

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «магістр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Розробка децентралізованої платформи електронної комерції на основі блокчейн-технологій»

Виконав студент групи ІПЗм-24-1
ХОЛОДНІЦЬКИЙ Владислав Ігорович

Керівник роботи:
ВАКАЛЮК Тетяна Анатоліївна

Рецензент:
ЛОКТИКОВА Тамара Миколаївна

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«18» вересня 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ХОЛОДНИЦЬКИЙ Владислав Ігорович**

Керівник роботи: **ВАКАЛЮК Тетяна Анатоліївна**

Тема роботи: **«Розробка децентралізованої платформи електронної комерції на основі блокчейн-технологій»,**

затверджена Наказом закладу вищої освіти від «18» вересня 2025 р. № 491/с
Вихідні дані для роботи: **моделі та процеси роботи децентралізованого маркетплейсу; смарт-контракти та стандарти токенів для реалізації безпечної оплати; механізми захисту взаєморозрахунків між покупцем і продавцем та вирішення спорів; методи розрахунку рейтингу товарів і репутації продавців з урахуванням історії угод і диспутів; підходи до ранжування товарів і формування персоналізованих рекомендацій із збереженням приватності; принципи інтеграції з розподіленою файловою системою IPFS та використання CID для зберігання офчейн-контенту.**

Консультанти з магістерської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Вакалюк Т. А.	22.09.2025р.	22.09.2025р.
2	Вакалюк Т. А.	22.09.2025р.	22.09.2025р.
3	Вакалюк Т. А.	22.09.2025р.	22.09.2025р.

РЕФЕРАТ

Випускна кваліфікаційна робота магістра присвячена проектуванню та розробці децентралізованої платформи електронної комерції на основі блокчейн-технологій, зокрема реалізації смарт-контрактної логіки маркетплейсу, ескроу-механізму, моделі врегулювання спорів, системи репутації та алгоритмів ранжування і персоналізованих рекомендацій. Пояснювальна записка до кваліфікаційної роботи складається з 92 сторінок, 24 ілюстрацій, 1 таблиці, 4 додатків та 36 літературних джерел.

Мета роботи – проектування та розробка децентралізованої платформи електронної комерції, яка забезпечує безпечне виконання угод через смарт-контракти, депонування оплати в ескроу, регламентоване врегулювання спорів, а також формування репутації, прозоре ранжування товарів і персоналізовані рекомендації із урахуванням вимог приватності.

У роботі визначено мету й завдання, проаналізовано централізовані та децентралізовані підходи до електронної комерції та показано доцільність використання смарт-контрактів як основи правил і гарантій угод. Спроектовано трирівневу архітектуру з гібридним зберіганням даних. Розроблено дворівневий механізм арбітражу з часовими фазами подання доказів і голосування, модель репутації та рейтингів на основі байєсівського згладжування з урахуванням результатів спорів, а також прозоре ранжування і гібридні рекомендації з персоналізацією. Реалізовано смарт-контракти та веб-застосунок, проведено тестування ключових сценаріїв і оцінено витрати газу для основних транзакцій.

У висновках було отримано результати досліджень та розробок за час виконання кваліфікаційної роботи.

КЛЮЧОВІ СЛОВА: ДЕЦЕНТРАЛІЗОВАНИЙ МАРКЕТПЛЕЙС, БЛОКЧЕЙН, СМАРТ-КОНТРАКТ, ЕСКРОУ, АРБІТРАЖ, ДИСПУТ, РЕПУТАЦІЯ, РЕЙТИНГ, РАНЖУВАННЯ, РЕКОМЕНДАЦІЙНА СИСТЕМА.

ABSTRACT

The master's qualification work consists of a study on the design and development of a blockchain-based decentralized e-commerce platform, the implementation of marketplace smart-contract logic and an escrow mechanism, the development of a regulated dispute resolution model, a reputation system, and algorithms for transparent ranking and personalized recommendations. The explanatory note to the qualification work consists of 92 pages, 24 illustrations, 1 table, 4 appendices and 36 literature sources.

The aim of the work is to design and develop a decentralized e-commerce platform that ensures secure deal execution via smart contracts, payment escrow, regulated dispute resolution, as well as reputation formation, transparent product ranking, and personalized recommendations with privacy requirements taken into account.

The work described the objectives and set tasks for research and development, analyzes centralized and decentralized approaches to e-commerce, and demonstrates the feasibility of using smart contracts as the foundation for enforcing rules and guarantees of transactions. A three-tier architecture with hybrid data storage is designed. A two-level arbitration mechanism with time-bounded phases for evidence submission and voting is developed, along with a reputation and rating model based on Bayesian smoothing that accounts for dispute outcomes, as well as transparent ranking and hybrid recommendations with personalization. Smart contracts and a web application were implemented, key scenarios were tested, and gas costs for the main transactions were evaluated.

The conclusion presents the results of research and development obtained during the completion of the qualification work.

KEYWORDS: DECENTRALIZED MARKETPLACE, BLOCKCHAIN, SMART CONTRACT, ESCROW, ARBITRATION, DISPUTE, REPUTATION, RATING, RANKING, RECOMMENDER SYSTEM.

ЗМІСТ

РЕФЕРАТ.....	3
ABSTRACT.....	4
ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ.....	6
ВСТУП.....	7
РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ	10
1.1 Аналіз проблем та особливостей сучасної електронної комерції.....	10
1.2 Огляд аналогів	12
1.3 Постановка задачі.....	17
1.4 Вибір програмних засобів для реалізації	20
Висновки до першого розділу	23
РОЗДІЛ 2 ПРОЄКТУВАННЯ ДЕЦЕНТРАЛІЗОВАНОЇ ПЛАТФОРМИ ЕЛЕКТРОННОЇ КОМЕРЦІЇ.....	25
2.1 Архітектура платформи та формалізація бізнес-процесів	25
2.2 Модель децентралізованого арбітражу та ескроу-механізму	29
2.3 Математична модель рейтингу товарів та репутації продавців.....	33
2.4 Модель ранжування товарів та персональних рекомендацій	38
2.5 Програмна реалізація методів та алгоритмів платформи.....	40
Висновки до другого розділу.....	49
РОЗДІЛ 3 ОПИС ПРОГРАМНОГО ПРОДУКТУ.....	50
3.1 Структура програмного продукту та взаємодія модулів.....	50
3.2 Функціональні сценарії роботи веб-застосунку	54
3.3 Результати тестування програмного продукту	63
Висновки до третього розділу	66
ВИСНОВКИ	68
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	70
ДОДАТКИ	75

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

CID – Content Identifier (Ідентифікатор вмісту)

ERC-20 – Ethereum Request for Comments 20 (стандарт токенів ERC-20)

EVM – Ethereum Virtual Machine (Віртуальна машина Ethereum)

IPFS – InterPlanetary File System (Міжпланетна файлова система)

NFT – Non-Fungible Token (Невзаємозамінний токен)

P2P – Peer-to-Peer (Однорангова мережа)

PGP – Pretty Good Privacy (Засоби криптографічного захисту та шифрування)

UI – User Interface (Інтерфейс користувача)

UX – User Experience (Користувацький досвід)

JSON – JavaScript Object Notation (Формат обміну даними)

GDPR - General Data Protection Regulation (Регламент ЄС про захист даних)

ВСТУП

Актуальність теми. Електронна комерція за останні роки стала одним із головних способів купувати й продавати товари та послуги, а маркетплейси – майданчиком, який забезпечує зв'язок між покупцями та продавцями. Однак зі зростанням популярності зростає й відповідальність таких платформ, адже вони мають узгоджувати умови угод, проводити безпечні оплати, підтверджувати виконання зобов'язань, формувати репутацію сторін і вирішувати конфлікти між ними. Тому від того, наскільки ці процеси надійні, прозорі та чесні, напряду залежить безпека транзакцій і фінансовий результат для усіх учасників.

У відповідь на ці виклики комбінація блокчейн технологій і смарт-контрактів може стати дієвим способом підсилити довіру та прозорість в інтернет торгівлі. Автоматизація виконання правил і фіксація умов та подій дають змогу зробити перебіг угод передбачуваним, а історію – незмінною та перевірюваною. Це особливо важливо для створення безпечного середовища, де права учасників захищені чіткими правилами автоматизованого процесу, а не суб'єктивними рішеннями третіх осіб.

Втім, на практиці цей потенціал поки що реалізовано не повністю. Водночас більшість наявних децентралізованих рішень поки що не пропонують комплексного підходу – одночасного поєднання надійного ескроу, формалізованого арбітражу, стійкої до маніпуляцій репутації та прозорого ранжування товарів. Окремою практичною проблемою залишається персоналізація рекомендацій, адже в децентралізованому середовищі складно забезпечити точний підбір пропозицій, не розкриваючи чутливі дані користувачів і не перевантажуючи систему зайвими обчисленнями.

Тому проєктування і програмна реалізація децентралізованої платформи електронної комерції є актуальним завданням, оскільки вона поєднує перевірюваність критичних даних, захист фінансових взаєморозрахунків, регламентоване вирішення спорів, а також прозорі механізми репутації,

ранжування й персональних рекомендацій зі збереженням приватності користувача.

Мета роботи полягає у проєктуванні та розробці децентралізованої платформи електронної комерції на основі блокчейн-технологій, яка забезпечує безпечне виконання угод через смарт-контракти, гарантії виконання фінансових зобов'язань сторін і врегулювання спорів, а також використовує прозорі підходи до формування репутації, ранжування та рекомендацій товарів.

Для досягнення мети виокремлено такі основні завдання:

- провести аналіз предметної області та наявних підходів до централізованої і децентралізованої електронної комерції;
- формалізувати ключові бізнес-процеси платформи та визначити ролі й стани взаємодії учасників;
- спроектувати механізм врегулювання спорів і правила залучення арбітражу;
- розробити математичну модель оцінювання рейтингу товарів і репутації продавців;
- визначити модель ранжування товарів і підхід до побудови персоналізованих рекомендацій із урахуванням вимог приватності;
- реалізувати програмне забезпечення платформи та виконати тестування основних сценаріїв, зокрема оцінити вартість виконання ключових транзакцій у блокчейн-мережі.

Об'єктом дослідження є процеси здійснення електронної комерції в середовищі децентралізованих платформ із використанням блокчейн-технологій та смарт-контрактів.

Предметом дослідження є архітектурні, алгоритмічні та математичні моделі виконання угод, врегулювання спорів, репутаційного оцінювання, ранжування товарів і гібридних рекомендацій у децентралізованому маркетплейсі.

Наукова новизна полягає у розробці комплексної моделі децентралізованої платформи електронної комерції, що інтегрує формалізоване врегулювання спорів із регламентованими фазами та відбором арбітрів, репутаційне оцінювання з урахуванням результатів диспутів, а також підхід до прозорого ранжування товарів і персоналізації рекомендацій на основі перевірюваних ончейн-даних.

Практичне значення роботи полягає у створенні програмного забезпечення децентралізованого маркетплейсу, придатного для відтворення повного циклу онлайн-торгівлі: публікації товарів, здійснення покупок, супроводу замовлень і розгляду спірних ситуацій. Реалізована система може використовуватися як більш безпечне та прозоре середовище для торгівлі, у якому ключові події угод фіксуються в незмінному реєстрі, а користувачі мають зрозумілу процедуру захисту своїх інтересів у разі конфлікту.

Апробація. За темою дослідження кваліфікаційної магістерської роботи було опубліковано 2 тези конференцій [1, 2].

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз проблем та особливостей сучасної електронної комерції

Електронна комерція за останні десятиліття перетворилася на один з ключових каналів продажу товарів та послуг. Великі централізовані платформи на кшталт глобальних маркетплейсів забезпечують покупцям широкий вибір товарів, зручні інтерфейси й інтегровані платіжні сервіси, а продавцям – доступ до великої аудиторії. Однак дана модель має принципову особливість: усі критично важливі функції зосереджені в руках одного оператора платформи, який одноосібно контролює бази даних, зміни у правилах, арбітражні рішення, замороження коштів і навіть доступ користувачів до їхніх облікових записів. Приклад інтерфейсу централізованого маркетплейсу наведено на рис. 1.1.

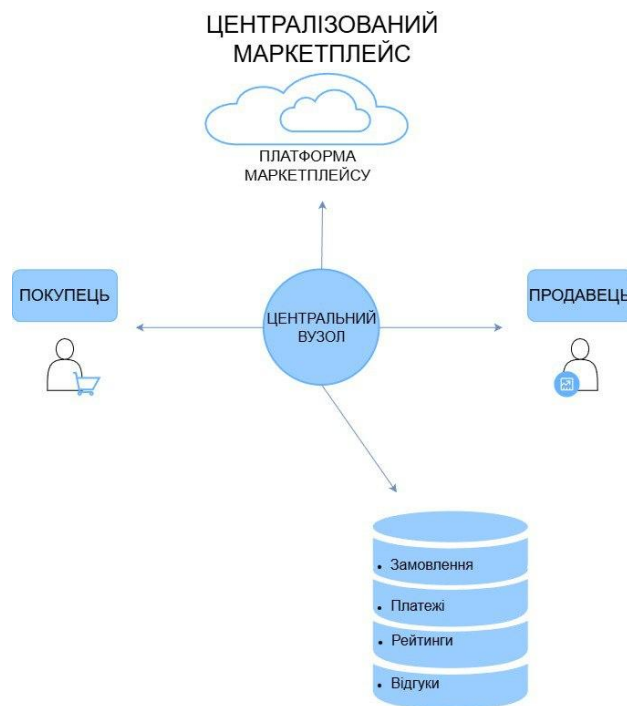


Рис. 1.1 – Приклад інтерфейсу централізованого маркетплейсу

У даній моделі існує низка недоліків. Насамперед, це концентрація довіри: користувачі змушені повністю покладатися на добросовісність і технічну надійність платформи. Будь-яка внутрішня помилка, зловживання або зовнішня атака може призвести до втрати даних, блокування коштів чи необґрунтованих рішень щодо спорів між сторонами. Додатково слід

зазначити, що централізований оператор виступає єдиною точкою відмови як з технічної, так і з регуляторної точки зору, що робить платформу вразливою до цензури, надмірного контролю чи примусового обмеження певних видів товарів.

Важливою проблемою є й ціноутворення послуг. Комісії великих платформ часто є досить суттєвими: частина вартості кожної транзакції йде на оплату інфраструктури, ризиків і прибутку компанії. Для покупця це означає вищу ціну, а для продавця – меншу маржу. При цьому механізм формування цих комісій непрозорий: користувач не може перевірити, як саме розподіляються кошти, і наскільки справедливою є структура зборів.

Ще одним важливим аспектом є формування репутації. Рейтинг продавця й товару зазвичай зберігається в закритій базі даних платформи та повністю залежить від її внутрішньої логіки. Користувачі не мають можливості незалежно верифікувати, чи не були окремі відгуки видалені, змінені або штучно додані. Репутація фактично належить маркетплейсу, а не самому продавцеві, і при переході на іншу платформу продавець втрачає накопичену історію.

На цьому фоні з'являється альтернативний підхід – децентралізовані маркетплейси, які будуються поверх блокчейн-мереж і використовують смарт-контракти для виконання угод без посередника. У загальному випадку децентралізований маркетплейс можна визначити як мережу, у якій покупці й продавці обмінюються товарами або послугами без централізованого оператора, а умовами угоди керують самовиконувані програми в блокчейні. Такі платформи спираються на відкритий і незмінний реєстр, що зберігає історію замовлень, платежів і дій сторін, що значно підвищує прозорість процесу та знижує необхідність довіряти окремій компанії. Загальну модель взаємодії учасників у децентралізованому маркетплейсі наведено на рис. 1.2.

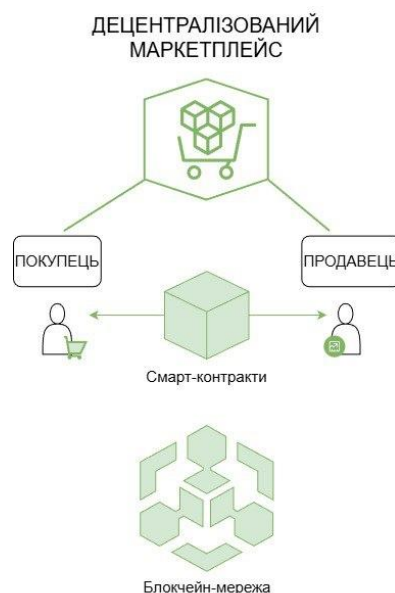


Рис. 1.2 – Схема взаємодії учасників у децентралізованому маркетплейсі

Попри очевидні переваги, децентралізована електронна комерція стикається зі своїми викликами. У більшості сучасних рішень відсутні зрозумілі моделі вирішення спорів між сторонами, бракує зручних механізмів ескроу для захисту обох сторін угоди, а репутаційні системи часто зводяться до спрощених оцінок, що не враховують історію конфліктів і реальну поведінку продавця. Окрім того, такі платформи повинні поєднувати безпеку блокчейну з комфортом використання для звичайного користувача, що значно ускладнює їх проєктування.

У контексті цього дослідження предметною областю є саме децентралізований маркетплейс, орієнтований на захист інтересів покупця і продавця за допомогою смарт-контрактів, прозорості системи рейтингів та вбудованого механізму вирішення спорів.

1.2 Огляд аналогів

Для формування вимог проєктованої системи спочатку потрібно детально розглянути наявні моделі організації інтернет торгівлі та проаналізувати їх переваги на недоліки з огляду на безпеку, прозорість і зручність використання. Сучасний ринок фактично поділений на два сегменти. Класичні централізовані платформи, які займають домінуюче місце завдяки

високому рівню сервісу та інтуїтивно зрозумілому інтерфейсу, до якого мільйони людей уже звикли. Натомість до другого сегмента ринку належать децентралізовані рішення на основі блокчейн-технологій, що роблять акцент на відсутності посередників та криптографічних гарантіях, проте стикаються зі значними бар'єрами на шляху до масового впровадження.

У ролі еталонного прикладу централізованого підходу доцільно розглядати модель функціонування таких гігантів електронної комерції, як eBay та Amazon. Архітектура подібних сервісів базується на існування єдиного довіреного посередника, який має велику довіру користувачів і повністю контролює інфраструктуру, бази даних, алгоритми роботи та фінансові потоки.

Ключовими перевагами даного підходу є висока швидкість обробки запитів, стабільність роботи, інтеграція з звичними усім платіжними сервісами та логістичними операторами. Проте ці переваги досягаються ціною втрати контролю з боку користувача над власним цифровим профілем. Дані про репутацію, історію угод, історію змін рейтингів та правила їх розрахунку зберігаються на закритих серверах компанії у форматі так званої чорної скриньки.

Варто теж зазначити, що адміністрація платформи має і технічну можливість, і юридичні підстави заблокувати обліковий запис, заморозити кошти чи видалити лістинг – інколи без попередження і без зрозумілого пояснення причин. Найгостріше це відчувається, коли виникають суперечки між покупцем і продавцем. У таких ситуаціях рішення ухвалюють співробітники служби підтримки, спираючись на внутрішні, неопрілюднені регламенти. Тому користувачеві зрештою лишається покладатися не стільки на формальні правила, скільки на репутацію та доброчесність самої платформи.

На противагу централізованим платформам розвиваються спроби побудувати однорангові торговельні майданчики без централізованого посередника. Історично важливим прикладом такого підходу став проєкт

OpenBazaar. У цій моделі ідея була проста: користувач встановлює програму на свій комп'ютер, підключається до однорангової мережі й торгує напряму з іншими людьми. Жодних серверів компаній, жодного «великого брата», який контролює, що ви продаєте чи купуєте [3].

Головні плюси OpenBazaar виглядали дуже привабливо: ніхто не міг цензурувати лістинги, а комісій не було взагалі – ні за розміщення товару, ні за угоду. На папері це ідеально вкладалося в цінності децентралізованої торгівлі й цифрової свободи. Але на практиці виявилось складніше. Щоб користуватися платформою, потрібно було розібратися з установленням окремого клієнта, тримати свій вузол постійно онлайн і миритися з повільним та неінтуїтивним пошуком товарів. Для пересічного покупця це виглядало занадто технічно й незручно, тож частина потенційних користувачів просто не доходила до реального користування сервісом.

До того ж, механізми репутації та арбітражу були реалізовані лише частково. Рівень довіри, який дають звичайні маркетплейси з відгуками, захистом покупця тощо, OpenBazaar забезпечити не зміг. У результаті проєкт не зумів наростити достатню базу користувачів, зіткнувся з фінансовими проблемами й зрештою припинив роботу: розробку клієнта зупинили, а платформа в своєму первісному вигляді більше не працює.

Наступним кроком у розвитку децентралізованих рішень стали платформи на кшталт Origin Protocol та інші системи, що працюють на базі Ethereum. Якщо спростити, вони винесли «мозок» торгівлі в смарт-контракти, а дані про товари й угоди розмістили в розподілених сховищах на зразок IPFS [4]. Завдяки цьому користувачеві вже не обов'язково постійно бути онлайн: дані та бізнес-логіка живуть у децентралізованому середовищі, яке підтримується великою кількістю учасників.

Однак частина проблем нікуди не зникла. Багато таких проєктів упираються в складну токеноміку й високий поріг входу: щоб впливати на розвиток платформи, голосувати за зміни або виступати арбітром у спорах, часто потрібно купити чи заблокувати помітний обсяг нативних токенів. Їхня

ціна постійно стрибає, тож участь у житті платформи перетворюється на фінансовий ризик і виглядає мало привабливою для пересічного користувача.

Додатковий виклик – система рекомендацій: без централізованих серверів дуже складно зробити персональні підбірки товарів на основі історії поведінки конкретної людини так, щоб і не порушити приватність [5], і водночас не створити новий центр збору чутливих даних.

