МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ФАХОВИЙ БІЗНЕС-КОЛЕДЖ
Циклова комісія (кафедра) комп’ютерної інженерії та інформаційних технологій

**КВАЛІФІКАЦІЙНА РОБОТА**

на тему

# ЦИФРОВИЙ ГАМАНЕЦЬ НА БАЗІ BLOCKCHAIN-ТЕХНОЛОГІЙ

Виконав: студент групи 3П-22
Спеціальності
121 Інженерія програмного забезпечення
Вадим КОМИШ
Керівник:
Станіслав МАРЧЕНКО

Черкаси 2025

**ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ БІЗНЕС-КОЛЕДЖ**

Кафедра комп'ютерної інженерії та інформаційних технологій

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Інженерія програмного забезпечення

**ЗАТВЕРДЖУЮ**

Завідувач кафедри КІ та ІТ

_____ Владислав ХОТУНОВ
(підпис)
«_____»_____2024 р.

# З А В Д А Н Н Я

# НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Комишу Вадиму Миколайовичу
(прізвище, ім'я, по батькові студента)

1. Тема випускної роботи Цифровий гаманець на базі blockchain-технологій
Керівник роботи Марченко Станіслав Віталійович, спеціаліст І категорії
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від «07» жовтня 2024 року № 68у.
2. Строк подання студентом випускної роботи 03.06.2024
3. Вихідні дані до випускної роботи мова програмування JavaScript для бекенду, фреймворк React для написання фронтенду, блокчейн-мережа Ethereum, інструменти для взаємодії з блокчейн-мережою (Ethers.js), технологія для зберігання гаманців MetaMask API
4. Зміст випускної роботи (перелік питань, які потрібно розробити) огляд технологій розробки цифрових гаманців (особливості створення цифрових гаманців на різних платформах, базові поняття в контексті блокчейн-технологій, сучасний стан розвитку технологій для управління цифровими активами), проєктування та реалізація цифрового гаманця (аналіз вимог до програмного забезпечення, проєктування програмної системи, програмна реалізація системи), тестування цифрового гаманця (перелік і обґрунтування обраних методів тестування, тестовий план проєкту, перевірка та аналіз результатів виконання тестових сценаріїв).
5. Дата видачі завдання 15.09.2024р.

# КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів кваліфікаційної роботи | Терміни виконання етапів | Примітка про виконання з підписами керівника і студента |
|---|---|---|---|
| 1 | Вступ | 14.10.2024 | |
| 2 | Розділ 1. Огляд технологій розробки цифрових гаманців | 09.12.2024 | |
| 3 | Розділ 2. Проєктування та реалізація цифрового гаманця | 10.03.2025 | |
| 4 | Розділ 3. Тестування цифрового гаманця | 28.04.2025 | |
| 5 | Висновки | 12.05.2025 | |
| 6 | Оформлення кваліфікаційної роботи (чистовий варіант) | 26.05.2025 | |
| 7 | Перевірка кваліфікаційної роботи на наявність ознак плагіату (за 10 днів до захисту) | 02.06.2025 | |
| 8 | Подання кваліфікаційної роботи на затвердження завідувачу кафедри (за 7 днів до захисту) | 10.06.2025 | |

Студент _____ Вадим КОМИШ
(підпис)

Керівник роботи _____ Станіслав МАРЧЕНКО
(підпис)

# АНОТАЦІЯ

Кваліфікаційна робота присвячена аналізу, проєктуванню та реалізації цифрового гаманця на основі Web3-технологій, що забезпечує безпечну взаємодію користувача з блокчейн-мережею. У межах роботи було розроблено програмне рішення – клієнтський застосунок, який дозволяє підключати криптогаманець, переглядати баланс токенів та здійснювати транзакції з використанням мережі Ethereum.

У роботі проведено аналіз функціональних та нефункціональних вимог до Web3-систем, зокрема визначено необхідність забезпечення безпеки, зручності користування та сумісності з популярними інструментами, такими як MetaMask та Infura. Запропоновано архітектуру системи за компонентно-орієнтованим підходом із розмежуванням функцій між сервісами, що відповідають за роботу з токенами, підключенням до гаманця та обробкою транзакцій.

Описано процес реалізації застосунку на основі технологій React, Ethers.js, Web3Modal та styled-components. Застосунок забезпечує стабільний доступ до блокчейн-мережі, підтримує роботу з токеном DAI та може бути розширений для роботи з іншими мережами або функціональністю, зокрема NFT чи стейкінгом.

Результати роботи становлять основу для створення повнофункціонального Web3-гаманця та можуть бути використані у рамках стартапів або децентралізованих застосунків нового покоління.

Ключові слова: WEB3, ЦИФРОВИЙ ГАМАНЕЦЬ, ETHERS.JS, БЛОКЧЕЙН, ETHEREUM, КРИПТОВАЛЮТА, ТРАНЗАКЦІЇ, ТОКЕНИ.

# ABSTRACT

The qualification work is dedicated to the analysis, design, and implementation of a digital wallet based on Web3 technologies, enabling secure user interaction with blockchain networks. The study presents a client-side application that allows users to connect a cryptocurrency wallet, view token balances, and execute transactions on the Ethereum network.

The work includes an analysis of functional and non-functional requirements for Web3 systems, emphasizing security, usability, and compatibility with widely used tools such as MetaMask and Infura. A component-oriented architecture is proposed, with distinct services handling token operations, wallet connectivity, and transaction processing.

The implementation process is described using React, Ethers.js, Web3Modal, and styled-components. The application ensures stable access to blockchain networks, supports interactions with the DAI token, and can be extended to integrate additional functionalities such as NFTs or staking.

The results of this work lay the foundation for developing a fully functional Web3 wallet and can be utilized in startups or next-generation decentralized applications (dApps).



Keywords: WEB3, DIGITAL WALLET, ETHERS.JS, BLOCKCHAIN, ETHEREUM, CRYPTOCURRENCY, TRANSACTIONS, TOKENS.

# ЗМІСТ

# ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ

| | |
|---|---|
| PoW | Proof of Work |
| PoS | Proof of Stake |
| ECDSA | Elliptic Curve Digital Signature Algorithm |
| NFT | Non-Fungible Token |
| DeFi | Decentralized Finance |
| dApp | Decentralized Application |
| ERC-20 | Ethereum Request for Comments 20 |
| SHA-256 | Secure Hash Algorithm 256-bit |
| API | Application Programming Interface |
| JSON-RPC | JavaScript Object Notation Remote Procedure Call |
| AES-256 | Advanced Encryption Standard 256-bit |
| DAO | Decentralized Autonomous Organization |
| BEP-20 | Binance Smart Chain Token Standard |

# ВСТУП

Стрімке поширення та зручність онлайн-платежів сприяють активному розвитку технологій цифрових гаманців. Одним із інноваційних рішень є цифрові гаманці на основі технології блокчейн. Дані гаманці забезпечують безпеку активів та децентралізований спосіб керування ними, що робить їх незамінними в час зростання популярності криптовалюти. Перевага гаманців, розроблених на основі технології блокчейн, – високий рівень безпеки завдяки криптографічному шифруванню та децентралізації. На відміну від стандартних цифрових гаманців, де банки виступають посередником між користувачами, що підвищує ризики несанкціонованого доступу до активів. Гаманці на основі блокчейн-технології усувають ці проблеми, надаючи користувачам можливість повністю контролювати свої активи без втручання третіх осіб. Важливим складником так само залишається легкість і швидкість проведення транзакцій. Завдяки блокчейн-технологіям виконувати транзакції з цифровими гаманцями можна миттєво, у будь-яку точку світу і без суттєвих комісій, що зазвичай супроводжують роботи з банками. Це актуальне питання, яке виникає при міжнародних грошових переказах і при здійсненні фінансових операцій в країнах з нерозвиненою економікою. Звідси, розроблення подібних гаманців є актуальним завданням, оскільки вони відповідають сучасним вимогам зручності та безпеки.

Мета проєкту – створити цифровий гаманець на основі технології блокчейн та мови програмування Python, що забезпечує користувачам високий рівень безпеки на основі децентралізації та неможливістю зміни даних транзакції, прозорості фінансових операцій і простоту управління цифровими активами.

Об’єктом дослідження є цифрові гаманці на основі технології блокчейн та методи їх реалізації. Передбачено розгляд архітектури блокчейн-систем, протоколи обміну даними, механізми гарантування надійності та цілісності транзакцій.

Предметом дослідження виступають технології розробки цифрових гаманців на базі блокчейн-платформ, такі як Ethereum та Bitcoin. У фокусі уваги процеси створення гаманців через генерацію криптографічних ключів, алгоритми верифікації транзакцій (Proof of Work, Proof of Stake), забезпечення децентралізованого управління за допомогою смарт-контрактів, а також протоколи безпеки (асиметричне шифрування, мультипідписи) для захисту приватних ключів і персональних даних користувачів.

Практичні завдання в контексті вказаної тематики та мети дослідження такі:

- розглянути принципи роботи блокчейн-технологій та їх застосування для створення цифрових гаманців;
- порівняти наявні платформ для розробки цифрових гаманців та вибір найбільш ефективної для реалізації проєкту;
- розроблення архітектури цифрового гаманця з урахуванням вимог безпеки, масштабованості та продуктивності;
- реалізувати прототип цифрового гаманця з використанням блокчейн-технологій;
- відтестувати та оцінити якість роботи цифрового гаманця для визначення ефективності та відповідності вимогам користувачів.

# РОЗДІЛ 1
# ОГЛЯД ТЕХНОЛОГІЙ РОЗРОБКИ ЦИФРОВИХ ГАМАНЦІВ

## 1.1 Особливості створення цифрових гаманців на різних платформах

Цифрові гаманці можна створювати на основі різноманітних технологій та платформ, кожна з яких має свої переваги, обмеження та специфіку використання. Серед основних підходів до реалізації сучасних цифрових гаманців можна виокремити використання централізованих серверів, хмарних сервісів, а також децентралізованих технологій, таких як блокчейн.

У централізованих системах усі дані користувача, включаючи його облікові записи та історію транзакцій, зберігаються на серверах компанії-розробника. Такий підхід дозволяє швидко розгортати сервіси та здійснювати управління з боку адміністратора. Проте він несе в собі суттєві ризики, пов’язані з безпекою та приватністю: при атаці на центральний сервер зловмисники можуть отримати доступ до великої кількості конфіденційної інформації.

Інший підхід – використання хмарних сховищ – також передбачає певний рівень централізації, але дозволяє користувачу отримувати доступ до свого гаманця з будь-якого пристрою. Проте й у цьому випадку дані все одно знаходяться під контролем стороннього сервісу.

Натомість блокчейн-технологія забезпечує децентралізовану модель зберігання та обміну даними. У такому разі користувач зберігає приватні ключі лише на власному пристрої, а всі транзакції записуються в розподілену базу даних (блокчейн), яка є доступною для всіх учасників мережі та захищена від змін і підробок. Звідси, користувач отримує повний контроль над своїми активами та особистою інформацією.

Порівняння наявних технологій показує, що блокчейн вирізняється підвищеним рівнем безпеки, прозорістю транзакцій та можливістю автономної роботи без потреби довіряти централізованим посередникам [6]. Саме ці характеристики є критично важливими для цифрових гаманців, особливо в

контексті зберігання криптовалют або цифрових токенів. За результатами огляду джерел [5-7] було обрано технологію блокчейн як основу для програмної реалізації цифрового гаманця.

### 1.2 Базові поняття в контексті блокчейн-технологій

Однією з новітніх технологій для оптимізації бізнес-процесів є блокчейн (blockchain technology) – розподілений одноранговий реєстр, що функціонує на основі механізмів консенсусу. У поєднанні з системою смарт-контрактів та іншими допоміжними інструментами блокчейн забезпечує високий рівень довіри, безпеки та прозорості обробки даних, транзакцій та взаємодії між сторонами без потреби в централізованому контролі.

На відміну від класичних реляційних баз даних, які організовують збереження інформації у вигляді таблиць, блокчейн реалізує структуру з послідовно пов’язаних між собою блоків, що формують ланцюг. Основоположним принципом є децентралізація: дані зберігаються на множині незалежних вузлів (nodes), кожен з яких містить копію повного реєстру. Це виключає необхідність довіри до єдиного центрального суб’єкта, значно підвищуючи стійкість системи до зовнішніх втручань та внутрішніх маніпуляцій.

Кожна транзакція, що відбувається у блокчейн-мережі, проходить перевірку та включається до блоку. Блок – це структура даних, яка містить список підтверджених транзакцій; хеш попереднього блоку; мітку часу (timestamp); випадкове число (nonce) для забезпечення цілісності при майнінгу; власний хеш; додаткові метадані залежно від специфікації мережі (наприклад, Ethereum або Hyperledger).

Зв’язок між блоками реалізується через хеш-посилання, що створює неперервний ланцюг, у якому зміна одного блоку автоматично робить нечинними всі наступні. Це забезпечує незмінність історії транзакцій і сприяє підвищенню надійності.

