Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем

Розробка системи криптоплатежів з використанням технології блокчейн

Текстова частина до кваліфікаційної роботи

за спеціальністю «Комп'ютерні науки» - 122

Керівник кваліфікаційної роботи

старший викладач

Гороховський К. С.

_____ (підпис)

"___"_____ 2025 року

Виконав студент

КН-4 Сметанюк В. О.

"___"_____ 2025 року

Київ 2025

Міністерство освіти у науки України

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем

ЗАТВЕРДЖУЮ

старший викладач

Гороховський К. С.

_____ (підпис)

“___”_____ 2024 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на кваліфікаційну роботу

студенту Сметанюку Володимиру Олексійовичу 4-го року навчання факультету інформатики

ТЕМА: РОЗРОБКА СИСТЕМИ КРИПТОПЛАТЕЖІВ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ БЛОКЧЕЙН

Зміст ТЧ до кваліфікаційної роботи:

Індивідуальне завдання
Календарний план
Анотація
Вступ
Розділ 1. Дослідження предметної області та аналіз існуючих рішень
Розділ 2. Аналіз технічного завдання та алгоритму захисту від підробки параметрів
Розділ 3. Розробка застосунку та тестування
Висновки
Список використаної літератури
Додатки (за необхідністю)

Дата видачі “_____”_____2024р. **Керівник** _____
(підпис)

Завдання отримав _____
(підпис)

Календарний план виконання роботи

№	Назва етапу кваліфікаційної роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на кваліфікаційну роботу	01.10.2024	
2.	Огляд літератури за темою роботи	01.11.2024	
3.	Проведення дослідження	01.12.2024	
4.	Написання програмного застосунку	01.02.2025	
5.	Написання текстової частини	10.04.2025	
6.	Захист кваліфікаційної роботи		

Студент _____

Керівник _____ “___”_____ 2024р.

Зміст

Зміст 4
Анотація 6
Вступ 7
РОЗДІЛ 1. Дослідження предметної області та аналіз існуючих рішень 8
1.1 Проблематика 8
1.2 Блокчейн 9
1.2.1 Обґрунтування використання Ethereum та EVM-сумісних мереж 9
1.3 Stablecoin 10
1.3.1 Обґрунтування використання стейблкоїнів 10
1.4 ECDSA 11
1.4.1 Обґрунтування використання ECDSA 12
1.5 Огляд наявних рішень 12
1.5.1 Stripe 12
1.5.2 NOWPayments 12
1.5.3 CoinsPaid 13
1.6 Висновки 13
РОЗДІЛ 2. Аналіз технічного завдання та алгоритму захисту від підробки параметрів 14
2.1 Вимоги до рішення 14
2.2 Ролі користувачів 14
2.2.1 Розподіл можливостей користувачів за ролями 14
2.3 Специфікації вимог до даних 15
2.4 User-flows 16
2.4.1 Розгортання контрактів 16
2.4.2 Створення заявки на оплату 17
2.4.3 Проведення оплати 18
2.4.4 Вивід коштів 18
2.4.5 Зміна підтримуваних токенів 19

2.5 Побудова алгоритму захисту від підробки параметрів 19
2.6 Висновки 21
РОЗДІЛ 3. Розробка застосунку та тестування 22
3.1 Опис архітектури 22
3.1.1 Processor microservice (core) 23
3.1.2 Signer microservice 24
3.1.3 Observer microservice 24
3.1.4 Notifications microservice 24
3.2 Використані інструменти та засоби розробки 24
3.2.1 Golang 24
3.2.2 PostgreSQL 25
3.2.3 CryptoCompare API 26
3.2.4 Solidity та OpenZeppelin 26
3.2.5 Solc, abigen, hardhat 27
3.2.6 Docker 27
3.3 Розробка смарт контракту 27
3.3.1 Реалізація алгоритму захисту від підробки параметрів на контракті 28
3.3.2 Реалізація алгоритму захисту від підробки параметрів на бекенді 30
3.4 Тестування 32
3.5 Результати роботи 33
3.6 Висновки 42
Список використаної літератури 43

Анотація

У даній кваліфікаційній роботі розглянуто наявні рішення криптоплатіжних систем, проаналізовано їхні переваги та недоліки, запропоновано компактне та портативне рішення.

Розроблено алгоритм захисту від повторного використання параметрів, проведено огляд ключових компонентів блокчейн екосистеми та платіжних платформ, проаналізовано та обґрунтовано використання алгоритму цифрового підпису ECDSA, стейблкоїнів, EVM-сумісних блокчейнів та їхніх переваг.

Сформовано технічні вимоги до платіжної системи, визначено основні ролі користувачів та їхні можливості. Проаналізовано специфікації вимог до даних, та обмежень.

Розроблено платіжну систему з використанням мікросервісної архітектури та різних мережених протоколів. Проведено тестування у двох стратегіях. Продемонстровано повний цикл використання готового програмного продукту.

Ключові слова: мікросервісна архітектура, блокчейн, сервіси, бази даних, функції хешування, ECDSA, Ethereum.

Вступ

У сучасному світі використання технології блокчейн стає все більш невід’ємною практикою та повсякденністю. Концепція Web3, складовою якої є розподілені децентралізовані сховища, надає можливість будь-якому пристрою, що має доступ до інтернету, використовувати всю потужність цих технологій.

З погляду на те, що більшість сучасних сервісів, що надають послуги, мають web-версію, а використання технології блокчейн надає можливість використовувати ціннісні криптоактиви, як засіб оплати за товари та послуги, то очевидним стає поєднання цих елементів для створення нових зручних застосунків.

Актуальність цієї роботи полягає в тому, що на нашу думку, інтеграція технології блокчейн з веб застосунками, що надають певні послуги, вирішуватиме наступні проблеми:

- Надмірна централізованість;
- Використання спеціалізованих закритих рішень;
- Завищені розміри комісійних стягнень за використання готових рішень;

Об’єктом дослідження є аналіз роботи існуючих рішень криптоплатіжних систем та виправлення їхніх недоліків.

Предметом дослідження є розробка компактного, портативного та легкого для інтегрування рішення для проведення криптоплатежів.

Метою дослідження є спроба вирішити недоліки популярних існуючих застосунків.

РОЗДІЛ 1. Дослідження предметної області та аналіз існуючих рішень

1.1 Проблематика

Із зростанням популярності децентралізованих технологій та криптовалют, і, як наслідок, поширення використання даних технологій, з’явився попит на їхню інтеграцію у традиційні web2 мобільні та веб-застосунки. Ключовим напрямком використання є можливість проводити платежі у криптовалюті на базі блокчейнів.

Сучасні застосунки переважно використовують централізовані платіжні системи такі як: PayPal, Stripe, LiqPay, або ж API банківський систем для оплати картками. Основними проблеми даних рішень є: інколи високі комісії за операції, залежність від банків та фінансових установ, обмежене використання у певних країнах та регіонах, ймовірність шахрайства та скасування платежів.

Незважаючи на переваги використання блокчейнів як децентралізованих платіжних систем таких як: зменшення витрат за проведення операцій, покращення прозорості та відкритості створення платежів; використання нативних та не стейблкойн токенів робить непривабливим інтеграцію блокчейнів через їхню високу волатильність. Саме тому у даних роботі більше уваги приділятиметься використанню стейблкойнів [1].

Хоча на даний момент уже існує багато сервісів, які надають послуги криптоплатежів, більшість з них для цього використовують або власні кастодіальні сервіси (тобто продавець не має прямого доступу до приватних ключів, тобто власних коштів), або додають комісію за користування власним або сторонніми посередницькими сервісами для успішної обробки та конвертації крипто активів.

Попри значні можливості та короткий час інтеграції існуючих рішень крипто-платіжних систем, у користувача з’являється вразливість у вигляді єдиної точки відмови і, як наслідок, можливість втрати контролю над власними активами, що є найбільшою проблемою. Запропоноване рішення у дані роботі має на меті виправлення цього недоліку.

1.2 Блокчейн

Блокчейн, що є технологією розподілених реєстрів, дозволяє користувачам створювати незмінні записи та обмінюватися загально-відомим станом системи в розподіленій мережі. Це надає можливість для обміну цінностями між різними користувачами; мати доступ до спільних довірених реєстрів як єдиного джерела істини; безпечно виконувати угоди з використанням смарт-контрактів [2].

Оглядово, блокчейн - це сукупність програм, алгоритмів та протоколів, які дозволяють будувати прозорі, надійні, швидкі та ефективні транзакції [3]. Основами будь-якого розподіленого реєстру є алгоритм консенсусу, еліптичні криві та функції хешування для забезпечення непідробності записів, та механізм підтвердження транзакцій та блоків [4].

Сьогодні існує багато варіацій поєднання різних алгоритмів консенсусу, валідації та перевірок, які реалізують різні протоколи. Найпоширенішою групою мереж наразі є система Ethereum та сукупність EVM-сумісних L2 блокчейнів.

