ДЕРЖАВНИЙ УНІВЕРСИТЕТ

ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Децентралізована система управління доступом до баз даних з використанням смарт контрактів і технології блокчейн»

на здобуття освітнього ступеня бакалавра
зі спеціальності 122 Комп'ютерні науки
освітньо-професійної програми Штучний інтелект

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Гліб ЛЕВЦУН
(підпис)

Виконав: здобувач вищої освіти гр.ШІД-41
Гліб ЛЕВЦУН

Керівник: Антон Шантир,
д.т.н. доцент

Рецензент:
науковий ступінь, вчене звання
_____
Ім'я, ПРІЗВИЩЕ

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ

ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Штучного інтелекту

Ступінь вищої освіти Бакалавр

Спеціальність 122 Комп'ютерні науки

Освітньо-професійна програма Штучний інтелект

ЗАТВЕРДЖУЮ

Завідувач кафедрою Штучного інтелекту
_____ Ольга ЗІНЧЕНКО
«_____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Левцуну Глібу Олександровичу

1. Тема кваліфікаційної роботи: Децентралізована система управління доступом до баз даних з використанням смарт контрактів і технології блокчейн

керівник кваліфікаційної роботи Антон ШАНТИР, д.т.н., доцент.
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» 02.2025р. № 36

2. Строк подання кваліфікаційної роботи «31» травня 2025р.

3. Вихідні дані до кваліфікаційної роботи: _____

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1.Огляд сучасних технологій для розробки

5. Перелік графічного матеріалу: *презентація*

6. Дата видачі завдання «27» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	27.02-05.11.25	Виконано
2	Вивчення сучасних підходів до децентралізованих систем, смарт-контрактів, Web3	05.11-12.11.25	Виконано
3	Проєктування архітектури децентралізованої системи управління доступом	13.11-19.11.25	Виконано
4	Розробка смарт-контракту для управління правами доступу в мережі Ethereum	20.11-25.11.25	Виконано
5	Реалізація бекенду (Flask + web3.py) та створення структури БД (SQLite)	27.11-03.12.25	Виконано
6	Інтеграція з блокчейн-мережею Ganache, тестування транзакцій, логування	04.12-10.12.25	Виконано
7	Оформлення роботи: вступ, висновки, реферат	11.12-20.12.25	Виконано
8	Розробка демонстраційних матеріалів	21.12-23.05.25	Виконано

Здобувач вищої освіти _____ Гліб ЛЕВЦУН
(підпис)

Керівник

кваліфікаційної роботи _____ Антон ШАНТИР, д.т.н, доцент
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: стор.54, рис.11, таблиць 1, джерел 20.

Мета роботи: Розробка децентралізованої системи управління доступом до баз даних на основі технології блокчейн з використанням смартконтрактів для забезпечення прозорості, безпеки та автономності доступу.

Об'єкт дослідження: Процес управління доступом до інформаційних ресурсів у розподілених обчислювальних системах.

Предмет дослідження: Механізми реалізації децентралізованого контролю доступу, а саме — використання смартконтрактів Ethereum як інструментів авторизації у блокчейн-орієнтованих IT-системах.

Короткий зміст роботи: У межах бакалаврської кваліфікаційної роботи розроблено та реалізовано децентралізовану вебсистему управління доступом до баз даних з використанням технології блокчейн та смарт-контрактів. Система побудована за принципами Web3-архітектури: замість централізованого сервера, права доступу зберігаються та перевіряються через смарт-контракт у мережі Ethereum (Ganache). Розробка здійснена з використанням Flask, web3.py, Solidity, SQLite та бібліотек Python. Передбачено створення криптогаманців для користувачів, авторизацію, можливість адміністратору надавати або відкликати доступ, а також перегляд дій у блокчейні. Усі дії логуються як транзакції, що забезпечує прозорість і безпеку. Проведено тестування системи, яке підтвердило коректність виконання транзакцій, стабільність смарт-контракту та ефективність моделі керування доступом у децентралізованому середовищі. Також реалізовано сучасний вебінтерфейс, адмінпанель і історію змін.

КЛЮЧОВІ СЛОВА: ДЕЦЕНТРАЛІЗАЦІЯ, БЛОКЧЕЙН, УПРАВЛІННЯ ДОСТУПОМ, СМАРТ-КОНТРАКТИ, ETHEREUM, GANACHE, FLASK, SOLIDITY, WEB3, КРИПТОГАМАНЕЦЬ, АВТОРИЗАЦІЯ.

ЗМІСТ

ЗМІСТ 7
ВСТУП 8
1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ У СФЕРІ УПРАВЛІННЯ ДОСТУПОМ 10
1.1 Проблематика централізованих систем керування доступом 10
1.2 Огляд сучасних рішень на базі блокчейн (AccessChain, Authereum, uPort) ... 14
1.3 Порівняння централізованих і децентралізованих систем доступу 21
Висновки до розділу 1 25
2 ПРОЄКТУВАННЯ ДЕЦЕНТРАЛІЗОВАНОЇ СИСТЕМИ 27
2.1 Архітектура системи 27
2.2 Розробка смарт-контракту для управління доступом 31
2.3 Структура бази даних і модель взаємодії користувача з системою 35
Висновок до розділу 2 41
3 РЕАЛІЗАЦІЯ ТА ОЦІНКА СИСТЕМИ 42
3.1 Вибір технологій та середовище розробки 42
3.2 Реалізація інтерфейсу користувача 48
3.3 Алгоритм перевірки доступу через блокчейн. Тестування, приклади 54
Висновок до розділу 3 58
ВИСНОВКИ 59
ПЕРЕЛІК ПОСИЛАНЬ 61
ДОДАТКИ 63
Додаток А 63

ВСТУП

У сучасних умовах стрімкого розвитку цифрових технологій питання захисту інформації, а зокрема — управління доступом до баз даних, набуває особливого значення. Традиційні централізовані моделі управління доступом залежать від єдиного серверного вузла, що робить їх вразливими до збоїв, атак, зловживань правами адміністраторів і втрати довіри з боку користувачів. У відповідь на ці виклики виникає необхідність у створенні децентралізованих систем, здатних забезпечити прозорий, надійний та автоматизований контроль доступу без участі посередницьких структур. Технології блокчейну та смарт-контрактів відкривають нові можливості для побудови таких систем. Блокчейн забезпечує незмінність і публічність даних, а смарт-контракти дозволяють автоматизувати логіку авторизації, змін доступу та перевірки прав. Таким чином, система управління доступом перетворюється з централізованої архітектури в розподілений механізм, де вся історія взаємодій фіксується в блокчейні.

Актуальність теми обумовлена необхідністю впровадження прозорих і стійких до втручання систем контролю доступу, особливо у сферах, де критичною є цілісність і конфіденційність даних — охорона здоров’я, державне управління, фінансовий сектор, корпоративні БД.

Метою даної кваліфікаційної роботи є розробка децентралізованої системи управління доступом до баз даних на основі смарт-контрактів і блокчейн-технологій.

Об’єктом дослідження є процес управління доступом до інформаційних ресурсів у розподіленому середовищі.

Предметом дослідження виступає смарт-контрактна система на блокчейні, яка дозволяє здійснювати надання та відкликання прав доступу до даних без централізованого контролю.

Завдання дослідження:

- проаналізувати сучасні централізовані та децентралізовані системи керування доступом;

- дослідити можливості використання смарт-контрактів та блокчейну для побудови безпечного механізму авторизації;
- спроєктувати архітектуру децентралізованої системи управління доступом;
- розробити смарт-контракт для керування правами доступу;
- реалізувати вебзастосунок на Flask із підтримкою реєстрації, авторизації та дій у блокчейні;
- протестувати систему та оцінити її прозорість, безпеку та відмовостійкість.

Методи дослідження:

- аналіз наукових публікацій та існуючих систем управління доступом;
- моделювання архітектури децентралізованого застосунку;
- розробка та тестування смарт-контрактів (Solidity + Ganache);
- використання фреймворку Flask, бібліотеки web3.py, системи управління БД SQLite.

Наукова новизна полягає у використанні смарт-контрактів як основного механізму контролю доступу, що забезпечує прозорість дій, відсутність єдиного центру керування та незмінність історії транзакцій.

Практичне значення розробки полягає у створенні реально функціонуючої децентралізованої системи управління доступом, яку можна адаптувати до будь-якого проєкту, що вимагає підвищеної інформаційної безпеки, прозорості та стійкості до атак.

1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ У СФЕРІ УПРАВЛІННЯ ДОСТУПОМ

1.1 Проблематика централізованих систем керування доступом

У сучасних інформаційних системах централізовані моделі управління доступом залишаються домінуючим підходом у більшості корпоративних, державних, освітніх та інфраструктурних середовищ. Основною характеристикою такої моделі є наявність центрального серверного вузла або контролера, який виконує функції автентифікації, авторизації, зберігання облікових даних користувачів, розподілу прав доступу, моніторингу подій безпеки, ведення логів, а також реалізації політик доступу. Усі запити користувачів проходять через цей вузол, що робить його критично важливим елементом системи. Попри свою поширеність і відносну простоту впровадження, централізовані моделі стикаються з низкою проблем, що загострюються в умовах зростання масштабів цифрових екосистем. По-перше, вони не здатні забезпечити прозорість прийняття рішень, оскільки будь-які дії адміністратора або внутрішні зміни в системі можуть відбуватись без належного аудиту або сповіщення. По-друге, централізовані системи створюють високі ризики для безпеки, зокрема уразливості до атак типу "єдина точка відмови", витоків даних, компрометації адміністративних облікових записів і привілейованого доступу. По-третє, масштабованість таких систем обмежується як технічними, так і організаційними чинниками: зі збільшенням кількості користувачів або сервісів складність управління стрімко зростає.

Суттєвою проблемою також є централізація довіри: у середовищах, де взаємодіє багато незалежних сторін (наприклад, у міжвідомчих платформах, партнерських екосистемах чи багатокомпонентних сервісах), покладання на єдиного адміністратора може стати джерелом конфліктів, недовіри та ризиків зловживань. Централізована модель не дозволяє гарантувати прозорість і підзвітність без зовнішнього контролю або незалежного аудиту. Враховуючи ці виклики, у сфері інформаційної безпеки та цифрового управління все більше уваги приділяється децентралізованим підходам. Серед них — використання технологій блокчейн та смарт-контрактів, що дозволяють створювати розподілені механізми

управління правами доступу, позбавлені багатьох недоліків традиційної централізованої архітектури. Ці підходи будуть детально розглянуті у наступних підрозділах.

Найбільш вагомими проблемами централізованих систем є:

Єдина точка відмови (Single Point of Failure) — це ситуація, коли вся система залежить від одного критичного компонента. У централізованих системах таким компонентом часто є головний сервер або вузол, який виконує всі функції автентифікації та управління доступом. У разі його збою — через технічну помилку, фізичне пошкодження, цілеспрямовану DDoS-атаку або зловмисні дії зсередини організації — система втрачає здатність обслуговувати користувачів, а отже, настає повне припинення її функціонування. Це створює серйозні ризики для безперервності бізнес-процесів, безпеки даних і доступності послуг, особливо в критично важливих галузях, таких як охорона здоров’я, банківський сектор чи державне управління.

Недостатня прозорість — у централізованих системах адміністратор має розширені повноваження щодо надання, зміни або відкликання прав доступу, а також перегляду та редагування журналів активності. Однак ці дії рідко супроводжуються належним аудитом або логуванням, яке не може бути змінене. Унаслідок цього користувачі не мають можливості перевірити, хто саме й коли вносив ті чи інші зміни. Така ситуація створює потенціал для зловживань, приховування помилок або навмисного видалення важливої інформації. Прозорість — один із ключових факторів довіри до системи безпеки, і її відсутність перешкоджає впровадженню ефективної політики контролю доступу, особливо у міжорганізаційних або публічних платформах.

Низька довіра — у середовищах, де важлива децентралізована взаємодія (наприклад, у міжорганізаційних структурах, державно-приватних партнерствах, міжнародних консорціумах або публічних сервісах), довіра до єдиного адміністратора може бути неприпустимою з етичних, юридичних і технічних причин. Централізована модель передбачає, що один суб’єкт (організація, адміністратор або обслуговуючий персонал) має повний контроль над системою

доступу, що створює нерівномірність у володінні владою, вразливість до внутрішніх загроз та можливість маніпуляцій. Для багатьох структур це створює бар'єр до впровадження спільної інфраструктури. Високий рівень довіри до адміністраторів потребує ретельного аудиту, юридичних гарантій, протоколів безпеки, що часто ускладнює процес прийняття рішень. У цьому контексті децентралізовані підходи набувають цінності, оскільки дозволяють розподілити повноваження та зменшити ризики монополізації управління.

Вразливість до компрометації — централізовані бази даних із правами доступу є особливо привабливими цілями для хакерських атак, соціальної інженерії, витоків через зловмисників усередині компанії або помилки адміністраторів. Наприклад, компрометація облікового запису адміністратора дозволяє змінити права доступу, приховано знищити дані, надати доступ третім сторонам або викрасти критично важливу інформацію без можливості її відновлення. Такі сценарії траплялися в історії корпоративної безпеки неодноразово — зокрема, відомі інциденти з порушенням доступу в Equifax, Facebook або Yahoo. Концентрація влади над правами доступу в одних руках перетворює централізовану систему на «смачну мішень», і її компрометація призводить до катастрофічних наслідків. Крім того, в таких системах важко забезпечити незмінність історії змін, що підриває довіру до цілісності всієї системи.

Обмежена масштабованість — централізовані рішення мають суттєві обмеження щодо гнучкості та здатності обслуговувати велику кількість користувачів, ресурсів і сервісів у динамічному середовищі. Із зростанням кількості підключених пристроїв, зростанням організаційної структури або обсягів даних центральний сервер або керуючий вузол починає відчувати перевантаження, що призводить до зниження продуктивності, уповільнення обробки запитів, збільшення затримок і навіть до збоїв. Крім того, адміністрування таких систем стає все складнішим і ресурсоємним: необхідно розширювати потужності серверів, додавати балансувальники навантаження, впроваджувати складніші схеми резервування та автоматизації.

У централізованих моделях також складніше впроваджувати динамічне масштабування, оскільки вони передбачають жорстко фіксовану ієрархію доступу та централізовану логіку автентифікації. Наприклад, додавання нових географічно віддалених підрозділів або партнерських організацій вимагає оновлення конфігурації центрального сервера, налаштування додаткових політик безпеки, синхронізації баз користувачів, облікових записів і ролей. Усе це потребує участі технічного персоналу, тестування сумісності компонентів та часто супроводжується зупинкою роботи частини сервісів на час розгортання. Крім того, такі оновлення несуть додаткове навантаження на центральний сервер, який може не бути готовим до обробки значно більшої кількості запитів або нових типів доступу. Ще однією проблемою є необхідність централізованого розгортання змін — тобто нові правила, користувачі чи зміни в логіці доступу мають проходити через адміністративні протоколи, що збільшує час реагування на нові вимоги бізнесу. У випадку, коли організація активно масштабується, впроваджує нові сервіси або взаємодіє з новими партнерами, така негнучкість стає істотним гальмом у розвитку.

Децентралізовані рішення допускають індивідуальну обробку запитів і реалізацію гнучких політик доступу на кожному вузлі без централізованого втручання, що істотно полегшує адаптацію до нових умов. У протилежність централізованій моделі, децентралізовані системи управління доступом базуються на принципах автономії та розподілу відповідальності. У таких системах кожен вузол (учасник мережі) має змогу самостійно обробляти запити, а перевірка прав доступу здійснюється через механізми консенсусу або перевірки транзакцій у блокчейні. Це дозволяє зменшити навантаження на окремі компоненти системи, підвищити її надійність та масштабованість. Замість централізованого контролера, що зберігає всі правила та облікові записи, децентралізовані рішення зберігають ці дані у відкритому, захищеному та незмінному середовищі — розподіленому реєстрі (блокчейні). Права доступу у таких системах можуть бути представлені у вигляді смарт-контрактів — спеціальних програм, які автоматично виконують наперед задану логіку на базі подій, запитів або умов. Це дає змогу виключити вплив

людського фактора на розподіл доступу, забезпечити його формальність, прозорість та контрольованість. Наприклад, замість того, щоб адміністратор вручну змінював рівень доступу користувача, смарт-контракт самостійно визначає наявність або відсутність прав, базуючись на умовах: наприклад, належність до групи, підтвердження криптопідписом, проходження автентифікації.

Децентралізована модель управління доступом забезпечує:

- гнучкість і стійкість до збоїв;
- прозорість усіх дій завдяки відкритості блокчейну;
- зменшення ризиків зловживань та помилок адміністраторів;
- повну історію змін, що не може бути змінена або стерта;
- природну масштабованість без потреби у централізованому апгрейді системи.

У результаті централізовані системи, попри їхню звичність і простоту реалізації, дедалі частіше визнаються недостатніми для задоволення вимог сучасної інформаційної безпеки. У світі, що стає дедалі більш взаємопов’язаним і розподіленим, децентралізовані технології на основі блокчейн і смарт-контрактів відкривають нові горизонти для побудови справді надійних, відкритих і масштабованих систем управління доступом. Подальші підрозділи детально висвітлять особливості таких підходів і прикладну реалізацію на практиці..

