

МІНІСТЕРСТВО ФІНАНСІВ УКРАЇНИ
ДЕРЖАВНИЙ ПОДАТКОВИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ ФІНАНСІВ ТА ЦИФРОВИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ
ТЕХНОЛОГІЙ І СИСТЕМ

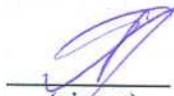
КВАЛІФІКАЦІЙНА РОБОТА
на тему: ПЕРЕДАЧА ПРАВА ВЛАСНОСТІ НА НЕРУХОМІСТЬ
ВИКОРИСТОВУЮЧИ ТЕХНОЛОГІЮ БЛОКЧЕЙН

Здобувача вищої освіти другого (магістерського) рівня, II курсу, КМІ-22-1 групи,
спеціальності 122 «Комп'ютерні науки»,
освітня програма «Інформаційні управляючі системи і технології в економіці»


(підпис)

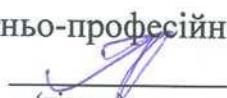
Жмудя Віталія Вікторовича
(прізвище, ім'я, по батькові)

Науковий керівник


(підпис)

Горбовий Артур Юліанович, д.т.н., професор
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Керівник освітньо-професійної
програми


(підпис)

Горбовий Артур Юліанович, д.т.н., професор
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту за рішенням кафедри комп'ютерних та інформаційних
технологій і систем від «28» грудня 2023 № 8

Завідувач
кафедри


(підпис)

Філоненко Михайло Миколайович, к.фіз.-мат.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Ірпінь – 2024

Реєстрація № 10

29.01.2024

(дата)



(підпис відповідальної особи)

Результати перевірки

до захисту

(до захисту, на доопрацювання)

29.01.2024

(дата)



(підпис)

Результат захисту

5 (відмінно)

(оцінка)

12.02.2024

(дата)



(підпис)

Голова комісії:



(підпис)

В.М. Франчук

Члени комісії:



(підпис)

А.Ю. Горбовий



(підпис)

О.В. Редич



(підпис)

В.О. Ніжегородцев

БЛАНК ОЦІНЮВАННЯ ВИПУСКНОЇ РОБОТИ

Об'єкти контролю	Діапазон оцінки в балах	Фактична кількість балів	Примітки	Дата, П.І.Б.: керівника, голови комісії членів комісії, секретаря, (підписи)
ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ КОМІСІЄЮ ПО ЗАХИСТУ				
1. Зміст випускної роботи (якість теоретичного матеріалу, його зв'язок з практичним, вміння представляти матеріал в логічній послідовності, наявність висновків та пропозицій)	від 0 до 30	29		12.02.2024 Франчук В.М. <i>В.М.</i> Горбовий А.Ю. <i>А.Ю.</i>
2. Дотримання вимог щодо оформлення	від 0 до 10	8		Редич О.В. <i>О.В.</i>
3. Вчасне виконання роботи на всіх етапах у встановлені кафедрою терміни	від 0 до 5	4		Ніжегородцев В.О. <i>В.О.</i> Суховерховська Л.П. <i>Л.П.</i>
4. Рецензент	від 0 до 5	4		
РАЗОМ БАЛІВ	від 0 до 50	45		

Оцінювання кваліфікаційної роботи науковим керівником - 47 балів.

ПЕРЕВЕДЕННЯ ОЦІНКИ ЗА 100-БАЛЬНОЮ ШКАЛОЮ ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою	Дата	П.І.Б.: голови, членів комісії, секретаря, (підпис)
92	A	5 (відмінно)	12.02.2024	Франчук В.М. <i>В.М.</i> Горбовий А.Ю. <i>А.Ю.</i> Редич О.В. <i>О.В.</i> Ніжегородцев В.О. <i>В.О.</i> Суховерховська Л.П. <i>Л.П.</i>

ПРАВИЛА ПЕРЕВЕДЕННЯ ОЦІНОК ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою
90-100	A	5 (відмінно)
80-89	B	4 (добре)
70-79	C	
60-69	D	
50-59	E	3 (задовільно)
35-49	FX	2 (незадовільно)

Факультет фінансів та цифрових технологій
 Кафедра комп'ютерних та інформаційних технологій і систем
 Спеціальність 122 «Комп'ютерні науки»
 Освітня програма «Інформаційні управляючі системи і технології в економіці»

Затверджую

Завідувач кафедри

Філоненко М.М.

« 04 » 04 2023

Завдання на кваліфікаційну роботу

здобувачу вищої освіти

Жмудю Віталію Вікторовичу

(прізвище, ім'я, по батькові)

1. Тема випускної кваліфікаційної роботи

Передача власності на нерухомість використовуючи технологію блокчейн.

Затверджена наказом ректора від «4» квітня 2023 №362

2. Строк здачі закінченої роботи «20» грудня 2023 року

3. Мета роботи дослідити можливість використання технології блокчейн для передачі права власності на нерухомість в Україні.

Об'єкт дослідження нормативно правова база та практики передачі права власності на нерухомість в Україні.

Предмет дослідження технологія блокчейн як потенційний інструмент для покращення процесів передачі права власності на нерухомість в Україні.

4. Вихідні дані до роботи теоретичні основи блокчейну та його застосування в різних сферах, особливості передачі права власності на нерухомість в Україні, можливі сценарії використання блокчейну для передачі права власності на нерухомість.

5. Перелік графічного матеріалу.

6. Зміст випускної кваліфікаційної роботи.

ЗМІСТ

ВСТУП

РОЗДІЛ 1. БЛОКЧЕЙН ТА ТЕПЕРІШНІЙ СТАН ПЕРЕДАЧІ ПРАВА ВЛАСНОСТІ НА НЕРУХОМІСТЬ

1.1. Законодавство про право власності

1.2. Аналіз сучасних способів передачі права власності на нерухомість

1.3. Переваги використання блокчейну для трансферу власності

РОЗДІЛ 2. СТВОРЕННЯ ІНФРАСТРУКТУРИ ДЛЯ ПЕРЕДАЧІ ПРАВА ВЛАСНОСТІ НА НЕРУХОМІСТЬ

2.1. Вибір блокчейн платформи для передачі власності

2.2. Розробка смарт-контракту для запису та передачі права власності

2.3. Забезпечення безпеки та відповідності до правових норм

РОЗДІЛ 3.ПРОПОЗИЦІЇ ЩОДО ВИКОРИСТАННЯ ТЕХНОЛОГІЙ БЛОКЧЕЙН В СФЕРІ ПЕРЕДАЧІ ПРАВА ВЛАСНОСТІ НА ОСВІТУ. РЕЗУЛЬТАТИ ПРОВЕДЕНОГО ДОСЛІДЖЕННЯ

3.1. Пропозиції щодо створення повноцінної інфраструктури на основі блокчейн для проведення дослідження

3.2. Пропозиції щодо забезпечення повноцінної перевірки смарт-контрактів на відповідність його до норм права та безпеки

ВИСНОВКИ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ДОДАТКИ

7. Календарний план виконання роботи

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	
		За планом	За фактом
1	2	3	4
1	Вибір теми, підбір літературних джерел, формування структури кваліфікаційної роботи	До 27.12.2022	27.12.2022
2	I розділ (розкрити сутність і стан дослідження наукової проблематики роботи, після чого навести її загальну характеристику; огляд літератури за темою і вибір конкретних напрямів досліджень)	26.06.2023	26.06.2023
3	II розділ (аналіз експериментальної частини про методики досліджень)	23.10.2023	23.10.2023
4	III розділ (проведені теоретичні і (або) експериментальні дослідження та проектні рішення)	30.11.2023	30.11.2023
5	Висновки	10.12.2023	10.12.2023
6	Доопрацювання роботи з урахуванням зауважень наукового керівника	20.12.2023	20.12.2023
7	Отримання відгуку керівника	29.01.2024	29.01.2024
8	Отримання зовнішньої рецензії	01.02.2024	01.02.2024

8. Дата видачі завдання «4» квітня 2023

9. Керівник кваліфікаційної роботи

(підпис)

Горбовий А.Ю.
(прізвище, ініціали)

10. Керівник освітньо-професійної програми

(підпис)

Горбовий А.Ю.
(прізвище, ініціали)

11. Завдання до виконання отримав здобувач вищої освіти

(підпис)

Жмудь В.В.
(прізвище, ініціали)

АНОТАЦІЯ

Передача права власності на нерухомість використовуючи технологію блокчейн

Метою роботи є дослідити можливість використання технології блокчейн для передачі права власності на нерухомість в Україні.

Завданнями роботи є:

- проведення аналізу ефективності передачі права власності через блокчейн;
- дослідити якою є максимально можлива автоматизація процесів при теперішньому законодавстві;
- розробити код смарт-контракту для передачі права власності;
- забезпечення смарт-контракту на відповідність до правових норм;
- розробити смарт-контракт який буде відповідати за збір податків.

Предметом дослідження є технологія блокчейн як потенційний інструмент для покращення процесів передачі права власності на нерухомість в Україні.

Об'єктом дослідження є нормативно-правова база та практики передачі права власності на нерухомість в Україні.

За результатами дослідження сформульовані рекомендації щодо оптимальної передачі права власності через блокчейн, а саме впровадження електронної гривні, використання окремої закритої бази даних для збереження надісланих документів, реалізація смарт-контракту для автоматизації процесів.

Одержані результати можуть бути використані державою Україна задля збільшення автоматизації при передачі права власності через блокчейн.

Кваліфікаційна робота містить 52 сторінки, 3 таблиці, 27 рисунків, список використаних джерел найменувань 43, 4 додатки.

Ключові слова: блокчейн, передача права власності, безпека, аудит , автоматизація процесів, аудит смарт-контрактів.

Рік виконання кваліфікаційної роботи 2023

Рік захисту роботи 2024

ABSTRACT

Property Ownership Transfer Using Blockchain Technology

The purpose of this study is to explore the possibility of utilizing blockchain technology for property ownership transfer in Ukraine.

The research tasks include:

- to analyze the efficiency of transferring ownership through the blockchain
- to investigate the maximum possible automation of processes under the current legislation
- develop a smart contract code for transferring ownership
- ensure the smart contract is in compliance with legal regulations
- develop a smart contract that will be responsible for tax collection.

The subject of the study is blockchain technology as a potential tool for improving the processes of transferring ownership of real estate in Ukraine.

The object of the study is the legal framework and practices of transferring real estate ownership in Ukraine.

Based on the results of the study, recommendations for the optimal transfer of ownership through the blockchain are formulated, namely the introduction of the electronic hryvnia, the use of a separate closed database for storing sent documents, and the implementation of a smart contract for automating processes.

The results obtained can be used by the state of Ukraine to increase automation in the transfer of ownership through the blockchain.

The qualification work contains 52 pages, 3 tables, 27 figures, a list of references, 43 titles, 4 appendix.

Keywords: blockchain, transfer of ownership, security, audit, process automation, audit of smart contracts.

Year of qualification work 2023

Year of thesis defense 2024

ЗМІСТ

ВСТУП.....	8
РОЗДІЛ 1. БЛОКЧЕЙН ТА ТЕПЕРІШНІЙ СТАН ПЕРЕДАЧІ ПРАВА ВЛАСНОСТІ НА НЕРУХОМІСТЬ	11
1.1. Законодавство про право власності	11
1.2. Аналіз сучасних способів передачі права власності на нерухомість	12
1.3. Переваги використання блокчейну для трансферу власності	16
РОЗДІЛ 2. СТВОРЕННЯ ІНФРАСТРУКТУРИ ДЛЯ ПЕРЕДАЧІ ПРАВА ВЛАСНОСТІ НА НЕРУХОМІСТЬ.	22
2.1. Вибір блокчейн платформи.....	23
2.2. Розробка смарт-контракту для запису та передачі права власності.....	25
2.3. Реалізація безпеки та відповідності до правових норм	35
РОЗДІЛ 3. ПРОПОЗИЦІЇ ЩОДО ВИКОРИСТАННЯ ТЕХНОЛОГІЇ БЛОКЧЕЙН В СФЕРІ ПЕРЕДАЧІ ПРАВА ВЛАСНОСТІ НА ОСНОВІ ПРОВЕДЕНОГО ДОСЛІДЖЕННЯ.....	45
3.1. Пропозиції щодо створення повноцінної інфраструктури на основі проведеного дослідження.....	46
3.2. Пропозиції щодо забезпечення повноцінної перевірки смарт-контракту та відповідності його до норм права та безпеки	49
ВИСНОВКИ	50
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ.....	52
ДОДАТКИ.....	57

ВСТУП

Актуальність теми. Передача прав власності через технологію блокчейн є актуальною завдяки забезпеченню децентралізованої та ефективної системи, яка пропонує прозорість, швидкість та високий рівень безпеки. Це рішення допомагає зменшити роль посередників, зменшити ризики шахрайства та забезпечити глобальну можливість обміну правами власності. У секторі нерухомості, блокчейн спрощує та прискорює процеси, роблячи передачу власності більш надійною та ефективною.

Метою кваліфікаційної роботи є дослідження можливості використання технології блокчейн для реалізації передачі права власності на нерухомість.

Для реалізації даного проекту ми виконали наступні завдання:

- провели дослідження нормативної бази задля відповідності нашого рішення правовим нормам;
 - підібрали блокчейн платформу для реалізації передачі права власності;
 - розробили інфраструктуру смарт-контрактів для реєстрації , оподаткування;
- та реалізації виконання процесів пов'язаних з обміном власності на нерухомість;
- забезпечили рольову модель у нашому смарт-контракті задля належного управління процесами, щоб відповідати правовим нормам;
 - надали рекомендації щодо покрокового впровадження нашого рішення.

Предмет дослідження — технологія блокчейн як потенційний інструмент для покращення процесів передачі права власності на нерухомість в Україні.

Методи дослідження — теоретичні: аналіз літературних джерел; статей; публікацій; дослідження блокчейн платформ; емпіричні — аналіз відповідності роботи створеного нами рішення на відповідність нашим очікуванням; безпековим та правовим нормам.

Науково-практична цінність дослідження полягає в визначенні технології блокчейн як потенційного інструменту для оптимізації процесів передачі прав

власності на нерухомість в Україні. Результати дослідження можуть послужити підґрунтям для розробки інноваційних та ефективних рішень у сфері нерухомості.

Практичне значення отриманих результатів виявляється у можливості впровадження рекомендацій та розроблених смарт-контрактів для поліпшення процесів обміну правами власності на нерухомість, що сприятиме створенню більш ефективної та безпечної системи передачі власності через технологію блокчейн.