1.2.1 Обґрунтування використання Ethereum та EVM-сумісних мереж

Ethereum був першим блокчейном, який започаткував використання смарт-контрактів у своєму середовищі. З того часу ця мережа є однією з найбільш популярних платформ для розробки dApps (децентралізованих застосунків).

Очевидною перевагою є наявність EVM (Ethereum Virtual Machine), що дає змогу писати уніфіковані контракти, які в свою чергу можуть працювати на усіх інших EVM-сумісних блокчейнах. Тому фіналізований контракт достатньо продублювати на інших мережах для розширення.

Також зручним є наявність широкої екосистеми та великої кількості загальноприйнятих стандартів, готових рішень, реалізованих контрактів. Прикладом є фреймворк OpenZeppelin.

Хоча недоліком Ethereum є відносно великий час підтвердження та достатньо висока вартість проведення транзакцій, рішенням цих проблеми є L2 блокчейни такі як: Polygon zkEVM, Optimism, Arbitrum, StarkNet та багато інших. Усі вони також працюють з використанням EVM, відповідно немає складнощів перенести написані контракти на ці мережі.

1.3 Stablecoin

Основною проблемою використання взаємозамінних токенів (fungible tokens) у сучасних блокчейнах є відсутність стабілізації ціни та стійкості. Стейблкоіни - це підмножина взаємозамінних токенів, які створені для урегулювання проблеми волатильності та підвищення децентралізації [1].

Традиційні криптовалюти, такі як ETH (Ethereum), BTC (Bitcoin), SOL (Solana), схильні до сильних частих стрибків у ціні, відповідно для реалізації платіжної системи використання їх є непривабливим. Можливим рішенням є проведення платежів з одночасною конвертацією отриманих коштів у стейблкоїн для забезпечення стійкості, проте таке рішення потребую додаткових витрат на створення конвертаційних транзакцій або використання сторонніх сервісів, які так само потребують комісії за дані послуги.

1.3.1 Обґрунтування використання стейблкоїнів

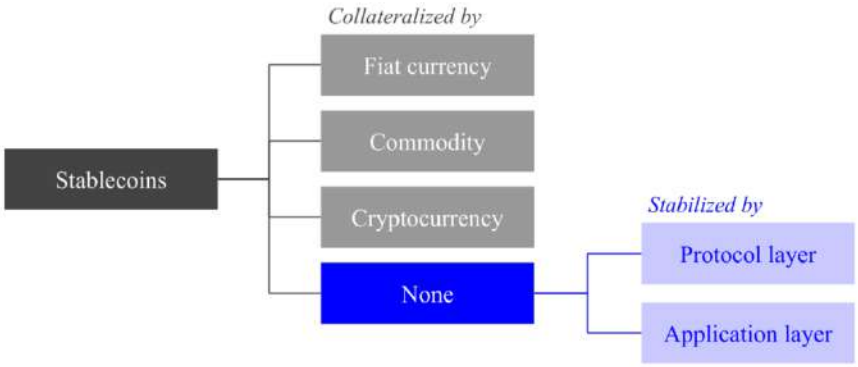

Рис. 1.3.1. Класифікація стейблкоїнів

Існує достатньо широкий набір класів стейблкоїнів, проте для рішення у даній роботі основне значення має не клас, а висока стійкість монети. Для забезпечення якомога меншого відхилення між транзакціями, зробленими в різний час, використання стейблкоїнів є оптимальним.

Отже, основною причиною використання такого типу монет є забезпечення незмінності розміру оплати для користувача платіжної системи.

1.4 ECDSA

Алгоритм цифрового підпису (DSA) - це пара дуже великих чисел, що представлені у текстовому або бінарному форматі [5].

Алгоритм цифрового підпису з використанням еліптичної кривої (ECDSA) - це криптографічний алгоритм для створення цифрового підпису, що використовує еліптичну криву. Такий підпис є вдосконаленням традиційного DSA та забезпечує цілісність і відмовостійкість у цифрових комунікаціях [6].

ECDSA використовує операцій додавання, множення та інші властивості еліптичних кривих для генерації пари приватного і публічного ключів та створення цифрового підпису [7].

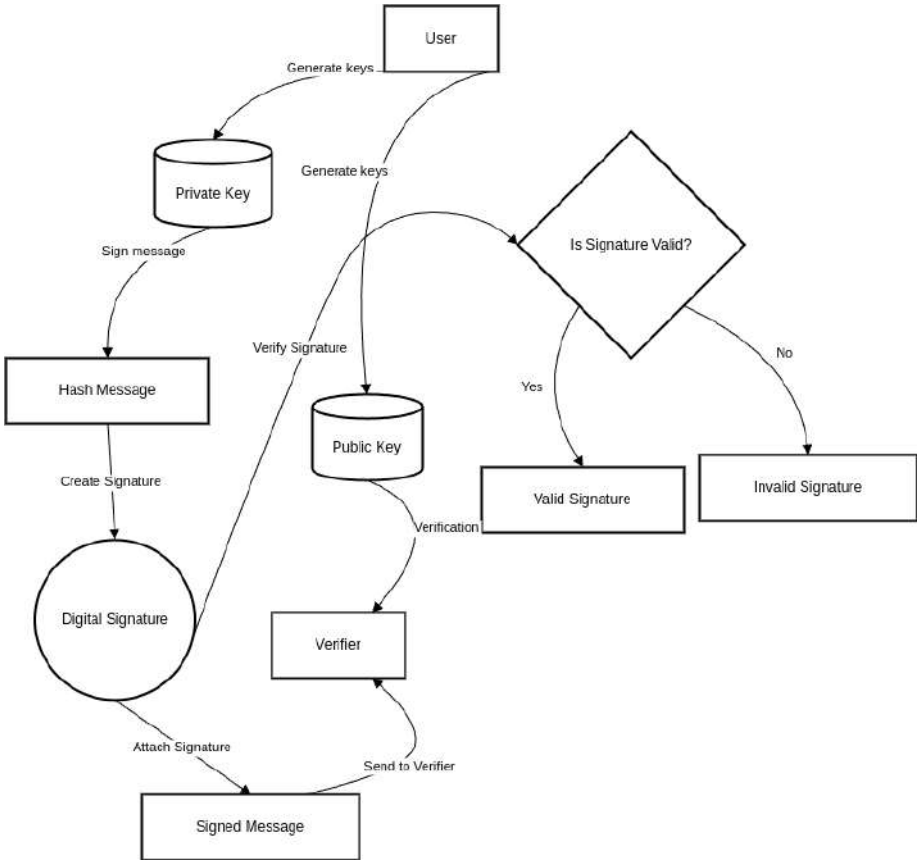

Рис. 1.4.1. Схема алгоритму підпису ECDSA

Перевагами використання ECDSA, як і інших DSA алгоритмів є: генерування приватного ключа з випадкової величини, підписання повідомлення фіксованої довжини та перевірка валідності підпису, якщо відомі публічний ключ підписанта та хеш повідомлення [7].

1.4.1 Обґрунтування використання ECDSA

Попри широке використання у протоколах, таких як TLS, SSL, PGP, SSH та інших; ECDSA є основним алгоритмом підпису транзакцій у мережі Ethereum та L2 рішеннях. Відповідно, використання даного алгоритму оптимізовано у наведених раніше блокчейнах, саме тому для реалізації алгоритму забезпечення додаткового захисту буде використано ECDSA.

1.5 Огляд наявних рішень

1.5.1 Stripe

Stripe - це одна з найпопулярніших платіжних платформ, яка підтримує роботу як з фіатними активами, так і з криптовалютами [8].

Дана платформа діє як посередник між покупцем та продавцем і забезпечує захищений переказ коштів. До переваг можна віднести наступне:

- підтримка багатьох криптовалют для оплати [9];
- налаштована інтеграція Web3 для взаємодії з криптогаманцями [8];
- автоматична конвертація отриманих коштів у фіат [8];
- можливість інтеграції з іншими Web3 інструментами [8].

Основними недоліками є: висока комісія за конвертацію, обмежений список підтримуваних токенів, централізованість.

1.5.2 NOWPayments

Дана платіжна платформа підтримує понад 160 криптовалют та надає простий API для інтеграції з мобільними застосунками, сайтами та dApps [10].

NOWPayments також працює як посередник між покупцем та продавцем. Серед переваг можна виокремити наступне:

- для кожної транзакції сервіс генерує нову проміжну адресу, на яку буде здійснено платіж [10];
- вивід коштів відбувається безпосередньо на адресу продавця [10];
- є можливість додати автоматичну конвертацію вхідних активів у обрану криптовалюту [10].

Недоліки:

- комісія за кожну транзакцію;
- обмежена інтеграція з банківськими системами;

- відсутність можливості конвертації у фіат.

1.5.3 CoinsPaid

Ця реалізація платіжної платформи працює за схемою частково кастодіальної системи (payment gateway). Продавець має частковий або не має доступу до отриманих коштів. З іншого боку, даний підхід забезпечує більшу безпеку роботи з приватними ключами та перетягує відповідальність за це на сам сервіс [11].