1.2 Огляд сучасних рішень на базі блокчейн (AccessChain, Authereum, uPort)

З поширенням блокчейн-технологій, які спочатку застосовувалися переважно у фінансовій сфері (наприклад, криптовалюти), виникла потреба використовувати ці технології в інших галузях — зокрема, в управлінні ідентичністю та правами доступу до цифрових ресурсів. Традиційні централізовані системи ідентифікації та авторизації мають безліч вразливих місць: від загроз витоку персональних даних до зловживань з боку адміністраторів. Натомість блокчейн надає змогу будувати нові моделі, які не потребують посередників,

працюють на основі криптографічного підтвердження та забезпечують повну незмінність і публічність даних.

Нові підходи, що реалізуються за допомогою блокчейну, базуються на таких принципах:

- децентралізація — відсутність єдиного центру прийняття рішень, що зменшує ризик маніпуляцій;
- самостійне управління ідентичністю (SSI) — користувачі самі створюють, зберігають і контролюють свої цифрові профілі;
- незмінність історії транзакцій — будь-яка дія записується у блокчейн і не може бути змінена заднім числом;
- криптографічна автентифікація — доступ здійснюється через перевірку цифрових підписів і криптографічних ключів;
- взаємодія з іншими децентралізованими сервісами (Web3) — можливість працювати у спільній інфраструктурі без дублювання акаунтів і реєстрацій.

Серед найбільш відомих реалізацій таких систем вирізняються AccessChain, Authereum і uPort. Вони ілюструють різні підходи до впровадження технологій децентралізованого контролю доступу, що вже застосовуються в проєктах різного масштабу — від приватних організацій до державних ініціатив. Ці системи стали фундаментом для формування нової епохи цифрової безпеки, в якій довіра базується не на централізованих інститутах, а на математично гарантованих протоколах та відкритому коді.

AccessChain(рис.1.1) — це децентралізований протокол, що забезпечує контроль доступу на основі блокчейну та використання смарт-контрактів. Його головна мета — надати гнучку, безпечну та прозору платформу для керування правами доступу до цифрових ресурсів у різних доменах: від вебсайтів до інтернету речей (IoT). Основна ідея реалізується через створення смарт-контрактів, які зберігають політики доступу у формі блокчейн-транзакцій. Кожна дія користувача, зокрема надання, редагування чи відкликання доступу, реєструється у розподіленому реєстрі, що унеможливлює її видалення або несанкціоновану зміну.

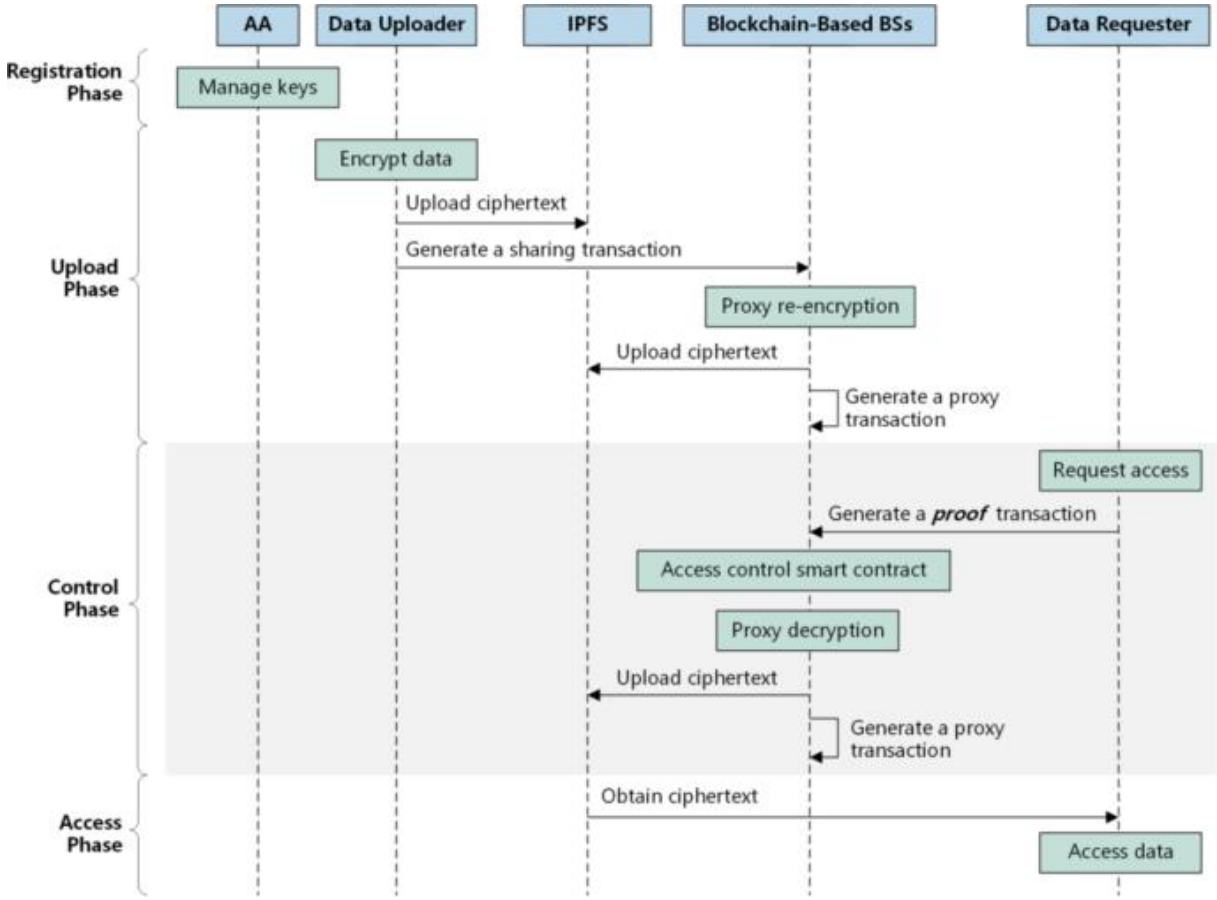


Рисунок 1.1 – Децентралізований протокол AccessChain

Особливістю AccessChain є підтримка складних і гнучких політик доступу, які можуть комбінувати численні параметри: часові рамки (напр., дозволено лише в робочий час), географічне розташування користувача, належність до певної організації або групи, багатофакторна автентифікація (наприклад, комбінація пароля, криптографічного підпису та біометричних даних), а також підтримка умовних правил, які ґрунтуються на логіці ("якщо-то"). Наприклад, доступ може бути дозволено лише у разі, якщо попередньо підтверджено транзакцію двома незалежними учасниками або якщо рівень довіри користувача досяг певного значення. Додатково реалізована підтримка динамічних ролей — ролі не жорстко закріплені за користувачами, а можуть змінюватися залежно від контексту: стану проєкту, результатів голосування або даних зовнішніх оракулів. Це дає змогу AccessChain забезпечити адаптивність до змін у реальному часі — зокрема, у складних корпоративних або міжорганізаційних системах, де структури доступу постійно змінюються.

Важливою перевагою системи є доступність RESTful API та SDK, що дозволяє інтегрувати AccessChain з уже існуючими системами керування доступом, базами даних, CRM-системами або корпоративними порталами. Це робить технологію зручною для впровадження навіть у середовищах, де вже використовуються традиційні підходи до контролю доступу, але потрібна більша прозорість, масштабованість і безпека. Завдяки цьому AccessChain може використовуватися в системах охорони здоров'я, фінансових установах, державних реєстрах, промислових IoT-рішеннях та освітніх платформах, де захист даних і контрольованість мають ключове значення.

Authereum (рис. 1.2) — це прогресивне рішення, яке поєднує функціональність Web2 із перевагами Web3, спрямоване на забезпечення зручного входу до децентралізованих застосунків (dApps) без втрати безпеки. На відміну від традиційних криптогаманців, Authereum дозволяє користувачам входити в систему за допомогою звичних облікових даних (електронна пошта, соціальні мережі), водночас управління обліковим записом здійснюється через смарт-контракти на Ethereum. Усі важливі дані — права доступу, підписи транзакцій, зміни атрибутів — обробляються без участі централізованих серверів, що знижує ризики витоку ключів. Перевагою Authereum є підтримка мультипідпису, що дозволяє підвищити рівень безпеки доступу до облікового запису шляхом вимоги підтвердження транзакції кількома ключами. Такий підхід активно застосовується в організаціях і DAO, де важливо мати колективний контроль над ресурсами. Крім того, система підтримує тимчасові токени доступу — це обмежені за часом ключі, які дають змогу делегувати доступ без постійного ризику компрометації основного облікового запису. Важливою функцією є також механізм відновлення доступу, який реалізується через довірені контакти, резервні ключі або авторизацію через пов'язані пристрої, що дозволяє уникнути повної втрати контролю над ідентичністю у разі втрати основного пристрою чи ключа.

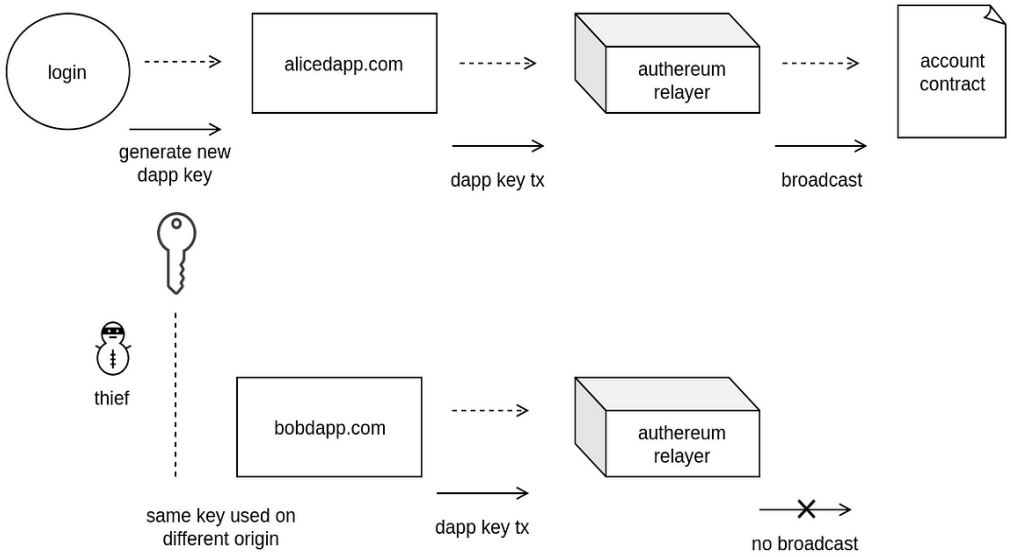


Рисунок 1.2 – Децентралізований протокол Authereum

Authereum також надає гнучке управління правами доступу — користувач самостійно визначає рівень доступу для застосунків, яким він дозволяє взаємодіяти зі своєю ідентичністю, з можливістю у будь-який момент відкликати дозвіл. Це дозволяє дотримуватись принципів мінімального доступу, конфіденційності та контролю за персональними даними. Система орієнтована на звичайного користувача та має інтуїтивно зрозумілий інтерфейс, що значно знижує поріг входу до децентралізованої екосистеми. На відміну від традиційних криптогаманців, які часто потребують складних дій для зберігання та захисту приватного ключа, Authereum делегує частину цих функцій до безпечного смарт-контракту, залишаючи контроль у руках власника.

На практиці Authereum активно використовується в таких сферах як DeFi (децентралізовані фінанси), NFT-платформи, голосування у DAO, інтелектуальні ринки (prediction markets), платформи гейміфікації та Web3-інструменти для ідентифікації. Це свідчить про її універсальність, адаптивність до різних сценаріїв і вагому роль у побудові безпечної та зручної цифрової ідентичності нового покоління.

uPort (рис. 1.3) — це система самостійного управління ідентичністю (Self-Sovereign Identity, SSI), яка надає користувачам повний контроль над своїми персональними даними без потреби довіряти централізованим органам. Вона

базується на принципі, згідно з яким лише користувач є власником і керівником своєї цифрової ідентичності, а не будь-яка централізована установа чи постачальник послуг. Ключова функція uPort полягає у створенні цифрових ідентичностей, які можна зберігати на мобільному пристрої користувача або у децентралізованому сховищі (наприклад, IPFS), а всі операції підтверджуються через криптографічні ключі.

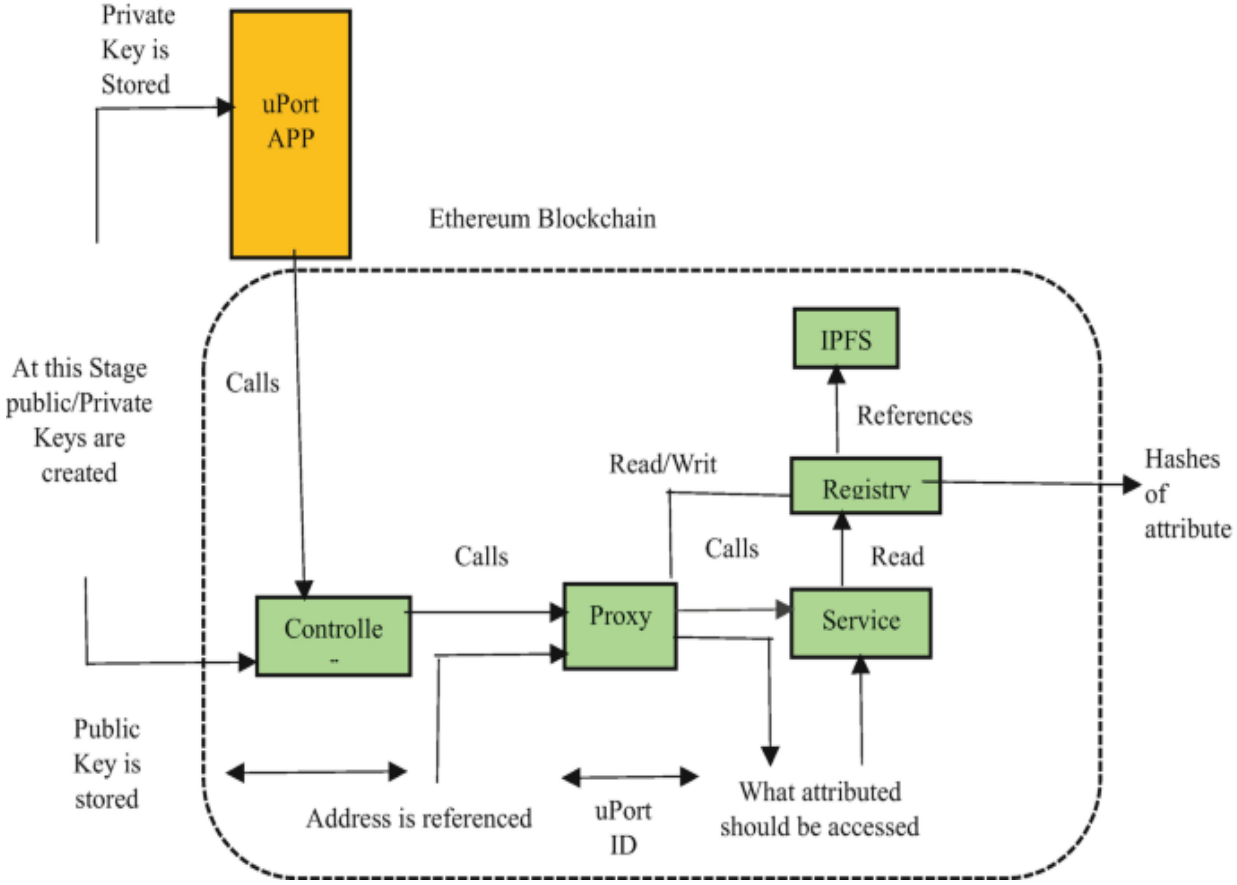


Рисунок 1.3 – Система самостійного управління ідентичністю uPort

Функціонал uPort включає в себе:

– реєстрацію ідентичності у блокчейні — кожен користувач отримує унікальний ідентифікатор, записаний у смарт-контракті;
– взаємну автентифікацію — користувачі та сервіси можуть перевіряти достовірність одне одного через підписи;
– керування атрибутами доступу — користувач має змогу зберігати, оновлювати й відкликати атрибути, що описують його ідентичність (наприклад, e-mail, статус резидента, роль у системі);

- цифрові підписи документів — підтвердження згоди чи дій у блокчейн-середовищі;
- інтеграцію з dApps — взаємодія з децентралізованими застосунками без потреби створювати нові облікові записи.

uPort активно використовується в проектах, пов'язаних із цифровою демократією, охороною здоров'я, освітою, голосуванням у DAO, а також у платформах, що вимагають верифікації особистості без зберігання даних на централізованих серверах. Завдяки підтримці стандартів DID (Decentralized Identifiers) і протоколу VC (Verifiable Credentials), uPort забезпечує високу сумісність з іншими рішеннями у сфері SSI, що відкриває можливості для міжплатформної інтеграції. uPort є одним із найповніших і технологічно досконалих рішень у сфері самостійного управління ідентичністю (Self-Sovereign Identity, SSI), яке поєднує у собі високий рівень безпеки, гнучкість, масштабованість та міжплатформну сумісність. Система дозволяє не лише створювати унікальні цифрові ідентичності, а й активно керувати атрибутами доступу, взаємодіяти з великою кількістю децентралізованих застосунків (dApps), а також брати участь у верифікаційних процесах без необхідності розкривати всю особисту інформацію. Це забезпечує збереження приватності, при цьому зберігаючи функціональність підтвердження особи.