Перший блок у ланцюзі – генезис-блок (genesis block) – має особливий статус: він створюється вручну або програмно під час ініціалізації мережі та не містить посилання на попередній блок, що робить його унікальним у структурі системи. Хеш-значення – це значення фіксованої довжини, яке генерується за допомогою криптографічного хеш-алгоритму. Таке значення унікальним чином ідентифікує дані, а стан блокчейну обчислюється за допомогою хеш-функції, реалізованої в межах криптографічних алгоритмів, на зразок SHA256. Вони використовуються для забезпечення цілісності та безпеки блокчейн-даних. Початковий хеш блоку обчислюється за допомогою початкових транзакцій. Для обчислення наступних блоків використовуються практично вся інформація з поточного блоку. Приклад сформованого хешу для блоку мережі представлено на рисунку 1.1.

| Data: Blockchain Concepts |
| --- |
| Hash: 06ecd9a034556c403064a9114d26e2d227324520e4c2d5b330cf5f881564ac9b |

Рисунок 1.1 – Хеш блоку blockchain-мережі

Навіть незначна зміна даних призводить до абсолютно нового хеш-значення [6]. Ця властивість хеш функцій забезпечує високий рівень безпеки. Оскільки кожен блок містить хеш попереднього блоку, будь-яка зміна даних у попередньому блоці змінить всі наступні хеші, що дає змогу учасникам мережі швидко виявити спроби маніпуляцій. Таким чином хеш-функція не лише забезпечує безпеку, а також дає можливість працювати в умовах децентралізації, де кожен учасник має копію ланцюга і в будь-який момент може перевірити його цілісність.

Після того, як хеш був створений для блоку, наступним етапом є формування ланцюга. Оскільки кожен блок має в собі як хеш попереднього блоку, так і власний, це дає змогу створювати неперервний ланцюг блоків, що є основою блокчейну. Звідси, система є безпечною та стійкою до маніпуляцій.

Цілісність даних забезпечується за допомогою структури, відомої як дерево Меркла (хеш-дерева), що має структуру бінарного дерева, де хеші даних транзакцій на найнижчому рівні називаються «листовими вузлами», проміжні хеші – «нелистовими вузлами», а хеш у верхній частині – «коренем». Така структура була розроблена для розбиття великих фрагментів даних на менші, що робить можливою перевірку більшої кількості транзакцій (рис.1.2).

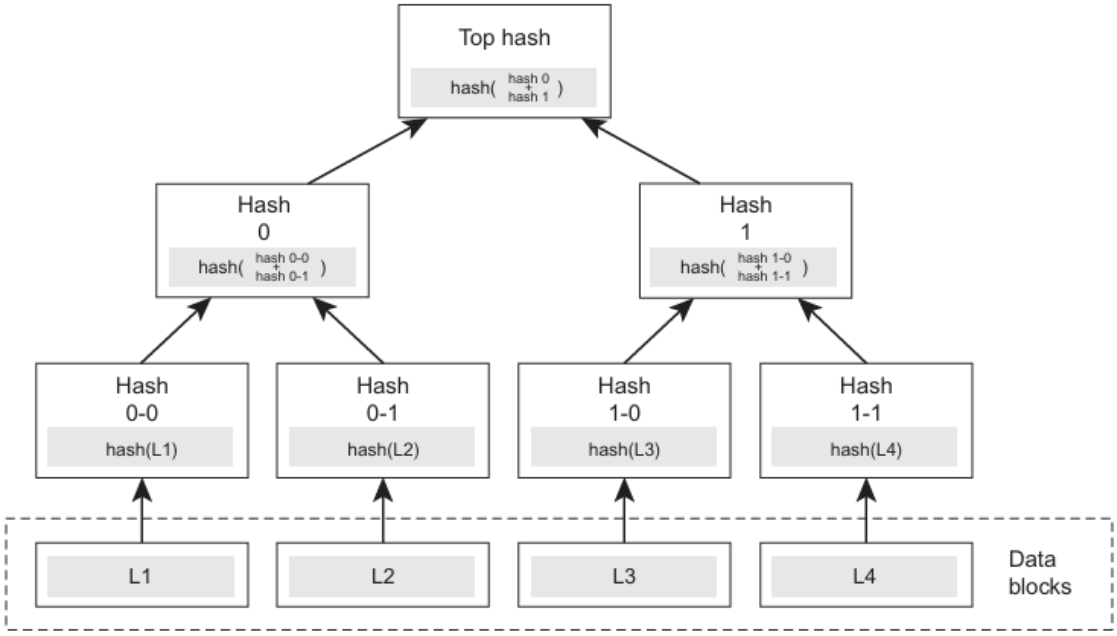


Рисунок 1.2 – Формування хеш-дерева Меркла (Merkle Tree) для забезпечення цілісності даних у блокчейн-структурі [15, с. 5]

Дерево Меркла виявилося важливим елементом блокчейн-технології, оскільки воно забезпечує швидкість і простоту перевірки достовірності [1]. Для кожного блоку за допомогою кореня Меркла генерується унікальне хеш-значення. Цей блок пов’язує один блок з іншим у блокчейні, включаючи хеш попереднього блоку. Щоразу, коли у транзакцію вносяться зміни, змінюється й її хеш. У результаті цієї зміни блок стає недійсним, тому що вона торкається всіх рівнів аж до кореня Меркла, і його значення змінюється. У свою чергу, це призводить до зміни хешу наступного блоку, тому решта блокчейну стає недійсною. Так дерево Меркла веде непорушні записи транзакцій блоку.

Процес формування ланцюга забезпечується тим, що блоки проходять перевірку з боку кожного учасника мережі. Важливим є те, що блоки можуть бути додані лише після того, як хеш пройде перевірку складності та перевірку всіма вузлами. Це забезпечує, що блоки не можуть бути вставлені в ланцюг без перевірки консенсусу.

Оскільки блокчейн являє собою децентралізовану систему, алгоритм консенсусу забезпечує досягання згоди між учасниками мережі про порядок транзакцій та загальний стан блокчейну. Алгоритм консенсусу – це свого роду набір правил, де закладено правила роботи мережі та умови, за яких може бути досягнуто загальної угоди. Кожен блокчейн має власний алгоритм консенсусу. Наприклад, мережа Bitcoin працює у системі PoW, а Tezos – на PoS. Алгоритм консенсусу не лише допомагає визначити, який блок має бути доданим до блокчейну, але й гарантує, що всі учасники мережі матимуть ідентичну версію реєстру. Це забезпечує консистентність і безпеку мережі навіть у випадку, коли учасники намагаються маніпулювати даними.

Алгоритм PoW (Proof of Work) – один із перших та найвідоміших алгоритмів. Для створення нового блоку в мережі, майнерам потрібно математичним способом обчислити хеш блоку. Вся робота виконується спеціальними потужними пристроями, які перебирають тисячі варіантів за секунду. Винагорода та право створити новий блок отримує той, кому вдалося раніше за всіх знайти необхідне значення. Цей алгоритм простий і надійний, проте досить неекологічний та енерговитратний: для швидкого обчислення нового хешу, майнерам необхідно використовувати потужне обладнання, яке споживає величезну кількість електроенергії. Блокчейн-мережами та монетами, які використовують цей алгоритм, є Bitcoin, Litecoin, Dash, Monero, Dogecoin [6].

PoS (Proof of Stake) – екологічна альтернатива PoW. Право створення нового блоку надається валідатору з більшою часткою. Тобто, чим більше в майнера коштів на рахунку, тим вища ймовірність того, що мережа дозволить йому створити новий блок. У мережі з алгоритмом PoS не потрібно конкурувати та здійснювати величезну кількість складних обчислень, а швидкість транзакцій

значно вища. Блокчейн-мережами та монетами, які використовують цей алгоритм, є Tezos, Cardano, Polkadot, Ethereum.

Оскільки механізм консенсусу визначає, хто і за яких умов може додати новий блок до блокчейну, важливим аспектом у цьому процесі є вміст самого блоку. А він безпосередньо залежить від транзакцій, які до нього входять. Саме транзакції є основною одиницею взаємодії в блокчейн-мережі: вони фіксують передачу цифрових активів, зміну стану смарт-контрактів або інші дії, що мають бути зафіксовані у розподіленому реєстрі.

Кожна транзакція в блокчейні має чітко визначену структуру, яка дозволяє мережі її ідентифікувати, обробити та валідувати. У типовій транзакції присутні такі основні складники: адреса відправника, адреса одержувача, сума передачі, комісія за транзакцію (gas fee), підпис відправника, nonce – лічильник транзакцій з певної адреси, що запобігає повторному використанню однієї і тієї ж транзакції, а також додаткові дані, наприклад, виклик функцій смарт-контрактів.

Підпис транзакції є важливим криптографічним складником: він створюється на основі закритого ключа відправника і підтверджує, що саме власник відповідного гаманця ініціював транзакцію. Будь-який вузол мережі може перевірити цей підпис за допомогою відкритого ключа, що забезпечує високий рівень безпеки без потреби довіряти централізованому органу. Власне підпис є результатом застосування криптографічного алгоритму, наприклад ECDSA (Elliptic Curve Digital Signature Algorithm) [1, 6], який забезпечує високий рівень стійкості до підробок. Створений підпис є унікальним не лише для кожного користувача, а й для кожної окремої транзакції, оскільки залежить як від закритого ключа, так і від вмісту транзакції.

Після створення транзакції вона передається в мережу та тимчасово зберігається у мемпулі (mempool) – буфері, де очікує підтвердження. Коли майнер або валідатор створює новий блок, він обирає певну кількість транзакцій з мемпулу (зазвичай ті, що пропонують вищу комісію), перевіряє їхню коректність, підписує блок, додає його до ланцюга і поширює серед інших вузлів.

Якщо вузли підтверджують дійсність транзакцій і блоку загалом, блок вважається доданим.

Отже, структура транзакції є не лише технічним описом об’єкта в мережі, а й основою безпечного і достовірного функціонування всього блокчейну. Валідація транзакцій через криптографічні механізми, алгоритми консенсусу та розподілену перевірку забезпечує незмінність і довіру в децентралізованому середовищі.

Усі вузли виконують перевірку коректності підпису до того, як транзакція потрапляє до блоку. Якщо підпис не проходить цю перевірку, транзакція вважається недійсною і відхиляється мережею. Це гарантує, що лише автентичні транзакції, авторизовані їхніми справжніми власниками, можуть бути оброблені і записані до блокчейну (рис. 1.3).

Криптографічний підпис також відіграє роль захисту від фальсифікації. Навіть якщо зловмисник перехопить дані транзакції, він не зможе змінити її вміст або переадресувати активи без доступу до закритого ключа. Звідси, підпис не тільки виконує функцію підтвердження авторства, а й забезпечує цілісність та незмінність переданих даних у децентралізованій мережі.

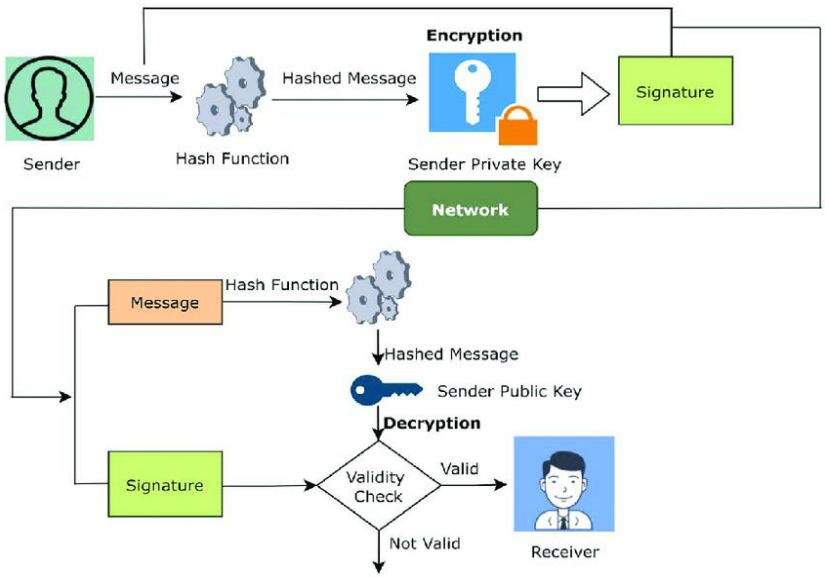


Рисунок 1.3 – Схема використання цифрового підпису в блокчейні [14]

## 1.3 Сучасний стан розвитку технологій для управління цифровими активами

Сучасні блокчейн-мережі відрізняються за способом організації, рівнем доступу, методами досягнення консенсусу та підходами до збереження даних. Основними моделями є публічні (відкриті), приватні та консорціумні (гібридні) мережі [7, 8]. Публічні блокчейни, такі як Ethereum, Bitcoin, Solana, дають змогу будь-якому користувачу приєднатися до мережі, брати участь у перевірці транзакцій і зберігати повну копію реєстру. Вони зазвичай використовують енергозатратні, але децентралізовані алгоритми консенсусу, зокрема Proof of Work або Proof of Stake, які забезпечують високу стійкість до змін та шахрайства.

Приватні блокчейни, навпаки, контролюються окремою організацією або групою суб'єктів. Доступ до участі в таких мережах обмежений, що дозволяє підвищити продуктивність та швидкість обробки транзакцій, але водночас знижує рівень децентралізації. Серед прикладів консорціумних мереж можна згадати Hyperledger Fabric та R3 Corda. Вони є компромісом між публічністю та централізацією: кілька організацій спільно контролюють мережу, що дозволяє досягати консенсусу швидше, зберігаючи при цьому прозорість.

Отже, організація мережі визначає не лише її технічні параметри (швидкість, масштабованість, стійкість до атак), але й можливості інтеграції з додатками для керування цифровими активами, зокрема цифровими гаманцями.

Функціональність цифрових гаманців значно варіюється залежно від платформи, типу гаманця (гарячий чи холодний), а також його цільового призначення. Основними функціями, спільними для більшості гаманців, є: створення та збереження приватних ключів, підпис транзакцій, зберігання криптовалют та токенів, перегляд історії операцій, а також взаємодія з децентралізованими застосунками (dApps).