Переваги даного рішення:

- створення нової платіжної адреси для кожної нової оплати [11];
- автоматизований розподіл коштів на внутрішніх гаманцях [11];
- можливість налаштувати автоматичну конвертацію між криптовалютами та фіатом [11];
- висока швидкість проведення транзакцій, оптимізована використанням “гарячих” гаманців [11];
- забезпечення сильного захисту через використання шифрування, регулярних аудитів та мультипідписів [11];
- підтримка інтеграції з DeFi-протоколами [11];
- велика кількість підтримуваних валют (понад 30) [11].

Недоліки:

- зберігання приватних ключів на платформі;
- висока залежність від системи через кастодіальну систему: відсутність прямого доступу до управління коштами.

1.6 Висновки

У цьому розділі було проведено огляд ключових компонентів блокчейн екосистеми та платіжних платформ, проаналізовано та обґрунтовано використання алгоритму цифрового підпису ECDSA, стейблкоїнів, EVM-сумісних блокчейнів та їхніх переваг.

Також було проведено аналіз популярних комерційних рішень, що дають можливість легко інтегрувати криптоплатежі у власні застосунки, наведено їхні переваги та недоліки

РОЗДІЛ 2. Аналіз технічного завдання та алгоритму захисту від підробки параметрів

2.1 Вимоги до рішення

З огляду на різноманітність існуючих рішень, їхні переваги та недоліки, запропонований у даній роботі застосунок має на меті зберегти якомога більше переваг та вирішити недоліки аналогів, що найчастіше зустрічаються.

Отже, застосунок має:

- бути реалізованим з використанням посередницького підходу;
- мінімізувати додаткові витрати на користування;
- надавати сповіщення продавцеві про статус проведення платежів;
- бути універсальним, розширюваним та портативним;
- забезпечувати механізм захисту під час проведення ключових транзакцій;
- мати зручний API та буде легким для інтегрування;

2.2 Ролі користувачів

У даному застосунку не планується використовувати модель акаунтів та чіткий розподіл на різні типи користувачів. Замість цього буде використано підхід розмежування рівнів API: для продавця (власника) буде доступна окрема група API ендпоінтів, а для покупця (користувача) - інша, з обмеженнями.

Абстрактних ролей все ж таки дві:

- продавець - має повний доступ над системою, окремий API для керування та збору інформації про платежі.
- покупець - має обмежений доступ до функцій, що необхідні створення та проведення платежу через окремий API.

2.2.1 Розподіл можливостей користувачів за ролями

Продавець може**:**

- додати підтримку нової мережі: розгорнути контракт з визначення підтримуваних токенів.
- додати\прибрати підтримуваний токен
- отримувати сповіщення про статус платежу

- переглянути дані про платіж\платежі
- вивести отримані кошти з смартконтрактів на бажану адресу у певній мережі

Покупець може:

- здійснити платіж у бажаній мережі та бажаному токені (з тих, що підтримуються)
- скасувати платіж (відмовитись до надсилання фактичної транзакції)

2.3 Специфікації вимог до даних

- Про підтримувані мережі (networks) має бути збережено:
  - назва мережі;
  - адреса контракту (після того, як його було розгорнуто).
- Про підтримуваний токен (token) має бути збережено:
  - назва мережі, на якій він існує;
  - назва токену;
  - адреса контракту токену.
- Про заявку на оплату (invoice) має бути збережено:
  - унікальний ідентифікатор;
  - назва токена закріплення суми оплати;
  - власне значення для оплати;
  - URL для надсилання сповіщень про статус оплати;
  - URL для переадресації покупця після будь-якого з варіантів завершення оплати;
  - відмітка часу, після якої заявка вважається недійсною;
  - час створення.
- Про оплату (payment) має бути збережено:
  - пов'язаний унікальний ідентифікатор заявки на оплату;
  - назва мережі здійснення оплати;
  - назва токена, у якому відбулась оплата;
  - фактичний розмір оплати (можливо конвертований, якщо оплата була проведена у відмінному від заявки токені);
  - ідентифікатор транзакції on-chain;
  - статус платежу;
  - час створення платежу.
- Заборонено створення платежу після того, як заявка набула статусу недійсності.

- Заборонено створювати заявку на платіж з токеном, який не підтримуються на цей момент.
- Заборонено створювати платіж на мережі та/або з токеном, які не підтримуються на цей момент.
- Заборонено створювати більше одного платежу на пов’язаний ідентифікатор заявки.

2.4 User-flows

Застосунок матиме п’ять основних користувацьких шляхів. Для успішного виконання продавцеві або покупцеві потрібно буде взаємодіяти з API частиною застосунку та інколи з графічним web інтерфейсом. Також продавцеві будуть доступні деякі інші можливість виключно через його API рівень, що даватимуть змогу отримати дані про оплати, історичні дані. Розглянемо основні з них.

2.4.1 Розгортання контрактів

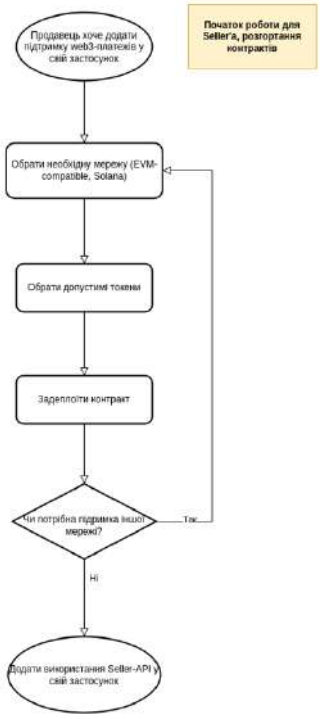

Рис. 2.4.1.1. User-flow розгортання контрактів

Для додавання підтримки нової мережі продавцеві необхідно зі списку доступних мереж (тих, що підтримує сама платіжна система) обрати потрібну, та ініціювати транзакцію розгортання. Після успішного підписання та підтвердження контракт стає доступним на обраній мережі і, як наслідок, розгорнута платіжна система також зможе опрацьовувати платежі у цьому блокчейні.

Власником (owner) контракту є користувач, з адреси якого була надіслана транзакція розгортання. Надалі тільки він буде мати дозвіл на рівні контракту виконувати виклики методів, що змінюють поведінку системи (оновлювати підтримувані токені та знімати накопичені кошти). Дане рішення забезпечує повний контроль над системою, адже дозвіл та приватні ключі залишаються у продавця.

2.4.2 Створення заявки на оплату

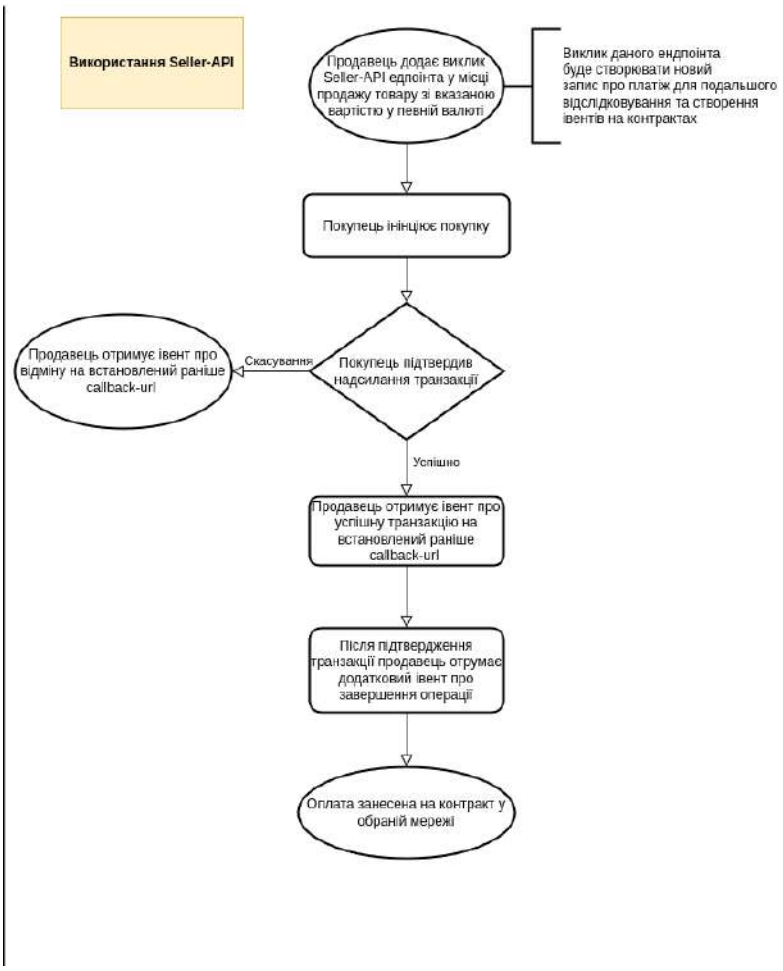

Рис. 2.4.2.1. User-flow створення заявки на оплату

Продавець інтегрує виклик ендпоінту для створення заявки на оплату у свій існуючий застосунок. Після успішного створення продавець надає URL з вбудованим ідентифікатором заявки покупцеві. Після переходу покупець взаємодіє з графічним web інтерфейсом, де виконує оплату. Продавець отримає сповіщення про дію покупця, після завершення взаємодії останнього. За умови успішного проведення оплати, продавець додатково отримає сповіщення про статус проведення транзакції (on-chain). На основі отриманих даних у сповіщеннях продавець коригує роботу свого застосунку для завершення внутрішньої своєї транзакції.