Окремий напрям у розвитку децентралізованих рішень – це протоколи, які працюють із реальними фізичними товарами. Один із найпомітніших прикладів став Boson Protocol, що пропонує нетиповий підхід до торгівлі так званими реальними світовими активами. Замість звичного ескроу, де гроші тимчасово тримає третя сторона, тут використовується система токенізованих зобов'язань: покупець отримує ваучер у вигляді NFT, а вся логіка виконання угоди знаходиться у протоколі [6].

Додатково застосовуються ігрові механіки, які мають мотивувати обидві сторони поводитися чесно. Продавець мусить внести заставу, яка може бути заблокована або частково, або повністю спалена, якщо він не виконає своїх зобов'язань, тож потреба в людському арбітражі значною мірою відходить на другий план.

З одного боку, це мінімізує роль посередників, з іншого – суттєво підвищує поріг входу для малих і середніх продавців, яким важко дозволити собі заморожувати значні суми під кожну окрему угоду. До того ж жорстко формалізовані правила, закладені в код протоколу, погано пристосовані до складних або нестандартних ситуацій, де важлива гнучка оцінка доказів, намірів і контексту – те, що зазвичай забезпечують живі арбітри.

Підсумовуючи проведений огляд, можна простіше сказати так: жоден із розглянутих підходів поки що не забезпечує комфортного поєднання зручності користування, прозорості процесів та захисту від зловживань. Централізовані маркетплейси й надалі залишаються беззаперечними лідерами за рівнем сервісу та звичністю для користувачів. Водночас це означає високу

залежність від адміністрації платформи: можливу цензуру, непрозорі правила арбітражу, а також закриті системи рейтингів і алгоритмів ранжування.

Децентралізовані рішення, навпаки, дають користувачеві більше контролю над власними активами та зменшують роль посередників [7]. Однак вони часто виявляються технічно складними, базуються на заплутаних економічних моделях і вимагають суттєвих фінансових вкладень для участі в забезпеченні безпеки системи. У результаті користувач змушений обирати між зручністю та прозорістю. Комплексного ж рішення, яке б одночасно поєднувало переваги обох наразі немає. Узагальнена порівняльна характеристика розглянутих рішень і їх ключових параметрів наведена в таблиці 1.1.

Таблиця 1.1

Порівняльна характеристика моделей організації електронної торгівлі

Тип рішення та приклади	Архітектура та принцип роботи	Переваги	Недоліки та бар'єри
Централізовані платформи (eBay, Amazon)	Клієнт-сервер. Єдиний оператор контролює інфраструктуру, бази даних та інтерфейси.	Висока швидкість і стабільність. Інтуїтивний UX/UI. Інтеграція з платіжними системами та логістикою.	Ризик блокування акаунту/коштів. Повна залежність від адміністрації. Цензура лістингів. Непрозорі рейтинги.
P2P мережі (OpenBazaar)	Однорангова мережа. Користувач – це вузол.	Відсутність комісій. Стійкість до цензури.	Технічна складність. Вузол повинен бути онлайн.

	через локальний клієнт	Відсутність центрального контролера.	Повільний пошук. Низька база користувачів.
Смарт-контракт платформи (Origin Protocol)	Блокчейн та IPFS. Бізнес-логіка в смарт-контрактах, дані у розподілених сховищах.	Прозорість і незмінність даних. Не потрібно тримати вузол онлайн. Відсутність єдиної точки відмови.	Складна токеноміка. Високий поріг входу. Складність реалізації персональних рекомендацій.
Протоколи фізичних активів (Boson Protocol)	Токенізація зобов'язань. Використання теорії ігор та фінансових застав.	Мінімізація людського фактора. Автоматичне виконання умов. Гарантії через фінансові втрати порушника.	Висока капіталоемність. Жорсткість правил. Негнучкість.

1.3 Постановка задачі

Узагальнюючи результати аналізу існуючих рішень та тенденцій розвитку електронної комерції, стає очевидним, що досягнення рівноваги між безпекою, прозорістю та зручністю залишається складним інженерним викликом. Сучасні платформи – від традиційних маркетплейсів до Web3-протоколів – часто змушені йти на архітектурні компроміси, віддаючи пріоритет одним характеристикам за рахунок інших. Саме необхідність нівелювання цих обмежень та створення комплексної моделі взаємодії визначає проблематику даної роботи.

Найбільш суттєвим недоліком сучасних централізованих платформ залишається асиметрія інформації та відсутність механізмів верифікації даних. Користувач позбавлений інструментів для перевірки коректності операцій: дані про замовлення, історія змін рейтингів і алгоритми врегулювання спорів функціонують у закритому режимі [8]. Це створює ситуацію вимушеної довіри, у якій учасники змушені покладатися на добросовісність адміністрації. Будь-які конфліктні ситуації вирішуються службою підтримки, причому її рішення є остаточними й непрозорими для зовнішнього аудиту.

Окремої уваги потребують підходи до формування репутації. Більшість існуючих систем обмежуються простим середнім балом, який, на жаль, надто легко спотворити за допомогою фейкових акаунтів чи накручених відгуків. Проте суха цифра рейтингу часто приховує реальну картину надійності продавця. Критично важливим, але часто ігнорованим показником є історія конфліктних ситуацій: як часто у продавця виникають диспути і, що найголовніше, на чию користь вони вирішуються. Адже висока оцінка за прості угоди втрачає сенс, якщо продавець систематично програє арбітражі у складних випадках, але це ніяк не відображається на його публічному профілі.

З іншого боку, аналіз наявних децентралізованих рішень висвітлює суттєві прогалини в організації арбітражу та захисту угод. Більшість блокчейн-орієнтованих маркетплейсів успішно вирішують технічну задачу усунення фінансового посередника, однак часто нехтують побудовою комплексної моделі врегулювання конфліктів, які неминуче виникають у торгівлі. Це призводить до крайнощів. В одних випадках платформи залишають за розробниками роль «верховного судді», що фактично нівелює ідею децентралізації та повертає нас до проблем централізованого контролю [9]. В інших – механізми захисту є настільки спрощеними (наприклад, автоматичне розблокування коштів за таймером), що вони не здатні адекватно оцінити нюанси ситуації, залишаючи добросовісних учасників беззахисними перед продуманими схемами шахрайства.

Не менш складним та нетривіальним викликом є побудова прозорої системи рекомендацій. На централізованих майданчиках алгоритми ранжування товарів традиційно працюють як «чорна скринька»: користувач не бачить логіки підбору і часто отримує пропозиції, що оптимізують не його досвід, а прибуток платформи через приховану рекламу чи просування товарів із вищою маржею [10]. У децентралізованому середовищі ми стикаємося з технічною дилемою. З одного боку, необхідно аналізувати реальну поведінку користувача щоб пропонувати йому дійсно корисні товари. З іншого – архітектура системи не повинна вимагати передачі цих чутливих даних на зовнішні сервери. Завдання полягає в тому, щоб знайти баланс: та забезпечити персоналізацію видачі, зберігши при цьому повну приватність даних на боці клієнта.

Враховуючи окреслену проблематику, метою роботи є створення комплексної моделі та програмна реалізація децентралізованої торговельної платформи, яка нівелює недоліки централізованих та існуючих блокчейн-аналогів. Проектована система має базуватися на технології розподіленого реєстру для забезпечення незмінності історії транзакцій, використовувати захищений ескроу-механізм, пропонувати прозорий і формалізований процес децентралізованого врегулювання спорів, а також інтегрувати стійку до маніпуляцій систему репутації та алгоритми персоналізованих рекомендацій зі збереженням приватності користувача.

Для досягнення поставленої мети необхідно вирішити таку сукупність взаємопов'язаних задач:

1. Формалізувати бізнес-процеси платформи, визначивши повний життєвий цикл товару та замовлення, починаючи від створення лістингу й депонування коштів у ескроу контракті до підтвердження виконання зобов'язань, а також сценаріїв ініціювання та технічного завершення процедури диспуту.
2. Розробити концептуальну модель децентралізованого арбітражу, яка регламентує ролі учасників, правила подання претензій та критерії

ухвалення рішень, приділяючи особливу увагу механізмам фіксації результатів розгляду в блокчейні для забезпечення незмінності та прозорості вердиктів.

3. Спроекувати архітектуру смарт-контрактів, що передбачає модульний розподіл зон відповідальності між ринком, системою арбітражу та реєстром користувачів, що дозволить мінімізувати права доступу для кожного компонента, формалізувати перехідні стани угоди та гарантувати безпеку виконання коду.
4. Побудувати комплексну математичну модель репутації, яка замість лінійного усереднення базується на статистичних вагах і враховує історію участі в диспутах та результативність їх вирішення для кожної сторони.
5. Розробити механізм прозорого ранжування товарів, де позиція у видачі визначається відкритою формалізованою функцією на основі показників репутації, релевантності та історії продажів.
6. Створити алгоритми гібридних рекомендацій, які поєднують аналіз глобальних трендів з блокчейну, зокрема показників популярності та патернів попиту, із локальною обробкою поведінкової історії на пристрої користувача, забезпечуючи персоналізацію без передачі чутливих даних на зовнішні сервери.
7. Здійснити програмну реалізацію платформи, імплементувавши запропоновані моделі арбітражу, рейтингування та рекомендацій на базі блокчейн-інфраструктури.
8. Провести експериментальну оцінку ефективності розроблених рішень.

1.4 Вибір програмних засобів для реалізації

Реалізація запропонованої архітектури децентралізованої платформи вимагає формування технологічного стеку, здатного забезпечити виконання ключових вимог дослідження: безпеки смарт-контрактів, прозорості

арбітражу стійкої до маніпуляцій системи рейтингів, формалізованого ранжування товарів та ефективної взаємодії користувача з блокчейном. Вибір інструментарію базується на аналізі функціональних можливостей сучасних Web3-рішень та їх придатності для швидкого прототипування, що є критично важливим на етапі перевірки архітектурних і математичних моделей, закладених у платформ.

Для імплементації програмної логіки ринку, ескроу-механізмів та системи рейтингів доцільно обрати мову програмування Solidity. Вона є стандартом для віртуальної машини Ethereum, що забезпечує сувору типізацію та доступ до перевірених патернів безпеки. Наявність бібліотек на кшталт OpenZeppelin спрощує реалізацію типових механізмів керування ролями, обмеження доступу та зберігання структурованих даних [11]. Саме на рівні смарт-контрактів повинні базуватися основні бізнес правила: умови замороження та розблокування коштів, можливі стани угоди в процесі вирішення диспуту, правила врахування голосів сторін або арбітрів, а також формальні обмеження на зміну репутаційних показників продавців і товарів.

Компіляція, налагодження, тестування та розгортання смарт-контрактів пропонується здійснювати у середовищі розробки Hardhat. Цей фреймворк надає гнучку систему конфігурації, тісну інтеграцію з мовами JavaScript та TypeScript, підтримку великої кількості плагінів і зручні інструменти для автоматизованого тестування сценаріїв роботи платформи. У межах дослідження система планується розгортатися в локальній мережі Hardhat Network. Такий підхід дозволяє повністю контролювати стан блокчейну, відтворювати складні сценарії взаємодії між учасниками, моделювати поведінку добросовісних і недобросовісних агентів, а також проводити численні експерименти без реальних фінансових витрат на комісії [12]. Можливість миттєвого підтвердження транзакцій дає змогу швидко перевіряти коректність переходів між станами угод, логіку відкриття та закриття диспутів, а також вплив тих чи інших подій на оновлення рейтингів.

Оскільки безпосереднє зберігання мультимедійних даних та великих текстових описів у смарт-контракті є технічно та економічно неефективним через обмеження розміру блоку і високу вартість операцій запису. Для вирішення цієї задачі децентралізованого хостингу контенту обрано протокол IPFS. Він забезпечує незмінність даних завдяки адресації за змістом: кожен товар, його зображення та розширений опис зберігаються як об'єкти, що мають унікальний ідентифікатор вмісту [13]. У блокчейні фіксуються лише ці ідентифікатори, що дає змогу відтворити потрібні дані у будь-який момент часу та уникнути надмірного зростання стану мережі. Для підвищення надійності доступу до контенту додатково застосовується сервіс пінінгу Pinata, який виступає своєрідним шлюзом-сховищем і забезпечує персистентність об'єктів IPFS.

Для реалізації клієнтської частини платформи було обрано фреймворк Next.js, вибір якого зумовлений його гнучкістю в побудові модульної архітектури та високою продуктивністю. Також важливим аспектом є те, що Next.js дозволяє виконувати значну частину логіки на стороні клієнта. Це дає змогу реалізовувати локальну обробку поведінкових даних користувача з метою подальшого їх використання для побудови персональних рекомендацій без передачі персональних даних користувачів на віддалені сервери.

Комунікацію між інтерфейсом та блокчейн-мережею забезпечуватиме бібліотека Ethers.js, яка відповідає за підключення до провайдера мережі, читання стану смарт-контрактів, обробку подій і безпечне формування транзакцій [14]. Це дасть змогу винести низькорівневу логіку запитів у відокремлені сервісні модулі, ізолювавши компоненти інтерфейсу від прямої взаємодії з кодом контрактів. Крім того, бібліотека забезпечує безперервне прослуховування подій, що генеруються в мережі. Як тільки відбувається зміна статусу диспуту чи перерахунок репутації, відповідні сигнали перехоплюються застосунком, що ініціює миттєве оновлення даних на екрані користувача без необхідності перезавантаження сторінки.

Ідентифікація користувачів буде здійснюватися за допомогою MetaMask. Цей гаманець дозволяє реалізувати схему, де контроль над доступом залишається повністю в руках власника акаунту. Замість того щоб довіряти серверу, користувач підписуватиме кожен транзакцію у своєму браузері. Такий підхід гарантуватиме, що ніхто сторонній не зможе втрутитися в процес оформлення замовлення чи вирішення спору.

У сукупності обраний технологічний стек, що поєднує Solidity та Hardhat для розроблення і тестування смарт-контрактів, IPFS і Pinata для децентралізованого зберігання контенту, Next.js для побудови клієнтської частини, а також Ethers.js і MetaMask для взаємодії з блокчейном, формує цілісне середовище для реалізації запропонованої моделі децентралізованої платформи електронної комерції. Така комбінація технологій забезпечує необхідний рівень безпеки, прозорості та контрольованості експериментів, а також дозволяє реалізувати ключові функціональні вимоги дослідження: прозорий арбітраж, відтворювану систему рейтингів, формально задане ранжування товарів і персоналізовані рекомендації без порушення приватності користувачів.

Висновки до першого розділу

У першому розділі було проаналізовано, як зараз влаштована електронна комерція, і виділено головні проблеми централізованих маркетплейсів. Попри високий рівень сервісу, зручний інтерфейс та готову інтеграцію з платіжними й логістичними сервісами, такі платформи зосереджують контроль у руках адміністрації. Правила арбітражу залишаються малозрозумілими, а системи репутації – закритими. У результаті виникає інформаційна нерівність: користувачі змушені довіряти платформі, не маючи можливості реально перевірити механізми, які впливають на їхні угоди та рейтинги.

Також було здійснено аналіз низки децентралізованих рішень, який показав, що відмова від посередника й використання блокчейну справді

підвищують прозорість і дають більше контролю над власними активами. Водночас це створює нові виклики: такі системи часто занадто складні для пересічного користувача, мають високий поріг входу, базуються на жорстко формалізованих правилах, а механізми арбітражу та репутації реалізовані неповною мірою. У підсумку жоден із розглянутих підходів не поєднує одночасно зручність, прозорість і надійний захист від зловживань.

На основі проведеного дослідження, було сформульовано мету роботи: розробити модель децентралізованої торговельної платформи, яка зменшує недоліки як класичних централізованих рішень, так і існуючих блокчейн-підходів. Визначено ключові задачі дослідження: формалізувати бізнес-процеси, спроектувати механізми арбітражу, репутаційну систему та систему рекомендацій. Також обґрунтовано вибір технологічного стеку, який слугуватиме основою для подальшої реалізації й експериментальної перевірки запропонованої моделі.

РОЗДІЛ 2 ПРОЄКТУВАННЯ ДЕЦЕНТРАЛІЗОВАНОЇ ПЛАТФОРМИ ЕЛЕКТРОННОЇ КОМЕРЦІЇ

2.1 Архітектура платформи та формалізація бізнес-процесів

Проектування сучасних децентралізованих систем вимагає комплексного підходу до вибору архітектурних патернів, оскільки помилки на етапі планування можуть призвести до критичних вразливостей, втрати коштів користувачів або неможливості масштабування системи в майбутньому.

В основі проєктованої платформи лежить концепція повної децентралізації, що виключає необхідність використання централізованих серверів для обробки критично важливих даних. Для реалізації цієї концепції пропонується архітектурний підхід, що передбачає розділення системи на три логічні рівні: рівень представлення, рівень бізнес-логіки та рівень даних. Така сегрегація дозволяє забезпечити гнучкість розробки та підвищити безпеку системи [15]. Загальна схема запропонованої архітектури наведена на рисунку 2.1.

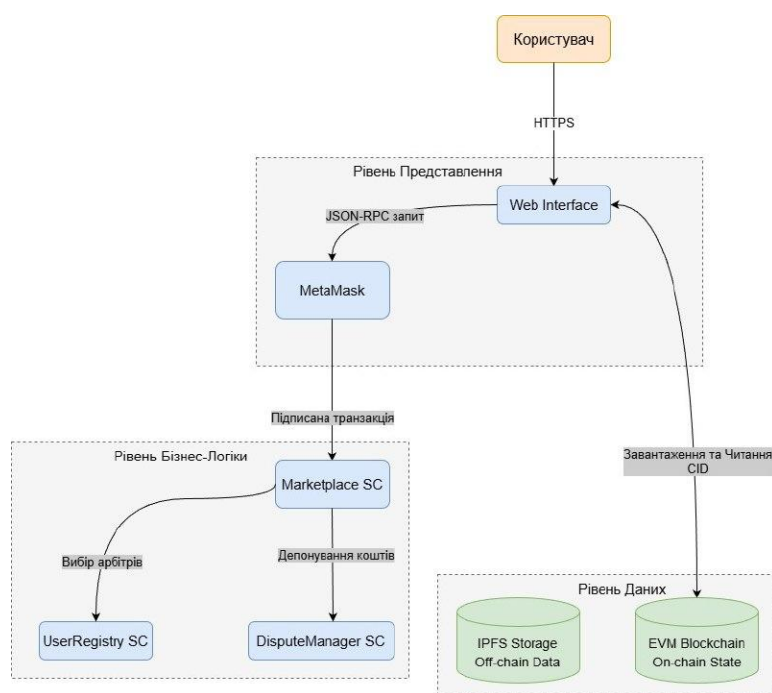


Рис. 2.1 – Трирівнева архітектура Web3-додатку

Розглянемо детальніше функціональне призначення та взаємодію рівнів, зображених на схемі, спираючись на обґрунтований інструментарій. Рівень

представлення виступає інтерфейсом взаємодії між кінцевим користувачем та блокчейн-мережею. Архітектурно цей рівень спроектовано як «тонкий клієнт», який не зберігає бізнес-логіки, пов'язаної з обробкою коштів, а лише формує запити та відображає стан системи [16].

Ключовою особливістю даного рівня є його ізоляція від приватних ключів користувачів: підписання транзакцій делегується зовнішньому провайдеру (гаманцю), що інтегрується в браузер. Це створює надійний контур безпеки, де компрометація веб-інтерфейсу не надає зловмиснику прямого доступу до активів користувача [17]. Взаємодія з нижніми рівнями відбувається асинхронно через механізм прослуховування подій, що дозволяє динамічно оновлювати інтерфейс при зміні стану смарт-контрактів.

Рівень бізнес-логіки є децентралізованим ядром системи. На відміну від класичної клієнт-серверної архітектури, де бекенд є закритою «чорною скринькою», тут логіка виконання угод є публічною та незмінною. Цей рівень складається з сукупності смарт-контрактів, які виконують роль детермінованого автомата станів [18]. Вони відповідають за валідацію вхідних даних (наявність товару, достатність балансу), управління життєвим циклом угоди (ескроу, арбітраж) та вибір арбітрів для вирішення потенційних диспутів. Виконання коду на цьому рівні гарантується консенсусом блокчейн-мережі, що забезпечує неможливість цензури або односторонньої зміни правил гри адміністратором платформи.

Третім рівнем є рівень даних, який реалізує гібридну модель зберігання для оптимізації вартості використання системи. Архітектура передбачає чіткий розподіл типів даних на дві категорії. До першої відносяться ончейн-дані, тобто критично важлива інформація, що вимагає консенсусної верифікації (баланси, статуси замовлень, хеші угод, репутаційні метрики), яка зберігається безпосередньо у сховищі смарт-контрактів. Другу категорію складають офчейн-дані, зокрема статичний контент великого обсягу (зображення товарів, детальні текстові описи, докази в диспутах), який зберігається у розподіленій файловій системі IPFS. Зв'язок між цими двома

підсистемами забезпечується через збереження унікального криптографічного ідентифікатора контенту (CID) у смарт-контракті, що гарантує цілісність та незмінність даних, завантажених офчейн.

Після визначення технічного базису, необхідно виконати концептуальну формалізацію бізнес-процесів платформи. Важливим етапом проєктування є визначення ключових суб'єктів системи та їхніх зон відповідальності. У межах платформи виділяються чотири основні ролі: покупець, продавець, арбітр та адміністратор. Взаємодія між ними відбувається не через довіреного посередника, а через набір децентралізованих протоколів, що візуалізовано на діаграмі варіантів використання (рис. 2.2).