Браузерні розширення (наприклад, MetaMask) зручні для роботи з dApps і підтримують інтеграцію з вебсайтами через JavaScript API. Мобільні

гаманці (наприклад, Trust Wallet) забезпечують доступність і мобільність, часто доповнюються функцією QR-сканування, обміну токенів, а також підтримкою NFT. Десктопні гаманці (Exodus та ін.) зазвичай мають розширену аналітику та кращу візуалізацію портфеля активів. Апаратні гаманці (наприклад, Ledger, Trezor) надають найвищий рівень безпеки, оскільки зберігають ключі в ізольованому середовищі й не підключені постійно до інтернету.

Важливою відмінністю є також підтримка стандартів токенів (ERC-20, BEP-20, TRC-10 тощо), доступ до кількох блокчейн-мереж, наявність мультиакаунтів, захист паролем або біометрією, а також підтримка функцій відновлення доступу через seed-фразу. Сучасні додатки можуть виступати не лише як гаманці, але й як повноцінні інструменти для DeFi, стейкінгу, обміну токенів, керування NFT та взаємодії з Web3.

Загалом, розмаїття функціональних можливостей дозволяє користувачам обирати інструменти відповідно до індивідуальних потреб: від простого зберігання активів до активної участі в децентралізованих фінансових екосистемах. MetaMask є одним із найпопулярніших криптовалютних гаманців у світі, який забезпечує зручний та безпечний доступ до екосистеми блокчейну Ethereum та сумісних мереж [2, 3]. Він був розроблений компанією Consensys і вперше представлений у 2016 році. Основною метою MetaMask є забезпечення користувачам можливості зберігати, надсилати, отримувати та керувати криптовалютами, а також взаємодіяти з децентралізованими застосунками [2].

MetaMask реалізований у вигляді браузерного розширення для популярних браузерів, таких як Google Chrome, Mozilla Firefox, Edge і Brave, а також має мобільну версію для пристроїв на iOS та Android. Такий мультиплатформний підхід дозволяє користувачам мати доступ до своїх коштів з будь-якого пристрою, що значно підвищує гнучкість та зручність використання.

Однією з ключових особливостей MetaMask є локальне зберігання приватних ключів, тобто всі ключі шифруються та зберігаються безпосередньо на пристрої користувача. Це означає, що тільки користувач має доступ до своїх коштів, і жодна централізована структура не може керувати ними. Для

шифрування використовується протокол AES-256, що забезпечує високий рівень криптографічного захисту. Крім того, підтримується двофакторна автентифікація, а також біометричний захист (відбитки пальців, Face ID) на мобільних пристроях [3].

MetaMask також підтримує інтеграцію з апаратними гаманцями, такими як Ledger та Trezor. Це дозволяє значно знизити ризик компрометації приватних ключів, оскільки підписання транзакцій відбувається на окремому фізичному пристрої, а не у браузері чи на смартфоні.

MetaMask підтримує величезну кількість криптовалют і токенів. Основним фокусом є Ethereum та токени стандарту ERC-20, а також NFT за стандартом ERC-721. Проте гаманець також надає можливість додавання інших мереж вручну, таких як Polygon, Binance Smart Chain (BNB Chain), Avalanche, Arbitrum, Optimism та багато інших. Загалом, MetaMask дозволяє працювати з понад 100 000 токенів у різних блокчейн-мережах.

Для взаємодії з блокчейном MetaMask використовує популярні JavaScript-бібліотеки, такі як ethers.js та web3.js [4]. Транзакції підписуються локально і тільки після цього відправляються в мережу через JSON-RPC API. Такий підхід забезпечує як безпеку, так і швидкість роботи. JSON-RPC дозволяє ефективно передавати дані між гаманцем і блокчейн-вузлом.

Крім того, MetaMask автоматично генерує мнемонічну фразу (seed phrase) під час створення нового гаманця. Цей набір із 12 або 24 слів є єдиним способом відновити доступ до гаманця у разі втрати пристрою або видалення застосунку. Користувач несе повну відповідальність за збереження цієї фрази, оскільки вона є ключем до всіх активів у гаманці.

MetaMask має інтуїтивно зрозумілий інтерфейс, що робить його зручним як для початківців, так і для досвідчених користувачів (рис. 1.4). Усі основні функції – створення гаманця, надсилання та отримання токенів, перегляд історії транзакцій, підключення до dApps – виконуються у декілька кліків. Крім того, застосунок підтримує функцію обміну токенів (Swap) безпосередньо в

інтерфейсі гаманця, обираючи найвигідніший курс через агрегатор децентралізованих бірж.

Станом на 2025 рік MetaMask налічує понад 30 мільйонів активних користувачів по всьому світу, що робить його одним із лідерів серед децентралізованих гаманців. Гаманець активно використовується для участі в DeFi-платформах, купівлі NFT, голосування у DAO та інших блокчейн-активностях.

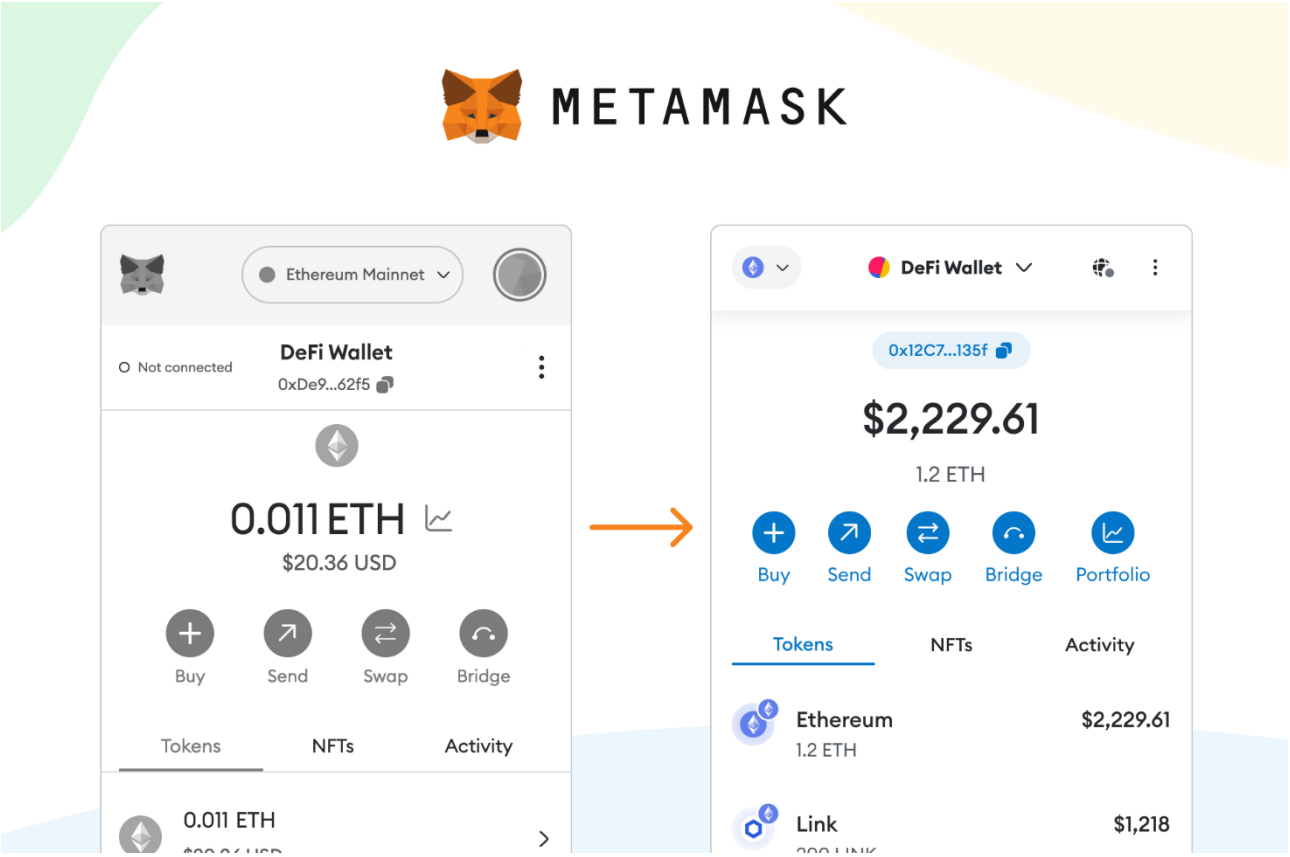


Рисунок 1.4 – Користувацький інтерфейс MetaMask [2]

Альтернативою MetaMask є мобільний децентралізований гаманець Trust Wallet, який також здобув широке визнання серед користувачів Web3-застосунків та криптовалютних платформ [5]. Trust Wallet позиціонується як офіційний гаманець біржі Binance, проте є незалежним non-custodial рішенням з відкритим вихідним кодом, що забезпечує високий рівень автономії та безпеки для кінцевого користувача.

Гаманець підтримує понад 70 блокчейн-мереж, включаючи Ethereum, Binance Smart Chain (BNB Chain), Polygon, Solana, Avalanche, Arbitrum, Optimism, Fantom та інші. Це забезпечує доступ до понад 1 мільйона токенів і NFT, зокрема за стандартами ERC-20, BEP-20, TRC-20, а також NFT-стандартами ERC-721 і BEP-721. Trust Wallet автоматично розпізнає токени в гаманці та надає можливість їх ручного додавання за контрактною адресою.

Платформа Trust Wallet функціонує на мобільних платформах (iOS, Android), а з 2023 року також доступна як браузерне розширення, що розширює його функціональність на десктопах. Усі приватні ключі та операції підписуються локально на пристрої, що виключає доступ до чутливих даних з боку сторонніх сервісів або серверів. Взаємодія з блокчейном реалізується через JSON-RPC протокол з використанням бібліотек web3.js та інших децентралізованих API. При створенні нового гаманця генерується мнемонічна фраза (seed phrase) з 12 слів, що є єдиним засобом відновлення доступу до активів у разі втрати пристрою чи видалення застосунку.

Однією з особливостей Trust Wallet є вбудований dApp-браузер, який дозволяє безпосередньо взаємодіяти з децентралізованими додатками (dApps), такими як Uniswap, PancakeSwap, Aave, OpenSea тощо. Це значно спрощує доступ до DeFi-протоколів, NFT-маркетплейсів та DAO-платформ. У браузерній версії для взаємодії з dApps використовується інжекція Web3-об’єктів, аналогічно до MetaMask.

Крім того, гаманець має інтегровану функцію обміну токенів (Swap), яка працює на основі агрегаторів децентралізованих бірж, обираючи найвигідніші курси обміну. Додатково Trust Wallet підтримує стейкінг криптовалют, що дозволяє користувачам отримувати пасивний дохід без необхідності залучення сторонніх сервісів.

Інтерфейс застосунку інтуїтивно зрозумілий (рис. 1.5), з логічно структурованими розділами, які забезпечують доступ до основних функцій – надсилання та отримання криптовалюти, перегляд транзакцій, підключення до dApps, управління мережами та налаштування безпеки. Станом на 2025 рік Trust

Wallet об’єднує понад 10 мільйонів активних користувачів по всьому світу. Він активно використовується для зберігання криптоактивів, участі в DeFi-екосистемах, торгівлі NFT та голосування у DAO. Завдяки підтримці численних мереж, високому рівню безпеки та мобільній доступності Trust Wallet є однією з провідних альтернатив MetaMask у сфері децентралізованих гаманців.

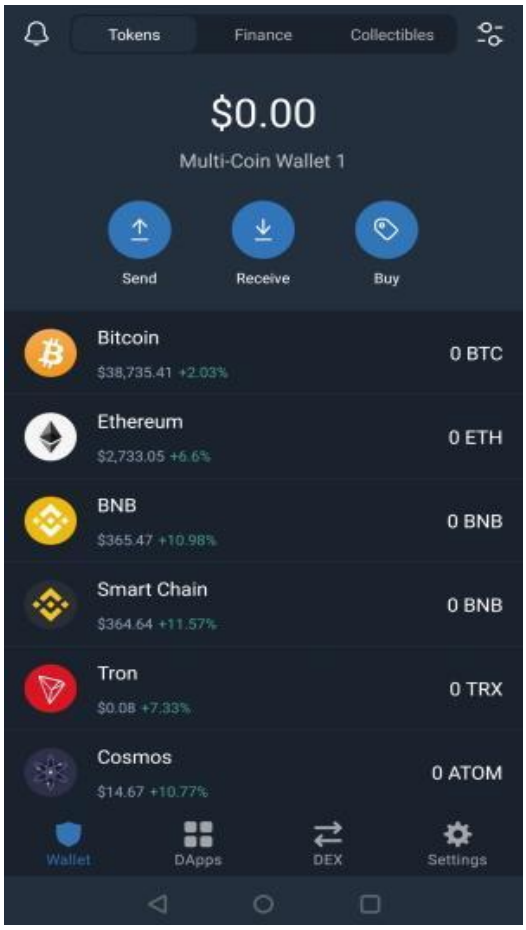


Рисунок 1.5 – Користувацький інтерфейс TrustWallet [3]

### 1.4 Постановка задачі на розробку

Для досягнення поставленої мети дослідження потрібно виконати низку дій. На першому етапі за результатами досліджень загальних принципів роботи блокчейн-технологій та порівняльного аналізу наявних платформ та бібліотек, що використовуються для розробки криптогаманців (зокрема MetaMask, Web3.js, Ethers.js), передбачено конструювання архітектуру цифрового гаманця, враховуючи ключові вимоги до безпеки, надійності, масштабованості та

зручності використання. Наступним кроком є реалізація прототипу вебзастосунку цифрового гаманця, що дає змогу користувачеві підключити власний гаманець, переглядати баланс, історію транзакцій, надсилати перекази та взаємодіяти зі смарт-контрактами. Наприкінці потрібно виконати тестування створеного прототипу, зокрема перевірити його функціональність, стійкість до помилок і відповідність очікуванням користувачів. Результатом виконання цих завдань має стати працюючий вебзастосунок, що демонструє можливості інтеграції з блокчейн-мережею Ethereum та забезпечує базову функціональність цифрового гаманця.