2.4.3 Проведення оплати

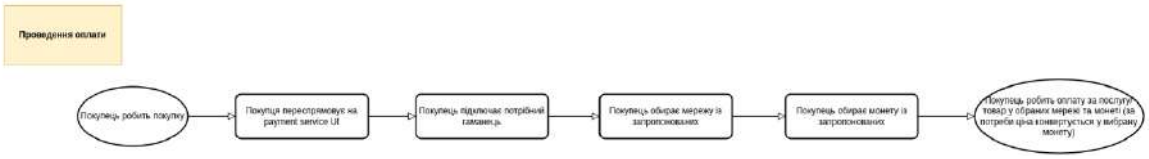

Рис. 2.4.3.1. User-flow проведення оплати покупцем

Покупець після переспрямування на оплату певної заявки має можливість обрати бажану мережу та токен для оплати. Допоки заявка є дійсною, покупець може обирати різні допустимі варіації для оплати, та має ініціювати транзакціє до завершення відведеного часу. Після успішної оплати чи відмови його буде переспрямовано до іншого ресурсу, який вказав продавець під час створення заявки на оплату.

2.4.4 Вивід коштів

Рис. 2.4.4.1. User-flow виводу накопичених коштів з контракту

Продавець може ініціювати транзакцію виводу коштів на обраних мережі та токені на зазначену адресу через графічний web інтерфейс. Варто зазначити, що лише власник контракту - продавець - користувач з адресою ініціалізації розгортання має доступ до цього методу контракту.

2.4.5 Зміна підтримуваних токенів

Рис. 2.4.5.1. User-flow оновлення підтримуваних токенів

Для виконання цих дій, продавцеві необхідно ініціювати транзакцію, в якій має бути зазначено яким токенам потрібно дозволити обробку, або видалити її. Як і з іншими діями, що впливають на поведінку системи, ці дії дозволені тільки власнику контракту.

2.5 Побудова алгоритму захисту від підробки параметрів

Для забезпечення більшої безпеки проведення платежів та додатково захисту власника контракту від можливої компрометації його приватного ключа було розроблено алгоритм, що реалізує механізм двох-факторного підтвердження для транзакцій типу deposit та withdraw.

Суть алгоритму полягає у тому, що усі параметри, що передаються на виклик методу контракту мають бути додатково згруповані та підписані незалежним автоматизованим підписантом, який надається засобами платіжної системи. На рівні контракту перед виконанням переказу токену відбувається перевірка відповідності долученого підпису шляхом повторного групування параметрів та використання функції підтвердження сумісності групи підпис-повідомлення-публічний ключ підписанта за допомогою ECDSA. Дане рішення застосовується до усіх методів, змінюють баланс токенів. На рисунку 2.6.1. зображено схематичне представлення роботи алгоритму створення підпису.

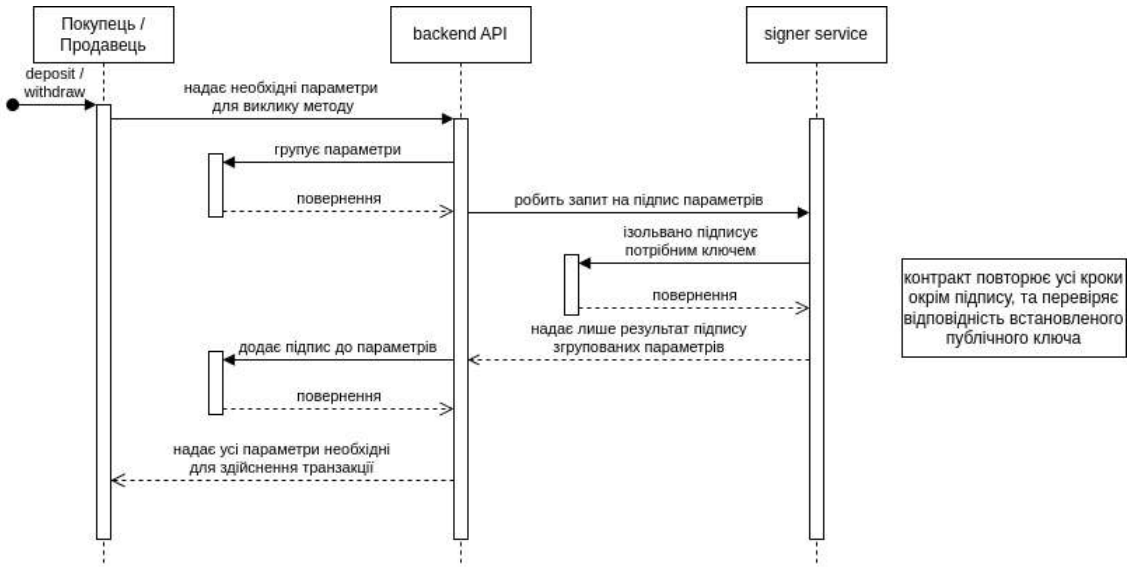

Рис. 2.5.1 Алгоритм створення захисного підпису

Незмінність та стійкість забезпечується властивостями ECDSA.

Також варто зазначити, що окрім публічних параметрів виклику метода, у повідомлення для підпису також докладаються внутрішні on-chain параметри: адреса відправника транзакції, ідентифікатор мережі, адреса контракту процесора. Це забезпечує успішне виконання транзакції тільки за незмінності вхідних умов.

Визначимо параметри, які мають бути включені до підпису для deposit методу:

- адреса відправника транзакції;
- адреса контракту виконання;
- адреса токена;
- розмір депозиту;
- відмітка часу дійсності підпису;

- ідентифікатор заявки на оплату;
- ідентифікатор мережі.

Також за аналогією визначимо параметри для withdraw методу:

- адреса відправника транзакції;
- адреса контракту виконання;
- адреса отримувача виплати;
- адреса токена;
- розмір виплати;
- відмітка часу дійсності підпису;
- ідентифікатор мережі.

Додатковим методом захисту для покупця є властивість контрактів ERC20 – allowance, що дозволяє стягнути з користувача обмежену кількість токенів. У даній реалізації це значення буде завжди рівне розміру обрахованої оплати.

2.6 Висновки

У цьому розділі було сформовано технічні вимоги до платіжної системи, визначено основні ролі користувачів та їхні можливості. Проаналізовано специфікації вимог до даних, та обмежень.

Розроблено user-flows для основних операції, які має надавати платформа. Описано ключові етапи та очікувану поведінку.

Розроблено основний механізм захисту від підробки та перевикористання параметрів, що забезпечує додатковий рівень безпеки під час проведення транзакцій та захист від шахрайський дій.