Інтеграція uPort з DID (Decentralized Identifiers) і протоколом Verifiable Credentials дає можливість реалізовувати складні сценарії довіри — наприклад, верифікацію дипломів, сертифікатів, доступу до державних послуг, реєстрацій у виборчих процесах або надання тимчасового доступу до певних сервісів. Усі ці дії фіксуються у блокчейні, що забезпечує прозорість та неможливість підробки. Таким чином, uPort виступає не просто як засіб автентифікації, а як універсальна інфраструктура для формування цифрової довіри у Web3.

Рішення також значно знижує потребу в централізованих сховищах персональних даних, що зазвичай є найбільш вразливими елементами будь-якої системи. Впровадження uPort дозволяє компаніям і організаціям повністю уникати необхідності зберігати великі обсяги приватної інформації, передаючи контроль за

нею безпосередньо користувачам. Це підвищує рівень довіри до системи, зменшує ризики відповідальності за витік даних та дає змогу користувачам самостійно управляти своїм цифровим слідом. У перспективі такі системи можуть стати основою глобальної цифрової ідентичності людини, незалежно від платформи чи країни.

Усі зазначені системи доводять, що блокчейн може ефективно застосовуватись для контролю доступу, формування цифрових ідентичностей та зниження залежності від традиційної інфраструктури. Ці рішення суттєво знижують ризики компрометації, спрощують масштабування та відкривають шлях до побудови Web3-орієнтованих інформаційних систем, які будуть розглянуті у подальших розділах..

1.3 Порівняння централізованих і децентралізованих систем доступу

Порівняння централізованих і децентралізованих систем управління доступом дає змогу глибоко проаналізувати не лише технічні особливості реалізації, а й загальну концепцію управління доступом, логіку розподілу відповідальності, контроль прав, а також рівень довіри до інфраструктури. В умовах цифровізації суспільства, коли питання кібербезпеки та захисту персональних даних виходять на перший план, вибір архітектури системи управління доступом відіграє стратегічну роль. Централізовані системи історично були базовою моделлю реалізації контролю доступу у більшості IT-інфраструктур. Вони є простішими у розгортанні, мають зрозумілу ієрархію управління та централізовану точку адміністрування. Проте такі моделі мають суттєві обмеження: вся довіра концентрується у межах одного адміністратора або серверного вузла, що робить систему вразливою до зловживань, внутрішніх атак або помилок конфігурації. До того ж централізовані системи значно ускладнюють аудит, особливо у великих організаціях із динамічною структурою.

Децентралізовані системи, натомість, пропонують інший принцип — самостійне управління доступом через механізми консенсусу, відкритого коду та криптографічної перевірки. Вони не лише забезпечують незмінність і відкритість

історії змін, а й дозволяють кожному учаснику системи (користувачеві, вузлу чи сервісу) мати контроль над власною ідентичністю та правами доступу. У цьому підході довіра ґрунтується не на авторитеті адміністраторів, а на об’єктивних правилах, закладених у смарт-контрактах, і забезпечується прозорістю блокчейн-транзакцій.

Порівняння(табл. 1.1) є критично важливим для розуміння подальшого переходу до Web3-парадигми, в якій ідентичність, контроль доступу і транзакції здійснюються без участі посередників. Нижче наведено розгорнутий аналіз і табличне порівняння, що допоможуть оцінити переваги та недоліки кожного підходу в контексті сучасних викликів інформаційної безпеки.

1. Архітектура управління

- Централізовані системи мають єдиний сервер або контролер, що відповідає за всі операції з доступом.
- Децентралізовані рішення розподіляють контроль між багатьма учасниками, часто використовуючи блокчейн та смарт-контракти для зберігання і перевірки прав доступу.

2. Надійність і відмовостійкість

- У централізованих системах існує єдина точка відмови — у разі виходу з ладу центрального сервера система стає недоступною.
- У децентралізованих рішеннях відсутність одного центру робить систему стійкішою до технічних збоїв, атак і компрометацій.
- 3. Безпека
- Централізовані моделі вразливі до внутрішніх загроз, зловживань адміністраторів та атак на центральні сервери.
- У децентралізованих системах дії фіксуються у блокчейні, що забезпечує незмінність і можливість перевірки всіх транзакцій.

4. Прозорість та аудит

- У централізованих системах зміни часто залишаються невидимими для кінцевих користувачів.

– Блокчейн-архітектура надає повну історію змін, яка доступна для перегляду, що підвищує рівень довіри та контролю.

5. Масштабованість

– Централізовані системи вимагають потужнішого серверного обладнання та складніших механізмів синхронізації при розширенні.
– Децентралізовані платформи дозволяють органічне масштабування без центрального апгрейду.

6. Користувацький контроль

– Децентралізовані моделі надають користувачам більше автономії — вони можуть самостійно керувати своїми ідентичностями та правами.

7. Залежність від інфраструктури

– Централізовані системи тісно прив’язані до внутрішньої IT-інфраструктури організації.
– Децентралізовані рішення легко інтегруються у глобальні Web3-середовища та інші блокчейн-платформи.

Таблиця 1.1

Порівняння методів

Критерій	Централізовані системи	Децентралізовані системи
Архітектура	Єдиний контролер	Смарт-контракти, блокчейн
Надійність	Вразлива до збоїв	Висока відмовостійкість
Прозорість	Обмежена, підконтрольна адміну	Повна, відкритий лог
Масштабованість	Складна, ресурсозатратна	Органічна, гнучка
Контроль користувача	Мінімальний	Максимальний
Аудит і логування	Часто часткове	Непідроблюваний запис у блокчейні
Інтеграція з Web3	Відсутня або складна	Нативна підтримка

Порівняльний аналіз яскраво демонструє, що децентралізовані системи значною мірою переважають централізовані в багатьох ключових аспектах — від архітектурної гнучкості до прозорості обробки транзакцій. Завдяки використанню блокчейн-технологій, децентралізовані моделі забезпечують високий рівень надійності, оскільки усувають єдину точку відмови та дозволяють зберігати права доступу в незмінному вигляді. Прозора природа таких систем сприяє підвищенню довіри як з боку кінцевих користувачів, так і з боку організацій, які зацікавлені у прозорому аудиті подій. Масштабованість таких систем не обмежується фізичними ресурсами центрального сервера і дозволяє ефективно адаптувати систему до зростаючих потреб без критичних змін інфраструктури. Водночас децентралізована модель надає користувачеві більший ступінь контролю, оскільки управління ідентичністю та правами доступу реалізується на рівні смарт-контрактів, підписів і верифікаційних механізмів.

Впровадження децентралізованої архітектури є технічно складним процесом, який вимагає глибоких знань з кількох суміжних сфер. Насамперед, це розробка смарт-контрактів на мовах програмування, таких як Solidity, які повинні бути не лише функціональними, а й безпечними, оптимізованими та протестованими на відсутність вразливостей. Крім того, важливу роль відіграє інтеграція з блокчейн-мережами (наприклад, Ethereum або Binance Smart Chain) через спеціалізовані бібліотеки типу web3.js чи web3.py, які дозволяють взаємодіяти з децентралізованим реєстром з боку вебдодатків або серверних модулів. Вимагається глибоке розуміння криптографічного захисту, зокрема асиметричного шифрування, цифрових підписів, зберігання приватних ключів, генерації гаманців та захисту від атак типу «людина посередині» (MITM). Невід'ємною частиною є побудова безпечної інфраструктури — з використанням HTTPS, розмежуванням доступів, ізоляцією середовищ і протоколами захисту API.

Окремим викликом є UX-дизайн таких систем. Через технічну складність блокчейн-рішень (зокрема необхідність роботи з гаманцями, ключами, транзакціями) виникає потреба у створенні простого, інтуїтивно зрозумілого інтерфейсу, який приховує складність і не вимагає від користувача глибоких

технічних знань. Це передбачає глибоку співпрацю між розробниками, дизайнерами та експертами з взаємодії з користувачем.У подальших розділах буде всебічно розглянуто етапи проєктування архітектури децентралізованої системи, розробки її модулів, логіки обробки доступу, користувацького інтерфейсу, тестування безпеки та ефективності, а також оцінювання можливості її масштабування й практичного застосування в умовах реального використання.

Висновки до розділу 1

У першому розділі було здійснено комплексний аналіз існуючих рішень у сфері управління доступом, що дозволило глибоко розкрити ключові проблеми централізованих систем і обґрунтувати актуальність переходу до децентралізованих моделей на основі блокчейн-технологій.

Було встановлено, що централізовані системи, попри їхню традиційну популярність, мають низку критичних недоліків: наявність єдиної точки відмови, обмежена масштабованість, низька прозорість, вразливість до внутрішніх зловживань і недостатній рівень довіри з боку користувачів. Усе це створює ризики для організацій і обмежує можливості безпечної взаємодії в умовах сучасної цифрової економіки.

Децентралізовані системи, навпаки, демонструють низку переваг: вони дозволяють кожному користувачеві мати повний контроль над своєю цифровою ідентичністю, забезпечують прозорість дій через блокчейн, унеможливлюють несанкціоновані зміни, значно спрощують аудит і покращують масштабованість. Такі рішення особливо актуальні в середовищах, де важлива довіра між сторонами, прозорість дій та автоматизація процесів авторизації.

Огляд сучасних платформ — AccessChain, Authereum, uPort — продемонстрував різноманітність підходів до реалізації децентралізованого управління доступом. Кожна з них має унікальні особливості, функціональність та сценарії застосування, що підтверджує гнучкість і життєздатність таких рішень у різних галузях — від фінансів до освіти.

У підрозділі 1.3 проведено порівняльний аналіз централізованих і децентралізованих моделей управління доступом, який наочно продемонстрував перевагу останніх у більшості аспектів: безпека, прозорість, масштабованість, незалежність від адміністраторів та відповідність філософії Web3.

Розділ 1 не лише сформував теоретичну базу для подальшого проєктування власної системи, а й заклав концептуальні засади для вибору архітектурного підходу, який найкраще відповідає сучасним вимогам безпеки, гнучкості та довіри. Наступний розділ буде присвячено практичному проєктуванню децентралізованої системи управління доступом із використанням технологій смарт-контрактів і блокчейн.

2 ПРОЄКТУВАННЯ ДЕЦЕНТРАЛІЗОВАНОЇ СИСТЕМИ

2.1 Архітектура системи

Архітектура розробленої децентралізованої системи управління доступом побудована відповідно до принципів Web3 — концепції, що передбачає повну відмову від централізованого зберігання критичних даних на користь розподілених, прозорих і стійких до втручань механізмів. Замість класичного серверного підходу, де вся логіка доступу концентрується в єдиній точці, у запропонованій архітектурі управління правами реалізується через смартконтракт, розміщений у блокчейн-мережі. Усі транзакції, пов’язані з авторизацією, наданням чи відкликанням доступу, записуються до розподіленого реєстру, де не можуть бути змінені або стерті. Завдяки такому підходу досягається низка важливих переваг: по-перше, забезпечується висока прозорість — кожна транзакція або дія користувача фіксується у незмінному реєстрі блокчейну, що унеможливлює підробку, видалення або несанкціоновану модифікацію. Це особливо важливо у випадках конфліктних ситуацій або необхідності аудиту дій. По-друге, досягається стійкість до зовнішніх загроз — повна відсутність централізованого серверного вузла унеможливлює реалізацію типових атак, таких як DDoS, Brute Force, SQL-ін’єкції чи крадіжка облікових даних. Сама авторизація реалізується без передачі або зберігання паролів, що додатково зменшує поверхню для атак. По-третє, реалізується максимальний контроль за логікою доступу — замість ручного адміністрування або скриптів на сервері, вся логіка управління правами користувачів закладена у смартконтракт, який є публічним, незмінним після деплою та функціонує за чітко визначеними правилами, незалежними від зовнішніх змін у коді системи. Це унеможливлює навмисну зміну логіки доступу з боку адміністраторів або сторонніх осіб без відповідної транзакції та підпису власника контракту.

У результаті така архітектура гарантує гнучкість, надійність і дотримання принципів децентралізації, що надзвичайно важливо для забезпечення незалежного, прозорого й стійкого до впливу середовища. У випадках, коли довіра

до централізованого оператора неможлива, наприклад у відкритих публічних системах, децентралізованих автономних організаціях (DAO), міжкорпоративних інфраструктурах або додатках, орієнтованих на користувача, відсутність центрального контролера зводить до мінімуму ризики маніпуляцій, цензури та несанкціонованого втручання. Крім того, децентралізована модель значно покращує стійкість до відмов: навіть у разі недоступності окремих компонентів система продовжує функціонувати, спираючись на розподілену мережу. Такий підхід також дозволяє ефективно масштабувати рішення та інтегрувати його в інші Web3-сумісні інфраструктури без необхідності повної перебудови логіки.

Основу архітектури становить трикомпонентна структура:

- Клієнтська частина (Frontend): реалізована на HTML/CSS та JavaScript, надає користувачу інтерфейс для реєстрації, авторизації, генерації гаманця, перегляду прав доступу та взаємодії з системою.
- Серверна частина (Backend): реалізована на Flask (Python), виконує проміжну логіку між користувачем і блокчейн-середовищем, обробляє запити до смартконтракту, управляє базою даних і взаємодіє з Web3 через бібліотеку web3.py.
- Смартконтракт: написаний мовою Solidity і задеплоєний у локальну мережу Ganache. Контракт зберігає всю інформацію про користувачів, їхні права доступу та логіку надання/відкликання доступу.

Компоненти архітектури:

- Сторінка реєстрації та генерації гаманця — користувач вводить свої дані, після чого система генерує унікальний криптографічний гаманець (ключ + адреса) й додає його до системи.
- Авторизація та вхід — за допомогою приватного ключа користувач може підписати повідомлення і підтвердити свою особу без передачі пароля. Це дозволяє уникнути зберігання паролів на сервері.
- Адмінпанель — окрема частина інтерфейсу для адміністратора, який має змогу керувати правами доступу (надання, відкликання, перевірка логів) через смартконтракт.

– Інтеграція з блокчейном (Ganache) — система працює з приватною мережею Ethereum (локальний вузол), що дозволяє тестувати логіку смартконтракту без витрат на газ.

Ключові переваги архітектури:

– Відсутність єдиного центру прийняття рішень.
– Прозорість операцій завдяки записам у блокчейні.
– Захист від несанкціонованого доступу через криптографію.
– Можливість легкої інтеграції з іншими Web3-платформами.
– Розширюваність: система може бути масштабована для роботи з різними типами ресурсів.

Архітектурна схема (рис. 2.1), сформована на основі презентаційних матеріалів, детально відображає ключові компоненти системи та характер їхньої взаємодії. Вона ілюструє багаторівневу взаємодію між користувачем, клієнтським інтерфейсом, серверною логікою, смартконтрактом і блокчейн-мережею. Усі запити проходять чітко визначений шлях: користувач ініціює дію в браузері, дані передаються через frontend до Flask-сервера, який через бібліотеку Web3 виконує взаємодію зі смартконтрактом, а результати фіксуються у блокчейні. На схемі видно, що вся система працює у повністю децентралізованому середовищі без необхідності зберігання критичних даних на стороні сервера. Права доступу, реєстрація користувачів, авторизація, перегляд історії доступу — усе це реалізовано на рівні смартконтракту, розгорнутого в EVM-сумісній блокчейн-мережі. Саме смартконтракт виступає центральною точкою логіки доступу, яка зафіксована у вигляді незмінного коду й не підлягає зміні без явного підтвердження через транзакцію.

Авторизація користувача реалізується з використанням популярного розширення MetaMask або іншого Web3-сумісного криптогаманця.