Апробація результатів роботи відбулася на міжнародній науково-практичній конференції «Цифрова трансформація фінансової системи України та країн V-4 в умовах євроінтеграції», Додаток Г, а також були опубліковані тези у збірнику матеріалів конференції. [18]

РОЗДІЛ 1. БЛОКЧЕЙН ТА ТЕПЕРІШНІЙ СТАН ПЕРЕДАЧІ ПРАВА ВЛАСНОСТІ НА НЕРУХОМІСТЬ

1.1. Законодавство про право власності

Право власності безперечно є однією з найвищих цінностей у сучасному світі. На меті в цьому підрозділі дослідити нормативно-правову базу, яка потрібна для трансферу власності, для подальшого використання відповідаючи юридичним, бюрократичним нормам у нашому дослідженні.

На основі документів які вказані у статті 27 Закону України "Про державну реєстрацію речових прав на нерухоме майно та їх обтяжень" [Реєстрація речових прав], наші нотаріуси (далі може використовуватись слово валідатор і у контексті даної роботи ці поняття означають одне і те ж) ми можемо приймати заявки від потенційних користувачів на внесення їх нерухомості до нашої системи з подальшою ціллю її зберігання або продажу. Одним з основних завдань безперечно є висока кваліфікаційна підготовка нотаріусів задля оперативного опрацювання отриманих документів від користувачів, перевірка їх достовірності згідно державного реєстру речових прав на нерухоме майно який можна перевірити за допомогою ресурсу <https://online.minjust.gov.ua/information/>.

Після підтвердження нотаріусом права власності, фізична особа яка звернулась з цим запитом може додати цей запис про нерухомість на основі дозволів виданих їй валідатором у нашій системі. Після того як запис про нерухомість була додана, людина може її просто утримувати як цифрове підтвердження цієї власності або ж виставити на продаж. Так як нашою метою не є створення маркетплейсу для продажу нерухомості, то цілком очевидно, що дві сторони угоди перед самим процесом купівлі-продажу будуть проводити перемовини і лише коли вони дійдуть згоди, тільки тоді продавець для конкретного запису нерухомості (за реєстраційним номером) вкаже ціну продажу та адресу потенційного покупця. Чітке вказання покупця дозволить нам реалізувати таку бізнес-логіку при якій продавцю не доведеться відмінити безліч вхідних заявок про покупку нерухомості від небажаних осіб.

Одним з важливих аспектів продажу нерухомості є її ціна, адже на її основі проводиться сплата податків, протеми повинні зазначити, що теоретично продавець міг би продати свою нерухомість і за 1 грн, проте згідно законодавства, повинна проводитись оцінка вартості нерухомого майна[3] і податки та збори повинні бути оплачені згідно реальної оцінки вартості нерухомості, тому одним із завдань є додавання функціоналу для нотаріуса в якому він буде вказувати мінімальну сумму для розрахунку податку, якщо сама сума продажу буде нижчою реальної оцінки вартості квартири. Ми хотіли б зазначити, що насправді наврядчи людина захоче продати свою нерухомість всього лише за 1 грн і потім сплатити податків на сумму 15000 грн (сумма є умовною), згідно реальної оцінки нерухомості оцінювачем, а це означає, що потрібно або додати параметр на обмеження мінімальної ціни продажу, або додати додатково параметр реальної оцінки нерухомості, по якому ми розраховуватимемо податки та прийматимемо їх на наш смарт контракт. У підсумках до цього абзацу зазначимо, що включення ринкової ціни до параметрів при продажі нерухомості може заохотити продавця встановлювати такі ціни які будуть не нижчі, ніж ті, що встановлені оцінювачами нерухомості.[3]

1.2. Аналіз сучасних способів передачі права власності на нерухомість

Згідно законодавства України нерухомість може бути передана відповідно положень вказаних в цивільному кодексі України. [4] Зокрема там вказані такі способи передачі права власності — спадщина, купівля-продаж, дарування, рента, договір довічного утримання, пожертвою, рентою.

Передача права власності у спадщину здійснюється згідно спадковим договором відповідно до статті 1302 ЦКУ та визначає, що за даним договором спадкоємець зобов'язується виконувати розпорядження відчужувача і у випадку смерті останнього спадкоємець отримує право власності на його майно у спадщину. [4] Якщо особа яка є потенційним набувачем майна спадкодавця відмовляється від спадщини, то спадщина передається або наступній особі вказаній у заповіті або ж згідно кола осіб які можуть претендувати на спадщину,

які визначені у статті 1258 ЦКУ. [4] Відповідно це означає, що у нашій системі для передачі права власності на нерухомість [18] , ми повинні розглянути можливість реалізації функціоналу додавання власником потенційних спадкоємців. Також згідно статті 5 ЦКУ нотаріус може засвідчити заповіт і в дійсності сам документ вважається дійсним лише якщо нотаріус вчинив таку дію. [4] З цього випливає, що ми могли б реалізувати наступну схему передачі власності на нерухомість у блокчейні:

1. Спадкодавець пише текст спадщини оформлює його у нотаріуса а далі вже завантажує цей файл у систему;
2. На основі тексту вказаному у документі нотаріус додає до відповідної нерухомості список осіб до яких може бути передана ця нерухомість у вказаному порядку відчужувачем;
3. Нотаріус через відповідний функціонал відправляє цю нерухомість згідно черги;
4. Спадкоємець приймає цю нерухомість собі або відмовляється від неї.

Проте головною проблемою при такому способі є уникнення маніпулювань [18] нотаріусом при передачі нерухомого майна, тому список ідентифікаторів спадкоємців звичайно ж повинен вказуватись в окремих колонках і передача права власності здійснюватиметься відповідно даних вказаних у цих колонках. Проте це і означає, що ми не можемо закрити список потенційних спадкоємців, так як по перше всі дані в блокчейні є відкритими і щоб зчитати їх він повинен мати доступ, а це означає, що має бути запис таких даних, по друге спадкодавці повинні бачити що функція викликана не нотаріусом а відчужувачем, що фактично означає, що нотаріус має видавати дозволи на те, адреси яких користувачів може вказувати той хто написав текст заповіту. Також зазначимо, що у випадку відсутності заповіту нотаріус має відправити спадщину тим особам які визначені статтею 1258 ЦКУ.[4] Це фактично означає, що ми зобов'язуємо нотаріусів додавати для користувачів у відповідні поля їх родичів і відповідно цих полів здійснювати набуття власності у спадщину в випадку відсутності заповіту. Тим не менш це є складним процесом адже тоді ми додаємо такий функціонал

валідаторові використовуючи який він може передати власність, теоретично ще при живому власникові, з цього випливає, що при такій передачі валідатор повинен публікувати документ(свідоцтво про смерть) на основі якого він передав дане право власності певній особі, задля збереження прозорості процесу та уникнення ситуацій коли нотаріус користуючись своїми службовими можливостями передає власність своїм друзям або знайомим, навіть не дивлячись на те, що такі дії підпадають під кримінальне законодавство, ми повинні уникати їх ще на етапі технологічних рішень.

Одним з найбільш зрозумілих на нашу думку є процес передачі права власності на нерухомість шляхом купівлі-продажу. Можна побачити, у додатку А, що найбільша кількість трансферу власності на нерухомість відбувається саме цим способом, а саме у 2023 році ця кількість становить 206043 посвідчений договір, Це означає, що даний спосіб є найпопулярнішим для трансферу власності і його реалізація у системі є найбільш бажанішою. Згідно з цим способом продавець та покупець виконують наступні кроки для його реалізації:

- готує документи для продажу нерухомості;
- покупець проводить експертну оцінку нерухомості яку він бажає продати;
- покупець виставляє нерухомість шляхом звернення до агента з продажу або на відповідних маркетплейсах;
- знаходить покупця(або покупець його), вони введуть перемовини про сумму купівлі нерухомості;
- сторони угоди укладають попередній договір(у випадку якщо нерухомість ще не введена в експлуатацію);
- покупець сплачує податки від доходу продажу(якщо дохід був більший, за оцінювану вартість), в іншому випадку сплачує відсоток від оцінюваної вартості;
- сторони угоди підписують основний договір після сплати всіх податків та договір затверджує нотаріус. [3], [5], [6].

На жаль неможливо точно встановити який термін займає перехід власності від минулого власника з моменту проведення оцінки вартості нерухомості до фактичного продажу нерухомості, адже процес знаходження покупця завжди займає різний час. Швидкість оформлення договору теж залежить від фактичної завантаженості нотаріуса до якого звернулись сторони, тому цей показник важко оцінювати, проте ми можемо провести аналіз цін нотаріальних послуг, які в майбутньому ми можемо використовувати для порівняння чи буде перехід на блокчейн ефективнішим за той спосіб, що є зараз. Щоб порівняти ціни на послуги різних нотаріусів скористаємось наступною таблицею:

Таблиця 1.1.

Ціни нотаріусів (договори купівлі-продажу)

Вартість послуги у гривнях (оформлення договору купівлі- продажу)	Джерело інформації
5500 грн. + 1 % д/м(але не менш ніж 1500 грн.) + 1 % ПФ+ адмін. збор	* https://lexukraine.com.ua/vartist-notarialnih-poslug/
Від 6000 грн та обов'язкові збори	* https://www.kyiv-notarius.com.ua/cini-notarialnih-poslug
Від 5500 грн	https://unidoc.com.ua/notarius-kiev/#cena
Від 1500 грн	https://notarius-ua.top/notariusy-sofiyivska-borshhagivka/#ceny
Від 500 до 3000 грн	https://notarius-ua.top/notariusi-odesa/

*Джерело

Як бачимо з данної таблиці, ціни на послуги нотаріуса залежать від міста в якому проводяться нотаріальні дії, так нотаріус у Києві буде вартувати найбільшу сумму, тоді як в інших містах ця ціна може бути нижчою. Далі ми використаємо цю таблицю для порівнянь наскільки ефективнішою буде купівля-продаж нерухомості через блокчейн.

Розглядаючи як приклад міну нерухомості [4] то там процес не суттєво є відмінним від того , що відбувається під час її продажу , головна відмінність лише

в тому , що під час міни нерухомість одного власника переходить до іншого , а нерухомість останнього до першого, іноді може бути обмін з доплатою , якщо обмінюється нерухомість з різною оцінковою вартістю. Загалом процес виглядає наступним чином:

1. сторони угоди домовляються про обмін нерухомості;
2. далі вони укладають договір, який має бути сфоровано згідно юридичних вимог;
3. на наступному етапі сторони реєструють договір;
4. після реєстрації договору відбувається оплата податків та зборів;
5. після здійснення всіх необхідних процедур обмін вважається завершеним.

Технічно обмін нерухомості можна прирівняти до купівлі-продажу, все залежить від того яку сторону ми вважаємо тією, що використовує свою власність більше як засіб оплати ніж як інструмент продажу але це філософське питання. Також зазначимо , що якщо вірити даним Міністерства Юстиції , які наведені у Додатку А, то ця можливість використовується не так вже й часто, проте враховуючи відносну легкість та зрозумілість цього процесу ми могли б перенести його на блокчейн щоб пришвидшити обмін нерухомості та зробити його більш доступним, мінімізуючи бюрократичні процедури.

1.3. Переваги використання блокчейну для трансферу власності

Однією з проблематик теперішньої передачі власності є велика залежність від нотаріусів та централізація процесу при якій іноді недобросовісні нотаріуси можуть змінювати власника адже вони мають доступ до реєстру і завдяки цьому теоретично можуть змінити там власника нерухомості переписавши її на іншу людину, що фактично означає, що ви насправді не контролюєте свою власність. Рада бізнес-омбудсмена у своєму звіті за третій квартал 2019 року зазначає, що трапляються випадки, коли недобросовісні державні реєстратори (включаючи нотаріусів) вносили реєстраційні зміни, завантажуючи в систему не документи, а чисті сторінки формату А4. [7] , що фактично означає наступне ми повинні

додати прозорості до процесу проведення реєстрації речових прав на нерухоме майно, а саме:

- забрати можливість у нотаріусів безпосередньо проводити реєстрацію права власності у системі, адже це створює залежність від його рішень та створює корупційне підґрунтя, адже все, що вам потрібно це знайти нотаріуса який погодиться внести підроблені документи до реєстру задля зміни власника на ваше ім'я;
- публікувати документи (або посилання на документи) на основі яких була проведена реєстрація;
- створення системи відслідковування дій кожного окремого нотаріуса задля виявлення реальним власником нерухомості можливих неправомірних дій;
- контроль за передачею та записом нерухомості повинен залежати від домовленості сторін а не від посередників.

З цих фактів випливає, що нам потрібна прозора система , в якій всі дії будуть видимі всім користувачам, а як відомо блокчейн є відкритим реєстром записів у якому неможливо приховати якісь дії які вже були записані в нього. Блокчейн забезпечує можливість відстеження слідів аудиту, автоматизований облік і узгодження, відстеження власності активів, аутентифікацію транзакцій. [8] Аналіз вищенаведеного тексту відповідає тому з яких окремих елементів складається прозорість в блокчейні та як вона забезпечується, а саме:

1. відстеження слідів аудиту: Кожна транзакція внесена до блокчейну залишає за собою слід, який можна легко відслідковувати. Це створює чітку історію всіх операцій, що покращує прозорість [18] ;
2. автоматизований облік і узгодження: Використання розподіленої бази даних в блокчейні дозволяє автоматизувати процес обліку і узгодження, зменшуючи ймовірність помилок та конфліктів даних;
3. блокчейн забезпечує точне і відстежене відображення власності активів, що робить можливим швидке та надійне визначення власників;

4. криптографічні методи в блокчейні гарантують аутентифікацію та валідацію транзакцій, запобігаючи фальсифікації та несанкціонованим змінам.

Не дивлячись на ці переваги блокчейну є критично важливим вказати, що абсолютна його прозорість може бути і мінусом, коли справа стосується конфіденційних даних та інформації, це означає, що недопустимо використовувати дану технологію задля збереження персональної інформації. Проте критично важливим водночас є мати змогу отримати інформацію про те кому саме належить той чи інший об'єкт власності, а це на нашу думку означає, що нам потрібно мати окрему захищену інфраструктуру задля зберігання посилань на фізичну особу яка є власником певного запису нерухомості. Теоретично це можна було б реалізувати наступним чином:

- всі записи та транзакції прив'язуються до певного гаманця;
- у блокчейні зберігається анонімність і ми ніде не розкриваємо там хто ким є, забезпечуючи конфіденційність;
- використовуємо окрему закриту базу даних задля збереження персональних даних , документів;
- ключем в такій БД буде адреса гаманця для певного користувача, значеннями відповідно прізвище, ім'я, по батькові, номер паспорту.