Рис. 2.2 – Діаграма варіантів використання платформи

Наведена діаграма демонструє, що кожна роль має чітко окреслений набір доступних функцій. Продавець ініціює наповнення платформи

контентом. Алгоритм його дій починається зі створення лістингу, де фіксуються параметри товару (ціна, опис, кількість). Після завантаження медіа-даних в IPFS та отримання хешу, продавець підписує транзакцію для запису метаданих у смарт-контракт. Важливою особливістю проєктування є неможливість зміни критичних параметрів товару після оформлення замовлення, що захищає покупців від маніпуляцій. Тільки після успішного підтвердження транзакції мережею товар переходить у стан «Активний» і стає доступним для пошуку в загальному каталозі.

Покупець фокусується на пошуку товарів та ініціюванні угод. Ключовим бізнес-процесом тут є життєвий цикл замовлення. Він починається з моменту виклику транзакції купівлі, коли система виконує атомарну перевірку наявності товару та блокує кошти на смарт-контракті. Далі процес розвивається за одним із двох сценаріїв. У «оптимістичному сценарії» продавець виконує свої зобов'язання, доставляє товар, а покупець підтверджує отримання, що призводить до розблокування коштів. У випадку ж виникнення проблем реалізується альтернативний потік, де управління передається арбітрам [19].

Для забезпечення коректності роботи системи критично важливо відстежувати стани замовлення, оскільки саме вони визначають доступні дії для кожної ролі в конкретний момент часу. Наприклад, залишити відгук можна лише в стані «Завершено», а відкрити диспут – лише в стані «Очікування». Переходи між цими станами суворо регламентовані логікою смарт-контракту, що унеможливорює порушення послідовності дій.

Окрім основних учасників, в архітектурі передбачено технічну роль адміністратора. Його повноваження суворо обмежені налаштуванням глобальних параметрів (розмір комісії, список резервних арбітрів) і не дозволяють втручатися в конкретні угоди, що гарантує децентралізований характер платформи.

Отже, розроблена архітектура та деталізовані бізнес-процеси формують структурну основу платформи, забезпечуючи прозорість та безпеку взаємодії

учасників. Визначені ролі та стани системи є вихідними даними для проєктування спеціалізованих механізмів захисту угод та управління репутацією, яким присвячено наступні підрозділи.

2.2 Модель децентралізованого арбітражу та ескроу-механізму

Ключова проблема децентралізованих торгових систем у тому, як змусити сторони виконувати свої зобов'язання без участі централізованого посередника, який у класичних моделях електронної комерції виступає гарантом чесної угоди. Коли немає третьої сторони, якій довіряють усі, потрібно будувати алгоритми, що зможуть автоматично або майже автоматично розв'язувати конфлікти: якщо товар не доставили, він не відповідає опису або ж одна зі сторін намагається шахраювати. Далі розглянемо детально вже наявні підходи до розв'язання цієї задачі.

Аналізуючи існуючі рішення, можна виокремити кілька основних підходів, кожен із яких має свої плюси та серйозні мінуси. Найпоширеніша модель – це централізований ескроу, який використовують звичайні маркетплейси. У такій схемі гроші покупця заморожуються на рахунку платформи, поки він не підтвердить, що отримав товар. Для користувача це виглядає просто й логічно, але на практиці створюється єдина точка відмови: адміністратор платформи технічно здатен заблокувати кошти, цензурувати транзакції або навіть привласнити активи – через злий умисел чи внаслідок хакерської атаки. До того ж ця модель передбачає повну довіру до оператора, що суперечить базовим принципам децентралізації.

Альтернативний варіант – це модель мультипідпису, яку свого часу активно використовували перші децентралізовані майданчики. Ідея проста: створюється спеціальна адреса, якою керують три ключі – покупця, продавця та обраного модератора. Щоб розблокувати кошти, потрібно, щоб погодилися будь-які двоє з трьох [20]. Однак у такого підходу є суттєвий мінус – ризик змови. Оскільки модератора зазвичай обирають ще до угоди, продавець може

заздалегідь домовитися з ним і спробувати неправомірно отримати гроші. До того ж ця схема раз і назавжди зафіксована і не враховує, як змінюється репутація модератора з часом.

Більш сучасний підхід – це протоколи децентралізованого правосуддя, які покладаються на краудсорсинг для розв’язання спорів. Там роль присяжних виконують випадково обрані власники токенів протоколу, які економічно зацікавлені голосувати чесно. Такі системи добре захищені від цензури, але мають свої практичні проблеми: їх складно вбудувати у звичайну електронну комерцію через залежність від волатильних зовнішніх токенів, високу вартість залучення присяжних і значні затримки в часі, поки збираються та обробляються всі голоси [21].

Система має бути максимально автономною: усе повинно працювати на базі смарт-контрактів, без ручного втручання адміністраторів. Механізм вибору арбітрів повинен бути захищеним від передбачуваності та змов, тож потрібні надійні джерела випадковості. Вартість арбітражу слід звести до мінімуму, аби вона не ставала перешкодою для угод із недорогими товарами. Окрему роль має відігравати репутація: вона має бути повноцінним активом у системі. Право вирішувати спори повинні отримувати лише ті учасники, які своєю торговою історією вже підтвердили чесність і надійність.

На основі сформульованих вимог пропонується дворівнева модель внутрішнього арбітражу. Її головна ідея в тому, що арбітрів обирають не ззовні, а з-поміж найкращих продавців самої платформи. Так формується «замкнена» екосистема, у якій учасники зацікавлені підтримувати порядок: будь-яке шахрайство зменшує привабливість майданчика, на якому вони самі заробляють [22]. Для того щоб формалізувати процес вирішення конфліктів, доцільно побудувати діаграму машини станів, яка наочно показує, як саме система переходить між етапами розгляду спору і за яких умов це відбувається. На рисунку 2.3 зображена діаграма машини станів внутрішнього арбітражу, що демонструє послідовність етапів диспуту та події, які запускають відповідні переходи між ними.



Рис. 2.3 – Діаграма станів процесу врегулювання спору

Як видно з діаграми станів, запропонований механізм диспутів організований як послідовність фаз із чітко визначеними правилами переходу між ними. Така побудова дозволяє уникнути невизначеності в поведінці системи й гарантує, що будь-який спір буде завершений за скінченний проміжок часу, а кошти не залишаться заблокованими назавжди [23].

Перший етап після відкриття диспуту відповідає фазі подання доказів покупцем. Покупець може ініціювати спір лише тоді, коли кошти ще перебувають у стані депонування, а строк автоматичного підтвердження доставки тривалістю сім днів ще не минув. Від моменту відкриття диспуту відраховується фіксований час тривалістю сімдесят дві години, протягом якого покупець повинен завантажити свої докази до децентралізованого сховища, зокрема текстовий опис проблеми та супровідні матеріали на зразок фотографій. Якщо у відведений час докази не надано, система автоматично завершує диспут на користь продавця, розблоковуючи ескроу на його користь.

Якщо покупець вчасно подав докази, система переходить до наступної фази – а саме подання доказів вже продавцем. Для продавця передбачене окреме вікно тривалістю сорок вісім годин, під час якого він може надати власні контраргументи та матеріали. Ігнорування цього етапу трактується як фактичне визнання претензій: після завершення відведеного часу без жодних дій продавця рішення автоматично ухвалюється на користь покупця, а кошти повертаються йому.

Коли обидві сторони надали докази, диспут переходить у фазу первинного арбітражу. На цьому етапі автоматично відбираються арбітри з пулу продавців платформи. Відбір відбувається з урахуванням низки умов, що підвищують неупередженість і якість рішень. Потенційний арбітр не може бути ні покупцем, ні продавцем у поточному диспуті. Його індивідуальний рейтинг має перевищувати середній рейтинг серед усіх продавців платформи, що відсіює посередніх та проблемних учасників.

Арбітр також повинен мати зафіксовану активність у системі протягом останніх тридцяти днів, що підтверджує його актуальну залученість. Додатково враховується історія участі в попередніх спорах: користувачі, які програвали диспути протягом останніх тридцяти днів, не можуть бути обрані арбітрами в нових конфліктах. Для забезпечення непередбачуваності та стійкості до змов сам вибір конкретних арбітрів здійснюється псевдовипадковим чином на основі даних блокчейну, зокрема хешу попереднього блока, що істотно ускладнює спроби завчасно вплинути на склад суддів. Обрані арбітри отримують доступ до матеріалів справи та мають обмежений час для голосування, рішення ухвалюється простою більшістю.

Якщо первинний арбітраж не приводить до остаточного рішення, наприклад через відсутність достатньої кількості чи пропущені терміни голосування, задіюється наступний рівень – довірений арбітраж. На цьому етапі до розгляду спору залучаються попередньо визначені учасники адміністратором. Для цієї фази також встановлено окреме фіксоване часове вікно тривалістю сорок вісім годин, протягом якого довірені арбітри повинні

ухвалити остаточний вердикт. Якщо ж навіть на цьому рівні рішення не було прийнято вчасно, система автоматично завершує диспут на користь покупця. Така поведінка підкреслює пріоритетний характер захисту покупця.

Завершальна фаза відповідає стану «Завершено». У цьому стані смарт-контракт виконує розподіл коштів відповідно до ухваленого рішення: у разі перемоги покупця вся сума замовлення повертається йому, у разі перемоги продавця ескроу розблоковується на його користь з урахуванням передбаченої комісії платформи. У моделі передбачені сценарії завершення для всіх можливих варіантів бездіяльності сторін та арбітрів, тому ситуація, коли кошти залишаються заблокованими на невизначений термін, принципово виключена, а загальний дизайн системи чітко орієнтований на максимальний рівень захисту інтересів покупця.

2.3 Математична модель рейтингу товарів та репутації продавців

Наступним етапом є проектування рейтингової моделі товарів для продавців. Проте спочатку необхідно визначити, які саме властивості така модель повинна включати в себе та які існуючі підходи вже пропонують часткові відповіді на ці вимоги. У сфері електронної комерції основні труднощі пов'язані з тим, що просте середнє дуже чутливе до поодиноких оцінок, складно коректно порівнювати товари й продавців із різною кількістю відгуків, а ще потрібно зважати не лише на оцінки користувачів, а й на реальну історію виконаних угод і наявних конфліктів.

Аналіз існуючих реалізацій у децентралізованому просторі демонструє кілька спроб вирішення цієї проблеми, кожна з яких має свої обмеження. Першим помітним підходом є модель Web of Trust, яка використовується в протоколах на кшталт PGP або ранніх версіях платформи OpenBazaar. У цій моделі довіра є транзитивною: якщо користувач А довіряє користувачу В, а В довіряє С, то А певною мірою довіряє С. Однак такий підхід погано масштабується і створює бар'єри для входу нових учасників, які не мають

зв'язків з існуючим графом довіри [24]. Крім того, у OpenBazaar рейтинги зберігалися локально на вузлах користувачів, що призводило до втрати репутаційної історії, якщо продавець йшов з мережі.

Іншим поширеним підходом є Token-Curated Registries, де спільнота голосує токенами за включення учасників до «білих списків» надійних продавців. Хоча цей механізм є стійким до цензури, він має тенденцію перетворюватися на плутократію, де рішення приймають ті, хто має більше капіталу, а не реального досвіду взаємодії. Це створює ризики для звичайних користувачів, які можуть постраждати від змови великих власників токенів.

Також варто розглянути сучасні протоколи соціальної репутації, зокрема Lens Protocol і Farcaster, які формують ончейн-граф соціальних зв'язків і взаємодій між користувачами. Однак ці рішення переважно орієнтовані на соціальні платформи, де наслідки помилкової оцінки зазвичай обмежуються репутаційними втратами або погіршенням якості рекомендацій [25]. У електронній комерції ціна помилки значно вища, оскільки довіра напряму впливає на фінансові ризики сторін та ймовірність шахрайства.

Тому для розроблюваної платформи доцільно створити власну репутаційну модель, яка поєднує ончейн-прозорість із механізмами фінансової відповідальності та забезпечує максимально об'єктивне формування рейтингу.

Найбільш очевидна відправна точка – просто взяти середнє арифметичне всіх оцінок. Якщо для товару або продавця відомі всі виставлені користувачами бали за шкалою від одного до п'яти, інтегральний рейтинг можна визначити як середнє значення цих оцінок. Такий підхід легко пояснити й інтуїтивно зрозуміти, але він має суттєвий недолік: система однаково ставиться до об'єктів із великою історією продажів і до тих, які отримали всього одну-дві оцінки. У підсумку нові товари чи продавці з кількома випадково високими відгуками можуть опинитися вище у рейтингу, ніж досвідчені учасники з великою, але більш різномірною історією, навіть якщо на практиці саме вони є надійнішими.

Альтернативний підхід – використовувати статистичні інтервали довіри, наприклад, будувати рейтинг за нижньою межею довірчого інтервалу в моделі Вілсона. Однак подібні методи важче інтерпретувати, їх складніше пояснити звичайним користувачам, а також вони потребують ретельного налаштування параметрів, щоб уникнути небажаних перекосів у результатах [26].

З огляду на те, що платформа працюватиме в блокчейновому середовищі, до рейтингової моделі висувається ще одна принципова вимога: усі ключові показники мають обчислюватися на основі вже агрегованих даних, що зберігаються на рівні смарт-контрактів, і так, щоб будь-який клієнтський застосунок міг відтворити їхні значення без додаткових припущень. Це підштовхує до використання байєсівського згладжування середнього значення. Інтуїтивно ця ідея виглядає так: ми не розглядаємо середню оцінку окремого товару чи продавця у відриві від контексту, а поєднуємо її з глобальним середнім рейтингом по всій платформі [27]. У такій постановці об'єкти з малою кількістю відгуків «підтягуються» до загального середнього, яке відіграє роль апріорного очікування, а ті, що мають велику історію угод і відгуків, поступово відхиляються від нього і відображають уже власний, більш інформативний профіль репутації.

Для товарів у межах платформи використовується така модель. Нехай оцінка кожного товару задається по шкалі від одиниці до п'яти. Для конкретного товару відома середня оцінка $\bar{x}$ та кількість відгуків v . Крім того, за всіма товарами платформи обчислюється глобальне середнє значення C , яке відображає типову оцінку в системі. Додатково вводиться параметр m , який задає вагу довіри до глобального середнього і може інтерпретуватися як умовна кількість віртуальних відгуків на рівні C . Тоді байєсівський рейтинг товару визначається за наступною формулою:

$$R_{\text{товар}} = \frac{v}{v + m} \bar{x} + \frac{m}{v + m} C, \quad (2.1)$$

У запропонованій моделі параметр m для товарів приймає значення п'ять, що означає, що система трактує глобальне середнє як еквівалент п'яти умовних оцінок, доданих до реальної історії товару. Якщо товар ще не має жодної оцінки, його рейтинг збігається з глобальним середнім; із накопиченням відгуків вплив власної статистики зростає, а вплив апріорного припущення зменшується. На рівні смарт-контрактів буде зберігатися лише сума оцінок і їх кількість, а також глобальні агрегати, проте будь-який клієнт може відновити байєсівський рейтинг, використовуючи наведене співвідношення.

Для продавців базова схема залишається аналогічною. Зберігається середній бал, який покупці виставляють за результатами угод, та кількість таких оцінок. Для всієї сукупності продавців обчислюється глобальне середнє $C_{\text{прод}}$ і на його основі визначається байєсівський рейтинг конкретного продавця. Формально це можна записати як:

$$R_{\text{прод}} = \frac{v_{\text{прод}}}{v_{\text{прод}} + m_{\text{прод}}} \overline{x_{\text{прод}}} + \frac{m_{\text{прод}}}{v_{\text{прод}} + m_{\text{прод}}} C_{\text{прод}}, \quad (2.2)$$

де $\overline{x_{\text{прод}}}$ – середня оцінка конкретного продавця, $v_{\text{прод}}$ – кількість отриманих ним оцінок, а $m_{\text{прод}}$ – параметр згладжування, який також дорівнює п'яти. Таким чином рейтинг новачка із невеликою кількістю угод не змінюється занадто різко через один вдалий або невдалий відгук, а поступово відхиляється від глобального середнього у міру накопичення статистики.

Однак для продавців простого врахування оцінок від покупців недостатньо. Критичною характеристикою є не лише загальний рівень задоволеності, а й поведінка продавця у конфліктних ситуаціях. Тому в моделі репутації вводиться окремий компонент, пов'язаний із результатами спорів. Для кожного продавця розглядаються три величини: кількість успішно завершених угод без конфлікту, загальна кількість спорів і кількість спорів, у яких покупець визнаний правим, тобто відповідальність за проблему покладено саме на продавця. На основі цих величин визначається показник надійності щодо диспутів. Нехай S позначає кількість успішно завершених

замовлень, D – загальну кількість спорів, а D_f – кількість спорів, у яких фінальне рішення формально зафіксувало вину продавця. Тоді загальна кількість взаємодій, що враховується, дорівнює $S + D$. Якщо продавець ще не має жодної завершеної угоди чи спору, то йому приписується максимальний початковий рівень довіри за цим критерієм. В усіх інших випадках коефіцієнт надійності щодо диспутів визначається співвідношенням:

$$Q_{\text{диспут}} = 1 - \frac{D_f}{S + D}, \quad (2.3)$$

причому, щоб уникнути некоректних значень за екстремальних випадків, результат додатково обмежується знизу нулем. Така модель принципово штрафує продавця лише за ті конфлікти, у яких його вина була формально підтверджена, і не карає автоматично за сам факт наявності спорів, оскільки частина з них може завершуватися на його користь.

Отримавши два незалежні компоненти – байєсівський рейтинг, який відображає якість взаємодії у нормальних умовах, і коефіцієнт надійності щодо спорів, який відображає поведінку в конфліктних сценаріях, платформа формує інтегральний показник репутації продавця. Для цього обидва компоненти приводяться до єдиного масштабу. Байєсівський рейтинг, що заданий у діапазоні від одиниці до п'яти, нормалізується до відрізка від нуля до одиниці шляхом ділення на п'ять. Коефіцієнт надійності вже знаходиться в цьому діапазоні. Далі визначається зважена комбінація цих двох величин, у якій більша частка припадає на оцінки покупців, а менша – на історію спорів. У поточній реалізації інтегральний нормалізований показник обчислюється за формулою:

$$T = w_{\text{рейтинги}} \times \frac{R_{\text{прод}}}{5} + w_{\text{диспут}} \times Q_{\text{диспут}}, \quad (2.4)$$

де $w_{\text{рейтинги}} = 0,7$, а $w_{\text{диспут}} = 0,3$, причому сума ваг дорівнює одиниці. Після цього значення T , що лежить між нулем і одиницею, відображається назад у звичну п'ятибальну шкалу шляхом афінного перетворення:

$$R_{\text{кінцевий}} = 1 + 4T, \quad (2.5)$$

Таким чином інтегральний рейтинг продавця завжди належить інтервалу від одиниці до п'яти, причому мінімальне значення відповідає ситуації, коли і оцінки покупців, і історія спорів сформували максимально негативну картину, а максимальне – коли обидві складові наближаються до ідеального рівня.

Запропонована модель свідомо розподіляє протокольну частину, яка фіксується на рівні блокчейну у вигляді агрегованих сум, кількостей оцінок та результатів спорів, і обчислювальну частину, що виконується в клієнтському застосунку. Завдяки цьому вдається поєднати прозорість і відтворюваність: будь-який клієнт може, використовуючи ті самі ончейн-дані й наведені формули, самостійно обчислити рейтинг товару або продавця та перевірити його коректність. Водночас складніша математична модель не обтяжує смарт-контракти надмірною логікою і не призводить до різкого зростання вартості транзакцій.

2.4 Модель ранжування товарів та персональних рекомендацій

У сучасних умовах, коли ринок переповнений великою кількістю товарів і сервісів, важливо мати чітку й ефективну систему, яка допомагає людям швидко знаходити потрібне. Проблема багатьох централізованих платформ у тому, що їхні алгоритми рекомендацій залишаються непрозорими – працюють як «чорна скриня», де результати можуть підлаштовуватись під комерційні інтереси самої платформи, а не користувача. У децентралізованому середовищі з'являються додаткові виклики. Через обмежені обчислювальні ресурси блокчейна та високу вартість операцій, неможливо напряму запускати складні алгоритми машинного навчання [28].

Тому пропонується створити гібридну архітектуру системи рекомендацій, яка поєднує відкриті ончейн-дані з індивідуальною історією активності користувача, що зберігається локально на його пристрої. Система умовно поділяється на два окремі компоненти: глобальне ранжування товарів

і персоналізовані рекомендації. Такий підхід дає змогу вирішити проблему «холодного старту» – нові користувачі отримують релевантні загальні результати, а активні – індивідуальні рекомендації з високою точністю.

Модель глобального ранжування має базуватися на інтегральному показнику якості товару $I_{\text{глоб}}$, який обчислюється для кожного активного лоту. Цей показник є зваженою сумою чотирьох нормованих компонентів, що відображають різні аспекти привабливості товару. Формально модель записується так:

$$I_{\text{глоб}} = w_1 \times R + w_2 \times P + w_3 \times T + w_4 \times D, \quad (2.6)$$

У наведеній формулі R позначає нормований байєсівський рейтинг товару, приведений до діапазону від нуля до одиниці. Другий доданок P відповідає показнику популярності, що визначається як функція від кількості продажів, також нормована до одиниці. Третій компонент T являє собою фактор новизни, який надає перевагу новим надходженням, запобігаючи стагнації видачі. Останній доданок D відображає надійність продавця, скориговану на історію диспутів. Для моделі пропонується використання наступних емпіричних вагових коефіцієнтів: $w_1 = 0,5$ для пріоритету якості товару, $w_2 = 0,20$ для врахування популярності, $w_3 = 0,1$ для підтримки новинок та $w_4 = 0,20$ для врахування надійності продавця. Такий розподіл ваг дозволяє збалансувати інтереси покупців, які шукають перевірені товари, та продавців, які виводять на ринок нові продукти.