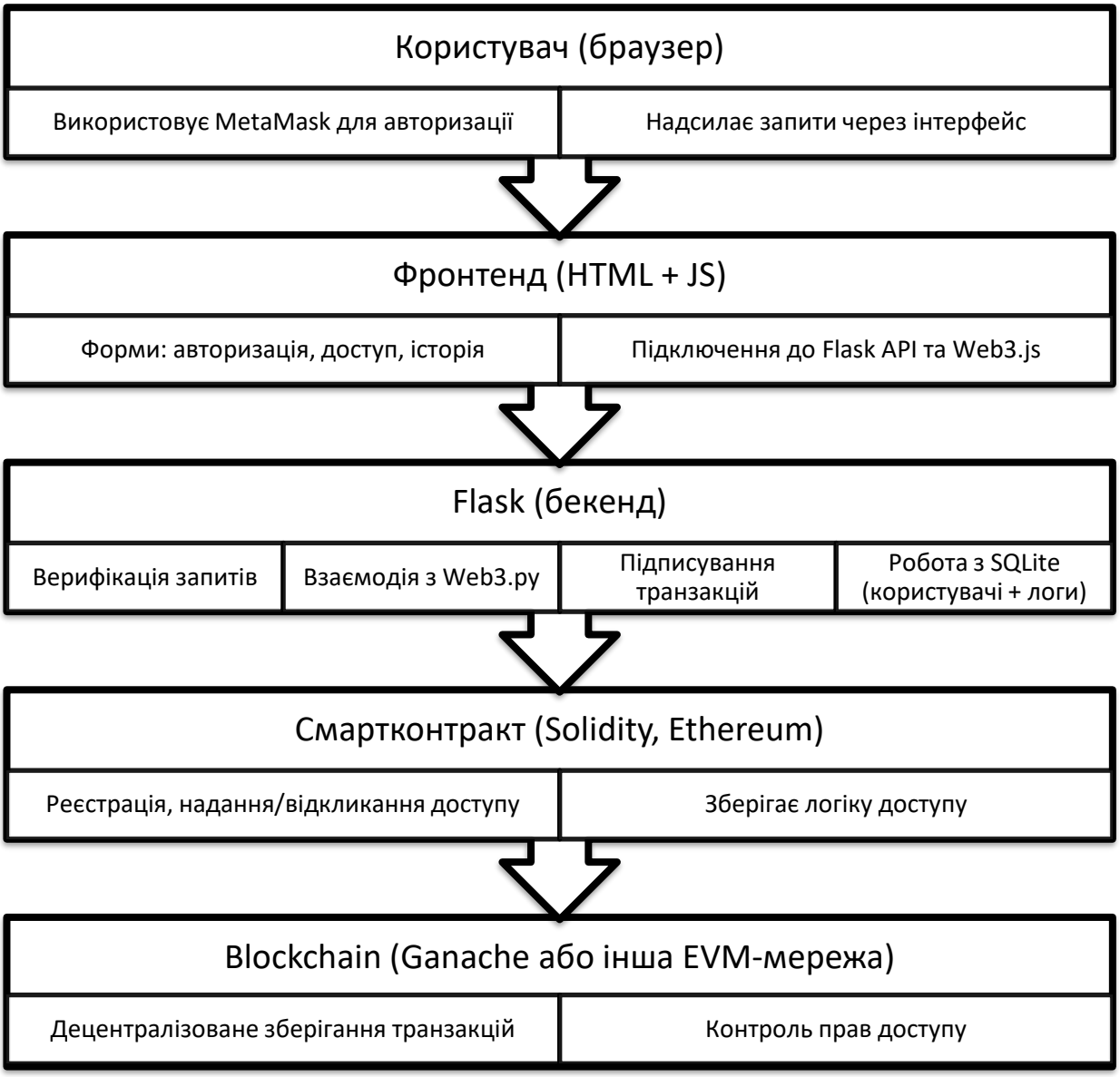


Рисунок 2.1 — Архітектура децентралізованого застосунку

Усі транзакції — включно з діями надання доступу, відкликання, перевірки ролей тощо — підписуються безпосередньо на клієнтському боці, що означає повну відсутність необхідності передавати або зберігати паролі, ключі чи сесійну інформацію на бекенді. Таким чином, бекенд-сервер Flask виступає лише проміжною ланкою, що перевіряє коректність запитів, взаємодіє з Web3-провайдером та забезпечує логічну маршрутизацію без обробки критичних даних. Подібна архітектура гарантує високий рівень безпеки: навіть у випадку компрометації сервера або клієнта зловмисник не може змінити права доступу без володіння приватним ключем користувача. Водночас всі події фіксуються у блокчейні, створюючи прозорий лог дій, який неможливо підробити або видалити.

Масштабованість досягається за рахунок відсутності централізованого вузла, що дозволяє додавати нових користувачів і ресурси без зміни логіки системи. Модель ілюструє справжню суть децентралізації — максимальний контроль користувача над власними даними, прозоре управління доступом і повна незалежність від людського фактору адміністрування. Це і є фундаментальна відмінність і перевага над класичними централізованими системами.

У наступному підрозділі буде детально розглянуто реалізацію та логіку смартконтракту, що виконує роль ядра системи управління доступом, з прикладами його ключових функцій і структур даних..

2.2 Розробка смарт-контракту для управління доступом

Основою децентралізованої системи управління доступом виступає смартконтракт, який виконує центральну роль у забезпеченні прозорого, безпечного та незмінного механізму керування і перевірки доступу до ресурсів. Смартконтракт реалізує логіку аутентифікації (визначення автентичності користувача), авторизації (надання або обмеження прав) та контролю ролей, що забезпечує гнучку модель доступу без участі централізованих серверів. Це дозволяє уникнути типових проблем централізованих систем, таких як злом серверу, підміна даних або технічні збої. Для реалізації смартконтракту було обрано мову програмування Solidity — основну мову для розробки смартконтрактів у мережі Ethereum. Solidity є стандартом де-факто для екосистеми Ethereum Virtual Machine (EVM) завдяки своїй підтримці складних логічних структур, івентів, модифікаторів доступу, вбудованого контролю транзакцій та можливості інтеграції з фронтенд-застосунками через ABI (Application Binary Interface). Крім того, мова має широку спільноту, багату базу знань і сумісність з багатьма інструментами розробки, такими як Truffle, Hardhat, Remix IDE та інші. Це забезпечує швидку розробку, перевірку і деплой контрактів у тестових і основних мережах.

Смартконтракт, як частина цієї системи, гарантує прозору фіксацію кожної операції в блокчейні — будь то призначення ролі, її відкликання чи перевірка поточного статусу користувача. Всі дії фіксуються незмінно, що забезпечує

високий рівень довіри між учасниками системи, а також дозволяє проводити незалежний аудит у разі суперечок або інцидентів.