Таким чином ми не зберігатимемо інформацію про користувачів відкрито, а триматимемо її в закритому стані, куди матимуть доступ лише нотаріуси та інші особи уповноважені на здійснення реєстраційної діяльності. Відповідаючи на питання як забезпечуватиметься справедливість роботи нотаріусів, то можемо сказати, що це відбуватиметься наступним чином:

1. кожен валідатор матиме свій ліцензійний номер по якому можна буде перевірити компетенцію нотаріуса;
2. відкритість інформації про дії кожного нотаріуса повинні змусити їх зменшити зловживання своїм становищем;

3. нотаріуси не зможуть самотійно реєструвати, вносити зміни в нерухомість власників, замість цього вони видаватимуть дозволи на створення запису власності;
4. слідуючи логіці наведеній в пункті 3 ми зазначаємо, що на нотаріуса в основному покладається завдання правильної ідентифікації користувача та його документів;
5. якщо ідентифікація пройдена успішно , нотаріус, завдяки наданим йому функціям підтверджує особу, також надає дозвіл на внесення своєї власності в реєстр.

Ми декілька разів акцентували увагу на тому, що нотаріус не матиме можливості реєструвати власність, а лише надавати дозвіл на таку реєстрацію, а все тому , що даючи дозвіл на реєстрацію, він фактично отримує владу вносити в реєстр ті дані які він забажає і ви стаєте заручником совісті конкретної особи. Звичайно, ми розуміємо, що обмежуючи дозвіл валідатора лише на надавання підтверджень, це ще не гарантує того, що він не надасть ці підтвердження підставній особі, проте відкритість та доступність відслідковування його дій має на меті на нашу думку мінімізувати ці наслідки. Відповідно перевагою використання блокчейну для передачі права власності є той аспект, що ми можемо мінімізувати вплив третіх сторін і відповідну затримку через це при передачі права власності, завдяки збереженню посилання на адресу гаманця яка належить тому чи іншому користувачеві відповідно, тобто при реєстрації у системі ми зберігатимемо інформацію про цю адресу та дані власників у окремій захищеній базі даних, щоб відслідковувати операції які були здійснені певним користувачем з його нерухомістю, тобто адреса гаманця користувача буде нашим певним ключем для доступу до інформації про те ким є цей користувач і так ми уникаємо проблем з конфіденційністю, адже публічно видимою буде лише інформація про його адресу його криптогаманця. Це фактично означає що публічно буде видима лише та інформація яка не розкриває персональних даних користувачів і також користувачі, так як вони знають адресу своїх гаманців зможуть завжди відслідкувати ті операції які вони робили у блокчейні. Це можна

зробити наприклад на таких ресурсах як etherscan.io, bscscan.com. Звичайно потрібно враховувати і людський фактор при якому фізична особа може пройти перевірку та бути підтверджена але в майбутньому теоретично через людські або інші фактори, вона може скомпрометувати свій приватний ключ, що фактично означатиме, що вона втратила контроль над своїм гаманцем і задля підвищення безпеки ми можемо розглянути систему взаємодії основного гаманця з сукупністю пов'язаних гаманців, використовуючи основний гаманець для запису, обміну власності, тощо, інші гаманці використовуються для підтвердження операцій здійснених основним гаманцем. Проте потрібно розуміти, що підвищення безпеки збільшує комісійні витрати, таким чином ми можемо залишити вибір за користувачем, використовувати систему з одним гаманцем для введення операцій в системі або ж використовувати систему описану вище. Як можна побачити описана система вище схожа на мультипідпис. Multisig (мультипідпис) – це особливий тип цифрового підпису, який дозволяє двом або більше користувачам підписувати документи групою. Таким чином, мультипідпис створюється шляхом поєднання декількох унікальних підписів. [14] Як наведено в статті [14] є багато варіацій того як ми можемо отримувати доступ до різних операцій використовуючи більшість ключів для підписів повідомлень або ж ми повинні використовувати абсолютно всі ключі щоб підписати нашу транзакцію. На цьому етапі ми хотіли б зазначити що використання абсолютно всіх ключів для підпису транзакцій не є гарним рішенням, так як при ньому зловмиснику досі буде достатньо взламати лише один гаманець задля блокування операцій користувача, тому виходячи з даного ствердження ми пропонуємо використовувати систему при якій задля підтвердження транзакцій нам потрібно буде використати більшість підписів.

Наступним важливим аспектом блокчейну є його безпека. Вона заснована на взаємодії різних криптографічних функцій та алгоритмів, однією з таких є хеш функція. [9] Автори статті [10] дають визначення хеш функції. Хеш-функція - це функція, яка відображає дані довільного розміру в дані фіксованого розміру. Значення, які повертає хеш-функція, називають хеш-значеннями, хеш-кодами,

сумами хешу або просто хешами. Хеш-функції в основному використовуються в хеш-таблицях. Хеш-таблиця - це колекція елементів даних, які зберігаються так, щоб їх легко було знайти пізніше. Кожне положення в хеш-таблиці називається слотом. Слот може утримувати елемент даних і має призначене ціле значення, починаючи з 0. Слот ідентифікується ключем. Елемент даних, збережений в слоті, називається значенням. Основні властивості якими має володіти хеш-функція, щоб вважатись надійною наводяться автором на ім'я Спенсер Міллер [11], зокрема вона повинна бути стійкою до колізій, тобто гарантувати мінімальну ймовірність того, що різні входні параметри функції повертатимуть відмінне значення.

Зокрема хеш-функція є способом гарантування незмінності всього блокчейну, так як хеш попереднього блоку включається в наступний блок, а зміна інформації в попередньому блоці відповідно змінює його хеш і він вже починає не співпадати з тим хешем який був включений початково в новий блок, ця властивість дозволяє зберігати цілісність мережі блокчейн. На наступному рисунку зображено загальний вигляд структури блоків у блокчейні.

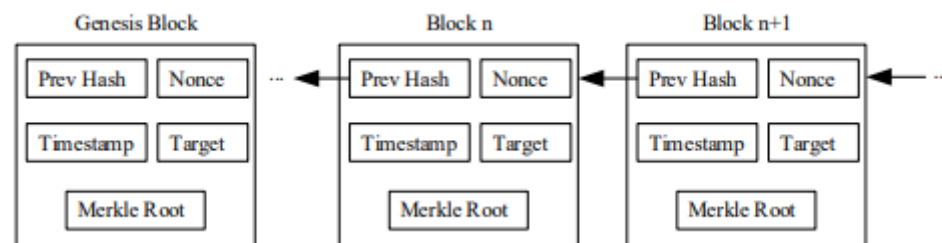


Рис.1.1. Блокчейн структура

Джерело: <https://text2fa.ir/wp-content/uploads/Text2fa.ir-Research-on-the-Application-of-Cryptography.pdf>

Відповідно до рис. 1.1., Genesis Block є первинним блоком в мережі тобто початковим блоком з якого ми починаємо вести облік даних. Потім на основі інформації отриманій при обробці входних даних криптографічними алгоритмами а також при використанні випадкового числа(nonce) ми розраховуємо хеш цього блоку і він включається в хеш нового блоку в поле prev hash(попередній хеш), targert(ціль) — забезпечує стабільну частоту генерації блоків, автоматично перераховується мережею кожні два тижні, визначається

значенням SHA256 у блоку. Timestamp(часова відмітка) — вказує час запису даних блоку і використовується для впорядкування блоків у ланцюгу. [12] У блокчейні також для забезпечення надійності транзакцій використовується асиметричне шифрування яке складається з пар публічного та приватного ключа. Приватний ключ це послідовність символів яким шифрується повідомлення, публічний ключ це спосіб розшифрувати повідомлення, гарантується, що знаючи публічний ключ майже неможливо визначити приватний ключ певного користувача, через складність математичних розрахунків. Відповідно схема даного шифрування виглядає так:

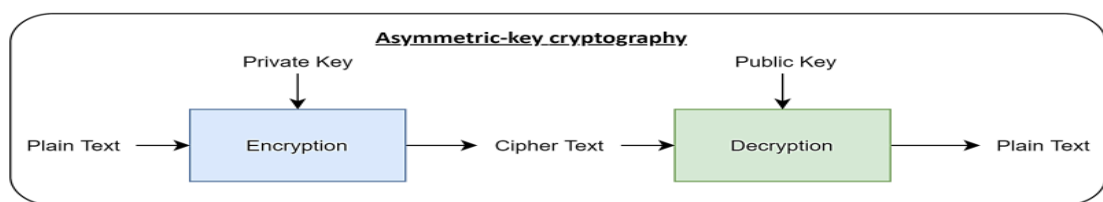


Рис.1.2. Асиметрична криптографія

Джерело: <https://shorturl.at/dkuL4> [13]

Додаючи пояснення до цього рисунку ми можемо бачити на ньому наступне, вхідне повідомлення за допомогою приватного ключа шифрується в цифровий текст, потім за допомогою відкритого публічного ключа це повідомлення розшифровується знову в початковий текст, така криптографія дозволяє нам перевіряти чи дійсно повідомлення було надіслано тим користувачем з яким ми обмінюємось інформацією, адже якщо воно надіслано кимось іншим, то ми не зможемо його розшифрувати публічним ключем особи з якою ми взаємодіємо, відповідно це гарантує те, що кожне повідомлення в мережі можливо ідентифікувати, що відповідно дає нам безпеку транзакцій. І це відповідно є ще однією з переваг використання блокчейну для передачі права власності, тобто високий захист при шифруванні повідомлень, так як математично майже неможливо за публічним ключем отримати приватний.

РОЗДІЛ 2. СТВОРЕННЯ ІНФРАСТРУКТУРИ ДЛЯ ПЕРЕДАЧІ ПРАВА ВЛАСНОСТІ НА НЕРУХОМІСТЬ

2.1. Вибір блокчейн платформи

Щоб реалізувати щось незалежно що це автомобіль, літак, програмний код для передачі права власності на блокчейн потрібна відповідна інфраструктура. Завдяки високій конкуренції у світі блокчейну зараз існує багато способів виконати конкретну бізнес логіку, все залежить лише від конкретних потреб замовника. Проте, що стосується реалізації якогось програмного коду на блокчейні, то нам потрібні такі інструменти як — сам блокчейн, можливість написання смарт-контрактів на ньому, наявність документації для написання нашого коду.

Одними з найчастіше використовуваних блокчейн-платформ для написання смарт-контрактів є Ethereum [15], [16], Hyperledger Fabric [17], Stellar , NEM [19], NEO [20]. Кожна з цих платформ має різні інструменти для написання відповідного програмного коду, так Ефіріум використовує мову Solidity [21], яка є об'єктно орієнтованою мовою програмування та має схожі характеристики та синтаксис з такими мовами як Python, Javascript та C++. Hyperledger Fabric відповідно дозволяє писати смарт-контракти на мовах програмування таких як (GO, JAVA, Javascript, Node.js). Stellar ж так само як і Hyperledger Fabric використовує — GO, JAVA, Javascript. NEM для написання смарт-контрактів використовує мови програмування JAVA та C++. Також NEO дозволяє писати смарт-контракти на таких мовах як C#, Go, Python, Java, або TypeScript. [22]

Щоб обрати платформу для реалізації нашого проекту ми повинні звернути увагу на такі аспекти:

- незмінюваність вихідного коду смарт-контракту;
- висока надійність та безпека;
- масштабованість;
- можливість відправки та отримання коштів;
- відкритість для всіх сторін;
- ціна виконання транзакцій;

швидкість виконання транзакцій в секунду.

У таблиці 2.1., створений порівняльний аналіз на основі деяких вищезгаданих показників.

Таблиця 2.1.

Порівняльний аналіз платформ для смарт-контрактів

Назва Платформи	Ethereum	Hyperledger Fabric	Stellar	NEM	NEO
Характеристика:					
Доступ	Відкритий	Закритий	Відкритий та Закритий	Відкритий та Закритий	Закритий
Швидкість транзакцій (в секунду) [23] [24]	Від 20000 До 100000	Від 1500 До 3000	Від 250 До 1000	3085	Від 1000 До 10000
Можливість відправки коштів	Так	Так	Так	Так	Так
Ціна виконання [25] [26] [27] [28]	0.981 USD *Станом на 17/11/2023	-	0.00002364% Від 1 USD	0,00018USD	Безоплатно, сплачується, якщо користувач хоче пришвидшити транзакцію
Криптовалюта	ETH	-	XLM	XEM	NEO

2.2.Розробка смарт-контракту для запису та передачі права власності

Перед розробкою самого смарт-контракту потрібно описати інфраструктуру яка повинна бути реалізована в ньому, відповідно враховуючи те, що нам потрібно підтверджувати справжність документів користувачів, а робитимуть це відповідно нотаріуси(валідатори), то спочатку нам потрібно додати в наш смарт-контракт валідаторів, проте ми не можемо дозволити, щоб валідатор був тією особою хто володіє смарт-контрактом, так як смарт-контракт також прийматиме податкові надходження, а потім власники (адміни) цього смарт-контракту будуть знімати дані податки з данного смарт-контракту. Це означає, що ми повинні мати 3 наступні ролі в нашому смарт контракті та вони повинні мати відповідні права та обов'язки, які будуть наведені в наступній таблиці.

Таблиця 2.2.

Рольова модель Смарт-контракту

Роль	Права та обов'язки
Адміни	Керують смарт-контрактом, додають та видаляють валідаторів і адміністраторів, управляють коштами з податків.
Валідатори	Проводять верифікацію користувачів та їх документів, надають дозволи на внесення об'єктів нерухомості в смарт-контракт, слідкують за належною сплатою податків. Встановлюють мінімальний розмір податків для кожної нерухомості згідно оцінкою майна експертами
Користувачі	Реєструються, підтверджують свою особистість, обмінюються власністю на нерухомість

Зазначимо, що валідатори зможуть робити всі ті ж дії, що і користувачі, проте враховуючи те, які вони права матимуть, ми повинні забезпечити, що валідатор не може підтвердити самого себе, а отже це означає, що ми повинні включити таке обмеження в наш програмний код, щоб валідатор не міг цього робити і таким чином зловживати своїми посадовими обов'язками. Важливо також для кожного валідатора вказувати його ліцензійний номер, по якому кожна бажуюча особа могла б перевірити чи правильно все виконує даний валідатор. Відкритість дій спонукатиме кожного з валідаторів, до меншого шахрайства та підвищуватиме його чесність.