РОЗДІЛ 3. Розробка застосунку та тестування

3.1 Опис архітектури

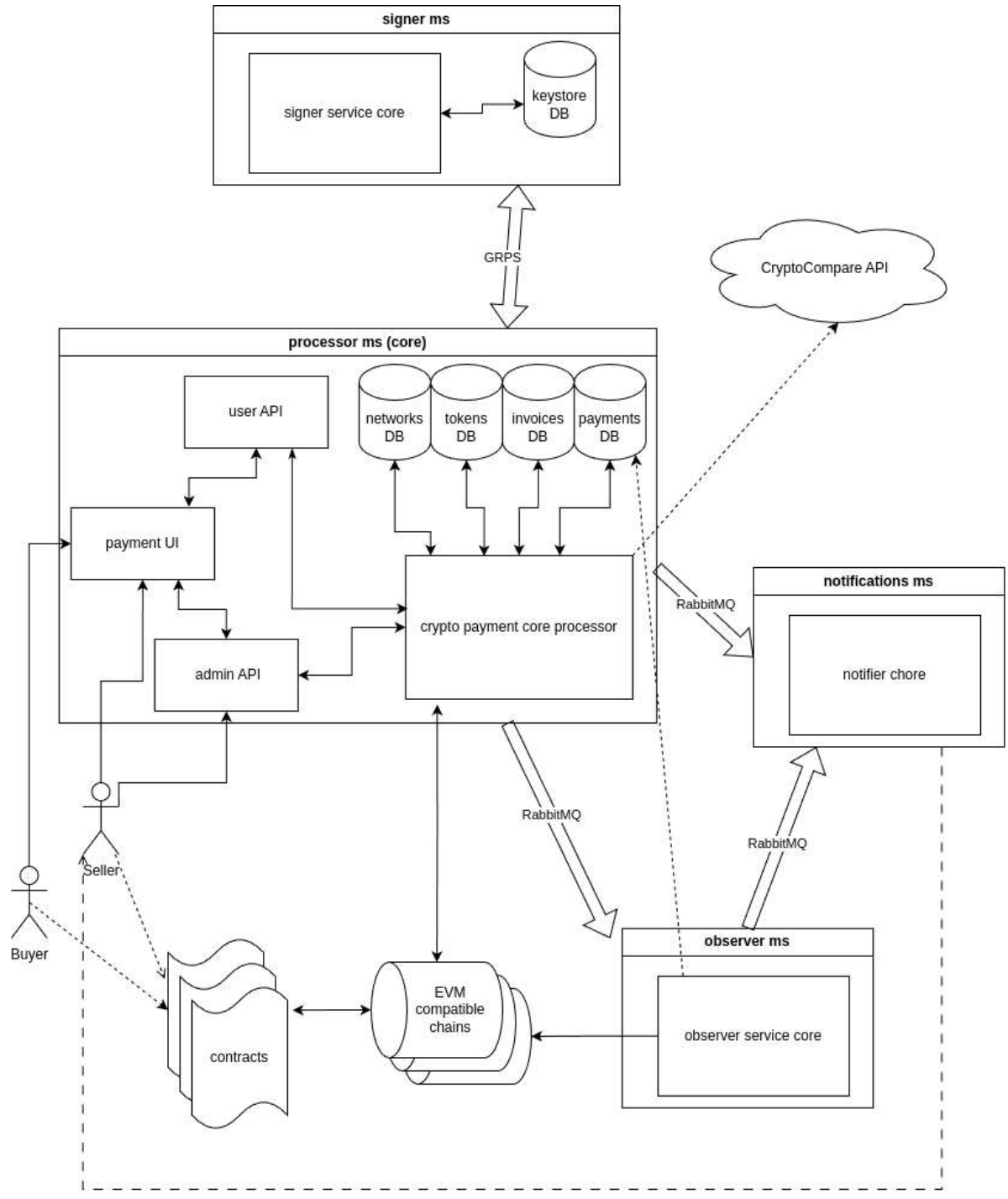

Рис. 3.1.1. Загальна архітектура застосунку

Для реалізації платіжної систему було використано мікросервісну архітектуру з використанням різних протоколів комунікації між сервісами. Дане рішення дозволяє легко проводити масштабування за потреби та ізолює логічні одиниці за степенем зв’язаності та поставленої задачі.

Загалом робота всієї системи зводиться до послідовності звернень сервісів один до одного у різних комбінаціях, оновленні стану бази даних, взаємодії з контрактами та створенні записів у блокчейни. Розглянемо детальніше яка відповідальність у кожного сервісу.

3.1.1 Processor microservice (core)

Для кінцевих користувачів взаємодія з платформою відбувається на найвищому рівні через API напряму або з використання графічного web інтерфейсу, як посередника, для автоматизації деяких дій для облегшення користувацького досвіду. Далі запити після початкової валідації перенаправляються на центральний сервіс - core payment processor.

Цей мікросервіс побудовано з використанням класичного підходу Separation of concerns та тришарового розподілу на представлення, бізнес логіку та доступ до даних [12].

Основними задачами цього мікросервісу є:

- хостинг сервера, що надає доступ до двох рівнів API та графічних web інтерфейсів для покупців та продавця;
- обробка запитів, отриманих від покупців під час створення платежів;
- оновлення стану основної бази даних;
- доповнення черги перевірки транзакцій;
- доповнення черги сповіщення;
- конвертація цін;
- побудова та підготовка параметрів транзакцій.

3.1.2 Signer microservice

Signer - це компактний сервіс, основна задача якого - інкапсулювати всю логіку роботи з приватними ключами та надати компактний інтерфейс для підписання параметрів та отримання публічних даних. У ньому реалізований алгоритм підпису ECDSA, що потрібний для роботи алгоритму забезпечення захисту від підробки параметрів.

Зв'язок з сервісом платежів реалізований через протокол GRPC - сучасний високопродуктивний фреймворк з відкритим кодом для віддаленого виклику процедур [13].

Дане рішення було обране для забезпечення швидкої синхронної роботи.

3.1.3 Observer microservice

Задача цього мікросервісу - слідкувати за перебігом виконання транзакцій, пов’язаних з платіжками, оновлювати базу даних (статуси платежів) та після змін сигналізувати про це через сервіс сповіщень. Для комунікації було обрано використання черги RabbitMQ для асинхронної обробки транзакцій та можливості повторного опрацювання у випадку неочікуваних помилок.

RabbitMQ надає готове рішення для роботи з чергою, має велику кількість SDK та легкий у інтеграції. Засобами цього рішення можна швидко створити диспетчерів та споживачів, що працюватимуть на власних чергах для організації асинхронної обробки [14].

3.1.4 Notifications microservice

Сервіс сповіщень займається надсиланням повідомлень (HTTP запитів) на встановлену продавцем URL адресу для сигналізації статусу перебігу транзакцій. Як і сервіс observer, на рівні комунікації використовує RabbitMQ для асинхронної роботи.

3.2 Використані інструменти та засоби розробки

3.2.1 Golang

Основною мовою програмування на бекенді було використано Golang. Вона є високопродуктивною та зручною для побудови високонавантажених розподілених систем. Хоча вона не підтримує об’єктно-орієнтовану парадигму, вона володіє потужною системою роботи з абстракціями (інтерфейсами), що дає змогу працювати зі звичними патернами навіть у процедурному стилі [15].

Також великою перевагою є наявність великої кількості готових рішень, SDK, клієнтів, фреймворків, інтеграцій та підтримки даної мови у блокчейн екосистемах. Одним із найбільш використовуваних фреймворків у даній роботі є go-ethereum, що надає повний доступ до взаємодії з блокчейном та його складовими.

3.2.2 PostgreSQL

Для зберігання та обробки даних було вирішено використовувати СУБД PostgreSQL, через її підтримку ACID транзакційності, високу продуктивність, велику кількість механізмів оптимізації та обширну документацію [16].

У всіх мікросервісах на рівні взаємодії з базою даних було реалізовано інтерфейс Database (DB), що описує поведінку абстрактного сховища. У поточній реалізації платіжної системи усі імплементації працюють з реляційною базою даних та PostgreSQL, проте такий підхід дає можливість за потреби реалізувати використання іншого типу сховища без зміни у роботі застосунку.

```
// DB provides access to Invoices repository methods.
type DB interface {  1 implementation   Volodymyr Smetaniuk
    // TransactionBeginner implementation to support transactionality.
    transactionality.TransactionBeginner[DB]

    // Create inserts new Invoice into the repository.
    Create(ctx context.Context, invoice *Invoice) error  1 implementation
    // Get returns Invoice by id  from the repository.
    Get(ctx context.Context, id uuid.UUID) (*Invoice, error)  1 implementation
    // List returns page of Invoices from the repository.
    List(ctx context.Context, limit, page int) ([]*Invoice, error)  1 implementation
    // Update updates Invoice in the repository.
    Update(ctx context.Context, invoice *Invoice) error  1 implementation
    // Delete removes Invoice by id from the repository.
    Delete(ctx context.Context, id uuid.UUID) error  1 implementation
}
```

Рис. 3.2.2.1. Приклад DB інтерфейсу

Також, використовуючи реляційну базу даних, було розроблено модель відношення сутностей для сервісу платежів (рис. 3.2.2.2).

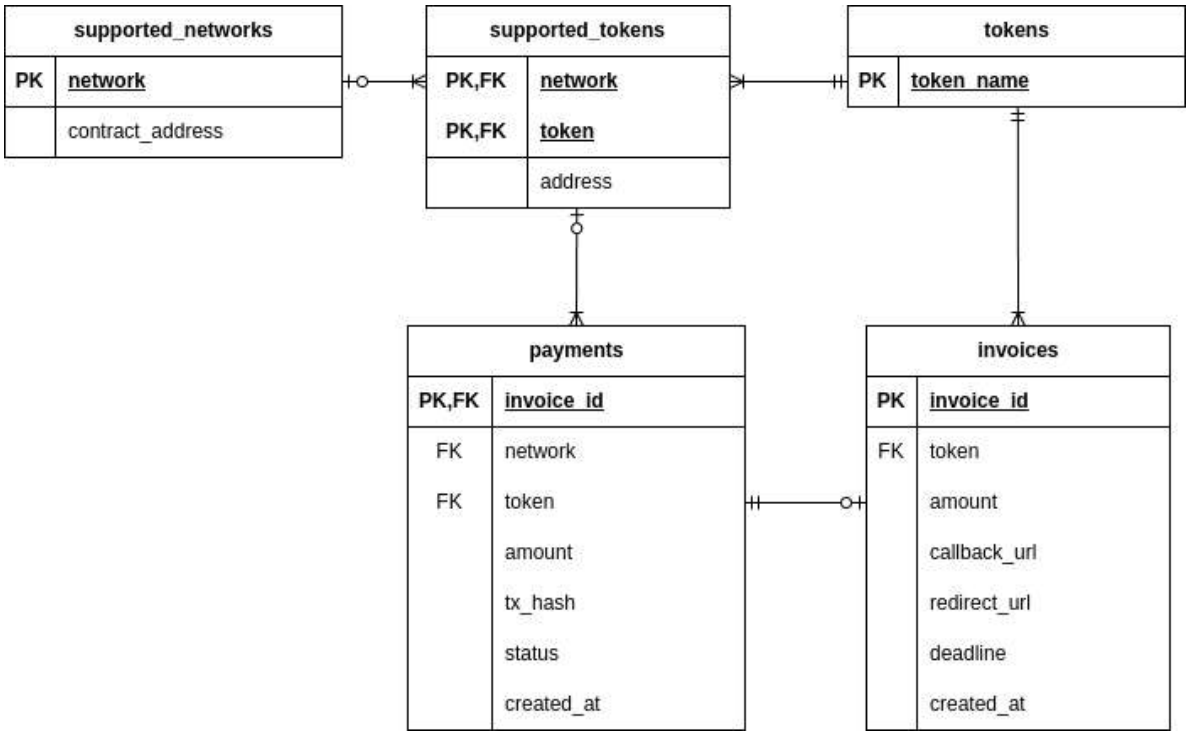

Рис. 3.2.2.2. ER-модель бази даних

Тут supported_networks описує підтримувані мережі, tokens описує можливі назви токенів, supported_tokens визначає яка монета підтримується на якому блокчейні. Invoices описує заявку на оплату від продавця, а payments - фактичну оплату.

