Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАСТОСУНКУ ДЛЯ СТВОРЕННЯ НЕВЗАЄМОЗАМІННОГО ТОКЕНУ (NFT) В БЛОКЧЕЙН З ВИКОРИСТАННЯМ КРИПТОГАМАНЦЯ METAMASK
(тема)

Виконав:
студент 4 курсу, групи ІТІНФ-18-1

Дігтяренко В.Е.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник ст.викл.Кіношенко Д.К.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2022 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри
_____
(підпис)
«_____»_____ 2022 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Дігтяренко Владіславу Едуардовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розроблення програмного застосунку для створення невзаємозамінного токену (NFT) в блокчейн з використанням криптогаманця Metamask

затверджена наказом університету від 16 травня 2022 року № 541Ст

2. Термін подання студентом роботи до екзаменаційної комісії 28 травня 2022 р.

3. Вихідні дані до роботи застосунок який виконує функцію створення NFT в блокчейні, інструкція для користувача, теоретичні відомості про особливості розробки, середовище розробки VSCode та Remix IDE.

4. Перелік питань, що потрібно опрацювати в роботі
1. Аналіз існуючі технології для реалізації веб застосунку.
2. Аналіз технічної інформація роботи з блокчейно.
3. Розробка функціоналу застосунку.
4. Розробити інструкцію про використання застосування.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) ключові частини коду, таблиці бази даних, тестова база даних, скріншоти готового застосунку, опис можливостей користувача, дослідження результатів, висновки.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Консультант з дотримання діючих стандартів та норм	Доцент Бєлова Н.В.		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	18.04.2022	
2	Аналіз завдання, підбір літератури	18.04.22-21.04.22	
3	Аналіз літератури з досліджуваної проблеми	22.04.22-25.04.22	
4	Аналіз технічних і програмних засобів	26.04.22-30.04.22	
5	Розробка методу	01.05.22-14.05.22	
6	Програмна реалізація	15.05.22-23.05.22	
7	Оформлення пояснювальної записки	24.05.22-26.05.22	
8	Перевірка на плагіат	28.05.22	
9	Рецензування	29.05.22	
10	Підготовка презентації та доповіді	29.05.22-30.05.22	
11	Занесення роботи в електронний архів	31.05.22	
12	Попередній захист кваліфікаційної роботи	31.05.22	

Дата видачі завдання 18 квітня 2022 р.

Студент _____

(підпис)

Керівник роботи _____ ____ст.викл. Кіношенко Д.К. _

(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 65 с., 47 рис., 32 джерела.

СТВОРЕННЯ НЕ ВЗАЄМОЗАМІННОГО ТОКЕНУ, КРИПТОГАМАНЕЦЬ, БЛОКЧЕЙН, МЕРЕЖА ETHEREUM, WEB3.

Об’єктом роботи є розробка веб застосунку створення NFT в блокчейн мережі Ethereum Ropsten за допомогою використання криптогаманця Metamask.

Метою роботи є розробка веб застосунку, який дасть можливість створювати NFT із зображення та його метаданих, використовуючи програмний гаманець.

Використано набори інструментів, які представленні відкрітім кодом, для взаємодиї з Web3. Проведено дослідження методів створення NFT у блокчейні за допомогою бібліотек з відкритим кодом та контрактів у блокчейні Ethereum.

У результаті роботи здійснена програмна реалізація веб застосунку, його серверна частина та контракт в мережі Ethereum Ropsten.

CREATION OF NON-INTERCHANGEABLE TOKEN, CRYPT WALLET, BLOCKCHAIN, ETHEREUM NETWORK, WEB3.

The object of the work is to develop a web application for creating NFT in the blockchain network Ethereum Ropsten using the cryptocurrency Metamask.

The aim of the work is to develop a web application that will allow you to create NFT from the image and its metadata, using a wallet.

Open source toolkits are used to interact with Web3. Research has been conducted on how to create NFT in a blockchain using open source libraries and contracts in the Ethereum blockchain.

As a result, the software application of the web application, its server part and the contract in the Ethereum Ropsten network were implemented.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

NFT – це одиниця даних у цифровій книзі, що називається блокчейном, де кожен NFT може представляти унікальний цифровий елемент, і тому кожен з них не взаємозамінний

Web3 – концепція нової ітерації розвитку Вебу, який би був децентралізованим та базувався на блокчейнах

JSON – JavaScript Object Notation

HTTP – HyperText Transfer Protocol

HTTPS – HyperText Transfer Protocol Secure

HTML – HyperText Markup Language

API – Application Programming Interface

ВСТУП

Блокчейн (Blockchain або Block Chain) – вибудуваний за певними правилами безперервний послідовний ланцюжок блоків, що містять інформацію. Найчастіше копії ланцюжків блоків зберігаються на безлічі різних комп'ютерів незалежно один від одного. Вперше термін з'явився як назва повністю реплікованої розподіленої бази даних, реалізованої в системі «біткоїнів», через що блокчейн часто відносять до транзакцій в різних крипто валютах, проте технологія ланцюжків блоків може бути поширена на будь-які взаємопов'язані інформаційні блоки.

На цій технології базуються криптовалюти. Взагалі криптовалюти можна поділити на два типи: взаємозамінні токени та невзаємозамінні. Пояснивши те, чим є взаємозамінні токени, поняття NFT стане зрозумілішим.

Взаємозамінні токени – це те, що всі люди називають звичайною криптовалютою. Біткоїн та етеріум вже є класичними прикладами взаємозамінних токенів. Як і звичайна валюта – гривні, долари чи євро, вони мають визначену вартість та можуть обмінюватися. Одна монета по суті не відрізняється від іншої та еквівалентна за вартістю.

Невзаємозамінні токени – унікальні криптовалюти, які так само як і класичні криптовалюти зберігаються в блокчейні, але не можуть бути замінені чимось іншим. Вони можуть представляти унікальні речі, приклади яких були описані раніше.

Взаємозамінні активи можуть обмінюватися без ніяких сумнівів. На противагу їм є невзаємозамінні активи – наприклад, антикваріат, мистецтво, дизайнерські речі тощо. Їх неможливо так просто обміняти один на одного.

Завдяки своїй унікальності, вони в основному використовуються як криптовалютні предмети для мистецтва та ігор.

Актуальність роботи полягає у тому що цифрове мистецтво до недавнього часу не мало можливості відрізнити оригінал від копії. Якесь зображення в інтернеті: наприклад, мем з Жабеням Пепе, розліталося

інтернетом та не мало ніякої унікальності відносно найпершого файлу, який був створений автором. NF-токени дають однозначну можливість прив’язатися саме до того зображення. Це відбувається в мережі блокчену та дозволяє визнати зображення оригіналом, а не копією, з можливістю верифікації цього. А коли законодавство може змінитись під нові технології, це дасть величезний скачок монетизації та відкриє нові можливості творцям контенту.

1 ОГЛЯД ОСНОВНИХ МЕТОДІВ

1.1 Огляд технології Web3

По-перше, важливо зазначити, що так звані версії Інтернету формувалися з часом поступово. Немає чіткого історичного переходу від Web 1.0 до Web 2.0. Ймовірно, людство також не зможе моментально перейти на Web 3.0. Мережа розвивалася, на веб-сайтах починали з’являтися нові функції, Інтернет прийшов до нових концепцій і користувачі помітили, що вони вже користуються зовсім іншою інфраструктурою. На цьому етапі описали ідеї нової епохи, а також підбили підсумки минулої.

Також самі концепції – умовності, зібрані впливовою особистістю.
Як показує історія, найчастіше вони виникали після порівняння сервісів та сайтів щодо використання технологій.

1.1.1 Web 1.0

Саме визначення Web 1.0 узвичаїлося, як не дивно, після появи Web 2.0. Порівняння показало, що Інтернет став зовсім іншим та вдалося виділити особливості та відмінності минулої «версії». Весь етап першої ітерації Мережі продовжився з 1991 до 2004 року. Web 1.0 можна описати однією фразою – Read-Only (тільки читання). Це і пояснює основні концепції. Користувачі мали можливість лише переглядати сторінки та взаємодіяти з контентом. В Інтернеті ще не були розвинені можливості участі користувачів у створенні контенту, вони лише споживали те, що на веб-ресурсах. Жодних авторизацій, трекерів та реєстрацій.

Дані сайтів зберігалися на серверах у файлових системах і часто видавалися у тому вигляді, в якому вони були. Ця особливість змушувала веб-містерів при додаванні нових сторінок заново верстати ті, що вже були, задля

додавання посилань. Для вирівнювання контенту застосовувалися таблиці, на сайтах був адаптивності і часто вказувалося рекомендований дозвіл, у якому вся інформація відкривалася б коректно. Також не кожен сайт підтримувався всіма браузерами та веб-майстри розміщували бейджі з логотипами тих веб-браузеров, які правильно працювали з ресурсом.

На заході епохи Web 1.0 стали з'являтися форуми та чати, які дозволяли користувачам самим брати участь у формуванні контенту. Але при цьому компанія Amazon із самого відкриття свого сайту дала можливість клієнтам залишати відгуки на товари. Певною мірою корпорація випередила час.

Підсумовуючи, виділимо основні риси Інтернету епохи Web 1.0:

- етап тривав із 1991 по 2004 рік;
- сайти лише для читання;
- відсутність інтерактивності та автоматизації;
- мінімальна участь користувачів у формуванні контенту;
- примітивний дизайн;
- веб-майстри публікують матеріали, а користувачі читають.

1.1.2 Web 2.0 та Web 3.0

У 2005 році американський видавець та активіст руху за вільне програмне забезпечення Тім О'Райлі (Tim O'Reilly) опублікував статтю What Is Web 2.0. У матеріалі О'Райлі зазначив, що у Мережі починає з'являтися дедалі більше сайтів, об'єднаних ідеями та єдиним принципом. У статті видавець чітко розділив Web 1.0 та Web 2.0 та намітив вектор розвитку. Саме за допомогою цієї статті досі визначають ключові принципи «нового покоління».

Вважається, що епоха Web 2.0 почалася в 2004 році і продовжується до сьогодні. Тепер у справу включилися великі корпорації та користувачі. Перші взяли під контроль Мережа та почали будувати онлайн-імперії, а другим

дозволили брати участь у створенні контенту. Сам Web 2.0 працює за принципом Read/Write Web (читання/запис у Мережі).

В Інтернет прийшла повсюдна авторизація та можливість створити обліковий запис практично на кожному сайті. Користувачі почали добровільно залишати свої дані та давати згоду на збір даних замість зручності та можливості користування ресурсами. Компанії отримали можливість заробляти на даних, продаючи їх рекламним агентствам, а деякі відкрили власні, що допомогло повністю зосередити прибуток у своїх руках.

У Web 2.0 з'явилися соціальні функції: дедалі більше ресурсів дозволяє користувачам спілкуватися між собою, обмінюватися повідомленнями та здійснювати дзвінки. До соціалізації можна віднести і персоналізацію: користувачі можуть оформляти власні профілі, додавати на сторінки фотографії та записи, розміщувати відеоролики та статті. Користувачі публікують матеріали та отримують реакції та оцінки від інших користувачів у вигляді лайків та коментарів. Також варто зазначити, що сайти почали вводити системи рейтингу – Карму чи Репутацію.

Зміни вплинули і на дизайн. Зовнішній вигляд сайтів став приємнішим, стали переважати округлі форми, прості точно підібрані кольори, дизайнери почали звертати увагу не лише на зовнішній вигляд, але й на зручність. Сайти стали простішими, але не менш інформативними. З'явилися складні анімації і пішла епоха GIF. Незважаючи на це, фахівці зазначають, що коли у кожного з'явилася можливість створюватися сайти, то світ прийшли шаблони і це призвело до одноманітності. Загалом зазначається, що в Web 2.0 переважають патерни, які вбивають індивідуальність та оригінальність сайтів. У побут прийшла друкарня: текст на сайтах почав виділятися в міру значущості. З'явилися заголовки, підзаголовки, різні шрифти, підкреслення та виділення. А сайти стали адаптивними: один і той самий веб-портал можна відкрити як на десктопі, так і на смартфоні.

Розробка сайтів стала складнішою порівняно з Web 1.0. Для зручності користувачів з'явилися веб-служби, дані почали передавати у форматах JSON

або XML. Більшість операцій перейшла на сервери підприємств, і в користувачів відпала необхідність дбати про оновлення даних і обчислювальних потужностях. AJAX (Asynchronous JavaScript and XML) дозволив сторінкам не перезавантажуватися щоразу, а асинхронно завантажувати ті дані, які необхідні користувачеві.

Інтернетом епохи Web 2.0 «правлять» рекомендаційні алгоритми. На користь корпорацій – утримувати увагу користувача на платформі, тому фірми прагнуть рекомендувати користувачеві той контент, який буде цікавий людині. Алгоритми враховують безліч факторів і працюють практично скрізь, починаючи із соціальних мереж та закінчуючи маркетплейсами. Саме тому, якщо два користувача відкриють, наприклад, головну сторінку YouTube, то вміст на ній кардинально відрізнятиметься. У цьому принципі видно ще одну відмінність Web 2.0 від першої «версії» Мережі, коли ту саму сторінку переглядали тисячі користувачів.

Підбиваючи підсумки, Web 2.0 відрізняється від Web 1.0 наступним:

– користувачі можуть самі брати участь у житті Інтернету та наповнювати його контентом;

– великі корпорації стали законодавцями трендів та ініціаторами змін;

– дані користувача стали «товаром» для рекламодавців, Web 2.0 можна описати фразою «ера таргетованої реклами і нестачі приватності»;

– розвинулися та стали частиною життя користувачів соціальні функції Мережі;

– користувачі не контролюють свої дані, і корпорації можуть видаляти невідповідний контент;

– інформація, як і раніше, зберігається на єдиних серверах і видається на вимогу.

Основні концепції Web 3.0 окреслив керівник компанії Netscape Джейсон Калаканіс (Jason Calacanis). По-перше, важливо зазначити, що, як і раніше, ніхто не має чіткого уявлення про те, яким буде новий виток розвитку Інтернету. По-друге, Web 3.0 знаходиться на ранній стадії, тому поки що

доступні тільки первинні уявлення про технологію. Проте Калаканіс опублікував своє бачення майбутнього ще у 2007 році і вважає, що на основі Web 2.0 має з'явитися новий простір, що вирішує основні проблеми.

Основною проблемою керівник вважає знецінення ресурсів та сервісів: відносна простота створення сайтів вплинула на виникнення одноманітності. Тім О'Райлі підтримав свого часу ідеї Калаканіса, а також зазначив, що Web 3.0 має вийти за межі звичного розуміння Мережі та почати «взаємодіяти з фізичним світом».

Визначальні характеристики Web 3.0:

– децентралізація тобто дані більше не зберігатимуться на єдиних серверах, а розподіляться між користувачами. Необхідні обчислення переїдуть із датацентрів на ноутбуки, смартфони та «розумні» гаджети користувачів. В даний час є технології, що дозволяють досягти цього, але немає єдиного рішення про те, яка з них лежатиме в основі «нового Інтернету»;

– ІІ та машинне навчання: інтелектуальні алгоритми не зникнуть із Мережі і все також продовжуватимуть допомагати користувачам шукати необхідний контент. Деякі дослідники відзначають, що в майбутньому ІІ можна буде використовувати для виявлення рекомендованих коментарів на маркетплейсах, що допоможе створити прозоріші сервіси;

– відкритість – ПЗ буде переважно з відкритим вихідним кодом, що дозволить досконально розуміти, як влаштовані інструменти і яким чином вони взаємодіють із користувачем;

– свобода – очікується, що цензура в Мережі буде скасована, і кожен матиме можливість публікувати будь-який контент, роль модерації він візьме співтовариство, а чи не корпорації;

– всюдисутність – фахівці припускають, що в епоху Web 3.0 Інтернет буде практично в будь-якому місці, а розповсюджувачами стануть IoT-пристрої та розумні гаджети;

– семантична павутина – машина погано розуміє запити природною мовою і все ще часто помиляється. Для покращення цього процесу планують

використовувати технологію семантичної павутини, коли з Мережі можна отримувати інформацію виду «предмет – вид взаємозв'язку – інший предмет» і вже за цими даними будувати логічні зв'язки.