# РОЗДІЛ 2
# ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ЦИФРОВОГО ГАМАНЦЯ

## 2.1 Аналіз вимог до програмного забезпечення

З урахуванням поставленої мети розробки системи цифрового гаманця, насамперед, враховано розмежування вимог на функціональні та нефункціональні. Такий підхід дозволяє чітко сформулювати очікувану поведінку системи, її продуктивні характеристики та заходи захисту, необхідні для гарантування цілісності та конфіденційності транзакційних даних у середовищі з відкритим доступом. Визначення функціональних вимог дозволяє окреслити базовий набір операцій, які повинен підтримувати цифровий гаманець у процесі взаємодії з блокчейн-мережею. До них належать створення гаманця, ініціювання та підписування транзакцій, отримання оновленого балансу, історії операцій, а також інтеграція з децентралізованими сервісами.

Для моделювання функціональної поведінки системи побудовано діаграму прецедентів (рис. 2.1), що відображає всі можливі сценарії взаємодії користувача з Web3-гаманцем. Користувач може підключити гаманець, переглянути баланс токенів, підписати довільну транзакцію, переглянути історію транзакцій, надіслати токени, перевірити дозволи контракту, а також вийти з гаманця. Прецедент «Надіслати токени» включає обов’язковий етап перевірки дозволу (include), а «Вийти з гаманця» є розширенням підключення (extend). Таке формальне моделювання забезпечує повне покриття функціональних вимог, визначених у попередньому підпункті.

Діаграма послідовності (рис. 2.2) ілюструє сценарій надсилання токенів. Користувач активує дію «Send Tokens», після чого App викликає connectWallet() у WalletService для ініціалізації з’єднання з MetaMask. Після отримання об’єкта signer, який дозволяє здійснювати підпис транзакцій, відбувається виклик sendTokens() з модуля TokenService. Цей метод виконує виклик контрактної функції transfer() з об’єктом ethers.Contract, що запускає транзакцію в мережі

Ethereum. Після її підтвердження мережею статус транзакції повертається назад до компонента App, який оновлює інтерфейс.

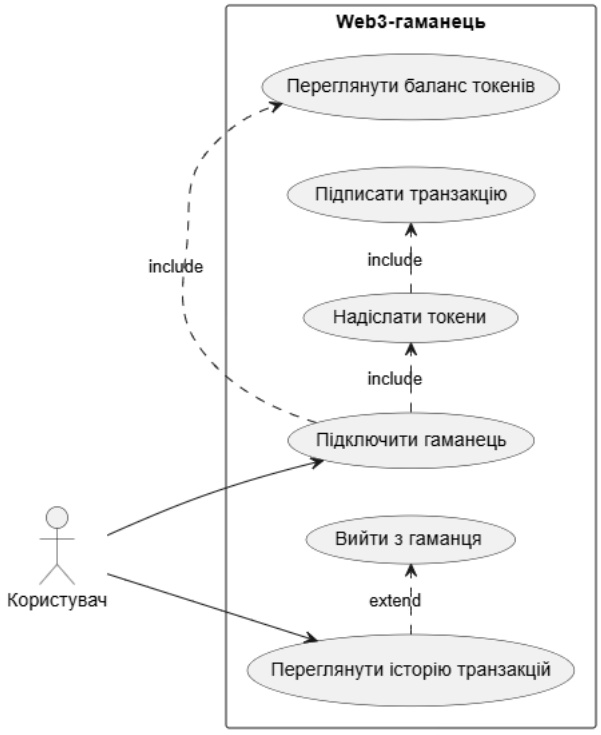


Рисунок 2.1 – Діаграма прецедентів Web3-гаманця

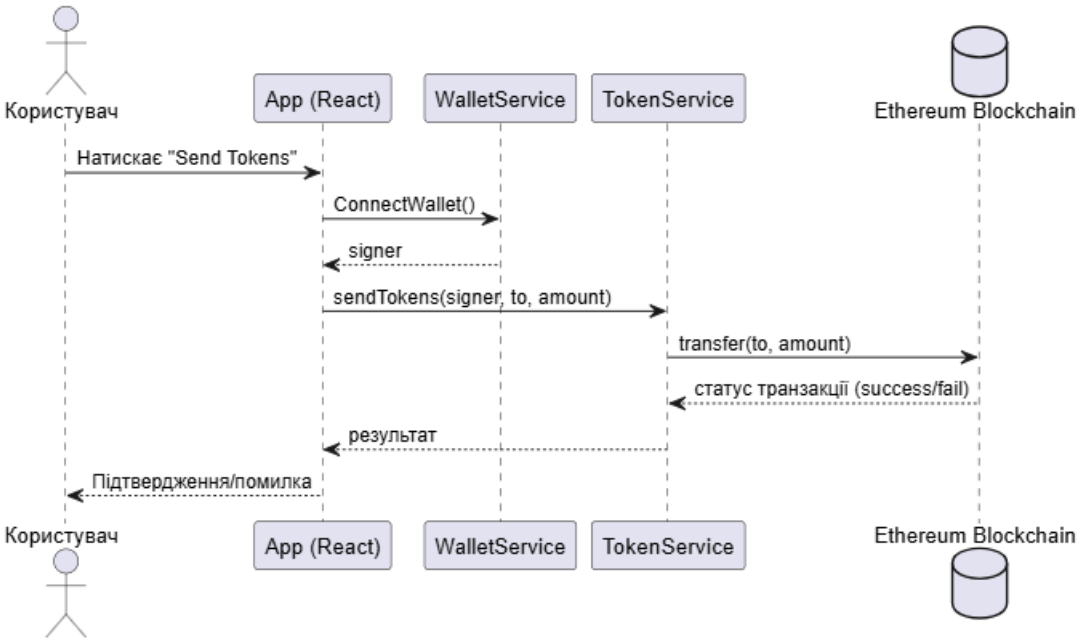


Рисунок 2.2 – Діаграма послідовності надсилання токенів

Таблиця 2.1 демонструє збірний опис основних функціональних вимог до майбутнього програмного продукту. Сформований вичерпний перелік визначає базову функціональність, передбачену для програмної реалізації.

Таблиця 2.1 – Функціональні вимоги до цифрового гаманця

| № | Вимога | Опис |
|---|---|---|
| 1 | Підключення гаманця | Підключення гаманця через MetaMask з отриманням публічної адреси; |
| 2 | Відправлення транзакцій | Формування, підписання та передача транзакцій до блокчейн-мережі |
| 3 | Перегляд історії транзакцій | Виведення переліку підтверджених та очікуваних транзакцій із блокчейну |
| 4 | Підключення до блокчейн-мереж | Встановлення з’єднання з обраною мережею через Infura |
| 5 | Інтеграція зі смарт-контрактами | Можливість виклику методів контракту та підписання транзакцій із payload |

На відміну від функціональних, нефункціональні вимоги фокусуються на якісних характеристиках системи, які впливають на її ефективність, адаптивність, масштабованість та інші параметри користувацького досвіду. Ці вимоги не залежать безпосередньо від поведінки користувача, але суттєво визначають спроможність системи функціонувати в реальному середовищі з відповідним навантаженням і технологічними обмеженнями. Загальні нефункціональні вимоги зібрано в таблиці 2.2.

Таблиця 2.2 – Нефункціональні вимоги до цифрового гаманця

| № | Вимога | Опис |
|---|---|---|
| 1 | Зручність використання | Повідомлення про помилки мають бути чіткими, зрозумілими та інформативними для користувача. |
| 2 | Безпека | Дані, отримані від MetaMask, не мають зберігатися в додатку |
| 3 | Сумісність | Застосунок має коректно працювати у браузерах Google Chrome, Mozilla Firefox та Microsoft Edge. |
| 4 | Адаптивність | Повноцінне відображення функцій на пристроях із різним розширенням екрану |

Окрему увагу приділено категорії вимог до безпеки, яка має критичне значення у контексті цифрових фінансових інструментів. Блокчейн-архітектура передбачає відкритість транзакцій, але для забезпечення конфіденційності, автентичності та захисту від атак необхідне впровадження перевірених криптографічних методів. Основна увага приділяється збереженню приватного ключа, захисту протоколу передачі даних, а також механізмам виявлення та запобігання підробці транзакцій.

Узагальнення зазначених вимог дозволяє сформувати цілісне уявлення про необхідні характеристики системи, на основі яких здійснюється архітектурне проектування. Усі категорії вимог є взаємопов’язаними: наприклад, реалізація функцій підписання транзакцій прямо залежить від надійності збереження ключів, а продуктивність – від обраного способу інтеграції з блокчейн-мережею.

## 2.2 Проєктування програмної системи

Процес проєктування інформаційної системи цифрового гаманця на базі блокчейн-технологій має на меті побудову багаторівневої архітектури, яка забезпечує модульність, безпеку, масштабованість і відповідність функціональним вимогам користувача. Проєктована система реалізує концепцію децентралізованого застосунку (dApp), що передбачає відсутність централізованих серверів для зберігання критичних даних, взаємодію з блокчейн-мережею напряму через клієнтський інтерфейс, а також повну відповідальність користувача за управління ключами.

Архітектурна модель системи побудована за принципами компонентно-орієнтованого підходу з чітким розмежуванням функцій між модулями. Клієнтська частина реалізована на базі React і відповідає за візуалізацію інформації, ініціацію подій взаємодії та обробку введення користувача. У її складі реалізовано основні компоненти застосунку, інтерфейсні елементи для відображення повідомлень про помилки, а також стилі, що визначають зовнішній вигляд інтерфейсу. Бізнес-логіка системи реалізована у вигляді окремих сервісів,

які забезпечують взаємодію з гаманцем, управління токенами та роботу з мережею Ethereum. Ці сервіси обробляють транзакції, керують станом підключення до блокчейн-мережі та інкапсулюють усі технічні деталі взаємодії з інфраструктурою Web3. Конфігураційна частина проекту визначає залежності, середовище розробки та параметри запуску, що забезпечує стабільну роботу системи та її узгоджене розгортання. Така організація дозволяє підтримувати модульність, спрощує масштабування та сприяє ефективному супроводу програмного коду.

На етапі логічного проєктування було сформовано модель компонентів системи, представлену на діаграмі компонентів (рис. 2.3). Основним координуючим компонентом є App, який реалізує взаємодію з користувачем та інкапсулює керування станами (адреса гаманця, токени, підключення, підтвердження транзакцій).

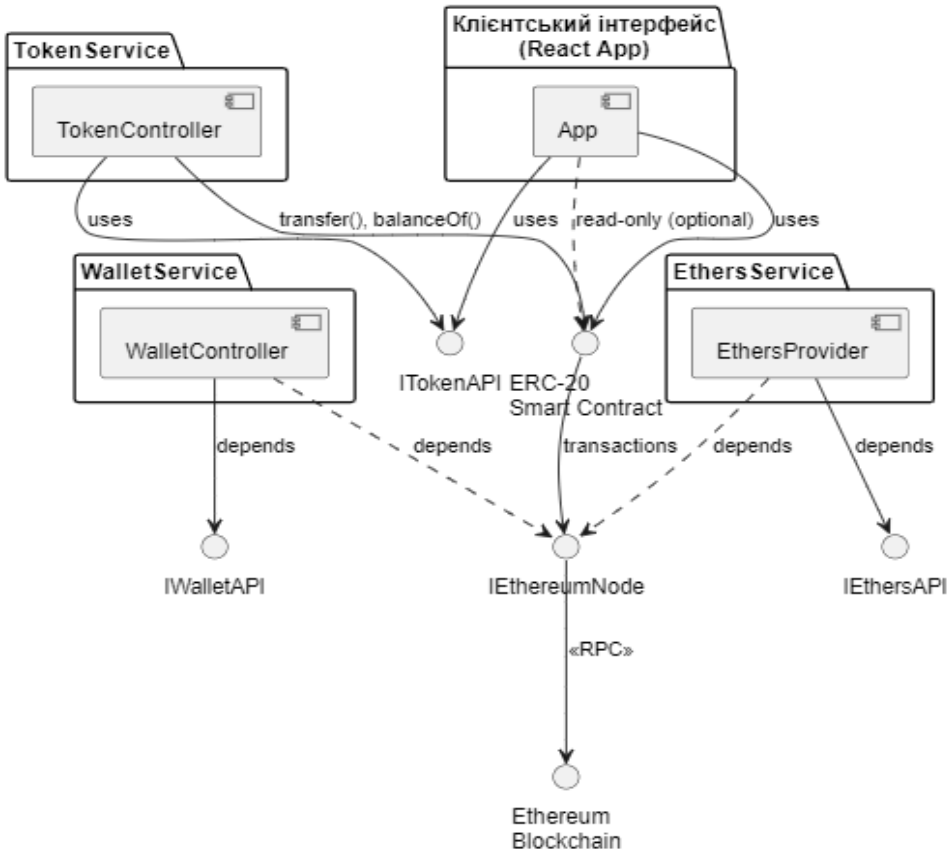


Рисунок 2.3 – Діаграма компонентів застосунку Web3-гаманця

App взаємодіє з WalletService через інтерфейс IWalletAPI для ініціалізації підключення користувача до Ethereum-мережі через Web3Modal. Компонент TokenService реалізує API для взаємодії з контрактами токенів, зокрема виклики transfer(), balanceOf(), approve() та allowance(). Компонент EthersService відповідає за налаштування провайдера Ethereum (через Infura або інші RPC) та генерацію signer-об’єкта для підпису транзакцій.