Для забезпечення персоналізації розроблено концепцію розрахунку коефіцієнта подібності. Оскільки зберігання повного графу взаємодій користувачів у блокчейні є економічно недоцільним, система проєктується для аналізу локальної історії дій користувача. Подібність має розраховуватися окремо для історії переглядів $K_{\text{пер}}$ та історії покупок $K_{\text{пок}}$. Ключовим елементом моделі є функція часового затухання, яка зменшує вплив старих дій. Вага кожної події $\lambda(t)$ визначається як гіперболічна функція від часу:

$$\lambda(t) = \frac{1}{1 + \Delta t}, \quad (2.7)$$

де Δt – час, що минув з моменту дії. Це дозволяє системі швидко адаптуватися до зміни інтересів користувача. Загальний коефіцієнт подібності для конкретного товару-кандидата $K_{\text{скуп}}$ визначається як лінійна комбінація подібності за покупками та переглядами:

$$K_{\text{скуп}} = 0,7 \times K_{\text{пок}} + 0,3 \times K_{\text{пер}}, \quad (2.8)$$

Вищий коефіцієнт для покупок відображає той факт, що фінансова транзакція є значно сильнішим сигналом зацікавленості, ніж пасивний перегляд сторінки. При розрахунку подібності передбачається врахування збігу категорії товару та ідентифікатора продавця.

Фінальний етап ранжування полягає у формуванні гібридного рейтингу $I_{\text{кінц}}$, який використовується для сортування персоналізованої видачі. Цей рейтинг поєднує об'єктивну якість товару та суб'єктивну релевантність для конкретного користувача:

$$I_{\text{кінц}} = 0,5 \times I_{\text{глоб}} + 0,5 \times K_{\text{скуп}}, \quad (2.9)$$

Використання рівних ваг забезпечує компроміс між рекомендацією якісних популярних товарів та товарів, що точно відповідають минулим інтересам користувача. Така модель дозволяє уникнути ефекту «бульбашки фільтрів», коли користувачу показують лише однотипні товари, і водночас гарантує, що у рекомендації не потраплять товари низької якості навіть за високої тематичної відповідності. Запропонована гібридна модель є стійкою до маніпуляцій, оскільки базується на верифікованих даних блокчейну, але при цьому забезпечує високу швидкість роботи та приватність завдяки виконанню обчислень на стороні клієнта.

2.5 Програмна реалізація методів та алгоритмів платформи

Реалізація запропонованої архітектури вимагала вирішення низки інженерних завдань для збалансування безпеки децентралізованих обчислень, приватності користувачів і зручності інтерфейсу. З метою оптимізації витрат газу використано позаланцюгове зберігання даних. Смарт-контракт

Marketplace.sol зберігає та обробляє лише фінансові параметри й унікальні ідентифікатори, а дані контенту (зображення та описи) винесено до децентралізованої мережі IPFS.

Розглянемо детальніше на прикладі структури Item у смарт-контракті, як реалізовано цей підхід. Ключовим елементом є поле itemCID, яке містить хеш-посилання на зовнішній файл метаданих. Це гарантує незмінність опису товару, оскільки будь-яка зміна файлу в IPFS призведе до зміни його CID, що зробить посилання у блокчейні невалідним.

```
struct Item {  
    uint256 id;  
    bytes32 publicId;  
    address seller;  
    uint256 price;  
    string itemCID;  
    uint256 stock;  
    uint256 sold;  
    bool active;  
}
```

Створення товару ініціюється на клієнтській стороні шляхом формування JSON-об'єкта, що відповідає специфікації ERC-721 Metadata [29]. Проаналізуємо структуру реального файлу метаданих, згенерованого платформою для товару «Футболка з принтом #4». Цей об'єкт містить детальні характеристики та посилання на зображення, яке також попередньо завантажується в IPFS:

```
{  
  "name": "Футболка з принтом #4",  
  "description": "100% бавовна, стильний принт",  
  "image": "ipfs://QmQF1eogYveNTvf7YZKw55akqAwAwA5a9Wo8eTM9Gx1Xh1",  
  "category": 1,  
  "properties": {  
    "createdAt": "2025-12-14T22:23:01.162Z"  
  }  
}
```

Безпосереднє завантаження сформованих метаданих у розподілену мережу реалізовано через API сервісу Pinata, що забезпечує стабільність

доступу до даних. Розглянемо на прикладі асинхронної функції `uploadMetadataToIPFS`, яка виконує POST-запит до ендпоінту `pinJSONToIPFS`. Важливим технічним моментом є використання JWT-токена для авторизації запиту та отримання у відповідь `IpfsHash`, який згодом буде записаний у смарт-контракт як `itemCID`:

```
export async function uploadMetadataToIPFS(metadata: object): Promise<string> {
  if (!PINATA_JWT) throw new Error("Pinata JWT is missing in env");
  const res = await fetch("https://api.pinata.cloud/pinning/pinJSONToIPFS", {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      Authorization: `Bearer ${PINATA_JWT}`,
    },
    body: JSON.stringify({
      pinataContent: metadata,
      pinataMetadata: { name: `metadata_${Date.now()}.json` },
    }),
  });
  if (!res.ok) {
    const text = await res.text().catch(() => "");
    throw new Error(Failed to upload metadata to IPFS. ${text});
  }
  const data = await res.json();
  return data.IpfsHash;
}
```

Оскільки дані тепер розподілені між блокчейном та IPFS, на клієнті виникає необхідність їх зворотного об'єднання. Варто розглянути реалізацію механізму «гідрації» даних [30]. Для забезпечення високої продуктивності завантаження списків товарів використовується асинхронна черга `mapLimit`, яка дозволяє паралельно зчитувати метадані для групи товарів за їхніми CID:

```
const hydratedItems = await mapLimit(topCandidates, 6, async (it) => {
  const metadata = await fetchMetadata(it.itemCID, signal);
  // Об'єднання on-chain даних (ціна, ID) з off-chain (назва, фото)
  return { ...it, ...metadata } as TopItem;
});
```

Наступним критичним аспектом для платформи електронної комерції є конфіденційність даних доставки. Оскільки блокчейн є публічним реєстром,

запис адреси покупця у відкритому вигляді є неприпустимим. Для вирішення цієї проблеми інтегровано протокол децентралізованого контролю доступу Lit Protocol. Розглянемо на прикладі функції `handleConfirmPurchase`, як відбувається шифрування даних перед відправкою транзакції. Механізм Access Control Conditions налаштовується таким чином, щоб право на розшифрування мав виключно власник гаманця продавця, що гарантує GDPR-сумісність [31]:

```
async function handleConfirmPurchase(deliveryDataString: string) {
  const total = (item.price as bigint) * BigInt(quantity);
  const encryptedDeliveryData = await encryptWithLit(
    deliveryDataString,
    item.seller
  );
  const buyHash = await walletClient.writeContract({
    address: MARKETPLACE_ADDRESS,
    abi: MARKETPLACE_ABI,
    functionName: "buyItem",
    args: [item.id as bigint, BigInt(quantity), encryptedDeliveryData],
    account: walletClient.account!, // Переконайся, що акаунт підключено
    chain: walletClient.chain,
  });
}
```

Після підготовки та шифрування даних починається етап транзакції. Логіка обробки замовлення реалізована з урахуванням вимог атомарності для запобігання стану гонитви, коли кілька користувачів намагаються придбати останню одиницю товару. Проаналізуємо фрагмент коду методу `buyItem`, де зменшення складського залишку відбувається до виконання фінансових операцій. Якщо подальший переказ коштів не вдасться, вся транзакція буде скасована, включаючи зміну залишку товару:

```
function buyItem(...) external {
  Item storage item = items[itemId];
  require(item.stock >= quantity, "Not enough stock");
  item.stock -= quantity;
  // ... подальша логіка оплати
}
```

Для забезпечення безпеки платежів застосовано патерн «Pull Payment» та бібліотеку SafeERC20 [32]. Кошти покупця блокуються на адресі смарт-контракту, а не перераховуються продавцю напряму. Зокрема, реалізація методу buyItem використовує модифікатор nonReentrant для захисту від атак повторного входу, гарантуючи коректність фінансових станів:

```
function buyItem(uint256 itemId, uint256 quantity, string calldata _deliveryData)
    external
    nonReentrant
    onlyUserRegistrySet
{
    uint256 totalPrice = item.price * quantity;
    stablecoin.safeTransferFrom(msg.sender, address(this), totalPrice);
    orders[orderId] = Order({
        escrowReleased: false,
        buyer: msg.sender,
        // ...
    });
}
```

Успішне завершення угоди відкриває можливість для оцінювання якості послуг. Для забезпечення довіри до платформи реалізовано захист від накрутки рейтингу. Розглянемо на прикладі перевірок у смарт-контракті Marketplace, які дозволяють залишати відгук лише після підтвердження доставки товару та блокують повторні виклики через встановлення прапорця reviewed:

```
function leaveReview(uint256 orderId, uint8 rating, string calldata reviewCID) external
{
    Order storage order = orders[orderId];
    require(msg.sender == order.buyer, "Not buyer");
    require(order.escrowReleased, "Order not completed");
    require(!order.reviewed, "Already reviewed");

    order.reviewed = true;
}
```

Однак не всі транзакції завершуються успішним підтвердженням доставки. У випадках, коли виникає конфлікт між сторонами (наприклад, товар не відповідає опису), замість завершення угоди активується механізм

арбітражу в контракті DisputeManager. Варто зазначити, що технічна реалізація збору доказів побудована за тим самим принципом, що й створення товару: файли доказів завантажуються в IPFS, а у смарт-контракт передається лише отриманий CID. Це забезпечує економію газу та незмінність аргументації сторін, не перевантажуючи блокчейн великими обсягами даних.

Ключовим елементом справедливості арбітражу є алгоритм вибору журі. Оскільки EVM є детермінованим середовищем, використано метод псевдовипадкової генерації на основі хешування. Проаналізуємо функцію `_pickRandomArbitrators`, яка використовує `keccak256` від параметрів блоку та адрес сторін для формування непередбачуваної вибірки кандидатів зі списку активних продавців:

```
function _pickRandomArbitrators(address buyer, address seller) internal view returns
(address[] memory) {
    uint256 totalSellers = userRegistry.getSellerCount();
    uint256 startIndex = uint256(keccak256(abi.encodePacked(
        block.timestamp, buyer, seller
    ))) % totalSellers;
    // Циклічний перебір кандидатів
}
```

Після накопичення в блокчейні масиву верифікованих статистичних даних (кількість продажів, відгуків, завершених угод), система переходить до їх аналітичної обробки. Основою для розрахунку репутації є алгоритм байєсівського згладжування, реалізований як універсальна утилітарна функція. Вона дозволяє зважувати локальну оцінку відносно глобального середнього, а також конвертувати значення зі смарт-контракту до звичайного числового формату:

```
export function bayesianScore(v: number, xBar: number, globalAvg: number, m: number):
number {
    if (v <= 0) return globalAvg;
    return (v / (v + m)) * xBar + (m / (v + m)) * globalAvg;
}
export function ratingFromTimes10(avgTimes10: bigint, count: number): number {
    if (count <= 0) return 0;
    return Number(avgTimes10) / 10;
}
```

Використовуючи ці інструменти, система спочатку обчислює рейтинг конкретного товару. Алгоритм зчитує глобальні показники по всій платформі та показники конкретного товару, після чого формує згладжений рейтинг із використанням параметру інерції:

```
const PRODUCT_M = 5;
const itemCount = Number(itemCountBn);
const itemGlobalCount = Number(globalItemTuple[0]);
const itemGlobalAvg = itemGlobalCount > 0 ? Number(globalItemTuple[1]) / 10 : 0;
const itemAvg = ratingFromTimes10(itemAvgTimes10Bn, itemCount);
const itemBayes = bayesianScore(itemCount, itemAvg, itemGlobalAvg, PRODUCT_M);
```

Наступним кроком є розрахунок інтегрального рейтингу продавця. Цей процес складніший, оскільки враховує не лише відгуки, а й історію диспутів. Варто звернути увагу на реалізацію, де спочатку розраховується показник надійності (від 0 до 1), який зменшується при наявності програних диспутів. Потім цей показник об'єднується з нормалізованим рейтингом відгуків у зважену суму і масштабується до шкали від 1 до 5:

```
const SELLER_M = 5;
const W_RATING = 0.7;
const W_DISPUTE = 0.3;
const sellerCount = Number(sellerCountBn);
const sellerGlobalCount = Number(globalSellerTuple[0]);
const sellerGlobalAvg = sellerGlobalCount > 0 ? Number(globalSellerTuple[1]) / 10 : 0;
const sellerAvg = ratingFromTimes10(sellerAvgTimes10Bn, sellerCount);
const sellerBayes = bayesianScore(sellerCount, sellerAvg, sellerGlobalAvg, SELLER_M);
const S = Number(completedOrdersBn);
const D = Number(disputeCountBn);
const Df = Number(disputeBuyerWinsBn);
const Dw = Number(disputeSellerWinsBnMaybe);
const totalInteractions = S + D;
const disputeScore = totalInteractions > 0 ? Math.max(0, 1 - Df / totalInteractions) : 1.0;
const normalizedBayes = sellerBayes / 5.0;
const T = W_RATING * normalizedBayes + W_DISPUTE * disputeScore;
const finalSellerRating = 1 + 4 * T;
```

Маючи розраховані рейтинги, система формує список «Топ товарів». Для коректного порівняння різнорідних метрик застосовано функції нормалізації. Нижче наведено код, де фінальний бал обчислюється як сума

чотирьох компонентів. Для кількості продажів використовується логарифмічне згладжування, а новизна визначається як відношення ідентифікатора конкретного товару до загальної кількості товарів у системі:

```
function computeFinalScoreFromRaw(params: {
  bayesRating: number; // 0..5
  sold: number;
  maxSold: number;
  itemId: number;
  itemCount: number;
  sellerFinal: number; // 1..5
}) {
  const ratingNorm = clamp01(params.bayesRating / 5);
  const popularityNorm = popularityNormFromSold(params.sold, params.maxSold);
  const recencyNorm = params.itemCount > 0 ? clamp01(params.itemId / params.itemCount)
: 0;
  const sellerNorm = clamp01((params.sellerFinal - 1) / 4);

  return Number(
    (
      W_RATING * ratingNorm +
      W_POPULAR * popularityNorm +
      W_RECENCY * recencyNorm +
      W_SELLER * sellerNorm
    ).toFixed(4)
  );
}
```

Також для покращення навігації було реалізовано алгоритм пошуку схожих товарів. Насамперед він підбирає товари з тієї ж категорії, додатково піднімає в результатах товари того самого продавця, а також враховує близькість рейтингу та рівень популярності, щоб пропонувати найбільш доречні альтернативи.

```
function computeSimilarScore(...) {
  const sameCategory = Number(item.category) === Number(current.category) ? 1 : 0;
  const sameSeller = item.seller?.toLowerCase() === current.seller?.toLowerCase() ?
0.3 : 0;
  const bayesNorm = bayes > 0 ? bayes / 5 : 0;
  return 1 * sameCategory + 0.7 * bayesNorm + 0.3 * popNorm + sameSeller;
}
```

Останнім етапом роботи аналітичного модуля є формування персональних рекомендацій. Для цього використовується гібридний механізм збору даних: історія переглядів фіксується виключно в локальному сховищі браузера для забезпечення приватності, тоді як історія покупок отримується безпосередньо з публічного реєстру блокчейну. Критичним етапом алгоритму є первинна перевірка активності: якщо у користувача відсутні дані як про перегляди, так і про покупки, процес генерації персональних рекомендацій не виконується. За наявності історії система використовує перелік товарів, попередньо відсортований за «Топом» на основі об'єктивної оцінки якості. Далі для кожного кандидата обчислюється персональний коефіцієнт схожості, який додається до вже наявного «топового» балу у рівних пропорціях. Отриманий фінальний результат використовується для побудови персональних рекомендацій конкретного користувача:

```
function computeCombinedSimilarity(  
  product: { id: number; category?: number; seller: `0x${string}` },  
  browsingHistory: any[],  
  purchaseHistory: any[]  
) : number {  
  const viewSim = computeSimilarityFromHistory(product, browsingHistory);  
  const purchaseSim = computeSimilarityFromPurchaseHistory(  
    { id: product.id, seller: product.seller },  
    purchaseHistory  
  );  
  if (viewSim === 0 && purchaseSim === 0) return 0;  
  const combined = 0.7 * purchaseSim + 0.3 * viewSim;  
  return Math.min(1, Math.max(0, combined));  
}  
  
const similarity = computeCombinedSimilarity(  
  { id: Number(item.id), category: Number((item as any).category ?? 7), seller:  
    item.seller },  
  browsingHistory ?? [],  
  purchaseHistory ?? []  
);  
  
const recScore = 0.5 * topScore + 0.5 * similarity;
```

Висновки до другого розділу

У другому розділі було виконано комплексне проектування архітектури та функціональних моделей децентралізованої платформи електронної комерції, призначеної для забезпечення безпечного обміну товарами без залучення третіх осіб.

Обґрунтовано вибір трирівневої архітектури, що включає в себе рівень представлення, рівень бізнес-логіки на базі смарт-контрактів і гібридний рівень даних. Останній поєднує блокчейн для зберігання критичних станів та IPFS для децентралізованого зберігання контенту. Життєвий цикл товарів і замовлень формалізовано через діаграми станів, що дало змогу чітко визначити правила переходів і обмеження доступу для ролей.

Сформовано модель ескроу та дворівневого арбітражу, яка забезпечує завершуваність спорів у межах заданих часових вікон і виключає сценарій нескінченного блокування коштів. Запропоновано принцип добору арбітрів з активних і надійних учасників платформи та механізми, що підвищують стійкість до зловживань і зменшують залежність від адміністратора.

Також розроблено та програмно реалізовано систему репутації на основі байєсівського згладжування, що знижує вплив поодиноких оцінок і доповнюється показником надійності продавця з урахуванням результатів диспутів. На базі цих метрик реалізовано алгоритм глобального ранжування із нормалізацією показників та механізм пошуку схожих товарів для покращення навігації.

Окрім цього, реалізовано гібридний підхід до персональних рекомендацій, у якому до вже відсортованого списку товарів додається коефіцієнт схожості, сформований з локальної історії переглядів і ончейн-історії покупок; за відсутності цих даних персоналізація не виконується. Такий підхід забезпечує релевантність рекомендацій і зберігає приватність користувача, оскільки перегляди не передаються у блокчейн.

РОЗДІЛ 3 ОПИС ПРОГРАМНОГО ПРОДУКТУ

3.1 Структура програмного продукту та взаємодія модулів

Реалізація децентралізованої платформи електронної комерції побудована за принципами модульної архітектури з чітким розділенням відповідальності між компонентами ончейн-логіки. Такий підхід обрано з огляду на властивості середовища виконання EVM: код смарт-контрактів після розгортання є незмінним, виконання має бути детермінованим для всіх вузлів мережі, а кожен запис у стан супроводжується витратами газу [32].

У зв'язку з цим архітектура проектувалася так, щоб ізолювати критичні фінансові операції, зменшити кількість дорогих ончейн-записів і забезпечити формалізовані точки інтеграції між модулями. Загальна схема взаємодії смарт-контрактів, їх наслідування та залежності наведена на рисунку 3.1.

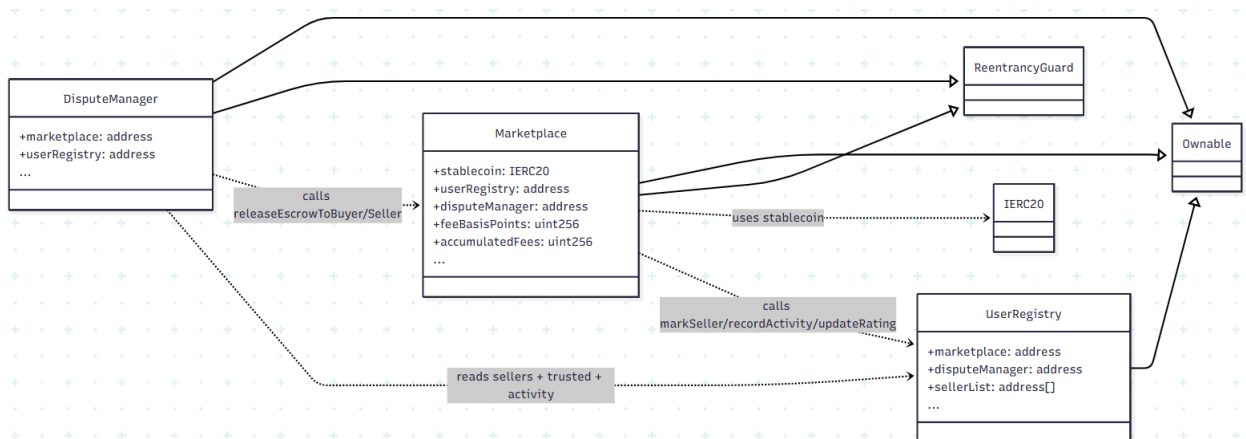


Рис. 3.1 – Схема взаємодії смарт-контрактів платформи

Ключовим елементом системи є смарт-контракт **Marketplace**, який концентрує торгову бізнес-логіку та керує життєвим циклом замовлення: створенням товарних позицій, формуванням замовлення, блокуванням оплати у режимі ескроу та завершенням угоди. Фінансова частина реалізована через використання токена стандарту ERC-20, що дозволяє зменшити вплив волатильності активів та забезпечити передбачуваність розрахунків [33].