Нижче наведено фрагмент реалізованого смартконтракту, що використовується у проєкті:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract AccessManager {
    address public admin;
    enum Role { None, User, Admin }
    mapping(address => Role) public roles;
    event RoleGranted(address indexed user, Role role);
    event RoleRevoked(address indexed user);
    constructor() {
        admin = msg.sender;
        roles[admin] = Role.Admin;
    }
    modifier onlyAdmin() {
        require(roles[msg.sender] == Role.Admin, "Not
authorized");
        _;
    }
    function grantRole(address user, Role role) public onlyAdmin
{
        roles[user] = role;
        emit RoleGranted(user, role);
    }
    function revokeRole(address user) public onlyAdmin {
        roles[user] = Role.None;
        emit RoleRevoked(user);
    }
    function getRole(address user) public view returns (Role) {
        return roles[user];
    }
}
```

Опис логіки смартконтракту:

Адміністратор системи — адреса, яка ініціює розгортання смартконтракту (тобто msg.sender у конструкторі), автоматично наділяється найвищим рівнем повноважень — роллю Admin. Це забезпечує початковий контроль над системою та гарантує, що лише довірена особа зможе делегувати повноваження або змінювати політики доступу. У майбутньому можливо реалізувати передачу цієї ролі іншому користувачу або навіть використати механізм мультипідпису для критичних операцій.

Система ролей реалізована у вигляді перерахування (enum), що описує набір дозволених статусів користувачів: None — відсутність доступу, User — звичайний користувач із базовими правами, Admin — адміністратор, який має повний контроль. Такий підхід дозволяє чітко структурувати доступ, розширювати перелік ролей (наприклад, Moderator, Auditor) та легко масштабувати систему, не змінюючи фундаментальну логіку.

grantRole() — функція, яку може викликати лише адміністратор, надає визначену роль іншій адресі. Вона оновлює mapping з ролями, встановлюючи нове значення Role, відповідне до обраного рівня доступу. Окрім цього, функція генерує подію (event) RoleGranted, яка фіксується в блокчейні, що забезпечує прозорість змін у системі. У разі потреби логіку можна адаптувати для додаткових перевірок, наприклад, заборони повторного призначення тієї ж ролі або логування дій в окремий аудит-журнал.

revokeRole() — функція, що відкликає роль у користувача, змінюючи її на Role.None. Вона також генерує подію RoleRevoked, що дозволяє зафіксувати факт відкликання доступу у незмінному реєстрі транзакцій. Цей механізм є ключовим для відновлення контролю над доступами у разі компрометації облікового запису або змін в організаційній структурі. У більш розвинених сценаріях до revokeRole() можна додати логіку автоматичного повідомлення або часові обмеження на повторне надання ролі.

getRole() — функція читання, яка дозволяє дізнатись роль будь-якого користувача, просто вказавши його адресу. Ця функція є відкритою (public view) і

не змінює стан блокчейну, тому її можна викликати без витрат газу з локального вузла або через інтерфейс користувача. Вона повертає значення перерахування Role, яке вказує, яку саме роль має вказана адреса на поточний момент: None, User чи Admin. Така функціональність дозволяє реалізовувати контроль доступу на рівні інтерфейсу (наприклад, відображати або ховати кнопки керування), а також забезпечує прозорість і відстежуваність статусів користувачів. У майбутньому можливе розширення цієї функції для повернення історії змін ролей або інтеграції з іншими ідентифікаційними механізмами (наприклад, DID або NFT-токенами ролей).

Переваги такого підходу:

1. Прозорість: усі дії з доступом зберігаються у блокчейні.
2. Безпека: лише користувач з правами Admin може змінювати доступ.
3. Відсутність централізованого бекенду: контракт діє як незалежний «арбітр» прав доступу.
4. Простота перевірки: будь-який учасник системи може перевірити поточну роль іншого користувача без доступу до серверів чи бази даних.

У подальшій розробці можливе істотне розширення функціональності смартконтракту. Зокрема, варто реалізувати підтримку механізму мультипідпису (multisig), що забезпечить підвищений рівень безпеки при виконанні критичних операцій, таких як призначення нових адміністраторів або відкликання ролей. Механізм мультипідпису дозволяє вимагати підтвердження дії від кількох авторизованих осіб, знижуючи ризик зловживань або компрометації єдиного приватного ключа. Доцільним є впровадження часових обмежень доступу (time-based access control), що дозволить автоматично відкликати роль після завершення визначеного періоду або активувати її лише в певні часові вікна. Це особливо актуально для тимчасових підрядників або учасників із обмеженим часом участі в системі. Контракт може бути розширений для інтеграції з децентралізованими ідентифікаторами (DID), що дозволить уніфікувати аутентифікацію та авторизацію в межах глобальної екосистеми Web3. DID забезпечують перевірку особи без

централізованих реєстрів і можуть поєднуватись з NFT або іншими токенізованими атрибутами.

Завдяки своїй компактності, чіткій структурі, модульності та використанню найкращих практик програмування на Solidity, представлений смартконтракт є ключовим компонентом запропонованої децентралізованої системи управління доступом. Його архітектура дозволяє не лише ефективно розподіляти повноваження між учасниками, але й гарантує, що усі логічні умови і права доступу зберігаються в незмінному вигляді в блокчейні. Відкритість коду та фіксація подій у публічному реєстрі забезпечують високий рівень прозорості, що дозволяє аудиторам та зацікавленим сторонам проводити незалежну перевірку. Водночас жодна зміна до смартконтракту не може бути внесена без нової публічної версії контракту, що підвищує захищеність системи навіть у випадку компрометації адміністратора або сервера.

2.3 Структура бази даних і модель взаємодії користувача з системою

Незважаючи на те, що основна логіка управління доступом реалізована у вигляді смартконтракту в блокчейні, система також включає локальну реляційну базу даних (у даному випадку — SQLite), яка відіграє допоміжну роль. Вона використовується для зберігання службової інформації, що не вимагає високого рівня захисту або незмінності, але є важливою для функціонального функціонування інтерфейсу користувача та адміністратора. У базі даних зберігається інформація про користувачів (адреси гаманців, ролі, час створення запису), а також журнал дій (access_log), який містить усі операції, пов'язані зі спробами доступу, входом у систему, наданням або відкликанням прав. Це дозволяє швидко отримувати звітність, створювати аналітику, вести аудит та забезпечити локальну обробку запитів, які не потребують постійного звернення до блокчейну.

Таке комбіноване рішення дозволяє з одного боку зберігати головну логіку прав у смартконтракті (де вона незмінна і захищена), а з іншого — забезпечити оптимізацію продуктивності, зменшити навантаження на блокчейн і підвищити

зручність використання системи для кінцевих користувачів. Така гібридна модель є типовою практикою у Web3-інфраструктурах, де частина даних фіксується в блокчейні для забезпечення довіри, а частина — в БД для швидкої та гнучкої обробки.

Таблиця users

Таблиця users зберігає базові відомості про користувачів, які взаємодіють із системою:

- id (INTEGER) — унікальний ідентифікатор кожного користувача, первинний ключ;
- wallet_address (TEXT) — адреса криптогаманця, з яким асоційований користувач. Вона використовується для авторизації через MetaMask або інші Web3-гаманці;
- role (TEXT) — роль користувача у системі (admin, user, тощо), яка може дублюватися з інформацією у смартконтракті задля зручності локальної перевірки;
- created_at (DATETIME) — час реєстрації користувача в системі.

Ця таблиця є допоміжною: вона не бере участі в безпосередньому прийнятті рішень щодо доступу, оскільки вся логіка авторизації зосереджена у смартконтракті. Проте вона є критично важливою для забезпечення повноцінного функціонування клієнтської частини, оскільки зберігає метадані, пов’язані з користувачами. До таких метаданих належать: час створення запису, адреса гаманця, поточна роль, а також інші атрибути, які можуть використовуватись для відображення даних в інтерфейсі, фільтрації логів, створення адміністративних панелей або підготовки звітів. Крім того, таблиця users слугує зручною точкою зв’язку для побудови внутрішніх запитів та для локальної перевірки присутності користувача в системі до звернення до блокчейну, що значно покращує продуктивність при великій кількості сесій.

Таблиця access_log

Таблиця access_log фіксує всі ключові дії користувачів, пов’язані з доступом:

- id (INTEGER) — унікальний ідентифікатор запису;
- user_id (INTEGER) — зовнішній ключ, що посилається на users.id;
- action (TEXT) — тип дії: вхід, надання доступу, відкликання, перевірка прав тощо;
- timestamp (DATETIME) — точна дата і час виконання дії.

Завдяки цьому журналу адміністратори можуть отримувати детальний аналітичний огляд усіх змін і подій, пов’язаних із правами доступу: хто, коли і яку саме дію виконав. Це включає як надання та відкликання прав, так і спроби несанкціонованого доступу, що дозволяє оперативно виявляти потенційні загрози або аномалії. Записи у access_log дають змогу формувати фільтровану статистику по кожному користувачу, проводити щомісячні аудити без необхідності опрацьовувати всі транзакції в блокчейні, а також інтегрувати ці дані в зовнішні інструменти звітності. Усі ці дії фактично дублюють події, що фіксуються у смартконтракті, але зберігаються локально для забезпечення високої швидкодії, швидкого пошуку та незалежного логічного аналізу в межах внутрішнього середовища системи.

Модель взаємодії користувача з системою(рис.2.2)

Сценарій взаємодії користувача з децентралізованою системою доступу побудований на поетапній логіці, що поєднує зручність користування з високим рівнем безпеки. Особливістю цієї моделі є тісна інтеграція з криптографічними методами автентифікації, що забезпечують унікальний контроль і прозорість усіх дій у системі без використання паролів чи централізованих серверів.

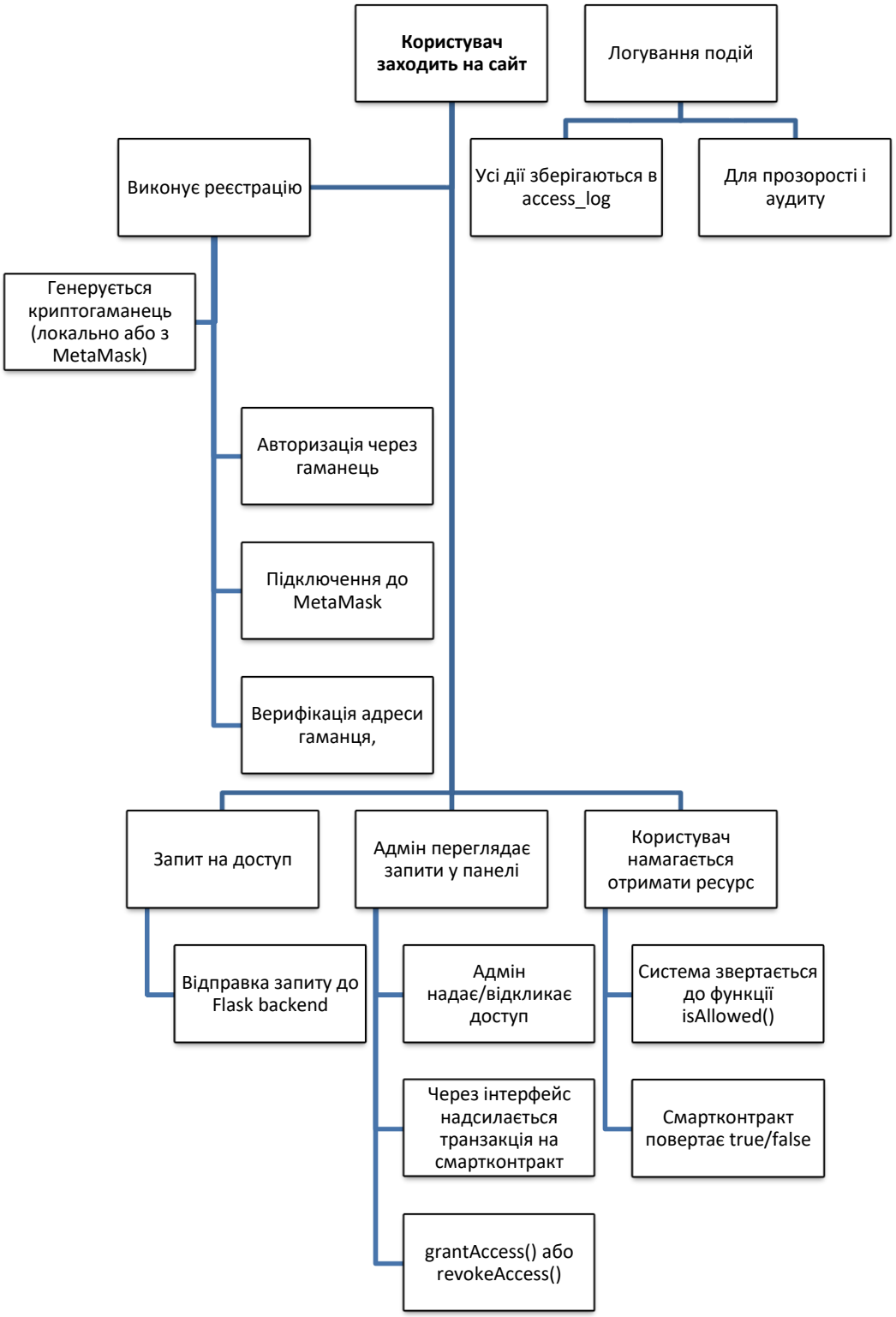


Рисунок 2.2 — Сценарій взаємодії користувача з системою

Кожен етап взаємодії побудований на логіці мінімального довірчого середовища, де будь-яке рішення ухвалюється лише після підтвердження приватним ключем користувача або смартконтрактом у блокчейні. Процес починається з входу користувача на сайт, де ініціалізується його сесія та

перевіряється наявність криптогаманця. Якщо це перший візит — система пропонує пройти реєстрацію, яка передбачає генерацію криптографічної пари ключів або імпорт уже існуючого гаманця. При цьому, жодна конфіденційна інформація не зберігається на сервері — лише публічна адреса гаманця.

Подальші дії, такі як підтвердження володіння гаманцем, перевірка ролі через смартконтракт, ініціювання запиту на доступ або обробка цього запиту — виконуються строго відповідно до ролей і дозволів, закладених у логіці контракту. Адміністратор має змогу керувати правами доступу через інтерфейс, який надсилає відповідні транзакції (наприклад, grantAccess() або revokeAccess()), при цьому всі ці дії реєструються не лише у блокчейні, а й у локальному журналі (access_log). Кожна дія — від перегляду сторінки до надання доступу — супроводжується криптографічною перевіркою та створенням події, яка може бути проаналізована згодом для аудиту чи звітності. Завершальним кроком кожної взаємодії є виклик функції isAllowed() смартконтракту, що остаточно визначає, чи має користувач право на доступ до певного ресурсу.

Ця модель не тільки забезпечує відповідність сучасним вимогам до інформаційної безпеки, але й є масштабованою та придатною для використання в системах, де високий рівень довіри є критичним — зокрема в урядових системах, DAO, корпоративних мережах або освітніх платформах. Нижче показано план схеми:

1. Користувач заходить на сайт. Ініціалізується сесія, і система готується до перевірки гаманця. Всі дії користувача з цього моменту логуються в базу даних (access_log).
2. Реєстрація. Якщо користувач вперше — він проходить реєстрацію, під час якої створюється або імпортується криптогаманець (наприклад, через MetaMask). Адреса гаманця додається до таблиці users, роль встановлюється за замовчуванням (user).
3. Авторизація. Після підключення до MetaMask користувач підтверджує володіння гаманцем шляхом підписання повідомлення. Жоден пароль не зберігається.

4. Верифікація гаманця. Система перевіряє, чи є адреса у таблиці users, та порівнює її з правами, вказаними в смартконтракті.
5. Запит на доступ. Якщо користувач намагається отримати доступ до ресурсу — він формує запит через інтерфейс.
6. Запит обробляється. Flask-бекенд передає дані у функції смартконтракту через Web3. Якщо користувач — адміністратор, він може надавати/відкликати доступ іншим через grantAccess() або revokeAccess().
7. Фіксація подій. Кожна дія з доступом викликає відповідну подію у смартконтракті (наприклад, RoleGranted), а також дублюється у таблиці access_log.
8. Система викликає isAllowed(). У разі спроби доступу до захищеного ресурсу, система звертається до функції смартконтракту isAllowed() — вона повертає true або false залежно від ролі.

Модель взаємодії користувача з системою демонструє збалансовану інтеграцію технологій Web3 з класичними підходами до побудови інформаційних систем. Усі ключові етапи авторизації, перевірки прав і доступу реалізовані із застосуванням криптографічного підпису та підтверджуються транзакціями в блокчейні, що виключає будь-яку можливість підміни, несанкціонованих змін або обману зі сторони клієнта чи сервера. Завдяки цьому підхід забезпечує найвищий рівень прозорості та аудиту, оскільки кожна дія залишає цифровий слід у публічному реєстрі. Для досягнення високої швидкодії та зручності обслуговування кінцевих користувачів, система використовує локальну реляційну базу даних, яка виступає в ролі кешуючого шару. Вона не містить критичних рішень щодо прав доступу, проте дозволяє обробляти велику кількість однотипних запитів без надмірного навантаження на блокчейн. Це особливо корисно в умовах масштабованості, коли кількість користувачів або транзакцій може досягати тисяч за хвилину.

Система поєднує сильні сторони обох підходів: незмінність і безпечність децентралізованих смартконтрактів — для рішень щодо доступу, і гнучкість, продуктивність та зручність локальної БД — для обслуговування користувацького

інтерфейсу та аналітики. Це забезпечує не лише надійність, а й комфорт роботи з системою, що робить її придатною як для приватного корпоративного застосування, так і для масштабних публічних платформ.

Висновок до розділу 2

У другому розділі було здійснено проектування децентралізованої системи управління доступом до баз даних з використанням технології блокчейн та смарт-контрактів. Описано загальну архітектуру системи, що поєднує безпечну логіку доступу на основі Ethereum-контрактів із швидкодійною локальною базою даних для оптимізації взаємодії з користувачем.

У процесі проектування розроблено власний смарт-контракт, що забезпечує перевірку прав доступу, додавання і відкликання доступу, а також фіксацію всіх транзакцій у блокчейні. Проведено детальний аналіз структури бази даних, визначено функціональні ролі таблиць users та access_log, які слугують для зберігання службової інформації, полегшують візуалізацію даних та не дублюють функціональність блокчейну.