3.2.3 CryptoCompare API

Для конвертації суми платежів було використано сервіс CryptoCompare. Він підтримує велику кількість криптовалют та надає можливість використовувати безкоштовно API до певної кількості запитів.

3.2.4 Solidity та OpenZeppelin

Мовою програмування смарт контрактів для EVM-сумісних мереж було обрано Solidity. Також ключовим фактором стала наявність великої кількості готових рішень та стандартів, які надаються OpenZeppelin. Одним із них є використаний у застосунку контракт ERC20.

3.2.5 Solc, abigen, hardhat

Для компіляції контрактів у бінарний формат та доступу до ABI (application binary interface) було використано компілятор solc. З отриманих bin та abi файлів генератором abigen було згенеровано клієнти мовою Golang для взаємодії з контрактами на блокчейні. Такий підхід пришвидшує розробку та гарантує найкращу сумісність та правильність взаємодії з контрактом.

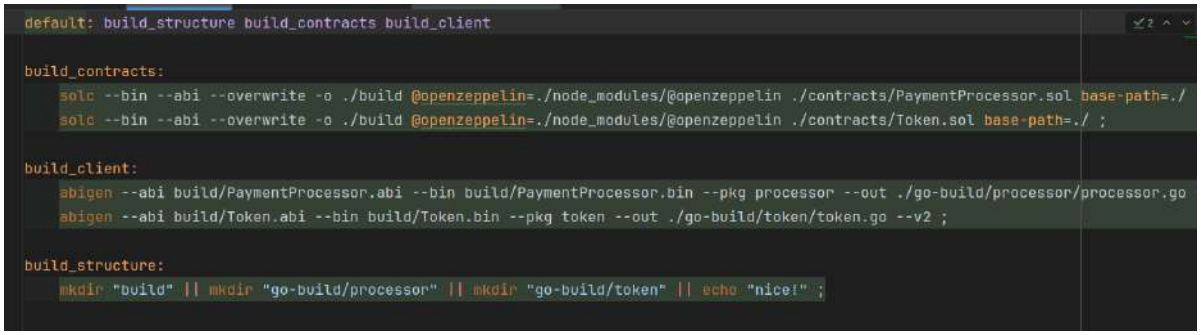

Рис. 3.2.5.1. Makefile з використанням програм solc та abigen

Для тестування було використано фреймворк hardhat. Він надає можливість тестувати поведінку контрактів на локально розгорнутій мережі для симуляції реалістичних умов [17]. Це стало ключовим фактором для вибору даного фреймворку для написання тестів перевірки правильності роботи механізму захисту від перевикористання параметрів.

3.2.6 Docker

Для контейнеризації застосунку для легкого транспортування та забезпечення однакової поведінки на різних системах та версіях було використано інструменти Docker та Docker Compose. Такий підхід дозволяє запуском однієї команди скомпілювати, розгорнути та запустити усі необхідні елементи для роботи системи, а також налаштувати внутрішню мережу для встановлення комунікації між сервісами.

3.3 Розробка смарт контракту

Усі операції переведення коштів та встановлення обмежень реалізовані у контракті PaymentProcessor. Він є проміжним елементом на який надходять кошти під час проведення оплати покупцем. На ньому акумулюються кошти від платежів, і доступ до виводу є тільки у власника контракту. Додатково методи deposit та withdraw захищені підписом з алгоритму, розробленого у даній роботі.

Розглянемо структуру контракту:

```
contract PaymentProcessor is Ownable {
    address private _signerAddress;

    mapping(address => bool) private _supportedTokens;

    event Deposit(address indexed depositor, address indexed token, string indexed invoiceId, uint256 amount);

    event Withdrawal(address indexed recipient, address indexed token, uint256 amount);

    /// @notice Deposit tokens to the payment processor
    function deposit(address token, uint256 amount, uint256 deadline, string calldata invoiceId, bytes calldata signature) external;

    /// @notice Withdraw tokens from the payment processor to the owner
    function withdraw(address recipient, address token, uint256 amount, uint256 deadline, bytes calldata signature) public onlyOwner;

    /// @notice Adds supported tokens
    function addSupportedTokens(address[] calldata tokens) public onlyOwner;

    /// @notice Removes supported tokens
    function removeSupportedTokens(address[] calldata tokens) public onlyOwner;
}
```

Рис. 3.3.1. Оголошення основних методів контракту PaymentProcessor

Як можна побачити, після розгортання, контракт міститиме адресу (_signerAddress) згенерованого підписанта параметрів для додаткового захисту платіжних методів. Також для цих операцій буде виконуватись перевірка чи дозволено використовувати токен на цьому контракті через пошук у мапінгу адрес (_supportedTokens).

3.3.1 Реалізація алгоритму захисту від підробки параметрів на контракті

```
/// @notice Verifies the signature of a given hash
/// @param signerAddress Address of the signer
/// @param hash Hash of the message to be verified
/// @param signature Signature to be verified
/// @return bool True if the signature is valid, false otherwise
function _verify(
    address signerAddress,
    bytes32 hash,
    bytes calldata signature
) private pure returns (bool) {
    return signerAddress == ECDSA.recover(hash: hash, signature: signature);
}
```

Рис. 3.3.1.1. Реалізація основної функції відновлення адреси підписанта з підпису та хеша повідомлення

На рисунку 3.3.1.1 реалізована перевірка відповідності встановленої адреси підписанта на контракті із відновленою з отриманих параметрів. Вона використовується у методах deposit та withdraw після отримання хешу зі згрупованих параметрів.

```
function deposit(
    address token,
    uint256 amount,
    uint256 deadline,
    string calldata invoiceId,
    bytes calldata signature
) external {
    if (block.timestamp > deadline) {revert(Errors.EXPIRED_SIGNATURE);}

    if (!_supportedTokens[token]) {revert(Errors.UNSUPPORTED_TOKEN);}

    if (!_verify(
        signerAddress: _signerAddress,
        hash: MessageHashUtils.toEthSignedMessageHash(
            keccak256( input: abi.encodePacked(
                <unknown>: _msgSender(), // senderAddress
                address(this), // contractAddress
                token,
                amount,
                deadline,
                invoiceId,
                getChainId()
            ))
        ),
        signature: signature
    )) {revert(Errors.INVALID_SIGNATURE);}

    IERC20( IERC20: token).safeTransferFrom( from: _msgSender(),  to: address(this),  value: amount);

    emit Deposit( depositor: _msgSender(),  token: token,  invoiceId: invoiceId,  amount: amount);
}
```

Рис. 3.3.1.2. Реалізація методу deposit з механізмом захисту параметрів

Перед перевіркою підпису, відбувається валідація часу виконання транзакції та допустимості токена. Після цього усі вхідні параметри, окрім підпису, групуються функцією abi.encodePacked, яка конкатенує усі параметри у байтовому представленні. З отриманого повідомлення береться хеш через функцію keccak256, що потім зводиться до стандартного формату підпису повідомлення. Тільки за умови правильності підпису виконається переведення токену від покупця до контракту продавця.

```
function withdraw(
    address recipient,
    address token,
    uint256 amount,
    uint256 deadline,
    bytes calldata signature
) public onlyOwner {
    if (block.timestamp > deadline) {
        revert(Errors.EXPIRED_SIGNATURE);
    }

    if (!_verify(
        signerAddress: _signerAddress,
        hash: MessageHashUtils.toEthSignedMessageHash(
            keccak256( input: abi.encodePacked(
                <unknown>: _msgSender(), // senderAddress
                address(this), // contractAddress
                recipient,
                token,
                amount,
                deadline,
                getChainId()
            ))
        ),
        signature: signature
    )) {revert(Errors.INVALID_SIGNATURE);}

    IERC20( IERC20: token).transfer( to: recipient,  value: amount);

    emit Withdrawal( recipient: _msgSender(),  token: token,  amount: amount);
}
```

Рис. 3.3.1.3. Реалізація методу withdraw з механізмом захисту параметрів

У даному методі використовується аналогічний підхід із методом deposit. Відрізняється лише кількість і типи параметрів, та відсутність перевірки дозволу проведення операції з токеном. Таке було рішення прийнято для можливості власнику виводити токен, який він попередньо міг “відключити”.