Для локального тестування та налагодження платіжної логіки в середовищі розробки було створено допоміжний смарт-контракт **TestUSDC**, який реалізує тестовий ERC-20 та надає можливість керованого мінтингу

токенів; це дає змогу симулювати реальні сценарії купівлі, утримання коштів в ескроу, завершення угод і повернень без використання реальних активів. Інтеграція з токеном виконана через SafeERC20 з OpenZeppelin для коректної обробки операцій `transfer/transferFrom` і зниження ризиків, пов'язаних із нетиповою поведінкою окремих токенів.

Оскільки Marketplace утримує кошти в ескроу, всі операції, що змінюють фінансовий стан і можуть призводити до зовнішніх викликів, захищено механізмом `ReentrancyGuard` через модифікатор `nonReentrant`, а адміністративні налаштування обмежено через `Ownable` [11].

Для контролю комісій застосовано накопичувальний підхід, коли сума комісій акумулюється у внутрішній змінній та може бути виведена власником контракту окремою операцією, що зменшує ризики помилок обліку та спрощує аудит. Внутрішні структури даних організовано так, щоб мінімізувати витрати газу: основні сутності товарів і замовлень зберігаються у `mapping` структурах, які забезпечують доступ до записів за константний час $O(1)$ та не потребують лінійного перебору, притаманного масивам.

Контракт `UserRegistry` виконує роль узгодженого джерела відомостей про учасників платформи, зберігаючи статус продавців, показники активності та репутаційні значення. З практичної точки зору це дозволяє уникнути дублювання і рознесення критичних даних по різних контрактах, а також централізувати правила оновлення профільної інформації.

Оскільки Solidity не підтримує числа з плаваючою комою, для рейтингів застосовано техніку масштабування, коли значення зберігаються у вигляді цілих чисел із фіксованою точністю, що забезпечує детермінованість і відсутність похибок округлення [14]. Додатково реєстр підтримує перелік довірених арбітрів та часові мітки останньої активності, що використовуються для відбору кандидатів у процесі арбітражу та виключення неактивних учасників.

Процес вирішення спорів реалізовано окремим контрактом `DisputeManager`, що дозволяє ізолювати арбітражну логіку від фінансового

контуру Marketplace та зменшити ризик спільних точок відмови. Диспут описано як формальну машину станів із фазами подання доказів, арбітражу та завершення, що забезпечує передбачувану поведінку і спрощує перевірку коректності переходів.

Відбір первинних арбітрів виконується псевдовипадковим способом на основі кеска256 від параметрів блоку та ідентифікатора замовлення, що дає детермінований для мережі, але складний для прогнозування на прикладному рівні результат [34]. Після прийняття рішення DisputeManager не обмежується зміною внутрішнього статусу, а ініціює керований виклик до Marketplace для фактичного розблокування коштів на користь покупця або продавця. Така взаємодія вимагає жорсткого контролю доступу, тому функції управління ескроу в Marketplace обмежені модифікатором, що дозволяє їх виклик виключно з адреси розгорнутого контракту DisputeManager, а оновлення активності в UserRegistry дозволено лише Marketplace або DisputeManager.

З метою оптимізації витрат газу та уникнення зберігання об'ємних текстових даних у стані контрактів застосовано гібридний підхід: критичний для узгодженості стан і фінансові параметри фіксуються ончейн, тоді як контентні дані винесено в IPFS. У смарт-контрактах зберігаються лише CID-посилання на відповідні IPFS-об'єкти, що зменшує розмір ончейн-записів і знижує вартість виконання операцій. Розмежування ончейн-структур та офчейн-об'єктів IPFS, а також зв'язки між ними через CID наведені на рисунку 3.2 і визначають механізм формування карток товарів та замовлень у клієнтському інтерфейсі шляхом читання CID із контракту та завантаження відповідного JSON-об'єкта з IPFS.

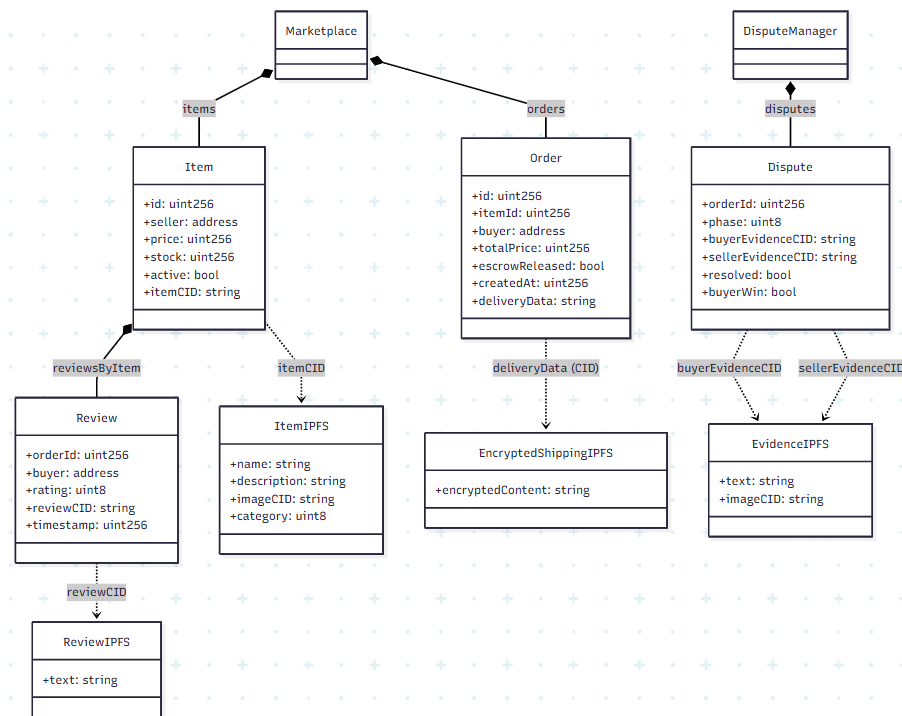


Рис. 3.2 – Гібридне зберігання даних

Ефективна взаємодія з позаблокчейн інфраструктурою та відтворення історії операцій забезпечуються подієвою моделлю, схема потоків якої наведена на рисунку 3.3. Замість збереження повної історії змін у стані контрактів кожна значуща зміна бізнес-процесу супроводжується генерацією типізованих подій, які записуються в логи блокчейну. Це дозволяє індексаторам зчитувати події, агрегувати їх у позаблокчейн-сховищі та надавати клієнтському застосунку узгоджену історію дій користувачів без надмірного навантаження на ончейн-сховище.

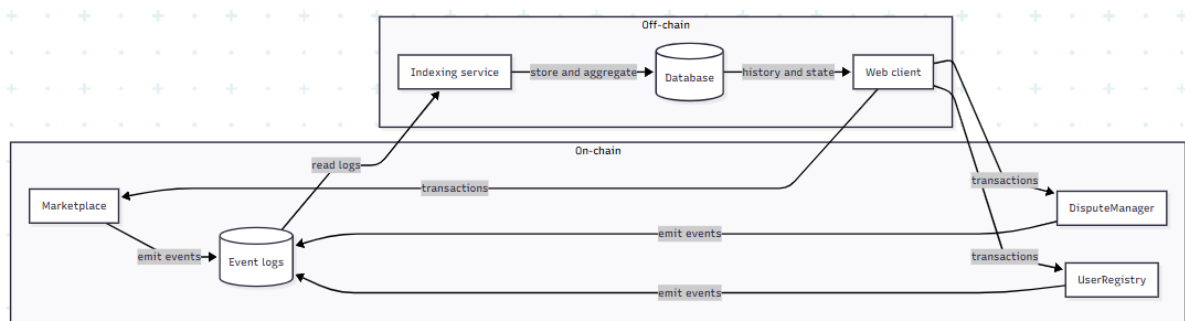


Рис. 3.3 – Подієва модель та потоки даних для відтворення історії операцій

У сукупності модульне розділення відповідальності, стандартизовані механізми безпеки OpenZeppelin, контрольований обмін між контрактами та гібридне зберігання даних (EVM + IPFS) забезпечують функціональність

платформи, стійкість до типових загроз і економічну ефективність реалізації з точки зору витрат газу.

3.2 Функціональні сценарії роботи веб-застосунку

Розроблена децентралізована платформа електронної комерції реалізує повний цикл роботи маркетплейсу з використанням криптогаманця та смарт-контрактів: від підключення користувача, перегляду каталогу та купівлі товарів до керування замовленнями, формування відгуків, а також вирішення спорів через механізм арбітражу. Клієнтська частина реалізована у вигляді веб-застосунку з темною темою інтерфейсу, який забезпечує зручну взаємодію з функціоналом, приховуючи складність блокчейн-операцій за простими екранними елементами.

Робота з системою починається з головної сторінки (рис. 3.3). У верхній панелі розташовано основні пункти навігації «Головна», «Кабінет», «Товари», а праворуч – кнопку підключення гаманця. Також передбачено сервісну кнопку отримання тестових токенів для роботи у тестовому середовищі. Центральна частина сторінки містить блок швидких дій та інформаційний банер, який підкреслює ключову особливість платформи – гарантоване повернення коштів завдяки ескроу механізму смарт-контрактів.

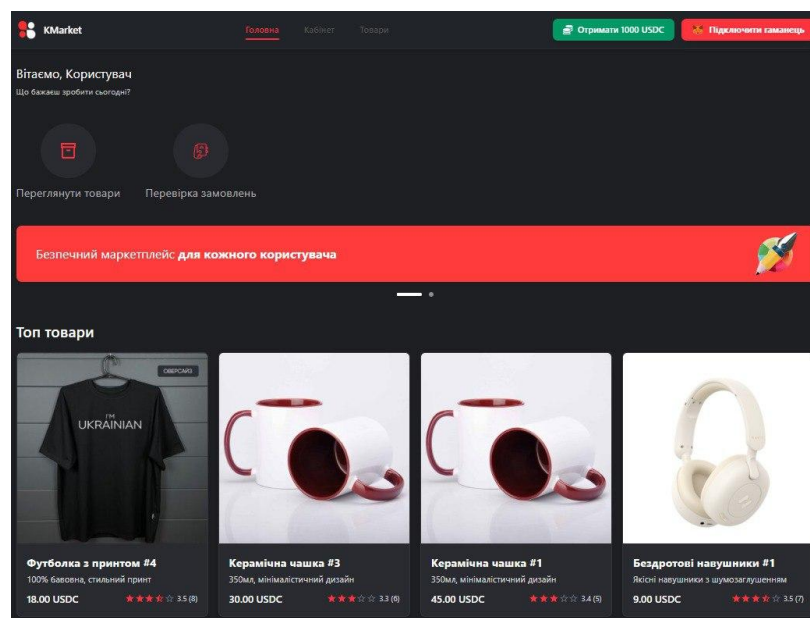


Рис. 3.3 – Головна сторінка веб-застосунку

Нижче на головній сторінці відображено добірку «Топ товари», що дозволяє користувачу швидко перейти до популярних позицій. Окремо реалізовано секцію «Рекомендовано для тебе» (рис. 3.4), яка є персоналізованою і активується лише за умови, що користувач виконав вхід через підключення криптогаманця та має сформовану історію взаємодії з системою, зокрема історію покупок або перегляду товарів. Якщо гаманець не під'єднано або історії взаємодії недостатньо, рекомендаційний блок не активується і не відображається як функціональна секція, щоб уникнути нерелевантних пропозицій та зберегти коректність персоналізації.

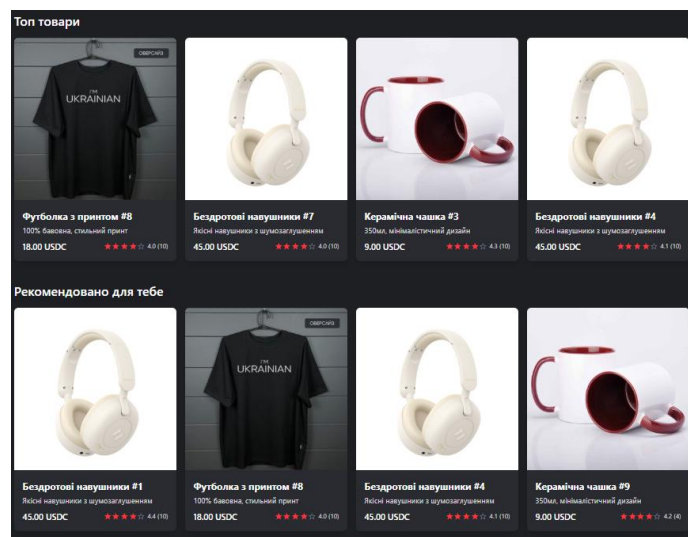


Рис. 3.4 – Блок персоналізованих рекомендацій та найкращих товарів

Для перегляду повного асортименту користувач переходить до розділу «Товари», який відкриває екран перегляду каталогу (рис. 3.5). На цьому екрані реалізовано пошук за назвою через текстове поле та фільтрацію за категоріями, а результати подаються у вигляді сітки карток. Кожна картка містить зображення, назву, короткий опис, ціну в USDC та агреговану оцінку у вигляді зірок, що дозволяє швидко порівнювати позиції й обирати потрібний товар.

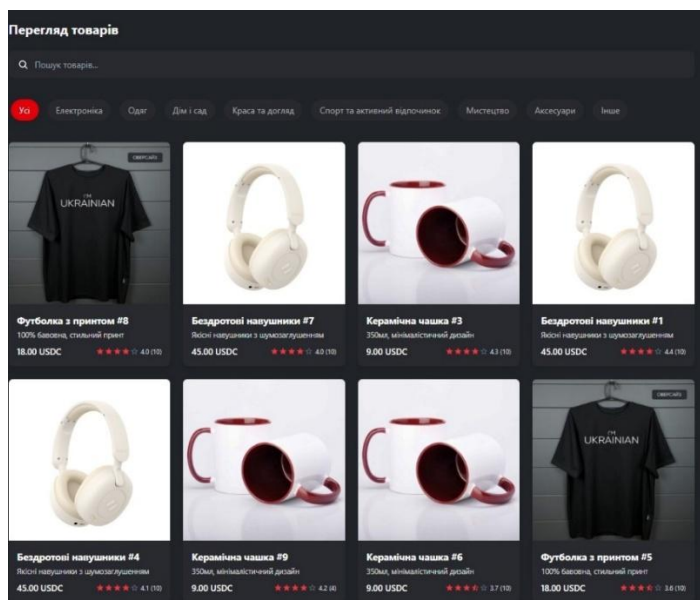


Рис. 3.5 – Каталог товарів

Після вибору конкретної позиції відкривається сторінка товару (рис. 3.6), що складається з візуального блоку зображення та інформаційної панелі з назвою і ціною, а також вкладок «Деталі» та «Відгуки». На вкладці «Деталі» користувачу відображаються ключові атрибути товару, зокрема категорія, опис, адреса продавця, кількість продажів та залишок на складі. Для здійснення покупки доступний вибір кількості з автоматичним розрахунком підсумкової суми та кнопка «Купити зараз». На сторінці також відображено блок «Схожі товари», який пропонує альтернативні позиції й спрощує навігацію по каталогу.

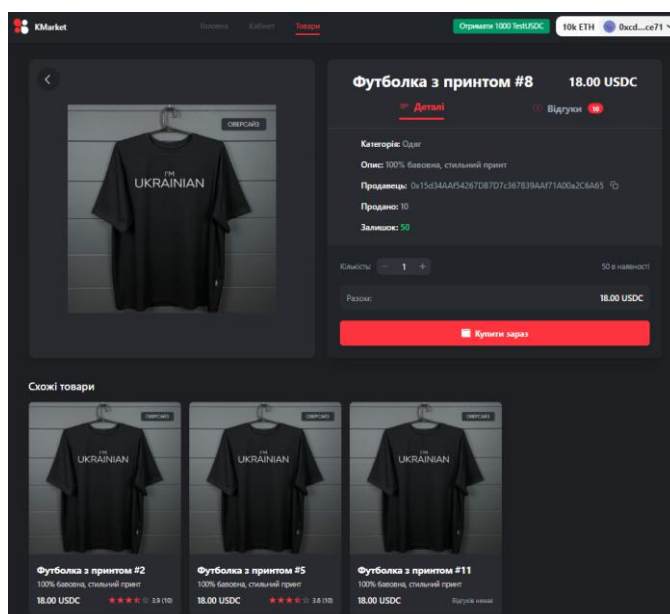


Рис. 3.6 – Сторінка товару

Операція купівлі пов'язана з блокчейн-транзакціями. Зокрема, перед оплатою токеном USDC система може вимагати підтвердження дозволу на списання tokenів, що відображається у модальному вікні гаранця MetaMask у вигляді запиту на надання права витрачання tokenів для відповідного смарт-контракту (рис 3.7). Під час очікування транзакції інтерфейс відображає стан обробки, завдяки чому користувач розуміє, що дія виконується, і не повторює натискання.

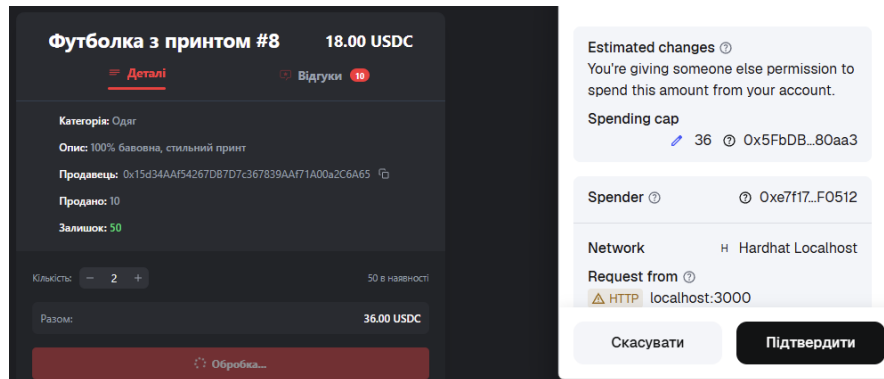


Рис. 3.7 – Підтвердження транзакції в MetaMask

Для оцінки якості товарів і продавців реалізовано механізм відгуків. На вкладці «Відгуки» (рис. 3.8) окремо виводяться агреговані рейтинги продавця та товару, а також кількість відгуків. Нижче відображається список відгуків у вигляді карток, що містять скорочену адресу автора, дату, оцінку у вигляді зірок та текстовий коментар. Для довгих списків застосовано механізм поступового виведення «Показати більше», який зменшує навантаження на інтерфейс.

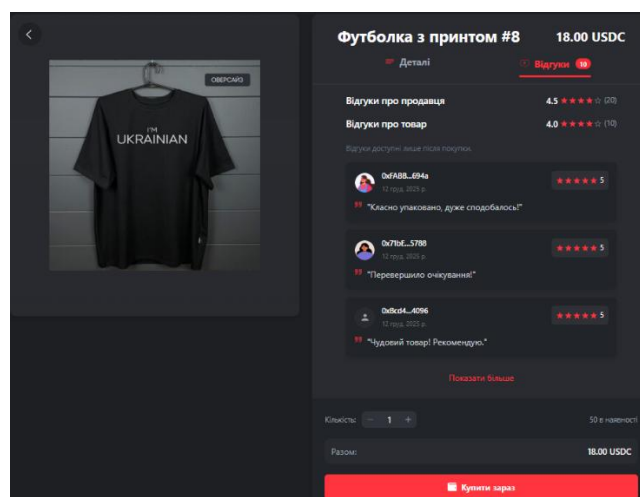


Рис. 3.8 – Вкладка з відгуками

Форма створення відгуку доступна лише користувачам, які здійснили успішну покупку даного товару. Також для перешкодження маніпуляцій з відгуками, 1 користувач може залишити лише 1 відгук про певний товар. При виконанні цих умов на вкладці «Відгуки» відображається блок «Залишити відгук» (рис. 3.9), що містить вибір оцінки та поле введення коментаря.

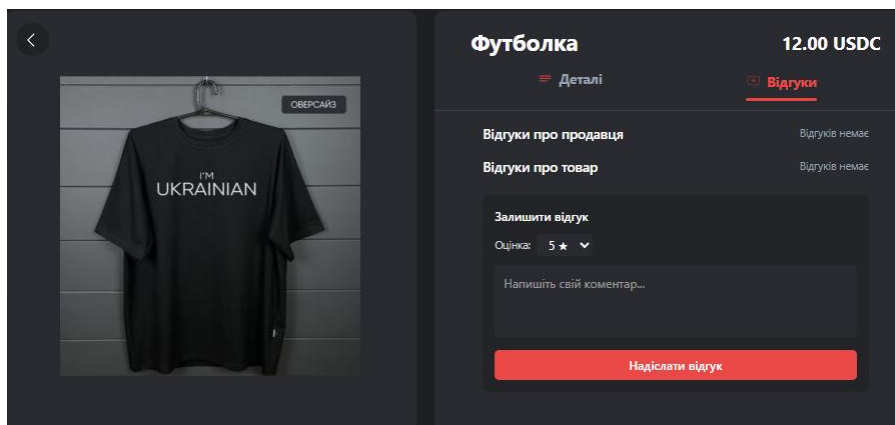


Рис. 3.9 – Додавання відгуку

Керування замовленнями реалізовано в розділі «Кабінет», який містить вкладки «Історія замовлень», «Мої товари», «Диспути», «Створити товар». Вкладка «Історія замовлень» відображає замовлення у вигляді карток (рис. 3.10), де для кожного замовлення виводяться: зображення товару, назва, адреса продавця, ціна за одиницю, загальна сума, ідентифікатор замовлення та поточний статус. Для активного замовлення доступні наступні дії: підтвердити доставку, відкрити спір та переглянути товар.