Важливо також описати схему за якою буде відбуватись передача власності нерухомості у блокчейні та як вона відбуватиметься. Спочатку користувач повинен додати нерухомість смарт-контракт але зробити він може це тільки, якщо отримає документ від власника, тобто дозвіл на додавання нерухомості це в свою чергу створює додаткові логічні процедури, проте це є обов'язковим, так як без згоди власника неможливо вносити його нерухомість в базу даних смарт-контракту.

Критично важливим є зазначити те, що при додаванні нерухомості яка є у спільній власності декількох осіб, відповідно до статті 368 [4], про право спільної сумісної власності, одним майном можуть володіти декілька осіб або його часткою, це означає, що користувач який бажає щоб ми додали його нерухомість до блокчейну, повинен отримати згоду від інших співвласників, якщо він не є одноосібним власником, таким чином важливо при первинному внесенні валідаторами власників, правильно зареєструвати їх, цілком і повністю, так як співвласники мають право першочергової купівлі частки іншого співвласника відповідно статті 362 [4].

Ще один важливий аспект це встановлення оціночної вартості, так як оподаткування буде проводитись згідно від цієї сумми якщо сума продажу, є меншою ніж дана оцінка, це визначено в податковому кодексі України та за посиланням [3].

Також в тому самому ж законі свідчиться, що оцінка майна діє протягом 6 місяців, що означає наступне, користувачі повинні слідкувати за термінами встановлення нотаріусами оціночної власності в блокчейні, а саме за датою встановлення, щоб вчасно та оперативно оновлювати це значення задля уникнення можливих юридичних проблем.

Для такого завдання як передача права власності на нерухомість також важливо схематично відобразити як все відбуватиметься, тому далі зображена схема процесу передачі власності на рис. 2.3., від ініціалізації смарт контракту, до продажу власності.

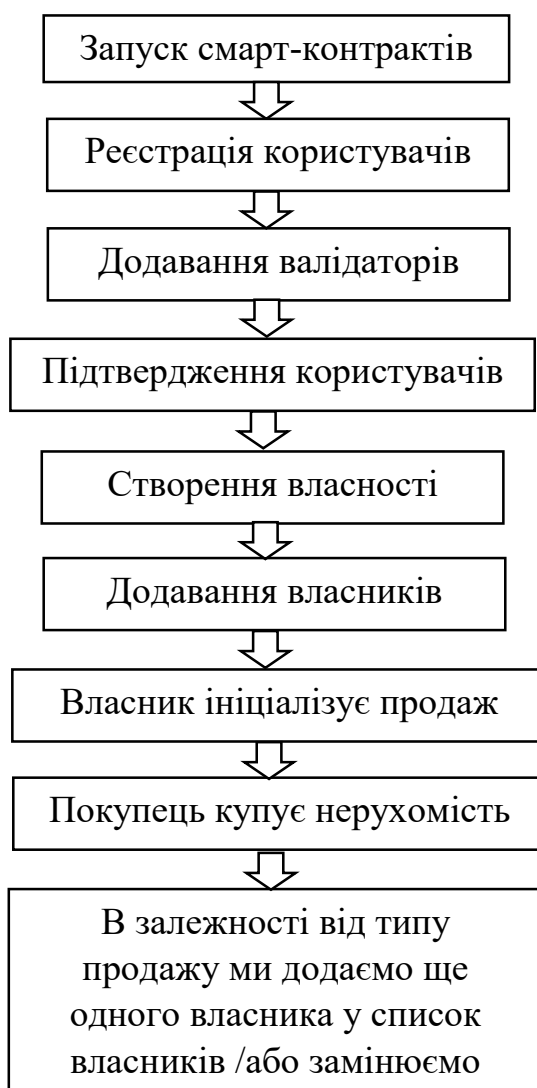


Рис.2.3. Етапи реалізації передачі права власності

Відповідно до вищезгаданої схеми та користуючись особливостями мови програмування Solidity, ми створили наступний контракт, наведений у додатку Б, який ініціалізує 4х адмінів (при реальному запуску це значення може бути

змінено), потім потенційний користувач через відповідну функцію проходить реєстрацію, і як відомо спочатку ми повинні зареєструвати валідаторів, саме для цього була створена функція додавання ліцензійного номеру і якщо адміністратори погоджуються з тим, що його ліцензійний номер валідний, то вони додають цього валідатора до системи і він вже відповідно може здійснювати свою діяльність.

Після того як ми зареєстрували валідаторів, завдяки функціям зображеним на рисунках 2.3. та 2.4., вони вже мають змогу підтверджувати інших користувачів, на основі надісланих ними документів, адже від статусу користувача залежать можливості в використанні смарт-контракту!, непідтверджені користувачі не повинні мати змогу користуватись смарт-контрактом. Так як необхідно вносити користувача в базу даних (де одним із ключів буде його адреса криптогаманця), тобто необхідно зберігати в закритій БД (до якої матимуть доступ лише адміністратори та валідатори) дані про кожного користувача, надіслані ним заяви, документи, а також його адресу криптогаманця. Так як в майбутньому по данній адресі криптогаманця ми зможемо відслідкувати які дії вчинив той чи інший користувач, зберігаючи при цьому інформацію про справжнє ім'я та дані користувача анонімними у блокчейні.

```
function addLicenseNumber(uint256 _licenseNumber) external {
    require(
        !usedLicenseNumbers[_licenseNumber],
        "License number already used"
    );
    require(
        users[msg.sender].licenseNumber == 0,
        "User already has a license number"
    );

    users[msg.sender].licenseNumber = _licenseNumber;
    usedLicenseNumbers[_licenseNumber] = true;
}
```

Рис. 2.4. Додавання потенційним валідатором ліцензійного номеру

Завдяки використанню цієї функції потенційний валідатор встановлює собі ліцензійний номер і якщо він дійсно відповідає тому документу який він надіслав адміністраторам на перевірку то адміністратори через функцію зображену на рис. 2.5. виконують підтвердження данного валідатора.

```
function approveValidator(address _userAddress) external onlyAdmins {
    require(users[_userAddress].id != 0, "User not found");
    require(
        users[_userAddress].licenseNumber != 0,
        "License number not set"
    );
    require(msg.sender != _userAddress, "Admin cannot approve themselves");
    require(
        !isAddressInList(msg.sender, users[_userAddress].adminsSigners),
        "Admin already signed for this Validator"
    );

    users[_userAddress].adminsSigners.push(msg.sender);
    users[_userAddress].adminConfirmationCount++;

    if (users[_userAddress].adminConfirmationCount >= 3) {
        users[_userAddress].isValidator = true;
        users[_userAddress].isVerified = true;

        validators[_userAddress] = true;
        validatorLicenseNumber[_userAddress] = users[_userAddress].licenseNumber;

        verifiedUsers[_userAddress] = true;

        delete users[_userAddress].adminsSigners;
    }
}
```

Рис. 2.5. Підтвердження валідатора

Для реалізації нашого проекту ми використовували здебільшого характеристики мапування в мовах програмування. Згідно з автором в статті [30] Мапування в перекладі на програмні мови втілюють інтерфейси, які, в основному, дозволяють: вставляти пари, отримувати або оновлювати значення, використовуючи ключ як параметр, а також видаляти пари, також використовуючи ключ як параметр. Тобто характеристика мапувань дозволяє нам створювати унікальні пари значень, такі як мапування тих хто є валідатором , а хто ні.

Після реєстрації користувачів у системі валідатори на основі наданих ними документів, які підтверджують їх власність, вносять нерухомість та конкретного власника до спеціального маппінгу який відповідає за збереження структур власності (така структура зокрема зображена на рис. 2.6), в який подальшому валідатори додадуть початкових власників, які вже відповідно зможуть розпорятись своїм нерухомим майном згідно власних потреб. Проте як вже згадувалось раніше, продаж власності, неможливий без встановлення мінімальної ціни, саме тому ми в нашому смарт-контракті, який успадковує деякі функції першого контракту, зробили таку умову(зображену на рис. 2.7.), що продати нерухомість неможливо без встановлення оціночної вартості продажу, що вимагає від користувача надавати нотаріусу документ про оціночну вартість, відповідно якого встановлюється сума оподаткування. Це дозволяє нам контролювати податкові надходження до нашого смарт-контракту та зберігати відповідність вимогам законодавства.

```
struct AddProperty {
    string location;
    string propertyType;
    string description;
    address[] owner;
    uint256 estCost;
```

Рис.2.6. Структура нерухомості

Ця структура зберігає такі дані як місцезнаходження нерухомості, тип нерухомості, опис нерухомості, масив власників та оціночну вартість нерухомості.

```
require(
    realEstates[_registrationNumber].estCost > 0,
    "Minimum cost isn't installed"
);
```

Рис.2.7. Обмеження встановлення нерухомості на продаж при відсутності встановлення мінімальної ціни

На рис. 2.7., можна бачити оператор `require` який згідно інформації наведених в статті [36] означає наступне — оператори `'require'` визначають умови, які повинні бути виконані перед виконанням функції, тобто вони встановлюють обмеження, які мають бути виконані перед виконанням коду. У випадку наведеному на рис. 2.6., ми вимагаємо щоб мінімальна оціночна вартість була більшою за нуль іншими словами перевіряємо таким чином чи вона встановлена. Перевірка зображена на рис. 2.7., використовується зокрема у функції представленої у рис. 2.8., яка в свою чергу дозволяє користувачеві виставляти свою нерухомість на продаж вказуючи ціну продажу та адресу бажаного покупця, що в свою чергу дозволяє оперативно виконувати операцію з нерухомістю тільки тій особі з якою продавець домовився про угоду. Зазначимо, що враховуючи деякі технічні особливості проектування дана функція дозволяє продавати лише конкретну нерухомість за один раз, що означає, що продаж кожної нової нерухомості повинен бути проведений лише після завершення минулого продажу

```

function setupSellSingle(
  uint _registrationNumber,
  address _buyer,
  uint _sellType,  infinite gas
  uint _price
) external {
  require(
    _sellType == 1 || _sellType == 2,
    "1 for full sale, 2 for part sale"
  );
  require(
    realEstates[_registrationNumber].owner.length == 1,
    "Not a single owner"
  );
  require(
    realEstates[_registrationNumber].estCost > 0,
    "Minimum cost isn't installed"
  );
  require(userRegistration.checkVerifiedUsers(msg.sender), "Not a verified user!");
  require(userRegistration.checkVerifiedUsers(_buyer), "Not a verified buyer!");
  require(
    userRegistration.isAddressInList(msg.sender, realEstates[_registrationNumber].owner),
    "You are not the owner"
  );

  singleOwnerSell[msg.sender] = PutForSellSingle({
    registrationNumber: _registrationNumber,
    countSigns: 0,
    sellerSigns: new address[](0),
    buyer: _buyer,
    sellType: _sellType,
    price: _price * 1 ether
  });
}

```

Рис. 2.8. Виставлення нерухомості на продаж

Дана функція працює наступним чином вона приймає на вхід такі параметри як реєстраційний номер, що слугує унікальним ідентифікатором нерухомості, ціну продажу, бажаного покупця та тип продажу. Тип продажу ми вирішили визначити як числа 1 або 2, де 1 це повноцінний продаж, що означає, що власність повністю переходить до нового власника, а 2 це частковий продаж, де теперішній власник також залишається володарем данної нерухомості але і тепер туди додається покупець данної нерухомості. Ми використовуємо тут мультиплікацію в 1 ether для приведення вартості нерухомості в ціну в ефірах а не в GWEI (GWEI найменше представлення одиниці виміру в блокчейні Ефіріум [37])

Ми також при розробці нашого смарт-контракту скористались бібліотекою SafeMath [32] від OpenZeppelin яка дозволяє коректно проводити математичні операції в смарт-контрактах побудованих на ефіріум. Сама ж реалізація даної бібліотеки зображена на рис. 2.9. та 2.10.

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

library SafeMath {

    function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            uint256 c = a + b;
            if (c < a) return (false, 0);
            return (true, c);
        }
    }

    function trySub(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            if (b > a) return (false, 0);
            return (true, a - b);
        }
    }

    function tryMul(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
            // benefit is lost if 'b' is also tested.
            // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
            if (a == 0) return (true, 0);
            uint256 c = a * b;
            if (c / a != b) return (false, 0);
            return (true, c);
        }
    }

    function tryDiv(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            if (b == 0) return (false, 0);
            return (true, a / b);
        }
    }
}
```

Рис. 2.9. Код бібліотеки SafeMath від OpenZeppelin