Крім ключових відмінностей ентузіасти та фахівці пророкують та інші зміни. По-перше, може змінитися метод авторизації в сервісах на єдиний прошарок, який буде ключем до всіх ресурсів в Мережі. Щось схоже реалізовано зараз: за допомогою облікового запису Google або Facebook можна увійти на практично будь-який сайт, але у випадку Web 3.0 єдиний обліковий запис може стати і гаманцем, і банківським додатком.

По-друге, Інтернет стане більш демократизованим і корисність контенту визначатимуть Децентралізовані автономні організації (ДАО), якими стануть великі компанії та послуги, а учасники ДАТ матимуть право голосу та прийматимуть важливі для контенту та самих організацій рішення. Іншими словами, це щось на зразок глобальної системи карми.

Зараз ідеї Web 3.0 все ще залишаються ідеями та викликають більше питань, ніж розуміння. До того ж, концепцію часто просувають криптоінвестори та NFT-ентузіасти, які зводять все до нового виду заробітку.

Як скоро очікувати перехід до Web 3.0, і чи готові корпорації відмовитися від даних і піти до децентралізації?

Децентралізація не означає відмову від даних. І немає серйозних передумов для того, щоб даних про користувачів ставало менше.

Користувачі із задоволенням завантажують усю свою інформацію у соцмережі. А корпорації успішно її аналізують та продають рекламодавцям.

Перехід відбудеться в той момент, коли децентралізовані мережі стануть зручнішими і привабливішими за класичні.

Біткоїн на кілька порядків простіше, швидше та дешевше, ніж SWIFT. Ютуб цікавіший, ніж телевізор. Новини у твіттері з'являються раніше, ніж у ЗМІ. Зручність і довіра – основні стимули, а чи не технології, що у їх основі.

Децентралізація – це складний механізм. За інших рівних централізована база даних швидше і дешевше в розробці, ніж блокчейн.

Якщо корпорації зможуть зберігати довіру користувачів до своїх платформ, децентралізація може залишитися не затребуваною широким колом користувачів.

1.2 Блокчейн та реалізація роботи Ефіріуму

Блокчейн – це криптографічно безпечна транзакційна одноелементна система із загальним станом. Далеко не найпростіше визначення, чи не так? Давайте розіб'ємо кожну складову цієї ухвали на окремі частини.

«Криптографічно безпечний» означає, що безпека криптовалюти забезпечується складними математичними алгоритмами, які практично неможливо оминути. Захист, побудований за допомогою даних алгоритмів, є подібністю файрвола: завдяки алгоритмам, що використовуються, обхід системи безпеки практично неможливий (наприклад, неможливо створення підроблених транзакцій, знищення транзакцій тощо).

«Транзакційна одноелементна система» означає, що існує тільки один заданий стан системи, завдяки якому відбуваються всі транзакції, створювані в даній системі. Іншими словами, для даної системи передбачено лише один стан, який є єдино вірним.

«Зі загальним станом» означає, що стан, заданий у системі, є спільним та відкритим для всіх.

Транзакція є криптографічно підписаною частиною інструкції, яка спочатку задається зовнішнім обліковим записом, а потім упорядковується і передається блокчейн як показано на рисунку 1.1.

Усього існує два типи транзакцій: відправлення повідомлень та створення контракту (іншими словами такі транзакції створюють нові контракти в мережі Ефіріуму).

Усі транзакції містять такі елементи, незалежно від типу перших:

– nonce – кількість транзакцій, надісланих відправником;

– gasPrice – кількість Wei, яку відправник готовий віддати за одиницю пального, необхідну для здійснення угоди;

– gasLimit – максимальна кількість пального, яку відправник готовий заплатити за проведення цієї транзакції. Така сума задається та оплачується заздалегідь, перш ніж будь-які обчислення будуть проведені;

– to – адреса одержувача. На момент виконання транзакції, пов’язаної зі створенням контракту, адреси облікового запису контракту ще немає, тому замість нього використовується пусте значення;

– value – кількість Wei, які будуть передані від відправника до отримувача. У транзакціях, пов’язаних із створенням контрактів, ця величина є стартовим балансом для новоствореного облікового запису;

– v, r, s – дані позначення, які використовуються для створення підпису, який ідентифікує відправника транзакції;

– init – призначений лише транзакцій, що з створенням контрактів. Фрагмент EVM-коду, який використовується для ініціалізації новоствореного облікового запису контракту. init запускається лише один раз і надалі не використовується. Коли init запускається вперше, цей елемент повертає тіло коду облікового запису, яке є частиною коду, постійно пов’язану з обліковим записом контракту;

– data – це вхідні дані (параметри) для виклику повідомлення (data є необов’язковим елементом, призначеним лише для викликів повідомлень). Наприклад, якщо смарт-контракт відіграє роль служби реєстрації домену, виклик цього контракту може очікувати поля введення (наприклад, домен та IP-адреса).

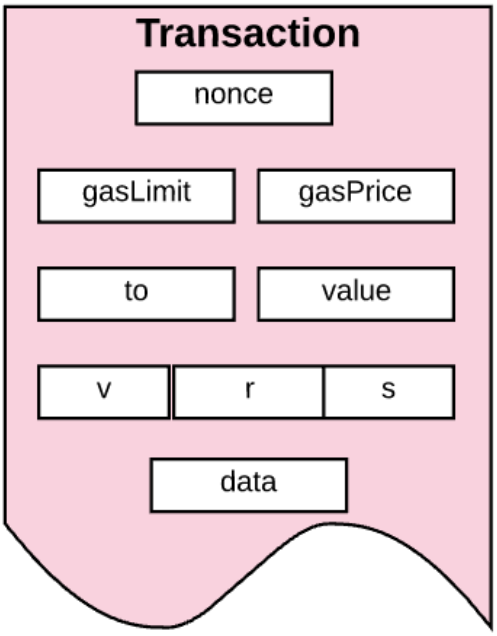


Рисунок 1.1 – Об’єкт транзакції

З інформації, наведеної в розділі «Облікові записи в блокчейні Ефіріум», ми з’ясували, що транзакції як для викликів повідомлень, так і для створення контрактів ініціюються зовнішніми обліковими записами, а потім перенаправляються в блокчейн. Іншими словами, транзакції – це своєрідний міст, що сполучає зовнішній світ та внутрішній стан платформи Ефіріуму.

Але це отже, що одні контракти що неспроможні взаємодіяти коїться з іншими: контракти, що у глобальному контексті стану Ефіріуму, можуть взаємодіяти друг з одним у межах даного контексту. Їхня взаємодія або спілкування відбувається за допомогою відправлення повідомлень або внутрішніх транзакцій. Єдина відмінність внутрішніх транзакцій від звичайних у тому, що перші створюються зовнішніми обліковими записами – але у результаті створення контрактів. Вони є віртуальними об’єктами, які, на відміну транзакцій, не впорядковуються і можуть існувати лише серед виконання Ефіріуму.

1.3 Смарт-контакти

Поряд із dApps, ще одна важлива частина, яку нам потрібно зрозуміти, щоб відповісти на питання «як працює Web3?» є смарт-контрактами.

Смарт-контракти – це кодифіковані угоди між двома сторонами. Таким чином, зміст смарт-контракту залишається таким самим, як і у традиційних контрактів; однак основна відмінність полягає в тому, що код блокчейне встановлює умови. Крім того, щоб зробити пояснення смарт-контрактів зрозумілішим, ми збираємося використовувати контракти Ethereum як приклад.

Усі смарт-контракти Ethereum зазвичай складаються із двох компонентів; код та дані. Код договору – це сукупність функцій, а дані – це договір. Всі смарт-контракти Ethereum знаходяться на певній адресі і є особливим типом облікового запису. Це дає можливість зберігати кошти і здійснювати транзакції. Однак важлива різниця між смарт-контрактами та звичайними обліковими записами полягає в тому, що перші управляються кодом.

Відмінною аналогією для роз’яснення та опису смарт-контрактів є торгові автомати, оскільки вони працюють аналогічно. Таким чином, ми можемо отримати бажаний результат через потрібне введення, як у традиційному торговому автоматі.

Більш того, розробники зазвичай пишуть смарт-контракти мовою програмування Solidity. Solidity – це об’єктно-орієнтована мова, призначена для розробки смарт-контрактів, а також сумісна з іншими блокчейнами на основі EVM. Однак є винятки і для інших мов, таких як Rust.

1.4 Постановка задачі

Актуальність даної теми полягає в тому, щоб створити вебзастосунок, який дозволить користувачеві самостійно створювати NFT в блокчейні Ethereum Ropston на базі смарт-контракту.

Об’єктом роботи є розробка веб застосунку створення NFT в блокчейн мережі Ethereum Ropsten за допомогою використання криптогаманця Metamask.

Метою роботи є розробка веб застосунку, який дасть можливість створювати NFT із зображення та його метаданих, використовуючи програмний гаманець.

Для досягнення мети необхідно вирішити такі завдання:

– вивчити існуючі технології для реалізації веб застосунку;

– огляд основних технічних моментів роботи з блокчейном;

– створити структуру проєкту;

– розробити методи: авторизації користувача, створення NFT у блокчейні з використанням власних контранктів, перегляд NFT користувача та їх пошуку за адресою гаманця, перегляд історії транзаків взаємодії з веб-применком;

– розробити клієнтську та серверну частину веб застосунку з використанням вибраних технологій;

– написати смарт-контракти і викласти їх у блокчейн мережу;

– розробити інструкцію, яка буде вказувати, як правильно користуватися даним вебзастосунком;

– зробити висновки, щодо виконаної роботи.

2 МАТЕМАТИЧНІ МОДЕЛІ ФІЛЬТРАЦІЇ ЗОБРАЖЕНЬ

2.1 Методи хешування даних

Перший метод хешування, який набрав високої популярності та широко використовується для хешування токенів безпеки – це хеш-функція SHA-256. SHA-256 (алгоритм безпечного хешування, FIPS 182-2) – це криптографічна хеш-функція з довжиною в 256 біти. Це хеш-функція без ключа, тобто код виявлення маніпуляцій. Повідомлення обробляється блоками 512 = 16 × 32 біта, кожен блок вимагає 64 раунди [27].

Розглянемо основні операції, які використовує цей тип шифрування:

– логічні операції *AND*, *XOR* та *OR*, позначені відповідно $\wedge$, $\oplus$ та $\vee$;

– побітове доповнення, що позначається $\neg$;

– ціле додавання за модулем 2^{32}, позначене $A + B$;

– *RotR (A, n)* позначає круговий зсув вправо на *n* бітів двійкового слова *A;*

– *ShR (A, n)* позначає зміщення вправо на *n* бітів двійкового слова *A*; − *A*||*B* позначає конкатенацію двійкових слів *A* і *B* [16].

64 двійкові раунди, задані 32 першими бітами дробових частин кубових коренів перших 64 простих чисел. Результат шифрування розглядається на рисунку 2.1 [27].

0x428a2f98	0x71374491	0xb5c0fbcf	0xe9b5dba5	0x3956c25b	0x59f111f1	0x923f82a4	0xab1c5ed5
0xd807aa98	0x12835b01	0x243185be	0x550c7dc3	0x72be5d74	0x80deb1fe	0x9bdc06a7	0xc19bf174
0xe49b69c1	0xefbe4786	0x0fc19dc6	0x240ca1cc	0x2de92c6f	0x4a7484aa	0x5cb0a9dc	0x76f988da
0x983e5152	0xa831c66d	0xb00327c8	0xbf597fc7	0xc6e00bf3	0xd5a79147	0x06ca6351	0x14292967
0x27b70a85	0x2e1b2138	0x4d2c6dfc	0x53380d13	0x650a7354	0x766a0abb	0x81c2c92e	0x92722c85
0xa2bfe8a1	0xa81a664b	0xc24b8b70	0xc76c51a3	0xd192e819	0xd6990624	0xf40e3585	0x106aa070
0x19a4c116	0x1e376c08	0x2748774c	0x34b0bcb5	0x391c0cb3	0x4ed8aa4a	0x5b9cca4f	0x682e6ff3
0x748f82ee	0x78a5636f	0x84c87814	0x8cc70208	0x90befffa	0xa4506ceb	0xbef9a3f7	0xc67178f2

Рисунок 2.1 – Результат шифрування у форматі SHA-256

Щоб переконатися, що результат шифрування, який був представлений вище на малюнку має довжину, кратну 512 бітам, спочатку додається біт 1,

далі додається k бітів 0, причому k є найменшим цілим додатним числом, таким, що $l + 1 + k \equiv 448$, де l – довжина в бітах вхідного значення.

Результуючий шифр завжди має бути заповнений, навіть якщо початкова довжина вже кратна 512 [28].

Далі розглянемо обчислення хешу. Спочатку для восьми змінних встановлюються початкові значення, задані першими 32 бітами дробової частини квадратних коренів перших 8 простих чисел.

$$\begin{aligned} H_1^{(0)} &= 0x6a09e667 \\ H_2^{(0)} &= 0xbb67ae85 \\ H_3^{(0)} &= 0x3c6ef372 \\ H_4^{(0)} &= 0xa54ff53a \cdot \\ H_5^{(0)} &= 0xa54ff53a \\ H_6^{(0)} &= 0x9b05688c \\ H_7^{(0)} &= 0x1f83d9ab \\ H_8^{(0)} &= 0x5be0cd19 \end{aligned} \qquad (2.1)$$

Далі блоки $M^{(1)}, M^{(2)}, M^{(3)}, \ldots, M^{(n)}$ обробляються по черзі згідно правилу, що $t = 1$ до N [16]. Конструкція для 64 блоків W_i з $M^{(t)}$, як було показано вище.

Наступний доволі популярний метод шифрування, який має назву SHA-1. Він використовується у вебзастосунку як метод шифрування паролю користувачів. Розглянемо детальніше цей метод шифрування даних [29].

SHA-1 – це криптографічна хеш-функція, яка вводить довільно довге повідомлення і видає 160-розрядний дайджест H. Для того, щоб надати дайджест повідомлення, SHA-1 діє вісімдесят разів на п’яти 32-розрядних словах A, B, C, D і E. Потрібно звернути увагу, що F_t визначається за формулою:

$$\begin{cases} F_0 = (B \wedge C) \vee \big((\neg B) \wedge D\big)\ t \in [0 \dots 19] \\ F_1 = \big(B \oplus C \oplus D\big)\ t\ \in [20 \dots 39] \\ F_2 = (B \wedge C) \vee (B \wedge D) \vee (C \wedge D)\ t\ \in [40 \dots 59] \\ F_3 = \big(B \oplus C \oplus D\big)\ t\ \in [60 \dots 79] \end{cases}. \quad (2.2)$$

Повідомлення *M* обробляється блоками розміром 512 біт, а саме шістнадцятьма 32-розрядними словами $W_0, \dots, W_{15}$, врешті-решт заповнення останнього блоку. Точніше, останній блок заповнюється одним бітом 1-го першого, тоді нуль або більше бітів 0, тому його довжина відповідає 448, за модулем 512. Решта 64 біти останнього 512-бітового блоку представляють довжину повідомлення *L*. Алгоритм SHA-1 розширює 32-розрядні слова $W_0, \dots, W_{15}$ на вісімдесят слів за допомогою функції планування наступних повідомлень:

$$W_i = ROT\ L(W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}), \quad (2.3)$$

де *ROTL(x, n)* – ліве обертання *x* на *n* бітів.