3.3.2 Реалізація алгоритму захисту від підробки параметрів на бекенді

На стороні бекенда було реалізовано encodePacked, яка копією роботу функції abi.encodePacked на контракті для консистентної побудови повідомлення для підпису.

```go
func encodePacked(values ...any) ([]byte, error) {  5 usages  Volodymyr Smetaniuk
    res := make([]byte, 0)
    for _, value := range values {
        switch v := value.(type) {
        case []byte: res = append(res, v...)
        case *big.Int: res = append(res, math.U256Bytes(new(big.Int).Set(v))...)
        case string: res = append(res, []byte(v)...)
        case common.Address: res = append(res, v.Bytes()...)
        default: return nil, Error.New("unsupported type for encodePacked")
        }
    }

    return res, nil
}
```

Рис. 3.3.2.1. Реалізація функції encodePacked на бекенді

```go
    signPayload, err := encodePacked(
        common.HexToAddress(sender),
        common.HexToAddress(contract),
        recipient_,
        token_,
        amount_,
        deadline_,
        chainID,
    )
    if err != nil {
        c.log.Error("failed to get encodePacked payload", Error.Wrap(err))
        return nil, Error.Wrap(err)
    }

    signResp, err := c.signer.Sign(ctx, signerclient.SignRequest{
        Network:  c.config.Network,
        Data:     prepareMessageToSign(crypto.Keccak256(signPayload)),
        DataType: keystore.TypeSignature,
    })
```

Рис. 3.3.2.2. Реалізація алгоритму створення підпису параметрів на бекенді для методу withdraw

Для методу deposit виконуються аналогічні операції, але з іншими параметрами.

3.4 Тестування

Тестування застосунку було проведено у два підходи:

- автоматичне тестування:
  - покриття юніт-тестами імплементацій рівні бази даних;
  - покриття юніт-тестами методів контракту з використанням hardhat;
  - покриття юніт-тестами утилітарних функцій та методів.
- ручне тестування:
  - виклики та аналіз роботи API за допомогою інструменту Postman
  - тестування роботи під час інтеграції web інтерфейсу

```
t.Run("Seed", func(t *testing.T) {
    require.NoError(t, networksRepository.Create(ctx, &network1))
    require.NoError(t, tokensRepository.Create(ctx, &token11))
    require.NoError(t, tokensRepository.Create(ctx, &token12))
})

t.Run("Create", func(t *testing.T) {
    require.NoError(t, invoicesRepository.Create(ctx, &invoice1))
    require.NoError(t, invoicesRepository.Create(ctx, &invoice2))
    require.Error(t, invoicesRepository.Create(ctx, &invoice1))
    require.Error(t, invoicesRepository.Create(ctx, &invoice2))
})

t.Run("Get", func(t *testing.T) {
    invoice, err := invoicesRepository.Get(ctx, invoice1.InvoiceID)
    require.NoError(t, err)
    requireEqualInvoices(t, invoice1, *invoice)

    invoice, err = invoicesRepository.Get(ctx, invoice2.InvoiceID)
    require.NoError(t, err)
    requireEqualInvoices(t, invoice2, *invoice)

    _, err = invoicesRepository.Get(ctx, uuid.Nil)
    require.ErrorIs(t, err, invoices.ErrNotFound)
})
```

Рис. 3.4.1. Приклад юніт-тестів реалізації DB інтерфейсу для репозиторія Invoices

```
it( title: 'should not allow deposits with not added token', fn: async function () : Promise<void> {
    const amount : bigint = tokenAmount( amount: '100', units: 6);
    const deadline : number = Math.floor( x: Date.now() / 1000) + 3600;
    const invoiceId : string = 'INV-123';
    const signature : Uint8Array<ArrayBufferLike> = await signMessage(
        depositMethodSignature,
        values: [
            user2.address,
            await paymentProcessor.getAddress(),
            await token2.getAddress(),
            amount,
            deadline,
            invoiceId,
            (await ethers.provider.getNetwork()).chainId
        ],
        signer);

    await expect(token2.connect(user2).approve(await paymentProcessor.getAddress(), amount)).to.not.reverted;
    await expect(
        paymentProcessor.connect(user2).deposit(await token2.getAddress(), amount, deadline, invoiceId, signature)
    ).to.be.revertedWith(Errors.UNSUPPORTED_TOKEN);
});

it( title: 'should add and remove supported tokens by owner', fn: async function () : Promise<void> {
    // Add new token
    await expect(paymentProcessor.addSupportedTokens([await token2.getAddress()]))
        .to.not.be.reverted;

    // Remove token
    await expect(paymentProcessor.removeSupportedTokens([await token2.getAddress()]))
        .to.not.be.reverted;
});
```

Рис. 3.4.2. Приклад юніт-тестів методів контракту

```
func TestMap(t *testing.T) {  Volodymyr Smetaniuk
    t.Run("convert int to string", func(t *testing.T) {
        input := []int{0, 1, 2, 3, 4, 5}
        expected := []string{"a", "b", "c", "d", "e", "f"}
        require.Equal(t, expected, slices.Map(input, func(x int) string { return string('a' + rune(x)) }))
    })

    t.Run("empty slice", func(t *testing.T) {
        input := make([]byte, 0)
        expected := make([]float32, 0)
        require.Equal(t, expected, slices.Map(input, func(_ byte) float32 { return 0.0 }))
    })
}
```

Рис. 3.4.3. Приклад юніт-тестів утилітарних функцій

3.5 Результати роботи

Продемонструємо використання застосунку на тестових мережах Sepolia та Avalanche Fuji C-Chain.

Для початку необхідно додати підтримку мережі. Для цього продавець через свій приватний графічний інтерфейс ініціює транзакцію розгортання контракту.

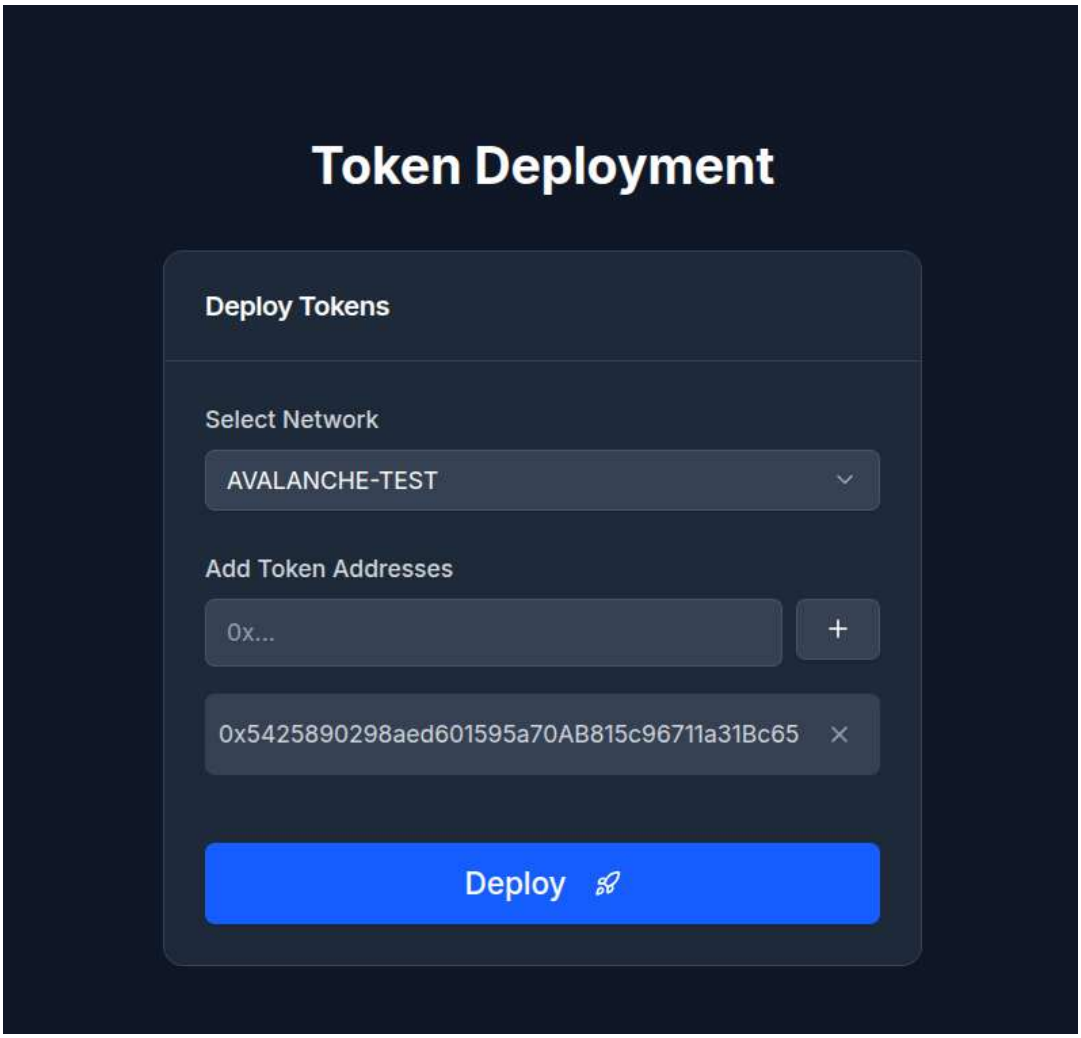

Рис. 3.5.1. Сторінка додавання нової мережі

Після заповнення усіх необхідних параметрів (у цьому випадку нова мережа - Avalanche testnet та один підтримуваний токен - USDC) необхідно натиснути кнопку Deploy та надіслати згенеровану транзакцію.