```

function tryMod(uint256 a, uint256 b) internal pure returns (bool, uint256) {
    unchecked {
        if (b == 0) return (false, 0);
        return (true, a % b);
    }
}

function add(uint256 a, uint256 b) internal pure returns (uint256) {
    return a + b;
}

function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    return a - b;
}

function mul(uint256 a, uint256 b) internal pure returns (uint256) {
    return a * b;
}

function div(uint256 a, uint256 b) internal pure returns (uint256) {
    return a / b;
}

function mod(uint256 a, uint256 b) internal pure returns (uint256) {
    return a % b;
}

function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
    unchecked {
        require(b <= a, errorMessage);
        return a - b;
    }
}

function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
    unchecked {
        require(b > 0, errorMessage);
        return a / b;
    }
}

function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
    unchecked {
        require(b > 0, errorMessage);
        return a % b;
    }
}

```

Рис. 2.10. Продовження коду зображеного на рис.2.9.

З метою запобігання переповнення або недостатку у цілочисельних операціях у смарт-контрактах, написаних на мові Solidity, розробники пропонують включати бібліотеку SafeMath в смарт-контракт. З 479 записів у CVE, 78% смарт-контрактів виконують арифметичні операції без використання бібліотеки SafeMath. Решта 22% використовують бібліотеку SafeMath, проте вони все ще стикаються з арифметичними проблемами через неодноразове використання математичних операцій у контрактах. Крім того, всі смарт-контракти безпосередньо включають арифметичні операції, визначені в бібліотеці SafeMath, у вміст своїх контрактів, замість використання ключового слова `import` для імпорту бібліотеки.[38] Вартість розгортання смарт-контракту з інтеграцією бібліотеки SafeMath збільшується в середньому на приблизно 228 газових одиниць, оскільки код бібліотеки збільшує розмір смарт-контракту. Однак його вартість транзакції зменшується в середньому на приблизно 228 газових одиниць завдяки цій інтеграції. Це зменшення є результатом передачі деяких

математичних операцій до бібліотеки SafeMath. Хоча збільшення вартості розгортання є значущим при інтеграції бібліотеки SafeMath у смарт-контракт, цю вартість сплачують лише один раз під час розгортання смарт-контракту. Натомість вартість транзакції стягується при кожній виконаній транзакції. Таким чином, вартість експлуатації є нижчою.[38]

Важливо також вказати за якими кроками буде відбуватись процес продажу згідно запропонованого нами рішення. Загалом процес безпосередньо продажу можна описати наступними кроками:

1. користувач надсилає документи, які підтверджують його власність та частку в власності на дану нерухомість;
2. валідатори на основі отриманих документів, реєструють нерухомість та вносять туди користувача;
3. користувач бажає ініціалізувати продаж, звертається до експертів з оцінки нерухомого майна;
4. користувач надає цю оцінку нерухомого майна валідатору;
5. валідатор вносить дані про оцінку майна в відповідне поле;
6. користувач виставляє нерухомість на продаж;
7. покупець купує нерухомість і в залежності від типу продажу стає повноцінним власником.

Варто зазначити, що задля економії місця та пам'ятю варто було пожертувати обсягом контракту, що в свою чергу призвело до наступних наслідків:

1. неможливості продажу двох різних нерухомостей за один раз;
2. відповідно користувач повинен завершити спочатку дії купівлі-продажу з однією нерухомістю, перед тим як перейти до продажу іншої.

2.3. Реалізація безпеки та відповідності до правових норм

Безпека створених нами смарт-контрактів заснована на ретельному тестуванні кейсових випадків, які можуть статись під час виконання смарт-контракту, такі перевірки включають чи правильно працюють перевірки

обмежень(“require”) [21], одним з прикладів таких перевірок, є перевірка на те чи не намагається користувач сам себе підтвердити або чи не намагається покупець придбати ту нерухомість яка виставлена на продаж не для нього. Також було встановлено обмеження для повторного додавання одного і того ж реєстраційного номеру(який виступає ключем по якому ми отримуємо характеристику про ту чи іншу власність) в мапування яке представляє нерухомість, щоб уникнути ситуацій перезаписання вже внесених даних.

Що стосується ж правових норм, то тут трохи складніше, так як користувачі повинні самі слідкувати за датою встановлення оціночної вартості і продавати нерухомість тільки у тому випадку, якщо вони впевнені що дата продажу не є пізнішою ніж створення оціночної вартості для конкретної нерухомості, а відповідно, після закінчення терміну дії цього документу про оціночну вартість, користувач повинен надати валідаторам нове свідоцтво про теперішню оціночну вартість майна і валідатор відповідно повинен оновити оціночну вартість згідно з наданим документом. Дотримання встановлених термінів та обов'язковість надання актуальних свідоцтв є ключовими аспектами у цьому процесі, сприяючи надійності та прозорості передачі права власності через блокчейн.

Як зазначено авторами статті [31] основними вразливостями смарт-контрактів є:

- Reentrancy - це ситуація, коли контракт А викликає функцію контракту В, який, в свою чергу, може знову викликати функцію контракту А. Цей механізм Solidity, який викликається за замовчуванням, може призводити до неочікуваних переказів Ether без необхідного контролю. Під час атаки, зловмисник викликає функцію `withdraw()` у контракті А, з метою отримання Ether. Контракт А використовує `call` для відправлення Ether, і функція за замовчуванням атакувальника знову викликає `withdraw` у контракті А, що може призвести до непередбачуваних переказів Ether та викликів функцій контракту. Це може призвести до великої кількості непередбачуваних переказів Ether та викликів функцій, і зловмисник може відкласти атаку до досягнення бажаного стану;

```

contract A {
    mapping (address => uint) public balances;
    function withdraw(uint _amount) public {
        msg.sender.call{value: _amount}('');
        balances[msg.sender] -= _amount;
    }
    /*Other codes*/
}

contract Attacker {
    A public a;
    fallback() external payable {
        a.withdraw(1 ether);
    }
    function attack(){
        a.withdraw(1 ether);
    }
}

```

Рис.2.11. Вразливий код до атаки повторного входу

Як вказано авторами є три методи для запобігання атакам рекурсивного виклику, а саме обмеження виклику функцій (за допомогою таких модифікаторів як `onlyAdmin` або `onlyValidator` наприклад). Використання функції `transfer` для відправлення Ether іншим рахункам. Ця функція відправляє лише 2300 газових одиниць із зовнішніми викликами, тому викликаний контракт не матиме достатньо газу для повторного входу в викликаючий контракт. (газ відноситься до одиниці, яка вимірює кількість обчислювальних зусиль, необхідних для виконання конкретних операцій в мережі Ethereum. [31]). Другий спосіб це додавання блокування на повторний виклик функції, до завершення дії попереднього виклику. Третій - здійснення всіх змін логіки перед викликом функції.

- Проблеми з арифметикою. У мові Solidity цілочисельні типи мають конкретний діапазон. Під час арифметичних операцій змінні можуть виходити за верхню чи нижню межу. У разі такої ситуації значення буде зміщено на інший бік межі. Задля уникнення неочікуваних результатів при роботі з арифметичними операціями рекомендується використовувати модуль `SafeMath` [32] від OpenZeppelin.

- Delegatecall - це спеціальна функція в Solidity, яка виконує код викликаного контракту в контексті викликаючого, зберігаючи при цьому його контекст. Використання delegatecall може призвести до небезпечних сценаріїв, таких як неочікувана модифікація змінних стану контракту. Це може створювати ризики для безпеки контрактів, тому слід бути обережним та враховувати цей аспект при розробці.

```
contract VulernableContract {
    address public owner;
    InsecureContract public c;
    constructor(InsecureContract _c) public {
        owner = msg.sender;
        c = InsecureContract(_c);
    }
    /*Other codes...*/
    c.delegatecall(abi.encodeWithSignature('doSomething()'));
    /*Other codes...*/
}

contract InsecureContract {
    address public owner;
    function doSomething() public {
        /*Other codes...*/
        owner = msg.sender;
        /*Other codes...*/
    }
}
```

Рис.2.12. Вразливий код до небезпеки delegatecall

Протидія — оскільки DELEGATECALL використовує зміну оточення смарт-контрактів, першим етапом є перевірка потенційних оточень контрактів та бібліотек, які їх можуть викликати (чи інших контрактів).

- Заморозка Ефіру. Існують ситуації коли контракт може лише приймати кошти на свою адресу але не може відправляти з цієї адреси на іншу тому важливо використовувати створення функцій для виведення грошей з контракту, ще на етапі його створення.

Щоб успішно протидіяти цим та іншим вразливостям, варто дотримуватись рекомендацій які надаються для подолання таких вразливостей, важливо використовувати сучасні методи аналізу коду , смарт-контракту, проводити незалежну аудиторську перевірку, тощо. Відповідно можна використати такий інструмент як slither [33]. Slither - це інструмент для аналізу безпеки розумних

контрактів на блокчейні Ethereum. Він призначений для виявлення потенційних уразливостей, аномалій і можливих проблем у смарт-контрактах, написаних мовою Solidity.

Slither допомагає розробникам та аудиторам забезпечувати високий рівень безпеки для їхніх смарт-контрактів, виявляючи можливі проблеми перед розгортанням на блокчейні Ethereum.

Ми також використали задля безпеки в нашому смарт-контракті наступні перевірки require:

```
require(
    adminsUsersMapping[_adminToToggle].adminAddress != address(0),
    "Admin user not registered"
);
```

Рис 2.13 Перевірка чи адмін якого намагаються звільнити інші адміни є в мапуванні адмінів

Дана перевірка використовується в коді функції яка відповідає за звільнення адміністраторів, так як ми розуміємо, що скоріш за все , якщо даний проект і буде реалізовано то він скоріш за все випускатиметься від імені державних структур та установ, адміністраторами відповідно будуть держдіячі і не дивлячись на високі стандарти безпеки криптогаманців ми не застраховані від того, що один з адміністраторів навмисно або випадково дискредитує свій приватний ключ, а це означає, що ми повинні мати змогу дати іншим адмінам звільнити того адміністратора приватний ключ якого був скомпрометований. Задля цього ми створили функцію зображену на рис. 2.13., яка дозволяє як і додавати нового адміністратора, так і звільняти існуючого якщо його дані були скомпрометовані.

В загальному ця функція призначена для перемикання статусу адміністратора в системі. Вона може бути викликана тільки іншим адміністратором, оскільки вона має модифікатор "onlyAdmins", що перевіряє, чи адреса викликаючого є адміністратором. Перевіряється, чи адміністратор, якого потрібно перемкнути, вже зареєстрований в системі. Якщо він не зареєстрований, генерується відповідне повідомлення про помилку. Далі перевіряється, чи той, хто викликає функцію, не підписав вже заяву про підтвердження того самого

адміністратора, щоб уникнути повторних підписів. Якщо адміністратор, якого потрібно перемкнути, не є адміністратором на момент виклику функції, тоді збільшується лічильник підтверджень та додається адреса того, хто викликав функцію, до списку осіб, які підтримують зміну статусу. Якщо кількість підтверджень стає рівною або більшою за 3, тоді статус адміністратора змінюється на "true", і список підтримуючих адміністраторів очищується. Якщо адміністратор вже є адміністратором на момент виклику функції, тоді зменшується лічильник підтверджень та додається адреса того, хто викликав функцію, до списку осіб, які відмовились від підтримки зміни статусу. Якщо лічильник підтверджень стає рівним нулю, тоді статус адміністратора змінюється на "false", і список підтримуючих адміністраторів очищується. Ця функція дозволяє адміністраторам системи підтверджувати або відмовлятися від зміни статусу іншого адміністратора.

```

function toggleAdminStatus(address _adminToToggle) external onlyAdmins {
    require(
        adminsUsersMapping[_adminToToggle].adminAddress != address(0),
        "Admin user not registered"
    );
    require(
        !isAddressInList(
            msg.sender,
            adminsUsersMapping[_adminToToggle].adminsSigners
        ),
        "That admin has already signed"
    );

    if (!admins[_adminToToggle]) {
        adminsUsersMapping[_adminToToggle].confirmationsCount++;
        adminsUsersMapping[_adminToToggle].adminsSigners.push(msg.sender);

        if (adminsUsersMapping[_adminToToggle].confirmationsCount >= 3) {
            admins[_adminToToggle] = true;
            delete adminsUsersMapping[_adminToToggle].adminsSigners;
        }
    } else {
        adminsUsersMapping[_adminToToggle].confirmationsCount--;
        adminsUsersMapping[_adminToToggle].adminsSigners.push(msg.sender);

        if (adminsUsersMapping[_adminToToggle].confirmationsCount == 0) {
            admins[_adminToToggle] = false;
            delete adminsUsersMapping[_adminToToggle].adminsSigners;
        }
    }
}

```

Рис. 2.14. Додавання та звільнення адміністраторів

Як вже було зазначено раніше ми також використали спеціальні модифікатори для обмеження несанкціонованого доступу до деяких з функцій, дані модифікатори зображені на рис.2.14., та 2.15.,

```

modifier onlyAdmins() {
    require(admins[msg.sender], "Only admins can call this function");
    _;
}

```

Рис. 2.15. Модифікатор який використовується для надання доступу до деяких функцій лише адміністраторам

Цей модифікатор забезпечує, що функції, до яких він застосований, можуть бути викликані тільки тими, хто має статус адміністратора в системі.

Основна ідея модифікатора полягає в тому, що він додає додаткову перевірку перед викликом функції. При виклику функції, яка використовує цей модифікатор, відбувається наступне:

- викликаючий повинен бути зареєстрованим адміністратором. якщо це не так, генерується відповідне повідомлення про помилку;
- якщо викликаючий є адміністратором, то функція, яку модифікатор захищає, буде викликана. у цьому випадку `_;` вказує на те, що управління передається до тіла самої функції.

Іншими словами, цей модифікатор додає умову перевірки перед викликом функції, і функція буде викликана лише в тому випадку, якщо умова виконується, тобто якщо викликаючий є адміністратором. Це дозволяє обмежити доступ до функцій тільки для адміністраторів.

```
modifier onlyValidator() {
    require(
        validators[msg.sender],
        "Only validator can call this function"
    );
    _;
}
```

Рис. 2.16. Модифікатор який використовується для надання доступу до деяких функцій лише валідаторам

Модифікатор наведений на рис. 2.15., працює точно таким ж способом як і той що наведений на рис. 2.14., з однією лише відмінністю у тому, що цей модифікатор дає доступ до функцій валідаторам а не користувачам. Ми також створили окремий смарт-контракт який відповідає за оподаткування в нашій системі. Даний смарт контракт має наступні функції та методи — `setMaxSellsCount` рис. 2.16., `getMaxSells` рис.2.17., `calculateTax` рис. 2.18., `setTaxRate` рис. 2.19.

```
function setupMaxSellsCount (uint _sellsCount) external {
    require(propertyRegistrationAdmins.checkAdmin(msg.sender));
    maxSellsCount = _sellsCount;
}
```

Рис. 2.17. Функція встановлення максимального числа розрахунку кількості продаж

Так як після 3х продажів нерухомості податок на нерухомість згідно податкового законодавства [6] становить 18% і потім вже не збільшується, то за допомогою даної функції ми встановлюємо обмеження щоб кількість продаж продавця не збільшувалась більше встановленого ліміту у даній функції, тому що кількість продаж виступає ключем до встановлення відсотку податку. І також за допомогою цієї функції ми блокуємо встановлення значення ключа для оподаткування таким чином щоб він не перевищував число записане у змінну maxSellsCount.

```
function getMaxSells() external view returns (uint256) {
    return maxSellsCount;
}
```

Рис. 2.18. Функція перевірки того яка кількість продажів була встановлена як максимальний поріг оподаткування

При виклику цієї функції повертається значення змінної maxSellsCount, яке представляє максимальну кількість продажів для оподаткування визначеній в функції на рис. 2.16.

```
function calculateTax(uint256 _value, uint256 _taxKey) external view infinite
returns (uint256)
{
    // Використання SafeMath для уникнення переповнень та точних обчислень
    return _value.mul(taxRate[_taxKey]).div(100);
}
```

Рис. 2.19. Функція для розрахунку розміру податку

Ця функція `calculateTax` призначена для обчислення суми податку на основі переданої вартості та ключа податку. Вона використовує `SafeMath`, щоб уникнути можливих переповнень чи втрати точності при арифметичних операціях. Функція не змінює стан контракту і є лише читацькою (`view`). Її результат - це обчислена сума податку, яку можна використовувати в інших частинах смарт-контракту або викликаючому коді. Функція `calculateTax` отримує два параметри: `_value` (вартість) та `_taxKey` (ключ податку). Звертаючись до мапи `taxRate` за ключем `_taxKey`, вона отримує ставку податку для даного ключа. Потім вона використовує `SafeMath` для множення `_value` на отриману ставку та ділення на 100, щоб отримати відсоткове значення.

