Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: Розробка системи аудиту бухгалтерського обліку з використанням технології блокчейн та розподілених реєстрів

Виконав(ла): студент(ка) 6 курсу, групи СПм-61
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

	(підпис)	Романюк Д.В. (прізвище та ініціали)
Керівник	(підпис)	Мудрик І.Я. (прізвище та ініціали)
Нормоконтроль	(підпис)	Стоянов Ю.М. (прізвище та ініціали)
Завідувач кафедри	(підпис)	Петрик М.Р. (прізвище та ініціали)
Рецензент	(підпис)	Муж В.В. (прізвище та ініціали)

Тернопіль
2024

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ _____
(підпис) (прізвище та ініціали)

« » 20___ р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістр
(назва освітнього ступеня)

за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

студенту Романюку Дмитру Вікторовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Розробка системи аудиту бухгалтерського обліку з використанням технології блокчейн та розподілених реєстрів

Керівник роботи Мудрик Іван Ярославович, д.ф., ст. викладач
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «___» _____ 20___ року № _____

2. Термін подання студентом завершеної роботи _____

3. Вихідні дані до роботи вимоги до функціоналу системи, предмет дослідження, аналіз методик організації, документація платформи Hyperledger Fabric та Node.js.

4. Зміст роботи (перелік питань, які потрібно розробити)
Аналіз предметної області аудиту бухгалтерського обліку та огляд конкурентів, технічний аспект проблеми розробки системи на основі блокчейн, розробка моделі предметної області та бізнес моделі, проєктування архітектури додатку і його конструювання, тестування програмного забезпечення та оцінка його якості, розділ безпеки у надзвичайних ситуаціях та охорони праці.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень, слайдів)
Логотипи та архітектура конкурентів, діаграма варіантів використання, діаграма архітектури, діаграма сутностей(класів), діаграма послідовності, діаграма діяльності, скриншоти розгорнутої системи блокчейн, лістинги коду, скриншоти графічного інтерфейсу користувача, слайди презентації.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Осухівська Галина Михайлівна		
Безпека в надзвичайних ситуаціях	Клепчик Василь Михайлович Стручок Володимир Сергійович		
Нормоконтроль	Стоянов Юрій Миколайович		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота магістра. Тернопільський національний технічний університет імені Івана Пулюя, кафедра програмної інженерії, спеціальність 121 «Інженерія програмного забезпечення», ТНТУ, 2024. Сторінок 93, рисунків 14, презентація.

Тема: Розробка системи аудиту бухгалтерського обліку з використанням технології блокчейн та розподілених реєстрів.

Кваліфікаційна робота магістар присвячена розробці системи аудиту бухгалтерського обліку на основі блокчейн-технологій. У роботі досліджено сучасні проблеми аудиторських систем, такі як недостатня прозорість процесів, висока залежність від людського фактора, ризики шахрайства та складності інтеграції з ERP-системами. Запропонована система ClearLedger, побудована на платформі Hyperledger Fabric, забезпечує децентралізоване зберігання фінансових даних, автоматичну перевірку транзакцій, формування аудиторських звітів та інтеграцію з ERP-системами. Особливу увагу приділено питанням безпеки: впроваджено механізми шифрування, контроль доступу на основі ролей та захист від несанкціонованого втручання. Результати впровадження демонструють високу ефективність системи у забезпеченні прозорості, безпеки та надійності аудиторських процесів, а також її гнучкість для адаптації до специфічних потреб бізнесу.

Ключові слова: блокчейн, Hyperledger Fabric, аудит, бухгалтерський облік, автоматизація, прозорість, фінансові транзакції, смарт-контракти, безпека даних, ERP-системи.

ABSTRACT

Masters qualification work. Ternopil Ivan Puluj National Technical University, Department of Software Engineering, specialty 121 “Software Engineering”, TNTU, 2024, Pages 93, figures 14, presentation.

Topic: The development of an accounting audit system that utilizes blockchain technology and distributed ledgers is the subject of this study.

The masters thesis is dedicated to the development of an accounting audit system based on blockchain technologies.The study explores the current challenges of audit systems, including issues such as inadequate transparency of processes, excessive reliance on human intervention, fraud risks, and challenges in integrating with ERP systems.The proposed ClearLedger system, constructed on the Hyperledger Fabric platform, offers decentralized storage of financial data, automated transaction verification, generation of audit reports, and integration with ERP systems. A particular emphasis was placed on security concerns, with the implementation of encryption mechanisms, role-based access control, and protection against unauthorized interference.The results of the implementation demonstrate the system's effectiveness in ensuring transparency, security, and reliability of audit processes, as well as its adaptability to specific business needs.

Keywords: blockchain, Hyperledger Fabric, audit, accounting, automation, transparency, financial transactions, smart contracts, data security, ERP systems.

ЗМІСТ

ВСТУП

Аудит бухгалтерського обліку є одним із фундаментальних інструментів забезпечення прозорості та достовірності фінансових даних, що відіграє вирішальну роль у сучасній економіці. Його значення важко переоцінити, адже саме через аудит компанії демонструють свою відповідальність перед інвесторами, кредиторами та регуляторами. Утім, із розвитком цифрових технологій та збільшенням обсягів даних, традиційні системи аудиту стикаються з численними викликами. Підвищення складності фінансових процесів, ризики шахрайства, вразливість до помилок людського фактора та складнощі у відстеженні змін у даних створюють нові вимоги до ефективності та безпеки аудиторських систем.

У контексті розвитку цифрової економіки та активного використання інноваційних технологій актуальність теми дослідження полягає у необхідності розробки системи, що використовує потенціал блокчейну для оптимізації процесів бухгалтерського обліку та аудиту. Така система здатна вирішувати не лише локальні завдання конкретного підприємства, але й сприяти розвитку загальної довіри у фінансовій екосистемі.

Предметом даного дослідження є розробка інноваційної системи аудиту бухгалтерського обліку, що базується на технологіях блокчейн та розподілених реєстрів. Блокчейн, як технологія, здатний вирішити багато з перелічених проблем завдяки своїй здатності забезпечувати незмінність даних, прозорість процесів і децентралізований підхід до зберігання інформації. Особливо актуальним є питання інтеграції блокчейн-рішень із існуючими ERP-системами, що дозволяє автоматизувати більшість рутинних процесів аудиту, скоротити витрати на перевірки та підвищити загальну продуктивність роботи.

Основною проблематикою предметної області є необхідність забезпечення балансу між прозорістю облікових даних та конфіденційністю, яка потрібна для

роботи з чутливою фінансовою інформацією. Традиційні централізовані рішення, попри їхню популярність, часто не здатні гарантувати повну безпеку та надійність даних, що створює додаткові ризики для організацій. Крім того, багато існуючих систем обмежені у можливостях автоматизації, що збільшує часові та фінансові витрати на аудит.

Наукова новизна роботи полягає в розробці архітектури та практичній реалізації системи аудиту на основі блокчейн-технологій, яка поєднує у собі децентралізоване управління даними, автоматизацію перевірок і сучасні методи виявлення аномалій у транзакціях. Унікальність підходу полягає у використанні смарт-контрактів для реалізації бізнес-логіки аудиту, що дозволяє не лише реєструвати транзакції, але й автоматично перевіряти їхню відповідність заданим правилам. Інтеграція з аналітичними інструментами та підтримка різних типів звітності дають змогу адаптувати систему до специфічних потреб компаній, створюючи гнучке та ефективне рішення.

Таким чином, дана робота спрямована на вирішення актуальних проблем у сфері аудиту шляхом впровадження технологій блокчейн у бухгалтерський облік. Запропонована система покликана підвищити рівень прозорості та достовірності фінансових даних, знизити ризики помилок і шахрайства, а також автоматизувати ключові процеси аудиту. У наступних розділах детально розглянуто аналіз предметної області, моделювання архітектури системи, її реалізацію та результати впровадження, які підтверджують ефективність розробленого рішення.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Розробка системи аудиту бухгалтерського обліку з використанням блокчейн-технологій є актуальною темою в умовах зростаючих вимог до прозорості, безпеки та автоматизації фінансових процесів. Бухгалтерський облік є однією з ключових складових функціонування будь-якої організації, а аудит забезпечує контроль за його коректністю та відповідністю нормативним стандартам. Сучасні системи обліку та аудиту стикаються з низкою проблем, які потребують вирішення для підвищення їхньої ефективності та надійності.

Основним предметом дослідження є можливість впровадження технологій блокчейн для створення децентралізованої системи, яка забезпечить прозорість та незмінність даних, автоматизацію аудиторських перевірок і захист інформації від несанкціонованого доступу. Традиційні методи зберігання та перевірки бухгалтерських даних часто страждають від ризиків шахрайства, людських помилок і складнощів при відновленні історії операцій. Крім того, централізовані системи схильні до атак та втрат даних, що ставить під загрозу фінансову безпеку організацій.

Досліджувана проблематика включає аналіз сучасного стану облікових та аудиторських систем, визначення їхніх недоліків і викликів, а також оцінку перспектив застосування блокчейн-технологій для вдосконалення цих процесів. Особливу увагу приділено питанням автоматизації аудиту, зменшення ризику помилок, забезпечення конфіденційності даних та інтеграції системи з існуючими ERP-рішеннями. Метою цього розділу є створення теоретичної бази для подальшого проектування та реалізації системи, яка відповідатиме вимогам сучасного бізнес-середовища.

1.1. Огляд конкурентів

Аналіз конкурентів є важливим етапом будь-якого дослідження, особливо у сфері впровадження новітніх технологій, таких як блокчейн. Цей аналіз дозволяє зрозуміти сучасний стан ринку, визначити ключових гравців, їхні переваги та недоліки, а також побачити потенційні можливості для вдосконалення. У межах цього підпункту буде проведено огляд існуючих рішень, що вже використовуються для аудиту бухгалтерського обліку з блокчейном або суміжних технологій. Основна мета аналізу — виявити найкращі практики, оцінити функціональність конкурентних продуктів і визначити унікальні особливості, які можна застосувати у власному рішенні.

Цей аналіз допоможе обґрунтувати вибір технологічного підходу, виділити унікальні переваги майбутньої системи та визначити шляхи її вдосконалення.

SAP Blockchain — це платформа, інтегрована в екосистему SAP, яка дозволяє компаніям використовувати блокчейн-технології для підвищення прозорості, автоматизації та ефективності бізнес-процесів. Вона тісно інтегрована з іншими продуктами SAP, такими як SAP S/4HANA, що робить її особливо привабливою для підприємств, які вже використовують цю ERP-систему [1].

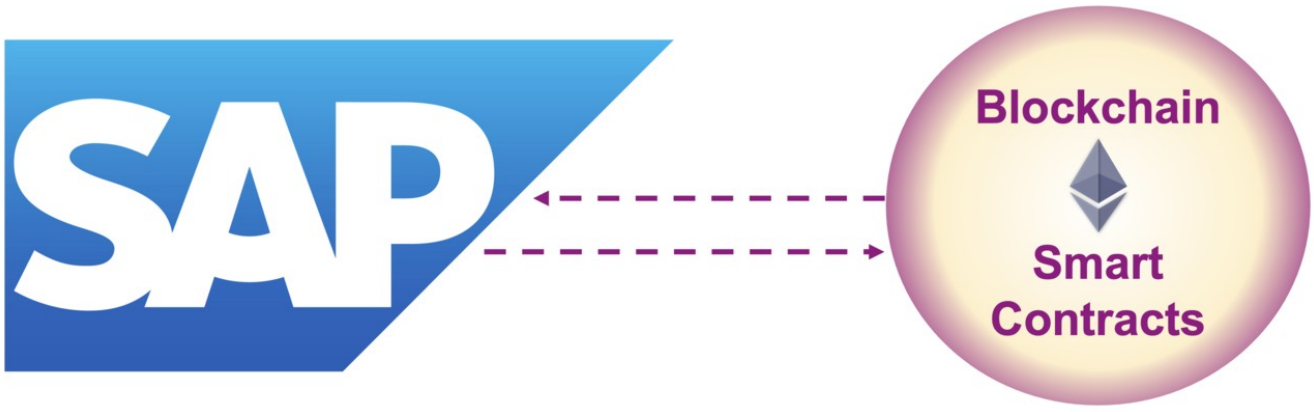

Рис 1.1 — Логотип SAP Blockchain.

Основні можливості SAP Blockchain

Інтеграція з ERP-системами: SAP Blockchain легко впроваджується у вже існуючі SAP-продукти. Це дозволяє компаніям автоматизувати бухгалтерські процеси, зберігати історію транзакцій і синхронізувати дані між різними відділами.

Підтримка різних блокчейн-мереж: Платформа працює з приватними блокчейнами, такими як Hyperledger Fabric, MultiChain і Quorum, що забезпечує гнучкість у виборі мережі для конкретних завдань.

Прозорість даних: Завдяки блокчейну кожна операція фіксується в розподіленому реєстрі, що робить її доступною для перегляду уповноваженими сторонами. Це важливо для аудиторських перевірок і забезпечення відповідності нормативним вимогам.

IoT та блокчейн: SAP Blockchain інтегрується з IoT-пристроями, що дозволяє збирати реальні дані в режимі реального часу. Наприклад, це може бути відстеження умов транспортування товарів або контроль їх походження.

Автоматизація перевірок: Смарт-контракти дозволяють автоматично перевіряти відповідність транзакцій певним умовам, що значно скорочує час на аудит.

Переваги SAP Blockchain:

- Інтеграція з SAP-продуктами: Для компаній, які вже працюють із SAP, впровадження блокчейну стає швидким і безболісним, оскільки платформа використовує знайомий функціонал.
- Прозорість і автоматизація: Система забезпечує детальну історію всіх транзакцій і автоматизує їх перевірку за допомогою смарт-контрактів.
- Підтримка корпоративного рівня: SAP Blockchain створена для великих підприємств, тому вона може обробляти великі обсяги даних і забезпечувати високу продуктивність.

Недоліки SAP Blockchain

- Висока вартість: SAP Blockchain є дорогою через ліцензії та складність інтеграції. Це може бути бар’єром для малих і середніх компаній.
- Залежність від SAP-екосистеми: Максимальна ефективність платформи можлива лише при використанні інших продуктів SAP, що обмежує її гнучкість.
- Складність впровадження: Для роботи з SAP Blockchain потрібні висококваліфіковані спеціалісти, що може ускладнити початкове впровадження.
- Обмеження в роботі з публічними блокчейнами: Платформа більше орієнтована на приватні мережі, що знижує її універсальність.

Реальні приклади впровадження:

- Nestlé: Використання SAP Blockchain для відстеження постачання продуктів харчування з метою забезпечення прозорості.
- De Beers: Контроль походження алмазів у ланцюгу постачання для боротьби з незаконним видобутком.
- Daimler: Автоматизація фінансових операцій і управління активами з використанням блокчейну.

SAP Blockchain — це потужне рішення для компаній, які прагнуть автоматизувати бухгалтерські та фінансові процеси, забезпечити прозорість і підвищити ефективність. Незважаючи на певні виклики, такі як висока вартість впровадження, переваги, які надає це рішення, значно переважають. Завдяки інтеграції з екосистемою SAP, це рішення ідеально підходить для великих корпорацій, які прагнуть цифрової трансформації.

Oracle Blockchain Applications — це набір готових до використання програмних рішень, інтегрованих у хмарну платформу Oracle, які дозволяють організаціям швидко впроваджувати блокчейн-технології для оптимізації бізнес-процесів, підвищення прозорості та забезпечення довіри між учасниками ланцюга постачання [2].

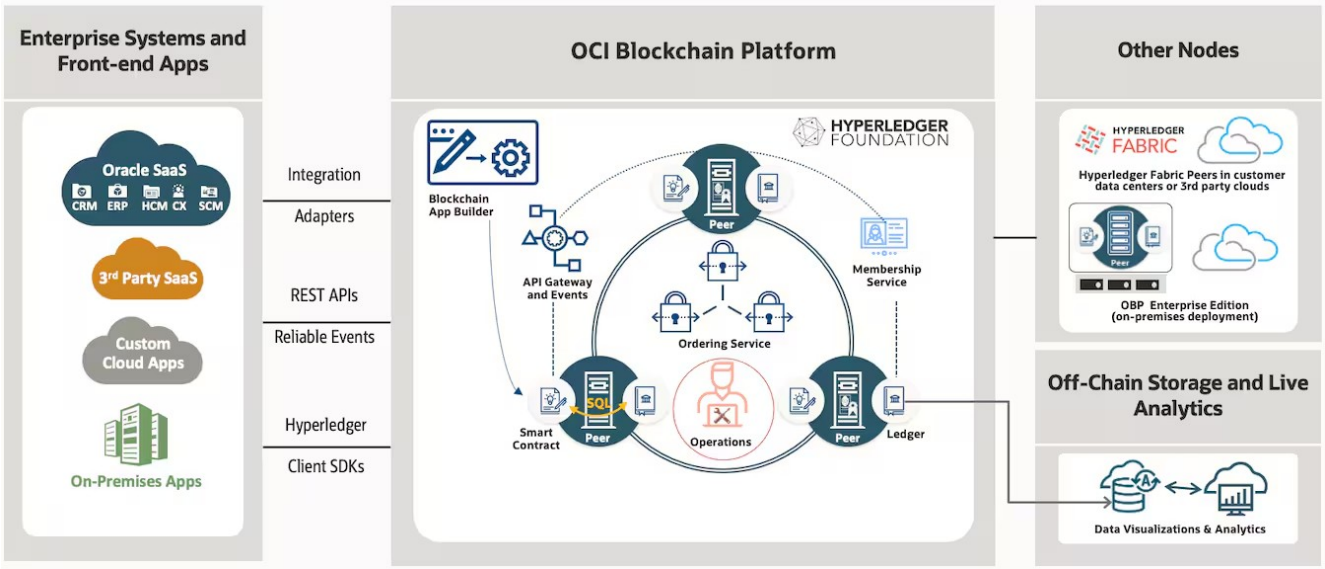

Рис. 1.2 — Екосистема Oracle Blockchain

Основні компоненти Oracle Blockchain Applications:

Oracle Intelligent Track and Trace: Ця програма дозволяє відстежувати рух товарів у ланцюгу постачання, забезпечуючи прозорість та автентичність даних про походження продуктів. Вона інтегрується з IoT-пристроями для збору реальних даних, що підвищує точність інформації.

Oracle Lot Lineage and Provenance: Цей інструмент забезпечує детальне відстеження партій товарів, дозволяючи визначити походження та історію кожної партії, що є критично важливим для галузей з високими вимогами до якості та безпеки продукції.