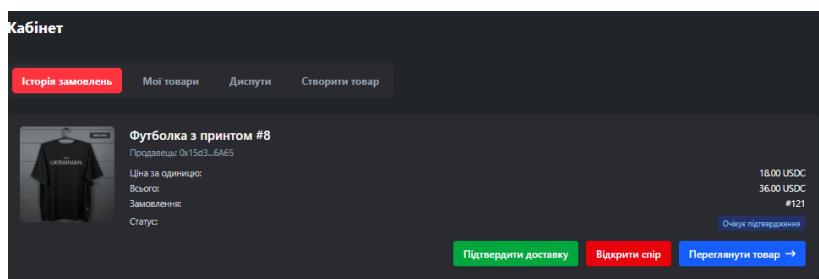


Рис. 3.10 – Історія замовлень

Після підтвердження отримання замовлення статус змінюється на завершений, а набір доступних дій скорочується (рис. 3.11). Це відображає завершення життєвого циклу угоди та фіксує факт виконання зобов'язань сторонами.

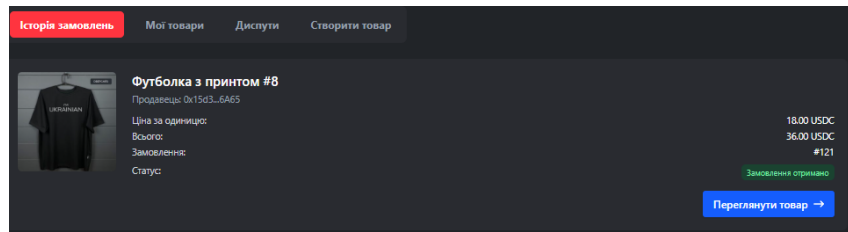


Рис. 3.11 – Завершене замовлення

У разі конфліктної ситуації користувач може відкрити спір. Після цього у картці замовлення відображається статус активного диспуту та з'являється кнопка переходу до деталей спору (рис. 3.12). Це дозволяє швидко перейти до сторінки диспуту та виконати необхідні дії згідно з поточним етапом.

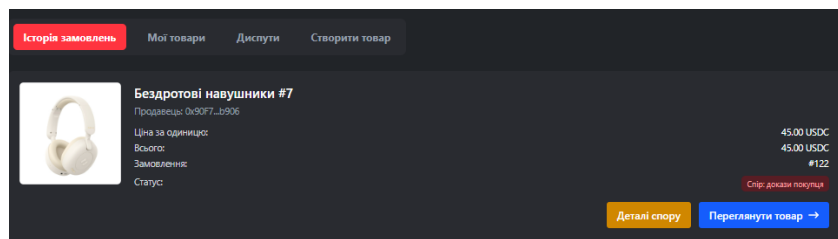


Рис. 3.12 – Активний диспут

Сторінка диспуту має структурований інтерфейс, що відображає номер спору, поточний статус і послідовні етапи: докази покупця – докази продавця – арбітраж – фінал (рис. 3.13). Для контролю часу надання матеріалів може відображатися таймер дедлайну поточного етапу. У правій частині розташовано панель «Ваші дії», де сторона спору вводить опис ситуації та може прикріпити файл-доказ, після чого надсилає матеріали кнопкою «Відправити докази». Нижче виведено блок «Історія доказів», який окремо акумулює матеріали покупця та продавця.

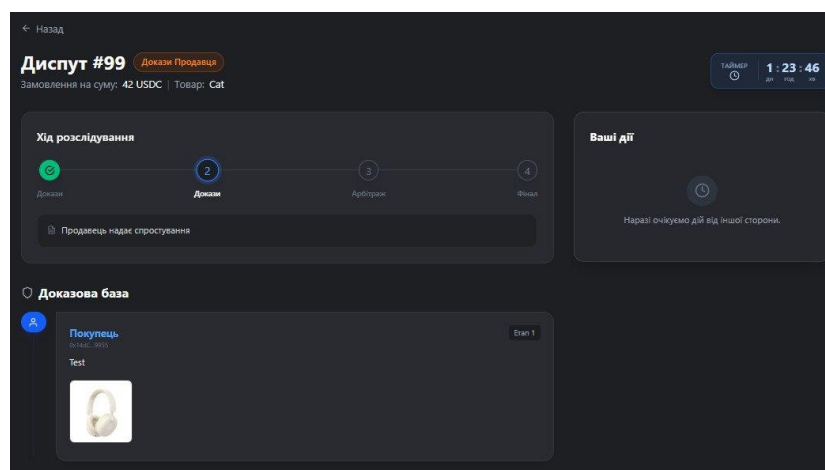


Рис. 3.13 – Сторінка диспуту

Після надання доказів обома сторонами система переходить до арбітражу. Коли голосування завершено, сторінка відображає фінальне рішення та закриття справи (рис. 3.14). Для збереження приватності даних доступ до сторінки певного диспуту мають лише його учасники.

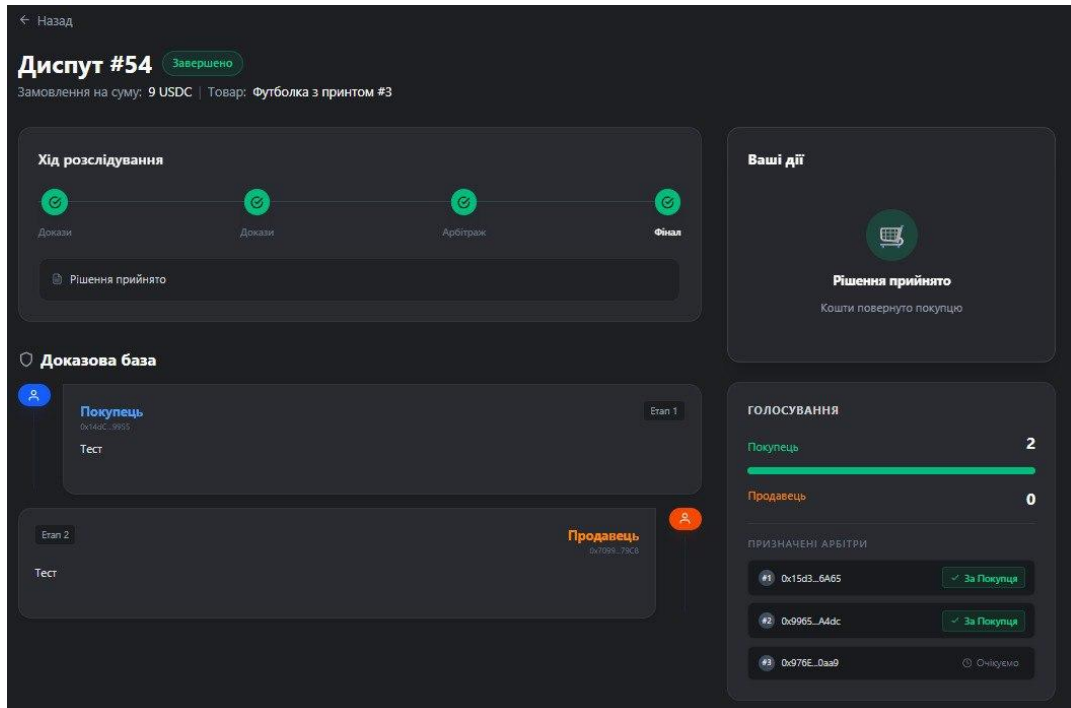


Рис. 3.14 – Рішення арбітражу

Окремо в системі передбачено адміністративний функціонал у вигляді адмін-панелі (рис. 3.15). В ній відображаються службові адреси компонентів (власник, менеджер диспутів, реєстр користувачів), параметр комісії, сума коштів, доступних до виводу, та відповідні дії для оновлення налаштувань і виведення коштів. Важливим блоком є список довірених арбітрів, де адміністратор може додавати або видаляти адреси, що беруть участь у вирішенні спорів, які не змогли вирішитися на етапі залучення арбітрів з випадково обраних системою. Також у нижній частині відображається список активних диспутів із ключовою інформацією про сторони та стан справи, що спрощує контроль поточних конфліктів.

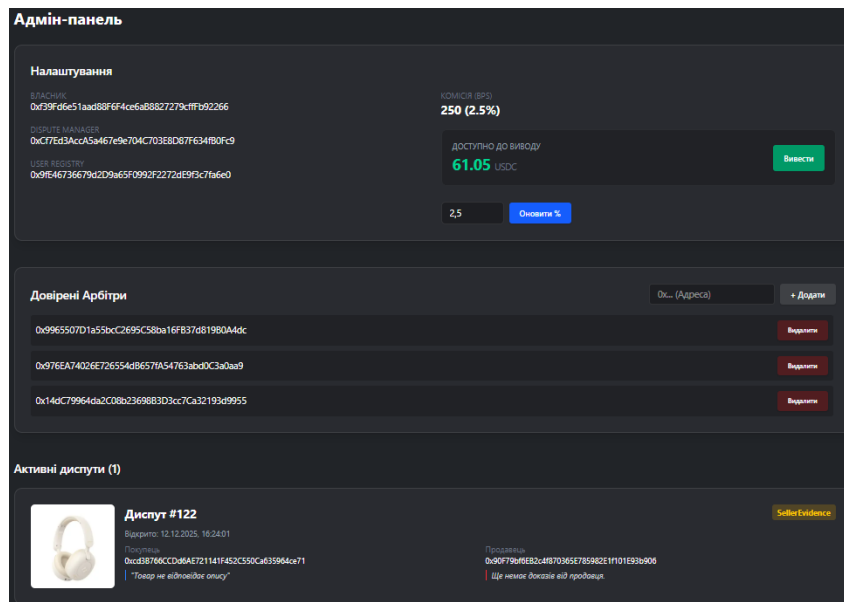


Рис. 3.15 – Адмін-панель

Для користувачів, що виступають у ролі продавця, реалізовано вкладку «Мої товари» (рис. 3.16). Вона відображає перелік створених товарів у вигляді карток із зображенням, назвою, описом, ціною в USDC, інформацією про відгуки та поточним складським запасом. На картці передбачено швидке керування запасом: продавець вводить нове значення кількості на складі та застосовує його кнопкою «Оновити». Це дозволяє швидко змінювати наявність без переходу до повного редагування.

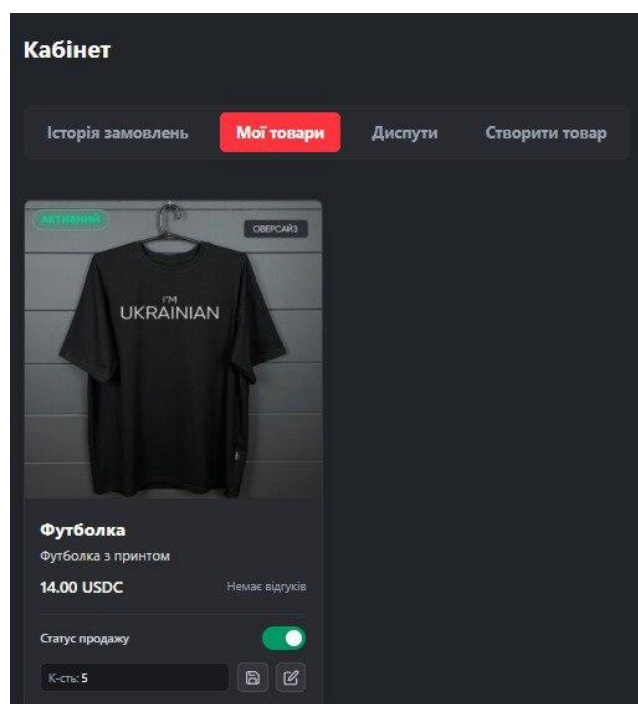


Рис. 3.16 – Мої товари

Створення нової позиції виконується у вкладці «Створити товар» (рис. 3.17). Екран містить форму з основними полями: зображення, назва, опис, категорія, ціна в USDC та кількість на складі. Вибір категорії реалізований через випадний список, а числові значення (ціна, кількість) вводяться у відповідні поля. Після заповнення форми продавець натискає кнопку «Створити товар», і позиція стає доступною у каталозі.

Рис. 3.17 – Створення товару

Для внесення змін у товар передбачено екран «Редагувати товар» (рис. 3.18), який містить поля редагування параметрів та кнопку «Зберегти зміни». При цьому в системі застосовано правило цілісності даних: повністю редагувати картку товару можна лише тоді, коли товар ще не мав продажів. Це обмеження запобігає нечесним змінам характеристик після здійснення покупок і підтримує прозорість історії угод. Якщо ж продажі вже були, продавцю залишається доступним лише контроль залишків.

Редагувати товар

Зображення товару

Назва

Футболка

Опис

Футбол з принтом

Категорія

Одяг

Ціна (USDC)

12

Кількість на складі

32

Зберегти зміни

Рис. 3.18 – Редагування товару

Таким чином, інтерфейс платформи охоплює повний життєвий цикл взаємодії сторін у маркетплейсі: перегляд товарів і покупка з підтвердженням транзакцій через гаманець, формування системи відгуків, управління замовленнями через кабінет користувача, а також механізм диспутів із поданням доказів і арбітражним вирішенням.

3.3 Результати тестування програмного продукту

Тестування розробленої системи проводилося з метою всебічно перевірити коректність реалізації бізнес-логіки, переконатися в надійній взаємодії між смарт-контрактами та оцінити стабільність ключових сценаріїв у середовищі, максимально наближеному до реального [35]. З огляду на те, що платформа виконує критичні операції з фінансовими активами, утримує кошти в ескроу і підтримує механізми врегулювання спорів, основний фокус було спрямовано на автоматизоване тестування контрактної частини. Перевірка якості поєднувала функціональне покриття смарт-контрактів тестами та моделювання навантаження за допомогою окремого скрипта наповнення

даними, що дозволяло відтворювати поведінку користувачів у динаміці й додатково підтверджувати коректність роботи системи на великій кількості транзакцій.

Контрактна частина системи складається з кількох взаємопов'язаних модулів, зокрема платіжного токена TestUSDC, ядра маркетплейсу Marketplace, реєстру користувачів UserRegistry та модуля керування диспутами DisputeManager. На стартовому етапі тестування перевірялася коректність розгортання та ініціалізації контрактів, а також правильність встановлення адрес взаємодії між ними, оскільки від цього залежить можливість виконання міжконтрактних викликів і коректність рольової моделі доступу.

Зокрема, було підтверджено, що Marketplace використовує UserRegistry для маркування продавців і збереження активності, а DisputeManager коректно взаємодіє з Marketplace під час фінального вирішення спору та розподілу коштів.

У межах функціонального тестування перевірявся повний життєвий цикл товару: від створення позиції та перевірки коректності вхідних параметрів ціни й запасу до логіки оновлення складських залишків. Окремий акцент робився на правилі цілісності даних, за яким після першого продажу картка товару вже не могла редагуватися «повністю».

Результати тестів показали, що після появи історії замовлень система дійсно блокує зміну критичних атрибутів і дозволяє оновлювати лише наявність товару. Така поведінка розглядалася як важливий механізм захисту покупців, оскільки унеможливлювала підміну умов угоди заднім числом.

Під час тестування окрему увагу приділялося фінансовим операціям і тому, як саме працює ескроу механізм у реальних сценаріях. Купівля відтворювалася так, як це зазвичай відбувається в токенних системах: спочатку перевірялося коректне надання дозволу на списання, а вже потім запускалася функція покупки, і в цей момент контролювалося, що сума

справді блокується на контракті, а товарний залишок зменшується без розривів у логіці.

Після завершення угоди додатково відстежувалося, чи правильно платформа нараховує власну комісію, чи зберігається вона у внутрішньому стані контракту та чи основна частина коштів безпомилково переказується продавцю. Також перевірялися адміністративні операції з виведення накопичених комісій, щоб підтвердити використання безпечних підходів до виплат, мінімізувати ризики повторного входу та зберегти точність фінансових розрахунків.

Система рейтингів і відгуків також проходила окрему детальну перевірку. Під час тестування було підтверджено, що залишити відгук може лише покупець конкретного товару, лише один раз і тільки після успішного завершення угоди.

Для перевірки роботи системи в динаміці використовувався спеціалізований скрипт наповнення даними, який моделював поведінку реальних користувачів. У тестовому середовищі створювались різні ролі, генерувалась історія замовлень, формувались відгуки зі змішаним розподілом оцінок та ініціювались процеси спорів. Це дозволило перевірити повний цикл роботи модуля арбітражу: від відкриття диспуту та подання доказів сторонами до голосування арбітрів і фінального розподілу коштів залежно від прийнятого рішення.

Завершальним етапом стала оцінка ефективності використання газу для ключових транзакцій [36]. Вимірювання виконувались під час прогону сценаріїв наповнення та взаємодії з маркетплейсом, а результати подано на рисунку 3.19 у вигляді газ-звіту.

Транзакція	N	AVG	MIN	MAX
buyItem	96	305 967	303 473	354 773
leaveReview	67	300 675	284 341	472 441
submitSellerEvidence	20	296 426	289 663	340 963
createItem	12	286 423	250 664	350 670
vote	40	186 494	138 122	268 172
openDispute	20	174 521	174 521	174 521
confirmDelivery	67	145 282	138 901	207 301
submitBuyerEvidence	20	104 476	104 476	104 476
addTrustedArbitrator	3	52 845	52 845	52 845
mintUSDC	12	52 472	51 036	68 148
approveUSDC	96	45 987	45 981	45 993
setItemActive	2	40 444	28 416	52 471
updateItem	1	38 478	38 478	38 478
updateStock	1	35 065	35 065	35 065
=====				

Рис. 3.19 – Газ-звіт транзакцій

За результатами вимірювань найбільші витрати газу припадали на ті транзакції, у яких контракт суттєво змінює свій стан і записує багато даних. Саме тому найдорожчими виявилися покупка товару та публікація відгуку. Операції оновлення й завершення угоди вимагали помірної кількості газу, оскільки в них одночасно виконуються токенні перекази та оновлюються дані замовлення чи товару.

Натомість підтвердження дозволу на списання токенів і оновлення залишків коштували дешевше, адже змін у стані контракту там значно менше. Загалом отримані результати показали, що архітектура дає змогу поєднати безпеку фінансових дій, зрозумілу логіку маркетплейсу та прийнятну вартість виконання основних сценаріїв у блокчейні.

Висновки до третього розділу

У третьому розділі подано опис програмного продукту децентралізованої платформи електронної комерції та ключових принципів його реалізації. Розглянуто модульну архітектуру смарт-контрактів із чітким розподілом відповідальності між торговим модулем, реєстром користувачів і репутації та модулем врегулювання спорів.

Обґрунтовано підхід до зберігання даних, за якого критичний стан і фінансові параметри фіксуються в смарт-контрактах, а об’ємний контент

зберігається в розподіленому сховищі з доступом за ідентифікатором вмісту. Показано, що відтворення історії дій забезпечується подієвою моделлю, яка дозволяє не зберігати повну історію змін безпосередньо у стані контрактів.

Також описано роботу клієнтської частини з погляду користувача. Розглянуто підключення гаманця, перегляд каталогу та сторінки товару. Описано процес купівлі з підтвердженням транзакцій і створення відгуків. Показано, як користувач керує замовленнями, ініціює спір і проходить усі його етапи до фінального рішення. Окремо висвітлено можливості продавця щодо створення товарів і керування залишками. Також розглянуто адміністративні функції для налаштування платформи та контролю активних конфліктів.

Проведено тестування ключових сценаріїв роботи системи. Перевірено коректність розгортання та взаємодії модулів, повний життєвий цикл товарів і замовлень, фінансові операції, обмеження на редагування товарів після продажів, правила залишення відгуків і роботу механізму диспутів із поданням доказів та голосуванням.

Додатково виконано навантажувальне моделювання та оцінено витрати ресурсів для основних транзакцій у мережі. Це дало змогу оцінити витрати комісій для ключових операцій, переконатися в їхній економічній доцільності та підтвердити практичну доцільність обраних архітектурних рішень.

ВИСНОВКИ

Під час виконання кваліфікаційної магістерської роботи було виконано проєктування та розробку децентралізованої платформи електронної комерції на основі блокчейн-технологій. У роботі опрацьовано проблему надмірної концентрації довіри в централізованих рішеннях та запропоновано підхід, за якого ключові правила угод, фіксація станів і критичні дії виконуються смарт-контрактами, а користувачі отримують прозорий механізм захисту інтересів у спірних ситуаціях. Досягнуто поставленої мети шляхом побудови моделі взаємодії учасників, формалізації життєвого циклу угоди та реалізації системи репутації, ранжування і рекомендацій на основі відкритих даних платформи.

У першому розділі проаналізовано предметну область електронної комерції. Обґрунтовано доцільність переходу від моделі довіри до оператора до моделі довіри до протоколу. Розглянуто типові обмеження централізованих рішень: непрозорість комісій, залежність репутації від платформи та закритий арбітраж. На основі цього визначено вимоги до децентралізованої системи. Вона має забезпечувати контрольоване депонування коштів у межах угоди, регламентоване врегулювання спорів, відтворювану репутацію учасників і прозору логіку ранжування товарів.

У другому розділі виконано проєктування архітектури платформи та ключових моделей. Запропоновано трирівневу структуру додатку, визначено ролі користувачів і стани угоди. Розроблено дворівневий механізм внутрішнього арбітражу з фазами подання доказів, арбітражу та завершенням справи у скінченний час. Побудовано та програмно реалізовану математичну модель рейтингування товарів і продавців із використанням байєсівського згладжування, а також репутаційний компонент, що враховує результати спорів. Реалізовано модель ранжування товарів на основі верифікованих метрик платформи та гібридний підхід до персоналізованих рекомендацій, де частина обчислень виконується на стороні клієнта для збереження приватності.