Для деталізації структури ключових компонентів сформовано діаграму класів (рис. 2.4). Кожен компонент описується через атрибути та методи. Наприклад, App містить змінні стану: walletAddress, ethBalance, tokenBalance, а також методи handleConnect(), handleDisconnect(), sendTransaction() тощо. WalletService інкапсулює роботу з Web3Modal, провайдером ethers.providers.Web3Provider та методами отримання signer. TokenService включає методи ініціалізації контракту (initContract), обчислення балансу (getTokenBalance), надсилання токенів (sendTokens) і перевірки дозволів (checkAllowance). Клас EthersService зосереджений на підключенні до провайдера (getProvider()) та підписі транзакцій (signTransaction()).

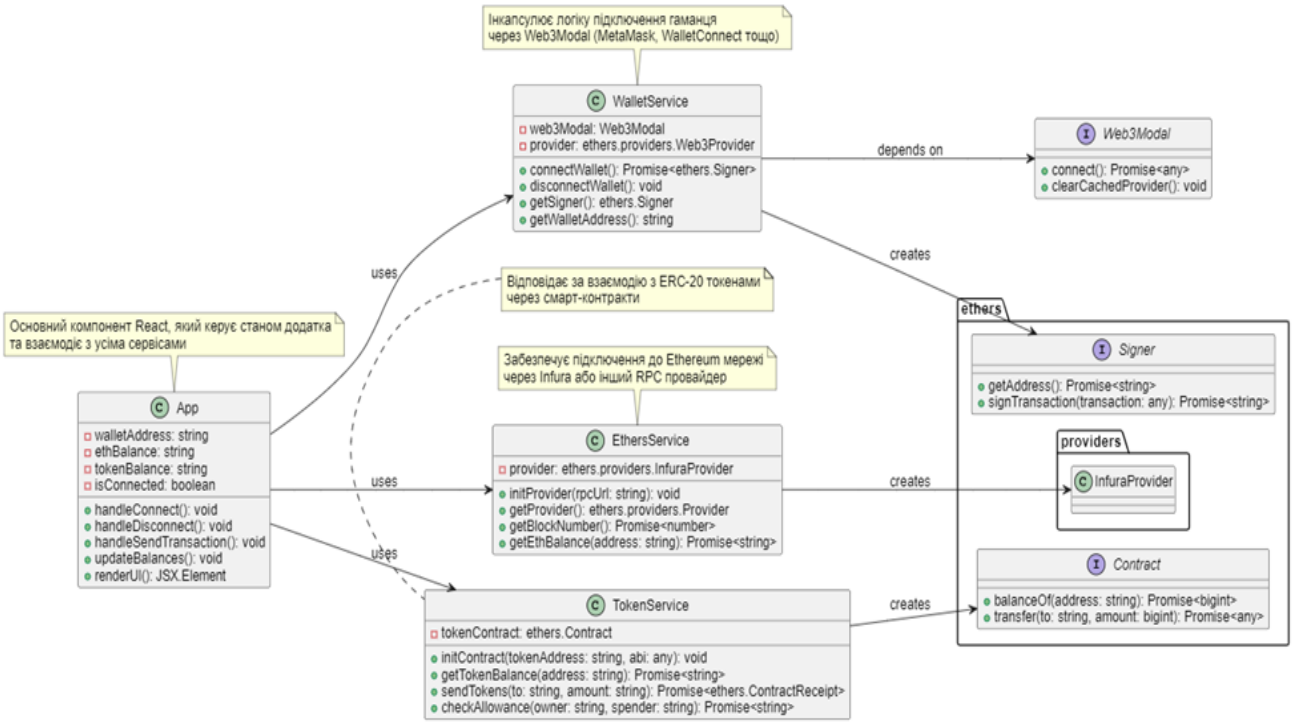


Рисунок 2.4 – Діаграма класів основних компонентів

У результаті проєктування було досягнуто балансу між простотою реалізації та технічною гнучкістю. Завдяки суворому дотриманню принципів SOLID та розділення обов’язків, система залишається розширюваною: до неї легко можна додати підтримку нових токенів, підключення інших блокчейн-мереж (наприклад, Polygon або Binance Smart Chain), або функції з Web3-екосистеми, такі як перегляд NFT чи стейкінг. Архітектурна модель підтримує перевірку транзакцій за допомогою відкритих інтерфейсів (наприклад, через Infura, Alchemy або власний RPC), а ізоляція взаємодії з мережею забезпечує безпечне оброблення приватних ключів на стороні користувача без потреби передачі їх на сервер.

Узагальнено, побудована архітектура системи цифрового гаманця відповідає основним вимогам до децентралізованих застосунків: повний контроль над активами користувача, прозорість транзакцій, інтеграція з відкритими мережами без потреби у централізованому сховищі та гнучке розширення функціоналу. У наступному підпункті буде представлено реалізацію цих архітектурних рішень на рівні програмного коду з використанням React та бібліотеки Ethers.js.

### 2.3 Програмна реалізація системи

Ініціалізація проєкту розпочинається зі створення базового React-застосунку, що виконується за допомогою команди npx create-react-app web3-wallet-web. Після створення директорії проєкту відбувається перехід у неї командою cd web3-wallet-web. Ця операція формує типову структуру проекту, яка включає у себе каталоги public та src, а також ключові файли, такі як index.html, index.js і App.js. Загальна файлова структура проєкту відображена на рис. 2.5. Для початкової перевірки працездатності середовища можна в файлі App.js залишити простий компонент, який виводить заголовок з текстом «Web3 Wallet». Запуск середовища розробки здійснюється командою npm start, яка ініціює сервер розробки за замовчуванням на адресі http://localhost:3000.

Відкривши цю адресу в браузері, користувач побачить вказаний заголовок, що підтверджує успішне створення та запуск базового застосунку.

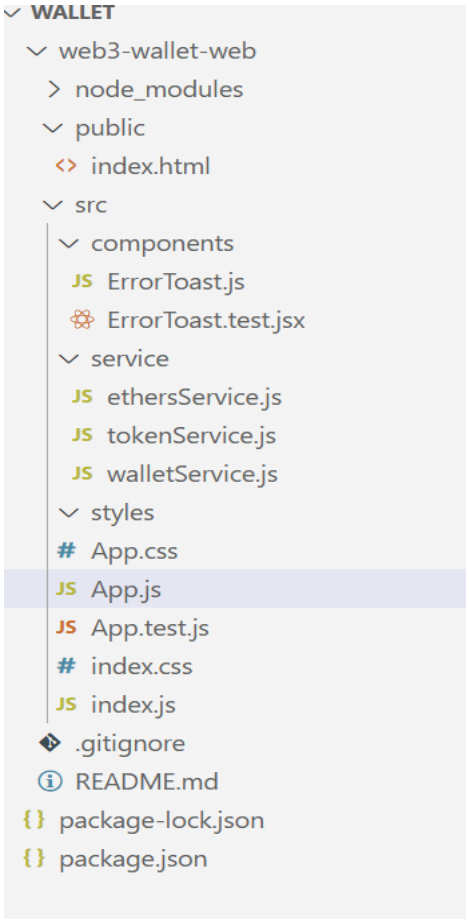


Рисунок 2.5 – Файлова структура програмного проєкта

Наступним кроком стає встановлення додаткових бібліотек, необхідних для реалізації Web3-функціональності та оформлення інтерфейсу. Для цього в терміналі виконується команда npm install ethers web3modal styled-components. Бібліотека ethers надає зручний API для роботи з блокчейном Ethereum, дозволяючи здійснювати підпис транзакцій, викликати смарт-контракти та отримувати інформацію з мережі. Компонент Web3Modal забезпечує універсальний інтерфейс для підключення різних криптогаманців, таких як MetaMask або WalletConnect, до вебзастосунку, спрощуючи взаємодію користувача з блокчейном. Styled-components використовується для оформлення користувацького інтерфейсу, дозволяючи писати CSS-розмітку напряму в коді

мовою JavaScript, що забезпечує гнучкість та інкапсуляцію стилів у компонентах React.

Для кращої організації коду та розмежування логіки застосунок структуровано відповідно до функціонального призначення компонентів, таких як логіка підключення до гаманця, взаємодія зі смарт-контрактами, інтерфейс користувача тощо. У каталозі src створюються підкаталоги, що відповідають за різні аспекти роботи застосунку. Тека components містить автономні UI-компоненти, такі як кнопки, форми чи інші елементи інтерфейсу, які можна багаторазово використовувати у різних частинах програми. У каталозі services розміщено модулі, що реалізують взаємодію з блокчейном за допомогою бібліотеки Ethers.js, зокрема функції для підключення гаманця, отримання балансу токенів і відправлення транзакцій. Папка styles призначена для зберігання стилів, зокрема як CSS-файлів, так і тем, створених із використанням styled-components, що допомагає підтримувати візуальне оформлення застосунку у впорядкованому вигляді.

Файл index.js виконує роль точки входу в застосунок, де здійснюється підключення бібліотеки React та відображення головного компонента App у DOM-дереві браузера. Далі основна бізнес-логіка та UI-компоненти розподілені між відповідними файлами й каталогами, що забезпечує зручність підтримки, масштабування та розвитку проєкту.

Для організації взаємодії з мережею Ethereum у вебзастосунку використовується сервіс Infura, який виступає як проміжна інфраструктура, що дозволяє клієнту безпосередньо підключатися до повноцінних вузлів Ethereum, уникаючи потреби запускати власний повний вузол – спеціальну програму, яка зберігає повну історію блоків і самостійно перевіряє транзакції. Завдяки Infura застосунок може швидко та зручно взаємодіяти з мережею Ethereum (наприклад, надсилати транзакції або отримувати дані), не витрачаючи ресурси на обслуговування власного вузла. Це суттєво спрощує інфраструктуру проєкту, знижує затрати на підтримку вузла і дозволяє швидко масштабувати додаток.

У проєкті створюється файл src/services/ethersService.js, в якому за допомогою бібліотеки ethers.js налаштовується провайдер, що звертається до мережі Ethereum через Infura. Конструкція ethers.providers.InfuraProvider приймає назву мережі. У цьому випадку ‘mainnet’ – це основна публічна мережа Ethereum, а також унікальний API-ключ, що ідентифікує ваш проєкт на сервісі Infura. Каркасний код для підключення до сервісу Infura наведено в лістингу 2.1.

Лістинг 2.1 – Підключення до сервісу Infura

```
import { InfuraProvider } from 'ethers';

const infuraProjectId = 'a5625ed45e074cfd95080181c221d038';
const provider = new InfuraProvider('mainnet', infuraProjectId);

// Function to get the latest block number
export async function getLatestBlock() {
  try {
    const blockNumber = await provider.getBlockNumber();
    console.log('Latest Block Number:', blockNumber);
    return blockNumber;
  } catch (error) {
    console.error('Error fetching block number:', error);
  }
}
```

Цей модуль забезпечує централізовану точку доступу до блокчейн-мережі, яку можуть використовувати різні частини застосунку для читання стану мережі, отримання інформації про блоки або стан смарт-контрактів.

Функція getLatestBlock() виконує запит до мережі для отримання поточного номера останнього блоку. Вона написана з використанням async/await і містить обробку помилок, що гарантує надійність при роботі в умовах нестабільного мережевого з’єднання або інших технічних проблем. Вивід у консоль допомагає розробнику контролювати роботу провайдера в ході розроблення.

Для перевірки працездатності налаштування Infura та провайдера у головному компоненті React (App.js) імпортується функція getLatestBlock() і викликається один раз під час монтування компонента через хук useEffect. Це

дозволяє автоматично перевірити з'єднання при старті застосунку. Відповідний уривок коду наведено в лістингу 2.2

Лістинг 2.2 – Перевірка працездатності

```
import React, { useEffect } from 'react';
import { getLatestBlock } from './services/ethersService';

function App() {
  useEffect(() => {
    getLatestBlock();
  }, []);

  return (
    <div>
      <h1>Web3 Wallet</h1>
    </div>
  );
}
export default App;
```

Вивід номера блоку у консолі браузера підтверджує успішне підключення і коректну роботу провайдера. Це є важливим кроком, який дозволяє розробнику переконатися в тому, що зв'язок із мережею встановлено і можна переходити до більш складної логіки взаємодії з користувацьким гаманцем і смарт-контрактами.

Наступним ключовим кроком є реалізація підключення криптовалютних гаманців безпосередньо в користувацькому інтерфейсі. Для цього застосовується бібліотека Web3Modal, яка забезпечує зручний інтерфейс для вибору і підключення різних типів гаманців, зокрема MetaMask, WalletConnect та інших.