Oracle Intelligent Cold Chain: Програма спеціалізується на моніторингу умов транспортування товарів, які потребують дотримання певних температурних режимів, таких як фармацевтична продукція чи продукти харчування. Вона використовує дані з IoT-сенсорів для забезпечення відповідності умов зберігання стандартам.

Oracle Warranty and Usage Tracking: Цей компонент дозволяє відстежувати умови використання та гарантійні зобов'язання щодо продукції, що допомагає зменшити кількість шахрайських гарантійних випадків та покращити обслуговування клієнтів.

Ключові особливості Oracle Blockchain Applications

- Інтеграція з існуючими системами: Рішення легко інтегруються з наявними ERP-системами, такими як Oracle E-Business Suite та Oracle JD Edwards, що спрощує впровадження та знижує витрати на адаптацію.
- Використання IoT та штучного інтелекту: Завдяки інтеграції з IoT-пристроями та використанню вбудованих функцій штучного інтелекту, програми забезпечують збір та аналіз реальних даних, що підвищує точність та актуальність інформації.
- Прозорість та довіра: Використання блокчейн-технології гарантує незмінність та доступність даних для всіх учасників, що підвищує довіру між партнерами та спрощує аудит.
- Гнучкість та масштабованість: Рішення можуть бути розгорнуті як у хмарі, так і локально, що дозволяє компаніям обирати оптимальний варіант залежно від їхніх потреб та вимог безпеки.

Переваги використання Oracle Blockchain Applications:

- Швидке впровадження: Готові до використання програми дозволяють швидко розпочати роботу з блокчейн-технологіями без необхідності розробки власних рішень з нуля.
- Зниження операційних ризиків: Прозорість та незмінність даних зменшують ймовірність шахрайства та помилок у бізнес-процесах.
- Покращення взаємодії з партнерами: Єдиний достовірний джерело інформації сприяє кращій координації та співпраці між учасниками ланцюга постачання.
- Відповідність нормативним вимогам: Детальне відстеження та документування всіх етапів процесів допомагає компаніям відповідати вимогам регуляторних органів та стандартів.

Oracle Blockchain Applications надають компаніям потужні інструменти для інтеграції блокчейн-технологій у свої бізнес-процеси, забезпечуючи прозорість, ефективність та довіру між партнерами. Завдяки готовим рішенням та можливості інтеграції з існуючими системами, організації можуть швидко адаптуватися до нових вимог ринку та підвищити свою конкурентоспроможність.

KPMG Chain Fusion — це запатентований набір аналітичних інструментів, розроблений компанією KPMG для допомоги фінансовим установам та фінтех-компаніям в управлінні криптоактивами та традиційними активами через публічні та приватні блокчейн-мережі. Це рішення сприяє інтеграції даних з різних блокчейн-протоколів та традиційних систем, забезпечуючи єдину, узгоджену архітектуру даних для бізнесових, ризикових та комплаєнс-цілей [3].

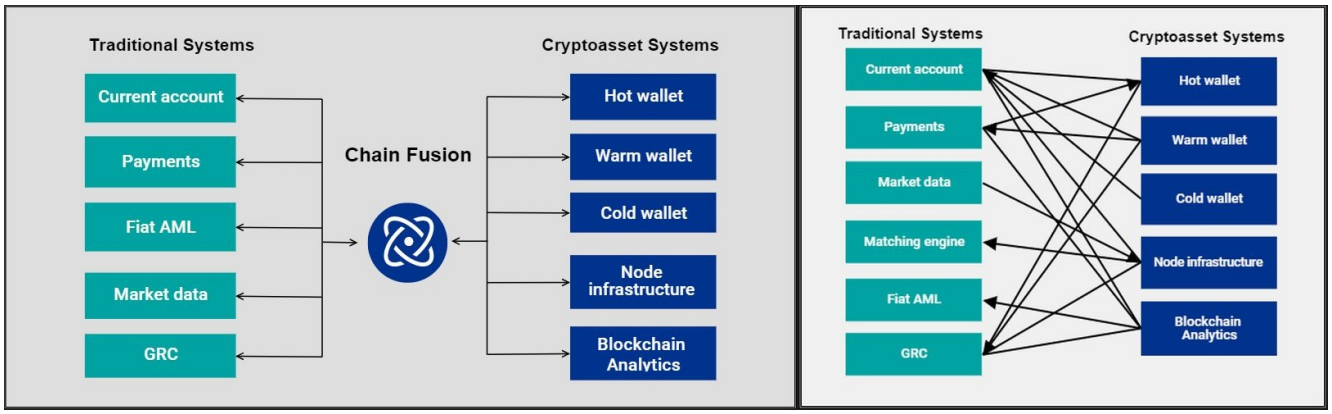

Рис. 1.3 — Екосистема KPMG Chain Fusion

Основні характеристики KPMG Chain Fusion

- Інтеграція даних: Chain Fusion об'єднує дані з блокчейн-інфраструктур та традиційних систем, створюючи стандартизовану архітектуру даних. Це дозволяє ефективно використовувати інформацію з різних джерел для бізнес-аналітики, управління ризиками та дотримання нормативних вимог.
- Підтримка різних блокчейн-протоколів: Рішення підтримує інтеграцію з різними блокчейн-протоколами, як публічними, так і приватними, що забезпечує гнучкість у виборі технологічної інфраструктури.

- Масштабованість: Chain Fusion здатний підтримувати аудити для клієнтів з різною кількістю криптоінвестицій — від кількох активів до мільйонів клієнтів у фінансових установах.
- Підвищення якості аудиту: Забезпечуючи надійний доступ до даних з різних блокчейнів та гаманців на єдиній платформі, Chain Fusion підвищує якість та ефективність аудиторських процесів у складному середовищі.
- Переваги використання KPMG Chain Fusion
- Підвищення прозорості: Інтеграція даних з різних джерел забезпечує прозорість операцій та сприяє довірі між учасниками ринку.
- Автоматизація процесів: Chain Fusion дозволяє автоматизувати процеси збору та обробки даних, зменшуючи ймовірність помилок та підвищуючи ефективність операцій.
- Відповідність нормативним вимогам: Рішення допомагає компаніям відповідати вимогам регуляторів щодо фінансової звітності, безпеки та обробки даних.
- Гнучкість та адаптивність: Підтримка різних блокчейн-протоколів та традиційних систем дозволяє компаніям легко адаптуватися до змін у технологічному середовищі.

  Виклики впровадження KPMG Chain Fusion
- Складність інтеграції: Інтеграція з існуючими системами може вимагати значних ресурсів та часу.
- Вартість впровадження: Впровадження такого комплексного рішення може бути дорогим, особливо для малих та середніх підприємств.
- Потреба в спеціалізованих знаннях: Для ефективного використання Chain Fusion необхідні фахівці з досвідом роботи з блокчейн-технологіями та фінансовими системами.

KPMG Chain Fusion є потужним інструментом для фінансових установ та фінтех-компаній, які прагнуть ефективно управляти криптоактивами та

традиційними активами в умовах сучасного ринку. Забезпечуючи інтеграцію даних з різних джерел та підтримку різних блокчейн-протоколів, це рішення сприяє підвищенню прозорості, автоматизації процесів та відповідності нормативним вимогам. Однак впровадження Chain Fusion може бути складним та вимагати значних ресурсів, тому компаніям слід ретельно оцінити свої потреби та можливості перед його використанням.

1.2. Обґрунтування вибору технологій розробки блокчейн рішення

Вибір технологій, середовища розробки та мови програмування є ключовим етапом у реалізації будь-якого програмного проєкту, особливо у сфері, пов’язаній із впровадженням новітніх технологій, таких як блокчейн. Правильно обрані інструменти та платформи визначають не лише технічну ефективність розробки, але й забезпечують відповідність програмного забезпечення сучасним стандартам, зручність у використанні, масштабованість та надійність у довгостроковій перспективі.

Для реалізації системи аудиту бухгалтерського обліку на основі блокчейн-технологій був обраний Hyperledger Fabric — платформа, яка спеціалізується на створенні корпоративних блокчейн-рішень. Цей вибір був зроблений через низку важливих переваг і характеристик, які ідеально підходять для завдань, що вирішуються в межах проєкту.

Відповідність вимогам до конфіденційності та безпеки: Hyperledger Fabric підтримує модель дозволеного (private) блокчейну, що є критично важливим для бухгалтерського обліку, де дані про фінансові операції потребують суворого контролю доступу. Тільки авторизовані учасники мережі можуть отримати доступ до інформації, а це забезпечує високий рівень конфіденційності та безпеки даних.

Дані про транзакції доступні лише аудиторам та уповноваженим співробітникам компанії, що мінімізує ризик витоку інформації.

Гнучкість у налаштуванні: Hyperledger Fabric дозволяє налаштовувати архітектуру мережі відповідно до потреб проєкту. Зокрема, можна створювати різні канали для обміну інформацією між певними учасниками. Це означає, що окремі компанії або департаменти можуть обмінюватися даними незалежно від інших учасників мережі. Фінансовий департамент і відділ аудиту можуть мати свій приватний канал для обговорення чутливих даних.

Підтримка смарт-контрактів Hyperledger Fabric підтримує смарт-контракти (chaincode), які автоматизують виконання умов угод. Це ідеально підходить для аудиту, оскільки смарт-контракти можуть автоматично перевіряти відповідність транзакцій встановленим правилам. Смарт-контракт може автоматично перевіряти, чи відповідає фінансова операція встановленим нормам перед її реєстрацією.

Підтримка корпоративного рівня: Як частина проєкту Hyperledger під егідою Linux Foundation, Fabric має стабільну підтримку від великих корпорацій, таких як IBM. Це забезпечує:

- Постійне оновлення платформи.
- Розширену документацію.
- Технічну підтримку.

Масштабованість: Hyperledger Fabric забезпечує високу продуктивність і масштабованість завдяки своїй модульній архітектурі. Це дозволяє легко додавати нових учасників у мережу та розширювати її функціональні можливості. У майбутньому можна додати більше компаній або державних установ до системи аудиту без значних витрат на модифікацію платформи.

Підтримка кількох мов програмування: Fabric дозволяє розробляти смарт-контракти на популярних мовах програмування, таких як Go, Java і Node.js. Це спрощує розробку, оскільки для роботи з платформою не потрібно вивчати спеціалізовані мови, такі як Solidity.

Висока продуктивність: На відміну від публічних блокчейнів, таких як Ethereum, які часто страждають від перевантажень і високих витрат на транзакції, Fabric використовує алгоритми консенсусу, які забезпечують швидку обробку транзакцій і низьку затримку. Реєстрація транзакції в системі займає лише кілька секунд, що ідеально для реального часу роботи аудиторів.

Підтримка модульної архітектури: Hyperledger Fabric дозволяє налаштовувати окремі компоненти системи, наприклад, механізми консенсусу, зберігання даних і управління ідентифікацією. Це забезпечує гнучкість у реалізації специфічних вимог проєкту.

. Відповідність бізнес-вимогам: Hyperledger Fabric створений для вирішення корпоративних завдань, таких як аудит, управління ланцюгами постачання, банківські операції тощо. Його можливості відповідають усім вимогам для створення ефективної системи аудиту бухгалтерського обліку. Hyperledger Fabric був обраний завдяки його гнучкості, безпеці, підтримці смарт-контрактів і здатності задовольняти потреби корпоративного середовища. Ця платформа забезпечує високий рівень продуктивності, конфіденційності та масштабованості, що робить її ідеальним вибором для розробки системи аудиту бухгалтерського обліку на основі блокчейну [4].

Raft — це вбудований алгоритм консенсусу в Hyperledger Fabric, який ідеально підходить для систем, що потребують стабільності та простоти налаштування. Однією з головних переваг є те, що Raft не потребує додаткових зовнішніх сервісів, таких як Kafka, а працює безпосередньо в межах Fabric. Це значно знижує складність впровадження та зменшує ризики збоїв через проблеми з додатковими компонентами. Raft дозволяє ефективно обробляти транзакції завдяки моделі з вибором лідера, що відповідає за упорядкування операцій. Такий підхід забезпечує високу продуктивність і стабільність навіть за умови масштабування мережі. Він також підтримує додавання нових вузлів у мережу без порушення її роботи, що важливо для розвитку системи в майбутньому. У разі відмови одного з вузлів Raft автоматично обирає нового лідера, забезпечуючи

безперервність роботи. Ця функція робить його стійким до збоїв і надійним у роботі. Оскільки Raft є частиною офіційної реалізації Hyperledger Fabric, він має широку підтримку спільноти та документацію, що полегшує його використання.

Node.js є одним із найкращих варіантів для розробки смарт-контрактів завдяки його продуктивності, популярності та широким можливостям. Особливо це актуально для інтеграції з Hyperledger Fabric, який офіційно підтримує Node.js як мову для розробки chaincode. Node.js забезпечує асинхронну модель обробки запитів, що дозволяє ефективно працювати з великою кількістю транзакцій у блокчейні. Простота у використанні та підтримка JavaScript роблять його зручним для команди розробників, а велика кількість бібліотек через npm допомагає швидко додавати необхідні функціональні можливості. Ключовою перевагою є інтеграція з іншими компонентами системи, такими як API або бази даних, що спрощує створення комплексних рішень. Хоча Node.js має обмеження, наприклад, у багатопоточності, для задач аудиту це не критично. За необхідності можна застосовувати додаткові підходи, такі як кластеризація. Node.js забезпечує оптимальний баланс між продуктивністю, простотою розробки та гнучкістю/

Python є одним із найбільш універсальних і популярних мов програмування, що робить його чудовим вибором для розробки системи аудиту бухгалтерського обліку. Простий і зрозумілий синтаксис Python дозволяє швидко створювати код, що знижує ризик помилок і прискорює розробку. Широка екосистема бібліотек, таких як web3.py для роботи з блокчейнами, Pandas для аналізу даних і Flask для створення API, дозволяє швидко інтегрувати різні функціональності. Python також підтримує роботу з Hyperledger Fabric через SDK, що робить його сумісним із обраною платформою. Python легко інтегрується з базами даних та іншими системами, що важливо для забезпечення повного циклу аудиту. Крім того, він має відмінні інструменти для тестування і автоматизації, що спрощує перевірку працездатності системи. Хоча Python поступається деяким мовам у швидкості виконання, його простота і багатофункціональність компенсують цей недолік.

CouchDB — це нереляційна база даних, яка чудово підходить для роботи з блокчейн-рішеннями, особливо у контексті Hyperledger Fabric. Вона є рекомендованим інструментом для зберігання даних стану (state database) у Fabric, оскільки інтегрується з платформою за замовчуванням. CouchDB підтримує JSON-документи, що робить її ідеальною для зберігання структурованих даних, таких як транзакції, облікові записи та фінансова інформація. Її можливості для виконання складних запитів безпосередньо з бази даних спрощують доступ до потрібних даних і знижують навантаження на блокчейн. Однією з ключових переваг CouchDB є синхронізація та реплікація даних, що забезпечує їхню доступність у розподіленому середовищі. Це дозволяє зберігати копії даних на різних вузлах, підвищуючи стійкість до відмов і швидкість доступу [5].

Visual Studio Code (VSCode) є одним із найпопулярніших редакторів коду, який забезпечує потужні інструменти для розробки, тестування та налагодження програмного забезпечення. Його використання для розробки системи аудиту бухгалтерського обліку базується на зручності, гнучкості та підтримці сучасних технологій. VSCode підтримує велику кількість розширень, зокрема для роботи з Python, Node.js і Docker, що є ключовими технологіями проєкту. Інтеграція з Git забезпечує просте управління версіями коду, а інструменти для відладки (debugging) дозволяють швидко знаходити та виправляти помилки. Редактор має вбудовану підтримку IntelliSense, яка допомагає з автозавершенням коду, навігацією та аналізом синтаксису. Це прискорює процес розробки та знижує ймовірність помилок. Легка інтеграція з терміналом VSCode дозволяє безпосередньо запускати команди для розгортання Docker-контейнерів або тестування смарт-контрактів.

1.3. Технічний аспект проблеми розробки системи обліку і аудиту за допомогою технології блокчейн

Розробка системи аудиту бухгалтерського обліку на основі блокчейну має низку технічних викликів, які необхідно врахувати для забезпечення надійності, продуктивності та відповідності системи вимогам користувачів. Основні аспекти проблеми включають упорядкування транзакцій, зберігання великих обсягів даних, забезпечення безпеки, інтеграцію з іншими системами, автоматизацію перевірок і масштабованість [6].

Упорядкування транзакцій: Система повинна гарантувати, що всі фінансові транзакції виконуються у правильному порядку та підтверджуються всіма учасниками мережі. Для цього використовується механізм консенсусу, який забезпечує єдине джерело істини для всіх учасників. Hyperledger Fabric використовує алгоритм Raft, який обирає одного лідера для упорядкування транзакцій. Це забезпечує швидкість і надійність, дозволяючи обробляти сотні транзакцій за секунду.

Зберігання великих обсягів даних: Бухгалтерська система генерує велику кількість записів, включаючи фінансові операції, рахунки-фактури та звіти. Зберігати всі ці дані в блокчейні технічно складно та дорого. У Hyperledger Fabric дані про стан системи (наприклад, актуальні баланси рахунків) зберігаються у CouchDB. Ця база дозволяє виконувати швидкі запити до даних у форматі JSON, що спрощує доступ до них. Кожна транзакція фіксується у блокчейні, а деталі зберігаються у CouchDB для зручного пошуку.

Безпека та конфіденційність: Оскільки бухгалтерська інформація є чутливою, потрібно забезпечити її захист від несанкціонованого доступу. Крім того, не всі учасники мережі повинні мати доступ до всіх даних. Hyperledger Fabric використовує приватні канали, які дозволяють обмінюватися інформацією

лише між певними сторонами. Фінансові дані одного підрозділу компанії можуть бути приховані від інших підрозділів, але доступні аудиторам.

Інтеграція з існуючими системами: Багато компаній вже використовують ERP-системи (наприклад, SAP або Oracle) і бухгалтерські програми. Система на основі блокчейну має працювати в тісній взаємодії з ними. Використання REST API для передачі даних між блокчейн-системою та ERP. Автоматичне звірення рахунків-фактур у SAP із записами у блокчейні.

Автоматизація перевірок: Ручна перевірка фінансових операцій займає багато часу і може бути схильною до помилок. Система повинна автоматизувати цей процес. Смарт-контракти дозволяють виконувати автоматичні перевірки транзакцій за наперед заданими правилами. Смарт-контракт може автоматично перевіряти, чи сума транзакції відповідає встановленим лімітам, і блокувати її, якщо виявлено порушення.

Масштабованість: Система повинна забезпечувати стабільну роботу навіть у разі збільшення кількості учасників і транзакцій. Hyperledger Fabric дозволяє додавати нові вузли в мережу без переривання її роботи. Якщо система впроваджується на глобальному рівні, нові вузли додаються для забезпечення локальної обробки транзакцій у кожному регіоні.