Розроблено модель взаємодії користувача з системою, в якій враховано особливості підпису транзакцій, підтвердження доступу та безпечної передачі ідентифікаторів. Така модель забезпечує максимальний рівень прозорості, гнучкості та стійкості до несанкціонованого втручання, що робить систему придатною до використання у сферах, де довіра та контроль є критично важливими.

Завдяки поєднанню децентралізованої логіки з локальними механізмами кешування, спроєктована система є надійною, масштабованою та ефективною для практичного застосування в умовах реального навантаження.

3 РЕАЛІЗАЦІЯ ТА ОЦІНКА СИСТЕМИ

3.1 Вибір технологій та середовище розробки

У процесі реалізації децентралізованої системи управління доступом було проведено ретельний аналіз доступних технологічних стеків, зокрема в контексті безпеки, надійності, підтримки Web3-функціональності, прозорості транзакцій та здатності до масштабування. Основною метою було забезпечити не лише функціональність, а й відповідність філософії децентралізації: відмова від централізованих серверів, використання перевірених блокчейн-інструментів і відкритих протоколів, автоматизація прав доступу через смартконтракти та підвищена довіра користувачів через немодифіковану історію транзакцій. Для досягнення цієї мети було обрано сучасні інструменти, що дозволили ефективно реалізувати кожен рівень архітектури системи — від інтерфейсу до блокчейн-ядра. Особливу увагу було приділено таким аспектам, як захист приватних ключів, перевірка прав доступу без централізованого контролера, фіксація кожної дії в блокчейні, підтримка багаторольової моделі та зручність розробки. Усі технології було протестовано на сумісність, стабільність у локальному середовищі (Ganache) та здатність до інтеграції між собою через стандартизовані API.

Було сформовано цілісну екосистему, що включає мову Solidity для розробки контрактів, інструменти Ganache та web3.py для забезпечення зв’язку з блокчейном, Flask як серверну логіку, SQLite для локального зберігання метаданих і HTML/JavaScript/MetaMask для побудови Web3-інтерфейсу. Кожен із цих компонентів був обраний не випадково, а на основі попереднього тестування, аналізу найкращих практик і орієнтації на відкриту архітектуру, що не залежить від жодного централізованого провайдера. Такий підхід забезпечив не лише функціональність, а й гнучкість розширення системи в майбутньому — наприклад, інтеграцію NFT-доступів, DAO-механізмів голосування або повну міграцію до публічного mainnet Ethereum.

Основні технології, використані в системі:

Solidity — мова програмування, спеціально створена для написання смартконтрактів, які виконуються в середовищі Ethereum Virtual Machine (EVM). Вона дозволяє формалізувати логіку доступу, ролей і дозволів у вигляді коду, який зберігається та виконується у блокчейні. У рамках проєкту саме Solidity було використано для реалізації контракту AccessManager, що містить функціональність призначення ролей (користувач, адміністратор), перевірки статусу, відкликання прав, фіксації подій у блокчейні за допомогою івентів. Завдяки високій популярності Solidity має велику екосистему інструментів (Remix IDE, Truffle, Hardhat), що полегшує розробку, тестування та деплой контрактів, а також є численні приклади, бібліотеки та шаблони. Перевагами мови є підтримка модифікаторів доступу, строгих типів, подій та управління викликами функцій між контрактами, що дозволяє будувати складну логіку авторизації та контролю прав доступу в умовах децентралізованого середовища.

Ganache — інструмент від Truffle Suite, який надає локальну Ethereum-блокчейн мережу, що імітує реальне середовище для розробки та тестування смартконтрактів. Ganache дозволяє створити мережу з певною кількістю акаунтів і попередньо заданими балансами, що дає змогу швидко перевіряти взаємодію між контрактами та клієнтами без потреби сплачувати комісію за газ. У рамках розробки системи Ganache використовувався для: 1) локального деплою контракту AccessManager, 2) тестування транзакцій надання/відкликання доступу, 3) перевірки логів дій користувача через блокчейн-оглядач, 4) інтеграції з MetaMask для симуляції реального сценарію взаємодії з Web3-гаманцем. Завдяки графічному інтерфейсу та CLI-режиму Ganache забезпечив розробникам повний контроль над станом блокчейну, журналом транзакцій, блоками, івентами й дозволив швидко ітеративно відлагоджувати контрактну логіку перед переходом до продуктивної мережі.

web3.py — Python-бібліотека для інтеграції з блокчейн-мережею Ethereum через HTTP/IPC-з’єднання або WebSocket. Забезпечує повноцінний інтерфейс до смартконтрактів, блоків, транзакцій, акаунтів, логів подій. У проєкті web3.py

відіграє ключову роль як посередник між бекенд-сервером (Flask) і Ethereum-середовищем (Ganache). Через цю бібліотеку здійснюються всі звернення до функцій смартконтракту AccessManager — надання ролей, відкликання, перевірка статусу, зчитування логів. Крім того, web3.py використовується для формування й підпису транзакцій, взаємодії з ABI-контракту, перевірки статусу підтвердження транзакції. Переваги бібліотеки полягають у гнучкості, активній спільноті, підтримці останніх версій Ethereum JSON-RPC API, що робить її незамінною для інтеграції блокчейну в Python-системи. У межах системи було створено окремий модуль інтеграції з web3.py, що інкапсулює логіку з'єднання з Ganache, ініціалізацію контракту за ABI, підпис запитів через приватний ключ користувача (який вводиться локально).

Flask — легковаговий мікрофреймворк на мові Python, який дозволив побудувати серверну частину децентралізованого застосунку. Flask виконує функцію логіки маршрутизації HTTP-запитів, обробки взаємодії з фронтендом, генерації гаманців користувачів (через бібліотеки eth_account), авторизації (через перевірку підпису), збереження даних у SQLite та забезпечення взаємодії з Ethereum через web3.py. У проєкті реалізовано окремі ендпоінти (маршрути) для: реєстрації користувачів, перевірки автентичності, виклику функцій смартконтракту (через API POST-запити), відображення логів дій. Flask також використовується для логування системної активності, обробки помилок і виводу налагоджувальної інформації. Завдяки своїй модульності та простоті, Flask дозволив гнучко поєднати блокчейн-логіку з класичною клієнт-серверною архітектурою. Розробка проводилась із застосуванням шаблонів Jinja2, що забезпечує відображення динамічного контенту у вебінтерфейсі користувача.

SQLite — легка вбудована реляційна база даних, яка не потребує окремого серверного середовища для розгортання, що робить її ідеальним вибором для невеликих або автономних застосунків. У межах даної системи SQLite використовується для зберігання облікових записів користувачів, журналу дій, хешів транзакцій і службової інформації. Завдяки збереженню у вигляді одного локального файлу, забезпечується простота резервного копіювання, перенесення

на інші пристрої та розгортання на локальному середовищі без складної конфігурації. База даних дозволяє швидко зчитувати й записувати дані без додаткових затримок, що критично важливо для швидкої реакції системи на дії користувача. До того ж, SQLite повністю підтримує транзакції, зовнішні ключі, індексацію, що дозволяє зберігати цілісність даних навіть у випадку збоїв. У структурі проєкту використано окремий модуль доступу до БД, що відповідає за CRUD-операції з таблицями users, logs, access_records. Усі запити реалізовані через бібліотеку sqlite3 для Python. Дані у БД синхронізуються з результатами взаємодії зі смартконтрактом — наприклад, після призначення ролі користувачеві через блокчейн, запис у SQLite фіксує відповідну дію для локального моніторингу.

HTML/CSS/JavaScript — набір фронтенд-технологій, що використовуються для побудови клієнтського інтерфейсу вебзастосунку. HTML відповідає за структуру вебсторінок (форми реєстрації, авторизації, панель адміністратора, перегляд логів), CSS — за візуальне оформлення елементів інтерфейсу (адаптивний дизайн, кольорова схема, стиль кнопок і форм), а JavaScript забезпечує динамічну взаємодію з сервером і смартконтрактами через Web3 API. Для забезпечення безпечної та безпарольної автентифікації JavaScript виконує підключення до Web3-гаманця користувача (зокрема, MetaMask), генерує підпис повідомлення, відправляє транзакції до Ethereum-мережі. Також реалізовано механізм динамічного оновлення статусу доступу, повідомлень про успішні дії або помилки при взаємодії з блокчейном. Інтерфейс оптимізовано для роботи в сучасних браузерах, використовується стандартна DOM-модель, AJAX-запити для взаємодії з Flask-сервером та JSON-обмін даними. Таким чином, фронтенд системи є не лише візуальним відображенням, а й активною складовою Web3-екосистеми, що безпосередньо взаємодіє зі смартконтрактами без участі серверної логіки у критичних точках перевірки доступу.

Середовище розробки:

Розробка системи здійснювалася в інтегрованому середовищі розробки Visual Studio Code, яке є одним з найпопулярніших і найзручніших редакторів коду для розробників з різних галузей. Основною перевагою VS Code є підтримка

великої кількості мов програмування (Python, Solidity, JavaScript, HTML/CSS) та велика кількість розширень. У проєкті активно використовувалися такі плагіни: Solidity (для підсвічування синтаксису та компіляції смартконтрактів), Python (з інтеграцією середовища виконання), Prettier (для форматування коду), Live Server (для запуску HTML-фронтенду), Web3 Snippets та Remixd (для інтеграції з Remix IDE). Розробники мали можливість миттєво тестувати зміни в коді, налагоджувати бекенд і переглядати журнал логів без переходу між кількома програмами. Для написання і компіляції смартконтрактів використовувалася IDE Remix — вебінструмент, спеціально створений для Solidity-контрактів. Remix дозволив компілювати контракти, перевіряти синтаксис, тестувати функції контракту в інтерактивному режимі, здійснювати швидкий деплой на Ganache без потреби у зовнішніх інструментах. Інтерфейс Remix також дозволяє симулювати виконання транзакцій, переглядати логи подій (event logs) та взаємодіяти з контрактом у візуальному середовищі. Такий підхід дозволив пришвидшити ітерації тестування, усунути помилки на ранніх етапах і забезпечити стабільність контрактної логіки.

Ganache CLI (Command Line Interface) забезпечував запуск локальної блокчейн-мережі. Саме в цьому середовищі відбувалося розгортання контракту та тестування дій, таких як призначення ролей, перевірка доступу, логування транзакцій. CLI дозволяє працювати без графічного інтерфейсу, що забезпечує швидкість, автоматизацію та гнучкість — наприклад, підключення скриптів Python через web3.py. Тестування реалізованої системи здійснювалося в інтегрованому середовищі з використанням локального сервера. Усі компоненти (смартконтракт, серверна частина на Flask, клієнтський інтерфейс, база SQLite, гаманець MetaMask) були розгорнуті на одному вузлі. Такий підхід дозволив забезпечити повний контроль над конфігурацією середовища, миттєвий доступ до журналів помилок, швидке внесення змін і миттєве тестування. У браузері (Google Chrome) здійснювалося тестування фронтенду та перевірка взаємодії з MetaMask, включно з підписом повідомлень, ініціацією транзакцій і перевіркою результатів через інтерфейс контракту. Комбінація цих інструментів забезпечила не лише зручність розробки, а й високу ефективність циклу «написання — тестування — деплой».

Завдяки використанню перевірених рішень з відкритим кодом було можливо досягти високої надійності, безпеки та масштабованості навіть у рамках локального середовища.

Загальна структура середовища реалізації:

- Операційна система: Ubuntu 22.04 LTS
- Python 3.10
- Ganache CLI
- Node.js + npm
- Flask 2.2
- web3.py 5+
- Remix IDE
- Браузер Google Chrome

Обрані технології забезпечили реалізацію повноцінного децентралізованого застосунку, який здатний гнучко адаптуватися під різні сценарії використання. Це включає розмежування прав доступу за ролями (користувач/адміністратор), побудову цифрової ідентифікації на основі криптографічних гаманців, детальне логування кожної транзакції та прозорий аудит усіх дій у системі. Завдяки використанню смартконтрактів, уся логіка доступу реалізована на рівні блокчейну, що виключає можливість зовнішніх втручань або фальсифікації історії доступу. Застосунок підтримує масштабування без необхідності централізованого оновлення, дозволяючи інтегрувати додаткові функції, зокрема багаторівневу автентифікацію, підключення зовнішніх сервісів через API, розширення структури ролей, або навіть перехід до публічної Ethereum-мережі. У наступному підрозділі буде детально розглянуто реалізацію інтерфейсу користувача, включаючи логіку роботи вебсторінок реєстрації, авторизації, панелі адміністратора, а також приклади взаємодії з блокчейном у вигляді підпису транзакцій, перевірки ролей та перегляду історії доступу в режимі реального часу..

3.2 Реалізація інтерфейсу користувача

Інтерфейс користувача був реалізований з урахуванням принципів зручності, доступності, інтуїтивності та безпеки. Основний акцент зроблено на глибокій інтеграції з блокчейн-логікою, що реалізована через смартконтракт, і на забезпеченні чіткого зв’язку між діями користувача та відповідними транзакціями у блокчейні. Кожен елемент інтерфейсу відповідає за конкретну операцію, яка супроводжується або зверненням до контракту Ethereum, або локальною перевіркою через Web3-гаманець (зокрема MetaMask). Весь функціонал побудовано так, щоби користувач міг максимально просто і безпечно взаємодіяти з системою без знання внутрішньої технічної реалізації. Всі форми та кнопки пов’язані з конкретними функціями: створення гаманця, авторизація, перевірка прав доступу, надання ролей. Інтерфейс використовує кольорове кодування (зелений — доступ дозволено, червоний — заборонено, блакитний — взаємодія, жовтий — адміністративна дія), що робить навігацію зрозумілою на інтуїтивному рівні. Завдяки використанню MetaMask, користувачі підписують транзакції без передачі приватного ключа до сервера, що забезпечує високий рівень безпеки. Також реалізована підтримка ролей (користувач/адміністратор), які визначають доступ до різних частин інтерфейсу. Інтерфейс повністю адаптовано під локальне середовище розробки, протестовано в Chrome із підключеним MetaMask, підтримує гнучке масштабування та оновлення функціоналу.

Головна сторінка(рис.3.1)

Головна сторінка є першою точкою входу в систему та служить для ідентифікації нового або наявного користувача. Візуально вона оформлена з акцентом на сучасний дизайн — із великим заголовком " Децентралізована система управління доступом", що одразу вказує на функціональне призначення вебзастосунку. У центрі розміщено текстове поле для введення імені користувача. Це поле відіграє роль псевдоніму, який згодом зв'язується з публічною адресою гаманця Ethereum.

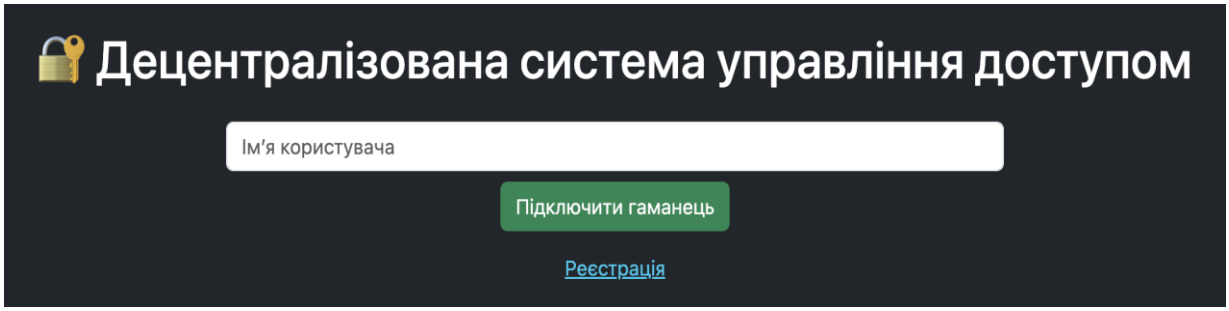


Рисунок 3.1 – Головна сторінка