У файлі src/services/walletService.js формується логіка для ініціалізації Web3Modal і управління сесією підключення гаманця. Спершу створюється інстанс Web3Modal з налаштуваннями провайдерів, які можуть бути розширені для підтримки різних варіантів підключення. Ключовими елементами ініціалізації є дві функції: connectWallet() та disconnectWallet(). Перша з них відкриває модальне вікно для вибору користувачем свого гаманця. Після вибору створюється об'єкт провайдера, який обгортається у ethers.providers.Web3Provider для сумісності з API бібліотеки ethers.js. З

отриманого провайдера отримується об’єкт signer, який дозволяє підписувати транзакції від імені користувача, не розкриваючи приватний ключ. Адреса гаманця виводиться у консоль для підтвердження успішного підключення.

Функція disconnectWallet() призначена для відключення користувача від гаманця, очищення кешу Web3Modal та скидання локальних змінних. Це важливо для безпеки і можливості повторного підключення або зміни гаманця.

Отже, описані сервісні модулі формують базову реалізацію Web3-застосунку: централізований доступ до блокчейну через Infura і гнучкий інтерфейс підключення гаманців через Web3Modal. Користувач може підключити гаманець, після чого на сторінці з’явиться його адреса та кнопка для відключення. При натисканні на кнопку «Підключити гаманець» викликається функція handleConnect, яка використовує сервіс connectWallet для встановлення з’єднання з гаманцем (рис. 2.6). Успішне підключення дає змогу отримати об’єкт signer, з якого за допомогою методу getAddress() витягується Ethereum-адреса користувача, яка зберігається у стані. Якщо адреса вже збережена (тобто гаманець підключено), тоді на сторінці з’являється ця адреса та кнопка для її відключення. Натиснувши «Відключити гаманець», користувач запускає функцію handleDisconnect, яка очищує стан, встановлюючи walletAddress в null. Таким чином, забезпечується просте управління станом підключення Web3-гаманця на основі логіки підключення і відключення. Відповідні уривки коду наведені в лістингу 2.3.

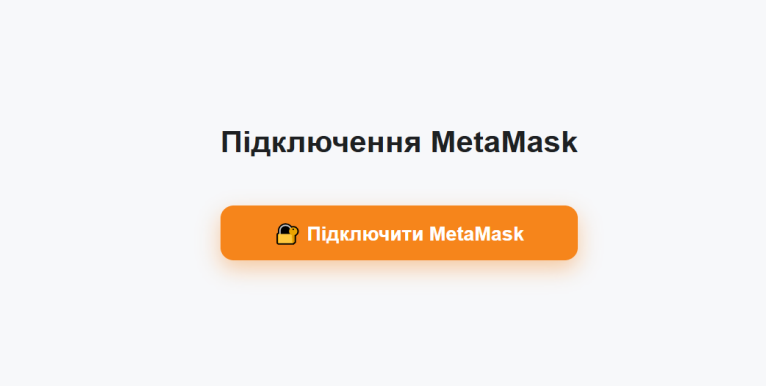


Рисунок 2.6 – Інтерфейс підключення цифрового гаманця

Лістинг 2.3 – Реалізація підключення та відключення гаманця в компоненті App.js

```
export async function connectWallet() {
  try {
    externalProvider = await web3Modal.connect();
    const ethersProvider = new BrowserProvider(externalProvider);
    signer = await ethersProvider.getSigner();
    const address = await signer.getAddress();
    console.log('Wallet connected:', address);
    return { address, signer, ethersProvider };
  } catch (error) {
    console.error(' Failed to connect wallet:', error);
    throw error;
  }
}

export async function disconnectWallet() {
  if (externalProvider?.disconnect) {
    await externalProvider.disconnect();
  }
  await web3Modal.clearCachedProvider();
  externalProvider = null;
  signer = null;
  console.log('Wallet disconnected');
}
```

У додатку також реалізовано можливість перевірки балансу токена DAI та його переказу на іншу адресу. Ці функції описано у файлі tokenService.js, де використовується бібліотека Ethers.js для взаємодії зі смарт-контрактом Ethereum токена DAI (лістинг 2.4). Контракт створюється за допомогою ethers.Contract, де як параметри передається адреса DAI-контракту в основній мережі Ethereum, скорочене ABI з методами balanceOf() і transfer(), а також провайдер. Метод getTokenBalance() викликає balanceOf() і повертає баланс у форматі десяткового числа, зручного для читання. Метод sendTokens() призначений для надсилання токенів і використовує signer, щоб авторизувати й підписати транзакцію. Спочатку контракт підключається до signer, потім викликається метод transfer, якому передається адреса одержувача і сума в токенах, перетворена з десяткового формату в формат wei. Після відправки виконується tx.wait() – очікування підтвердження транзакції в мережі.

Лістинг 2.4 – Реалізація сервісу взаємодії з токеном DAI за допомогою бібліотеки Ethers.js

```
import { ethers } from 'ethers';
import provider from './ethersService';

const DAI_ABI = [
  "function balanceOf(address owner) view returns (uint256)",
  "function transfer(address to, uint256 amount) returns (bool)",
];

const DAI_ADDRESS = '0x6b175474e89094c44da98b954eedeac495271d0f';

const daiContract = new ethers.Contract(DAI_ADDRESS, DAI_ABI, provider);

export async function getTokenBalance(address) {
  try {
    const balance = await daiContract.balanceOf(address);
    return ethers.utils.formatUnits(balance, 18);
  } catch (error) {
    console.error('Помилка отримання балансу токена:', error);
  }
}

export async function sendTokens(signer, to, amount) {
  try {

    const daiWithSigner = daiContract.connect(signer);
              const     tx     =     await     daiWithSigner.transfer(to,
ethers.utils.parseUnits(amount, 18));
    await tx.wait();
    console.log('Транзакція успішна:', tx);
    return tx;
  } catch (error) {
    console.error('Помилка відправки токенів:', error);
  }
}
```

У компоненті App.js розширено стан за допомогою React-хука useState, додавши змінні balance, receiver та amount, щоб зберігати поточний баланс DAI, адресу отримувача та кількість токенів для переказу. Нова асинхронна функція fetchBalance() приймає адресу гаманця, викликає сервіс getTokenBalance і відразу оновлює локальний стан балансом, що дозволяє відображати актуальні дані відразу після підключення. Функція handleConnect() запускає процес підключення через connectWallet, отримує об'єкт signer, визначає його адресу за допомогою getAddress() і зберігає її в walletAddress, після чого викликає fetchBalance() для автоматичного завантаження балансу. При надсиланні токенів користувач вводить у відповідні поля адресу отримувача та суму, які зв'язуються з полями введення через властивості value та обробник onChange. Під час

виклику handleSendTokens() перевіряється наявність повних даних (адреси гаманця, адреси отримувача та суми), тільки після цього знову відбувається підключення до гаманця, отримання поточного signer та виклик sendTokens(), який ініціює транзакцію переказу токенів. Після підтвердження транзакції виконується повторний виклик fetchBalance, що гарантує синхронізацію інтерфейсу з фактичним станом гаманця в мережі. Функція handleDisconnect() повертає застосунок у початковий стан, знищуючи збережену адресу та скидаючи баланс до нуля. Відповідно до стандарту ERC-20, запити до смарт-контракту для отримання балансу та виконання переказу реалізовані у сервісі tokenService, а інтерфейс динамічно відображає блок введення даних та кнопки для надсилання або відключення гаманця залежно від того, чи підключено користувача до мережі (рис. 2.7).

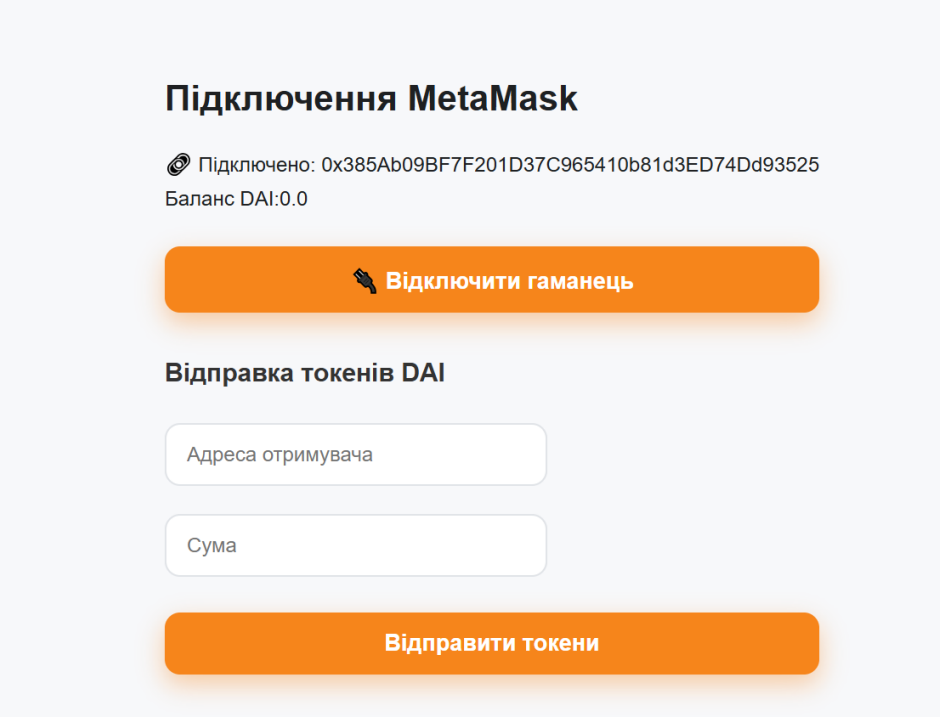


Рисунок 2.7 – Інтерфейс цифрового гаманця

Отже, користувач отримує можливість підключити гаманець, переглянути поточний баланс DAI і здійснити переказ токенів через інтерфейс React. Ця функціональність базується на стандарті ERC-20, який визначає мінімальний інтерфейс для токенів у мережі Ethereum. Метод balanceOf() дозволяє дізнатись

кількість токенів, що належать певній адресі, а transfer використовується для відправлення токенів з одного рахунку на інший. Обидві ці функції реалізовані відповідно до стандарту і працюють через контракт DAI, адреса якого вже вказана у коді. У результаті, цей сервіс дозволяє легко інтегрувати функціональність взаємодії з DAI в будь-який React-компонент, що працює з Web3.

Для побудови інтерфейсу користувача Web3-застосунку в React використовується бібліотека styled-components, яка дозволяє створювати стилі безпосередньо в JavaScript-коді. Такий підхід забезпечує кращу модульність та ізоляцію стилів, оскільки кожен компонент має свої власні CSS-правила, які не конфліктують з іншими частинами застосунку. Стилізовані компоненти оголошуються як звичайні JavaScript-змінні, де вказується HTML-елемент і задаються стилі за допомогою шаблонних рядків. Після цього стилізовані елементи можна використовувати так само, як і звичайні React-компоненти, без необхідності створення окремих CSS-файлів. Наприклад, можна створити окремі компоненти для контейнера, кнопки, заголовка або інформаційного повідомлення, кожен з яких має власне оформлення, відповідне до загальної теми інтерфейсу. Це дозволяє зробити зовнішній вигляд Web3-гаманця сучасним, адаптивним і зручним для користувача. Крім того, styled-components підтримує динамічні стилі, зміни тем, анімації та інші сучасні можливості, що дає змогу створювати повноцінні інтерфейси без потреби у додаткових CSS-інструментах або препроцесорах.

За підсумками другого розділу кваліфікаційної роботи було здійснено комплексний аналіз вимог до програмного забезпечення цифрового гаманця, що дало змогу чітко визначити як функціональні, так і нефункціональні характеристики системи. Зокрема, було встановлено, що система повинна забезпечувати базові функції взаємодії з блокчейном, включаючи підключення гаманця, перегляд балансу токенів, здійснення транзакцій та перегляд історії операцій, при цьому дотримуючись стандартів безпеки, зручності використання та сумісності з популярними Web3-інструментами. На основі результатів аналізу

було спроєктовано архітектуру застосунку за принципами компонентно-орієнтованого підходу, що забезпечує чітке розмежування функціональних блоків, таких як App, WalletService, TokenService та EthersService. Реалізовано повноцінну функціональність цифрового гаманця, який дозволяє здійснювати переказ токенів DAI, переглядати стан балансу, а також взаємодіяти з блокчейном у зручному інтерфейсі. Отримані результати підтверджують, що розроблена система відповідає сучасним вимогам до Web3-застосунків.

# РОЗДІЛ 3
# ТЕСТУВАННЯ ЦИФРОВОГО ГАМАНЦЯ

## 3.1 Перелік і обґрунтування обраних методів тестування

Підхід до тестування Web3-гаманця, реалізованого з використанням бібліотеки Ethers.js, було адаптовано до особливостей децентралізованих додатків (dApps) та обмежено локальним середовищем тестування. Застосунок взаємодіє з Ethereum-мережею через зовнішній гаманець, а для забезпечення безпечного й контрольованого тестування було використано Ganache – локальний емулятор Ethereum-блокчейну, який дозволяє створювати тестову мережу, здійснювати миттєву обробку блоків та використовувати попередньо згенеровані облікові записи.

Зосереджено увагу на перевірці типових сценаріїв користувацької взаємодії: ініціалізація з’єднання з мережею, підключення гаманця, отримання адреси, визначення балансу токена DAI (стандарт ERC-20), ініціювання трансакцій. Усі перевірки здійснювались у межах середовища Ganache, що забезпечило імітацію умов взаємодії з реальним блокчейном без потреби у використанні мережі Ethereum mainnet чи публічних тестнетів.