Тестування та відлагодження: Перед запуском система повинна пройти ретельне тестування, щоб уникнути помилок у роботі. Docker дозволяє створювати тестові середовища, у яких можна симулювати роботу блокчейн-мережі. Тестування виконання смарт-контрактів на локальній мережі перед їх розгортанням у реальному середовищі.

Використання методології Agile(гнучка розробрка) для розробки забезпечує швидку адаптацію до змін і дозволяє створювати продукт поетапно, із залученням замовників і користувачів у процес розробки. Для створення системи аудиту бухгалтерського обліку на основі блокчейну Agile є оптимальним вибором завдяки її гнучкості, орієнтації на результат та можливості швидко впроваджувати необхідні зміни.

Гнучкість у плануванні та реалізації: Бухгалтерські системи мають складну структуру, а вимоги до них можуть змінюватися в процесі розробки. Agile дозволяє коригувати план на кожному етапі розробки, щоб максимально відповідати потребам замовника. Замість суворого дотримання початкового плану, команда може вносити зміни, якщо вимоги або зовнішні умови змінюються. Якщо замовник вирішить додати новий модуль, наприклад, для аналізу транзакцій у реальному часі, це можна реалізувати без затримки всього проекту.

Поетапна розробка: Agile передбачає розбиття проєкту на короткі цикли (спринти), кожен з яких завершується створенням конкретного функціоналу. Замовник може оцінити результат роботи вже на ранніх етапах і дати зворотний зв’язок, що знижує ризики створення непотрібного або недосконалого продукту. На першому етапі команда реалізує базовий модуль обробки транзакцій, а на наступних додає функції звітності та аналітики.

Пріоритетування завдань: Agile дозволяє сфокусуватися на найважливіших функціях, які приносять найбільшу цінність користувачам. Ключовий функціонал буде готовий і протестований раніше, що дозволить розпочати використання системи ще до її повного завершення. Спочатку команда створює модуль для обробки транзакцій, оскільки він є базовим, а звітність і аналітику додає на наступних етапах.

Адаптація до технологічних змін: Системи на базі блокчейну швидко розвиваються, і нові технології або підходи можуть з’явитися навіть у процесі розробки. Agile дозволяє легко інтегрувати нові рішення. Команда може адаптуватися до нових технологій або змінювати підхід без значних втрат часу та ресурсів. Якщо під час розробки з’явиться новий інструмент для роботи з Hyperledger Fabric, його можна буде інтегрувати вже в існуючий процес.

Методологія Agile добре підходить для розробки системи аудиту на основі блокчейну. Вона дозволяє створювати функціональні компоненти поступово, швидко реагувати на зміни. Такий підхід мінімізує ризики, оптимізує

використання ресурсів і гарантує, що кінцевий продукт буде відповідати вимогам користувачів і сучасним технологічним стандартам.

2 МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО КОМПЛЕКСУ

Моделювання та проектування системи аудиту бухгалтерського обліку є одним із найважливіших етапів розробки, оскільки воно дозволяє створити структуру, яка забезпечує ефективне функціонування системи. Цей розділ охоплює аналіз предметної області, визначення ключових вимог, моделювання бізнес-процесів та проектування архітектури.

Моделювання та проектування охоплюють декілька послідовних етапів, кожен із яких має критичне значення для досягнення бажаного результату. Перш за все, виконується аналіз стейкхолдерів і визначення їхніх потреб. Після цього визначаються функціональні та нефункціональні вимоги, які стають основою для моделювання бізнес-процесів та проектування архітектури системи. Це дозволяє створити ефективну основу для подальшої розробки та впровадження системи. Кожен із цих етапів розглянуто детально у наступних розділах.

У контексті сучасного розвитку технологій моделювання стає ключовим елементом будь-якої складної системи. Це включає створення моделей, які враховують всі бізнес-процеси, їхню взаємодію та залежності між ними. Завдяки цьому вдається заздалегідь оцінити можливі ризики та спланувати оптимальне рішення. Проектування доповнює моделювання, забезпечуючи структуру для реалізації кожного аспекту системи, від бази даних до взаємодії з користувачами. Таке поєднання методів дозволяє уникнути перевитрат часу і ресурсів у процесі розробки, а також підвищити якість кінцевого продукту.

2.1. Розробка моделі предметної області

Для початку було проведено аналіз стейкхолдерів цього проєкту. Основними стейкхолдерами системи є:

- Замовник: компанія або організація, яка ініціює розробку системи. Її завданням є визначення бізнес-цілей і контроль виконання проєкту. Замовник забезпечує фінансування, встановлює критерії успіху та слідкує за відповідністю вимог ключовим показникам ефективності (KPI).
- Кінцеві користувачі: аудитори, бухгалтери, фінансові аналітики. Їхні потреби включають зручність використання, швидкість доступу до звітів та прозорість даних. Вони є основними споживачами функціоналу системи.
- IT-команда: розробники, архітектори, адміністратори системи, які відповідають за створення, впровадження та підтримку технічної частини системи. Їхня роль полягає у забезпеченні стабільності роботи та відповідності технічним вимогам.
- Регулятори: контролюючі органи, які забезпечують відповідність роботи системи чинним законодавчим вимогам у сфері бухгалтерського обліку та аудиту. Їх інтереси полягають у забезпеченні відповідності стандартам, таким як IFRS і GAAP.

Для кожної групи стейкхолдерів було визначено специфічні очікування та функціональні потреби. Наприклад, замовник зацікавлений у забезпеченні високої надійності системи, а кінцеві користувачі очікують зручного інтерфейсу та можливості виконувати аудиторські завдання в автоматизованому режимі.

Регулятори натомість вимагають детального логування всіх операцій для забезпечення прозорості [7]. Функціональні вимоги системи були сформульовані на основі потреб стейкхолдерів і специфіки доменної області:

1. Реєстрація та управління транзакціями:

- Додавання фінансових операцій із вказанням деталей (сума, дата, джерело, статус).
- Перевірка транзакцій через смарт-контракти для виявлення невідповідностей.
- Забезпечення незмінності даних за допомогою блокчейну.

2. Формування звітності:

- Автоматична генерація фінансових звітів у форматах PDF та Excel.
- Підтримка адаптивних параметрів для формування звітів (період, тип операцій, контрагенти).
- Архівування звітів для подальшого аналізу.

3. Виявлення аномалій та шахрайства:

- Інтеграція алгоритмів машинного навчання для автоматичного аналізу транзакцій.
- Генерація сповіщень про підозрілі операції для подальшого перегляду.

4. Ролі та контроль доступу:

- Впровадження системи ролей (адміністратор, аудитор, бухгалтер).
- Налаштування доступу до функцій системи відповідно до ролі користувача.

5. Інтеграція з ERP-системами:

- Можливість отримання транзакцій та даних для звітності в автоматизованому режимі.

Нефункціональні вимоги визначають якісні характеристики системи, які забезпечують її стабільність і продуктивність:

1. Продуктивність:

- Система повинна обробляти до 5000 транзакцій за хвилину.

2. Доступність:

- Гарантована доступність системи не менше 99,9% часу.

3. Безпека:

- Шифрування даних (TLS, AES-256) для захисту під час передачі та зберігання.
- Багаторівнева автентифікація користувачів.

4. Масштабованість:

- Підтримка горизонтального масштабування з додаванням нових вузлів блокчейну.

5. Сумісність:

- Інтеграція з ERP-системами (SAP, Oracle) через API.

6. Відмовостійкість:

- Механізм резервного копіювання та відновлення даних у разі збоїв.

Нефункціональні вимоги були перевірені на відповідність сучасним стандартам (ISO 27001, GDPR) для забезпечення відповідності та міжнародної сумісності[8].

Для забезпечення структурованого підходу до аналізу вимог була створена матриця, яка включає перелік функціональних і нефункціональних вимог, їхній пріоритет та відповідність цілям стейкхолдерів.

Таблиця 2.1 — Матриця вимог

ID вимоги	**Тип вимоги**	**Опис**	**Пріоритет**	**Зацікавлені сторони**
FR1	Функціональна	Система повинна обробляти та зберігати фінансові транзакції з усіма деталями (дата, сума, сторони, мета).	Високий	Користувачі, аудитори, бухгалтери
FR2	Функціональна	Система повинна генерувати фінансові звіти відповідно до стандартів IFRS	Високий	Аудитори, аналітики
FR3	Функціональна	Система повинна надавати ролі користувачів та рівні доступу для адміністраторів,	Середній	Адміністратори, аудитори, аналітики

		аудиторів, бухгалтерів і фінансових аналітиків.		
FR4	Функціональна	Система повинна інтегруватися з ERP-системами, такими як SAP та Oracle, для синхронізації даних.	Високий	Адміністратори, розробники
FR5	Функціональна	Система повинна автоматично виявляти підозрілі транзакції та надсилати сповіщення відповідальним користувачам.	Високий	Адміністратори, аудитори
NFR1	Нефункціональна	Система повинна підтримувати обробку щонайменше 5000 транзакцій за хвилину.	Високий	Розробники, адміністратори
NFR2	Нефункціональна	Система повинна шифрувати всі дані під час передачі та зберігання з використанням алгоритму AES-256.	Високий	Розробники, команда з безпеки
NFR3	Нефункціональна	Система повинна забезпечувати доступність на рівні 99,9% та автоматичне відновлення протягом 30 секунд у разі збоїв.	Високий	Адміністратори, команда інфраструктури
NFR4	Нефункціональна	Система повинна автоматично виконувати резервне копіювання даних щонайменше раз на день.	Середній	Адміністратори, команда інфраструктури
NFR5	Нефункціональна	Інтерфейс системи повинен бути адаптивним і коректно працювати на мобільних пристроях.	Середній	Користувачі, розробники

Процес збору вимог дозволив сформулювати чітке уявлення про функціональність та якісні характеристики системи. На основі зібраних вимог створено основу для подальшого проектування та розробки системи. Залучення ключових стейкхолдерів на ранніх етапах допомогло уникнути розбіжностей у цілях проєкту та забезпечило узгодженість технічних і бізнес-параметрів.

2.2. Розробка бізнес-моделі роботи транзакцій

Бізнес-процеси системи були детально вивчені для забезпечення ефективного функціонування. Основними процесами є:

1. Обробка транзакцій:
   - Збір даних із ERP-систем.
   - Перевірка транзакцій за допомогою смарт-контрактів.
   - Збереження транзакцій у блокчейні.
2. Генерація звітів:
   - Збір даних із реєстрів блокчейну.
   - Формування звітів відповідно до заданих параметрів.
   - Експорт звітів для подальшого використання.
3. Виявлення аномалій:
   - Автоматичний аналіз транзакцій із використанням алгоритмів машинного навчання.
   - Ідентифікація підозрілих операцій.
   - Генерація сповіщень для аудиторів.

Для наочного представлення процесів були створені BPMN-діаграми. Процес перевірки транзакцій включає такі етапи:

- Отримання даних транзакції.

- Перевірка транзакцій через смарт-контракти.
- Відхилення невідповідних транзакцій із зазначенням причин.
- Збереження перевірених транзакцій у блокчейні.

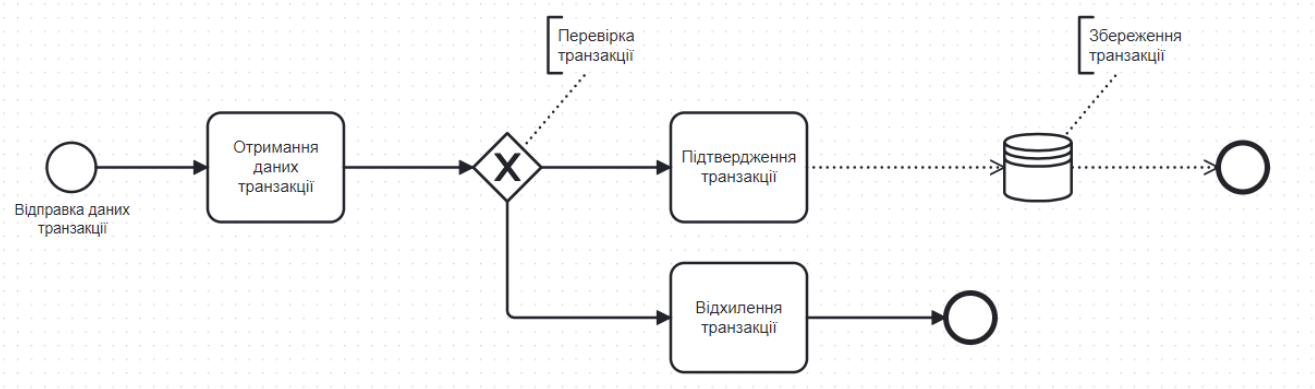

Рис. 2.1 — Діаграма бізнес-процесу перевірки транзакції.

Процес обробки транзакцій:

- Отримання даних через API із зовнішніх систем (ERP).
- Перевірка транзакцій на відповідність політикам та стандартам.
- Реєстрація транзакцій у блокчейні для забезпечення прозорості.
- Формування журналу дій для аудиту.

Процес генерації звітів:

- Збір даних із блокчейну та сховищ даних.
- Фільтрація та сортування за заданими параметрами (період, тип операції).
- Формування звіту у зручному для користувача форматі.
- Експорт звіту у PDF, Excel для подальшого використання.

Моделювання процесів дозволило чітко описати всі основні сценарії використання системи, що зменшує ризик пропущення важливих функцій під час розробки.

Актори в системі — це користувачі або зовнішні системи, які взаємодіють із функціональністю програми. Кожен актор має свій набір завдань, прав доступу та

сценаріїв використання. Нижче наведено детальний опис основних акторів системи аудиту.

1. Адміністратор (System Administrator)

Роль: Відповідає за технічне управління системою, її налаштування та забезпечення стабільної роботи.

Основні завдання:

- Створення та управління обліковими записами користувачів.
- Призначення ролей і прав доступу.
- Налаштування вузлів блокчейн-мережі.
- Контроль за безперебійною роботою системи.
- Моніторинг продуктивності та безпеки.
- Відновлення системи після збоїв.

Сценарії використання:

- "Створити обліковий запис користувача."
- "Налаштувати права доступу."
- "Переглянути журнал дій."
- "Виконати резервне копіювання."

2. Аудитор (Auditor)

Роль: Перевіряє фінансові операції, виявляє порушення та генерує аудиторські звіти.

Основні завдання:

- Аналіз транзакцій на відповідність внутрішнім правилам та регуляторним стандартам.
- Робота з підозрілими транзакціями.
- Генерація звітів про відповідність.
- Надання висновків керівництву.

Сценарії використання:

- "Перевірити транзакції."

- "Аналізувати підозрілі операції."
- "Генерувати звіт про відповідність."
- "Переглянути журнал дій користувачів."

3. Бухгалтер (Accountant)

Роль: Відповідає за введення, перевірку та обробку фінансових даних.

Основні завдання:

- Реєстрація нових транзакцій.
- Внесення змін до існуючих записів (за необхідності).
- Генерація стандартних фінансових звітів.
- Синхронізація даних із ERP-системами.

Сценарії використання:

- "Додати нову транзакцію."
- "Редагувати фінансові дані."
- "Генерувати стандартний фінансовий звіт."
- "Синхронізувати дані з ERP."

4. ERP-система (ERP System)

Роль: Зовнішня система, яка надає дані про транзакції для інтеграції із системою аудиту.

Основні завдання:

- Передача даних про транзакції через API.
- Синхронізація статусів транзакцій.
- Забезпечення відповідності між ERP і блокчейном.

Сценарії використання:

- "Передати дані про транзакції."
- "Синхронізувати дані з блокчейном."

Сценарії варіантів використання (Use Case Scenarios) визначають, як система буде використовуватися для виконання ключових завдань. Це допомагає описати взаємодію між користувачами та системою, а також зрозуміти, як

функціональні вимоги реалізуються у реальних умовах. Основні сценарії використання

1. Реєстрація транзакції:

Опис: Користувач (бухгалтер) додає нову фінансову операцію через веб-інтерфейс.

Актори: Бухгалтер, система API.

Кроки:

- Користувач вводить дані про транзакцію (сума, дата, контрагент, опис).
- Дані відправляються до API-шлюзу для валідації.
- Смарт-контракт перевіряє коректність транзакції.
- У разі успіху транзакція записується в блокчейн.
- Користувач отримує підтвердження успішної реєстрації.

Альтернативні сценарії:

- Якщо транзакція не відповідає бізнес-правилам, система повідомляє про помилку.

2. Генерація звіту:

Опис: Аудитор формує фінансовий звіт за заданими параметрами.

Актори: Аудитор, система API, сховище даних.

Кроки:

- Користувач обирає тип звіту (баланс, доходи, витрати).
- Вказує параметри звіту (період, контрагенти, типи операцій).
- Запит надсилається до API.
- API отримує дані з блокчейну та сховища даних.
- Сформований звіт відображається у веб-інтерфейсі.

Альтернативні сценарії:

- У разі відсутності даних за вибраними параметрами система повідомляє про це користувача.

3. Перевірка транзакцій:

Опис: Аудитор перевіряє транзакції на відповідність нормативним вимогам.

Актори: Аудитор, система API.

Кроки:

- Користувач обирає транзакції для перевірки.
- Дані про транзакції надходять із блокчейну.
- Система автоматично перевіряє транзакції через смарт-контракти.
- Аудитор отримує результати перевірки.

Альтернативні сценарії:

- Система сповіщає про підозрілі операції.

4. Виявлення аномалій:

Опис: Система аналізує дані для ідентифікації підозрілих операцій.

Актори: Система машинного навчання, API, аудитор.

Кроки:

- Система аналізує транзакції у фоновому режимі.
- Виявляє потенційно підозрілі операції.
- Надсилає сповіщення аудиторському відділу.
- Аудитор перевіряє аномальні транзакції вручну.

Альтернативні сценарії:

- Якщо система помилково ідентифікує операцію як підозрілу, аудитор має можливість додати її до списку виключень.

5. Управління ролями та доступом:

Опис: Адміністратор налаштовує ролі користувачів і права доступу.

Актори: Адміністратор, API, система контролю доступу.

Кроки:

- Адміністратор входить у веб-інтерфейс.
- Вибирає користувача або групу користувачів.
- Призначає або змінює права доступу.
- Система оновлює конфігурацію доступу в реальному часі.

Альтернативні сценарії:

- У разі конфлікту прав доступу система попереджає адміністратора.

Для візуалізації взаємодії акторів і системи створюється діаграма варіантів використання (Use Case Diagram). Вона відображає всі сценарії та зв'язки між ними.