Можна звернути увагу, що рівняння вимагає зберігання вісімдесяти 32-розрядних слів. Якщо пам’ять обмежена (наприклад, вбудовані пристрої та графічні процесори), слід застосувати альтернативний метод.

Рівняння вимагає лише шістнадцяти слів, таким чином заощаджуючи шістдесят чотири 32-розрядні слова пам’яті. Можна замінити рівняння на інше рівняння:

$$W_{[i]} = \begin{cases} ROT\ L_1(W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}) & i \in [16 \dots 31], \\ ROT\ L_2(W_{i-6} \oplus W_{i-16} \oplus W_{i-28} \oplus W_{i-32}) & i \in [32 \dots 63], \\ ROT^{L_4}(W_{i-12} \oplus W_{i-32} \oplus W_{i-56} \oplus W_{i-64}) & i \in [64 \dots 79]. \end{cases} \quad (2.4)$$

Потім замінити W_{29}, W_{30}, W_{31}, W_{60} і W_{62} наступними і більш оптимізованим (зменшується кількість *XOR*) рівняннями:

$$\begin{cases} W_{29} = ROT\ L^2(W_{23}) \oplus k[29] \\ W_{30} = ROT\ L^2(W_{24}) \oplus k[16] \\ W_{31} = ROT\ L^2(W_{25}) \oplus k[29] \oplus k[31] \\ W_{60} = ROT\ L^4(W_{48} \oplus W_{28} \oplus W_0) \\ W_{62} = ROT\ L^4(W_{50} \oplus W_{30} \oplus W_0) \end{cases}. \quad (2.5)$$

В кінці формула (2.6) може бути розширена та переписана таким чином:

$$\begin{cases} W_{16} = W_0^1 \oplus W_2^1 \oplus W_8^1 \oplus W_{13}^1 \\ W_{17} = W_1^1 \oplus W_3^1 \oplus W_9^1 \oplus W_{14}^1 \\ W_{18} = W_2^1 \oplus W_4^1 \oplus W_{10}^1 \oplus W_{15}^1 \\ W_{19} = W_0^2 \oplus W_2^2 \oplus W_3^1 \oplus W_5^1 \oplus W_8^2 \oplus W_{11}^1 \oplus W_{13}^1 \\ W_{20} = W_1^2 \oplus W_3^2 \oplus W_4^1 \oplus W_6^1 \oplus W_9^2 \oplus W_{12}^1 \oplus W_{14}^1 \\ W_{21} = W_2^2 \oplus W_4^2 \oplus W_5^1 \oplus W_7^1 \oplus W_{10}^2 \oplus W_{13}^1 \oplus W_{15}^2 \\ W_{22} = W_0^3 \oplus W_2^3 \oplus W_3^2 \oplus W_5^2 \oplus \dots \oplus W_{11}^2 \oplus W_{13}^3 \oplus W_{14}^1 \\ W_{23} = W_1^3 \oplus W_3^3 \oplus W_4^2 \oplus W_6^2 \oplus \dots \oplus W_{12}^2 \oplus W_{14}^3 \oplus W_{15}^1 \\ \dots \\ W_{79} = W_0^8 \oplus W_0^{22} \oplus W_1^7 \oplus \dots \oplus W_{15}^{14} \oplus W_{15}^{17} \oplus W_{15}^{18} \end{cases}, \quad (2.6)$$

де $W^i_j = ROTL^j(W_i)$ [16].

Можна побачити, що хоча формула (2.6) має доволі більшу кількість операцій *XOR*, якщо шифрувати дані за допомогою алгоритмів PBKDF2HMAC-SHA-1, вони потребують зберігання лише п’яти 32-бітових слів, а саме

W0,…,W4 , так як *W6,…,W14* дорівнюють нулю, а *W5,W15* − постійні значення.

Даний метод хешування даних є більш надійний чим функція хешування SHA256, так як SHA-1 має більше етапів обробки вхідного повідомлення.

На рисунку 2.2 показано приклад зашифрованих даних за допомогою SHA-1 [16].

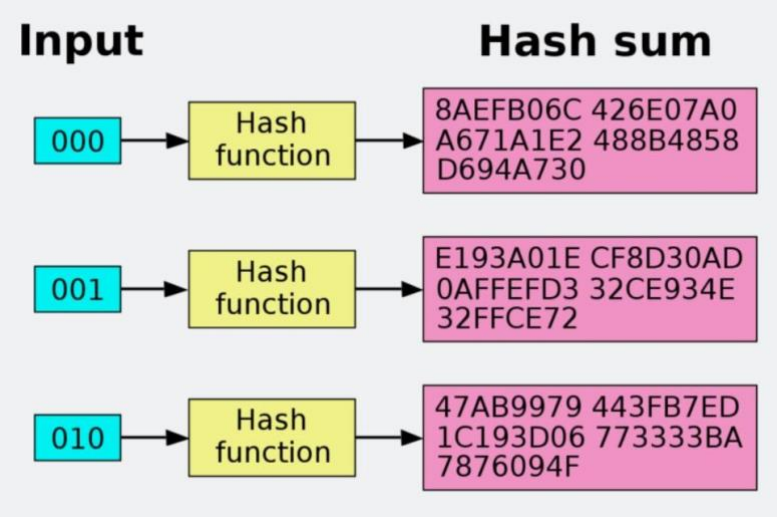


Рисунок 2.2 – Приклад хешу типу SHA-1

Було розглянуто два основних методи шифрування даних. Шифрування даних є невід'ємною частиною будь-якого вебзастосунку. Ці методи вже мають програмну імплементацію, яка входить в бібліотеку технології Spring.

Завдяки цьому можна легко шифрувати паролі та токени.

2.1 Облікові записи в блокчейні Ефіріум

Глобальний загальний стан платформи Ефіріум складається з багатьох невеликих об'єктів – облікових записів, які взаємодіють між собою за рахунок парадигми обміну повідомленнями. Кожен обліковий запис має певний стан і 20-байтову адресу. Адресою в Ефіріум є 160-бітний ідентифікатор, який використовується для ідентифікації будь-якого з облікових записів.

Усього існує два види облікових записів:

– зовнішні облікові записи контролюються за допомогою закритих ключів. У цьому такі записи немає жодного коду, що з ними;

– контрактні облікові записи контролюються спеціальним кодом, зазначеним в умовах контракту, і мають пов’язаний з ними код.

Давайте розберемося з основними відмінностями між зовнішніми та контрактними обліковими записами. Для зовнішнього облікового запису передбачено можливість надсилати повідомлення іншим зовнішнім обліковим записам, а також іншим контрактним обліковим записам. Для цього необхідно створити і зареєструвати нову транзакцію, використовуючи закритий ключ. Повідомлення між двома зовнішніми обліковими записами є лише значенням для передачі. З іншого боку, повідомлення, відправлене від зовнішнього облікового запису до контрактного, має на увазі активацію коду контрактного облікового запису, при цьому з’являється можливість здійснення певних дій (наприклад, за допомогою такого повідомлення можна перекладати токени, записувати значення у вбудовану пам’ять, створювати токени, виконувати деякі обчислення, створювати нові контракти тощо.

З допомогою контрактних облікових записів, на відміну зовнішніх, самостійно ініціювати нові транзакції неможливо. Натомість за допомогою контрактних облікових записів можна лише запускати транзакції у відповідь на інші отримані транзакції (наприклад, отримані із зовнішнього облікового запису або з іншого контрактного облікового запису) на рисунку 2.3.

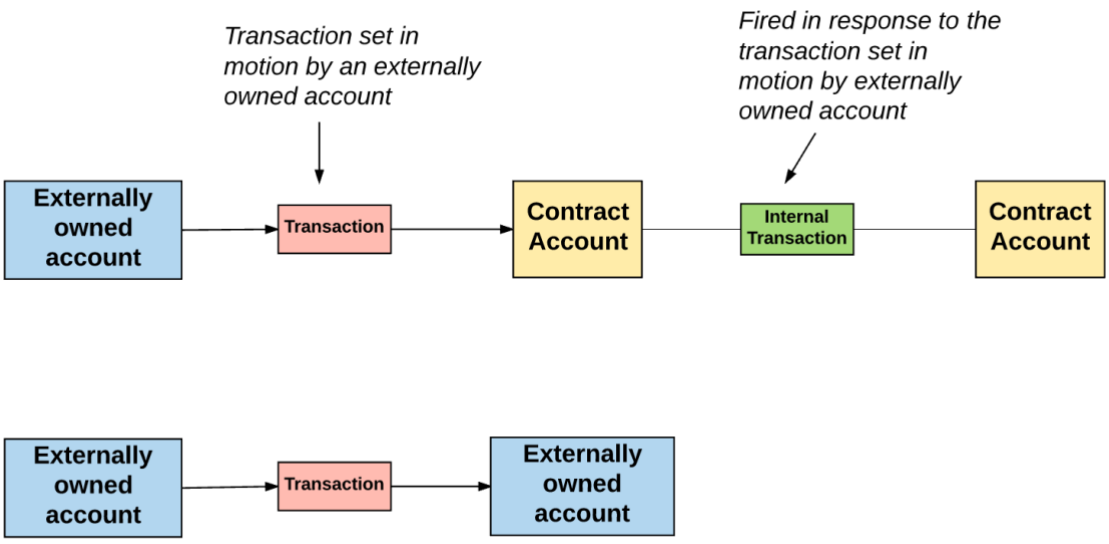


Рисунок 2.3 – Як облікові записи працюють із транзакціями

Кожна дія в блокчейні Ефіріуму відбувається завдяки транзакціям, що ініціюються зовні контрольованими обліковими записами.

2.2 Робота VM з транзакціями

Частина протоколу, який виконує обробку транзакцій в операційній системі Ефіріуму, називається віртуальною машиною Ефіріуму (BME).

ВМЭ является машиной Тьюринга, как это уже упоминалось в данной статье ранее. Единственное отличие BME от типичной машины Тьюринга заключается в том, что для работы первой требуется виртуальное «горючее». Таким образом, все вычисления, которые могут быть выполнены в BME, так или иначе ограничены количеством циркулирующего в ней, виртуальной машине, «горючего».

Розмір будь-якого елемента стека у BME дорівнює 256 бітам, а максимальний розмір стека досягає 1024 бітів.

Для BME передбачено певний обсяг пам’яті, який є постійним. У ньому елементи зберігаються як масивів байтів зі зверненням до слів.

Для BME також передбачено певну область зберігання. На відміну від обсягу пам’яті, таке сховище (або область зберігання) не змінюється і є частиною системи. У BME програмний код зберігається в окремій віртуальній ROM, доступ до якої можна отримати лише за допомогою певних інструкцій. З цієї точки зору, така BME відрізняється від типової архітектури фон Неймана, в якій програмний код зберігається в пам’яті комп’ютера.

Тепер перейдемо до транзакцій. Перед тим, як виконати певне обчислення, процесор повинен переконатися в тому, що наведена нижче інформація є валідною та доступною:

– стан системи;

– інформація про достатню для виконання необхідної операції кількість пального;

– адреса облікового запису, якому належить виконуваний код;

– адреса відправника транзакції – ініціатора виконання поточної операції;

– адреса облікового запису – ініціатора виконуваного коду (може відрізнятися від адреси відправника-ініціатора);

– інформація про потрібну для виконання транзакції кількості пального;

– вхідні дані для виконання операції;

– кількість Wei, яка має бути надіслана на рахунок цього облікового запису в результаті проведення поточної операції;

– інформація про машинний код, що виконується;

– інформація про заголовок блоку для поточного блоку;

– глибина виконання поточного повідомлення або створення контракту.

Безпосередньо на початок виконання програми пам’ять системи є абсолютно порожньою, а лічильник команд дорівнює нулю. Після чого у ВМЕ починається рекурсивне виконання транзакції: обчислення стану системи та стан машини для кожного циклу. Стан системи – це глобальний стан Ефіріуму. Стан машини включає:

– доступна кількість пального;

– лічильник команд;

– вміст пам’яті;

– активну кількість слів у пам’яті;

– контент стека.

Елементи стека додаються або видаляються з лівого краю фрагмента коду.

Для кожного циклу з кількості пального, що залишилася, віднімається його певна частина, при цьому лічильник команд збільшується.

Усього існує три можливі варіанти закінчення циклу:

– операції, що виконуються машиною, досягають виняткового стану (наприклад, через брак віртуального пального, неправильних інструкцій, недостатньої кількості елементів стека, значення елемента стека, що перевищує розмір 1024 біт, неправильного призначення JUMP/JUMPI) і, таким чином, процес виконання операції припиняється;

– послідовність дій переходить до виконання наступного циклу;

– операції, що виконуються машиною, досягають логічного завершення (завершення виконання процесу);

– у випадку якщо обчислення, що виконуються машиною, досягають логічного завершення, а не виняткового стану, то в результаті цього машина видає результуючий стан, а також інформацію про паливо, що залишилося, і результуючі вихідні дані.

Ось такі справи. Щойно ми з вами засвоїли найскладнішу та заплутану частину Ефіріуму. Не хвилюйтеся якщо ви чогось до кінця не зрозуміли: вам не потрібно вникати в кожну дрібницю і розуміти всі процеси, що відбуваються в даній системі, ну якщо тільки ви не збираєтеся по-справжньому повністю її вивчити і працювати на досить глибокому рівні.

2.3 Парадигма блокчейна платформи Ефіріум

Блокчейн Ефіріум є, власне, системою стану транзакцій. В інформатиці таке поняття, як «система станів» або «машина станів» – це система, яка обробляє інформацію, що вводиться, і на підставі останньої перетворюється на новий стан на рисунку 2.4.

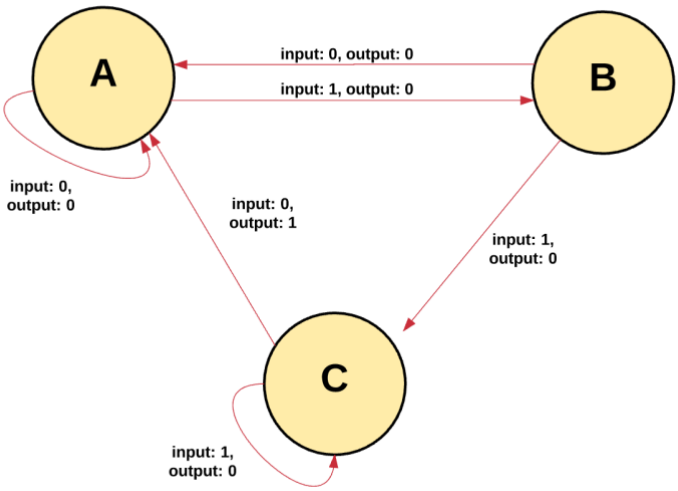


Рисунок 2.4 – Система стану транзакцій

У машині станів Ефіріуму всі процеси починаються з «первісного стану». Такий стан є аналогом нульового стану, в якому знаходиться машина до того моменту, як в її мережі почнуть відбуватися будь-які дії, пов’язані з транзакціями. Коли такі дії почнуть відбуватися, початковий стан замінюється на кінцевий, при цьому будь-якої миті часу кінцевий стан відображає поточний стан Ефіріуму на рисунку 2.5.

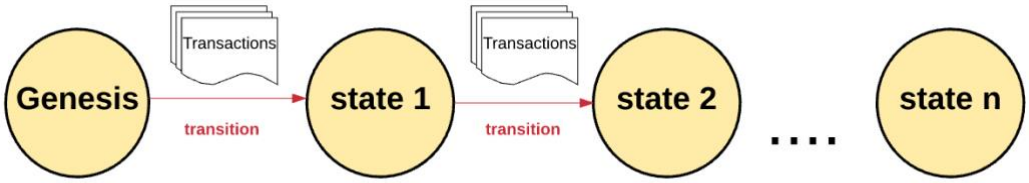


Рисунок 2.5 – Ланцюг станів

Стан Ефіріуму має мільйони транзакцій. Ці транзакції згруповані в блоки. Блок містить ряд транзакцій, при цьому кожен наступний блок з’єднаний з попереднім, завдяки чому забезпечується своєрідний ланцюжок блоків на рисунку 2.6.