У третьому розділі описано реалізацію програмного продукту: модульну структуру смарт-контрактів із розмежуванням торгової логіки, реєстру користувачів і модуля спорів; гібридне зберігання даних із винесенням об'ємного контенту до розподіленого сховища та збереженням посилань у контрактах; подієву модель для відтворення історії операцій без надмірного накопичення стану. Наведено опис роботи веб-застосунку з позиції користувача: підключення гаманця, робота з каталогом, купівля, відгуки, керування замовленнями, ініціювання спору та перегляд рішення, а також функції продавця й адміністративні налаштування.

Окремо виконано тестування ключових сценаріїв роботи платформи: перевірено коректність розгортання та взаємодії модулів, життєвий цикл товарів і замовлень, фінансові операції, обмеження на редагування товарів після продажів, правила публікації відгуків, а також повний цикл спору з поданням доказів і прийняттям рішення. Додатково проведено моделювання навантаження через наповнення даними та оцінено ресурсні витрати основних транзакцій, що дозволило виявити операції з найбільшим впливом на вартість виконання й підтвердити обґрунтованість обраних архітектурних рішень.

У результаті отримано працездатне програмне рішення, яке формує більш безпечне та прозоре середовище для онлайн-торгівлі: ключові події угод фіксуються в незмінному реєстрі, механізм врегулювання спорів є регламентованим і відтворюваним, а оцінювання надійності учасників і якості товарів базується на відкритих показниках системи. Це підвищує довіру між сторонами та зменшує залежність від централізованого посередника під час здійснення операцій купівлі-продажу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Kholodnitskyi V., Vakaliuk T. Using blockchain technology to ensure security and transparency of financial transactions. Тези VII Всеукраїнської науково-технічної конференції «Комп'ютерні технології: інновації, проблеми, рішення», м. Житомир, 02-03 грудня 2024 р. – Житомир: Житомирська політехніка, 2024. С. 19-20.
2. Kholodnitskyi V., Vakaliuk T. Using blockchain technology to enhance data privacy in e-commerce. Тези XV Міжнародної науково-технічної конференції «Інформаційно-комп'ютерні технології», м. Житомир, 28-29 березня 2025 р. – Житомир: Житомирська політехніка, 2025. С. 37-38.
3. Arps, J.E., Christin, N. (2020). Open Market or Ghost Town? The Curious Case of OpenBazaar. In: Bonneau, J., Heninger, N. (eds) Financial Cryptography and Data Security. FC 2020. Lecture Notes in Computer Science(), vol 12059. Springer, Cham. https://doi.org/10.1007/978-3-030-51280-4_30.
4. H. Huang, J. Lin, B. Zheng, Z. Zheng and J. Bian, "When Blockchain Meets Distributed File Systems: An Overview, Challenges, and Open Issues," in IEEE Access, vol. 8, pp. 50574-50586, 2020, doi: 10.1109/ACCESS.2020.2979881.
5. Himeur Y., Sayed A., Alsalemi A., Bensaali F., Amira A., Varlamis I., Eirinaki M., Sardianos C., Dimitrakopoulos G., Blockchain-based recommender systems: Applications, challenges and future opportunities, Computer Science Review, Volume 43, 2022, 100439, ISSN 1574-0137, <https://doi.org/10.1016/j.cosrev.2021.100439>.
6. A. Asgaonkar and B. Krishnamachari, "Solving the Buyer and Seller's Dilemma: A Dual-Deposit Escrow Smart Contract for Provably Cheat-Proof Delivery and Payment for a Digital Good without a Trusted Mediator," 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), Seoul, Korea (South), 2019, pp. 262-267, doi: 10.1109/BLOC.2019.8751482.
7. Christian Catalini and Joshua S. Gans. 2020. Some simple economics of the blockchain. Commun. ACM 63, 7 (July 2020), 80–90. <https://doi.org/10.1145/3359552>.

8. Rosenblat, Alex and Stark, Luke, Algorithmic Labor and Information Asymmetries: A Case Study of Uber's Drivers (July 30, 2016). International Journal Of Communication, 10, 27, <http://dx.doi.org/10.2139/ssrn.2686227>.
9. Walch, Angela, 'In Code(rs) We Trust: Software Developers as Fiduciaries in Public Blockchains', in Philipp Hacker, and others (eds), Regulating Blockchain: Techno-Social and Legal Challenges (Oxford, 2019; online edn, Oxford Academic, 22 Aug. 2019), <https://doi.org/10.1093/oso/9780198842187.003.0004>.
10. Milano, S., Taddeo, M. & Floridi, L. Recommender systems and their ethical challenges. AI & Soc 35, 957–967 (2020). <https://doi.org/10.1007/s00146-020-00950-y>.
11. OpenZeppelin Contracts. *OpenZeppelin Docs*. URL: <https://docs.openzeppelin.com/contracts> (дата звернення: 17.11.2025).
12. M. di Angelo and G. Salzer, "A Survey of Tools for Analyzing Ethereum Smart Contracts," 2019 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPCON), Newark, CA, USA, 2019, pp. 69-78, doi: 10.1109/DAPPCON.2019.00018.
13. Nizamuddin, N., Hasan, H.R., Salah, K. (2018). IPFS-Blockchain-Based Authenticity of Online Publications. In: Chen, S., Wang, H., Zhang, L.J. (eds) Blockchain – ICBC 2018. ICBC 2018. Lecture Notes in Computer Science(), vol 10974. Springer, Cham. https://doi.org/10.1007/978-3-319-94478-4_14.
14. Multi-Chain API Documentation - RPC, REST, gRPC & Specialized APIs | Quicknode. *High-Performance Blockchain Infrastructure*. URL: <https://www.quicknode.com/docs/welcome> (дата звернення: 17.11.2025).
15. S. Porru, A. Pinna, M. Marchesi and R. Tonelli, "Blockchain-Oriented Software Engineering: Challenges and New Directions," 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), Buenos Aires, Argentina, 2017, pp. 169-171, doi: 10.1109/ICSE-C.2017.142.
16. Gastón Márquez, Mónica M. Villegas, and Hernán Astudillo. 2018. A pattern language for scalable microservices-based systems. In Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings (ECSA '18).

Association for Computing Machinery, New York, NY, USA, Article 24, 1–7.
<https://doi.org/10.1145/3241403.3241429>.

17. G. Ascia, V. Catania, J. Jose, S. Monteleone, M. Palesi and D. Patti, "Improving Inference Latency and Energy of Network-on-Chip based Convolutional Neural Networks through Weights Compression," 2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), New Orleans, LA, USA, 2020, pp. 54-63, doi: 10.1109/IPDPSW50202.2020.00017.

18. S. Wang, L. Ouyang, Y. Yuan, X. Ni, X. Han and F. -Y. Wang, "Blockchain-Enabled Smart Contracts: Architecture, Applications, and Future Trends," in IEEE Transactions on Systems, Man, and Cybernetics: Systems, vol. 49, no. 11, pp. 2266-2277, Nov. 2019, doi: 10.1109/TSMC.2019.2895123.

19. Delmolino, K., Arnett, M., Kosba, A., Miller, A., Shi, E. (2016). Step by Step Towards Creating a Safe Smart Contract: Lessons and Insights from a Cryptocurrency Lab. In: Clark, J., Meiklejohn, S., Ryan, P., Wallach, D., Brenner, M., Rohloff, K. (eds) Financial Cryptography and Data Security. FC 2016. Lecture Notes in Computer Science(), vol 9604. Springer, Berlin, Heidelberg.
https://doi.org/10.1007/978-3-662-53357-4_6.

20. J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll and E. W. Felten, "SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies," 2015 IEEE Symposium on Security and Privacy, San Jose, CA, USA, 2015, pp. 104-121, doi: 10.1109/SP.2015.14.

21. Dylag, M., & Smith, H. (2023). From cryptocurrencies to cryptocourts: blockchain and the financialization of dispute resolution platforms. *Information, Communication & Society*, 26(2), 372–387.
<https://doi.org/10.1080/1369118X.2021.1942958>.

22. Z. Yang, Z. Yang and D. Han, "Alternating Direction Method of Multipliers for Sparse and Low-Rank Decomposition Based on Nonconvex Nonsmooth Weighted Nuclear Norm," in IEEE Access, vol. 6, pp. 56945-56953, 2018, doi: 10.1109/ACCESS.2018.2872688.

23. Martiny, I., Miers, I., Wustrow, E. (2018). Proof of Censorship: Enabling Centralized Censorship-Resistant Content Providers. In: Meiklejohn, S., Sako, K. (eds) *Financial Cryptography and Data Security. FC 2018. Lecture Notes in Computer Science()*, vol 10957. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-58387-6_6.
24. Chang, Victor & Xiao, Lina & Nikiforova, Anastasija & Xu, Qianwen & Liu, Ben. (2021). The Study of PGP Web of Trust Based on Social Network Analysis. *International Journal of Business Information Systems*. <https://doi.org/10.1504/IJBIS.2023.134956>.
25. Ohlhaber, Puja and Weyl, Eric Glen and Buterin, Vitalik, *Decentralized Society: Finding Web3's Soul*. May 10, 2022. <http://dx.doi.org/10.2139/ssrn.4105763>
26. Agresti, A., & Coull, B. A. (1998). Approximate is Better than “Exact” for Interval Estimation of Binomial Proportions. *The American Statistician*, 52(2), 119–126. <https://doi.org/10.1080/00031305.1998.10480550>.
27. H. Yu, Z. Shen, C. Miao, C. Leung and D. Niyato, "A Survey of Trust and Reputation Management Systems in Wireless Communications," in *Proceedings of the IEEE*, vol. 98, no. 10, pp. 1755-1772, Oct. 2010, doi: 10.1109/JPROC.2010.2059690.
28. Salah, K., Rehman, M. H. U., Nizamuddin, N., & Al-Fuqaha, A. (2019). Blockchain for AI: Review and Open Research Challenges. *IEEE Access*, 7, 10127–10149. <https://doi.org/10.1109/ACCESS.2018.2890507>.
29. Bella, G., Cantone, D., Longo, C., Nicolosi Asmundo, M., Santamaria, D.F. (2022). Blockchains Through Ontologies: The Case Study of the Ethereum ERC721 Standard in oasis. In: Camacho, D., Rosaci, D., Sarné, G.M.L., Versaci, M. (eds) *Intelligent Distributed Computing XIV. IDC 2021. Studies in Computational Intelligence*, vol 1026. Springer, Cham. https://doi.org/10.1007/978-3-030-96627-0_23.
30. Mishra, Amitesh & verma, sumit & ankit,. (2023). Blockchain-Based File Storage using IPFS: A Secure and Transparent Approach, doi: 10.13140/RG.2.2.22075.18722.

31. Piao, Y.; Ye, K.; Cui, X. A Data Sharing Scheme for GDPR-Compliance Based on Consortium Blockchain. *Future Internet* 2021, 13, 217. <https://doi.org/10.3390/fi13080217>.
32. M. Wöhrer and U. Zdun, "Design Patterns for Smart Contracts in the Ethereum Ecosystem," 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), Halifax, NS, Canada, 2018, pp. 1513-1520, doi: 10.1109/Cybermatics_2018.2018.00255.
33. ERC-20 Token Standard. ethereum.org. URL: <https://ethereum.org/developers/docs/standards/tokens/erc-20/> (дата звернення: 17.11.2025).
34. A. Pinna, S. Ibba, G. Baralla, R. Tonelli and M. Marchesi, "A Massive Analysis of Ethereum Smart Contracts Empirical Study and Code Metrics," in *IEEE Access*, vol. 7, pp. 78194-78213, 2019, doi: 10.1109/ACCESS.2019.2921936.
35. Ghazizadeh, E., Sun, T. (2021). A Systematic Literature Review of Smart Contract Applications. In: Arai, K., Kapoor, S., Bhatia, R. (eds) *Proceedings of the Future Technologies Conference (FTC) 2020, Volume 3. FTC 2020. Advances in Intelligent Systems and Computing*, vol 1290. Springer, Cham. https://doi.org/10.1007/978-3-030-63092-8_59.
36. C. Li, "Gas Estimation and Optimization for Smart Contracts on Ethereum," 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), Melbourne, Australia, 2021, pp. 1082-1086, doi: 10.1109/ASE51524.2021.9678932.

ДОДАТКИ

Смарт-контракт Marketplace

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
import "@openzeppelin/contracts/utils/ReentrancyGuard.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

interface IUserRegistryLite {
    function markSeller(address seller) external;
    function recordActivity(address user) external;
    function updateRating(address seller, uint256 sum, uint256 count) external;
}

contract Marketplace is ReentrancyGuard, Ownable {
    using SafeERC20 for IERC20;

    IERC20 public stablecoin;

    uint256 public itemCount;
    uint256 public orderCount;

    uint256 public feeBasisPoints;
    uint256 public accumulatedFees;

    mapping(address => uint256) public sellerCompletedOrders;
    mapping(address => uint256) public sellerDisputeCount;
    mapping(address => uint256) public sellerDisputeBuyerWins;
    mapping(address => uint256) public sellerDisputeSellerWins;

    address public userRegistry;
    address public disputeManager;

    struct Item {
        uint256 id;
        bytes32 publicId;
        address seller;
        uint256 price;
        string itemCID;
        uint256 stock;
        uint256 sold;
        bool active;
    }

    struct Order {
        uint256 id;
        bytes32 publicId;
        uint256 itemId;
        address buyer;
        uint256 quantity;
        uint256 totalPrice;
        string deliveryData;
        bool escrowReleased;
        uint256 createdAt;
        uint256 deliveryConfirmedAt;
        bool reviewed;
        bool countedAsSold;
    }
}
```

```

struct Review {
    uint256 orderId;
    uint256 itemId;
    address buyer;
    uint8 rating;
    string reviewCID;
    uint256 timestamp;
}

mapping(uint256 => Item) public items;
mapping(uint256 => Order) public orders;
mapping(uint256 => Review[]) public reviewsByItem;
mapping(uint256 => mapping(address => bool)) public hasReviewedItem;

mapping(address => uint256) public ratingSum;
mapping(address => uint256) public ratingCount;
mapping(uint256 => uint256) public itemRatingSum;
mapping(uint256 => uint256) public itemRatingCount;

uint256 public totalItemRatingSum;
uint256 public totalItemRatingCount;
uint256 public totalSellerRatingSum;
uint256 public totalSellerRatingCount;

// ----- EVENTS -----
event UserRegistryChanged(address registry);
event DisputeManagerChanged(address newManager);
event ItemCreated(uint256 id, bytes32 publicId, address seller, string itemCID);
event ItemUpdated(uint256 id, bytes32 publicId);
event ItemBought(uint256 orderId, bytes32 publicId);
event DeliveryConfirmed(uint256 orderId, address buyer);
event ReviewPosted(uint256 orderId, uint8 rating, string reviewCID);
event FeeUpdated(uint256 newFee);
event FeeWithdrawn(address admin, uint256 amount);

modifier onlyUserRegistrySet() {
    require(userRegistry != address(0), "UserRegistry not set");
    _;
}

modifier onlyDisputeManager() {
    require(msg.sender == disputeManager, "Only dispute manager");
    _;
}

modifier notReleased(uint256 orderId) {
    require(!orders[orderId].escrowReleased, "Escrow released");
    _;
}

constructor(address _stablecoin) Ownable(msg.sender) {
    require(_stablecoin != address(0), "Zero token");
    stablecoin = IERC20(_stablecoin);
    feeBasisPoints = 250;
}

function setDisputeManager(address dm) external onlyOwner {
    disputeManager = dm;
    emit DisputeManagerChanged(dm);
}

function setUserRegistry(address reg) external onlyOwner {

```

```

        userRegistry = reg;
        emit UserRegistryChanged(reg);
    }

    function setFeeBps(uint256 _feeBps) external onlyOwner {
        require(_feeBps <= 1000, "Fee too high");
        feeBasisPoints = _feeBps;
        emit FeeUpdated(_feeBps);
    }

    function withdrawFees() external onlyOwner {
        uint256 amount = accumulatedFees;
        require(amount > 0, "No fees");
        accumulatedFees = 0;
        stablecoin.safeTransfer(owner(), amount);
        emit FeeWithdrawn(owner(), amount);
    }

    function _computeFee(uint256 price) internal view returns (uint256 fee, uint256
    payout) {
        fee = (price * feeBasisPoints) / 10000;
        payout = price - fee;
    }

    function _markOrderSold(Order storage order, Item storage item) internal {
        if (!order.countedAsSold) {
            item.sold += order.quantity;
            order.countedAsSold = true;
        }
    }

    function createItem(uint256 price, uint256 stock, string calldata itemCID)
    external onlyUserRegistrySet {
        require(price > 0, "Price > 0");
        require(stock > 0, "Stock > 0");
        require(bytes(itemCID).length > 0, "Empty CID");

        itemCount++;
        uint256 itemId = itemCount;

        bytes32 publicId = keccak256(abi.encodePacked(block.chainid, address(this),
        msg.sender, itemId, block.timestamp));

        items[itemId] = Item({
            id: itemId,
            publicId: publicId,
            seller: msg.sender,
            price: price,
            itemCID: itemCID,
            stock: stock,
            sold: 0,
            active: true
        });

        IUserRegistryLite(userRegistry).markSeller(msg.sender);
        emit ItemCreated(itemId, publicId, msg.sender, itemCID);
    }

    function updateStock(uint256 itemId, uint256 newStock) external {
        Item storage item = items[itemId];
        require(item.seller == msg.sender, "Only seller");
        item.stock = newStock;
    }

```

```

        if (item.stock > 0 && !item.active) item.active = true;
        if (item.stock == 0 && item.active) item.active = false;
        emit ItemUpdated(itemId, item.publicId);
    }

    function setItemActive(uint256 itemId, bool newActiveStatus) external {
        Item storage item = items[itemId];
        require(item.seller == msg.sender, "Only seller");

        if (newActiveStatus) {
            require(item.stock > 0, "Stock is 0");
        }

        item.active = newActiveStatus;
        emit ItemUpdated(itemId, item.publicId);
    }

    function updateItem(uint256 itemId, uint256 newPrice, string calldata newItemCID,
        bool newActiveStatus) external {
        Item storage item = items[itemId];
        require(item.seller == msg.sender, "Only seller");

        if (item.sold > 0) {
            require(newPrice == item.price, "Sold items: Price locked");
            item.active = newActiveStatus;
        } else {
            require(newPrice > 0, "Price > 0");
            require(bytes(newItemCID).length > 0, "Empty CID");
            item.price = newPrice;
            item.itemCID = newItemCID;
            item.active = newActiveStatus;
        }

        if (item.stock == 0) item.active = false;

        emit ItemUpdated(itemId, item.publicId);
    }

    function buyItem(uint256 itemId, uint256 quantity, string calldata _deliveryData)
    external nonReentrant onlyUserRegistrySet {
        Item storage item = items[itemId];
        require(item.active, "Inactive");
        require(quantity > 0, "Qty > 0");
        require(quantity <= item.stock, "No stock");
        require(bytes(_deliveryData).length > 0, "Empty delivery data");

        uint256 totalPrice = item.price * quantity;
        stablecoin.safeTransferFrom(msg.sender, address(this), totalPrice);

        item.stock -= quantity;
        if (item.stock == 0) item.active = false;

        orderCount++;
        uint256 orderId = orderCount;
        bytes32 publicId = keccak256(abi.encodePacked(block.chainid, address(this),
            msg.sender, itemId, orderId, block.timestamp));

        orders[orderId] = Order({
            id: orderId,
            publicId: publicId,
            itemId: itemId,
            buyer: msg.sender,

```

```

        quantity: quantity,
        totalPrice: totalPrice,
        deliveryData: _deliveryData,
        escrowReleased: false,
        createdAt: block.timestamp,
        deliveryConfirmedAt: 0,
        reviewed: false,
        countedAsSold: false
    });

    IUserRegistryLite(userRegistry).recordActivity(msg.sender);
    emit ItemBought(orderId, publicId);
}

function confirmDelivery(uint256 orderId) external nonReentrant
notReleased(orderId) {
    Order storage order = orders[orderId];
    require(order.buyer == msg.sender, "Only buyer");
    Item storage item = items[order.itemId];

    (uint256 fee, uint256 payout) = _computeFee(order.totalPrice);
    order.escrowReleased = true;
    order.deliveryConfirmedAt = block.timestamp;

    accumulatedFees += fee;
    stablecoin.safeTransfer(item.seller, payout);

    _markOrderSold(order, item);
    sellerCompletedOrders[item.seller]++;
    emit DeliveryConfirmed(orderId, msg.sender);
}

function releaseEscrowToBuyer(uint256 orderId) external onlyDisputeManager
notReleased(orderId) {
    Order storage order = orders[orderId];
    Item storage item = items[order.itemId];

    stablecoin.safeTransfer(order.buyer, order.totalPrice);
    order.escrowReleased = true;

    item.stock += order.quantity;
    if (!item.active && item.stock > 0) item.active = true;
}

function releaseEscrowToSeller(uint256 orderId) external onlyDisputeManager
notReleased(orderId) {
    Order storage order = orders[orderId];
    Item storage item = items[order.itemId];

    (uint256 fee, uint256 payout) = _computeFee(order.totalPrice);
    order.escrowReleased = true;
    accumulatedFees += fee;

    stablecoin.safeTransfer(item.seller, payout);
    _markOrderSold(order, item);
}

function resolveDisputeToBuyer(uint256 orderId) external onlyDisputeManager
notReleased(orderId) {
    Order storage order = orders[orderId];
    Item storage item = items[order.itemId];

```

```

        sellerDisputeCount[item.seller] += 1;
        sellerDisputeBuyerWins[item.seller] += 1;

        stablecoin.safeTransfer(order.buyer, order.totalPrice);
        order.escrowReleased = true;

        item.stock += order.quantity;
        if (!item.active && item.stock > 0) item.active = true;
    }

    function resolveDisputeToSeller(uint256 orderId) external onlyDisputeManager
    notReleased(orderId) {
        Order storage order = orders[orderId];
        Item storage item = items[order.itemId];

        sellerDisputeCount[item.seller] += 1;
        sellerDisputeSellerWins[item.seller] += 1;