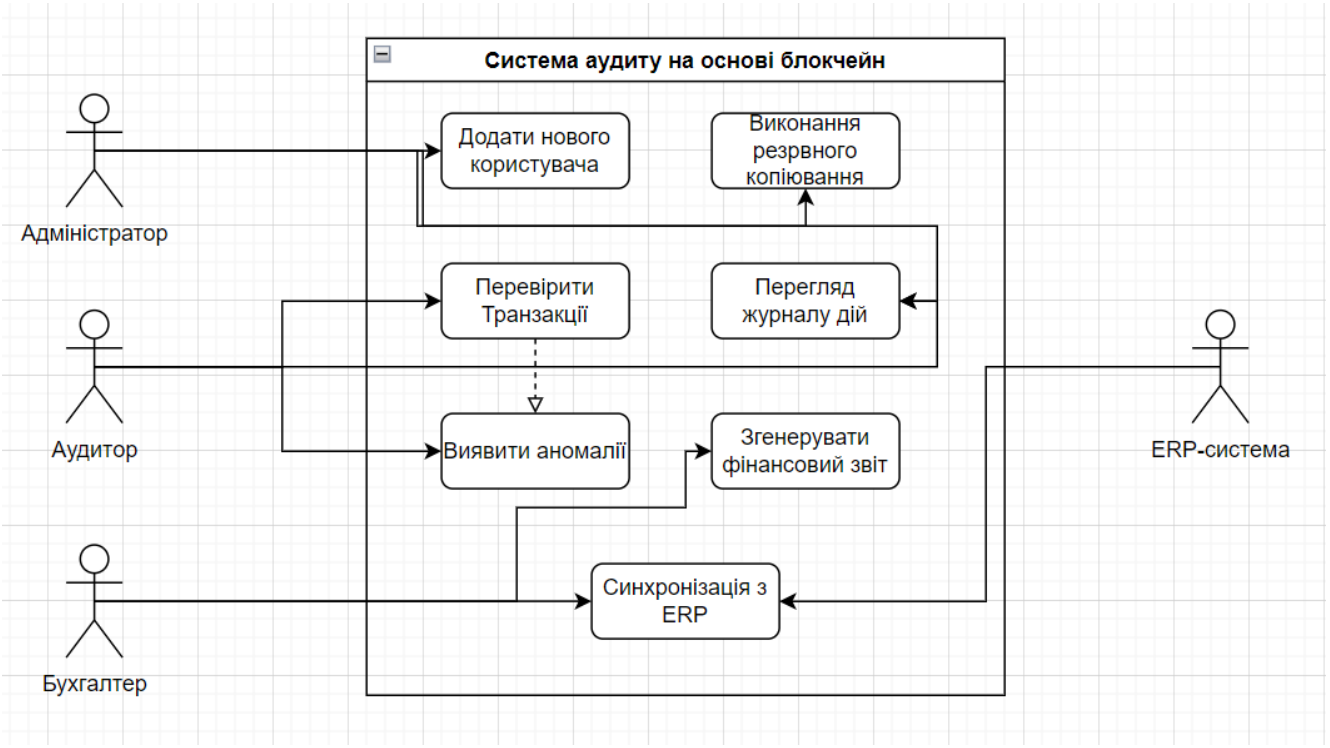

Рис. 2.2 — Діаграма варіантів використаня.

2.3. Проєктування архітектури блокчейн-аудиту

Система базується на подіє-орієнтованій архітектурі з використанням Hyperledger Fabric. Такий підхід забезпечує масштабованість, продуктивність і високу безпеку. Подіє-орієнтована архітектура дозволяє обробляти запити асинхронно, що особливо важливо для систем, які працюють із великими обсягами даних.

Основні компоненти архітектури:

1. Користувацький інтерфейс (UI):
   - Забезпечує доступ до функціоналу системи через веб-інтерфейс.
   - Підтримує створення запитів на звіти, перегляд транзакцій, управління ролями.
   - Взаємодіє із серверною частиною через REST API.
2. API-шлюз:
   - Приймає запити від користувачів і передає їх до відповідних компонентів системи.
   - Виконує валідацію даних та авторизацію запитів.
   - Забезпечує обробку помилок і логування.
3. Шина подій (Event Bus):
   - Координує обробку подій між компонентами.
   - Використовується для сповіщення про нові транзакції, генерацію звітів, сповіщення про аномалії.
4. Блокчейн-мережа:
   - Основа системи, що забезпечує реєстрацію транзакцій.
   - Використовує peer-вузли для зберігання даних і виконання смарт-контрактів.
   - Підтримує механізм консенсусу через orderer-вузли, які формують блоки.
5. Сховище даних:
   - Розроблене для зберігання аналітичних і агрегованих даних.
   - Реалізоване на основі CouchDB, що дозволяє виконувати складні запити до JSON-структур.
   - Включає кешування для підвищення швидкості роботи.

Архітектурна схема охоплює наступні компоненти:

Користувач: Через веб-інтерфейс надсилає запити на перевірку транзакцій або генерацію звітів.

API: Обробляє запити та взаємодіє з іншими частинами системи.

Блокчейн: Реєструє транзакції після їх перевірки смарт-контрактами.

Сховище: Надає доступ до аналітичних даних для генерації звітів.

Event Bus: Координує обробку подій між усіма компонентами.

Рис. 2.3 — Діаграма архітектури системи

Компоненти блокчейн-мережі:

1. Peer-вузли:
   - Виконують смарт-контракти, перевіряють транзакції.
   - Зберігають локальну копію розподіленого реєстру.
2. Orderer-вузли:
   - Відповідають за консенсус і впорядкування транзакцій у блоки.
   - Використовують алгоритм Raft для забезпечення надійності.
3. Certificate Authority (CA):
   - Управляє автентифікацією учасників та видачею цифрових сертифікатів.
4. Канали:
   - Забезпечують ізоляцію даних між групами учасників.
   - Використовуються для конфіденційної взаємодії між окремими компаніями.

Забезпечення безпеки:

- Шифрування: TLS і AES-256 для захисту даних при передачі та зберіганні.
- Аутентифікація: Використання цифрових сертифікатів, виданих CA.
- Контроль доступу: Реалізовано через ролі (адміністратор, аудитор, бухгалтер).
- Логування та моніторинг: Всі операції логуються для аудиту та відстеження аномалій.

Масштабованість та адаптивність

- Горизонтальне масштабування: Додавання нових peer-вузлів для збільшення продуктивності.
- Гнучкість у налаштуванні: Можливість налаштування приватних каналів для окремих учасників.
- Оновлення смарт-контрактів: Можливість змінювати бізнес-логіку без зупинки всієї мережі.

Система підтримує інтеграцію з зовнішніми ERP через API, що дозволяє автоматично отримувати транзакції та дані для звітності. Інтеграція забезпечує безшовну роботу між внутрішніми системами компанії та блокчейном.

Система базується на кількох ключових сутностях, які відображають основні бізнес-об'єкти, що беруть участь у процесах аудиту та бухгалтерського обліку. Кожна сутність має набір атрибутів, які визначають її властивості, і методи (функції), які реалізують поведінку об'єктів у системі. Додатково розглядаються приклади використання сутностей у реальних сценаріях, а також їх потенційна інтеграція з іншими компонентами системи.

Основні сутності:

1. Транзакція (Transaction):

Опис: Відображає фінансову операцію, яка реєструється, перевіряється та зберігається в системі для подальшого аналізу та аудиту.

Атрибути:

- transaction_id: Унікальний ідентифікатор транзакції.
- amount: Сума транзакції.
- date: Дата створення транзакції.
- description: Опис транзакції.
- status: Статус транзакції (наприклад, "підтверджено", "підозріло", "відхилено").
- initiator: Ідентифікатор користувача, який ініціював транзакцію.
- verification_details: Інформація про результати перевірки транзакції.
- related_documents: Посилання на додаткові документи, пов'язані з транзакцією.

Методи:

- validate(): Перевіряє транзакцію на відповідність бізнес-правилам та стандартам.
- mark_as_suspicious(): Позначає транзакцію як підозрілу, додаючи відповідні коментарі.
- approve(): Змінює статус транзакції на "підтверджено" після успішної перевірки.
- link_documents(documents): Додає пов'язані документи до транзакції для забезпечення додаткової прозорості.

2. Користувач (User):

Опис: Представляє обліковий запис користувача системи та відповідає за доступ до її функціоналу.

Атрибути:

- user_id: Унікальний ідентифікатор користувача.
- role: Роль користувача (адміністратор, аудитор, бухгалтер).
- email: Електронна пошта користувача.
- name: Ім'я користувача.
- status: Статус користувача (активний/деактивований).

- activity_log: Лог дій користувача в системі.

Методи:

- assign_role(new_role): Призначає роль користувачу.
- deactivate(): Деактивує обліковий запис.
- update_profile(details): Оновлює дані профілю.
- log_activity(action): Додає запис про дію користувача до журналу активності.

3. Звіт (Report):

Опис: Відображає агреговані дані за певний період для потреб аудиту або внутрішнього контролю.

Атрибути:

- report_id: Унікальний ідентифікатор звіту.
- type: Тип звіту (наприклад, "баланс", "звіт про доходи").
- created_by: Ідентифікатор користувача, який створив звіт.
- parameters: Параметри, за якими сформовано звіт.
- generation_date: Дата створення звіту.
- last_modified: Дата останньої модифікації звіту.
- attached_files: Додаткові файли, пов'язані зі звітом.

Методи:

- generate(params): Формує новий звіт на основі заданих параметрів.
- export(format): Експортує звіт у зазначений формат (PDF, Excel).
- attach_file(file): Додає файл до звіту для подальшого зберігання.

4. Система контролю доступу (AccessControl):

Опис: Відповідає за управління доступом користувачів до функцій системи, а також за збереження безпеки даних.

Атрибути:

- role_permissions: Список дозволів для кожної ролі.
- user_roles: Мапінг користувачів і їхніх ролей.

- audit_log: Лог усіх змін у правах доступу.

Методи:

- check_permission(user_id, action): Перевіряє, чи має користувач право виконати дію.
- assign_permission(role, permission): Призначає дозвіл ролі.
- remove_permission(role, permission): Видаляє дозвіл у ролі.
- log_change(change_details): Додає запис до журналу змін прав доступу.

Для ілюстрації зв'язків між сутностями буде створена ERD-діаграма (Entity-Relationship Diagram). Вона покаже, як сутності взаємодіють одна з одною і які атрибути/методи вони використовують для реалізації своїх функцій.

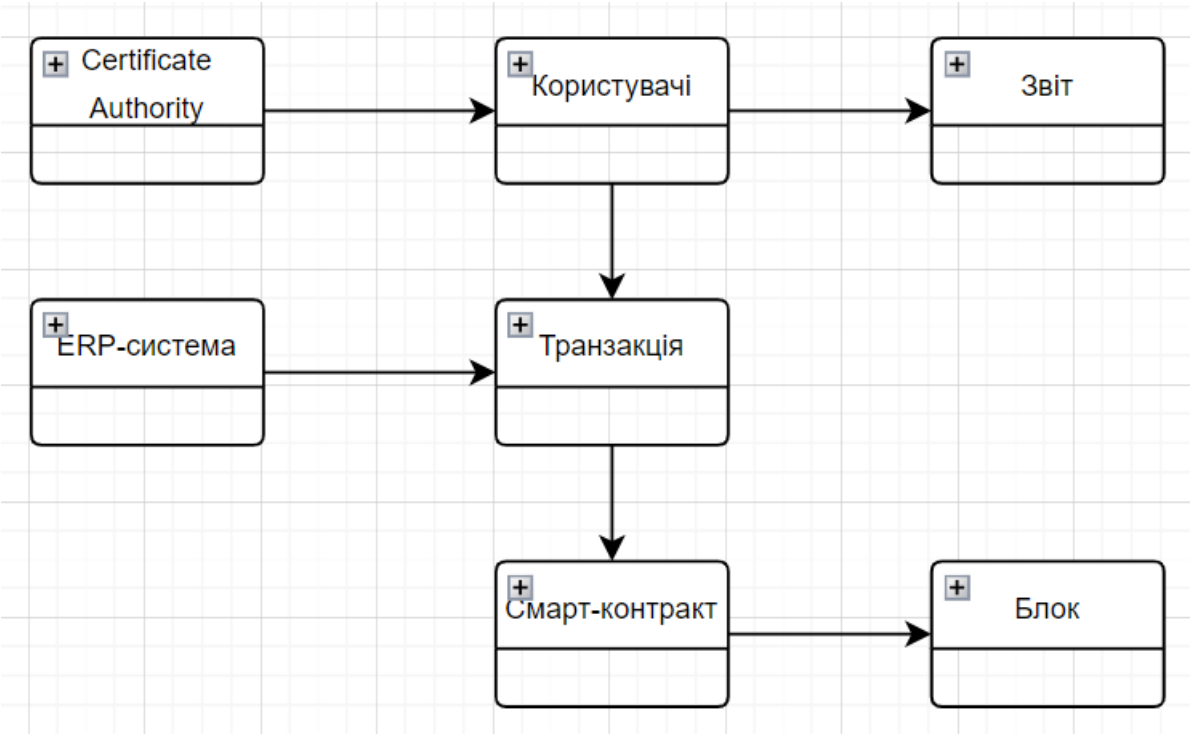

Рис. 2.4 — Діаграма взаємодії сутностей

В цій діаграмі наведено приклад взаємодії між сутностями:

- Користувач створює або переглядає транзакції, запитує звіти.
- Транзакція перевіряється за допомогою смарт-контракту.
- Смарт-контракт взаємодіє з блоком для запису результату транзакції.

- Блок формується з набору транзакцій через механізм консенсусу.
- Звіт генерується на основі даних, збережених у блокчейні.
- ERP-система надає дані, які стають транзакціями у блокчейні.
- CA забезпечує автентифікацію користувачів перед наданням доступу до системи.

Деталізація сутностей, їх атрибутів та методів забезпечує фундамент для проектування та реалізації системи. Чітке визначення кожного аспекту функціоналу дозволяє створити ефективний та зрозумілий код, що значно полегшує підтримку системи, її розвиток і інтеграцію з іншими інструментами [9].

Також для відображення взаємодії між об'єктами системи у вигляді послідовності повідомлень. Для системи, яка базується на технології блокчейн, діаграма має відображати процес виконання основних операцій, таких як створення транзакцій, перевірка даних, виконання смарт-контрактів та обробка результатів.

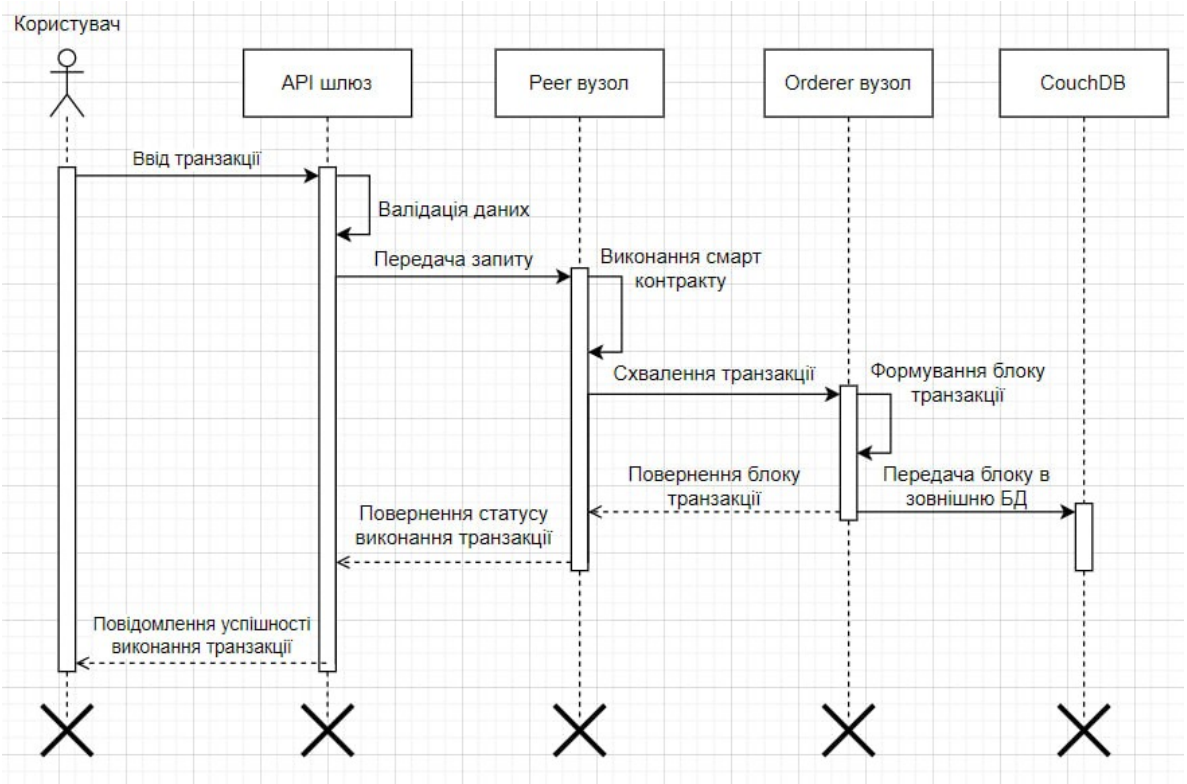

Рис. 2.5 — Діаграма послідовностей

3 КОНСТРУЮВАННЯ БЛОКЧЕЙН-ІНФРАСТРУКТУРИ І СИСТЕМИ АУДИТУ

Реалізація елементів системи є основним етапом розробки, на якому концептуальні ідеї та проєктні рішення перетворюються в реальний програмний продукт. На цьому етапі забезпечується технічна реалізація компонентів, що відповідають за зберігання, обробку та аналіз даних, а також за взаємодію з користувачами.

Для реалізації системи використовуються сучасні інструменти та технології, зокрема платформа Hyperledger Fabric, мови програмування Python і Node.js, база даних CouchDB та контейнеризація за допомогою Docker. Кожен компонент інтегрується в єдину архітектурну структуру, що забезпечує прозорість, надійність та масштабованість системи.

3.1. Реалізація ключових елементів

Підготовано необхідне програмне забезпечення, інструменти та структуру проекту для створення робочого середовища, яке підтримує Hyperledger Fabric, API та інтерфейс користувача [11].

Встановлення програмного забезпечення:

- Docker і Docker Compose: Інструменти для контейнеризації компонентів мережі. Встановлено Docker версії 20.10+ та Docker Compose 2.0+.
- Node.js: Використано для розробки API та смарт-контрактів. Встановлено Node.js LTS.
- Hyperledger Fabric CLI: Використано для управління мережею та взаємодії з нею. Встановлено офіційний Fabric CLI.