```
function setTaxRate(uint256 _quantitySales, uint256 _newTaxRate) external {
    require(propertyRegistrationAdmins.checkAdmin(msg.sender));
    require(_quantitySales <= maxSellsCount);
    // Додайте логіку перевірки прав доступу, якщо потрібно
    taxRate[_quantitySales] = _newTaxRate;
}
```

Рис. 2.22. Функція яка встановлює розмір податку

Функція `setTaxRate` дозволяє адміністраторам встановлювати ставки податків для конкретної кількості продажів. При виклику цієї функції спершу перевіряється, чи викликаючий має права адміністратора, оскільки лише адміністратори можуть змінювати ставки податків. Після цього виконується друга перевірка, яка стосується максимальної кількості продажів. Це важливо для

уникнення неправомірних змін податкових ставок, які можуть вплинути на фінансовий стан контракту.

Якщо обидві перевірки пройдені успішно, функція встановлює нову ставку податку для вказаної кількості продажів. Це може бути важливо в контексті регулювання оподаткування в залежності від обсягу бізнес-операцій. Адміністратори можуть динамічно налаштовувати ставки податків, щоб адаптувати їх до змін в бізнес-стратегії чи економічних умов. Такий підхід дозволяє гнучко керувати оподаткуванням, забезпечуючи при цьому стабільність фінансових операцій контракту. Смарт-контракт написаний для реалізації розрахунку системи оподаткування є елементом системи обміну нерухомістю з податковою логікою на блокчейні. Він використовує SafeMath для уникнення переповнень та точних обчислень. Контракт має можливість встановлення максимальної кількості продажів (`maxSellsCount`), а також зберігає ставки податків (`taxRate`) для різної кількості продажів.

Адміністратори, які визначаються за допомогою іншого контракту наведений у Додатку Б, можуть встановлювати максимальну кількість продажів і ставки податків. Функція `setupMaxSellsCount` дозволяє адміністраторам встановлювати максимальну кількість продажів, тоді як функція `setTaxRate` дозволяє встановлювати ставки податків в залежності від кількості продажів.

Крім того, є функція `getMaxSells`, яка повертає поточну максимальну кількість продажів. Функція `calculateTax` призначена для розрахунку податків з урахуванням вартості та визначеної ставки податку для конкретного ключа.

Загалом, смарт-контракт дозволяє динамічно налаштовувати оподаткування обміну нерухомістю в залежності від кількості продажів та вартості операцій, що забезпечує гнучкість та адаптивність в управлінні фінансовими параметрами системи.

РОЗДІЛ 3. ПРОПОЗИЦІЇ ЩОДО ВИКОРИСТАННЯ ТЕХНОЛОГІЇ БЛОКЧЕЙН В СФЕРІ ПЕРЕДАЧІ ПРАВА ВЛАСНОСТІ НА ОСНОВІ ПРОВЕДЕНОГО ДОСЛІДЖЕННЯ

3.1. Пропозиції щодо створення повноцінної інфраструктури на основі проведеного дослідження

Під час розробки одного із наших смарт-контрактів, завдяки функції зображений на рисунку 3.1., нам вдалось успішно проводити реєстрацію користувачів.

```
function registerUser(
    string memory _name,
    string memory _surname  infinite gas
) external {
    require(users[msg.sender].id == 0, "User already registered");
    uint256 userId = generateUserId(_name, _surname, msg.sender);

    users[msg.sender] = User({
        id: userId,
        name: _name,
        surname: _surname,
        validatorConfirmationCount: 0,
        validatorsSigners: new uint[](0),
        isVerified: false,
        adminConfirmationCount: 0,
        adminsSigners: new address[](0),
        isValidator: false,
        licenseNumber: 0
    });
}
```

Рис. 3.1. Код який відповідає за реєстрацію користувачів

Завдяки коду цього смарт-контракту користувач може внести себе в систему і потім очікувати на підтвердження від валідаторів, щоб в майбутньому через функцію, яка зображена на рисунку 3.2. валідатор міг його підтвердити та він був допущений до участі в процесі обміну нерухомістю. Відповідно ім'я та прізвище яке вводить користувач не обов'язково повинно бути реальним так як в самому блокчейні ми пропонуємо не використовувати реальних імен користувачів, задля збереження конфіденційності згідно закону України про захист персональних даних. [34]

```

function verifyUser(address _userAddress) external onlyValidator {
    require(users[_userAddress].id != 0, "User not found");
    require(
        msg.sender != _userAddress,
        "Validator cannot modify own status"
    );
    require(
        !isNumberInList(
            users[msg.sender].licenseNumber,
            users[_userAddress].validatorsSigners
        ),
        "Validator already signed for this user"
    );

    users[_userAddress].validatorConfirmationCount++;
    users[_userAddress].validatorsSigners.push(
        users[msg.sender].licenseNumber
    );

    if (users[_userAddress].validatorConfirmationCount >= 3) {
        users[_userAddress].isVerified = true;