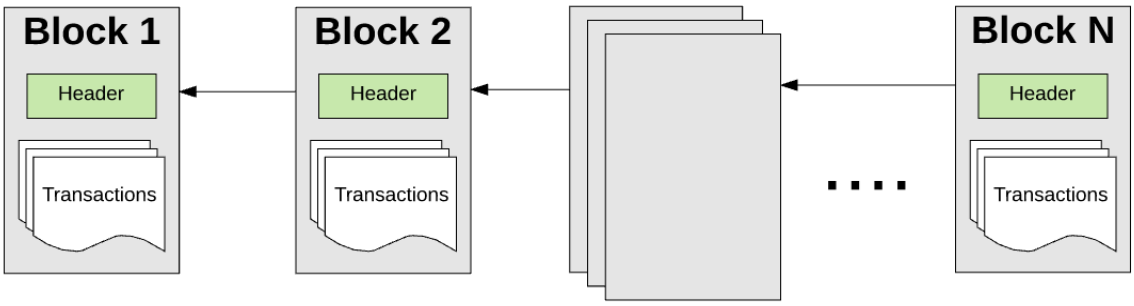


Рисунок 2.6 – Ланцюг блоків

Транзакція має бути коректною, щоб викликати її перехід із одного стану до іншого. Транзакція вважається коректною лише тоді, коли вона пройшла процес перевірки – так званий майнінг. Майнінг це коли група вузлів (комп'ютерів) витрачає свої обчислювальні ресурси для створення блоку коректних транзакцій.

Будь-який вузол у мережі, що оголошує себе майнером, може спробувати створити та перевірити блок транзакцій. Поширеним досвідом є спроби безлічі майнерів одночасного створення та перевірки блоку транзакцій.

Кожен майнер надає свій математичний «доказ» при відправленні блоку в блокчейн, і цей доказ виступає в ролі своєрідної гарантії: якщо доказ існує, транзакції в блоці вважаються коректними.

Майнер повинен надати свій математичний доказ швидше, ніж це зробить будь-який інший конкурент, щоб його блок був доданий в основний блокчейн. Процес перевірки кожного блоку, який полягає у наданні майнером свого математичного доказу, називається «доказом роботи».

Майнер, який обґрунтовує новий блок, отримує певну винагороду за виконання цієї роботи. Про яку винагороду йдеться? У блокчейні Ефіріуму використовується вбудований цифровий токен, який зветься «ефір» (від англ. Ether-»ефір»). Щоразу, коли майнер обґрунтовує свій блок транзакцій, створюється новий токен або новий ефір, а майнер отримує винагороду за його створення. Тоді, у вас може виникнути цілком логічне питання: де гарантія того, що кожен майнер буде дотримуватися лише одного ланцюжка блоків? Як

мені переконатися, що інша команда майнерів не вирішить створити свій власний ланцюжок блоків?

Виходячи з визначення «транзакційна одноелементна система із загальним станом» можна зробити висновок, що не буває двох і більш коректних поточних станів – воно є єдиним у своєму роді. Таким чином, кожен, хто бере участь у процесі обґрунтування нових блоків, має ухвалити це твердження за істину. Наявність кількох станів (чи ланцюгів) зруйнувало всю систему, бо було б неможливо домовитися у тому, який із станів є коректним. Наприклад, уявімо, що існувало б кілька ланцюжків блоків. Тоді, теоретично, ви могли б зібрати 10 монет на одному ланцюжку, на іншому – 20 монет, на третій – 40 монет тощо. У такому разі було б неможливо визначити, який ланцюг є найбільш «коректним» на рисунку 2.7.

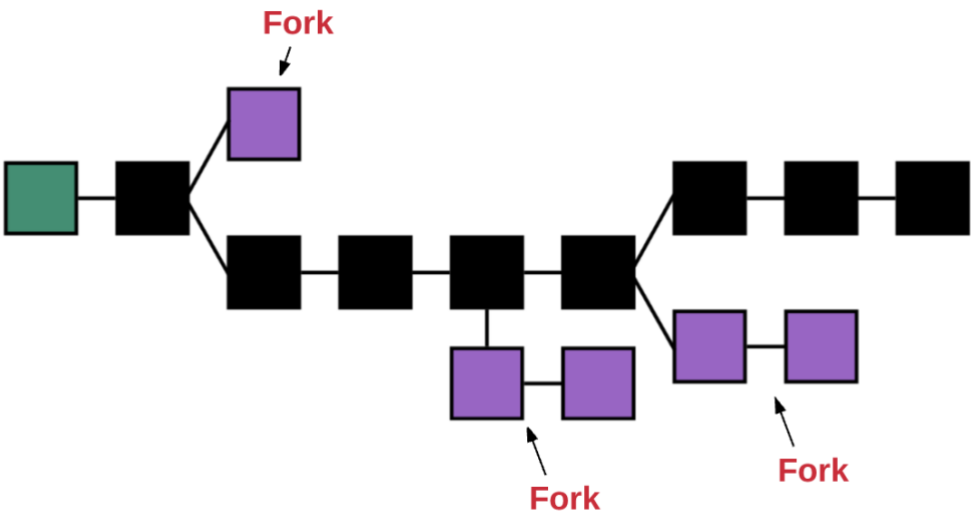


Рисунок 2.7 – Зображення системи розгалуження

Щоб визначити, який із можливих шляхів є коректним, і запобігти утворенню безлічі ланцюгів, в Ефіріумі застосовується метод, званий «протокол GHOST».

GHOST – «Жадібне-і-Саме-Весоме-з-Відомих-Дочірніх-Дерев» (Greedy Heaviest Observed Subtree).

Протокол GHOST оголошує, що ми маємо вибрати тільки той шлях, на якому було виконано найбільшу кількість обчислень. Для визначення такого шляху можна використовувати номер блоку, який був визначений останнім («листовий блок»). Завдяки такому підходу можна визначити загальну кількість блоків, що перебувають у поточному шляху (не враховуючи блок початкового стану). Чим вище блок, тим довший шлях і тим більше обґрунтувань повинні надати майнери. З таких міркувань, приймається єдино правильна версія для поточного стану.

Основні компоненти, з яких складається система Ethereum:

- учетные записи;
- состояние;
- горючее о вознаграждение;
- транзакции;
- блоки;
- выполнение транзакций;
- майнинг;
- обоснование.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАСТОСУНКУ

3.1 Інструменти для розробки програмного застосунку

Так як метою роботи є створення здебільшого вебзастосунку, і враховуючи, що найпопулярніша мова в цьому напрямку JavaScript. А так само, що мова якою написані смарт-контракти є дуже близькою до JavaScript.

Для вирішення даного завдання будуть обрані наступні інструменти розробки:

− мова програмування JavaScript. Як одна з найпоширеніших для розробки різного роду вебзастосунків;

− контейнер для збереження даних, а саме облікових записів користувачів та їх сесій буд представлений у вигляді бази даних Postgres SQL. Ця база даних доволі легко налаштовується під обрану вебтехнологію, так вона має необхідний набір функціоналу;

− технологія для публікації вебзастосунку у глобальній мережі буде представлений сервісом Heroku. За допомогою цього популярного та потужного сервісу вдасться виконати розгортання сайту швидко та безпечно;

− формат для передачі даних між клієнтом та сервером буде представлений форматом JSON. Він значно швидший та легший у використанні ніж формат XML;

− фреймворк для зручної роботи з базою даних Sequelize. Він дозволить використовувати спеціальні функції для взаємодії з базою даних;

− для того, щоб якісно реалізувати безпеку даного вебзастосунку, будуть використовуватися спеціальні токени, які дозволять зберігати зашифрований дані доступу до панелі взаємодії з основним функціоналом та буде контролювати сесію певного користувача, тобто протягом визначеного часу, користувач зможе користуватися своїми правами доступу без повторної авторизації;

− середовищем для вебзастосунку було обрано VSCode як найкраще середовище для програмування на мові програмування JavaScript;

− середовищем для смарт-контракту було обрано веб редактор Remix IDE як найкраще середовище для програмування на мові програмування Solidity.

3.2 Програмна реалізація

3.2.1 Серверна частина

Почнемо з ініціалізація серверної частини проекту. Для написання цієї частини було обрано фреймворк JavaScript під назвою NestJS на рисунку 3.1.

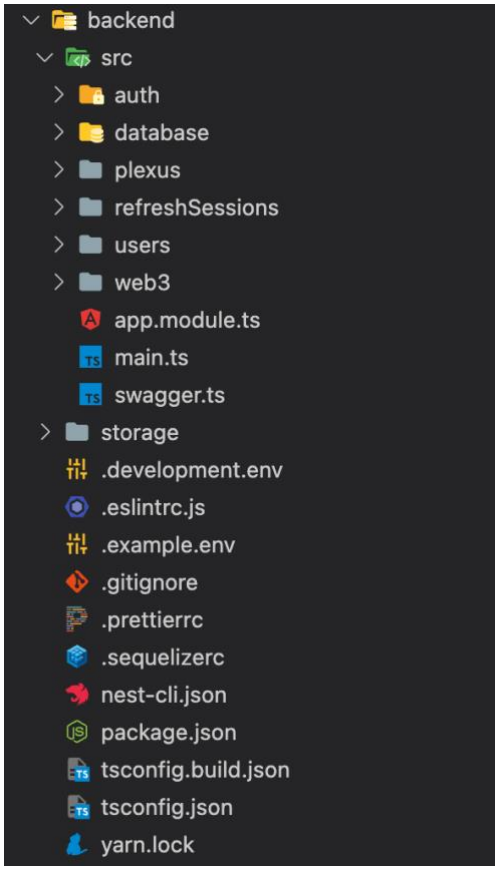


Рисунок 3.1 – Ієрахія серверної частини проекту

Головним файлом у якому задаються глобальні налаштування є «main.ts», його вміст зображено на рисунку 3.2.

Рисунок 3.2 – Лістинг коду головного файлу «main.ts»

На рисунку 3.2 можна побачити функцію bootstrap всередині якої відбувається:

– запуск основного модуля сервера;
– автоматичне створення swagger документації до API;
– дозвіл на роботу з файлами cookie за ключом «nft-data»;
– запуск на перевірку будь-яких запитів, що надходять;
– вилучення cookie із запитів;
– виведення в консоль порту після успішного запуску сервера.

Далі були зроблені модулі.

Модуль auth відповідає за авторизацію користувача, його структура зображена на рисунку 3.3.

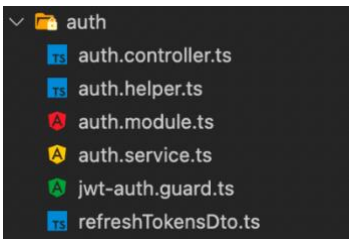


Рисунок 3.3 – Структура модуля авторизації

Контролер відповідає за ендпорти звернення клієнтської частини. У ньому відбувається виклик необхідних функцій для роботи з даними та відповідь клієнту. У контролері також описані налаштування роботи з cookie і токенами авторизації зображеними на рисунку 3.4.

```
@ApiTags('Authorization')
@Controller('auth')
export class AuthController {
    baseCookie = {
        httpOnly: true,
        secure: true,
        path: '/auth',
    };

    baseRefreshCookieOpt = {
        maxAge: Number(process.env.TOKEN_REFRESH_LIFETIME) * 1000,
        ...this.baseCookie
    };

    baseAccessCookieWallet = {
        maxAge: 60 * 1000 * 5,
        ...this.baseCookie
    };

    constructor(
        private authService: AuthService,
        private userService: UsersService,
        private web3Service: Web3Service,
    ) {
    }
```

Рисунок 3.4 – Лістинг коду налаштувань контролеру авторизації

Так само на рисунку 3.4 в конструкторі можна побачити сервірси з якими взаємодіє контролер авторизації. Для авторизації користувача клієнтська частина проекту повинна надіслати GET запит на адресу сервера за допомогою /auth/signing-data зображеному на рисунку 3.5

```
@Get('/signing-data')
async signingData(@Res({passthrough: true}) res: Response) {
    Logger.log('Signing data')
    const signData = randomString()
    console.log('signData', signData)
    res.cookie('sign-data', signData, this.baseAccessCookieWallet);
    return {signData};
}
```

Рисунок 3.5 – Лістинг коду контролеру GET /auth/signing-data

Цей контролер генерує випадковий рядок із 120 символів. Зберігає її в cookie відповіді, а також відповідає їй на запит. Це необхідно, щоб після підписання користувачем цього рядка своїм приватним ключем із гаманця. Сервер звіряючи початковий рядок і підписаний побачити публічний адрес користувача, за яким і відбувається індентефікація. Цей процес зображено на рисунку 3.6.

```
@Post('/wallet-verification')
async walletVerification(@Req() req: Request, @Res({passthrough: true})
res: Response, @Body() {
    signedData,
    fingerprint
}: WalletVerificationDto) {
    Logger.log('walletVerification')
    const signData: string = req.cookies['sign-data'];
    console.log('signData', signData)
    console.log('signedData', signedData)
    Logger.log('ecRecover')
    const walletAddress = this.web3Service.ecRecover(signData, signedData)
    console.log('userAddress', walletAddress)
    Logger.log('Get user by wallet address')
    const user = await this.userService.getUserByWalletAddress
    (walletAddress)
    console.log('user', user)
    if (user) {
        const {accessToken, refreshToken} = await this.authService.
        loginByWallet({walletAddress, fingerprint})
        res.cookie('refreshToken', refreshToken, this.baseRefreshCookieOpt);
        return {token: accessToken};
    } else {
        const {accessToken, refreshToken} = await this.authService.
        registrationByWallet({
            walletAddress,
            fingerprint,
        });
        res.cookie('refreshToken', refreshToken, this.baseRefreshCookieOpt);
        return {token: accessToken};
    }
}
```

Рисунок 3.6 – Лістинг коду контролеру POST /auth/wallet-verification

Контролер POST /auth/wallet-verification відповідає за реєстрацію та авторизацію користувача. Він отримує підписаний рядок, а також отримує першочерговий рядок який користувач підписував. Далі за допомогою функції recover web3, що надається бібліотекою, порівнюємо підписаний і початковий

рядок. Це дає публічний адрес користувача. На цю адресу ми можемо створити обліковий запис або авторизувати користувача. Для цього йде запит до бази даних, сутність користувача, яка буде описана нижче. Де ми просимо знайти повзувача з цією адресою. Якщо такого користувача ще не було, то створюється обліковий запис по і видається пара токенів для авторизації. Якщо ж такий користувач був, то відбувається атворизація, тим самим способом, створенням пари токенів. Пара токенів це access і refresh токен. Вони потрібні для сесій користувача. Їх роботи будуть описані нижче.

Для реалізації сесій користувачів використовують вже згадані токени. Access токен це якась хешована бібліотекою JWT рядок містить у собі дані, в нашому випадку це об’єкт з індетефікатором користувача в базі даних, а також записом що його ім’я це web3 аккаунт. Цей токен має в собі так само запис про тривалість його життя, після чого при його декодуванні можна дізнатися чи є він простроченим. Після закінчення його дії, ми звертаємося до refresh токену, який завжди лежить у cookie та скрипти браузера не мають до нього доступу. Його можна витягти тільки на сервері заздалегідь встановленим шляхом, яким він і буде прикріплюватися до запиту. Сам же токен складається з унікального рядка, fingerprint користувача і його індетифікатор. Всі ці дані зберігаються по суті сесія та оновлюються при оновленні сесії, або виходу користувача з акаунта. Сам же fingerprint є хешированим рядком, що містить у собі більшу частину даних про налаштування клієнта, завдяки яким можна визначити, чи є клієнт, що надіслав цей токен, тим самим котому він і був виданий.