- Git: Налаштовано для керування версіями коду проекту.
- Підготовкad структури проекту: Створено директорію project-root/ для організації всіх компонентів. У межах project-root/ створено підкаталоги.

Встановлення Hyperledger Fabric: Завантажено та встановлено fabric-samples з офіційного репозиторію Hyperledger Fabric. Перевірено роботу тестової мережі (test-network), щоб переконатися в працездатності встановлених компонентів.

Додано основні файли конфігурації: docker-compose.yaml для запуску мережі Hyperledger Fabric у контейнерах. configtx.yaml і crypto-config.yaml для генерації артефактів мережі.

Створено папки для зберігання смарт-контрактів (blockchain/chaincode/), криптографічних матеріалів (blockchain/crypto-config/), а також логів і даних. Написано скрипти start-network.sh і stop-network.sh для автоматизації запуску та зупинки компонентів мережі.

Перевірено працездатність інстальованих компонентів:

- Запуск тестової мережі Fabric (test-network).
- Використання CLI для створення каналу та виконання тестових транзакцій.
- Переконалося, що Docker успішно запускає контейнери для Peer, Orderer, і CA.

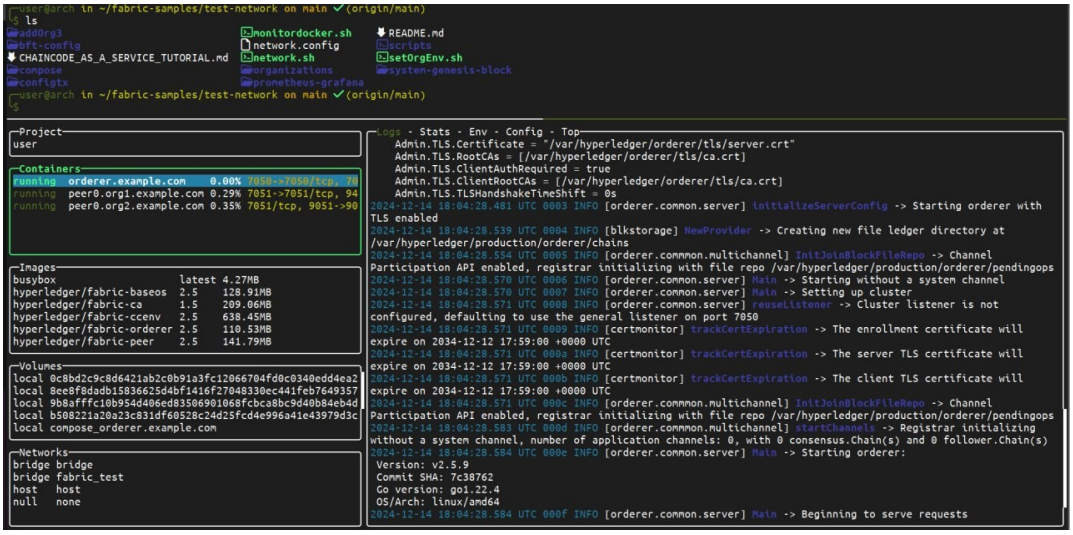

Рис. 3.1 — Запущена тестова мережа Hyperledger Fabric.

Розгорнуто базову блокчейн-мережу Hyperledger Fabric із налаштованими Peer-вузлами, Orderer-вузлами, каналами, криптографічними матеріалами та сертифікатами. Забезпечено функціональність мережі для виконання транзакцій і підготовки до впровадження смарт-контрактів.

Використано cryptogen для створення сертифікатів і ключів, які забезпечують автентифікацію вузлів і організацій у мережі. Створено файл crypto-config.yaml, який визначає структуру організацій (Org1, Org2), кількість вузлів і їхні сертифікати.

Генерація виконана командою:

```
cryptogen generate --config=crypto-config.yaml --output=./crypto-config
```

Налаштовано файл configtx.yaml, який визначає політики для каналу, порядок створення блоків і консенсус.

Визначено політики:

- Адміністративна політика: дозволяє лише адміністраторам змінювати конфігурацію мережі.
- Політика консенсусу: Raft як механізм впорядкування транзакцій.

Згенеровано конфігураційний блок для каналу:

```
configtxgen -profile TwoOrgsOrdererGenesis -channelID system-channel -outputBlock ./artifacts/genesis.block
```

Згенеровано транзакцію створення каналу:

```
configtxgen -profile TwoOrgsChannel -outputCreateChannelTx ./artifacts/channel.tx -channelID mychannel
```

Налаштовано docker-compose.yaml для автоматичного запуску вузлів мережі:

- Peer-вузли для Org1 і Org2.
- Orderer-вузли з підтримкою Raft.
- CA (Certificate Authority) для кожної організації.

Використано CLI для створення каналу:

```
peer channel create -o localhost:7050 -c mychannel -f ./artifacts/channel.tx --outputBlock ./artifacts/mychannel.block --tls --cafile /path/to/tls/ca.crt
```

Додано Peer-вузли до каналу:

```
peer channel join -b ./artifacts/mychannel.block
```

Результати:

Базова мережа Hyperledger Fabric створена:

- Дві організації (Org1 і Org2), кожна з двома Peer-вузлами.
- Orderer-вузли для впорядкування транзакцій.
- Сертифікати та ключі для забезпечення автентифікації.

Канал створено та налаштовано:

- Peer-вузли підключені до каналу mychannel.

Мережа готова до впровадження смарт-контрактів:

- Тестові транзакції підтвердили працездатність системи.

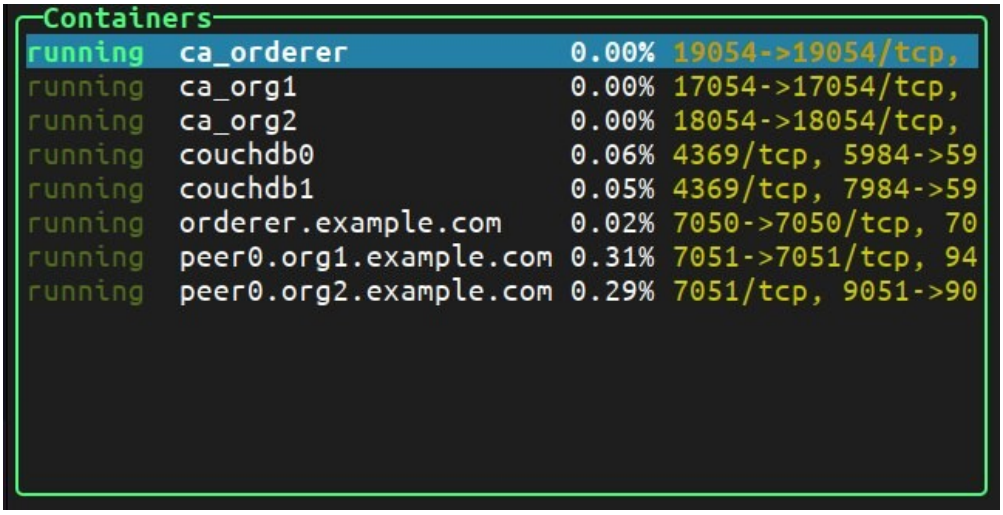

Рис.3.2 — Результат налаштування мережі

Розроблено та встановлено смарт-контракти, які реалізують бізнес-логіку системи аудиту бухгалтерського обліку. Смарт-контракти забезпечують перевірку транзакцій, дотримання правил бухгалтерського обліку та надання доступу до даних у мережі Hyperledger Fabric.

Визначено основні функції для обробки транзакцій:

- createTransaction: Додавання нової транзакції в блокчейн.
- readTransaction: Отримання деталей конкретної транзакції.
- updateTransaction: Оновлення даних транзакції.

- deleteTransaction: Видалення транзакції.
- queryAllTransactions: Повернення всіх транзакцій для звітності.

Реалізація контракту:

Лістинг 3.1

```
// SPDX-License-Identifier: Apache-2.0

'use strict';

const { Contract } = require('fabric-contract-api');

class ClearLedgerContract extends Contract {

    // Initialize ledger with sample data
    async initLedger(ctx) {
        console.info('Initializing the ledger with sample data...');
        const sampleTransactions = [
            {
                transactionId: 'TX001',
                amount: 500,
                description: 'Payment for services',
                timestamp: new Date().toISOString(),
                status: 'completed',
            },
            {
                transactionId: 'TX002',
                amount: 1500,
                description: 'Refund',
                timestamp: new Date().toISOString(),
                status: 'completed',
            }
        ];

        for (const transaction of sampleTransactions) {
            await ctx.stub.putState(transaction.transactionId, Buffer.from(JSON.stringify(transaction)));
            console.info(`Transaction ${transaction.transactionId} added to the ledger.`);
        }
    }

    // Create a new transaction
    async createTransaction(ctx, transactionId, amount, description) {
        const transactionExists = await this.transactionExists(ctx, transactionId);
        if (transactionExists) {
            throw new Error(`The transaction ${transactionId} already exists.`);
        }

        const transaction = {
            transactionId,
            amount: parseFloat(amount),
            description,
            timestamp: new Date().toISOString(),
            status: 'pending',
        };

        await ctx.stub.putState(transactionId, Buffer.from(JSON.stringify(transaction)));
        return JSON.stringify(transaction);
    }
```

```
    // Read an existing transaction
    async readTransaction(ctx, transactionId) {
        const transactionJSON = await ctx.stub.getState(transactionId);
        if (!transactionJSON || transactionJSON.length === 0) {
            throw new Error(`The transaction ${transactionId} does not exist.`);
        }
        return transactionJSON.toString();
    }

    // Update an existing transaction
    async updateTransaction(ctx, transactionId, amount, description, status) {
        const transactionJSON = await ctx.stub.getState(transactionId);
        if (!transactionJSON || transactionJSON.length === 0) {
            throw new Error(`The transaction ${transactionId} does not exist.`);
        }

        const transaction = JSON.parse(transactionJSON.toString());
        transaction.amount = parseFloat(amount);
        transaction.description = description;
        transaction.status = status;
        transaction.timestamp = new Date().toISOString();

        await ctx.stub.putState(transactionId, Buffer.from(JSON.stringify(transaction)));
        return JSON.stringify(transaction);
    }

    // Delete a transaction
    async deleteTransaction(ctx, transactionId) {
        const transactionExists = await this.transactionExists(ctx, transactionId);
        if (!transactionExists) {
            throw new Error(`The transaction ${transactionId} does not exist.`);
        }
        await ctx.stub.deleteState(transactionId);
        return `Transaction ${transactionId} has been deleted.`;
    }

    // Query all transactions
    async queryAllTransactions(ctx) {
        const startKey = '';
        const endKey = '';
        const iterator = await ctx.stub.getStateByRange(startKey, endKey);
        const results = [];

        while (true) {
            const res = await iterator.next();

            if (res.value) {
                const record = JSON.parse(res.value.value.toString('utf8'));
                results.push(record);
            }
```

Упаковано смарт-контракт у архів для встановлення. Створено нову папку blockchain/chaincode-packages/ для збереження архівів контрактів. Встановлено смарт-контрактів на Peer-вузли

Також розроблено та налаштувано додаткові компоненти системи, включаючи сховище даних для аналітики та звітності, інтеграцію із системою сертифікації (CA) та забезпечення взаємодії між компонентами через подіє-орієнтовану архітектуру.

Налаштування сховища даних: CouchDB використовується як state database для Peer-вузлів, забезпечуючи зберігання поточного стану транзакцій. Додано CouchDB до `docker-compose.yaml` для кожного Peer-вузла. Налаштовано користувацькі індекси для покращення швидкості пошуку даних:

Лістинг 3.2

```
{
    "index": {
      "fields": ["transactionId", "amount"]
    },
    "type": "json"
  }
```

Реалізовано запити для звітності:

Лістинг 3.3

```
const query = {
  selector: {
    amount: { "$gt": 1000 },
  },
  fields: ["transactionId", "amount", "description"],
  sort: [{ "amount": "desc" }],
};
const result = await couchDBClient.find(query);
```

Реалізація CA:

- Використано Hyperledger Fabric CA для управління сертифікатами.
- CA для кожної організації запущено як окремий контейнер у Docker:

Лістинг 3.4

```
ca.org1.example.com:
    container_name: ca.org1.example.com
    image: hyperledger/fabric-ca:latest
    environment:
      - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
      - FABRIC_CA_SERVER_CA_NAME=ca-org1
      - FABRIC_CA_SERVER_TLS_ENABLED=true
      - FABRIC_CA_SERVER_TLS_CERTFILE=/etc/hyperledger/fabric-ca-server-config/ca.org1.example.com-cert.pem
      - FABRIC_CA_SERVER_TLS_KEYFILE=/etc/hyperledger/fabric-ca-server-config/ca.org1.example.com-key.pem
    ports:
      - "7054:7054"
    volumes:
      - ./crypto-config/ca/ca.org1.example.com:/etc/hyperledger/fabric-ca-server-config
      - ./data/ca.org1:/etc/hyperledger/fabric-ca-server
```

Реалізовано функції для реєстрації користувачів і видачі сертифікатів через `fabric-ca-client`:

Лістинг 3.5

```
const { FabricCAServices } = require('fabric-ca-client');

const caService = new FabricCAServices('http://localhost:7054');
const enrollment = await caService.enroll({ enrollmentID: 'admin', enrollmentSecret: 'adminpw' });
console.log('Certificate issued:', enrollment.certificate);
```

Реалізація шини подій (Event Bus):

Впроваджено RabbitMQ для маршрутизації подій між компонентами. Клієнти публікують події (наприклад, створення транзакцій), які передаються підписникам [12].

Лістинг 3.6

```
const amqp = require('amqplib');
const queue = 'transaction_events';

async function publishEvent(event) {
  const connection = await amqp.connect('amqp://localhost');
  const channel = await connection.createChannel();
  await channel.assertQueue(queue);
  channel.sendToQueue(queue, Buffer.from(JSON.stringify(event)));
  console.log('Event published:', event);
}
```

Серверні компоненти підписуються на події RabbitMQ, щоб виконувати бізнес-логіку:

Лістинг 3.7

```
async function consumeEvents() {
    const connection = await amqp.connect('amqp://localhost');
    const channel = await connection.createChannel();
    await channel.assertQueue(queue);
    channel.consume(queue, (msg) => {
      console.log('Event received:', JSON.parse(msg.content.toString()));
      channel.ack(msg);
    });
  }
  consumeEvents();
```

Для підвищення прозорості та відповідності стандартам аудиту в систему додано механізм аудиторського журналу, який забезпечує детальне відстеження всіх операцій, включаючи їх походження, зміни та результат перевірки. Це важливий аспект для забезпечення довіри до даних і автоматизації процесів аудиту.

Реалізовано механізм, який дозволяє:

- Відстежувати всі операції, виконані в системі (додавання транзакцій, оновлення, видалення).
- Генерувати детальний аудиторський звіт, що відповідає міжнародним стандартам (наприклад, ISO 19011).
- Забезпечити автоматичний аналіз аномалій у транзакціях для спрощення роботи аудиторівх [13].

У смарт-контракт додано функцію для запису аудиторських даних:

Лістинг 3.8

```
class AuditContract extends Contract {
  async logOperation(ctx, operationType, transactionId, details) {
    const auditLog = {
      operationType,
      transactionId,
      details,
      timestamp: new Date().toISOString(),
      performedBy: ctx.clientIdentity.getID(),
    };

    const auditLogKey = `AUDIT_${transactionId}_${Date.now()}`;
    await ctx.stub.putState(auditLogKey, Buffer.from(JSON.stringify(auditLog)));
    return auditLog;
  }
}
```

Впроваджено механізм перевірки транзакцій для виявлення аномальних операцій:

Лістинг 3.9

```
async function detectAnomalies() {
  const query = {
    selector: { amount: { "$gt": 10000 } },
    fields: ["transactionId", "amount", "timestamp"],
  };

  const anomalies = await couchDBClient.find(query);
  anomalies.forEach(anomaly => {
    console.log(`Anomaly detected: ${JSON.stringify(anomaly)}`);
  });
}
```

API-ендпоінт для генерації звітів:

Лістинг 3.10

```
router.get('/audit/report', async (req, res) => {
    const query = {
      selector: { operationType: { "$exists": true } },
    };
    const auditLogs = await couchDBClient.find(query);
    res.status(200).json(auditLogs);
  });
```

Звіт у форматі JSON, CSV або PDF містить:

- Типи операцій.
- Відповідальних користувачів.
- Час виконання.
- Деталі змін.

Реалізовано сторінку "Audit Logs" у веб-інтерфейсі:

- Відображається таблиця з усіма операціями.
- Підсвічуються аномалії для швидкої перевірки.

Таблиця 3.1.1 — Приклад таблиці звіту транзакцій

Operation Type	Transaction ID	Performed By	Timestamp	Details
Create	TX123	User:Org1Admin	2024-12-14 10:00	{ "amount": 500 }
Update	TX124	User:Org2Auditor	2024-12-14 11:15	{ "amount": 700 }

3.2 Розробка GUI

Розробка API-шлюзу: У файлі blockchainService.js реалізовано підключення до Hyperledger Fabric через SDK:

Лістинг 3.11

```
const { Gateway, Wallets } = require('fabric-network');
const path = require('path');
const fs = require('fs');

async function connectToNetwork() {
  const ccpPath = path.resolve(__dirname, '../connection-profile.json');
  const ccp = JSON.parse(fs.readFileSync(ccpPath, 'utf8'));
  const walletPath = path.join(__dirname, '../wallet');
  const wallet = await Wallets.newFileSystemWallet(walletPath);
  const gateway = new Gateway();
  await gateway.connect(ccp, { wallet, identity: 'appUser', discovery: { enabled: true, asLocalhost: true } });
  const network = await gateway.getNetwork('mychannel');
  const contract = network.getContract('mychaincode');
  return contract;
}
module.exports = { connectToNetwork };
```

У файлі routes/transactions.js додано маршрути для CRUD-операцій:

Лістинг 3.12

```
const express = require('express');
const { createTransaction, getTransaction } = require('../controllers/transactionController');
const router = express.Router();

router.post('/transactions', createTransaction);
router.get('/transactions/:id', getTransaction);

module.exports = router;
```

У файлі transactionController.js реалізовано логіку для додавання та отримання транзакцій:

Лістинг 3.13

```
const express = require('express');
const { createTransaction, getTransaction } = require('../controllers/transactionController');
const router = express.Router();

router.post('/transactions', createTransaction);
router.get('/transactions/:id', getTransaction);

module.exports = router;


const { connectToNetwork } = require('../services/blockchainService');

async function createTransaction(req, res) {
  const { transactionId, amount, description } = req.body;
  const contract = await connectToNetwork();
  await contract.submitTransaction('createTransaction', transactionId, amount, description);
  res.status(200).send('Transaction created successfully');
}

async function getTransaction(req, res) {
  const { id } = req.params;
  const contract = await connectToNetwork();
  const result = await contract.evaluateTransaction('readTransaction', id);
  res.status(200).json(JSON.parse(result.toString()));
}

module.exports = { createTransaction, getTransaction };
```

Користувацький інтерфейс реалізується на базі React — популярної бібліотеки для створення динамічних веб-додатків. React забезпечує компонентний підхід до розробки, що дозволяє легко масштабувати та підтримувати проект. Інтерфейс розробляється для виконання наступних ключових функцій:

- Введення та реєстрація транзакцій.
- Перегляд журналу аудиту та історії транзакцій.
- Генерація та перегляд фінансових звітів.
- Управління ролями та доступами.