        verifiedUsers[_userAddress] = true;
    }
}

```

Рис. 3.2. Підтвердження користувача валідаторами

Як показує робота цього коду користувачу потрібно три підтвердження від трьох різних валідаторів, що забезпечує мінімізацію уникнення ситуацій коли якась особа може домовитись з одним з валідаторів про внесення її в систему на основі неправдивих даних.

Так як підтвердження користувачів повинно відбуватись так чи інакше на основі відправлених користувачем документів, то ми рекомендуємо додатково створити рішення яке дозволить приймати документи від користувачів через відповідну архітектуру, таку як веб інтерфейс та закриті бази даних з ключами доступу у адміністраторів та валідаторів, куди користувачі могли б надсилати свої документи.

Смарт-контракт, наведений у Додатку В дозволяє проводити операції з нерухомістю, зокрема подавати заявки на реєстрацію нерухомості, реєструвати безпосередньо нерухомість, додавати власників нерухомості, купувати та продавати нерухомість, завдяки функціоналу створеному у даному контракті, нам вдалось отримувати оплату податку також на цей контракт. У даному смарт-контракті ми імпортували також інший створений нами контракт, в якому

визначені ставки податків для правильного розрахунку розміру оподаткування при операціях з нерухомістю в залежності від кількості продаж нерухомості за терміни визначені в податковому кодексі. [6] Проте під час проведення нашого дослідження ми також виявили, що автоматичне встановлення і перевірка того чи проходить трансфер власності в межах поточного податкового року може суттєво збільшувати комісійні витрати під час здійснення трансферу власності але з іншого боку не включення автоматичного визначення того в який податковий період відбувається трансфер власності вимагає постійного обнулення кількості продажів для кожного учасника обміну нерухомості валідаторами на початку кожного наступного звітного року, що в свою чергу створюватиме велике навантаження на валідаторів в цей період.

Згідно проведеного нами дослідження реалізація передачі права власності, через блокчейн, зменшує кількість документообігу та може пришвидшити передачу власності від однієї особи до іншої. Проте чинні правові обмеження всеодно заважають частій змінні (частіше ніж півроку) обміну власності без посередників, оскільки потрібно здійснювати оцінку нерухомості кожні півроку

Для повноцінної реалізації даного проекту ми рекомендуємо використання існуючих або створення нових датацентрів які будуть у себе зберігати інформацію, про користувачів, адміністратори видаватимуть ключі доступу до цього датацентру для валідаторів, завдяки чому вони могли б перевіряти документи від потенційних користувачів. Як зазначають автори у статті [35] бази даних дозволяють доступ лише авторизованим користувачам, забезпечуючи тим самим безпеку даних. Користувачам та групам користувачів надаються номери облікових записів та паролі. Підсистема безпеки та авторизації Системи управління базами даних відповідає за збереження безпеки даних. База даних забезпечує різноманітні структури зберігання та техніки оптимізації запитів для швидкого витягування даних. Вона зберігається на диску, тому підтримуються ефективні техніки пошуку, такі як індекси.

Для забезпечення рівномірного розподілення навантажень на валідаторів ми рекомендуємо щоб кожен валідатор обробляв лише ті завдання які були

встановлені для нього згідно розподіленню адміністраторами валідаторів по регіонам чи іншим показникам.

3.2. Пропозиції щодо забезпечення повноцінної перевірки смарт-контракту та відповідності його до норм права та безпеки

Враховуючи те, що ми максимально намагались відповідати правовим нормам в ході нашої роботи ми так чи інакше не є юристами, тому ми б рекомендували пройти консультацію у висококваліфікованого юриста на питання того чи відповідає даний проект правовим нормам тощо і на основі висновку наданому цим спеціалістом внести зміни в проект або ж залишити його таким як він є зараз, якщо заключення експерта було б успішним.

Також ми хотіли б зазначити, що перевірка смарт-контракту хоч і коштує доволі великих грошей, може дозволити нам виявити ті помилки які ми не змогли помітити або ж переконатись у правильності нашого безпекового дослідження, серед найпопулярніших аудиторських фірм виділяють:

- certik. [39] заснована в 2018 році, компанія була створена професорами з йельського та колумбійського університетів і, на момент 2023 року, провела аудити для понад 3 500 проектів, виявила понад 60 000 вразливостей та захистила більше 300 мільярдів активів.certik використовується найбільшими протоколами та обмінами децентралізованих фінансів, такими як binance, okex, aave, polygon та багатьма іншими;

- hashlock. [40] hashlock - це провідна незалежна фірма з кібербезпеки блокчейну та аудиту смарт-контрактів в Австралії. вони є високоспеціалізованою фірмою з кібербезпеки блокчейну, яка виходить із задніх фонів ручного аналізу та аудитів спільнот, і відрізняється кількістю знайдених вразливостей та високим рівнем співпраці з клієнтами як в Австралії, так і по всьому світу;

- consensys diligence [41] ретельно виконує тестування смарт-контрактів, аудити, автоматизований аналіз, моделювання загроз та багато іншого. крім того, consensys пропонує різні інструменти, які використовуються для аудиту смарт-контрактів на ланцюгу ethereum;

- hacken [42] - це кібербезпекова екосистема, заснована експертами у галузі кібербезпеки та етичними хакерами.

Клієнтами є такі компанії, як Solana, Vechain, Gate.io, Kucoin, Huobi, 1inch, Avalanche та інші. Hacken допомагав захищати активи клієнтів та користувачів на суму понад 10 мільярдів доларів. [43]

Проведення аудиту смарт-контракту в однієї з даних компаній буде суттєвою перевагою та допоможе виявити можливі недоліки ще на етапі підготовки проекту до його запуску.

Також ми рекомендували б розглянути можливість розробки криптовалютного представлення гривні у світі блокчейну, це дозволило б уникнути залежності від високої волатильності криптовалют таких як Ефіріум, Біткоїн, тощо.

Оподаткування в Україні проводиться в гривневому еквіваленті і проведення операцій в іншій валюті дещо ускладнює бухгалтерський облік та фінансові розрахунки.

ВИСНОВКИ

У роботі нам вдалось досягнути наступних результатів:

- Розробити смарт контракт який відповідає за реєстрацію користувачів та адміністраторів.
- Розробити смарт-контракт який відповідає за систему оподаткування.
- Розробити смарт-контракт, що створює систему реалізації передачі права власності шляхом купівлі-продажу.

Під час проведеного дослідження було виявлено, що дане рішення може пришвидшити та податкові надходження завдяки автоматизації процесів, терміни передачі права власності, зменшити об'єм бюрократії.

Реалізація права власності через блокчейн вимагає високого рівня автоматизації процесів, відповідного законодавства для цієї процедури і також наявності електронного аналогу гривні задля належної оплати податків, так як курс ефіріум не є стабільним і це не завжди дозволяє чітко встановлювати його, а будь яка автоматизація з підключенням конвертера валют ETH/UAN. Наразі автоматизувати повністю процес неможливо навіть якщо обмежити ролі нотаріусів в системі до тих хто лише проводить перенос даних з минулої бази даних в блокчейн, так як всеодно особи бажаючи продати нерухомість повинні звертатись до них задля встановлення мінімальної ціни продажу, на основі оцінки вартості майна який діє 6 місяців. Ситуацію також ускладнює і те, що не всі власники є одноосібними володарями майна, що вимагає у нас отримання підпису від кожного власника, коли один із володарів майна хоче продати свою частку іншому, а це в залежності від того скільки осіб володіє майном здорожчує комісійні витрати. Також в залежності від того вводимо ми автоматизацію при оплаті податків з оновленням податкового звітного року чи ні, залежить те потрібно нам вести окремий журнал подій чи ні, якщо ми вводимо автоматизацію в питанні оновлення подій, то це підвищує витрати, якщо ж ми не вводимо її, то тут вимагається, щоб кожен нотаріус для кожного користувача оновлював його кількість продажів, так як від кількості продажів залежить те скільки людина заплатить податків. Також мінливість законів не дозволяє нам встановити чіткий

розмір податків назавжди, що додатково вводить ситуацію коли нам потрібно в смарт-контракті прописувати функції які б встановлювали цей самий податок.

Щодо суттєвих переваг цього способу трансферу власності, то це є те, що реєстр власності, тепер не можна буде відредагувати та змінити, якщо туди вже були записані якісь дані, це створює високий поріг прозорості та надійності. Децентралізація теж є перевагою хоч вона тут і часткова при наявності нотаріусів у системі але це так чи інакше дозволяє уникнути цензури в системі та робить її більш стійкішою до корупції та зловживань. Запропонований спосіб також зменшує бюрократію та час очікування при передачі права власності. Блокчейн також забезпечує цілодобовий доступ до інформації реєстру права власності, завдяки чому користувачі завжди матимуть змогу його переглянути і впевнитись, що їх власність досі належить їм. Щодо отримання податків використання блокчейну дозволяє їх автоматизувати, а разом з тим передавати власність в такий спосіб швидше ніж в той що є зараз, це пришвидшує податкові надходження до бюджету. Завдяки чому держава може запускати швидше нові реформи, закуповлювати необхідні ресурси, оплачувати пенсії і так далі. Також це відкриває прозорість з точки зору витрат держави, адже завжди можна відслідкувати на яку адресу контракту або особи, були відправлені гроші того смарт-контракту, який відповідає у нашій системі за оподаткування.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Закон України Про державну реєстрацію речових прав на нерухоме майно та їх обтяжень.URL:<https://zakon.rada.gov.ua/laws/show/1952-15#Text>
2. Онлайн Будинок Юстиції [Електроний ресурс].URL: <https://online.minjust.gov.ua/information/>
3. Про оцінку майна, майнових прав та професійну оціночну діяльність в Україні [Електроний ресурс].URL: <https://zakon.rada.gov.ua/laws/show/2658-14#Text>
4. Цивільний кодекс України [Електроний ресурс].URL: <https://zakon.rada.gov.ua/laws/show/435-15#Text>
5. Закон України про Нотаріат URL: <https://zakon.rada.gov.ua/laws/show/3425-12#Text>
6. Податковий кодекс України URL: <https://zakon.rada.gov.ua/laws/show/2755-17#Text>
7. Business Ombudsman Council. (2019). Quarterly Report (July – September 2019). URL:https://boi.org.ua/wp-content/uploads/2019/11/iii-kv-19_en_for-web.pdf
8. Pandurangiah, S., & Lakshmana Rao, U. (June 2018). What is Blockchain? WILL IT DISRUPT OR DISCIPLINE ACCOUNTANTS? CMA. Sastra Technologies Private Limited. Costechonomist. URL: https://www.researchgate.net/publication/328408283_What_is_Blockchain_WILL_IT_DISRUPT_OR_DISCIPLINE_ACCOUNTANTS_CMA_U_Lakshmana_Roi
9. Hash function URL: <https://www.iso.org/standard/67116.html>
10. Swathi, E., Vivek, G., & Sandhya Rani, G. (2016). Role of Hash Function in Cryptography. URL: https://www.researchgate.net/publication/312242372_Role_of_Hash_Function_in_Cryptography
11. Miller, S. (рік публікації). From Scratch: An Introduction to Hash Functions. URL: <https://math.uchicago.edu/~may/REU2020/REUPapers/Miller.pdf>
12. Zhai, S., Yang, Y., Li, J., Qiu, C., & Zhao, J. (2019). Research on the Application of Cryptography on the Blockchain. IOP Conf. Series: Journal of Physics:

Conf. Series, 1168(3), 032077. doi:10.1088/17426596/1168/3/032077 URL: <https://text2fa.ir/wp-content/uploads/Text2fa.ir-Research-on-the-Application-of-Cryptography.pdf>

13. Ahmed, M. (Year of Publication). BLOCKCHAIN TECHNOLOGY CRYPTOGRAPHIC SECURITY, HASHING AND DIGITAL SIGNATURE. MNIT, JAIPUR: Author's Affiliation. URL: <https://egovstandards.gov.in/sites/default/files/2023-05/Blockchain%20Cryptographic%20Security%2C%20Hashing%20and%20Digital%20Signature.pdf>

14. Що таке гаманець з мультипідписом? [Електронний ресурс] URL: <https://academy.binance.com/uk/articles/what-is-a-multisig-wallet>

15. Inroduction to smart contracts Ethereum URL: <https://ethereum.org/developers/docs/smart-contracts>

16. Vitalik Buterin “Ethereum: a next generationsmart contract and decentralized application platform”, white paper URL: <https://ethereum.org/en/whitepaper> platform”, white paper, <https://ethereum.org/en/whitepaper/>, 2017

17. Hyperledger Fabric. Smart contracts and chain code URL: <https://hyperledgerfabric.readthedocs.io/en/release2.2/smartcontract/smartcontract.html>

18. Жмудь В.В., Горбовий А.Ю., Передача права власності через блокчейн// Цифрова трансформація фінансової системи України та країн V-4 в умовах євроінтеграції, Ч І: III міжн. наук.-практ. інтернет-конф., 26 квіт. 2023 р. : тези доп. – Дубляни, 2023. – С. 26-30. <http://surl.li/iyorn>

19. NEM URL: <https://nemproject.github.io/nem-docs/pages/>

20. NEO Smart Contracts introduction URL: <https://docs.neo.org/v2/docs/en-us/sc/gettingstarted/introduction.html>

21. Solidity Docs URL: <https://docs.soliditylang.org/en/v0.8.24/>

22. Comparing Blockchain Platforms for Smart Contracts: A Preliminary Framework Giuseppe A. Di Lucca, Maria Tortorella URL: https://www.academia.edu/83609279/Comparing_Blockchain_Platforms_for_Smart_Contracts_A_Preliminary_Framework

23. Which Cryptocurrencies Have Fastest Transaction Speeds?[Электроний ресурс] URL: <https://u.today/guides/blockchain/which-cryptocurrencies-have-fastest-transaction-speeds>

24. What is the Ethereum Merge? ETH 2.0 explained [Электроний ресурс] URL: <https://www.moonpay.com/learn/cryptocurrency/ethereum-merge-eth-2>

25. Ethereum average transaction fee URL: https://ycharts.com/indicators/ethereum_average_transaction_fee

26. NEO Explained - What is NEO Blockchain & GAS Cryptocurrency Token [Электроний ресурс] URL: <https://coin.guru/pages/neo-explained-what-is-neo-blockchain-gas-cryptocurrency-token>

27. NEM Price (I:XEMUSD) URL: https://ycharts.com/indicators/nem_price

28. Welcome to Stellar Network [Электроний ресурс] URL: <https://stellar.org/learn/intro-to-stellar>

29. Top 7 smart contract platforms to consider in 2023[Электроний ресурс] URL: <https://www.techtarget.com/searchcio/tip/Top-smart-contract-platforms-to-consider>

30. Santiago Tapia-Fernández, Daniel García-García, Pablo García-Hernandez. "Key Concepts, Weakness and Benchmark on Hash Table Data Structures." 2022. URL: https://www.researchgate.net/publication/359385193_Key_Concepts_Weakness_and_Benchmark_on_Hash_Table_Data_Structures

31. Gas and Fees Ethereum URL: <https://ethereum.org/developers/docs/gas>

32. SafeMath OpenZeppelin [Електронний ресурс] URL: <https://docs.openzeppelin.com/contracts/3.x/api/math>

33. Інструмент аналізу смарт-контрактів Slither [Електронний ресурс]

URL: <https://github.com/crytic/slither>

34. Закон України про захист персональних даних URL: <https://zakon.rada.gov.ua/laws/show/2297-17#Text>

35. Patni, J.C., Sharma, H.K., Tomar, R., & Katal, A. (2021). Database Management System: An Evolutionary Approach. University of Petroleum & Energy Studies. DOI:10.1201/9780429282843 URL: https://www.researchgate.net/publication/356934804_Database_Management_System_An_Evolutionary_Approach

36. The Require Statement in Solidity URL: <https://dev.to/shlok2740/the-require-statement-in-solidity-5d6j#:~:text=In%20Solidity%2C%20the%20require%20statement,the%20condition%20is%20not%20met>

37. What is Gwei? The cryptocurrency explained. URL: [https://www.investopedia.com/terms/g/gwei-ethereum.asp#:~:text=Key%20Takeaways-,Gwei%20is%20a%20denomination%20of%20the%20cryptocurrency%20ether%20\(ETH\)%2C,to%20specify%20Ethereum%20gas%20prices.](https://www.investopedia.com/terms/g/gwei-ethereum.asp#:~:text=Key%20Takeaways-,Gwei%20is%20a%20denomination%20of%20the%20cryptocurrency%20ether%20(ETH)%2C,to%20specify%20Ethereum%20gas%20prices.)

38. Khor, J.H., Aliyu, M.M., Sidorov, M., та ін. (2020). "An Improved Gas Efficient Library for Securing IoT Smart Contracts Against Arithmetic Vulnerabilities." Y: Proceedings of the 2020 9th International Conference on Software and Computer Applications (ICSCA 2020), Langkawi, Malaysia. DOI: 10.1145/3384544.3384577. URL: https://www.researchgate.net/publication/340809304_An_Improved_Gas_Efficient_Library_for_Securing_IoT_Smart_Contracts_Against_Arithmetic_Vulnerabilities

39. Certik Smart Contract Audit Company. URL: <https://www.certik.com/products/smart-contract-audit>

- 40. Hashlock Smart Contract Audit Company. URL: <https://www.hashlock.com.au/>
- 41. Consensus Diligence. URL: <https://consensys.io/diligence/>
- 42. Hacken Smart Contract Company. URL: <https://hacken.io/>
- 43. Best Smart Contract audit companies. URL: <https://cryptojobslist.com/blog/smart-contract-audit-companies>

ДОДАТКИ

Додаток А

Кількість посвідчених договорів з нерухомістю за 2023 рік

2

І. Показники діяльності державних нотаріальних контор, державних нотаріальних архівів та приватних нотаріусів

	Номер рядка	Державні нотаріуси	Приватні нотаріуси
А	Б	1	2
Загальна кількість вчинених нотаріальних дій, (од.) (сума рядків 2–16, 18–33)	1	972 783	9 026 036
Договори відчуження нерухомого майна, крім земельних ділянок (од.) (сума рядків 2.1–2.7), у тому числі:	2	19 788	287 423
купівлі-продажу (од.) (сума рядків 2.1.1–2.1.3), з них:	2.1	9 147	196 896
квартир та житлових будинків (од.)	2.1.1	8 410	164 228
садиб (од.)	2.1.2	14	93
іншого нерухомого майна (од.)	2.1.3	723	32 575
міни (од.) (сума рядків 2.2.1–2.2.3), з них:	2.2	50	1 865
квартир та житлових будинків (од.)	2.2.1	47	1 665
садиб (од.)	2.2.2	0	2
іншого нерухомого майна (од.)	2.2.3	3	198
дарування (од.) (сума рядків 2.3.1–2.3.3)	2.3	10 138	80 400
квартир та житлових будинків (од.)	2.3.1	9 644	70 703
садиб (од.)	2.3.2	6	75
іншого нерухомого майна (од.)	2.3.3	488	9 622
довічного утримання (догляду) (од.)	2.4	258	1 996
договори пожертви (од.)	2.5	4	314
договори ренти (од.)	2.6	0	17
інші договори відчуження нерухомого майна (од.)	2.7	191	5 935
Договори відчуження земельних ділянок (у тому числі сільськогосподарського призначення), а також земельних часток (паїв) (од.) (сума рядків 3.1–3.3), з них:	3	21 681	285 941
купівлі-продажу (од.) (сума рядків 3.1.1–3.1.3), з них:	3.1	14 086	217 795
земельних ділянок сільськогосподарського призначення (од.) (сума рядків 3.1.1.1, 3.1.1.2), з них:	3.1.1	6 558	97 817
для ведення особистого селянського господарства, виділених в натурі (на місцевості) власникам земельних часток (паїв) (од.)	3.1.1.1	1 923	29 828

Джерело: <https://minjust.gov.ua/m/zvit-pro-robotu-derjavnih-privatnih-notariusiv-ministerstva-yustitsii-ukraini-ta-yogo-teritorialnih-organiv-u-sferi-notariatu>

Смарт-контракт для реєстрації користувачів

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

contract PropertyRightsRegistration {
    address public admin;

    mapping(address => bool) public admins;
    mapping(address => adminsUsers) public adminsUsersMapping;

    struct adminsUsers {
        address adminAddress;
        address[] adminsSigners;
        uint confirmationsCount;
    }

    constructor() {
        // Ініціалізація адмінів
        admins[msg.sender] = true;
        adminsUsersMapping[msg.sender] = adminsUsers({
            adminAddress: msg.sender,
            adminsSigners: new address[](0),
            confirmationsCount: 3
        });

        // Інші адміни (приклад, можна ініціалізувати їх адреси)
        address admin2 = 0x617F2E2fD72FD9D5503197092aC168c91465E7f2;
        address admin3 = 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4;
        address admin4 = 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2;
```

```

.....admins[admin2] = true;
.....adminsUsersMapping[admin2] = adminsUsers({
.....    adminAddress: admin2,
.....    adminsSigners: new address[] (0),
.....    confirmationsCount: 3
.....});

.....admins[admin3] = true;
.....adminsUsersMapping[admin3] = adminsUsers({
.....    adminAddress: admin3,
.....    adminsSigners: new address[] (0),
.....    confirmationsCount: 3
.....});

.....admins[admin4] = true;
.....adminsUsersMapping[admin4] = adminsUsers({
.....    adminAddress: admin4,
.....    adminsSigners: new address[] (0),
.....    confirmationsCount: 3
.....});
}

modifier onlyAdmins() {
.....require(admins[msg.sender], "Only admins can call this function");
....._;
}

```

```

function registerPotentialAdmin(
    address _potentialAdmin
) external onlyAdmins {
    require(!admins[_potentialAdmin], "Already an admin");
    adminsUsersMapping[_potentialAdmin] = adminsUsers({
        adminAddress: _potentialAdmin,
        adminsSigners: new address[](0),
        confirmationsCount: 0
    });
}

function generateUserId(
    string memory _name,
    string memory _surname,
    address _userAddress
) internal pure returns (uint256) {
    bytes32 hash = keccak256(abi.encode(_name, _surname, _userAddress));
    return uint256(hash);
}

function isAddressInList(
    address _target,
    address[] memory _list
) public pure returns (bool) {
    for (uint256 i = 0; i < _list.length; ++i) {
        if (_list[i] == _target) {
            return true;
        }
    }
    return false;
}

```

```

function isNumberInList(
    uint256 targetNumber,
    uint256[] memory numberList
) public pure returns (bool) {
    for (uint256 i = 0; i < numberList.length; ++i) {
        if (numberList[i] == targetNumber) {
            return true;
        }
    }
    return false;
}

```

```

function toggleAdminStatus(address _adminToToggle) external onlyAdmins {
    require(
        adminsUsersMapping[_adminToToggle].adminAddress != address(0),
        "Admin user not registered"
    );
    require(
        !isAddressInList(
            msg.sender,
            adminsUsersMapping[_adminToToggle].adminsSigners
        ),
        "That admin has already signed"
    );

    if (!admins[_adminToToggle]) {
        adminsUsersMapping[_adminToToggle].confirmationsCount++;
        adminsUsersMapping[_adminToToggle].adminsSigners.push(msg.sender);

        if (adminsUsersMapping[_adminToToggle].confirmationsCount >= 3) {
            admins[_adminToToggle] = true;
            delete adminsUsersMapping[_adminToToggle].adminsSigners;
        }
    } else {
        adminsUsersMapping[_adminToToggle].confirmationsCount--;
        adminsUsersMapping[_adminToToggle].adminsSigners.push(msg.sender);

        if (adminsUsersMapping[_adminToToggle].confirmationsCount == 0) {
            admins[_adminToToggle] = false;
            delete adminsUsersMapping[_adminToToggle].adminsSigners;
        }
    }
}