Після успішної авторизації користувач і закінчення терміну «access» токена, клієнт буде звертатися до ендпоїнт POST /auth/refresh-tokens зображеного на рисунку 3.7.

```
@Post('/refresh-tokens')
async refreshTokens(@Req() req: Request, @Res({passthrough: true}) res:
Response, @Body() refreshDto: RefreshTokensDto) {
    Logger.log('REFRESH')
    const refreshTokenOld: string = req.cookies['refreshToken'];
    const {
        accessToken,
        refreshToken
    } = await this.authService.refreshTokens(refreshTokenOld, refreshDto.
    fingerprint);
    res.cookie('refreshToken', refreshToken, this.baseRefreshCookieOpt);
    return {token: accessToken};
}
```

Рисунок 3.7 – Лістинг коду контролеру POST /auth/refresh-tokens

Цей ендпоінт є єдиним, куди прихід cookie refreshToken. Отримуючи його ендпоінт перевіряє, чи є токен дійсним, тому видає нову пару токенів і оновлює запис у сутності сесія або ж дає помилку на клієнт. Також є ендпоінт POST /auth/loguot який відповідає за видалення сесії з сутності, якщо користувач вийшов з облікового запису, це зображено на рисунку 3.8.

```
@Post('/logout')
@HttpCode(200)
async logout(@Req() req: Request) {
    Logger.log('LOGOUT')
    const refreshToken: string = req.cookies['refreshToken'];
    if (refreshToken) {
        await this.authService.logout(refreshToken);
    }
    return;
}
```

Рисунок 3.8 – Лістинг коду контролеру POST /auth/logout

Також до цього модуля відноситься «JWT guard», який використовується в запитах інших модулів. Він відповідає за перевірку сесії користувача з «access» токеном. Виймаючи із заголовка запиту «access» токен за ключом «authorization», він перевіряється на час життя, а також ідентифікатор користувача. Без успішного проходження цього етапу, жоден ендпоінт, який використовує цей захист, не відпрацює і це зображене на рисунку 3.9.

```
async canActivate(context: ExecutionContext): Promise<boolean> {
  const req = context.switchToHttp().getRequest();
  try {
    const authHeader = req?.headers?.authorization || req?.handshake?.auth?.
    Authorization;
    const bearer = authHeader.split(' ')[0];
    const token = authHeader.split(' ')[1];

    if (bearer !== 'Bearer' || !token) {
      throw new Error();
    }
    const accessPayload: AccessTokenPayload = this.jwtService.verify(token);
    const user = await this.userService.getUserById(accessPayload.id);

    req.accessPayload = accessPayload;
    req.user = user;

    return !!user;
  } catch (e) {
    throw new UnauthorizedException({ message: 'User is not logged in' });
  }
}
```

Рисунок 3.9 – Лістинг коду контролеру «JWT guard»

Наступним етапом буде створення бази даних. Для взаємодії з базою даних використовуватиметься бібліотека Sequalize, яка надає зручний інтерфейс та методи. Підключення до бази даних відбувається через надання бібліотеці Sequalize необхідних параметрів. Так як сервер бази даних обов’язково повиннен мати логін та пароль для входу, то треба ввести однойменні параметри, та останнім параметром буде діалект, який використовує база даних (в даному випадку – це діалект для бази даних типу Postgres SQL).

Після налаштування відносин між базою даних та сервером переходимо до наступного етапу створення сутностей для проєкту. Необхідні параметри записані в змінні оточення і ініціалізуються в налаштування підключення до бази даних безпосередньо в головному модулі серверної частини програми. У цьому модулі відбувається підключення бібліотеки до бази даних, а також передача моделей (сутностей).

Далі буде створення сутності. Так як наш сервіс орієнтується на web3 акаунти, то і такі взаємодії з користувачем як вхід і реєстрація не вимагатимуть логін або пароль. Нам лише потрібно буде дати користувачеві підписати повідомлення своїм криптогаманцем. Я вибрав найпопулярніший гаманець Metamask для реалізації взаємодії. Тому в сутності «User» нам необхідно

зберігати лише публічну адресу користувача. На рисунку 3.10 зображено структуру таблиці «User», що відповідає за сутність користувачів.

```
@Table({tableName: 'users'})
export class User extends Model<User> {
    @ApiProperty({example: '1', description: "Unique identifier"})
    @Column({type: DataType.INTEGER, autoIncrement: true, primaryKey: true})
    id: number;

    @ApiProperty({example: '0xeae7a8a8a1df8a75e656728005h39ud9k183c996',
    description: 'walletAddress'})
    @Column({type: DataType.STRING, unique: true, defaultValue: null,
    allowNull: true})
    walletAddress: string;
}
```

Рисунок 3.10 – Лістинг коду який описує сутність «User»

Для вже описаної роботи сесії була створена сутність «refreshSession» зображена на рисунку 3.11.

```
@Table({ tableName: 'refreshSessions' })
export class RefreshSession extends Model<RefreshSession> {
  @Column({ type: DataType.INTEGER, autoIncrement: true, primaryKey: true })
  id: number;

  @Column({ type: DataType.INTEGER, allowNull: false })
  userId: number;

  @Column({ type: DataType.STRING, unique: true, allowNull: false })
  refreshToken: string;

  @Column({ type: DataType.STRING, allowNull: false })
  fingerprint: string;

  @Column({ type: DataType.DATE, allowNull: false })
  expiresIn: number;

}
```

Рисунок 3.11 – Лістинг коду який описує сутність «refreshSession»

Оскільки створення NFT на контракті проекту необхідно мати сигнатру підписану приватним ключем акаунта творця контракту, було створити модуль plexus який зображенно на рисунку 3.12.

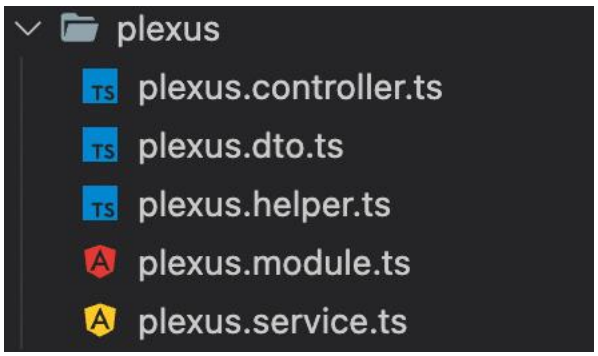


Рисунок 3.12 – Структура модуля «plexus»

Контролер цього модуля має лише один ендпоінт та зображен на рисунку 3.13.

```
@Post('/nft-data')
@UseInterceptors(FileInterceptor('image', imageOptions))
@ApiConsumes('multipart/form-data')
@UseGuards(JwtAuthGuard)
@HttpCode(200)
async nftGeneration(@UploadedFile() image: any, @Body() req: EmbedPlexusDto,
@Res() res: Response, @UserDecorator() { walletAddress }: User): Promise<any>
{
  Logger.log('NFT Generation')
  if (!walletAddress) throw new HttpException('User does not have
  walletAddress', HttpStatus.NOT_FOUND);
  const { signature, tokenId, hashImage, filename } = await this.
  plexusService.getSignFromFile(image, req, walletAddress);
  res.setHeader('Content-Type', exts[filename.split('.')[1]]);
  res.setHeader('nft-data', JSON.stringify({
    hashImage: hashImage,
    description: req.description,
    walletAddress,
    itemName: req.itemName,
    tokenId,
    url: req.url,
    signature,
  }));

  return fs.createReadStream(path.resolve(__dirname, '../../storage/' +
  filename)).pipe(res);
}
```

Рисунок 3.13 – Лістинг коду контролеру POST /plexus/nft-data

Він приймає запит POST з наступними даними:

– файл (зображення формату jpg, png та розміром до 50мб);

– form-data яка містить поля «itemName», «description», «url».

Якщо клієнт звернувся, і його сесія активна, викликається функція «getSignFromFile» сервісу модуля, яка зображена на рисунку 3.14.

```
async getSignFromFile(
  image: fileDto,
  req: EmbedPlexusDto,
  walletAddress: string,
): Promise<{ signature: string, tokenId: string, hashImage: string, filename:
string }> {
  const filename = await this.saveImageFromBuffer(image, 'storage');
    const hashImage = new BN(await this.getHashImage(path.resolve(__dirname,
    '../../storage/' + filename)), 16);
    const tokenId: string = Web3Service.randomHex();
    const { signature } = await this.sign(req, tokenId, hashImage,
    walletAddress);
    return {
      signature,
      tokenId,
      hashImage: hashImage.toString(),
      filename,
    }
}
```

Рисунок 3.14 – Лістинг коду функції «getSignFromFile» з сервісу модуля «plexus»

Функція отримує зображення, об’єкт з даними з надісланої «form-data» і адресу користувача. Зберігає отриманий файл на корені сервера, в папці «storage». Далі викликає функцію отримання хеша зображення у форматі sha256 і переводить у формат даних BigNumber. Далі генерується випадковий рядок формату Hex. Для отримання підпису викликається функція «sign» із сервісу модуля. У функцію передається об’єкт із метаданими NFT, який складається з: «itemName», «description», «url», «tokenId», хеш зображення та публічна адреса користувача.

Далі відокремлюємо префікс 0x від індетефікатора токену та переводимо в ІшBigNumber. Записуємо у змінну «tokenId_BN». Також записуємо в змінну метод «web3.utils.solidity Sha3», що надається web3 бібліотекою, для отримання хеша переданого значення у форматі keccak256. Далі отримуємо хеш від усіх хеш даних NFT. І наприкінці підписуємо отриманий хеш за допомогою методу «web3.eth.accounts.sign», що надається web3 бібліотекою, куди першим параметром передаємо хеш, а другим приватний ключ гаманця який викладав контракт у блокчейн мережу. Таким чином ми отримуємо об’єкт, що містить сигнатуру NFT. Тепер, маючи всі необхідні дані, ми можемо відповісти клієнту, передавши зображення та об’єкт усіх необхідних даних для створення NFT.

Модуль «plexus» і сервіс web3, описаний вище, необхідний для реалізації маханізму захисту створення NFT на контракті проекту, і є необов’язковим. Це гарантує нам, що це NFT будуть випущені через обслуговування проекту, а чи не якісь можливі способи звернення до договору.

3.2.2 Клієнтська частина

Клієнтська частина була заснована на JavaScript-бібліотеці під назвою React, з ієрархією файлів зображеної на рисунку 3.15.

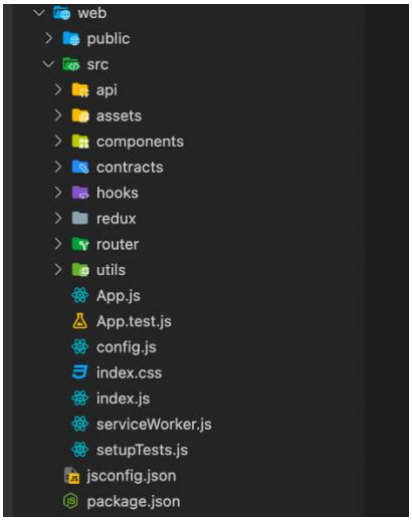


Рисунок 3.15 – Ієрархія файлів на клієнтській частині

У папці «api» створюється інстанс функції, що відповідає за запити до серверної частини, а також виступає посередником для прикріплення та оновлення токена сесії.

У папці «assets» розташовані файли, що відповідають за стилізацію компонентів, а також весь медіа контент.

У папці «components» розташовані всі компоненти клієнтської частини, з яких складаються сторінки. Архітекрут передбачає багаторазове перевикористання компонентів для оптимізації та мінімізації коду. Таким

чином було створено папку «shared_components», в якій розташовані всі компоненти з яких складаються сторінки.

Клієнтська частина має 4 основних сторінок:

– головна, яку користувач до і після входу в обліковий запис;

– сторінка створення NFT (Create);

– сторінка Транзакцій (Transactios);

– сторінка перегляду NFT користувача та їх пошук за адресою гаманця користувача (Explorer).

На головній сторінці користувач може увійти до облікового запису. При натисканні кнопки авторизації або «Get start now» буде викликаний метод «walletVerification», код якого зображений на рисунку 3.16.

```
const walletVerification = async () => {
  try {
    if (web3) {
      await dispatch(userActions.signIn.call());
      await dispatch(userActions.stage.current('Receiving data for signature')
      );
      const signdataRes = await api.get('/auth/signing-data');
      const { data: signdata } = signdataRes;
      const acc = await getAccount();
      await dispatch(userActions.stage.current('Waiting for signing'));
      const signedData = await signMessage(signdata.signData, acc);
      const fingerprint = await Fingerprint.get();
      await dispatch(userActions.stage.current('Verification'));
      const walletLoginRes = await api.post('/auth/wallet-verification', {
      signedData, fingerprint });
      const { data } = walletLoginRes;
      Token.set(data.token);
      await dispatch(userActions.signIn.success(Token.getDecode()));
      await dispatch(userActions.stage.reset());
      await history.push(routes.home);
    } else {
      await initWeb3();
    }
    await dispatch(userActions.stage.reset());
  } catch (error) {
    await dispatch(userActions.signIn.failure(error));
  }
};
```

Рисунок 3.16 – Лістинг коду функції «walletVerification»

Для розпізнавання користувача сервісом, користувачеві необхідно підписати дані своїм приватним ключем із гаманця. Це дозволить сервісу маючи початковий і підписаний рядок вичислити адресу гаманця, що і буде авторизацією та реєстрацією. Для цього клієнтська частина надсилає запит на сервер, щоб отримати випадковий рядок, який також буде збережено в «cookie» з прапорами «httpOnly» і «secure», що зробить «cookie» безпечними.

Отриманий рядок віддає користувачеві на підписання, що зображено на рисунок 3.17.

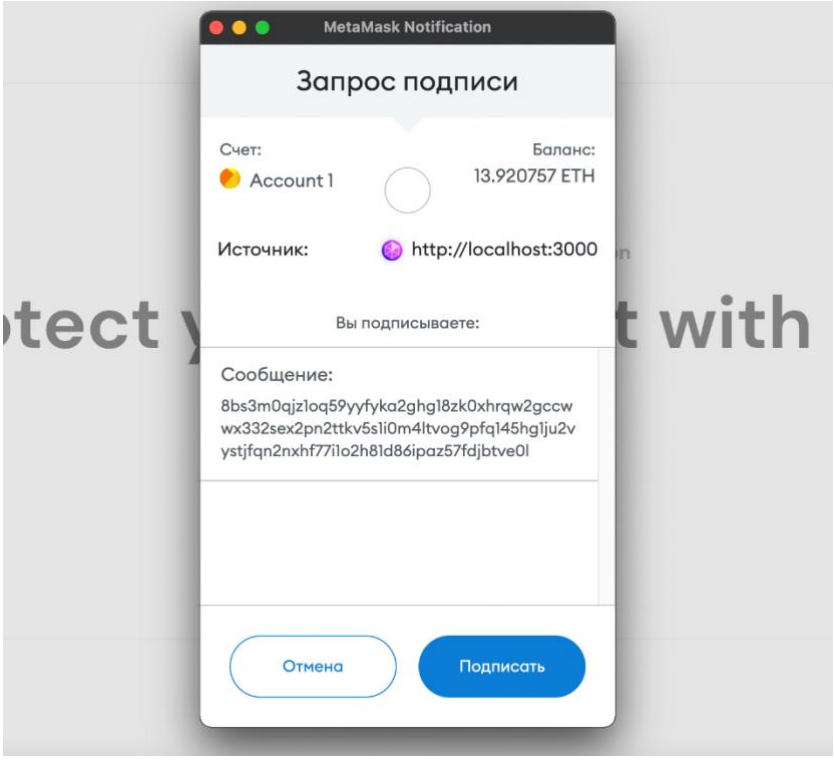


Рисунок 3.17 – Підписання користувачем

Підписаний рядок клієнт посилає на сервер, де йде порівняння даних та обчислення адреси. У відповіді клієнт отримує токени авторизації.