Для реалізації модульних тестів було використано Jest – сучасний інструмент для модульного тестування JavaScript-застосунків, який підтримує асинхронну логіку, мокінг зовнішніх залежностей та перевірку очікуваних результатів. Зокрема, тестувалась поведінка функцій, що відповідають за отримання балансу гаманця, побудову трансакцій та взаємодію з контрактом токена. Крім того, значну увагу приділено перевірці реакції системи на потенційні помилки: відсутність підключення, некоректну адресу, помилки JSON-RPC або несправність контракту. У межах таких перевірок оцінювалось, чи отримує користувач адекватні повідомлення та чи фіксуються критичні помилки в логах застосунку.

На практиці було відтестовано не лише логіку обробки станів гаманця, а й інтерфейсну поведінку: зокрема, правильність рендерингу кнопок, відображення адреси, оновлення балансу після відправки токенів, а також повідомлення про помилки в окремому компоненті інтерфейсу. Усі тести пройдені успішно, що засвідчує стабільну роботу основної функціональності як у штатних, так і в нестандартних умовах.

Звідси, обрана стратегія передбачає тестування ключової функціональності гаманця в умовах ізольованого середовища, максимально наближеного до реальної взаємодії з Ethereum-мережею. Такий підхід дозволив забезпечити належний рівень достовірності перевірок, підтвердити працездатність критичних компонентів застосунку та перевірити коректність обробки як успішних, так і помилкових сценаріїв, що мають місце при роботі з блокчейн-інфраструктурою.

### 3.2 Тестовий план проєкту

Тестовий план охоплює базову та критично важливу функціональність цифрового Web3-гаманця, реалізованого за допомогою бібліотеки Ethers.js і фреймворку React. Мета тестування полягає в забезпеченні стабільної роботи застосунку при взаємодії з мережею Ethereum через локальне середовище Ganache. Перевірка функціональності виконувалася як у вигляді інтеграційного тестування взаємодії з блокчейном, так і через модульні та інтерфейсні перевірки компонентів React-застосунку за допомогою Jest.

Передбачене функціональне тестування цифрового гаманця здійснюватиметься за допомогою емулятора блокчейн-мережі Ganache, що надаватиме змогу змоделювати роботу Web3-застосунку в контрольованому середовищі з повною історією транзакцій, блоків та адрес. У ході виконання тестових сценаріїв заплановано здійснення підключення гаманця, перевірку балансу, надсилання транзакцій та взаємодію з контрактами, що даватиме можливість охопити основні аспекти функціональності застосунку. Основні

розглянуті сценарії для функціонального тестування засобами інструменту Ganache зібрано в таблиці 3.1.

Таблиця 3.1 – Тестові сценарії з використанням інструменту Ganache

| № | Сценарій тестування | Вхідні дані | Очікуваний результат | Фактичний результат |
|---|---|---|---|---|
| 1 | Виклик функції смарт-контракту (Contract Call) | Введення адреси контракту, виклик функції `balanceOf` через ethers.js | Транзакція успішна, згенеровано TX HASH, функція повертає значення | У Ganache з'являється TX з типом `Contract Call`, відображено хеш транзакції |
| 2 | Перевірка запису транзакції в блокчейн | Виклик функції, очікування підтвердження | Транзакція записана в новий блок, має статус «confirmed» | У Ganache видно підтверджену транзакцію в `Transactions`, відображено gas used |
| 3 | Повторні виклики функцій контракту | Послідовні виклики `balanceOf`, `transfer`, `symbol`, тощо | Кожна транзакція формує окремий запис, має унікальний TX HASH | Всі транзакції зафіксовані, видно послідовні `Contract Call` із різними хешами |
| 4 | Статистика виконання транзакції | Аналіз gasUsed, часу підтвердження | Значення gasUsed відповідає функціям без змін стану контракту (до 22k) | Всі виклики мають `gasUsed = 21632`, що свідчить про читання або легку логіку |
| 5 | Перевірка контрактної адреси | Порівняння адреси контракту у коді та Ganache | Адреса збігається з тією, на яку надсилаються виклики | Адреса контракту однакова в усіх запитах: `0x6B17...271d0F` |

Варто зазначити, що використання Ganache слід розглядати в контексті, ближчому до інтеграційного тестування, оскільки емулятор формує імітацію зовнішнього компонента – блокчейн-мережі. У той же час, потреби в модульному тестуванні в межах кваліфікаційної роботи покриваються засобами фреймворку Jest. Відповідні тестові сценарії для компонентів React показано в таблиці 3.2.

Таблиця 3.2 – Тестові сценарії компонентів React (Jest)

| № | Сценарій тестування | Вхідні дані | Очікуваний результат | Фактичний результат |
|---|---|---|---|---|
| 1 | Відображення кнопки підключення | Відсутній підключений гаманець | Кнопка `Connect Wallet` доступна | Кнопка рендериться при ініціалізації |
| 2 | Підключення гаманця | Натискання кнопки при активному MetaMask | Відображення адреси та балансу | Адреса та баланс з’являються |
| 3 | Помилка при підключенні | Відключений MetaMask або інша помилка | Відображення повідомлення про помилку | Повідомлення з’являється у UI |
| 4 | Відключення гаманця | Натискання кнопки `Disconnect` | Очищення адреси та балансу | Адреса та баланс очищені |
| 5 | Відсутність гаманця при відправці токенів | Натискання `Send` без підключення | Показ помилки «Гаманець не підключено» | Повідомлення виведено |
| 6 | Відображення повідомлення про недостатній баланс | Імітація помилки `insufficient funds` | Компонент ErrorToast виводить відповідне повідомлення | Повідомлення відповідає очікуваному |
| 7 | Відображення повідомлення про скасування користувачем | Імітація скасування транзакції | Вивід тексту «Транзакцію скасовано користувачем» | Повідомлення присутнє |
| 8 | Закриття повідомлення | Клік по кнопці `×` у повідомленні | Виклик обробника події `onClose` | onClose викликається успішно |

## 3.3 Перевірка та аналіз результатів виконання тестових сценаріїв

Відповідно до першого сценарію функціонального тестування, в середовищі з попередньо підключеним MetaMask у Ganache було ініційовано процес з’єднання з додатком. Після натискання кнопки “Підключити гаманець” в інтерфейсі додатку відображалася адреса підключеного гаманця. Це свідчило про правильну інтеграцію Web3-провайдера, а також підтверджувало, що асинхронна обробка відповіді та зміна стану React-компонента реалізовані

коректно. Усі дані збігалися з інформацією, що зберігалась у Ganache, зокрема адреса гаманця відповідала даним середовища.

Під час перевірки сценарію з надсиланням транзакції користувач вводив адресу отримувача та суму в ETH, після чого ініціював надсилання. Застосунок формував та підписував транзакцію за допомогою бібліотеки Ethers.js, після чого її було підтверджено через MetaMask. У Ganache одразу фіксувалася поява нової транзакції з відповідним хешем, що підтверджувало її успішне завершення. У вкладці Transaction History відображалися всі ключові параметри: відправник, отримувач, сума переказу та плата за gas fee (рис. 3.1). Баланс гаманця в інтерфейсі застосунку оновлювався після виконання запиту до провайдера. Значення змінювалося з урахуванням суми транзакції та комісії. Реакція UI була без аномалій або затримок, що вказувало на стабільну обробку станів після транзакції.

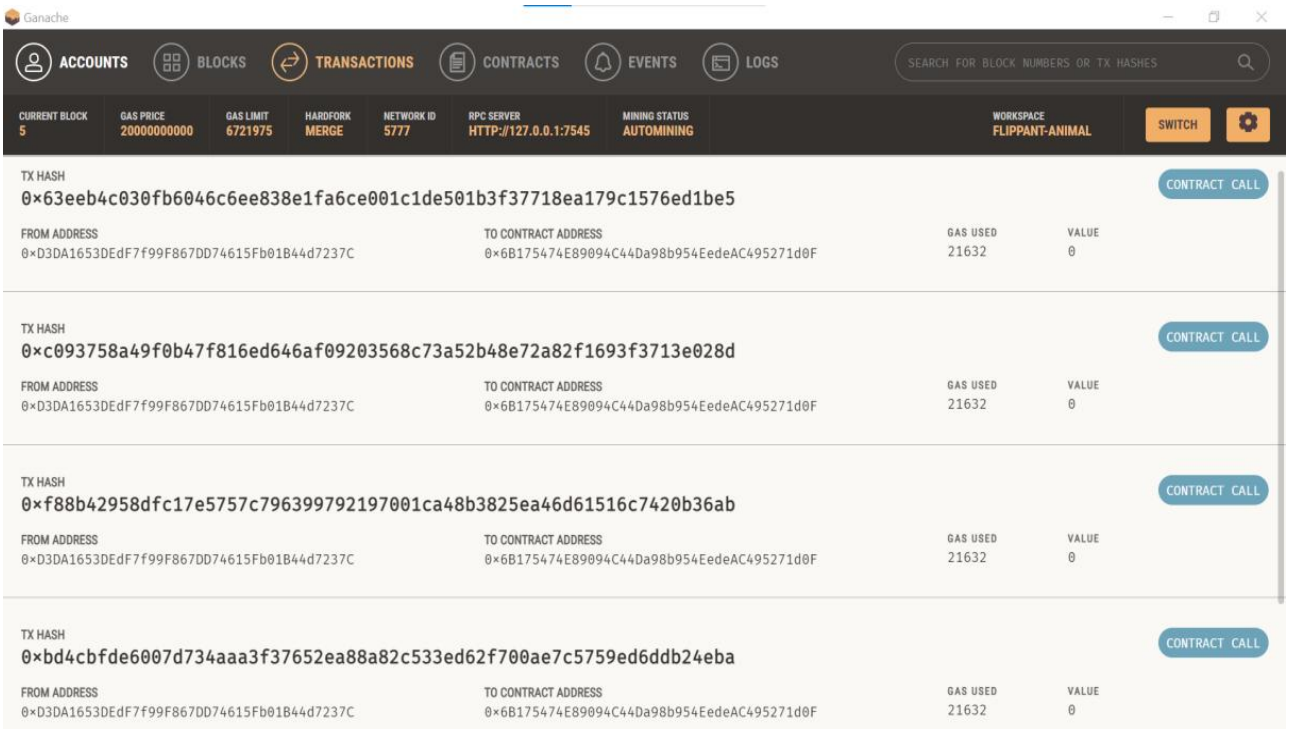


Рисунок 3.1 – Вкладка Transactions після надсилання транзакцій

Також було проаналізовано реакцію застосунку на взаємодію з блоками. Кожна транзакція спричиняла появу нового блоку у Ganache (рис. 3.2). У вкладці

Blocks зберігалася інформація про всі транзакції в межах блоку, їх кількість, час створення та хеш блоку. Це дозволяло верифікувати як саму транзакцію, так і те, що блокчейнова логіка (формування ланцюга блоків) була дотримана у рамках тестового середовища. Дані про блоки підтверджували завершення транзакції та відповідність архітектури очікуваній моделі.

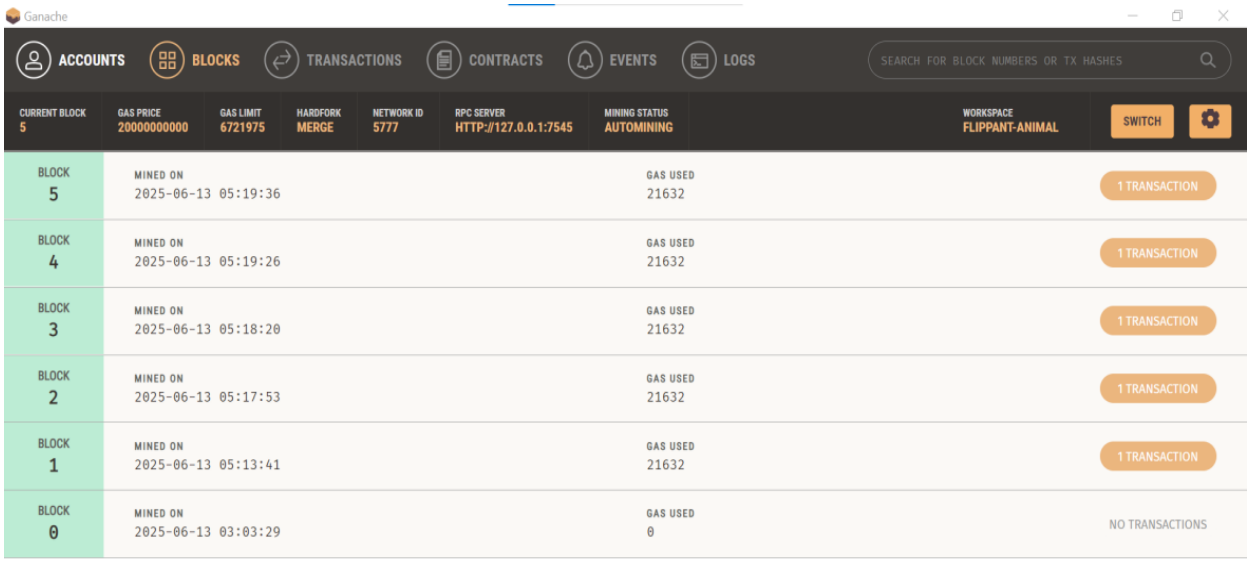


Рисунок 3.2 – Вкладка Blocks після підтвердженої транзакції

У ході тестування сценарію з отриманням балансу було підтверджено, що запит до провайдера (provider.getBalance) коректно зчитує значення, а результат правильно форматовано у десяткову систему. В інтерфейсі виводилося значення з точністю до чотирьох знаків після коми, що повністю відповідало значенню, зафіксованому в Ganache.