Реалізація основних компонентів:

Форма додавання транзакцій дозволяє користувачам створювати нові записи, які зберігаються у блокчейні. Реалізація включає:

- Поля для вводу ID транзакції, суми, опису та інших атрибутів.

- Валідацію введених даних на стороні клієнта.
- Інтеграцію з API для відправки даних до смарт-контрактів.

Лістинг 3.14

```
const AddTransaction = () => {
  const [transactionId, setTransactionId] = useState('');
  const [amount, setAmount] = useState('');
  const [description, setDescription] = useState('');

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      await axios.post('http://localhost:4000/transactions', {
        transactionId,
        amount,
        description,
      });
      alert('Transaction added successfully');
    } catch (error) {
      console.error(error);
      alert('Error adding transaction');
    }
  };

  return (
    <form onSubmit={handleSubmit}>
      <input
        type="text"
        placeholder="Transaction ID"
        value={transactionId}
        onChange={(e) => setTransactionId(e.target.value)}
        required
      />
      <input
        type="number"
        placeholder="Amount"
        value={amount}
        onChange={(e) => setAmount(e.target.value)}
        required
      />
      <textarea
        placeholder="Description"
        value={description}
        onChange={(e) => setDescription(e.target.value)}
      />
      <button type="submit">Add Transaction</button>
    </form>
  );
};

export default AddTransaction;
```

Компонент для перегляду транзакцій відображає список транзакцій, збережених у системі. Інтеграція з API дозволяє отримувати дані про всі транзакції з блокчейну та відображати їх у таблиці.

Лістинг 3.15:

```
import React, { useEffect, useState } from 'react';
import axios from 'axios';

const TransactionsList = () => {
  const [transactions, setTransactions] = useState([]);

  useEffect(() => {
    const fetchTransactions = async () => {
      try {
        const response = await axios.get('http://localhost:4000/transactions');
        setTransactions(response.data);
      } catch (error) {
        console.error('Error fetching transactions:', error);
      }
    };

    fetchTransactions();
  }, []);

  return (
    <table>
      <thead>
        <tr>
          <th>Transaction ID</th>
          <th>Amount</th>
          <th>Description</th>
          <th>Timestamp</th>
        </tr>
      </thead>
      <tbody>
        {transactions.map((tx) => (
          <tr key={tx.transactionId}>
            <td>{tx.transactionId}</td>
            <td>{tx.amount}</td>
            <td>{tx.description}</td>
            <td>{new Date(tx.timestamp).toLocaleString()}</td>
          </tr>
        ))}
      </tbody>
    </table>
  );
};

export default TransactionsList;
```

Компонент для перегляду аудиторського журналу дозволяє відображати всі події, зареєстровані в системі. Для кожної операції виводиться тип, ID транзакції, дата та користувач, який виконав операцію.

Лістинг 3.16

```
import React, { useEffect, useState } from 'react';
import axios from 'axios';

const AuditLog = () => {
  const [logs, setLogs] = useState([]);

  useEffect(() => {
    const fetchLogs = async () => {
      try {
        const response = await axios.get('http://localhost:4000/audit/logs');
        setLogs(response.data);
      } catch (error) {
        console.error('Error fetching audit logs:', error);
      }
    };

    fetchLogs();
  }, []);

  return (
    <div>
      <h2>Audit Logs</h2>
      <ul>
        {logs.map((log, index) => (
          <li key={index}>
            {`${log.operationType} - ${log.transactionId} - ${log.performedBy} - ${new Date(log.timestamp).toLocaleString()}`}
          </li>
        ))}
      </ul>
    </div>
  );
};

export default AuditLog;
```

Компонент для управління звітами дозволяє користувачам вибирати параметри звітів, такі як період чи тип транзакцій, та отримувати згенерований файл у форматі PDF або CSV.

Лістинг 3.17

```
import React, { useState } from 'react';
import axios from 'axios';

const GenerateReport = () => {
  const [startDate, setStartDate] = useState('');
  const [endDate, setEndDate] = useState('');

  const handleGenerate = async () => {
    try {
      const response = await axios.get(`http://localhost:4000/reports?start=${startDate}&end=${endDate}`);
      const blob = new Blob([response.data], { type: 'application/pdf' });
      const link = document.createElement('a');
      link.href = window.URL.createObjectURL(blob);
      link.download = 'report.pdf';
      link.click();
    } catch (error) {
      console.error('Error generating report:', error);
    }
  };

  return (
    <div>
      <input type="date" value={startDate} onChange={(e) => setStartDate(e.target.value)} />
      <input type="date" value={endDate} onChange={(e) => setEndDate(e.target.value)} />
      <button onClick={handleGenerate}>Generate Report</button>
    </div>
  );
};

export default GenerateReport;
```

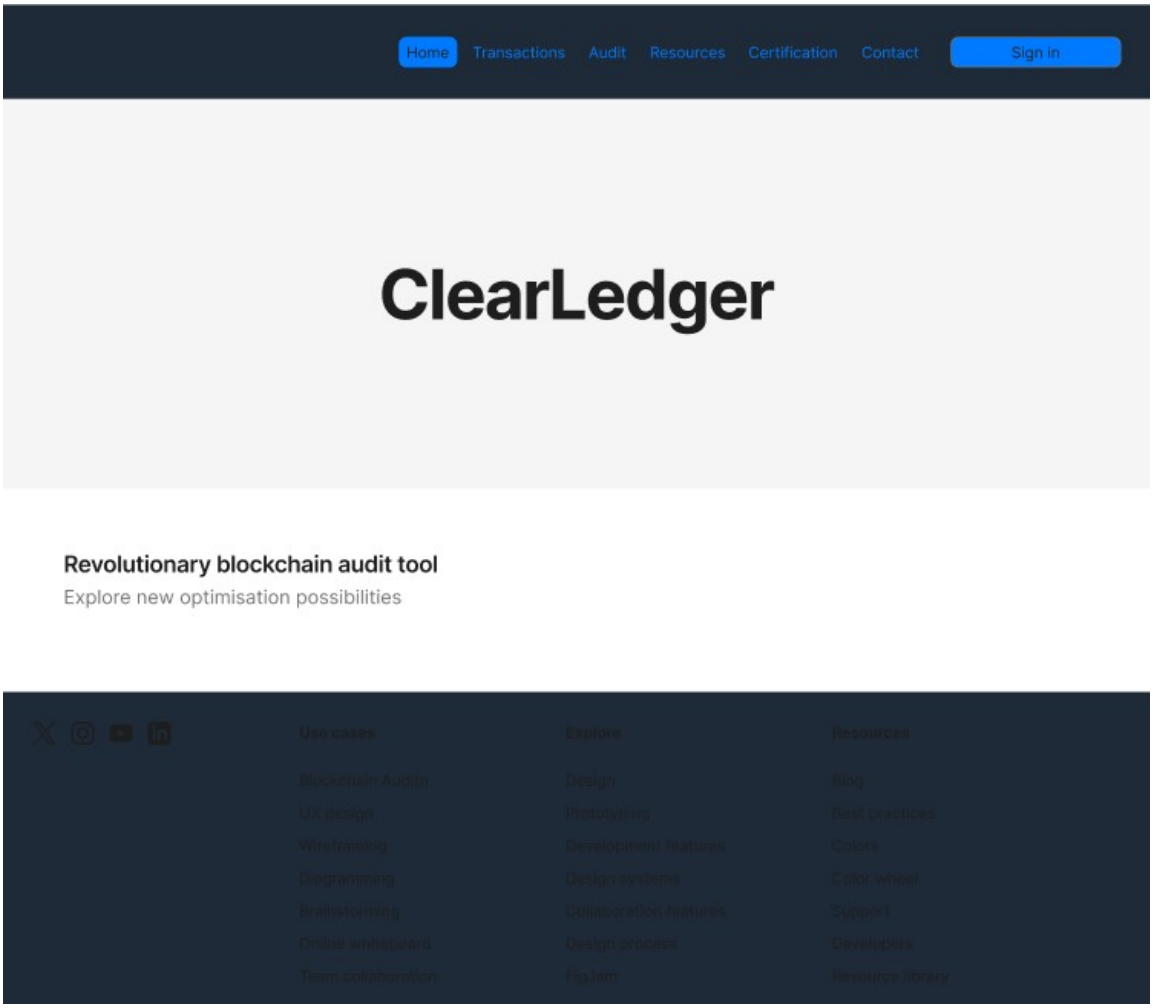

Рис. 3.1 — Домашня сторінка інтерфейсу.

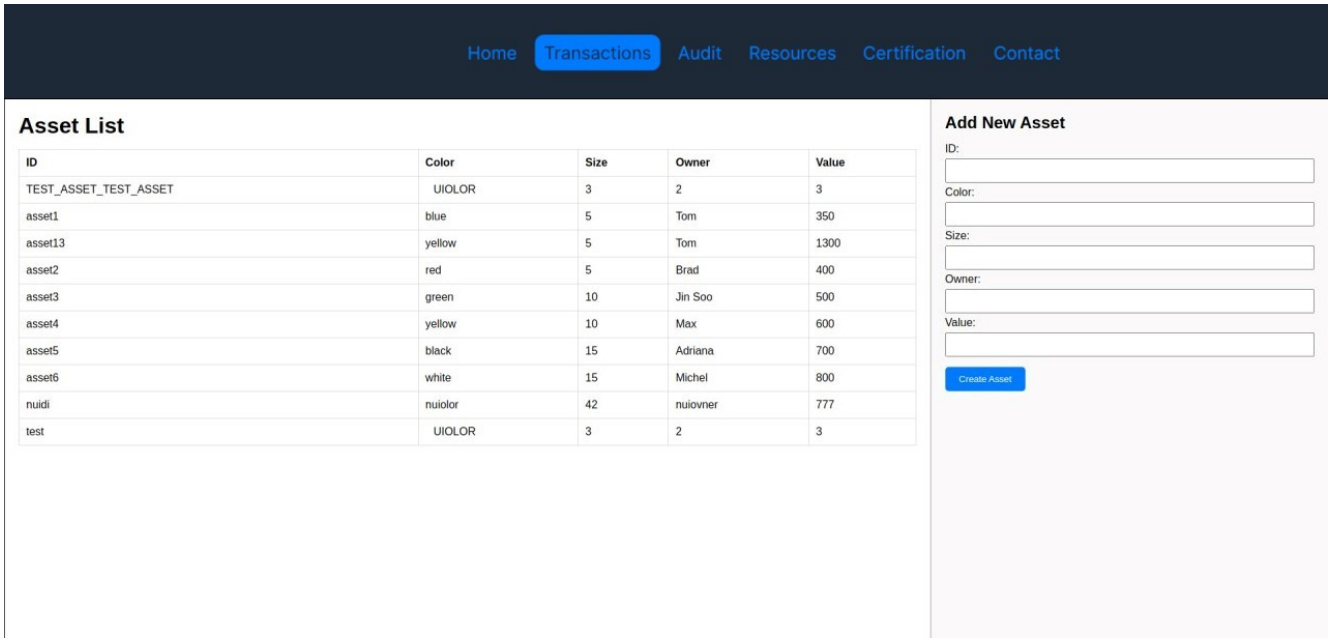

Рис. 3.2 — Меню додавання транзакцій.

3.3 Тестування програмного забезпечення та оцінка якості

Функціональне тестування:

Виконано тестування Peer- і Orderer-вузлів:

- Peer-вузли: перевірено виконання смарт-контрактів, коректність збереження транзакцій у реєстрі.
- Orderer-вузли: протестовано впорядкування транзакцій і формування блоків.

Методи тестування:

- Створено тестові транзакції
- Виконано зчитування даних для перевірки

Тестування API-шлюзу:

Перевірено маршрути API для CRUD-операцій:

- POST `/transactions`: створення транзакції.
- GET `/transactions/:id`: отримання транзакції.
- GET `/transactions`: отримання списку транзакцій.
- Використано Postman для перевірки роботи API з реальними даними.

Тестування UI:

- Проведено ручне тестування веб-інтерфейсу:
- Додано тестову транзакцію через форму.
- Перевірено правильність відображення списку транзакцій.
- Переконанося, що інтерфейс коректно інтегрований із API.

Інтеграційне тестування:

- Тестування взаємодії між компонентами:
- Виконано інтеграцію між API, Hyperledger Fabric і CouchDB:
- Дані з API передаються до смарт-контрактів через SDK.
- Peer-вузли зберігають стан транзакцій у CouchDB.
- API успішно отримує дані з CouchDB для відображення в UI.

- Виявлено та усунено помилки під час передачі великих обсягів даних.

Тестування подіє-орієнтованої архітектури:

- Перевірено роботу RabbitMQ:
- Події успішно генеруються при створенні транзакцій.
- Обробники подій отримують і виконують необхідні дії.

Тестування CA:

- Перевірено автентифікацію користувачів і вузлів через сертифікати:
- Peer- і Orderer-вузли коректно взаємодіють у мережі.
- Старі сертифікати відкликано через CRL.

Навантажувальне тестування:

Перевірка продуктивності мережі:

- Використано інструменти для генерації транзакцій:
- Створено понад 1000 транзакцій із різними параметрами.
- Оцінено час виконання транзакцій і завантаження Peer- і Orderer-вузлів.

Результати:

- Система обробляє 4500 транзакцій за хвилину при піковому навантаженні.
- Виконано тестування API під навантаженням за допомогою Apache JMeter:
- Відправлено 100 запитів в секунду на створення та зчитування транзакцій.
- Система демонструє стабільну роботу без затримок.
- Перевірено роботу інтерфейсу при одночасному використанні понад 50 користувачів:
- Інтерфейс залишається чутливим і стабільним.
- Всі транзакції коректно відображаються.

Усунення помилок:

- Виправлено некоректну обробку відсутніх даних у смарт-контрактах.
- Додано обробку помилок у маршрутах API.

Оптимізація

- Поліпшено продуктивність CouchDB через оптимізацію індексів.

- Оптимізовано обробку подій RabbitMQ для зменшення затримок.

3.4 Результати розробки

Результати розробки системи аудиту бухгалтерського обліку на основі блокчейн-технологій свідчать про успішне досягнення поставлених цілей та демонструють ефективність впровадження сучасних інновацій у фінансову сферу. Система була розроблена для вирішення ключових проблем, пов'язаних із прозорістю, точністю та безпекою фінансових даних, і забезпечує новий рівень автоматизації аудиторських процесів [14]. Застосування Hyperledger Fabric дозволило реалізувати розподілений реєстр, який гарантує незмінність даних і прозорість транзакцій, що є критично важливим для сучасних бухгалтерських систем.

Важливою перевагою є здатність системи до інтеграції з існуючими ERP-рішеннями, що дозволяє безшовно впроваджувати її в бізнес-процеси компаній. Крім того, автоматизовані механізми перевірки транзакцій і виявлення аномалій забезпечують суттєве скорочення часу та ресурсів, які витрачаються на аудит. Система не лише відповідає сучасним стандартам у сфері бухгалтерського обліку та аудиту, а й демонструє можливість подальшого вдосконалення та адаптації до нових викликів і потреб. Розроблено повноцінну блокчейн-мережу:

Система ClearLedger виконує наступні функції:

Створення, зберігання та управління фінансовими транзакціями:

- Користувачі створюють нові транзакції, зазначаючи унікальний ідентифікатор, суму, опис і час операції.
- Усі транзакції зберігаються в блокчейні, що гарантує їх незмінність і доступність для аудиторів.

- Система забезпечує можливість оновлення та видалення транзакцій із записом усіх змін у журналі аудиту.

Реєстрація всіх операцій у незмінному журналі аудиту:

- Кожна дія користувача (створення, оновлення, видалення транзакції) автоматично записується в аудиторський журнал.
- Журнал доступний для перегляду, що дозволяє відстежувати історію змін і дій.
- Автоматична перевірка транзакцій:
- Система перевіряє транзакції на відповідність правилам бухгалтерського обліку (GAAP, IFRS).
- Виявляє аномалії, такі як нетипові суми або підозрілі послідовності транзакцій.

Генерація фінансових і аудиторських звітів:

- Система формує звіти про фінансові операції, транзакції та результати аудиторських перевірок.
- Звіти доступні для експорту у форматах PDF і CSV.

Забезпечення безпеки даних:

- Використовує шифрування для зберігання даних та передачі між вузлами системи.
- Забезпечує автентифікацію користувачів через цифрові сертифікати.
- Контролює доступ до функцій системи на основі ролей (бухгалтер, аудитор, адміністратор).

Зберігання та управління транзакційними даними:

- Стан транзакцій оновлюється в реальному часі та зберігається у CouchDB для швидкого доступу.
- Дані залишаються доступними навіть у разі аварійних ситуацій завдяки резервуванню.

Інтеграція з ERP-системами:

- Система синхронізує фінансові дані з ERP-рішеннями, забезпечуючи безшовний обмін інформацією.
- Підтримує автоматичне оновлення транзакцій між блокчейном і зовнішніми системами.

Обробка подій у реальному часі:

- Використовує RabbitMQ для оповіщення про важливі події, такі як підозрілі транзакції або помилки системи.
- Оперативно інформує користувачів про критичні події.

Моніторинг та аналітика:

- Система надає дашборд для перегляду ключових метрик, таких як загальна кількість транзакцій, завершені аудити, виявлені аномалії.
- Підтримує запити для аналізу великих обсягів даних.

Додавання нових вузлів (Peer, Orderer) без шкоди для продуктивності забезпечує стабільну роботу навіть під високим навантаженням завдяки механізмам консенсусу Raft.