Після авторизації користувач може перейти на сторінку створення NFT, яка зображена на рисунку 3.18.

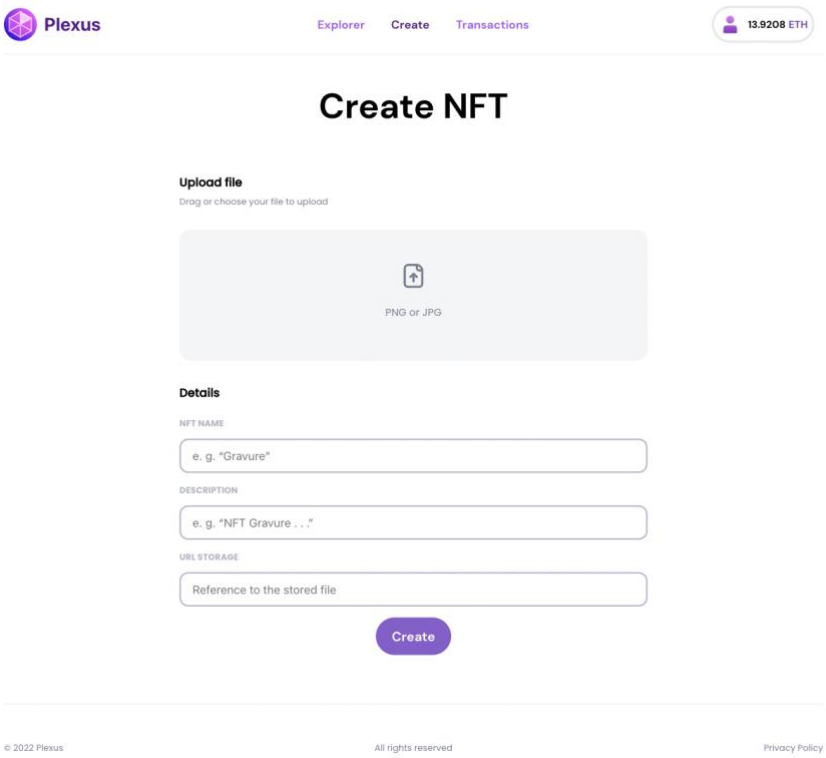
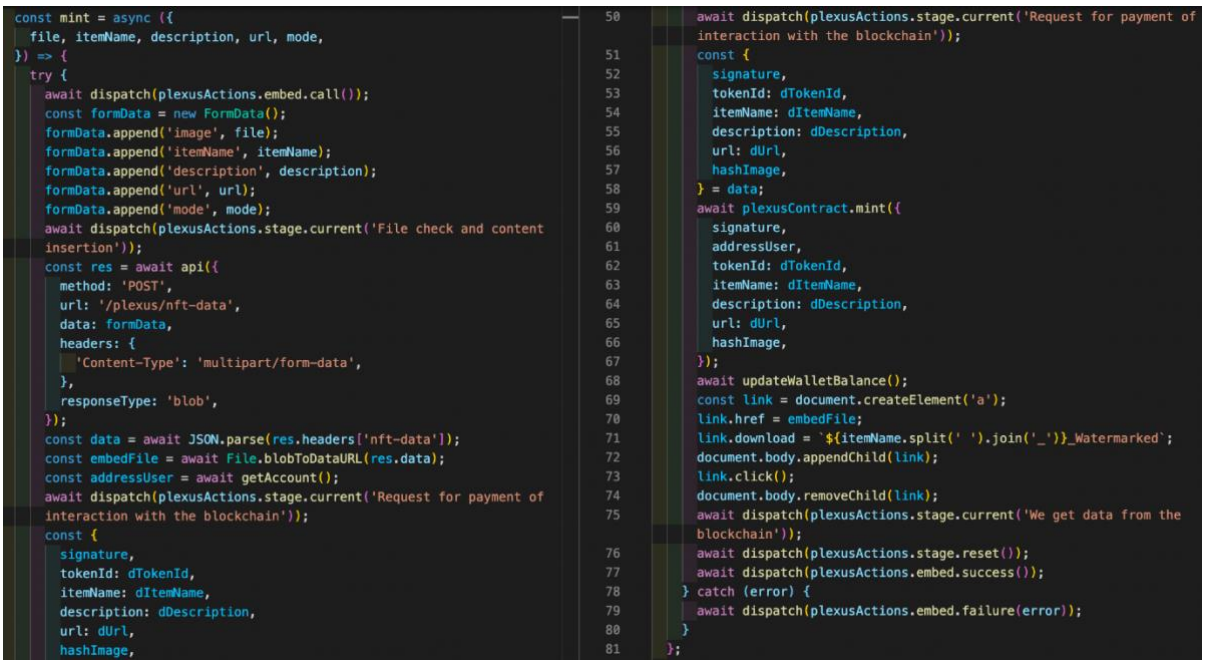


Рисунок 3.18 – Сторінка створення NFT

На ній користувачеві необхідно прикріпити файл формату png або jpg, а також заповнити обов’язкові поля, що відповідають за посилання на вихідний файл, його ім’я та опис. Як тільки форми буде заповнена, користувач зможе почати кнопку створення, чим буде викликаний метод mint зображений на рисунку 3.19.



```
const mint = async ({
  file, itemName, description, url, mode,
}) => {
  try {
    await dispatch(plexusActions.embed.call());
    const formData = new FormData();
    formData.append('image', file);
    formData.append('itemName', itemName);
    formData.append('description', description);
    formData.append('url', url);
    formData.append('mode', mode);
    await dispatch(plexusActions.stage.current('File check and content
    insertion'));
    const res = await api({
      method: 'POST',
      url: '/plexus/nft-data',
      data: formData,
      headers: {
        'Content-Type': 'multipart/form-data',
      },
      responseType: 'blob',
    });
    const data = await JSON.parse(res.headers['nft-data']);
    const embedFile = await File.blobToDataURL(res.data);
    const addressUser = await getAccount();
    await dispatch(plexusActions.stage.current('Request for payment of
    interaction with the blockchain'));
    const {
      signature,
      tokenId: dTokenId,
      itemName: dItemName,
      description: dDescription,
      url: dUrl,
      hashImage,
      await dispatch(plexusActions.stage.current('Request for payment of
      interaction with the blockchain'));
      const {
        signature,
        tokenId: dTokenId,
        itemName: dItemName,
        description: dDescription,
        url: dUrl,
        hashImage,
      } = data;
      await plexusContract.mint({
        signature,
        addressUser,
        tokenId: dTokenId,
        itemName: dItemName,
        description: dDescription,
        url: dUrl,
        hashImage,
      });
      await updateWalletBalance();
      const link = document.createElement('a');
      link.href = embedFile;
      link.download = `${itemName.split(' ').join('_')}_Watermarked`;
      document.body.appendChild(link);
      link.click();
      document.body.removeChild(link);
      await dispatch(plexusActions.stage.current('We get data from the
      blockchain'));
      await dispatch(plexusActions.stage.reset());
      await dispatch(plexusActions.embed.success());
    } catch (error) {
      await dispatch(plexusActions.embed.failure(error));
    }
};
```

Рисунок 3.19 – Лістинг коду функції mint

Функція mint створить об’єкт типу «FormData», і відправить його на сервер для отримання підписаної приватним ключем сигнатури цих даних. Це потрібно виключення можливості створення NFT на контаркті сервісу, без його використання. Після відповіді сервера клієнт викликає метод контракту, який і дозволить створити NFT в блокчейні. Сам контракт та його методи будуть описані нижче, у розділі 3.2.3. Виклик цього методу передається на гаманець користувача, якому необхідно підтвердити транзакцію створення NFT, це зображено на рисунку 3.20.

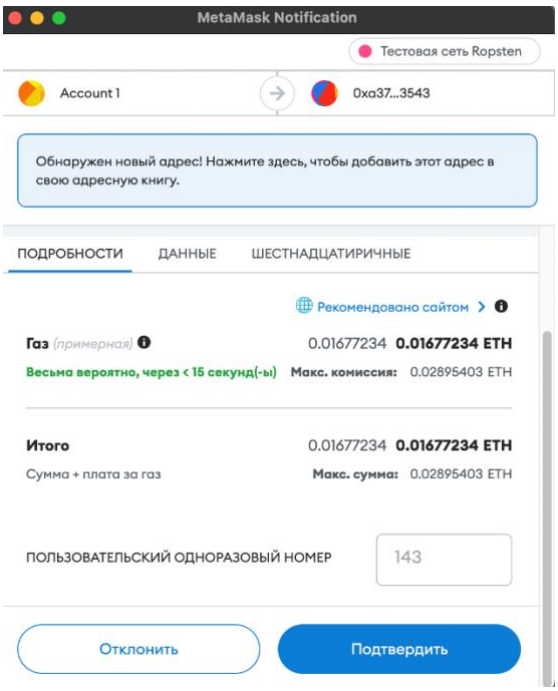


Рисунок 3.20 – Підтвердження транзакції користувачем

У період виконання транзакції користувач зможе взаємодіяти з інтерфейсом сервісу. Після завершення транзакції користувач зможе побачити створену ним NFT на вкладці «Explorer» у списку NFT, які йому належать, це зображено на РС 3.21.

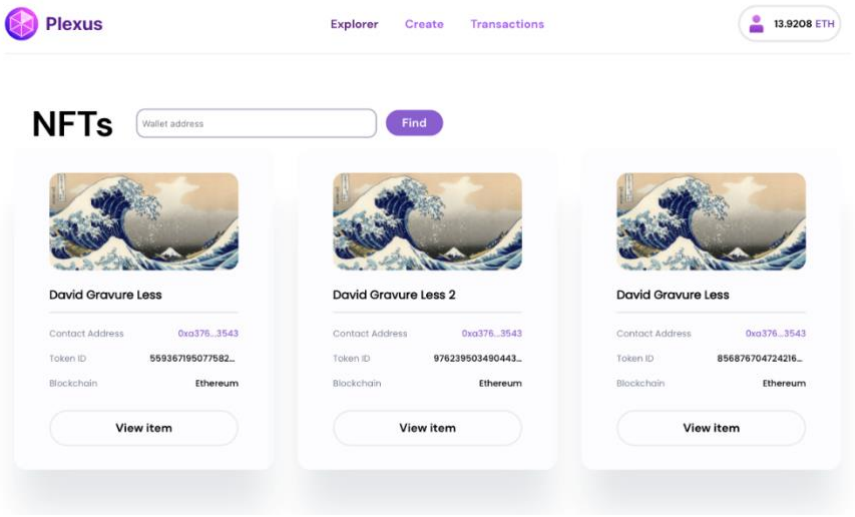


Рисунок 3.20 – Сторінка «Explorer»

Цей список виводиться завдяки функції «getCollection» зображеного на рисунку 3.21.

```
const getCollection = async (payload) => {
  try {
    await dispatch(plexusActions.collection.call());
    await dispatch(plexusActions.stage.current('Getting a list of watermarks
    from the blockchain'));
    if (payload?.ownerId) {
      const res = await plexusNodeContract.getAllDataOfOwnerTokens(payload?.
      ownerId);
      await dispatch(plexusActions.collection.success(res.map((item) => ({ ...
      item, contractAddress }))));
    } else {
      const addressUser = await getAccount();
      const res = await plexusContract.getAllDataOfOwnerTokens(addressUser);
      await dispatch(plexusActions.collection.success(res.map((item) => ({ ...
      item, contractAddress }))));
    }
    await dispatch(plexusActions.stage.reset());
  } catch (error) {
    if (error.message === 'Returned error: execution reverted:
    getOwnerImages: invalid balance') {
      toast.error('This user does not have NFTs');
    } else if (error.message.substr(0, 20) === 'bad address checksum') {
      toast.error('Bad address checksum. Check if the wallet entered is
      correct!');
    }
    await dispatch(plexusActions.collection.failure(error));
  }
};
```

Рисунок 3.21 – Лістинг коду функції «getCollection»

Функція «getCollection» викликає метод «getAllDataOfOwnerTokens» із контракту, куди передається адреса гаманця користувача. Який повертає нам масив NFT користувача, який виводиться на сторінці Explorer.

При натисканні на кнопку «View item», користувачеві відкриється вкладка сервісу перегляду даних у блокчейн мережі Ethereum Ropsten на сторінці відкритої ним NFT, це зображено на рисунку 3.22.

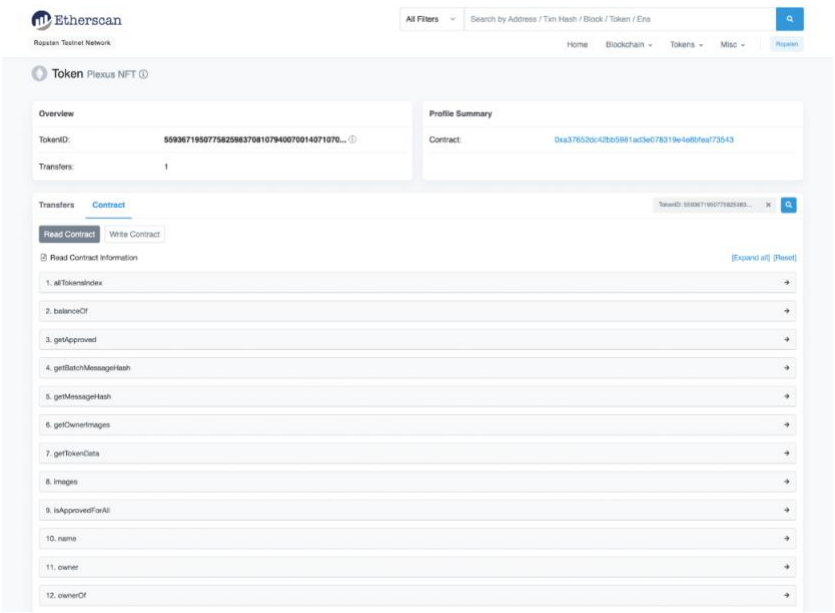


Рисунок 3.22 – Сторінка сервісу перегляду даних у блокчейн мережі Ethereum Ropsten

На сторінці «Transactions» користувач може побачити свої транзакції щодо створення ним NFT, це зображено на сторінці 3.23.

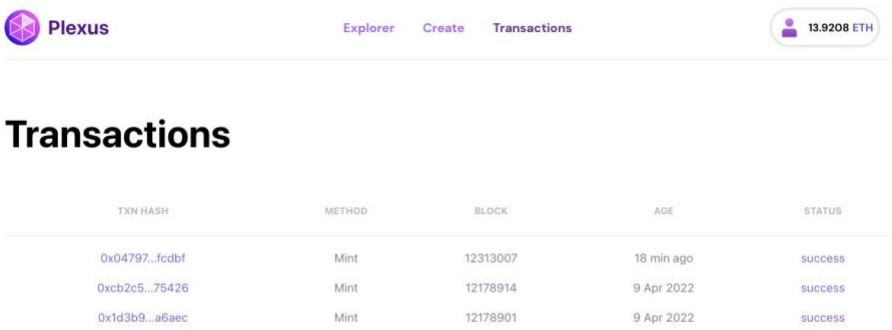


Рисунок 3.23 – Сторінка «Transactions» з транзакціями користувача

У папці «contracts» знаходиться «abi» та адресу договору сервісу, вони необхідні для створення інстансу договору на клієнті. Це дозволить взаємодіяти з методами договору.

Вміст папки «hooks» зображено на рісунку 3.24.