Взаємодія з токеном стандарту ERC-20 здійснювалася через виклик функції balanceOf(), що реалізована в контракті. У полі для введення адреси зазначався підключений гаманець, і після виклику виводилася кількість токенів, яка відповідала значенню у тестовому середовищі Ganache. Було перевірено правильність передачі ABI, реакцію на некоректні адреси, а також стабільність оновлення UI після успішного зчитування балансу.

Додатково було перевірено сценарій повторного підключення та відключення гаманця. При кожному натисканні на кнопку “Підключити

гаманець” з’являлася адреса, тоді як після натискання кнопки “Відключити” інтерфейс повертався до початкового стану. Повторні спроби не спричиняли дублювання запитів або неочікуваних оновлень, що засвідчило коректність логіки обробки стану застосунку.

Окрім тестування через Ganache, застосовувалося модульне тестування з використанням Jest. У рамках цих перевірок здійснювалася емуляція функцій підключення, зчитування балансу, а також перевірка валідації форми надсилання транзакцій. Тестові сценарії були реалізовані з використанням мок-версій Ethers.js та провайдера, що дало змогу перевірити поведінку окремих функцій та компонентів без необхідності доступу до блокчейн-мережі. Демонстраційні уривки програмного коду модульних тестів наведено в лістингу 3.1.

Лістинг 3.1 – Зразки коду модульних тестів

```
jest.mock('./service/walletService');
jest.mock('./service/tokenService');

describe('App component full tests', () => {
  const mockAddress = '0x123...abc';
  const mockSigner = { mock: 'signer' };
  const mockTxHash = '0xabc123txhash';

  beforeEach(() => {
    jest.clearAllMocks();
  });

  test('відображає кнопку підключення гаманця, якщо не підключено', () => {
    render(<App />);
    expect(screen.getByText(/підключити MetaMask/i)).toBeInTheDocument();
  });

  test('успішне підключення гаманця показує адресу та баланс', async () => {
    walletService.connectWallet.mockResolvedValue({ address: mockAddress,
signer: mockSigner });
    tokenService.getTokenBalance.mockResolvedValue('100.0');

    render(<App />);
    fireEvent.click(screen.getByText(/підключити MetaMask/i));

    await waitFor(() => {
      expect(screen.getByText(`🔗 Підключено:
${mockAddress}`)).toBeInTheDocument();
      expect(screen.getByText(/Баланс DAI: 100.0/)).toBeInTheDocument();
    });
  });
```

Усі тести виконувалися локально, і повністю відповідали заданій логіці: перевірка відображення адреси після підключення, обробка помилки при відсутності Web3-провайдера, обробка формових значень. Проведене тестування дозволило не лише перевірити реакцію інтерфейсу на дії користувача, а й оцінити здатність системи до коректної обробки виняткових ситуацій, включаючи відсутність підключення, неправильні вхідні дані та інші нестандартні сценарії. Таким чином, модульне тестування за допомогою Jest забезпечило комплексну перевірку логіки компонентів і загальної стійкості системи до потенційних збоїв. Приклад результатів модульного тестування наведено на рисунках 3.3 та 3.4.

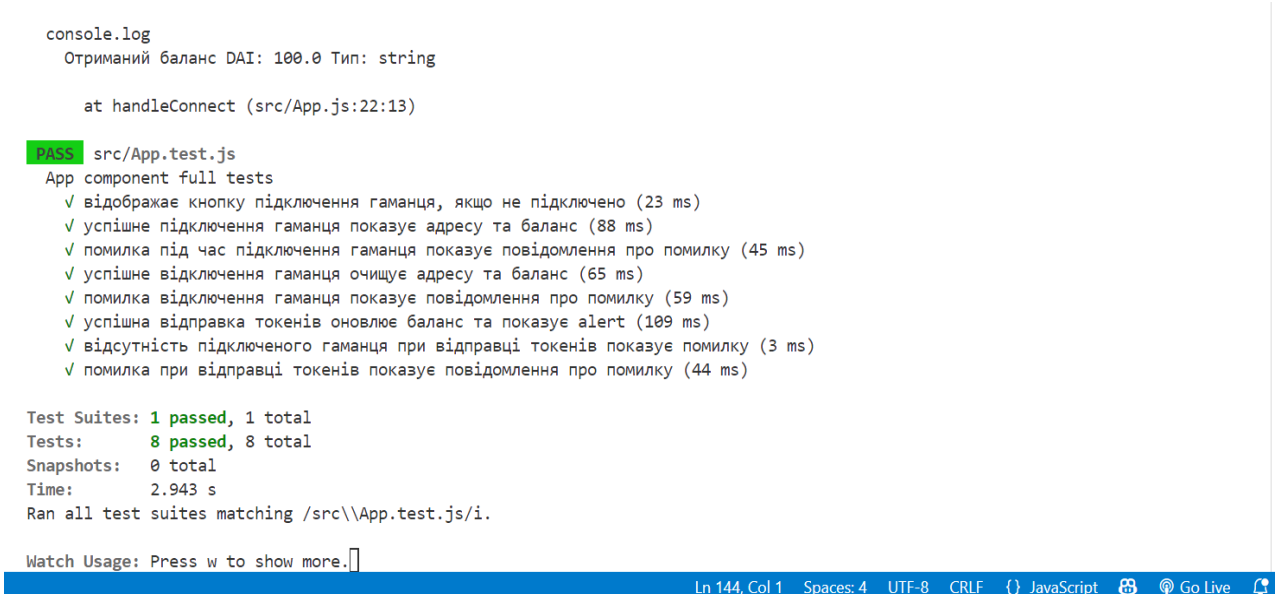


Рисунок 3.3 – Тестування користувацької взаємодії

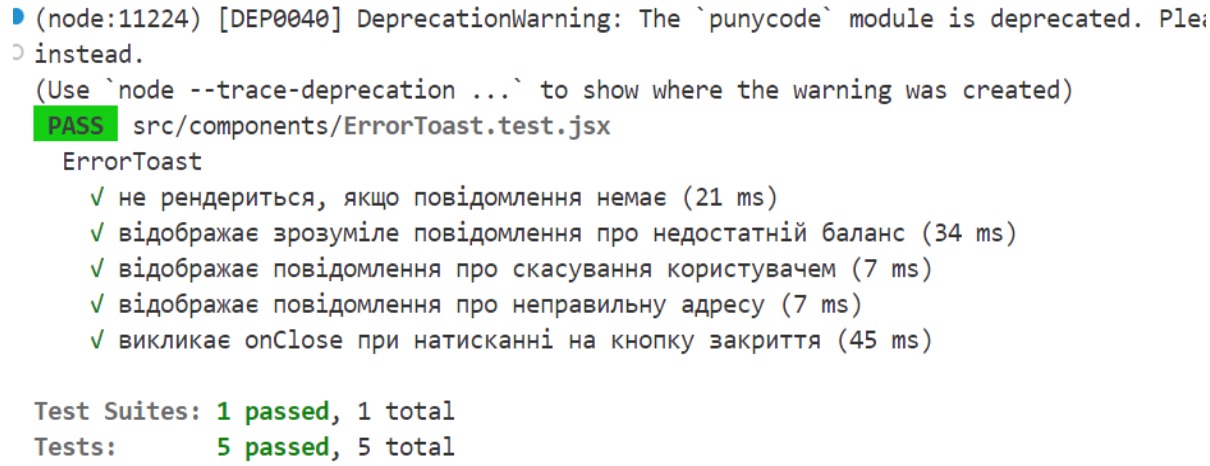


Рисунок 3.4 – Тестування повідомлень про помилки

Отримані результати як функціонального тестування в середовищі Ganache, так і модульного тестування засобами Jest, підтверджують відповідність реалізованого Web3-гаманця вимогам до функціональності, стійкості інтерфейсу та надійності взаємодії з блокчейн-мережею. Це свідчить про стабільну реалізацію застосунку та його готовність до подальшого розгортання в публічному середовищі.

# ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було розроблено, теоретично обґрунтовано та практично реалізовано інформаційну систему цифрового гаманця, яка функціонує на основі блокчейн-технологій. Запропоноване рішення враховує ключові вимоги до безпеки, децентралізації, масштабованості й функціональності, що відповідає сучасним тенденціям розвитку фінансових Web3-застосунків.

На основі аналізу предметної області було досліджено архітектуру цифрових гаманців на різних програмних і блокчейн-платформах, окреслено ключові принципи функціонування систем на базі Ethereum, включаючи концепцію розподіленого реєстру, механізми консенсусу, структуру блоків, а також криптографічні методи забезпечення цілісності й достовірності транзакцій. Окрему увагу було приділено стандартам взаємодії з токенами, зокрема ERC-20, а також застосуванню смарт-контрактів як механізмів реалізації логіки цифрового активу.

В процесі проєктування було сформовано архітектурну модель системи, що базується на чотирирівневій структурі з розподілом на модулі frontend, services, blockchain та utils. Побудовано UML-діаграми, що відображають логічні залежності між компонентами, динаміку обробки транзакцій, сценарії взаємодії користувача та структурні аспекти класів. Застосовано підхід компонентної ізоляції, що дозволяє легко масштабувати систему та інтегрувати її з іншими блокчейн-мережами чи Web3-сервісами.

Реалізацію гаманця виконано з використанням JavaScript-стеку, де клієнтську частину побудовано на React, а взаємодію з блокчейном забезпечує бібліотека Ethers.js. Система підтримує функціонал підключення гаманця через Web3Modal, перегляду балансу, виконання транзакцій токенів стандарту ERC-20, підпису повідомлень та перевірки дозволів. Всі ключові дії користувача супроводжуються обробкою виключень та відповідною реакцією інтерфейсу, що

дозволяє гарантувати стабільність і передбачуваність поведінки системи навіть у випадку зовнішніх збоїв або відмов мережі.

Результати тестування підтвердили працездатність реалізованого рішення та його відповідність функціональним вимогам. Реалізований цифровий гаманець забезпечує повну автономність управління активами, зберігаючи ключову перевагу блокчейн-технологій – відсутність довіри до сторонніх сервісів. Система готова до використання як окремий застосунок або як модуль для інтеграції у більші програмні продукти у сфері децентралізованих фінансів (DeFi).

Отже, поставлену в роботі мету – створення цифрового гаманця на основі блокчейн-технологій з дотриманням вимог безпеки, децентралізації та функціональної повноти – було досягнуто повністю. Отримані результати мають практичну цінність і можуть бути використані як основа для подальшого вдосконалення систем керування цифровими активами, реалізації NFT-функціоналу, багатомережевої підтримки чи інтеграції з DAO-механізмами.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Gate.io. Cryptographic Hashing and Digital Signatures in Blockchain. 2022. URL: https://www.gate.io/uk/blog/1807/Cryptographic-hashing-and-digital-signatures-in-blockchain (дата звернення: 01.06.2025).
2. MetaMask Docs. URL: https://docs.metamask.io (дата звернення: 01.06.2025).
3. Binance Academy. What Is MetaMask and How Do You Use It? URL: https://academy.binance.com/en/articles/what-is-metamask (дата звернення: 01.06.2025).
4. MetaMask. Developers – JSON-RPC API Reference. URL: https://docs.metamask.io/wallet/reference/ (дата звернення: 01.06.2025).
5. Antonopoulos A. Mastering Ethereum: Building Smart Contracts and DApps. 1st ed. Sebastopol: O'Reilly Media, 2018. 424 p.
6. Nakamoto S. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008. URL: https://bitcoin.org/bitcoin.pdf (дата звернення: 01.06.2025).
7. Wood G. Ethereum: A Secure Decentralised Generalised Transaction Ledger. Ethereum Project Yellow Paper. 2014. URL: https://ethereum.github.io/yellowpaper/paper.pdf (дата звернення: 01.06.2025).
8. Buterin, V. A Next-Generation Smart Contract and Decentralized Application Platform. 2014. URL: https://ethereum.org/en/whitepaper/ (дата звернення: 01.06.2025).
9. Ethereum Foundation. Ethereum Developer Documentation. URL: https://ethereum.org/en/developers/docs/ (дата звернення: 01.06.2025).
10. Al-Bassam, M. Scalable and Trustless Computation with Blockchain and Smart Contracts. *IEEE Security & Privacy*. 2019. Vol. 17. No. 2. pp. 62–65. DOI: 10.1109/MSEC.2019.2899383.
11. Antonopoulos, A. M. Mastering Bitcoin: Programming the Open Blockchain. O'Reilly Media, 2017. 496p.

12. Monero Project. Monero Research Lab. URL: https://www.getmonero.org/resources/research-lab/ (дата звернення: 01.06.2025).
13. Інтерфейс мультивалютного гаманця. URL: https://images.app.goo.gl/XvP4dFjjFrbTQWgX8 (дата звернення: 01.06.2025).
14. Blockchain-Based Identity Management System and Self-Sovereign Identity Ecosystem: A Comprehensive Survey / M. R. Ahmed et al. *IEEE Access*. 2022. Vol. 10. P. 113436–113481. URL: https://doi.org/10.1109/access.2022.3216643 (дата звернення: 01.06.2025).
15. Nehra V., Sharma A. K., Tripathi R. K. Blockchain Implementation for Internet of Things Applications. *Handbook of Research on Blockchain Technology*. 2020. P. 113–132. URL: https://doi.org/10.1016/b978-0-12-819816-2.00005-8 (дата звернення: 01.06.2025).

# ДОДАТКИ

## Додаток А – Програмна реалізація

Репозиторій з програмною реалізацією, виконаною в межах кваліфікаційної роботи, розташовано за адресою https://github.com/VaDimk12/web3-wallet-web