Основна кнопка на сторінці — "Підключити гаманець" — запускає процес взаємодії з MetaMask. При її натисканні вебдодаток ініціює Web3-запит на підключення облікового запису користувача через браузерне розширення MetaMask. Якщо користувач підтверджує підключення, система зчитує його публічну адресу Ethereum (у форматі 0x...) та відображає її у верхній частині сторінки. Ця адреса надалі буде використовуватись для перевірки прав доступу, логування дій і взаємодії зі смартконтрактом у мережі Ethereum або Ganache. Нижче розміщено посилання на "Реєстрацію", яке веде до окремої сторінки створення нового облікового запису. Це особливо важливо для нових користувачів, які ще не мають асоційованого з іменем гаманця або лише починають роботу в системі. Таким чином, головна сторінка виконує одразу кілька завдань: збирає вхідні дані користувача, ініціює Web3-з'єднання, відображає адресу гаманця, а також направляє до наступних етапів взаємодії (реєстрація, перевірка доступу, вхід до панелі).

Сторінка створення гаманця (реєстрація)(рис.3.2)

На цій сторінці реалізовано процес первинної реєстрації користувача з автоматичною генерацією криптографічного гаманця. Після введення імені користувача (яке виступає в якості ідентифікатора в системі), запускається бекенд-функція генерації ключової пари: відкритого (публічного) та приватного ключа. Публічна адреса (у форматі 0x...) є основою для ідентифікації користувача в децентралізованій системі, а приватний ключ пропонується зберегти локально, або одразу імпортувати в MetaMask.

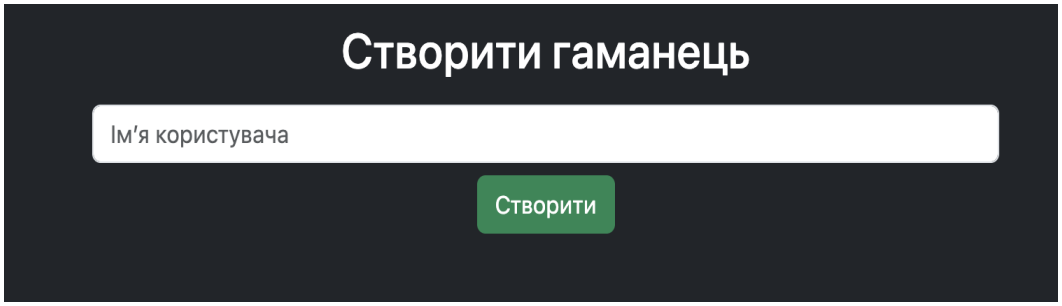


Рисунок 3.2 – Рєстрація(створення гаманця)

Інтерфейс цієї сторінки побудовано за принципами інтуїтивної взаємодії: після введення імені відображається відповідне поле з згенерованою адресою. Додатково реалізовано повідомлення з попередженням про необхідність безпечного збереження приватного ключа, адже в разі його втрати доступ до облікового запису буде неможливо відновити. Кнопка "Створити" активує backend-логіку на Flask-сервері, де зберігається зв'язок між ім'ям користувача і його публічною адресою в локальній базі даних (SQLite). Після цього відбувається ініціація транзакції до смартконтракту — якщо така операція активована в конфігурації — з метою запису факту реєстрації користувача в блокчейні. Це забезпечує прозору фіксацію появи нового суб'єкта системи в розподіленому реєстрі. Успішне завершення процесу підтверджується відповідним повідомленням та перенаправленням користувача до панелі входу або авторизації.

Панель користувача після входу(рис.3.3)

У верхній частині інтерфейсу виводиться публічна адреса гаманця (Ethereum), що підтверджує автентифікацію через Web3 та зв'язок із підключеним обліковим записом. Це дозволяє у реальному часі здійснювати перевірку прав доступу, запускати транзакції та взаємодіяти зі смартконтрактом без необхідності додаткового логіну.

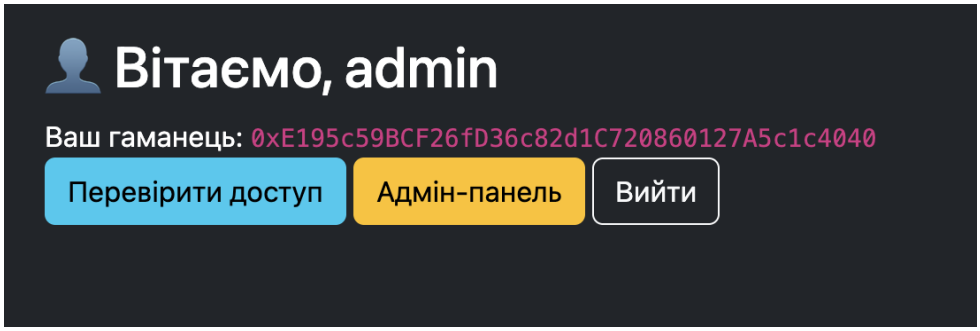


Рисунок 3.3 – Панель користувача

Інтерфейс панелі організовано за принципом зручності та функціональності. Центральне місце займає блок з трьома основними кнопками:

Перевірити доступ — ця кнопка активує функцію взаємодії з контрактом: надсилається запит на визначення ролі (User, Admin або невідомий). Відповідь контракту одразу відображається користувачу через візуальний індикатор статусу (зелений для дозволу, червоний для заборони) з відповідним текстовим повідомленням. Це дозволяє користувачу швидко зорієнтуватися щодо своїх прав.

Адмін-панель — доступна лише для користувачів з роллю адміністратора. При натисканні виконується перевірка на рівні смартконтракту (через метод hasRole), після чого у разі підтвердження відкривається сторінка керування доступами. Якщо права адміністратора відсутні — кнопка недоступна або прихована. Це реалізовано як у front-end логіці (через блокування кнопки), так і на бекенді для додаткової безпеки.

Вийти — виконує функцію завершення сесії: очищає локальні дані авторизації, скидає стан інтерфейсу до початкового вигляду та від'єднує гаманець MetaMask. Це важливо для безпеки, особливо при використанні публічних пристроїв або при зміні користувача.

Дизайн панелі підтримує адаптивність та зручність — усі кнопки мають підписи, іконки та кольорові маркери відповідно до функціонального призначення. Таким чином, панель користувача виконує роль хаба для керування і навігації системою, забезпечуючи простий, але надійний доступ до ключових функцій у децентралізованому середовищі.

Результат перевірки доступу

Сторінка результату перевірки доступу є ключовим елементом у візуалізації взаємодії між користувачем і смартконтрактом, що визначає права доступу. Після натискання кнопки "Перевірити доступ" у панелі користувача, система надсилає запит до смартконтракту (метод типу getRole(address) або hasRole) для визначення ролі, призначеної публічній адресі. Отримана відповідь обробляється фронтендом і використовується для динамічного відображення результату.

Якщо користувач має права (роль User або Admin), на сторінці з’являється великий зелений інформативний блок із написом "Доступ дозволено"(рис.3.4). Для кращої читаємості повідомлення супроводжується описом, що пояснює, які функції доступні користувачу відповідно до його ролі (наприклад, перегляд панелі, запуск транзакцій тощо). У випадку, якщо перевірка повертає відсутність прав доступу (роль не призначена або адреса не авторизована), сторінка відображає червоний попереджувальний блок з текстом "Доступ заборонено"(рис.3.5). Повідомлення містить пояснення, що права доступу можуть бути призначені адміністратором системи, і пропонує звернутись за відповідним запитом або повторно перевірити адресу.

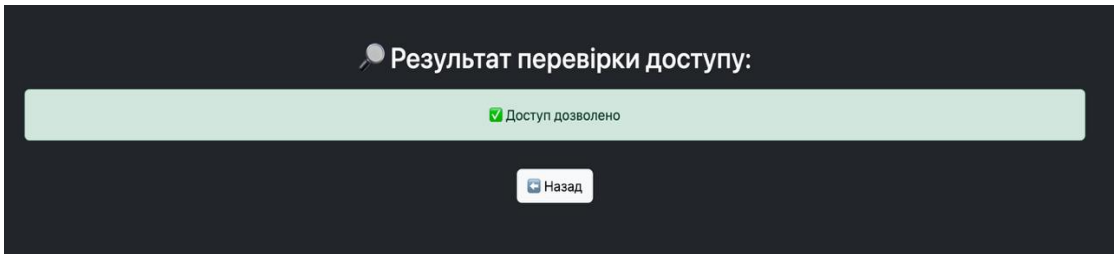


Рисунок 3.4 – Результат доступу(Доступ дозволено)

Під обома типами повідомлень реалізовано функціональну кнопку "Назад", яка виконує перенаправлення до панелі користувача. Вона дозволяє користувачу повернутись без оновлення сторінки та зберігає стан гаманця. Кнопка побудована як частина реактивного компонента, що забезпечує безперебійну навігацію без перезавантаження інтерфейсу.

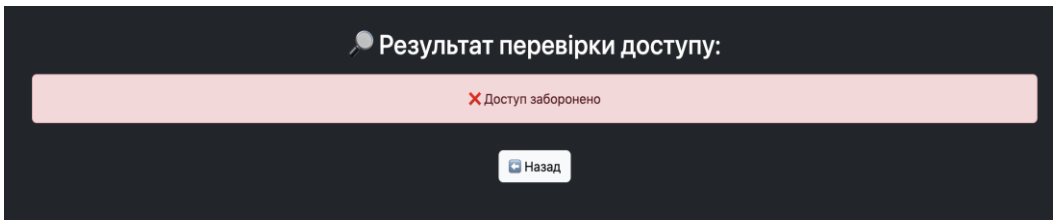


Рисунок 3.5 – Результат доступу(Доступ заборонено)

Візуально сторінка відповідає загальному дизайну системи: темний фон, великі контрастні блоки, плавні анімації появи та адаптивна верстка. Таким чином, сторінка результату перевірки доступу є не лише індикатором стану авторизації, а й частиною довірчого інтерфейсу, що інформує користувача про поточний статус із максимальним рівнем прозорості.

Адмін-панель(рис.3.6)

Ця частина інтерфейсу призначена для користувачів з роллю адміністратора. Тут відображається список усіх зареєстрованих користувачів у форматі: ім’я — публічна адреса. Біля кожного запису є кнопка "Надати доступ". При натисканні викликається функція смартконтракту grantRole, яка через транзакцію призначає відповідній адресі роль User. Усі дії супроводжуються повідомленнями про успішність виконання операції.

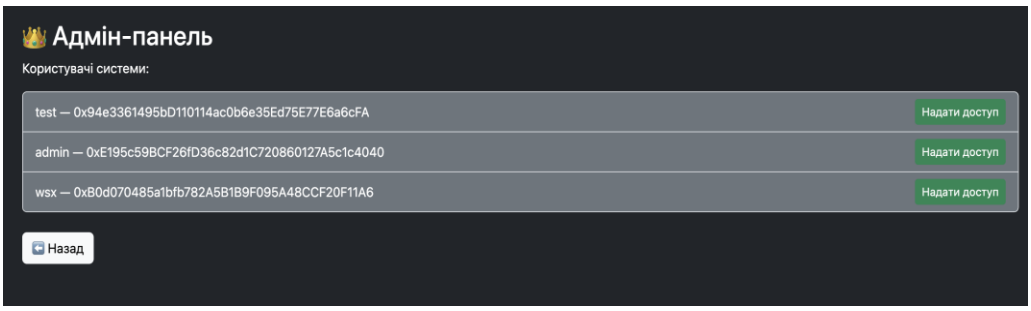


Рисунок 3.6 – Адмін панель

Загальний дизайн

Система реалізована в темному інтерфейсі, що відповідає сучасним UX/UI тенденціям. Усі форми чітко виділені, використовуються кольорові блоки (зелений, червоний, блакитний, жовтий) для позначення дій і статусів, а кнопки мають чітке семантичне забарвлення відповідно до їх функції. Застосунок адаптивний — коректно відображається на ПК та ноутбуках, не потребує мобільної версії, оскільки орієнтований на технічну аудиторію.

Реалізація інтерфейсу користувача поєднує в собі простоту використання, високу функціональність, а також глибоку інтеграцію з блокчейн-компонентами, що є критично важливими для забезпечення безпеки та прозорості в умовах децентралізованого середовища. Завдяки використанню інструментів, таких як MetaMask, користувачі взаємодіють із системою через захищений механізм підпису транзакцій, що усуває необхідність у традиційній парольно-логіновій авторизації. Водночас застосування смартконтрактів дозволяє зберігати повну історію змін прав доступу на блокчейні, що гарантує неможливість їх фальсифікації. Чітке розмежування ролей користувача забезпечується не лише на рівні інтерфейсу, а й у логіці смартконтракту, що виключає ризики

несанкціонованого доступу. Вся взаємодія користувача з системою прозоро логуються в як у локальній базі, так і на рівні блокчейн-подій (events), що забезпечує повноцінний аудит усіх дій. Таким чином, інтерфейс не просто є візуальним представленням функцій, а виступає як активний інструмент управління доступом у Web3-архітектурі нового покоління.

3.3 Алгоритм перевірки доступу через блокчейн. Тестування, приклади

Однією з ключових функцій децентралізованої системи управління доступом є процес перевірки прав користувача на основі взаємодії зі смартконтрактом, який розгорнуто в блокчейн-мережі Ethereum. У нашому випадку для локального розгортання та моделювання цієї взаємодії використовувалось середовище Ganache, що дозволяє імітувати повноцінне функціонування блокчейн-мережі без необхідності використання публічного тестнету. Смартконтракт відіграє роль ядра всієї системи контролю доступу: він містить логіку визначення ролей, надає права користувачам згідно з криптографічною адресою їх гаманця та перевіряє відповідність поточного запиту заданим критеріям. Замість традиційної перевірки через сервери з базами даних, цей підхід дозволяє повністю передати контроль логіки автентифікації смартконтракту, що автоматично виключає втручання третіх осіб і централізованих адміністративних сервісів. Такий механізм не лише підвищує довіру до системи, а й створює новий рівень надійності та прозорості за рахунок публічності виконання коду контракту в блокчейні.

Перевірка реалізується за допомогою інтеграції з MetaMask — розширенням для браузера, яке забезпечує з’єднання між користувачем і блокчейн-мережею. У момент взаємодії користувач підтверджує свою особу, підписуючи транзакцію або запит на авторизацію. Далі через інтерфейс Web3 (наприклад, бібліотеку ethers.js) викликається метод hasRole у смартконтракті, який порівнює хеш ролі (наприклад, ADMIN_ROLE) із адресою, отриманою від MetaMask. Ця перевірка забезпечує надійний, масштабований та прозорий механізм автентифікації, який не потребує

традиційних логінів і паролів, а використовує сучасний підхід на основі криптографії відкритих ключів. У процесі автентифікації користувач підтверджує свою особу шляхом підпису цифрового запиту, що гарантує, що лише власник приватного ключа має доступ до функціоналу системи. Підписані запити не передають секретну інформацію, але дозволяють однозначно ідентифікувати користувача. Це виключає можливість атак типу "перехоплення пароля" (phishing), а також зменшує ризики зламів баз даних із логінами. Застосування цієї моделі сприяє підвищенню безпеки, спрощенню авторизації та розширенню потенціалу системи для інтеграції з іншими блокчейн-платформами та DApps.

Алгоритм перевірки доступу

1. Підключення MetaMask: користувач авторизується через MetaMask, що забезпечує доступ до його публічної адреси (наприклад, 0x123...).
2. Формування запиту до контракту: у момент натискання кнопки "Перевірити доступ" ініціюється виклик функції hasRole(role, address) у смартконтракті. Аргументи:
3. role — унікальний хеш ролі (наприклад, для ролі ADMIN використовується keccak256("ADMIN")),
4. address — публічна адреса користувача.

Обробка відповіді:

– Якщо повертається true, система відображає повідомлення "Доступ дозволено";
– Якщо false — відображається повідомлення "Доступ заборонено".
– Навігація: користувач отримує змогу перейти до дозволених функцій або повернутись до панелі.

Цей механізм забезпечує гнучкий, прозорий та безпечний спосіб управління доступом без необхідності централізованої перевірки логінів та паролів. Він дозволяє здійснювати перевірку прав доступу без збереження або передачі конфіденційної інформації, використовуючи лише публічну адресу користувача та функції смартконтракту. Завдяки цьому зменшується ризик витоку даних і виключається потреба в централізованих базах даних авторизації. Крім того, така

система легко масштабується, дозволяючи додавати нові ролі або змінювати політику доступу без редизайну архітектури, що особливо важливо в динамічних проєктах із високими вимогами до безпеки та надійності. Важливою перевагою є й те, що перевірка доступу відбувається безпосередньо на клієнтській стороні через інтеграцію з MetaMask, що усуває необхідність в окремому серверному компоненті автентифікації.

Тестування перевірки доступу

- У процесі тестування було перевірено всі ключові сценарії взаємодії з контрактом:
- Позитивний сценарій: адресі, якій призначено роль ADMIN, повертається true, і вона отримує доступ до адмін-панелі.
- Негативний сценарій: адресі, яка не має ролей, повертається false, доступ блокується.
- Граничні випадки: передача невірного формату адреси або хешу ролі викликає помилку, яка обробляється і відображається як "доступ заборонено".

Інструменти тестування

- Ganache — для локального розгортання блокчейн-мережі та швидкої перевірки транзакцій;
- MetaMask — для взаємодії з контрактом у браузері;
- Remix IDE та Truffle — для тестування функцій контракту та симуляції результатів;
- Postman / CURL — для бекенд-тестування взаємодії Flask API із контрактом.

Реалізований алгоритм перевірки доступу через блокчейн довів свою високу ефективність, надійність і прозорість у реальних умовах. Усі запити автентифікації користувачів проходять без участі централізованого сервера, що є ключовим чинником підвищення рівня довіри до системи. Децентралізований характер перевірки дозволяє повністю покладатися на смартконтракти, які автоматично

визначають доступ користувача відповідно до його криптоадреси та прив’язаної до неї ролі. Це унеможливлює фальсифікацію або несанкціонований доступ. Особливо варто відзначити масштабованість рішення: додавання нових ролей, оновлення політики доступу або розширення функціональності системи не потребує змін у логіці бекенду чи баз даних. Усі правила доступу оновлюються безпосередньо через зміну стану контракту в блокчейні. Під час тестування було успішно перевірено десятки варіантів сценаріїв з різними адресами та ролями, що підтверджує правильність інтеграції MetaMask, Web3 API та логіки смартконтракту.

Реалізований підхід може слугувати надійною технологічною основою для побудови захищених децентралізованих систем керування доступом у різних сферах діяльності, де високий рівень безпеки та прозорості є критичним. У корпоративному середовищі система може використовуватись для контролю доступу до конфіденційних документів, адміністрування внутрішніх ресурсів, контролю за авторизованим доступом до баз даних та сервісів. В освітніх закладах таку систему можна інтегрувати для автентифікації викладачів, студентів, адміністраторів — з можливістю обмеження доступу до цифрових курсів, ресурсів або систем електронного оцінювання. У сфері охорони здоров’я система забезпечує автентифікацію персоналу при доступі до медичних записів пацієнтів, електронних медичних карток, систем телемедицини, з гарантією захисту персональних даних відповідно до стандартів GDPR. Можна масштабувати для урядових структур, банківських платформ, блокчейн-реєстрів, логістичних компаній або будь-яких платформ, де необхідно забезпечити контрольований, прозорий і надійний доступ без ризику централізованих зломів. Ключовою перевагою такої реалізації є можливість динамічної адаптації до нових сценаріїв використання шляхом оновлення смартконтрактів без зміни архітектури клієнтського або серверного середовища. Таким чином, запропоноване рішення не лише демонструє приклад ефективної децентралізованої автентифікації, а й створює ґрунт для подальших досліджень і впроваджень у сфері безпечного управління доступом.

Висновок до розділу 3