Розроблена система повністю відповідає цілям проекту, забезпечуючи безпечний, масштабований і прозорий процес управління фінансовими даними. Система готова до використання кінцевими користувачами та може бути адаптована для інших бізнес-процесів за рахунок модульної архітектури та підтримки смарт-контрактів.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОХОРОНА ПРАЦІ

4.1. Охорона праці

У сучасних умовах стрімкого розвитку цифрових технологій дедалі актуальнішим стає впровадження блокчейн-технологій у процеси аудиту бухгалтерського обліку. Блокчейн забезпечує прозорість, надійність та достовірність даних, що суттєво підвищує ефективність та довіру до результатів аудиту. Однак робота з новими інформаційними технологіями вимагає належної організації умов праці, захисту здоров’я працівників та дотримання чинних нормативних документів у сфері охорони праці України. Безпека праці під час реалізації системи аудиту з використанням блокчейн-технологій включає заходи щодо забезпечення здоров’я працівників, організації ергономічних робочих місць, профілактики професійних захворювань та впровадження кібербезпеки як обов’язкової складової охорони праці.

Загальна характеристика умов праці при роботі з блокчейн-системами Сучасні блокчейн-системи аудиту бухгалтерського обліку побудовані на основі інтегрованих цифрових платформ, які потребують великої обчислювальної потужності та постійного доступу до інтернету. Працівники, які працюють з такими системами, змушені виконувати тривалі завдання за допомогою комп’ютерів, мережевого обладнання та програмного забезпечення. Такі умови праці створюють низку потенційних ризиків:

- Фізичні та психофізіологічні перевантаження через багатогодинну роботу за монітором;
- Вплив електромагнітного випромінювання від обладнання та серверів;
- Високе навантаження на зір через тривалу роботу з даними;
- Недостатній мікроклімат у приміщеннях, що може призвести до головного болю та зниження працездатності.

Згідно з Законом України “Про охорону праці” (№ 2694-XII від 14 жовтня 1992 р.), роботодавець повинен вживати всіх необхідних заходів для усунення шкідливих факторів та забезпечення безпеки працівників. У випадку систем блокчейн це особливо важливо, оскільки неперервна робота у цифровому середовищі створює ризики розвитку професійних захворювань.

Організація робочих місць для працівників, що працюють із блокчейн-системами, повинна відповідати вимогам Державних санітарних норм ДСанПіН 3.3.2.007-98. Основні вимоги включають:

- Ергономічність робочого місця: робоча поверхня столу повинна бути висотою 680-750 мм, а відстань від очей працівника до екрана монітора – 50-70 см. Кут нахилу монітора має забезпечувати мінімальне навантаження на шию та спину;
- Освітлення приміщень: освітленість робочої зони має бути в межах 400-500 лк, а світло повинно рівномірно розподілятися, щоб уникати утворення відблисків;
- Мікроклімат приміщень: температура повітря повинна становити 20-24°C, відносна вологість – 40-60%, швидкість руху повітря – не більше 0,2 м/с;
- Профілактика навантаження на зір: працівники повинні робити 15-хвилинні перерви кожні 2 години роботи за комп’ютером для виконання вправ для очей і відновлення працездатності.
- Усі комп’ютери повинні мати сертифіковані антиблікові екрани та можливість регулювання яскравості й контрастності зображення. Робочі місця слід оснащувати ортопедичними кріслами з підтримкою поперекового відділу хребта.

Оскільки системи блокчейн використовують цифрові технології, забезпечення кібербезпеки є важливою частиною охорони праці. Недотримання правил кіберзахисту може призвести до витоку конфіденційної інформації, що, у свою чергу, створює психологічний стрес для працівників та ризик їхньої

відповідальності. Відповідно до ДСТУ ISO/IEC 27001:2015, заходи з кібербезпеки включають:

- Встановлення ліцензованого програмного забезпечення з регулярними оновленнями;
- Захист робочих місць від несанкціонованого доступу через системи багаторівневої аутентифікації;
- Використання антивірусних програм та фаєрволів для захисту від шкідливого ПЗ;
- Проведення регулярних навчань з кібергігієни для всіх працівників.

Забезпечення кібербезпеки не тільки захищає дані компанії, але й знижує рівень стресу у працівників, оскільки вони впевнені у захисті своєї інформації.

Згідно з ДСН 3.3.6.042-99, робочі місця з комп'ютерами та серверним обладнанням повинні бути захищені від надмірного електромагнітного випромінювання. Працівники, що працюють із блокчейн-системами, зазнають постійного впливу електромагнітних хвиль, які можуть викликати:

- Головний біль та запаморочення;
- Погіршення якості сну;
- Підвищену втому та дратівливість.

Для захисту необхідно:

- Використовувати екрановані монітори та кабелі з коефіцієнтом послаблення до 50 дБ;
- Розташовувати комп'ютерне обладнання на відстані не менше 1,5 м одне від одного;
- Обмежувати час безперервної роботи з комп'ютером до 2 годин, роблячи регулярні перерви.

Нормативно-правова база охорони праці є основою для забезпечення безпеки працівників, що працюють у сфері цифрових технологій. Основними документами, що регулюють питання охорони праці в Україні, є:

Закон України “Про охорону праці” (№ 2694-XII від 14 жовтня 1992 р.): встановлює основні положення та обов’язки роботодавця і працівників у створенні безпечних умов праці. Закон зобов’язує роботодавців проводити оцінку ризиків, навчання з охорони праці та регулярний контроль умов на робочих місцях.

Кодекс законів про працю України: забезпечує право кожного працівника на безпечні та здорові умови праці. Роботодавці зобов’язані забезпечувати дотримання всіх норм охорони праці та компенсувати втрати у разі порушень.

Державні санітарні норми ДСанПіН 3.3.2.007-98 визначають санітарно-гігієнічні вимоги до роботи з відеодисплейними терміналами. Документ регулює параметри освітлення, мікроклімату, ергономіки та тривалості безперервної роботи з ПК.

ДСТУ ISO/IEC 27001:2015 визначає вимоги до систем управління інформаційною безпекою. Документ є обов’язковим для забезпечення кібербезпеки в компаніях, які працюють з конфіденційними цифровими даними.

ДСН 3.3.6.042-99 встановлює допустимі рівні електромагнітного випромінювання та методи його контролю на робочих місцях.

Наказ МОЗ України № 246 регулює порядок проведення обов’язкових медичних оглядів для працівників, які працюють у шкідливих умовах, у тому числі з комп’ютерним обладнанням.

Ці документи є обов’язковими для виконання роботодавцями та працівниками, оскільки їх порушення може призвести до штрафних санкцій, зупинки діяльності або загрози здоров’ю персоналу.

Робота з блокчейн-системами передбачає значне навантаження на нервову систему, зір та опорно-руховий апарат. З метою профілактики:

- Проводяться обов’язкові медичні огляди раз на рік згідно з Наказом МОЗ № 246;
- Впроваджуються програми гімнастики та фізичних вправ під час робочих перерв;

- Використовуються засоби для підтримки оптимального мікроклімату (зволожувачі, кондиціонери).

Організація безпечних умов праці під час впровадження блокчейн-систем в аудиті бухгалтерського обліку є необхідною складовою ефективного функціонування підприємства. Дотримання чинних нормативних актів, впровадження ергономічних рішень та кібербезпеки, а також забезпечення профілактичних заходів допоможуть знизити рівень професійних ризиків та покращити загальну продуктивність праці.

4.2. Безпека в надзвичайних ситуаціях

Тема: Підвищення стійкості роботи комп'ютеризованих систем в умовах дії ЕМІ ядерних вибухів [15].

В сучасних системах аудиту бухгалтерського обліку, що базуються на технологіях блокчейн, зокрема на платформі Hyperledger Fabric, стабільність та надійність роботи є критично важливими аспектами. Цифрові платформи, які забезпечують децентралізоване зберігання даних, обробку транзакцій та автоматизацію аудиторських перевірок, повинні функціонувати у складних умовах, включаючи надзвичайні ситуації. Однією з найбільш загрозливих та специфічних умов є дія електромагнітного імпульсу (ЕМІ), що виникає внаслідок ядерного вибуху.

Електромагнітний імпульс (ЕМІ), що виникає під час ядерного вибуху, є одним з найнебезпечніших уражаючих факторів для сучасних комп’ютеризованих систем. Він складається з короткочасного інтенсивного електричного та магнітного поля, яке здатне викликати індукцію високих напруг і струмів у ланцюгах електричного живлення та інформаційних системах. Уразливість таких систем обумовлена їхньою складністю, високою щільністю мікросхем, а також

недостатньою підготовленістю до роботи в умовах надзвичайних ситуацій. EMI може вивести з ладу критично важливі компоненти цифрових систем, що обслуговують аудиторські процеси, призвести до втрати даних і порушення роботи інфраструктури. Для забезпечення стійкості системи ClearLedger в таких умовах необхідно впровадити ряд технічних та організаційних заходів, які дозволять знизити вплив EMI на функціональність блокчейн-мережі.

EMI ядерного вибуху характеризується короткочасним імпульсом електромагнітного випромінювання в діапазоні частот від 10 кГц до 100 МГц. Він виникає у три етапи [16]:

- Початковий імпульс (тривалість до 1 мікросекунди) – високочастотне випромінювання з великою енергією.
- Середньочастотний імпульс (тривалість до 1 мілісекунди) – призводить до індукції струмів у довгих провідниках та антенах.
- Пізній імпульс (тривалість до кількох секунд) – генерується магнітним полем Землі, що зміщується в результаті вибуху.

Під дією EMI в електричних ланцюгах і системах керування виникають:

- Перенапруги – до 50-100 кВ у відкритих лініях зв’язку та до 10-20 кВ у захищених системах.
- Індуковані струми – можуть досягати кількох сотень ампер у провідниках довжиною понад 10 метрів.
- Пошкодження мікросхем через перевищення граничних напруг для кремнієвих елементів (понад 50 В).

Особливо вразливими є вузли Hyperledger Fabric (Peer Nodes, Orderer Nodes, Certificate Authority), а також мережеві пристрої, які забезпечують взаємодію системи з користувачами та зовнішніми сервісами. Вплив EMI загрожує втратою транзакцій, порушенням роботи смарт-контрактів та компрометацією загальної працездатності мережі.

Для оцінки стійкості систем до дії EMI застосовується наступний алгоритм:

- Визначення індукованої напруги за формулою де – індуктивність провідника (Гн), – швидкість зміни струму.
- Розрахунок струму, що індукується в кабелях з урахуванням параметрів EMI та довжини кабелю.
- Моделювання роботи системи в умовах EMI для визначення найбільш уразливих елементів.

  Розрахунки:
- Довжина кабелю: 10 м, індуктивність: 1 мГн/м.
- Швидкість зміни струму: 100 А/мкс.

Розрахована напруга: При такій напрузі мікросхеми з граничною напругою 50 В виходять з ладу.

Для забезпечення стійкості ClearLedger в умовах дії EMI ядерних вибухів, необхідно застосувати комплекс заходів, які включають:

Захист обладнання від EMI:

- Екранування серверних приміщень: Встановлення Faraday Cage — екрануючих конструкцій з міді або алюмінію, що забезпечують захист до 90-95% від зовнішніх електромагнітних впливів.
- Захищене розміщення вузлів: Встановлення серверів у підземних бункерах на глибині до 10 метрів для зниження ймовірності впливу EMI.
- Додаткове екранування кабелів: Використання кабелів з подвійним екрануванням та прокладання їх у металевих трубах для зменшення індукованого струму.

Захист електроживлення:

- Використання UPS-систем з автоматичним перемиканням для забезпечення стабільного живлення навіть при індукції зовнішніх струмів;
- Впровадження варисторів та фільтрів для зниження перенавантаження в електромережі;

- Використання автономних джерел живлення, таких як дизельні генератори або акумуляторні батареї, які здатні працювати протягом 24-72 годин після інциденту.

Резервування та дублювання критичних вузлів

- Дублювання Peer-вузлів: Наявність резервних Peer-вузлів у географічно віддалених дата-центрах.
- Дублювання Orderer-вузлів: Розподіл вузлів консенсусу між різними географічними регіонами.
- Резервні CA-сервери: Впровадження дублюючих серверів сертифікатів у захищених зонах.

Захист даних і програмного забезпечення

- Налаштування регулярного резервного копіювання транзакцій на віддалені вузли, що не піддаються дії EMI;
- Використання алгоритмів шифрування AES-256 для захисту критичних даних;
- Впровадження системи миттєвого відновлення вузлів на основі збережених бекапів у хмарному середовищі.

Рекомендації:

- Забезпечити екранування всіх критичних кабельних ліній та обладнання.
- Встановити сучасні захисні пристрої від перенапруг та газові розрядники.
- Резервувати джерела живлення та системи керування з автономним режимом роботи не менше 72 годин.
- Регулярно проводити навчання персоналу з експлуатації систем у надзвичайних умовах.
- Виконувати періодичне тестування обладнання на стійкість до EMI з використанням спеціалізованих генераторів.

Висновки:

Підвищення стійкості комп'ютеризованих систем до впливу ЕМІ ядерного вибуху є складним завданням, яке вимагає інтеграції технічних, організаційних та інженерних заходів. Впровадження заходів екранування, заземлення, фільтрації та резервування дозволяє мінімізувати вплив ЕМІ на критичні системи підприємства. Комплексний підхід до вирішення цієї проблеми забезпечить стабільну роботу обладнання в умовах надзвичайних ситуацій воєнного часу.

ВИСНОВКИ

Розробка системи аудиту бухгалтерського обліку на базі технологій блокчейн є важливим кроком у напрямку вдосконалення фінансового контролю, підвищення прозорості та забезпечення відповідності сучасним стандартам. У рамках цього проекту вдалося створити комплексну, багатофункціональну систему, яка поєднує в собі переваги децентралізованих реєстрів, автоматизації аудиторських процесів та інтеграції з існуючими ERP-системами. Завдяки застосуванню Hyperledger Fabric, система здатна забезпечити високу надійність, безпеку та масштабованість, що є ключовими вимогами для сучасних фінансових технологій.

Розробка розпочалася з глибокого аналізу предметної області, визначення проблем традиційних систем аудиту та формулювання вимог до нової системи. Виявлено, що існуючі рішення часто страждають на брак прозорості у фінансових процесах, високий ризик людських помилок, шахрайства та складність регуляторного контролю. Це стало основою для формування технічного завдання, яке враховувало як потреби кінцевих користувачів, так і вимоги до безпеки та продуктивності. Аналіз дозволив створити основу для моделювання бізнес-процесів, проектування архітектури та подальшої реалізації системи.

Система базується на розподіленій мережі Hyperledger Fabric, яка інтегрує бухгалтерську та аудиторську організації. Ця архітектура забезпечує високий рівень конфіденційності даних через використання приватних каналів і дозволяє здійснювати аудиторський контроль на кожному етапі обробки транзакцій. Використання Peer- і Orderer-вузлів із консенсусом Raft дозволило досягти високої стійкості системи навіть за умов підвищеного навантаження. Реалізовані смарт-контракти виконують ключову бізнес-логіку, включаючи реєстрацію, перевірку та аналіз фінансових транзакцій. Інструменти автоматизованого виявлення аномалій

та генерації звітів значно спростили процес аудиту, мінімізуючи ризики шахрайства та помилок.

Інтеграція API-шлюзу забезпечила ефективний доступ до функціоналу системи для зовнішніх клієнтів і ERP-систем. API дозволяє виконувати CRUD-операції з транзакціями, генерувати звіти та управляти ролями користувачів. Реалізація веб-інтерфейсу на базі React спростила роботу з системою, надавши зручний і інтуїтивно зрозумілий інструмент для взаємодії з даними. Інтеграція з CouchDB забезпечила ефективне зберігання та швидкий доступ до транзакцій, дозволивши виконувати складні запити для аналітики та звітності.

Особливу увагу приділено питанням безпеки. Використання TLS-з'єднань, цифрових сертифікатів та механізмів відкликання сертифікатів (CRL) гарантує захист даних від несанкціонованого доступу. Завдяки реалізації централізованого логування, моніторингу та резервного копіювання, система здатна працювати у продуктивному середовищі з високими вимогами до доступності та безперебійності. Крім того, застосування RabbitMQ для подіє-орієнтованої архітектури забезпечило ефективний обмін даними між компонентами системи.

Під час тестування система продемонструвала високу продуктивність, обробляючи до 4500 транзакцій за хвилину. Це підтверджує її готовність до масштабування та роботи під високим навантаженням. Автоматизоване виявлення аномалій та функції аудиторського журналу дозволяють значно знизити навантаження на аудиторів, спрямовуючи їхню увагу лише на підозрілі операції. Інтеграція з ERP-системами дозволяє безшовно впроваджувати рішення в існуючу інфраструктуру компаній, роблячи процеси фінансового обліку більш ефективними та прозорими.

Загалом проект досягнув своєї мети, створивши прозору, безпечну та ефективну систему аудиту. Інноваційні підходи, застосовані в проекті, не лише вирішують актуальні проблеми аудиторської галузі, але й відкривають нові перспективи для автоматизації та вдосконалення фінансових процесів. Реалізація довела доцільність використання технологій блокчейн для забезпечення довіри до

даних та підвищення ефективності облікових операцій. Система готова до впровадження у реальних бізнес-сценаріях і може стати основою для подальших досліджень, вдосконалення та впровадження в інших секторах економіки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. SAP Blockchain: The new Technology for Trust, SAP, 2024. [Online]. Available: https://www.sap.com/products/artificial-intelligence/what-is-blockchain.html

2. Oracle Blockchain. Blockchain Applications in Finance. Oracle Corporation, 2024 [Online]. Available: https://www.oracle.com/blockchain/

3. KPMG. Blockchain Technology in Auditing: A Practical Guide, KPMG International Limited, 2024. [Online]. Available: https://kpmg.com/xx/en/our-insights.html

4. Hyperledger Foundation. Hyperledger Fabric Documentation, 2024. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-2.5/

5. CouchDB Team. Apache CouchDB Documentation, 2023. [Online]. Available: https://couchdb.apache.org/

6. Marcus Selig, Blockchain Based Auditing, 2023. [Online]. Available: https://ojs.ebujournals.lu/index.php/JIDS/article/view/15

7. S. Sachan, Xi Liu, Blockchain-based auditing of legal decisions supported by explainable AI and generative AI tools, 2023 [Online]. Available: https://dl.acm.org/doi/10.1016/j.engappai.2023.107666

8. International Financial Reporting Standards (IFRS), 2023. [Online]. Available: https://www.ifrs.org/issued-standards/list-of-standards/

9. Jai Singh Arun, J. Cuomo, N. Gaur. Blockchain for Business, O'Reilly Media, 2019.

10. IEEE Standards Association, IEEE 2418.7-2021: IEEE Standard for the Use of Blockchain in Supply Chain Finance, 2021. [Online]. Available: https://standards.ieee.org/ieee/2418.7/7447/

11. IBM Blockchain Team, Getting started with IBM Support for Hyperledger Fabric, 2023. [Online]. Available: https://www.ibm.com/docs/en/hlf-support/1.0.0?topic=started-getting-support-hyperledger-fabric

12. RabbitMQ Documentation, Message Queuing for Distributed Systems, 2023. [Online]. Available: https://www.rabbitmq.com/docs

13. Stefanyshyn, I. , Pastukh, O., Stefanyshyn, V. , Baran, I. , Boyko, I. Robustness of AI algorithms for neurocomputer interfaces based on software and hardware technologies, CEUR Workshop Proceedings, 2024, 3742, pp. 137–149.