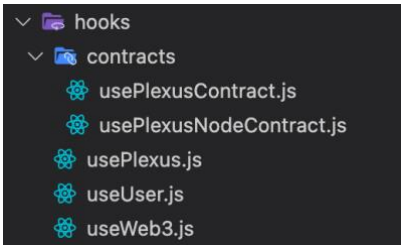


Рисунок 3.24 – Вміст папки «hooks»

У файлі «usePlexusContract.js» розташована функція «mint» зображена рисунку 3.19.

У файлі «usePlexusNodeContract.js» розташовані функції отримання даних про NFT, а також колекції NFT користувача.

У файлі «usePlexus.js» розташовані функції обгортки для двох описаних вище файлів.

У файлі «useUser.js» розташовані функції входу, реєстрації, виходу та оновлення сеансу користувача.

У файлі «useWeb3.js» розташовані функції взаємодії з користувальницьким крипто гаманцем.

3.2.3 Смарт-контракт частина

Для написання смарт контракту з створення NFT було використано веб-середовище розробки Remix IDE. Контракт буде базуватися на загальноприйнятих шаблонних контраках, що дозволить максимально скоротити написання базового функціоналу. Спочатку була написана функція отримання хешу NFT з її метаданих. Отриманий рядок після підписання приватним ключем гаманця, який задеплоїв контракт, можна буде використовувати як гарант, що NFT буде випущена з використанням серверної частини, а не прямою взаємодією з контрактом через сторону послуги. Ця функція зображена на рисунку 3.25.

```
43    /**
44     * @dev Returns encoded message hash for mint() function.
45     */
46    function getMessageHash(
47        address _to,
48        uint256 _tokenId,
49        string memory _tokenName,
50        string memory _description,
51        string memory _tokenURL,
52        uint256 _imageHash
53    ) public pure returns (bytes32) {
54        return keccak256(abi.encodePacked(
55                keccak256(abi.encodePacked(_tokenId, _to)),
56                keccak256(abi.encodePacked(_tokenName)),
57                keccak256(abi.encodePacked(_description)),
58                keccak256(abi.encodePacked(_tokenURL)),
59                keccak256(abi.encodePacked(_imageHash))
60            ));
61    }
```

Рисунок 3.25 – Лістинг коду функції «getMessageHash»

Якщо всі перевірки успішно пройдені, тоді об’єкт з метаданими про NFT додається масив стану контракту. Що дозволить надалі працювати з цим

масивом взяття інформації про NFT, у ньому зберігатися все NFT та його інформація. А так само буде створено токен стандарту «ERC721», що належить гаманцю, що підтвердив транзакцію.

Для отримання інформації про NFT з контракту було написано функцію «getTokenData», яка перевіряє наявність токена за його унікальним індетефекатором. Якщо токен існує, то повертає його метадані. Ця функція зображена на рисунок 3.26.

```
function getTokenData(uint256 _tokenId) external view returns (Image memory) {
    require(_exists(_tokenId), "getTokenData: Nonexistent token");

    uint256 tokenIndex = allTokensIndex(_tokenId);

    return images[tokenIndex];
}
```

Рисунок 3.26 – Лістинг коду функції «getTokenData»

Для отримання інформації про всі NFT з контракту була написана функція «getOwnerImages», яка повертає масив NFT, що належать адресі гаманця, переданому в функцію. Ця функція зображена на рисунку 3.27.

```
function getOwnerImages(address _owner) external view returns (Image[] memory) {
    uint256 balance = balanceOf(_owner);
    // safe gas
    require(balance > 0, "getOwnerImages: invalid balance");

    Image[] memory ownerImages = new Image[](balance);

    for (uint i = 0; i < balance; i++) {
        uint256 tokenIndex = allTokensIndex(tokenOfOwnerByIndex(_owner, i));
        ownerImages[i] = Image(images[tokenIndex].id,
        images[tokenIndex].name,
        images[tokenIndex].description,
        images[tokenIndex].image,
        images[tokenIndex].imageHash);
    }
    return ownerImages;
}
```

Рисунок 3.27 – Лістинг коду функції «getOwnerImages»

Це всі функції, необхідні для роботи сервісу. Після їх написання договір був скомпільований завдяки відповідному функціоналу, що надається веб застосунку Remix IDE. Який зображено на рисунку 3.28.

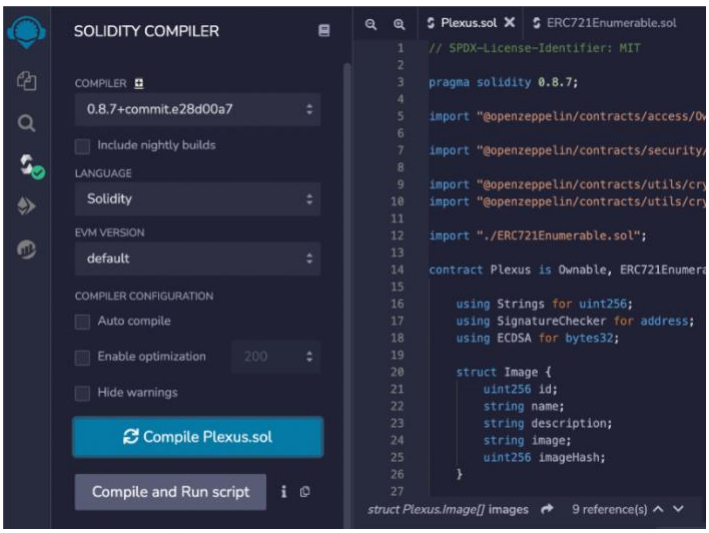


Рисунок 3.28 – Інтерфейс веб застосування Remix IDE для компілювання контракту

Маючи скомпільований договір, він був закріплений в блокчейн завдяки відповідному функціоналу наданим веб застосунку Remix IDE натиснутою кнопки «Deploy» і підтвердженням транзакції гаманцем. Цей процес зображено на рисунку 3.29.

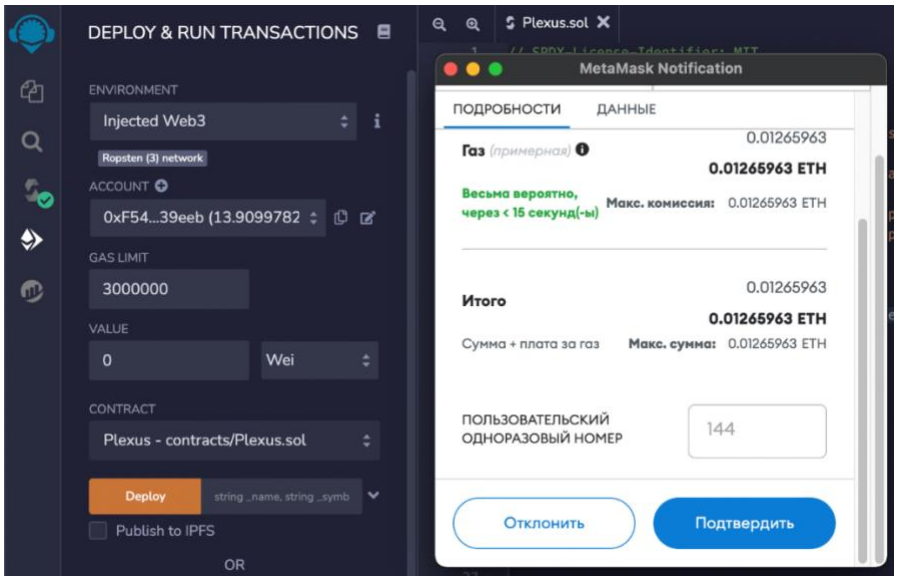


Рисунок 3.29 – Зображення процесу деплою контракту

Після успішного деплою контракту було отримано його адресу «0xa37652Dc42bB5981AD3e078319e4E8BfeaF73543». А гаманець виконав

деплой стає власником контракту, що дозволяє використовувати приватний ключ для перевірки підпису.

3.3 Інструкція користувача

У наступних розділах буде описана взаємодія користувача з веб застосуном, починаючи від авторизації. Для використання веб-застосування його необхідно запустити локально, цей процес описаний у окремому файлі.

3.3.1 Авторизація

Після переходу на адресу де запущена клієнтська частина застосунку, користувач побачить головну сторінку. Де він зможе авторизуватися за допомогою гаманця натисканням кнопки «Authorization» або «Get started now». Дана сторінка зображена на рисунку 3.30.

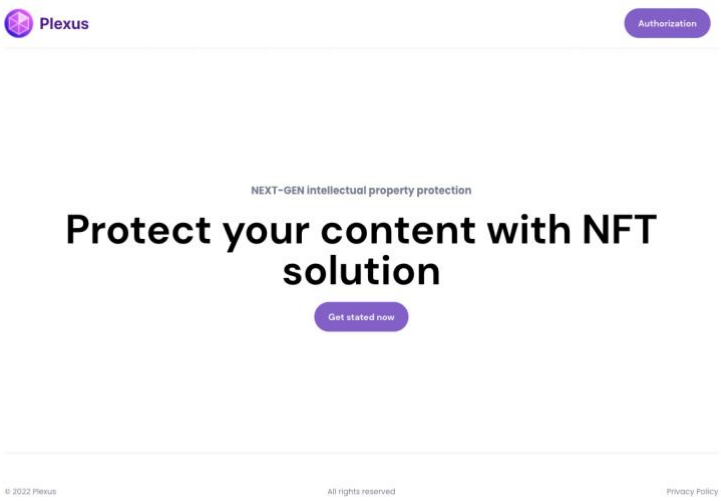


Рисунок 3.30 – Головна сторінка

Після ініціалізації авторизації користувачеві спливе вікно Metamask для підписання рядка, цей процес зображений на рисунку 3.31.

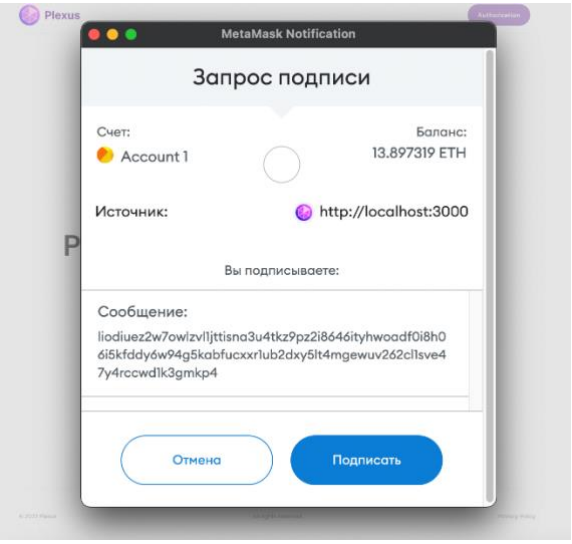


Рисунок 3.31 – Запит на підпис користувача

Після підписання рядка, користувач авторизує, і він побачить головну сторінку зі зміненим навігаційним меню. Тепер користувач може побачити баланс свого гаманця, а також перейти на сторінки:

– створення NFT, вона ж «Create»;

– перегляду своєї колекції або пошуку колекції на адресу гаманця, вона ж «Explorer»;

– Перегляд транзакцій, пов’язаних зі створенням NFT, вона ж «Transactions».

3.3.2 Створення NFT

Після натискання кнопки «Create» у навігаційному меню. Користувач перейде на сторінку створення NFT. Де він може побачити, а також заповнити форму з наступними полями:

– «file», де користувачеві необхідно вибрати локальний файл у форматі png або jpg;

– «NFT Name», де користувачу необхідно вказати назву, яка буде присвоєна NFT;

– «Description», де користувачу необхідно вказати опис, який буде присвоєно NFT;

– «URL Storage», де користувачеві необхідно вказати посилання, яке веде на місце зберігання прикріпленого ним файлу.

Приклад заповненої форми можна побачити на рисунку 3.32.

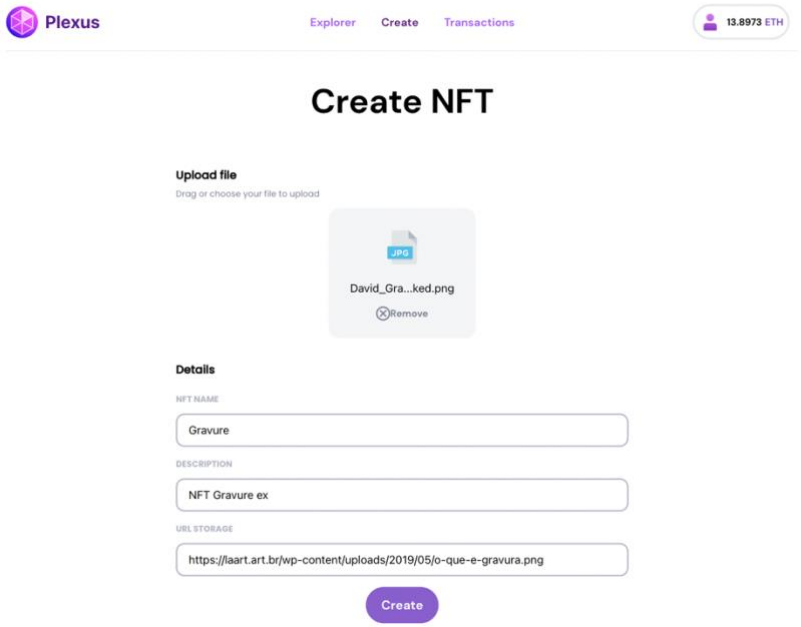


Рисунок 3.32 – Сторінка «Create» із заповненою формою створення NFT

Під заповненою формою користувач може побачити, як виглядатиме картка NFT після її створення в роздалі «Explorer», що зображено на рисунку 3.33.

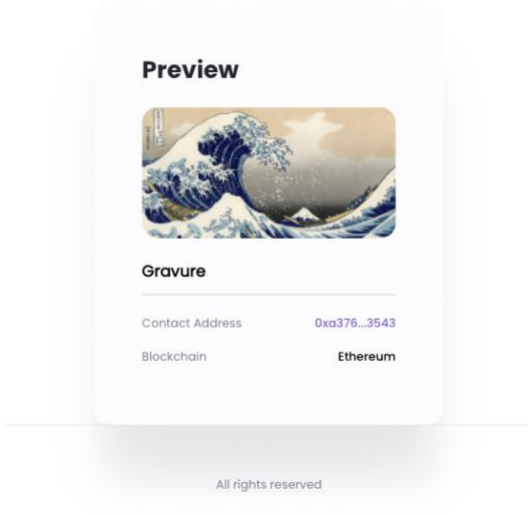


Рисунок 3.33 – Приклад картки після заповнення форми створення NFT

Коли форму заповнено, користувач може розпочати кнопку «Create». Що запустить процес перевірки даних на відповідність вимогам, і якщо введені дані коректні, користувач побачить підписання транзакції Metamask, що спливло вікна, що зображеному на рисунку 3.34, інакше отримає повідомлення про невідповідне поле.

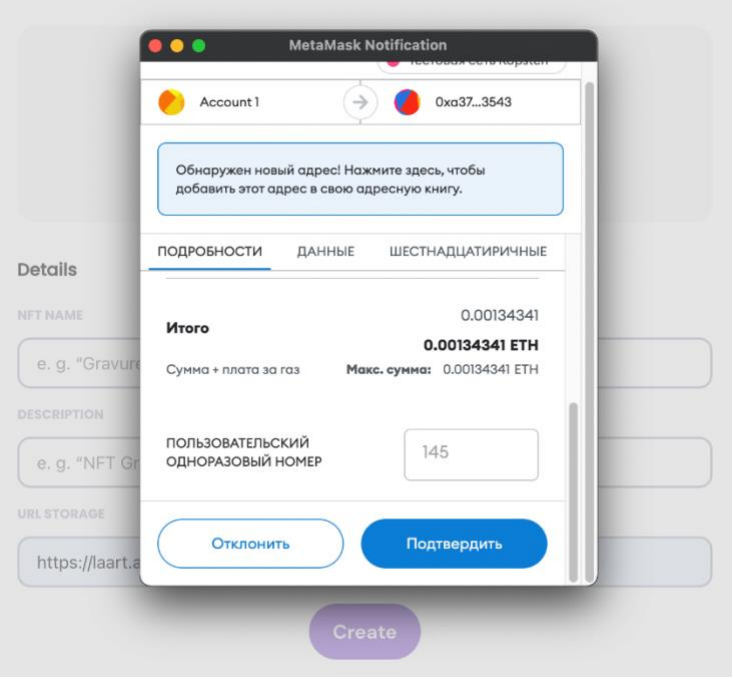


Рисунок 3.34 – Підписання користувачем транзакції створення NFT

Під час виконання транзакції інтерфейс буде заблокований, шляхом відображення завантаження на всі веб-інтерфейсу, до її завершення. Як тільки транзакція виконається, користувач розблокує можливість взаємодії з інтерфейсом. Він зможе створити наступну NFT або перейти на вкладку перегляду своєї колекції, натиснувши кнопку «Explorer» у навігаційному меню.

