-0-web-2-0-and-web-3-0-with-their-difference-article>
4. What are Web1, Web2, and Web3 in simple terms, <https://www.cryptopolitan.com/what-are-web1-web2-and-web3-in-simple-terms/>
5. Web 2 vs. Web 3: What’s the Difference and Why It Matters, <https://www.spiceworks.com/tech/tech-general/articles/web-2-vs-web-3/>
6. MetaMask: The crypto wallet for Defi, Web3 Dapps and NFTs, <https://metamask.io/>
7. Ethereum Test Networks, <https://docs.ethhub.io/using-ethereum/test-networks/>
8. Alchemy, <https://www.alchemy.com/>
9. Etherscan, <https://etherscan.io/>
10. Hardhat, <https://hardhat.org/>
11. Solidity, <https://docs.soliditylang.org/en/v0.8.16/>
12. Ether, <https://ethereum.org/en/eth/>
13. ABI, <https://www.quicknode.com/guides/solidity/what-is-an-abi>
14. React, <https://reactjs.org/>
15. react-moralis, <https://www.npmjs.com/package/react-moralis>
16. Web3UI Kit, <https://web3uikit.com/>
17. Centralized vs Decentralized vs Distributed,

<https://medium.com/delta-exchange/centralized-vs-decentralized-vs-distributed-41d92d463868>

1. What is blockchain, <https://money.com/what-is-blockchain/>
2. Bitcoin vs Ethereum Smart Contracts,

<https://xiaohuiliu.medium.com/bitcoin-vs-ethereum-smart-contracts-921e0a12b043>

1. Sailian blockchain training: the core elements of Web3 - blockchain, encrypted assets, smart contract and Oracle, <https://qdmana.com/2022/03/202203040140511860.html?fbclid=IwAR2BBw9XZOINZqog-OhQw9WvwatcIU51ZTHw2ms6A0fec4qGT-CYT1L1kuY>

# Биографија

Петар Марковић рођен је 29.10.1999. године у Нишу. Похађао је основну школу „Прва војвођанска бригада“ у Новом Саду, коју је завршио 2014. године, а затим је уписао општи смер гимназије „Светозар Марковић“, такође у Новом Саду. Средње образовање је стекао 2018. године, и исте године је уписао Факултет техничких наука, смер софтверско инжењерство и информационе технологије. Положио је све испите предвиђене планом и програмом са просечном оценом 9.20.