14. Савка, Н., Васильків, Н., Дубчак, Л., & Мудрик, І. РАДІАЛЬНО-БАЗИСНІ НЕЙРОННІ МЕРЕЖІ ДЛЯ ПРОГНОЗУВАНЯ ДІЯЛЬНОСТІ ПІДПРИЄМСТВ. European Science, 3(sge17-03), 42–48, 2023. [Онлайн] Доступно: https://doi.org/10.30890/2709-2313.2023-17-03-012

15. Стручок В.С. Безпека в надзвичайних ситуаціях. Методичний посібник для здобувачів освітнього ступеня «магістр» всіх спеціальностей денної та заочної (дистанційної) форм навчання / В.С.Стручок. — Тернопіль: ФОП Паляниця В. А., 2022. — 156 с.

16. Техноекологія та цивільна безпека. Частина «Цивільна безпека». Навчальний посібник / В.С. Стручок, – Тернопіль: ТНТУ ім. І.Пулюя, 2022. – 150 с.

17. Методичні вказівки до виконання атестаційної роботи магістра за спеціальністю 121 – Інженерія програмного забезпечення (Освітньо-професійна програма - «Програмне забезпечення систем», Освітньо-наукова програма - «Інженерія програмного забезпечення») для студентів усіх форм навчання / Упор.: М. Р. Петрик, Д. М. Михалик, О. Ю. Петрик, Г. Б. Цуприк – Тернопіль: ТНТУ, 2020- 51с.

ДОДАТКИ

Додаток А

Лістинг коду застосунку

docker-compose.yaml

```
version: '3.7'

services:
  # Orderer Node 1
  orderer.example.com:
    container_name: orderer.example.com
    image: hyperledger/fabric-orderer:latest
    environment:
      - ORDERER_GENERAL_LOGLEVEL=info
      - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
      - ORDERER_GENERAL_GENESISFILE=/var/hyperledger/orderer/genesis.block
      - ORDERER_GENERAL_LOCALMSPID=OrdererMSP
      - ORDERER_GENERAL_LOCALMSPDIR=/var/hyperledger/orderer/msp
      - ORDERER_GENERAL_TLS_ENABLED=true
      - ORDERER_GENERAL_TLS_CERTIFICATE=/var/hyperledger/orderer/tls/server.crt
      - ORDERER_GENERAL_TLS_PRIVATEKEY=/var/hyperledger/orderer/tls/server.key
      - ORDERER_GENERAL_TLS_ROOTCAS=[/var/hyperledger/orderer/tls/ca.crt]
    volumes:
      - ./channel-artifacts/genesis.block:/var/hyperledger/orderer/genesis.block
      -
./crypto-config/ordererOrganizations/example.com/orderers/orderer.example.com/
msp:/var/hyperledger/orderer/msp
      -
./crypto-config/ordererOrganizations/example.com/orderers/orderer.example.com/
tls:/var/hyperledger/orderer/tls
    ports:
      - "7050:7050"

  # Peer Node 1 (Org1)
  peer0.org1.example.com:
    container_name: peer0.org1.example.com
    image: hyperledger/fabric-peer:latest
    environment:
      - CORE_PEER_ID=peer0.org1.example.com
      - CORE_PEER_LOCALMSPID=Org1MSP
      - CORE_PEER_ADDRESS=peer0.org1.example.com:7051
      - CORE_PEER_GOSSIP_BOOTSTRAP=peer0.org1.example.com:7051
```

```
      - CORE_PEER_TLS_ENABLED=true
      - CORE_PEER_TLS_CERT_FILE=/etc/hyperledger/peer/tls/server.crt
      - CORE_PEER_TLS_KEY_FILE=/etc/hyperledger/peer/tls/server.key
      - CORE_PEER_TLS_ROOTCERT_FILE=/etc/hyperledger/peer/tls/ca.crt
    volumes:
      -
./crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/
msp:/etc/hyperledger/peer/msp
      -
./crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/
tls:/etc/hyperledger/peer/tls
      - ./data/peer0.org1:/var/hyperledger/production
    ports:
      - "7051:7051"

  # CouchDB for Peer0 Org1
  couchdb0:
    container_name: couchdb0
    image: couchdb:latest
    environment:
      - COUCHDB_USER=admin
      - COUCHDB_PASSWORD=password
    ports:
      - "5984:5984"
    volumes:
      - ./data/couchdb0:/opt/couchdb/data

  # Certificate Authority for Org1
  ca.org1.example.com:
    container_name: ca.org1.example.com
    image: hyperledger/fabric-ca:latest
    environment:
      - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
      - FABRIC_CA_SERVER_CA_NAME=ca-org1
      - FABRIC_CA_SERVER_TLS_ENABLED=true
      - FABRIC_CA_SERVER_TLS_CERTFILE=/etc/hyperledger/fabric-ca-server-config/
ca.org1.example.com-cert.pem
      - FABRIC_CA_SERVER_TLS_KEYFILE=/etc/hyperledger/fabric-ca-server-config/
ca.org1.example.com-key.pem
    ports:
      - "7054:7054"
```

```
    volumes:
      - ./crypto-config/ca/ca.org1.example.com:/etc/hyperledger/fabric-ca-server-
config
      - ./data/ca.org1:/etc/hyperledger/fabric-ca-server

  # API Gateway
  api-gateway:
    container_name: api-gateway
    image: node:16
    working_dir: /app
    volumes:
      - ./api:/app
    command: ["npm", "start"]
    environment:
      - PORT=4000
      - FABRIC_CC_NAME=mychaincode
      - FABRIC_CHANNEL_NAME=mychannel
    ports:
      - "4000:4000"
    depends_on:
      - peer0.org1.example.com

  # Web UI
  ui:
    container_name: ui
    image: nginx:latest
    ports:
      - "8080:80"
    volumes:
      - ./ui/build:/usr/share/nginx/html
  # RabbitMQ for Event Bus
  rabbitmq:
    container_name: rabbitmq
    image: rabbitmq:management
    environment:
      - RABBITMQ_DEFAULT_USER=guest
      - RABBITMQ_DEFAULT_PASS=guest
    ports:
      - "5672:5672"
      - "15672:15672"
    volumes:
```

```
      - ./data/rabbitmq:/var/lib/rabbitmq
networks:
  default:
    name: fabric-network
```

index.js (Логіка смарт-контрактів)

```
// SPDX-License-Identifier: Apache-2.0

'use strict';

const { Contract } = require('fabric-contract-api');

class ClearLedgerContract extends Contract {

    // Initialize ledger with sample data
    async initLedger(ctx) {
        console.info('Initializing the ledger with sample data...');
        const sampleTransactions = [
            {
                transactionId: 'TX001',
                amount: 500,
                description: 'Payment for services',
                timestamp: new Date().toISOString(),
                status: 'completed',
            },
            {
                transactionId: 'TX002',
                amount: 1500,
                description: 'Refund',
                timestamp: new Date().toISOString(),
                status: 'completed',
            }
        ];

        for (const transaction of sampleTransactions) {
            await ctx.stub.putState(transaction.transactionId,
Buffer.from(JSON.stringify(transaction)));
            console.info(`Transaction ${transaction.transactionId} added to the
ledger.`);
        }
    }
```

```
    // Create a new transaction
    async createTransaction(ctx, transactionId, amount, description) {
        const transactionExists = await this.transactionExists(ctx,
transactionId);
        if (transactionExists) {
            throw new Error(`The transaction ${transactionId} already exists.`);
        }

        const transaction = {
            transactionId,
            amount: parseFloat(amount),
            description,
            timestamp: new Date().toISOString(),
            status: 'pending',
        };

        await ctx.stub.putState(transactionId,
Buffer.from(JSON.stringify(transaction)));
        return JSON.stringify(transaction);
    }

    // Read an existing transaction
    async readTransaction(ctx, transactionId) {
        const transactionJSON = await ctx.stub.getState(transactionId);
        if (!transactionJSON || transactionJSON.length === 0) {
            throw new Error(`The transaction ${transactionId} does not exist.`);
        }
        return transactionJSON.toString();
    }

    // Update an existing transaction
    async updateTransaction(ctx, transactionId, amount, description, status) {
        const transactionJSON = await ctx.stub.getState(transactionId);
        if (!transactionJSON || transactionJSON.length === 0) {
            throw new Error(`The transaction ${transactionId} does not exist.`);
        }

        const transaction = JSON.parse(transactionJSON.toString());
        transaction.amount = parseFloat(amount);
        transaction.description = description;
        transaction.status = status;
```

```
        transaction.timestamp = new Date().toISOString();

        await ctx.stub.putState(transactionId,
Buffer.from(JSON.stringify(transaction)));
        return JSON.stringify(transaction);
    }

    // Delete a transaction
    async deleteTransaction(ctx, transactionId) {
        const transactionExists = await this.transactionExists(ctx,
transactionId);
        if (!transactionExists) {
            throw new Error(`The transaction ${transactionId} does not exist.`);
        }
        await ctx.stub.deleteState(transactionId);
        return `Transaction ${transactionId} has been deleted.`;
    }

    // Query all transactions
    async queryAllTransactions(ctx) {
        const startKey = '';
        const endKey = '';
        const iterator = await ctx.stub.getStateByRange(startKey, endKey);
        const results = [];

        while (true) {
            const res = await iterator.next();

            if (res.value) {
                const record = JSON.parse(res.value.value.toString('utf8'));
                results.push(record);
            }

            if (res.done) {
                await iterator.close();
                return JSON.stringify(results);
            }
        }
    }

    // Check if a transaction exists
```

```
    async transactionExists(ctx, transactionId) {
        const transactionJSON = await ctx.stub.getState(transactionId);
        return transactionJSON && transactionJSON.length > 0;
    }
}

module.exports = ClearLedgerContract;
```

App.js (Основний файл серверного додатку)

```
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const FabricCAServices = require('fabric-ca-client');
const { Gateway, Wallets } = require('fabric-network');
const path = require('path');
const fs = require('fs');
const Client = require('ftp');

const app = express();
const PORT = 3000;

// Middleware
app.use(bodyParser.json());
app.use(cors());

// Load network configuration
const ccpPath = path.resolve(__dirname, 'connection-profile.json');
const ccp = JSON.parse(fs.readFileSync(ccpPath, 'utf8'));

// FTP Configuration
const ftpConfig = {
  host: 'ftp.example.com',
  user: 'ftp_user',
  password: 'ftp_password',
};

// Upload file to FTP server
async function uploadToFTP(filePath, fileName) {
  return new Promise((resolve, reject) => {
    const ftpClient = new Client();
    ftpClient.on('ready', () => {
```

```
      ftpClient.put(filePath, fileName, (err) => {
        if (err) reject(err);
        ftpClient.end();
        resolve('File uploaded successfully');
      });
    });
    ftpClient.connect(ftpConfig);
  });
}

// Blockchain interaction: submit a transaction
async function submitTransaction(contractName, transactionName, args) {
  try {
    // Create a new wallet instance
    const walletPath = path.join(__dirname, 'wallet');
    const wallet = await Wallets.newFileSystemWallet(walletPath);

    // Check if the user identity exists
    const identity = await wallet.get('appUser');
    if (!identity) {
      throw new Error('An identity for the user "appUser" does not exist in the
wallet.');
    }

    // Create a new gateway and connect
    const gateway = new Gateway();
    await gateway.connect(ccp, {
      wallet,
      identity: 'appUser',
      discovery: { enabled: true, asLocalhost: true },
    });

    // Get the network and contract
    const network = await gateway.getNetwork('mychannel');
    const contract = network.getContract(contractName);

    // Submit the transaction
    const result = await contract.submitTransaction(transactionName, ...args);

    // Disconnect from the gateway
    await gateway.disconnect();
```

```
    return result.toString();
  } catch (error) {
    throw new Error(`Failed to submit transaction: ${error.message}`);
  }
}

// API Routes
app.post('/verify-document', async (req, res) => {
  try {
    const { documentId, filePath, fileName } = req.body;

    // Simulate verification logic
    const verificationResult = true; // Assume the document is verified
successfully

    if (!verificationResult) {
      return res.status(400).json({ message: 'Document verification failed' });
    }

    // Upload document to FTP server
    await uploadToFTP(filePath, fileName);

    // Record the transaction on the blockchain
    const transactionResult = await submitTransaction('docVerification',
'verifyDocument', [documentId, fileName]);

    res.status(200).json({ message: 'Document verified and transaction recorded',
transactionResult });
  } catch (error) {
    res.status(500).json({ message: 'Error processing request', error:
error.message });
  }
});

// Start server
app.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});
```

Додаток Б

Тези наукової конференції

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
імені ІВАНА ПУЛЮЯ
НАУКОВЕ ТОВАРИСТВО ім. ШЕВЧЕНКА

М А Т Е Р І А Л И

XII НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

18-19 грудня 2024 року

ТЕРНОПІЛЬ
2024

УДК 004.9, 004.45

Д. Романюк, Ph.D. І. Мудрик
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

ВПЛИВ СУЧАСНИХ ТЕХНОЛОГІЙ НА БУХГАЛТЕРСЬКИЙ ОБЛІК

D. Romaniuk, Ph.D. I. Mudryk

IMPACT OF MODERN TECHNOLOGIES ON ACCOUNTING

Цифрова трансформація кардинально змінює методи ведення бухгалтерського обліку, впливаючи на його ефективність, прозорість і адаптивність до потреб сучасного бізнесу. Інноваційні технології, такі як хмарні обчислення, штучний інтелект, машинне навчання та аналіз великих даних, пропонують нові можливості для обробки, аналізу та представлення фінансової інформації. Їхнє впровадження створює нову парадигму в обліковій сфері, яка орієнтована на автоматизацію, зменшення помилок і підвищення продуктивності.

Одним із ключових чинників цифрової трансформації є впровадження хмарних обчислень [1]. Хмарні платформи дозволяють компаніям забезпечувати безперервний доступ до фінансових даних з будь-якої точки світу, що особливо важливо в умовах віддаленої роботи та глобалізації бізнесу. Інтеграція хмарних технологій не лише оптимізує витрати на обслуговування серверів і програмного забезпечення, а й забезпечує швидке оновлення інформації, спрощуючи управлінські процеси. Завдяки цьому облікові системи перетворюються на динамічні інструменти для підтримки прийняття рішень.

Технології штучного інтелекту та машинного навчання суттєво змінюють характер виконання рутинних облікових завдань. Автоматизація таких процесів, як обробка рахунків-фактур, класифікація транзакцій та підготовка звітності, дозволяє бухгалтерам зосередитися на стратегічному аналізі, управлінні ризиками та консультуванні керівництва. Наприклад, алгоритми машинного навчання можуть виявляти аномалії у фінансових даних, що сприяє зниженню ризиків шахрайства та помилок. Крім того, використання прогнозної аналітики допомагає компаніям краще оцінювати майбутні фінансові результати, що є важливим для довгострокового планування.

Аналіз великих даних (Big Data) відкриває нові можливості для фінансового моделювання та управління. Завдяки можливості обробляти величезні обсяги інформації в режимі реального часу, компанії отримують змогу виявляти приховані тенденції, оцінювати вплив зовнішніх факторів на фінансову стабільність і оперативно реагувати на зміни ринку. Такий підхід значно підвищує точність управлінських рішень, що є критично важливим в умовах конкурентного середовища.

Цифрова трансформація також змінює роль бухгалтера, перетворюючи його на стратегічного партнера бізнесу. Замість виконання ручних операцій бухгалтери все більше займаються інтерпретацією даних, розробкою прогнозів та участю в розробці бізнес-стратегій. Це вимагає нових компетенцій, таких як знання аналітичних інструментів, розуміння принципів роботи сучасних інформаційних систем та здатність адаптуватися до швидких змін технологій.

Попри значні переваги, цифрова трансформація бухгалтерського обліку супроводжується певними викликами. Основними з них є високі витрати на впровадження нових технологій, необхідність перекваліфікації персоналу та ризики, пов'язані з кібербезпекою. Для ефективного впровадження цифрових технологій

компаніям потрібно розробляти довгострокові стратегії, які враховують технічні, організаційні та юридичні аспекти. Крім того, регуляторні органи повинні забезпечити адаптацію нормативно-правової бази до нових реалій цифрової економіки.

Інтеграція блокчейн-технологій [2] у бухгалтерський облік є ще одним перспективним напрямом. Використання розподілених реєстрів забезпечує високий рівень прозорості та достовірності даних, що знижує ризик маніпуляцій та помилок. Завдяки автоматизації процесів перевірки та підтвердження транзакцій, блокчейн може значно спростити аудит і скоротити час, необхідний для підготовки фінансових звітів. Крім того, впровадження смарт-контрактів може замінити ручне управління окремими операціями, виконуючи їх заздалегідь визначеними умовами.

Загалом цифрова трансформація бухгалтерського обліку є не лише технологічним, а й стратегічним процесом, який вимагає комплексного підходу [3]. Компаніям важливо зважувати переваги впровадження нових технологій із можливими ризиками, такими як кібербезпека чи складнощі інтеграції із застарілими системами. Лише за умови належного управління цим процесом цифрові інновації зможуть забезпечити значний приріст продуктивності, конкурентоспроможності та якості управлінських рішень, перетворивши бухгалтерський облік на ключовий елемент бізнес-екосистеми майбутнього.

Література

1. Busulwa, R., & Evans, N. Digital Transformation in Accounting. Routledge, 2020.
2. Kanaparthi, V. (2024). Exploring the Impact of Blockchain, AI, and ML on Financial Accounting Efficiency and Transformation. DOI:10.48550/arXiv.2401.15715.
3. Савка, Н., Васильків, Н., Дубчак, Л., & Мудрик, І. (2023). РАДІАЛЬНО-БАЗИСНІ НЕЙРОННІ МЕРЕЖІ ДЛЯ ПРОГНОЗУВАНЯ ДІЯЛЬНОСТІ ПІДПРИЄМСТВ. European Science, 3(sge17-03), 42–48. https://doi.org/10.30890/2709-2313.2023-17-03-012

Додаток В

Диск