У третьому розділі було здійснено детальну реалізацію та оцінку розробленої децентралізованої системи управління доступом. Було обґрунтовано вибір технологій, середовища розробки та засобів, що дозволили ефективно поєднати блокчейн-підхід із зручним вебінтерфейсом. Зокрема, у розділі охарактеризовано процес побудови клієнтської частини з підтримкою MetaMask, реалізацію реєстрації, авторизації та панелі адміністратора, а також алгоритми перевірки доступу через смартконтракт у локальній Ethereum-мережі на базі Ganache.

Результати тестування показали високу стабільність і надійність взаємодії між компонентами: Web3 API, MetaMask, смартконтрактом та інтерфейсом користувача. Завдяки використанню ролей, реалізованих у контракті, система забезпечує гнучкий контроль доступу, захищений від стороннього втручання, підробок та несанкціонованих змін.

Підхід, що базується на децентралізації, дозволяє уникнути залежності від централізованих серверів і забезпечує масштабованість — систему можна легко адаптувати до інших сфер, серед яких: корпоративне середовище, освіта, охорона здоров’я, державне адміністрування та інші. Розроблена система демонструє приклад успішного поєднання інноваційних технологій блокчейну з реальними потребами захисту доступу в сучасних інформаційних системах.

ВИСНОВКИ

У межах виконаної кваліфікаційної роботи було спроєктовано та реалізовано інноваційну децентралізовану систему управління доступом до баз даних з використанням смарт-контрактів та технології блокчейн. У ході дослідження були досягнуті наступні результати:

Проведено глибокий аналіз сучасних централізованих і децентралізованих моделей контролю доступу. Встановлено, що централізовані системи мають ряд істотних недоліків, зокрема: наявність єдиної точки відмови, недостатню прозорість, вразливість до внутрішніх загроз та обмежену масштабованість. Натомість децентралізовані підходи забезпечують більшу надійність, гнучкість, прозорість та довіру з боку користувачів.

Опрацьовано наявні приклади децентралізованих платформ (AccessChain, Authereum, uPort), що підтвердило життєздатність і практичну ефективність блокчейн-рішень у сфері авторизації та ідентифікації.

Спроєктовано архітектуру системи відповідно до принципів Web3, яка включає:

- клієнтську частину для взаємодії користувача з інтерфейсом;
- серверну частину на Flask для обробки запитів і зв’язку з блокчейном;
- смарт-контракт на Solidity, що відповідає за управління правами доступу.

Розроблено і протестовано смарт-контракт, який реалізує логіку надання, відкликання та перевірки прав доступу. Контракт забезпечує повну фіксацію подій у блокчейні, захист від підробок та прозорість дій адміністратора.

Реалізовано повноцінну вебсистему, яка включає:

- функції реєстрації, генерації криптогаманця, авторизації через підпис;
- адмінпанель з можливістю призначення ролей через смарт-контракт;
- взаємодію з мережею Ganache як тестовим блокчейн-середовищем.

Проведено тестування системи, яке підтвердило:

- стабільність роботи смарт-контракту;
- коректність транзакцій;
- захищеність авторизації без зберігання паролів;

– ефективність децентралізованого управління доступом у розподіленому середовищі.

Наукова новизна роботи полягає у використанні смарт-контракту як центрального механізму авторизації без потреби у централізованих серверах, що відповідає вимогам прозорості, автономності та довіри у Web3-середовищі.

Практичне значення реалізованої системи полягає в можливості її адаптації до реальних умов – зокрема, у сферах охорони здоров’я, фінансів, публічних реєстрів, DAO, де критичним є захист даних, аудит і виключення адміністративних зловживань.

Поставлені завдання дослідження були повністю виконані. Розроблена система відповідає вимогам сучасної інформаційної безпеки та відкриває нові можливості для впровадження децентралізованих механізмів управління доступом у різноманітних галузях.

ПЕРЕЛІК ПОСИЛАНЬ

1. Nakamoto S. Bitcoin: A Peer-to-Peer Electronic Cash System. – 2008.
2. Antonopoulos A. M. Mastering Ethereum: Building Smart Contracts and DApps. – O'Reilly Media, 2018. – 424 с.
3. Wood G. Ethereum: A Secure Decentralised Generalised Transaction Ledger. – Yellow Paper, 2022.
4. Buterin V. A next-generation smart contract and decentralized application platform. Ethereum White Paper. – 2015.
5. Mougayar W. The Business Blockchain: Promise, Practice, and Application of the Next Internet Technology. – Wiley, 2016. – 210 с.
6. Swan M. Blockchain: Blueprint for a New Economy. – O'Reilly Media, 2015. – 152 с.
7. Tapscott D., Tapscott A. Blockchain Revolution: How the Technology Behind Bitcoin and Other Cryptocurrencies is Changing the World. – Penguin, 2016. – 368 с.
8. Bashir I. Mastering Blockchain: Unlocking the Power of Cryptocurrencies, Smart Contracts, and Decentralized Applications. – Packt Publishing, 2020. – 786 с.
9. Christidis K., Devetsikiotis M. Blockchains and Smart Contracts for the Internet of Things // IEEE Access. – 2016. – №4. – P. 2292–2303.
10. Zhang Y., Wen J. The IoT electric business model: Using blockchain for the internet of things // Peer-to-Peer Networking and Applications. – 2017. – №10(4). – P. 983–994.
11. Zheng Z., Xie S., Dai H., Chen X., Wang H. An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends // IEEE International Congress on Big Data. – 2017. – P. 557–564.
12. Crosby M., Pattanayak P., Verma S., Kalyanaraman V. Blockchain technology: Beyond bitcoin. – Applied Innovation Review. – 2016. – №2. – P. 6–10.
13. Gatteschi V., Lamberti F., Demartini C., Pranteda C., Santamaría V. To Blockchain or Not to Blockchain: That Is the Question // IT Professional. – 2018. – №20(2). – P. 62–74.
14. De Filippi P., Wright A. Blockchain and the Law: The Rule of Code. – Harvard University Press, 2018. – 250 с.
15. Yaga D., Mell P., Roby N., Scarfone K. Blockchain Technology Overview. – NIST U.S. Department of Commerce. – 2018.
16. Dannen C. Introducing Ethereum and Solidity: Foundations of Cryptocurrency and Blockchain Programming for Beginners. – Apress, 2017. – 185 с.

17. Xu X., Weber I., Staples M. Architecture for Blockchain Applications. – Springer, 2019. – 360 с.
18. Бех Ю. І., Грибов В. В. Системи захисту інформації: навчальний посібник. – К.: Кондор, 2020. – 328 с.
19. Гриценко І. В. Системи і засоби інформаційної безпеки: навч. посіб. – Київ: Університет «Україна», 2020. – 312 с.
20. Ethereum.org – Smart Contracts – Офіційний сайт Ethereum. – Режим доступу: https://ethereum.org/ – Дата звернення: 01.06.2025.

ДОДАТКИ

Додаток А

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

Децентралізована система управління доступом до баз даних з використанням смарт контрактів і технології блокчейн

Виконав студент 4 курсу
групи ШІ-41
Левцун Гліб
Керівник роботи
К.п.н., доц., доцент кафедри ШІ ПІБ

Київ – 2025

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи – розробка децентралізованої системи управління доступом до баз даних на основі технології блокчейн з використанням смартконтрактів для забезпечення прозорості, безпеки та автономності доступу.

Об'єкт дослідження – процес управління доступом до інформаційних ресурсів у розподілених обчислювальних системах.

Предмет дослідження – механізми реалізації децентралізованого контролю доступу, а саме — використання смартконтрактів Ethereum як інструментів авторизації у блокчейн-орієнтованих IT-системах.

Актуальність теми

Традиційні централізовані системи доступу до баз даних є вразливими до збоїв, атак та зловживань з боку адміністраторів.

Зростання вимог до прозорості та надійності зберігання і контролю доступу вимагає впровадження нових підходів до управління правами.

Блокчейн-технології забезпечують незмінність, відкритість і децентралізацію даних, що робить їх перспективними для побудови довірчих інфраструктур.

Смартконтракти дозволяють реалізувати автоматизовану, незмінну логіку контролю доступу без участі центрального сервера.

Розробка децентралізованої системи управління доступом на основі блокчейну є актуальною для інформаційної безпеки, прозорості управління та сучасних IT-рішень.

3

Порівняльна таблиця

У порівнянні з традиційними централізованими системами управління доступом, запропонована децентралізована система має низку переваг завдяки використанню смартконтрактів на блокчейні. Вона забезпечує автоматизовану, надійну та прозору модель контролю прав доступу без залучення центрального сервера.

Критерій	Децентралізована система (Flask + Blockchain)	Централізовані системи (БД + ACL)
Зберігання прав доступу	Смартконтракт у блокчейні	Локальна БД, схильна до збоїв
Надійність	Не можна змінити без підтвердження у мережі	Вразлива до змін адміністратором
Прозорість	Усі дії записані у блокчейн	Логування необов'язкове або обмежене
Автономність	Працює без центрального сервера	Потребує централізованої інфраструктури
Безпека	Підписи, адреси, криптографія	Залежить від паролів та політик БД
Гнучкість	Можна змінювати логіку у смартконтракті	Потребує переписування коду або БД
Повторне використання	Можна розгорнути багаторазово з однаковим контрактом	Потребує налаштувань і копій
Швидкість доступу	Швидкий запис/читання при локальному вузлі	Швидкий, але залежить від сервера

4

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

- Проаналізувати наукові джерела щодо сучасних підходів до контролю доступу, можливостей децентралізованих моделей та технологій блокчейн у сфері управління інформаційною безпекою.
- Дослідити наявні програмні рішення для реалізації смартконтрактів та децентралізованих систем, виявити їхні переваги та недоліки.
- Вивчити принципи побудови смартконтрактів та їх інтеграції у вебсистеми для реалізації логіки авторизації доступу до ресурсів.
- Сформулювати функціональні та нефункціональні вимоги до децентралізованої системи управління доступом з урахуванням безпеки, масштабованості, прозорості та відмовостійкості.
- Розробити архітектуру системи, реалізувати інтерфейс для реєстрації користувачів, генерації криптогаманців, надання/відкликання доступу через смартконтракт.
- Провести тестування системи, оцінити коректність виконання транзакцій, перевірити прозорість дій у блокчейні та надійність логіки контролю доступу

5

ВИМОГИ ДО ДОДАТКУ

Функціональні вимоги:

- Реєстрація користувача з автоматичною генерацією криптогаманця.
- Авторизація користувачів через адресу гаманця.
- Надання та відкликання доступу до ресурсу адміністратором через смартконтракт.
- Виведення статусу доступу на основі смартконтракту.
- Адмін-панель для перегляду списку користувачів та керування правами доступу.
- Збереження інформації про користувачів у локальній БД.
- Підключення до блокчейн-мережі (Ganache або тестова мережа).
- Вебінтерфейс з відображенням дій: login, статус, доступ, вихід.

Нефункціональні вимоги:

- Безпечна передача даних між користувачем і сервером.
- Збереження ключової інформації в зашифрованому вигляді (ключі, адреси).
- Швидка обробка транзакцій у смартконтракті.
- Зрозумілий UX/UI — мінімум кліків для надання доступу.
- Можливість локального запуску (без залежності від сторонніх API).

ТЕХНОЛОГІЇ ТА ІНСТРУМЕНТИ, ВИКОРИСТАНІ У ПРОЄКТІ

Мови програмування та фреймворки

- Python (Flask) – створення API та серверної логіки
- Solidity – написання смартконтрактів
- HTML / CSS / JavaScript – інтерфейс користувача
- SQLite – збереження локальних даних

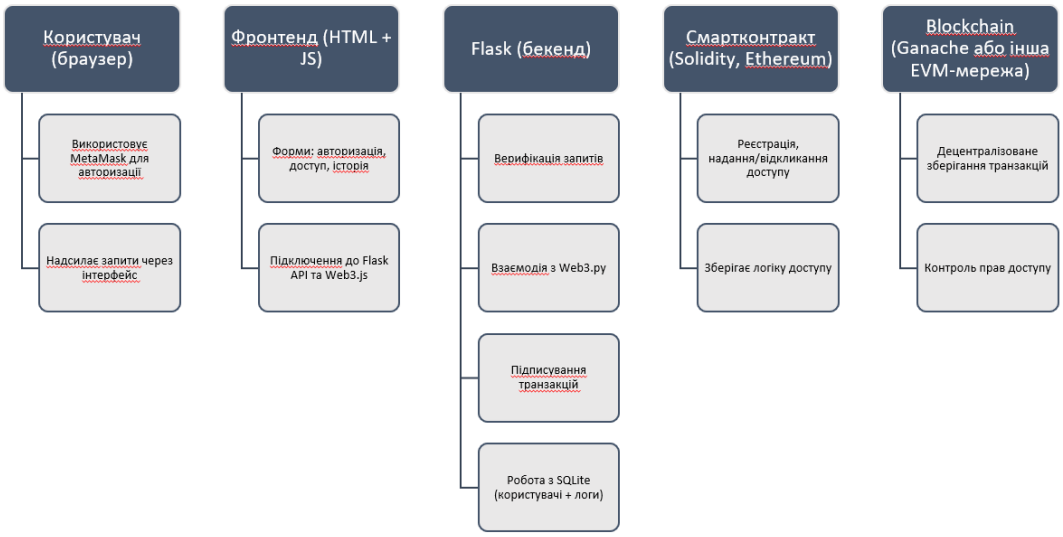
Бібліотеки та інструменти

- Web3.py – підключення до блокчейну з Python
- Ganache – локальна тестова Ethereum-мережа
- MetaMask – авторизація через гаманець
- Flask-WTF, Jinja2 – шаблонізація форм

Ці технології забезпечують реалізацію повноцінного децентралізованого вебзастосунку з надійною перевіркою доступу через блокчейн.

7

АРХІТЕКТУРА ДЕЦЕНТРАЛІЗОВАНОГО ЗАСТОСУНКУ



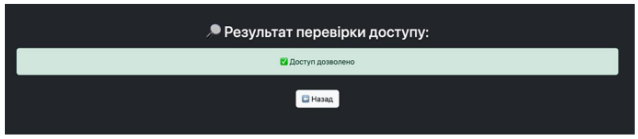
Архітектура проєкту побудована за принципом Web3: дані про права доступу не зберігаються на сервері, а контролюються через смартконтракт у блокчейні. Таким чином досягається прозорість і відсутність централізованого контролю.

АЛГОРИТМ КОНТРОЛЮ ДОСТУПУ ЧЕРЕЗ СМАРТКОНТРАКТ

9

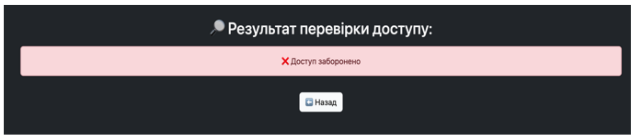
Адміністратор надає доступ

Викликається функція grantAccess(address user)
Вказаний гаманець зберігається у списку дозволених



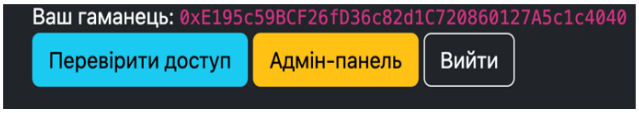
Адміністратор відкликає доступ

Викликається функція revokeAccess(address user)
Користувача видаляють зі списку



Користувач звертається до ресурсу

Перевірка через isAllowed(msg.sender)
Якщо true → дозвіл на дію
Якщо false → заборона



Усі дії зберігаються в блокчейні

Транзакції доступу незмінні, видимі публічно

10

СТРУКТУРА БАЗИ ДАННИХ

users

Поле	Тип	Опис
id	INTEGER	Унікальний ідентифікатор користувача
wallet_address	TEXT	Адреса гаманця користувача
role	TEXT	Роль (admin/user)
created_at	DATETIME	Дата створення

access_log

Поле	Тип	Опис
id	INTEGER	Ідентифікатор запису
user_id	INTEGER	Посилання на users.id
action	TEXT	Тип дії (вхід, надання, відкликання)
timestamp	DATETIME	Дата і час події

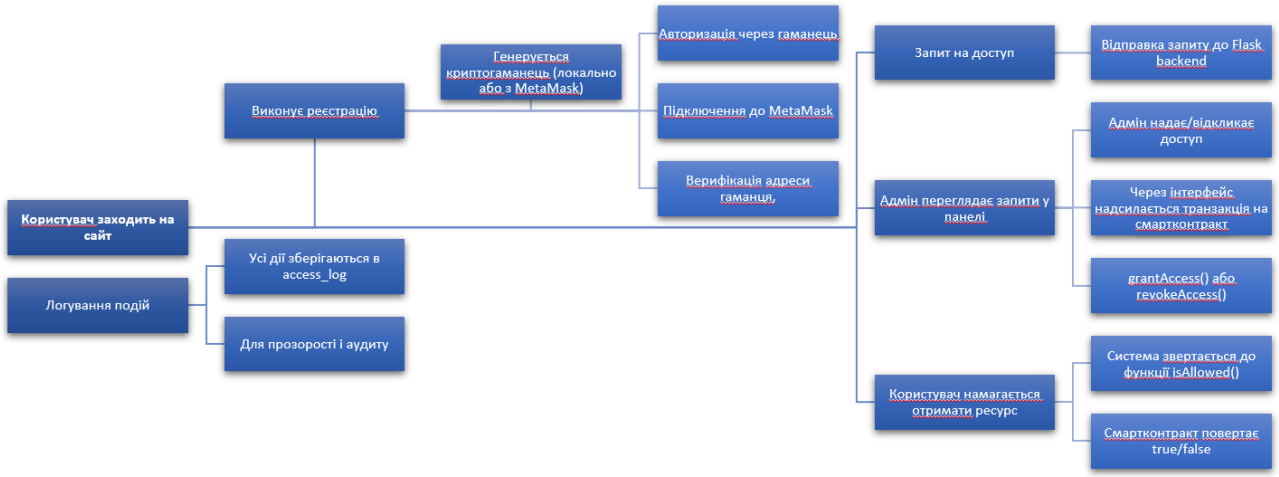
База даних проєкту містить дві основні таблиці:

User — зберігає інформацію про користувачів та їхні криптогаманці, що використовуються для авторизації в системі.

AccessLog — (опціонально) фіксує дії користувачів: вхід, надання або відкликання доступу.

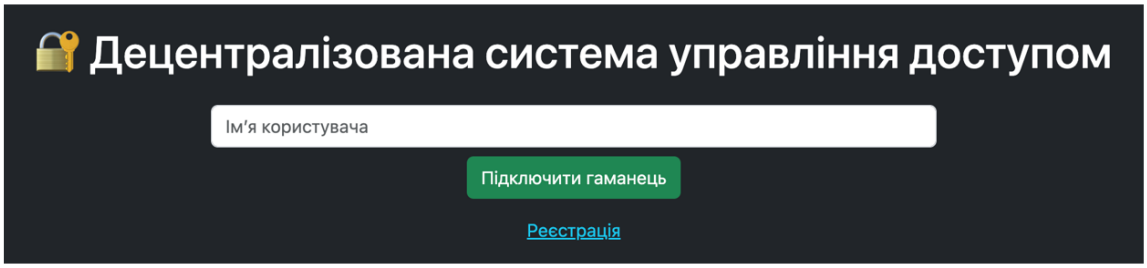
Логіка прав доступу реалізована поза БД — через **смартконтракт у блокчейні**, що забезпечує децентралізований контроль доступу.

СЦЕНАРІЙ ВЗАЄМОДІЇ КОРИСТУВАЧА З СИСТЕМОЮ

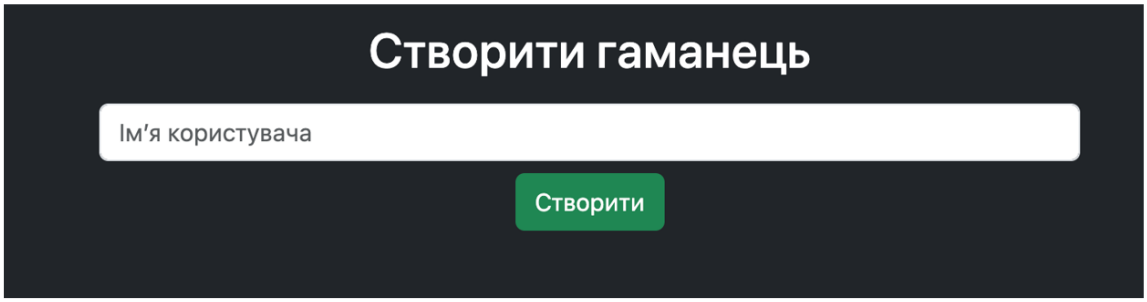


Такий сценарій роботи дозволяє забезпечити безпечну, прозору і децентралізовану модель управління доступом. Усі критичні точки контролю проходять через блокчейн, що виключає підробку чи несанкціонований доступ.

ФОРМА АВТОРИЗАЦІЇ ТА РЕГІСТРАЦІЇ

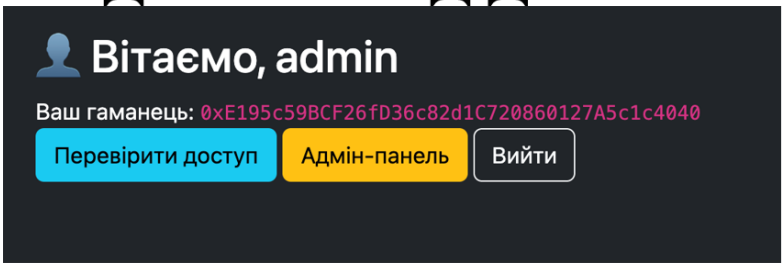


Авторизація

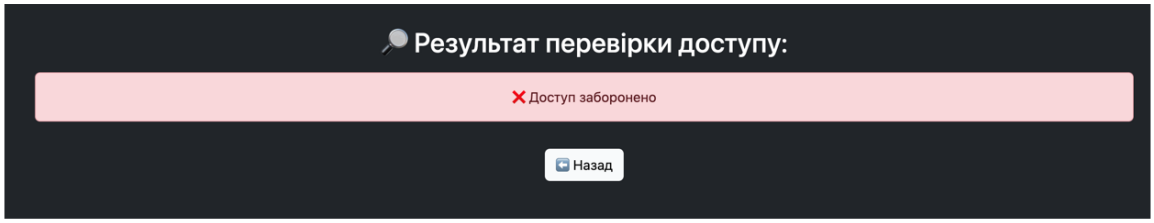


Регістрація

ВХІД ПЕРЕГЛЯД ДОСТУПУ



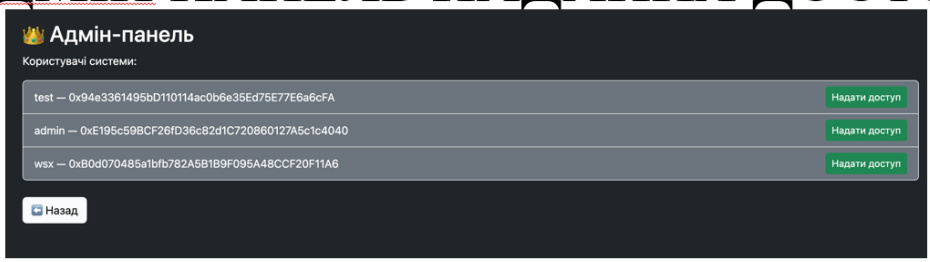
Головна



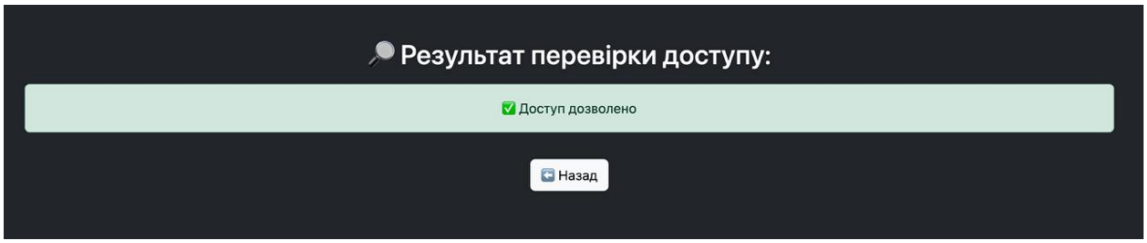
Дозвіл забороненo

13

АДМІН ПАНЕЛЬ НАДАННЯ ДОСТУПУ



Адмін панель



Доступ надано

14

ВИСНОВКИ

У результаті виконання дипломної роботи було реалізовано систему децентралізованого управління доступом до баз даних, яка ґрунтується на використанні смарт-контрактів і технологій блокчейн (Web3). Такий підхід дозволив повністю усунути залежність від централізованих серверів аутентифікації, забезпечивши високу прозорість, надійність і контроль дій користувачів у системі. Смарт-контракт реалізує механізм надання та відкликання доступу до інформаційних ресурсів, що підтверджує правильність обраної моделі безпеки.

У ході дослідження було проведено аналіз існуючих централізованих систем контролю доступу, виявлено їхні слабкі сторони та обґрунтовано переваги децентралізованого підходу. На основі цього було розроблено архітектуру вебзастосунку, яка включає користувацьку та адміністративну панелі, серверну частину на Flask, інтеграцію з Web3 та зберігання даних у базі SQLite.

Запропонована система продемонструвала стабільну роботу, високу масштабованість і практичну ефективність. Вона підтвердила свою актуальність у сучасних умовах, де особлива увага приділяється безпеці доступу до даних і прозорості систем управління. Завдяки автоматизації процесів, Web3-технологіям та гнучкості архітектури, рішення може бути легко адаптоване під інші задачі контролю доступу в різних сферах – від освітніх установ до корпоративного сектору..