        (uint256 fee, uint256 payout) = _computeFee(order.totalPrice);
        order.escrowReleased = true;
        accumulatedFees += fee;

        stablecoin.safeTransfer(item.seller, payout);
        _markOrderSold(order, item);
    }

    function leaveReview(uint256 orderId, uint8 rating, string calldata reviewCID)
    external onlyUserRegistrySet {
        require(rating >= 1 && rating <= 5, "Rating 1-5");
        require(bytes(reviewCID).length > 0, "Empty CID");

        Order storage order = orders[orderId];
        require(order.buyer == msg.sender, "Only buyer");
        require(order.escrowReleased, "Not complete");
        require(!order.reviewed, "Reviewed");

        uint256 itemId = order.itemId;
        require(!hasReviewedItem[itemId][msg.sender], "Already reviewed");
        hasReviewedItem[itemId][msg.sender] = true;

        Item storage item = items[itemId];

        ratingSum[item.seller] += rating;
        ratingCount[item.seller]++;
        totalSellerRatingSum += rating;
        totalSellerRatingCount++;

        itemRatingSum[itemId] += rating;
        itemRatingCount[itemId]++;
        totalItemRatingSum += rating;
        totalItemRatingCount++;

        order.reviewed = true;
        reviewsByItem[itemId].push(Review({
            orderId: orderId,
            itemId: itemId,
            buyer: msg.sender,
            rating: rating,
            reviewCID: reviewCID,
            timestamp: block.timestamp
        }));
    }

```

```

        IUserRegistryLite(userRegistry).updateRating(item.seller,
ratingSum[item.seller], ratingCount[item.seller]);
        emit ReviewPosted(orderId, rating, reviewCID);
    }

    // ----- VIEWS -----

    function getItemSeller(uint256 itemId) external view returns (address) {
        return items[itemId].seller;
    }

    function getSellerRating(address seller) external view returns (uint256) {
        uint256 count = ratingCount[seller];
        return count == 0 ? 0 : (ratingSum[seller] * 10) / count;
    }

    function getItemRating(uint256 itemId) external view returns (uint256) {
        uint256 count = itemRatingCount[itemId];
        return count == 0 ? 0 : (itemRatingSum[itemId] * 10) / count;
    }

    function getReviewCount(uint256 itemId) external view returns (uint256) {
        return reviewsByItem[itemId].length;
    }

    function getReview(uint256 itemId, uint256 index) external view returns (Review
memory) {
        return reviewsByItem[itemId][index];
    }

    function getItemsBySeller(address seller) external view returns (uint256[]
memory) {
        uint256 total = itemCount;
        uint256 count = 0;
        for (uint256 i = 1; i <= total; i++) {
            if (items[i].seller == seller) count++;
        }
        uint256[] memory result = new uint256[](count);
        uint256 j = 0;
        for (uint256 i = 1; i <= total; i++) {
            if (items[i].seller == seller) {
                result[j] = i;
                j++;
            }
        }
        return result;
    }

    function getOrdersByUser(address user) external view returns (uint256[] memory) {
        uint256 total = orderCount;
        uint256 count = 0;
        for (uint256 i = 1; i <= total; i++) {
            if (orders[i].buyer == user || items[orders[i].itemId].seller == user) {
                count++;
            }
        }
        uint256[] memory result = new uint256[](count);
        uint256 j = 0;
        for (uint256 i = 1; i <= total; i++) {
            if (orders[i].buyer == user || items[orders[i].itemId].seller == user) {
                result[j] = i;
                j++;
            }
        }
    }

```

```

    }
    }
    return result;
}

function getActiveItems(uint256 offset, uint256 limit) external view returns
(Item[] memory) {
    uint256 total = itemCount;
    if (offset >= total) return new Item[](0);
    uint256 end = offset + limit;
    if (end > total) end = total;
    Item[] memory result = new Item[](end - offset);
    for (uint256 i = 0; i < (end - offset); i++) {
        result[i] = items[offset + i + 1];
    }
    return result;
}

function getGlobalItemRating() external view returns (uint256 count, uint256 avg)
{
    count = totalItemRatingCount;
    if (count == 0) return (0, 0);
    avg = (totalItemRatingSum * 10) / count;
}

function getGlobalSellerRating() external view returns (uint256 count, uint256
avg) {
    count = totalSellerRatingCount;
    if (count == 0) return (0, 0);
    avg = (totalSellerRatingSum * 10) / count;
}

// ♦ НОВА ФУНКЦІЯ: Тільки легкі дані для DisputeManager
function getOrderInfoForDispute(uint256 orderId) external view returns (
    address buyer,
    uint256 itemId,
    uint256 createdAt,
    bool escrowReleased
) {
    Order storage o = orders[orderId];
    return (o.buyer, o.itemId, o.createdAt, o.escrowReleased);
}
}

```

Смарт-контракт DisputeManager

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/utils/ReentrancyGuard.sol";

interface IUserRegistry {
    function getSellerCount() external view returns (uint256);
    function getSellerAtIndex(uint256 index) external view returns (address);
    function isSeller(address user) external view returns (bool);
    function lastActivity(address user) external view returns (uint256);
    function getRating(address user) external view returns (uint256);
    function trustedArbitrator(address arb) external view returns (bool);
    function recordActivity(address user) external;
}

interface IMarketplace {
    function getItemSeller(uint256 itemId) external view returns (address);
    function releaseEscrowToBuyer(uint256 orderId) external;
    function releaseEscrowToSeller(uint256 orderId) external;

    function resolveDisputeToBuyer(uint256 orderId) external;
    function resolveDisputeToSeller(uint256 orderId) external;

    function getOrderInfoForDispute(uint256 orderId) external view returns (
        address buyer,
        uint256 itemId,
        uint256 createdAt,
        bool escrowReleased
    );
}

contract DisputeManager is Ownable, ReentrancyGuard {
    uint256 public constant BUYER_WINDOW = 72 hours;
    uint256 public constant SELLER_WINDOW = 48 hours;
    uint256 public constant ARBITRATION_WINDOW = 48 hours;
    uint256 public constant TRUSTED_ARBITRATION_WINDOW = 48 hours;
    uint256 public constant AUTO_DELIVERY_CONFIRM = 7 days;
    uint256 public constant ACTIVITY_WINDOW = 30 days;

    enum DisputePhase { None, BuyerEvidence, SellerEvidence, Arbitration,
        TrustedArbitration, Finished }
    enum VoteOption { None, Buyer, Seller }

    struct Dispute {
        uint256 orderId;
        address buyer;
        address seller;
        uint256 openedAt;
        uint256 phaseStartedAt;
        DisputePhase phase;
        string buyerEvidenceCID;
        string sellerEvidenceCID;
        address[3] primaryArbitrators;
        mapping(address => VoteOption) votes;
        address[] primaryVoters;
        uint256 buyerVotes;
        uint256 sellerVotes;
    }
}
```

```

        mapping(address => VoteOption) trustedVotes;
        address[] trustedVoters;
        uint256 trustedBuyerVotes;
        uint256 trustedSellerVotes;
        bool resolved;
        bool buyerWin;
    }

    struct DisputeView {
        uint256 orderId;
        address buyer;
        address seller;
        uint256 openedAt;
        uint256 phaseStartedAt;
        DisputePhase phase;
        string buyerEvidenceCID;
        string sellerEvidenceCID;
        address[3] primaryArbitrators;
        uint256 buyerVotes;
        uint256 sellerVotes;
        uint256 trustedBuyerVotes;
        uint256 trustedSellerVotes;
        bool resolved;
        bool buyerWin;
    }

    IMarketplace public marketplace;
    IUserRegistry public userRegistry;

    mapping(uint256 => Dispute) public disputes;
    mapping(address => uint256[]) public arbitratorAssignments;

    event DisputeOpened(uint256 indexed orderId, address buyer, address seller);
    event BuyerEvidenceSubmitted(uint256 orderId, string evidenceCID);
    event SellerEvidenceSubmitted(uint256 orderId, string evidenceCID);
    event ArbitrationStart(uint256 orderId, address[3] arbitrators);
    event VoteCast(uint256 orderId, address arb, VoteOption vote);
    event TrustedVoteCast(uint256 orderId, address arb, VoteOption vote);
    event DisputeResolved(uint256 orderId, bool buyerWin);
    event PhaseTimedOut(uint256 orderId, DisputePhase newPhase);

    constructor(address marketplaceAddr, address userRegistryAddr)
    Ownable(msg.sender) {
        marketplace = IMarketplace(marketplaceAddr);
        userRegistry = IUserRegistry(userRegistryAddr);
    }

    function getDisputesForArbitrator(address arb) external view returns (uint256[]
memory) {
        return arbitratorAssignments[arb];
    }

    function getDisputeDetails(uint256 orderId) external view returns (DisputeView
memory) {
        Dispute storage d = disputes[orderId];
        return DisputeView(
            d.orderId, d.buyer, d.seller,
            d.openedAt, d.phaseStartedAt,
            d.phase,
            d.buyerEvidenceCID, d.sellerEvidenceCID,
            d.primaryArbitrators,
            d.buyerVotes, d.sellerVotes,

```

```

        d.trustedBuyerVotes, d.trustedSellerVotes,
        d.resolved, d.buyerWin
    );
}

function openDispute(uint256 orderId) external {
    (address buyer, uint256 itemId, uint256 createdAt, bool escrowReleased) =
    marketplace.getOrderInfoForDispute(orderId);
    require(buyer != address(0), "Order not found");
    require(buyer == msg.sender, "Only buyer");
    require(!escrowReleased, "Already completed");
    require(block.timestamp <= createdAt + AUTO_DELIVERY_CONFIRM, "Too late");

    Dispute storage d = disputes[orderId];
    require(d.phase == DisputePhase.None, "Already disputed");

    address seller = marketplace.getItemSeller(itemId);

    d.orderId = orderId;
    d.buyer = msg.sender;
    d.seller = seller;
    d.phase = DisputePhase.BuyerEvidence;
    d.openedAt = block.timestamp;
    d.phaseStartedAt = block.timestamp;

    emit DisputeOpened(orderId, d.buyer, d.seller);
}

function submitBuyerEvidence(uint256 orderId, string calldata evidenceCID)
external {
    Dispute storage d = disputes[orderId];
    require(d.phase == DisputePhase.BuyerEvidence, "Wrong phase");
    require(msg.sender == d.buyer, "Only buyer");

    if (block.timestamp > d.phaseStartedAt + BUYER_WINDOW) {
        _resolveSeller(orderId);
        return;
    }
    d.buyerEvidenceCID = evidenceCID;
    emit BuyerEvidenceSubmitted(orderId, evidenceCID);
    d.phase = DisputePhase.SellerEvidence;
    d.phaseStartedAt = block.timestamp;
}

function submitSellerEvidence(uint256 orderId, string calldata evidenceCID)
external {
    Dispute storage d = disputes[orderId];
    require(d.phase == DisputePhase.SellerEvidence, "Wrong phase");
    require(msg.sender == d.seller, "Only seller");

    if (block.timestamp > d.phaseStartedAt + SELLER_WINDOW) {
        _resolveBuyer(orderId);
        return;
    }
    d.sellerEvidenceCID = evidenceCID;
    emit SellerEvidenceSubmitted(orderId, evidenceCID);
    d.phase = DisputePhase.Arbitration;
    d.phaseStartedAt = block.timestamp;
    _selectPrimaryArbitrators(orderId);
}

function _selectPrimaryArbitrators(uint256 orderId) internal {

```

```

Dispute storage d = disputes[orderId];
address[] memory candidates = _pickRandomArbitrators(d.buyer, d.seller);

if (candidates.length < 3) {
    d.phase = DisputePhase.TrustedArbitration;
    d.phaseStartedAt = block.timestamp;
    emit PhaseTimedOut(orderId, DisputePhase.TrustedArbitration);
    return;
}

for (uint256 i = 0; i < 3; i++) {
    d.primaryArbitrators[i] = candidates[i];
    arbitratorAssignments[candidates[i]].push(orderId);
}
emit ArbitrationStart(orderId, d.primaryArbitrators);
}

function _pickRandomArbitrators(address buyer, address seller) internal view
returns (address[] memory) {
    uint256 totalSellers = userRegistry.getSellerCount();
    if (totalSellers == 0) return new address[](0);

    address[] memory temp = new address[](3);
    uint256 found = 0;
    uint256 startIndex = uint256(keccak256(abi.encodePacked(block.timestamp,
buyer, seller))) % totalSellers;
    uint256 maxAttempts = 50;
    if (totalSellers < maxAttempts) maxAttempts = totalSellers;

    for (uint256 i = 0; i < maxAttempts; i++) {
        if (found >= 3) break;
        uint256 idx = (startIndex + i) % totalSellers;
        address candidate = userRegistry.getSellerAtIndex(idx);
        if (candidate == buyer || candidate == seller) continue;
        if (block.timestamp - userRegistry.lastActivity(candidate) >
ACTIVITY_WINDOW) continue;
        temp[found] = candidate;
        found++;
    }

    address[] memory result = new address[](found);
    for(uint256 k=0; k<found; k++) result[k] = temp[k];
    return result;
}

function vote(uint256 orderId, VoteOption v) external {
    Dispute storage d = disputes[orderId];
    require(d.phase == DisputePhase.Arbitration, "Not arbitration");
    if (block.timestamp > d.phaseStartedAt + ARBITRATION_WINDOW) {
        d.phase = DisputePhase.TrustedArbitration;
        d.phaseStartedAt = block.timestamp;
        emit PhaseTimedOut(orderId, DisputePhase.TrustedArbitration);
        return;
    }
    require(v == VoteOption.Buyer || v == VoteOption.Seller, "Invalid");

    bool isPrimary = false;
    for (uint256 i = 0; i < 3; i++) {
        if (d.primaryArbitrators[i] == msg.sender) {
            isPrimary = true;
            break;
        }
    }
}

```

```

    }
    require(isPrimary, "Not primary");
    require(d.votes[msg.sender] == VoteOption.None, "Already voted");

    d.votes[msg.sender] = v;
    d.primaryVoters.push(msg.sender);

    if (v == VoteOption.Buyer) d.buyerVotes++;
    else d.sellerVotes++;

    userRegistry.recordActivity(msg.sender);
    emit VoteCast(orderId, msg.sender, v);

    if (d.buyerVotes >= 2) return _resolveBuyer(orderId);
    if (d.sellerVotes >= 2) return _resolveSeller(orderId);
}

function trustedVote(uint256 orderId, VoteOption v) external {
    Dispute storage d = disputes[orderId];
    require(d.phase == DisputePhase.TrustedArbitration, "Wrong phase");
    if (block.timestamp > d.phaseStartedAt + TRUSTED_ARBITRATION_WINDOW) {
        _resolveBuyer(orderId);
        return;
    }
    require(userRegistry.trustedArbitrator(msg.sender), "Not trusted");
    require(d.trustedVotes[msg.sender] == VoteOption.None, "Already voted");
    require(v == VoteOption.Buyer || v == VoteOption.Seller, "Invalid");

    d.trustedVotes[msg.sender] = v;
    d.trustedVoters.push(msg.sender);

    if (v == VoteOption.Buyer) d.trustedBuyerVotes++;
    else d.trustedSellerVotes++;

    userRegistry.recordActivity(msg.sender);
    emit TrustedVoteCast(orderId, msg.sender, v);

    if (d.trustedBuyerVotes >= 2) return _resolveBuyer(orderId);
    if (d.trustedSellerVotes >= 2) return _resolveSeller(orderId);
}

function checkTimeouts(uint256 orderId) external {
    Dispute storage d = disputes[orderId];
    require(!d.resolved, "Resolved");
    uint256 t = block.timestamp;
    if (d.phase == DisputePhase.BuyerEvidence) {
        if (t > d.phaseStartedAt + BUYER_WINDOW) _resolveSeller(orderId);
    } else if (d.phase == DisputePhase.SellerEvidence) {
        if (t > d.phaseStartedAt + SELLER_WINDOW) _resolveBuyer(orderId);
    } else if (d.phase == DisputePhase.Arbitration) {
        if (t > d.phaseStartedAt + ARBITRATION_WINDOW) {
            d.phase = DisputePhase.TrustedArbitration;
            d.phaseStartedAt = t;
            emit PhaseTimedOut(orderId, DisputePhase.TrustedArbitration);
        }
    } else if (d.phase == DisputePhase.TrustedArbitration) {
        if (t > d.phaseStartedAt + TRUSTED_ARBITRATION_WINDOW)
        _resolveBuyer(orderId);
    }
}

function autoConfirmDelivery(uint256 orderId) external {

```

```

        (, , uint256 createdAt, bool escrowReleased) =
marketplace.getOrderInfoForDispute(orderId);
        require(!escrowReleased, "Released");
        require(disputes[orderId].phase == DisputePhase.None, "Dispute active");
        require(block.timestamp > createdAt + AUTO_DELIVERY_CONFIRM, "Too early");

        marketplace.releaseEscrowToSeller(orderId);
    }

    function _resolveBuyer(uint256 orderId) internal {
        Dispute storage d = disputes[orderId];
        if (d.resolved) return;

        d.resolved = true;
        d.buyerWin = true;
        d.phase = DisputePhase.Finished;

        marketplace.resolveDisputeToBuyer(orderId);
        emit DisputeResolved(orderId, true);
    }

    function _resolveSeller(uint256 orderId) internal {
        Dispute storage d = disputes[orderId];
        if (d.resolved) return;

        d.resolved = true;
        d.buyerWin = false;
        d.phase = DisputePhase.Finished;

        marketplace.resolveDisputeToSeller(orderId);
        emit DisputeResolved(orderId, false);
    }

    function getPrimaryVote(uint256 orderId, address arb) external view returns
(VoteOption) {
        return disputes[orderId].votes[arb];
    }

    function getTrustedVote(uint256 orderId, address arb) external view returns
(VoteOption) {
        return disputes[orderId].trustedVotes[arb];
    }

    function getTrustedVoters(uint256 orderId) external view returns (address[]
memory) {
        return disputes[orderId].trustedVoters;
    }
}

```

Смарт-контракт UserRegistry

```

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/access/Ownable.sol";

contract UserRegistry is Ownable {
    address public marketplace;
    address public disputeManager;

    event MarketplaceUpdated(address indexed newMarketplace);
    event DisputeManagerUpdated(address indexed newDisputeManager);
    event SellerAdded(address indexed seller);
    event ActivityRecorded(address indexed user, uint256 timestamp);
    event RatingUpdated(address indexed seller, uint256 newSum, uint256 newCount);
    event TrustedArbitratorStatusChanged(address indexed arbitrator, bool status);

    modifier onlyMarketplace() {
        require(msg.sender == marketplace, "Only marketplace");
        _;
    }

    modifier onlyMarketplaceOrDisputeManager() {
        require(msg.sender == marketplace || msg.sender == disputeManager, "Only Mkt
or DM");
        _;
    }

    constructor() Ownable(msg.sender) {}

    function setMarketplace(address m) external onlyOwner {
        require(m != address(0), "Zero marketplace");
        marketplace = m;
        emit MarketplaceUpdated(m);
    }

    function setDisputeManager(address dm) external onlyOwner {
        require(dm != address(0), "Zero disputeManager");
        disputeManager = dm;
        emit DisputeManagerUpdated(dm);
    }

    // ----- SELLERS -----
    address[] public sellerList;
    mapping(address => uint256) public sellerIndex; // 1-based
    mapping(address => uint256) public ratingSum;
    mapping(address => uint256) public ratingCount;
    mapping(address => uint256) public lastActivity;

    function markSeller(address seller) external onlyMarketplace {
        require(seller != address(0), "Zero seller");
        if (sellerIndex[seller] == 0) {
            sellerList.push(seller);
            sellerIndex[seller] = sellerList.length;
            emit SellerAdded(seller);
        }
        lastActivity[seller] = block.timestamp;
    }

```

```

        emit ActivityRecorded(seller, block.timestamp);
    }

    function getAllSellers() external view returns (address[] memory) {
        return sellerList;
    }

    function getSellerAtIndex(uint256 index) external view returns (address) {
        require(index < sellerList.length, "OOB");
        return sellerList[index];
    }

    function getSellerCount() external view returns (uint256) {
        return sellerList.length;
    }

    function recordActivity(address user) external onlyMarketplaceOrDisputeManager {
        lastActivity[user] = block.timestamp;
        emit ActivityRecorded(user, block.timestamp);
    }

    function updateRating(address seller, uint256 sum, uint256 count) external
onlyMarketplace {
        ratingSum[seller] = sum;
        ratingCount[seller] = count;
        emit RatingUpdated(seller, sum, count);
    }

    // ----- VIEWS -----
    function isSeller(address a) external view returns (bool) {
        return sellerIndex[a] != 0;
    }

    function getRating(address a) external view returns (uint256) {
        if (ratingCount[a] == 0) return 0;
        return (ratingSum[a] * 10) / ratingCount[a];
    }

    // ----- Trusted Arbitrators -----
    mapping(address => bool) public trustedArbitrator;

    function addTrustedArbitrator(address arb) external onlyOwner {
        trustedArbitrator[arb] = true;
        lastActivity[arb] = block.timestamp;
        emit TrustedArbitratorStatusChanged(arb, true);
    }

    function removeTrustedArbitrator(address arb) external onlyOwner {
        trustedArbitrator[arb] = false;
        emit TrustedArbitratorStatusChanged(arb, false);
    }
}

```

Смарт-контракт TestUSDC

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract TestUSDC is ERC20 {
    constructor() ERC20("Test USDC", "USDC") {}

    function mint(address to, uint amount) external {
        _mint(to, amount);
    }
}
```