3.3.3 Перегляд колекції NFT користувача

На сторінці «Explorer» зображені створені користувачем NFT, а також є поле для пошуку NFT інших користувачів за адресою їх гаманця. Це можна побачити на рисунку 3.35.

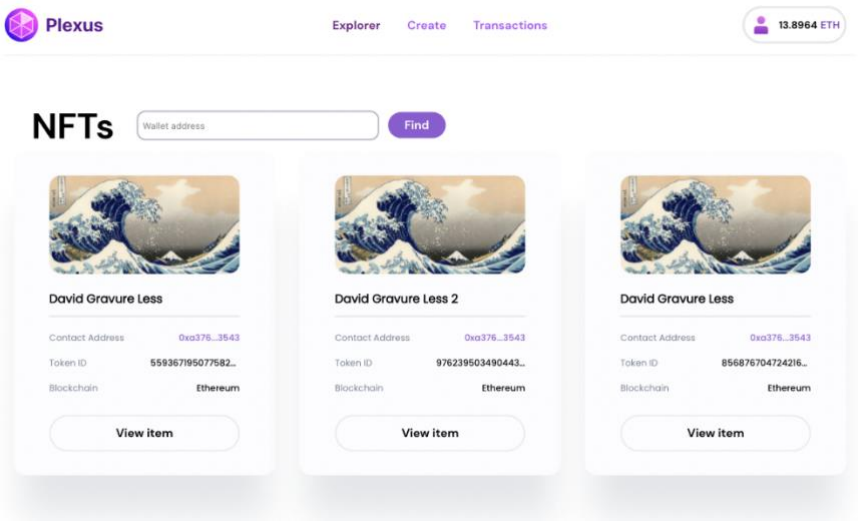


Рисунок 3.35 – Сторінка «Explorer» з картками NFT користувача

Натисканням на кнопку «View item», яка розташована в нижній частині кожної картки NFT, користувач може перейти на сторінку NFT у сервісі перегляду даних у блокчейні Ethereum Ropsten, що зображено на рисунку 3.36.

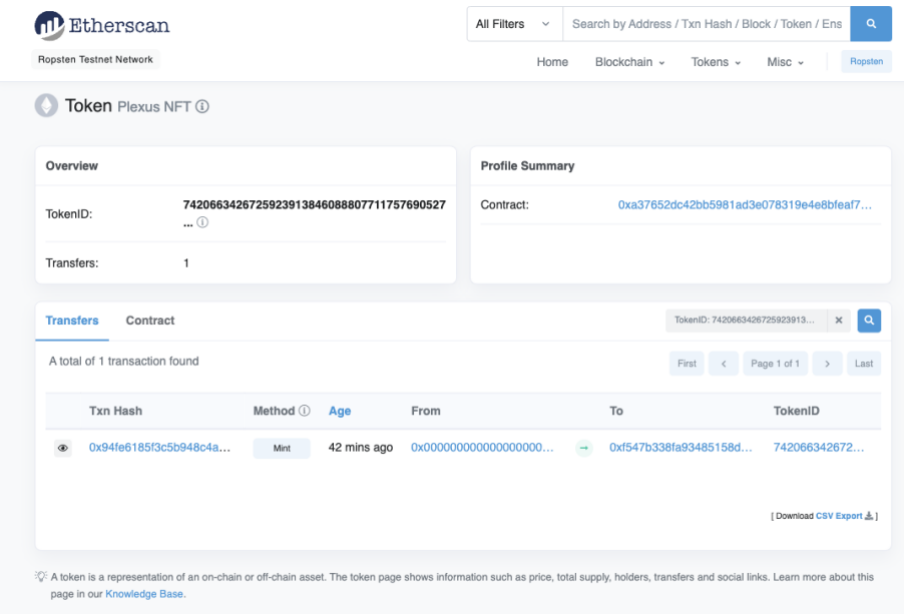


Рисунок 3.36 – Сторінка сервісу перегляду даних у блокчейні Ethereum Ropsten під назвою «Etherscan»

На сторінці сервісу «Etherscan» на розділі взаємодії з NFT користувач може відкрити вкладку «Contract», і викликати метод «getTokenData», вказавши унікальний індентефекатор NFT щоб отримати дані про NFT. Це можна побачити на рисунку 3.37.

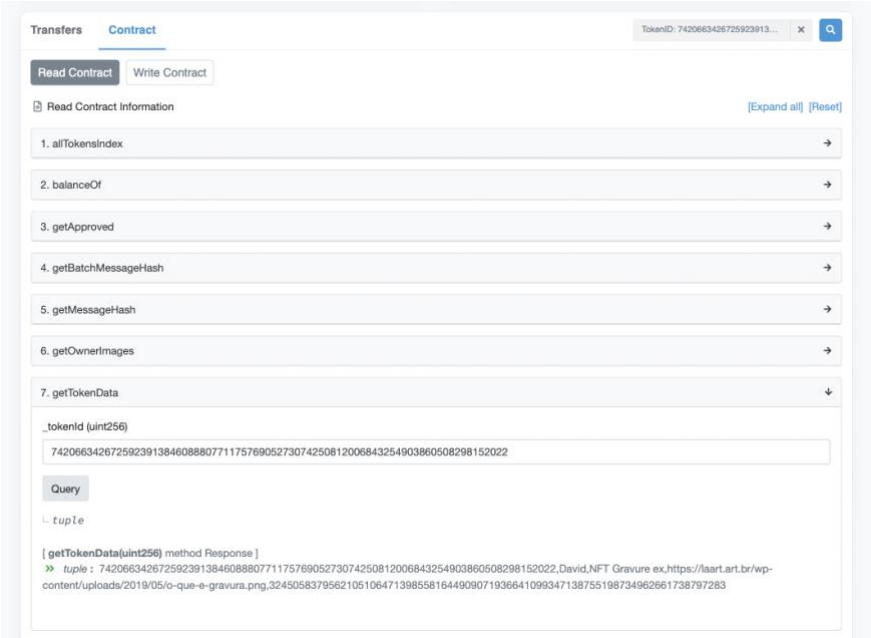


Рисунок 3.37 – Взаємодія з методами NFT на сторінці сервісу «Etherscan»

3.3.4 Перегляд історії транзакцій створення NFT

На сторінку «Transactions», користувач може потрапити, натиснувши відповідну кнопку навігації. На цій сторінці він може побачити список усіх транзакцій з створення NFT, що зображено на рисунку 3.38.

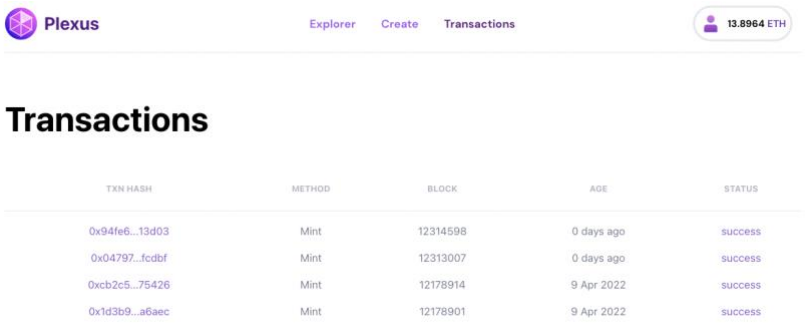


Рисунок 3.38 – Сторінка з історією країни користувача

Натисканням на хеш транзакції користувач буде перенаправлено на сторінку перегляду деталей транзакції у сервісі «Etherscan», що зображено на рисунку 3.39.

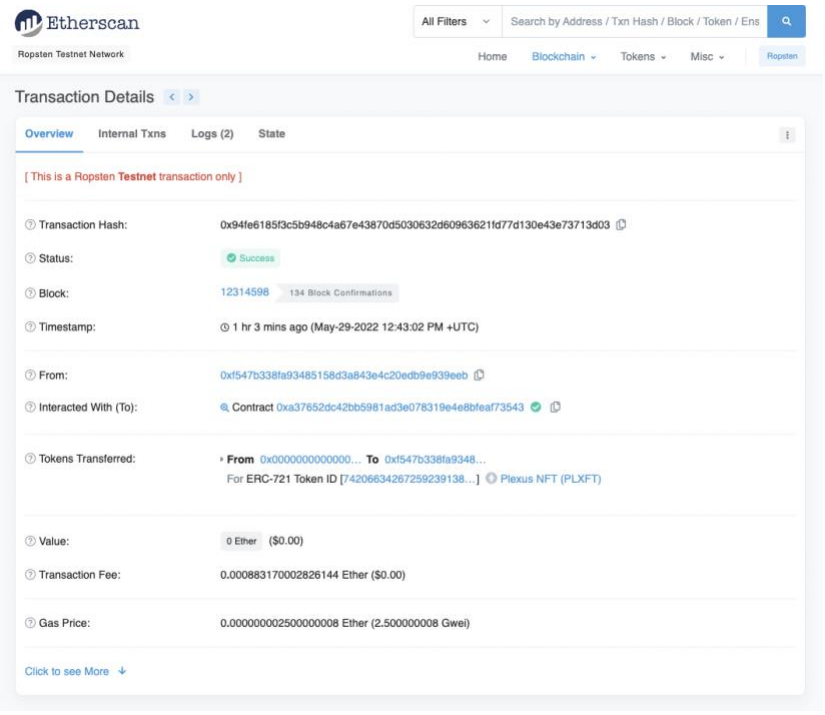


Рисунок 3.39 – Сторінка перегляду деталей транзакції сервісу «Etherscan»

ВИСНОВКИ

У рамках кваліфікаційної роботи був розроблений і реалізован веб застосункок за для створення NFT в блокчейн мережі Ethereum Ropsten за допомогою використання криптогаманця Metamask. Клієнтська та серверна частина була написана мовою програмування JavaScript, у середовищі програмування VSCode. Смарт-контракти були написані мовою програмування Solidity, в середовищі програмування Remix IDE, за допомогою якого і були викладені в мережу. Також були вивчені існуючі технології для реалізації веб застосунків. На основі вивченого матеріалу були обрані найбільш сучасні та унівесальні рішення.

Вивчена та використана основна технічна інформація роботи з блокчейно.

Були розроблені методи, в саме авторизації користувача, створення NFT у блокчейні з використанням власних контранктів, перегляд NFT користувача та їх пошуку за адресою гаманця та перегляд історії транзакій взаємодії з веб застосунком.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Бекірова Е. А., Халілова З. Е. (2019) Основні етапи створення вебзастосунків (укр.) - Державна бюджетна освітня установа вищої освіти Республіки Крим «Кримський інженерно-педагогічний університет імені Февзі Якубова» (Сімферополь), С. 84-91. - ISSN 2658-5944.

2. Раджпут Д. (2019) Spring. Усі патерни проєктування, 320 с.

3. Лонг Д., Бастані До. (2019) Cloud Native Java: Designing Resilient Systems with Spring Boot, Spring Cloud, and Cloud Foundry., 624 с.

4. Волков А. С., Волкова К. А. (2019) Огляд архітектурних компонентів сучасного вебзастосунку, 961 с.

5. Jake Spurlock (2013) Bootstrap. Responsive Web-Development, 128 с.

6. David Cochran, Ian Whitley (2014) Bootstrap Site Blueprints., 304 с.

7. David Cochran. (2012) Twitter Bootstrap Web Development How-To., 68 с.

8. Фрімен Ерік, Фрімен Елізабет (2010) Head First HTML with CSS & XHTML., 656 с.

9. Пітер Лабберс, Брайан Олберс, Френк Салім (2011) HTML5 для професіоналів: потужні інструменти для розробки сучасних вебдодатків., 272 с.

10. Макфарланд, Девід. (2015) JavaScript та jQuery: вичерпне керівництво., 880с.

11. Стівен Шафер. (2011) HTML, XHTML, та CSS Bible, 5th Edition., 656 с.

12. Ніксон Р. (2016) Створюємо динамічні вебсайти за допомогою PHP, MySQL, JavaScript, CSS та HTML5. 4-те вид., 768 с.

13. Peter Shaw. (2014) witter Bootstrap 3 Succinctly, 110 с.

14. Джеймс Р. Грофф, Пол Н. Вайнберг, Ендрю Дж. Оппель. (2014) SQL: The Complete Reference, Third Edition., 960 с.

15. Кріс Фіайлі. (2003) SQL: Посібник з вивчення мови., 456 с.

16. К. Дж. Дейт. (2005) Введення у системи баз даних, 1328 с.

17. Kinoshenko, D., Mashtalir, V., & Shlyakhov, V. (2007). A partition metric for clustering features analysis.

18. Kussul, N., Skakun, S., Kravchenko, O., Shelestov, A., Gallego, J. F., Kussul, O., ... & Sokolov, G. Архитектурно-структурные особенности средств автоматизации процесса извлечения знаний из естественно-языковых текстов.

19. Kinoshenko, D., Mashtalir, S., Stephan, A., & Vinarski, V. (1993). Neural Network Segmentation Of Video Via Time Series Analysis. INFORMATION THEORIES & APPLICATIONS, 232.

20. Kinoshenko, D., Mashtalir, V., & Shlyakhov, V. (2007). A partition metric for clustering features analysis.

21. Егорова, Е. А., Киношенко, Д. К., Машталир, С. В., & Шляхов, Д. В. (2006). Метрическое сравнение результатов сементации изображений.

22. Mashtalir, V., Shcherbinin, K., Shlyakhov, V., & Yegorova, E. (2011). REGIONS OF SUFFICIENCY FOR METRICAL DATA RETRIEVAL. INFORMATION TECHNOLOGIES & KNOWLEDGE, 31.

23. Kinoshenko, D., Mashtalir, V., & Shlyakhov, V. (2007). A partition metric for clustering features analysis.

24. Bodyanskiy, Y., Kinoshenko, D., Mashtalir, S., & Mikhnova, O. (2012). On-line video segmentation using methods of fault detection in multidimensional time sequences. International Journal of Electronic Commerce Studies, 3(1), 1-20.

25. Brytik, V., Grebinnik, O., & Kobziev, V. (2016). Research the possibilities of different filters and their application to image recognition problems. ECONTECHMOD: An International Quarterly Journal on Economics of Technology and Modelling Processes, 5(4), 21-27.

26. Andreas M. Antonopoulos (2014). Mastering Bitcoin: Unlocking Digital Cryptocurrencies.

27. Don Tapscott, Alex Tapscott. (2016). Blockchain Revolution: How the Technology Behind Bitcoin is Changing Money, Business, and the World.

28. Chris Skinner. (2016). Value Web.

29. William Mougayar. (2016). The Business Blockchain: a Primer on the Promise, Practice and Application of the Next Internet Technology.

30. Roger Wattenhofer. (2016). The Science of the Blockchain.

31. Pavan Duggal. (2015). Blockchain Contracts and Cyberlaw.

32. Jacob William. (2016). Blockchain: The Simple Guide To Everything You Need To Know.