```

```

function revokeAdminStatusByOwn() external onlyAdmins {
    admins[msg.sender] = false;
}

modifier onlyValidator() {
    require(
        validators[msg.sender],
        "Only validator can call this function"
    );
    _;
}

```

```

mapping(address => User) public users;
mapping(address => UserRelatedAddresses) public relatedAddresses;
mapping(uint256 => bool) public usedLicenseNumbers;
mapping(address => uint256) validatorLicenseNumber;
mapping(address => bool) public validators;
mapping(address => bool) public verifiedUsers;

struct User {
    uint256 id;
    string name;
    string surname;
    uint validatorConfirmationCount;
    uint[] validatorsSigners;
    bool isVerified;
    uint adminConfirmationCount;
    address[] adminsSigners;
    bool isValidator;
    uint256 licenseNumber;
}

struct UserRelatedAddresses {
    address relatedAddress1;
    address relatedAddress2;
    address relatedAddress3;
}

```

```

function registerUser(
    string memory _name,
    string memory _surname
) external {
    require(users[msg.sender].id == 0, "User already registered");
    uint256 userId = generateUserId(_name, _surname, msg.sender);

    users[msg.sender] = User({
        id: userId,
        name: _name,
        surname: _surname,
        validatorConfirmationCount: 0,
        validatorsSigners: new uint[](0),
        isVerified: false,
        adminConfirmationCount: 0,
        adminsSigners: new address[](0),
        isValidator: false,
        licenseNumber: 0
    });
}

```

```

function addLicenseNumber(uint256 _licenseNumber) external {
    require(
        !usedLicenseNumbers[_licenseNumber],
        "License number already used"
    );
    require(
        users[msg.sender].licenseNumber == 0,
        "User already has a license number"
    );

    users[msg.sender].licenseNumber = _licenseNumber;
    usedLicenseNumbers[_licenseNumber] = true;
}

```

```

function approveValidator(address _userAddress) external onlyAdmins {
    require(users[_userAddress].id != 0, "User not found");
    require(
        users[_userAddress].licenseNumber != 0,
        "License number not set"
    );
    require(msg.sender != _userAddress, "Admin cannot approve themselves");
    require(
        !isAddressInList(msg.sender, users[_userAddress].adminsSigners),
        "Admin already signed for this Validator"
    );

    users[_userAddress].adminsSigners.push(msg.sender);
    users[_userAddress].adminConfirmationCount++;

    if (users[_userAddress].adminConfirmationCount >= 3) {
        users[_userAddress].isValidator = true;
        users[_userAddress].isVerified = true;

        validators[_userAddress] = true;
        validatorLicenseNumber[_userAddress] = users[_userAddress].licenseNumber;

        verifiedUsers[_userAddress] = true;

        delete users[_userAddress].adminsSigners;
    }
}

```

```

function revokeValidatorByOwn() external onlyValidator {
    validators[msg.sender] = false;
}

function verifyUser(address _userAddress) external onlyValidator {
    require(users[_userAddress].id != 0, "User not found");
    require(
        msg.sender != _userAddress,
        "Validator cannot modify own status"
    );
    require(
        !isNumberInList(
            users[msg.sender].licenseNumber,
            users[_userAddress].validatorsSigners
        ),
        "Validator already signed for this user"
    );

    users[_userAddress].validatorConfirmationCount++;
    users[_userAddress].validatorsSigners.push(
        users[msg.sender].licenseNumber
    );

    if (users[_userAddress].validatorConfirmationCount >= 3) {
        users[_userAddress].isVerified = true;

        verifiedUsers[_userAddress] = true;
    }
}
}

```

```

function checkValidator(address _validator) external view returns (bool) {
    return validators[_validator];
}

function checkVerifiedUsers(address _user) external view returns (bool) {
    return verifiedUsers[_user];
}

function getLicenseNumber(address _validator) external view returns (uint256) {
    return validatorLicenseNumber[_validator];
}
}

```

Смарт-контракт для взаємодії з нерухомістю

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

import "../PropertyRightsRegistration.sol";

contract PropertyRightsExchange {

    PropertyRightsRegistration public userRegistration;

    // Конструктор, де ви передаєте адресу смарт-контракту
    constructor(address _propertyRightsRegistration) {
        userRegistration = PropertyRightsRegistration(_propertyRightsRegistration);
    }

    //Helping internal functions
    function cleanAndToLower(
        string memory str
    ) internal pure returns (string memory) {
        bytes memory bStr = bytes(str);
        bytes memory bCleanLower = new bytes(bStr.length);
        uint j = 0;

        for (uint i = 0; i < bStr.length; ++i) {
            // Пропускаємо пробіли
            if (uint8(bStr[i]) != 32) {
                // Переводимо в нижній регістр
                bCleanLower[j++] = (uint8(bStr[i]) >= 65 &&
                    uint8(bStr[i]) <= 90)
                    ? bytes1(uint8(bStr[i]) + 32)
                    : bStr[i];
            }
        }
    }
}
```

```

struct RequestProperty {
    string location;
    string propertyType;
    string description;
    uint256 validatorsConfirmationCount;
    uint[] validatorsSigners;
}

struct AddProperty {
    string location;
    string propertyType;
    string description;
    address[] owner;
    uint256 estCost;
}

struct AddOwner {
    uint registrationNumber;
    uint partInOwnership;
    uint validatorsConfirmationCount;
    uint[] validatorsSigners;
}

```

```

// Mapping for real estates
mapping(uint256 => RequestProperty) public propertyRequests;
mapping(uint256 => AddProperty) public realEstates;

// Mappings for Property allowance
mapping(address => AddOwner) public additionalOwnerRequests;
mapping(uint256 => mapping(address => uint)) public OwnersParts;
mapping(uint256 => bool) public existedProperty;
mapping(uint256 => string) public registrationNumberToPropertyAddress;
mapping(string => uint256) public addressRegistrationNumberpair;
mapping(uint256 => string) public registrationNumberTypeProperty;

// Function for requesting before adding the Property
// It needs to successful Property registration
// mapping(uint256 => RequestProperty) propertyRequests;

```

```

function requestedProperty(
  uint256 _registrationNumber,
  string memory _location,
  string memory _propertyType,
  string memory _description
) external {
  require(userRegistration.checkValidator(msg.sender), "You are not a validator");
  require(!existedProperty[_registrationNumber], "Property was added");

  propertyRequests[_registrationNumber] = RequestProperty({
    location: _location,
    propertyType: _propertyType,
    description: _description,
    validatorsConfirmationCount: 0,
    validatorsSigners: new uint[](0)
  });
}

```

```

function addProperty(uint256 _registrationNumber) external {
  require(userRegistration.checkValidator(msg.sender), "You are not a validator");
  require(
    !existedProperty[_registrationNumber],
    "Property already exists"
  );
  require(
    !userRegistration.isNumberInList(
      userRegistration.getLicenseNumber(msg.sender),
      propertyRequests[_registrationNumber].validatorsSigners
    ),
    "Repeated Sign!"
  );

  propertyRequests[_registrationNumber].validatorsConfirmationCount++;
  propertyRequests[_registrationNumber].validatorsSigners.push(
    userRegistration.getLicenseNumber(msg.sender)
  );

  if (
    propertyRequests[_registrationNumber].validatorsConfirmationCount >=
    3
  ) {
    existedProperty[_registrationNumber] = true;

    realEstates[_registrationNumber] = AddProperty({
      location: propertyRequests[_registrationNumber].location,
      propertyType: propertyRequests[_registrationNumber].propertyType,
      description: propertyRequests[_registrationNumber].description,
      owner: new address[](0),
      estCost: 0
    });

    delete propertyRequests[_registrationNumber];
  }
}

```

```

// You need to set up owner part!
// If an owner has 25% of ownership, you should put for the user number 4
// If it's a single owner, you should put 1, for example
// If part isn't declared, install the number which represents the total quantity of owners
function addingOwner(
    address _userAddress,
    uint256 _registrationNumber,
    uint _partInOwnership
) external {
    require(userRegistration.checkValidator(msg.sender), "You are not a validator");
    require(
        !userRegistration.isAddressInList(
            _userAddress,
            realEstates[_registrationNumber].owner
        ),
        "Already added"
    );
    require(
        msg.sender != _userAddress,
        "Validator can't modify own parameters"
    );
    require(userRegistration.checkVerifiedUsers(_userAddress), "Not a verified user!");
    require(existedProperty[_registrationNumber], "Property does not exist");

    additionalOwnerRequests[_userAddress] = AddOwner({
        registrationNumber: _registrationNumber,
        partInOwnership: _partInOwnership,
        validatorsConfirmationCount: 0,
        validatorsSigners: new uint[](0)
    });
}

```

```

function confirmOwner(address _userAddress) external {
    require(userRegistration.checkValidator(msg.sender), "You are not a validator");
    require(msg.sender != _userAddress, "Validator can't sign for itself");
    require(
        !userRegistration.isNumberInList(
            userRegistration.getLicenseNumber(msg.sender),
            additionalOwnerRequests[_userAddress].validatorsSigners
        ),
        "Repeated Sign!"
    );

    additionalOwnerRequests[_userAddress].validatorsConfirmationCount++;
    additionalOwnerRequests[_userAddress].validatorsSigners.push(
        userRegistration.getLicenseNumber(msg.sender)
    );

    if (
        additionalOwnerRequests[_userAddress].validatorsConfirmationCount >=
        3
    ) {
        realEstates[
            additionalOwnerRequests[_userAddress].registrationNumber
        ].owner.push(_userAddress);

        OwnersParts[
            additionalOwnerRequests[_userAddress].registrationNumber
        ][_userAddress] = additionalOwnerRequests[_userAddress]
            .partInOwnership;

        delete additionalOwnerRequests[_userAddress];
    }
}

```

```
// Mapping for ownership history
mapping(uint256 => address[]) public ownershipHistoryMapping;

// Sell Property
mapping(string => uint) public PropertyTaxes;

// Before selling, you need to send a request to any of the validators to set up the minPrice for your property
// Don't update
function setupMinPrice(
    uint _registrationNumber,
    uint _minPrice
) external {
    require(userRegistration.checkValidator(msg.sender), "You are not a validator");
    require(existedProperty[_registrationNumber], "Property does not exist");

    realEstates[_registrationNumber].estCost = _minPrice * 1 ether;
}

// Use this structure for Property sell if you are a single owner

struct PutForSellSingle {
    uint registrationNumber;
    uint countSigns;
    address[] sellerSigns;
    address buyer;
    uint price;
}
```

```
mapping(address => PutForSellSingle) public singleOwnerSell;

function setupSellSingle(
    uint _registrationNumber,
    address _buyer,
    uint _sellType,
    uint _price
) external {
    require(
        _sellType == 1 || _sellType == 2,
        "1 for full sale, 2 for part sale"
    );
    require(
        realEstates[_registrationNumber].owner.length == 1,
        "Not a single owner"
    );
    require(
        realEstates[_registrationNumber].estCost > 0,
        "Minimum cost isn't installed"
    );
    require(userRegistration.checkVerifiedUsers(msg.sender), "Not a verified user!");
    require(userRegistration.checkVerifiedUsers(_buyer), "Not a verified buyer!");
    require(
        userRegistration.isAddressInList(msg.sender, realEstates[_registrationNumber].owner),
        "You are not the owner"
    );

    singleOwnerSell[msg.sender] = PutForSellSingle({
        registrationNumber: _registrationNumber,
        countSigns: 0,
        sellerSigns: new address[](0),
        buyer: _buyer,
        price: _price * 1 ether
    });
}
```

```
function setForSaleSingle() external {
    require(userRegistration.checkVerifiedUsers(msg.sender), "User not verified");

    singleOwnerSell[msg.sender].countSigns++;
    singleOwnerSell[msg.sender].sellerSigns.push(msg.sender);
}
```

```

function purchaseSingleProperty(
  address _userAddress,
  uint _registrationNumber
) external payable {
  PutForSellSingle storage sellInfo = singleOwnerSell[_userAddress];

  require(msg.sender == sellInfo.buyer, "Not the specified buyer");
  require(sellInfo.registrationNumber == _registrationNumber);
  require(sellInfo.countSigns >= 1);
  require(msg.value == sellInfo.price, "Incorrect payment amount");

  uint estCost = realEstates[_registrationNumber].estCost;
  uint taxRate = 5; // Assume a tax rate of 5%, you can adjust this accordingly

  // Full sale
  address previousOwner = realEstates[_registrationNumber].owner.length > 0
    ? realEstates[_registrationNumber].owner[0]
    : address(0);

  // Verify that the previous owner is a valid member of the array
  require(previousOwner != address(0), "Invalid previous owner");

  // Calculate tax amount
  uint taxAmount = (msg.value < estCost)
    ? ((estCost - msg.value) * taxRate) / 100
    : 0;

  // Update ownership information
  OwnersParts[_registrationNumber][previousOwner] = 0;
  OwnersParts[_registrationNumber][msg.sender] = 1;

```

```

    // Update ownership information
    OwnersParts[_registrationNumber][previousOwner] = 0;
    OwnersParts[_registrationNumber][msg.sender] = 1;

    // Transfer funds to the previous owner
    payable(previousOwner).transfer(msg.value - taxAmount);

    // Pay the tax to the contract address
    payable(address(this)).transfer(taxAmount);

    // Update the owner in the realEstates mapping
    realEstates[_registrationNumber].owner[0] = msg.sender;

    // Clean up the singleOwnerSell mapping for the user
    delete singleOwnerSell[_userAddress];
  }

  // Function to get ownership history for a specific property
  function getOwnershipHistory(
    uint256 registrationNumber
  ) public view returns (address[] memory) {
    return ownershipHistoryMapping[registrationNumber];
  }
}

```

Міністерство освіти і науки України
Львівський національний університет природокористування

СЕРТИФІКАТ
УЧАСНИКА

Жмудь В.В.
(прізвище, ім'я)

взяв(ла) участь у конференції

«ЦИФРОВА ТРАНСФОРМАЦІЯ ФІНАНСОВОЇ СИСТЕМИ
УКРАЇНИ ТА КРАЇН V-4 В УМОВАХ ЄВРОІНТЕГРАЦІЇ»

«DIGITAL TRANSFORMATION OF THE FINANCIAL SYSTEM OF UKRAINE
AND V-4 COUNTRIES IN THE CONTEXT OF EUROPEAN INTEGRATION»

Кафедра фінансів, банківської справи та страхування

Ректор ЛНУП,
Академік НААНУ

 Снітинський В.В.



26 КВІТНЯ 2023 р. м.Дубляни