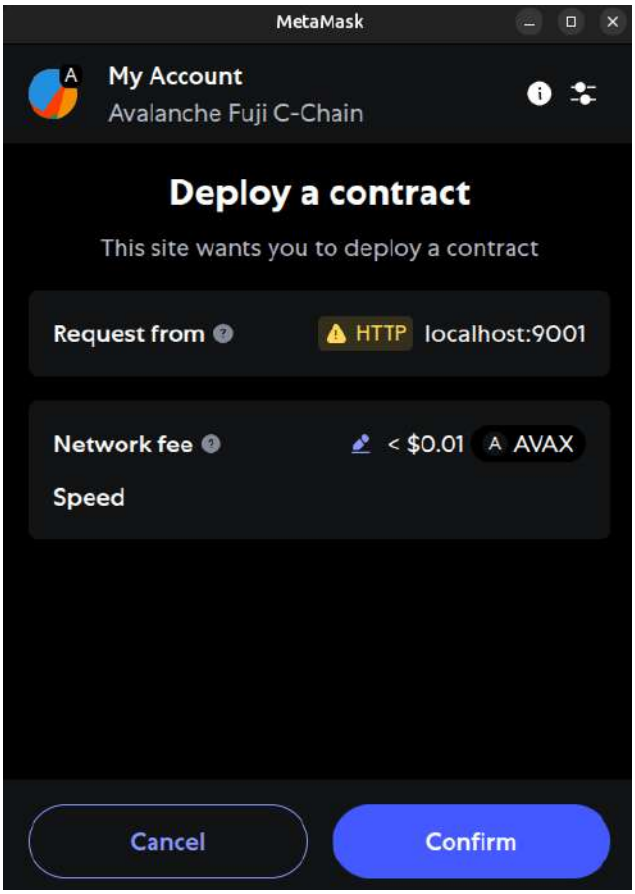

Рис. 3.5.2. Підтвердження транзакції розгортання нового контракту

Після підтвердження транзакції на блокчейні observer сервіс із даних транзакції отримає адресу розгорнутого контракту та оновить базу платформи цими значеннями.

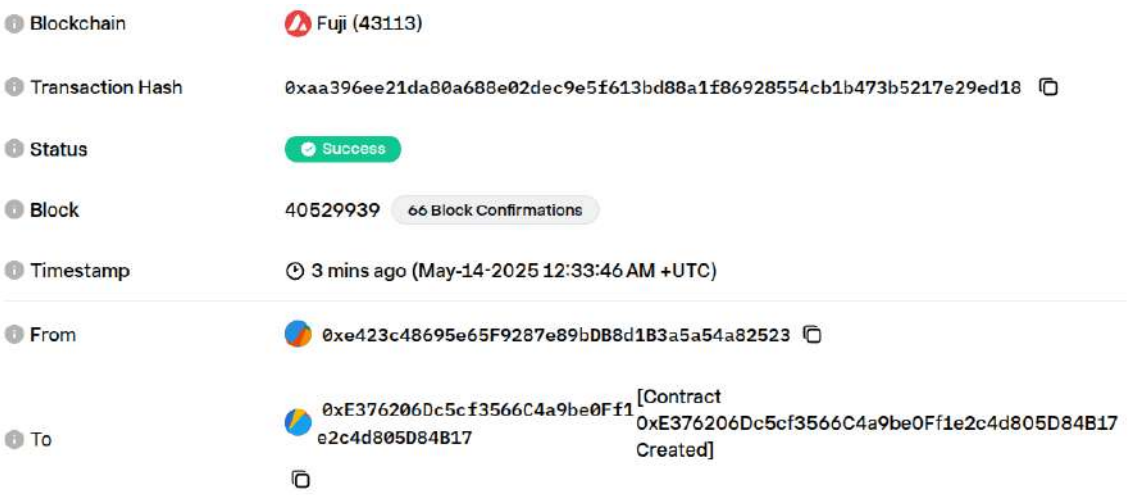

Рис .3.5.3. Успішна транзакція розгортання контракту

Рис. 3.5.4. Observer сервіс отримав запит на відслідковування транзакції та опрацював її дані

Тепер можна просимулювати роботу інтегрованого запиту на створення заявки на оплату використовуючи сервіс Postman. Для перевірки роботи сервісу сповіщень скористаємось сервісом pipedream.

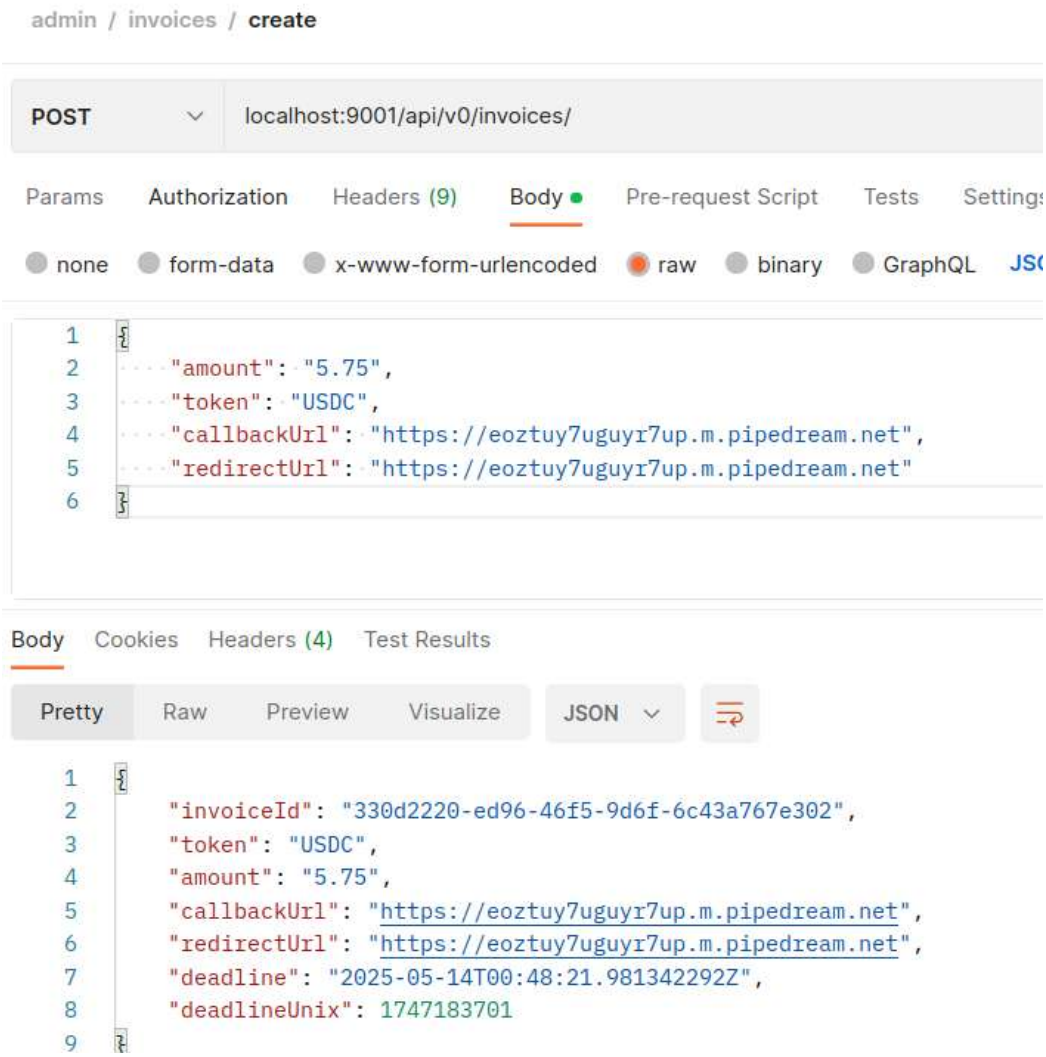

Рис. 3.5.5. Успішне створення заявки на оплату

Тепер можемо просимулювати переадресацію покупця на сторінку оплати з ідентифікатором цієї заявки.

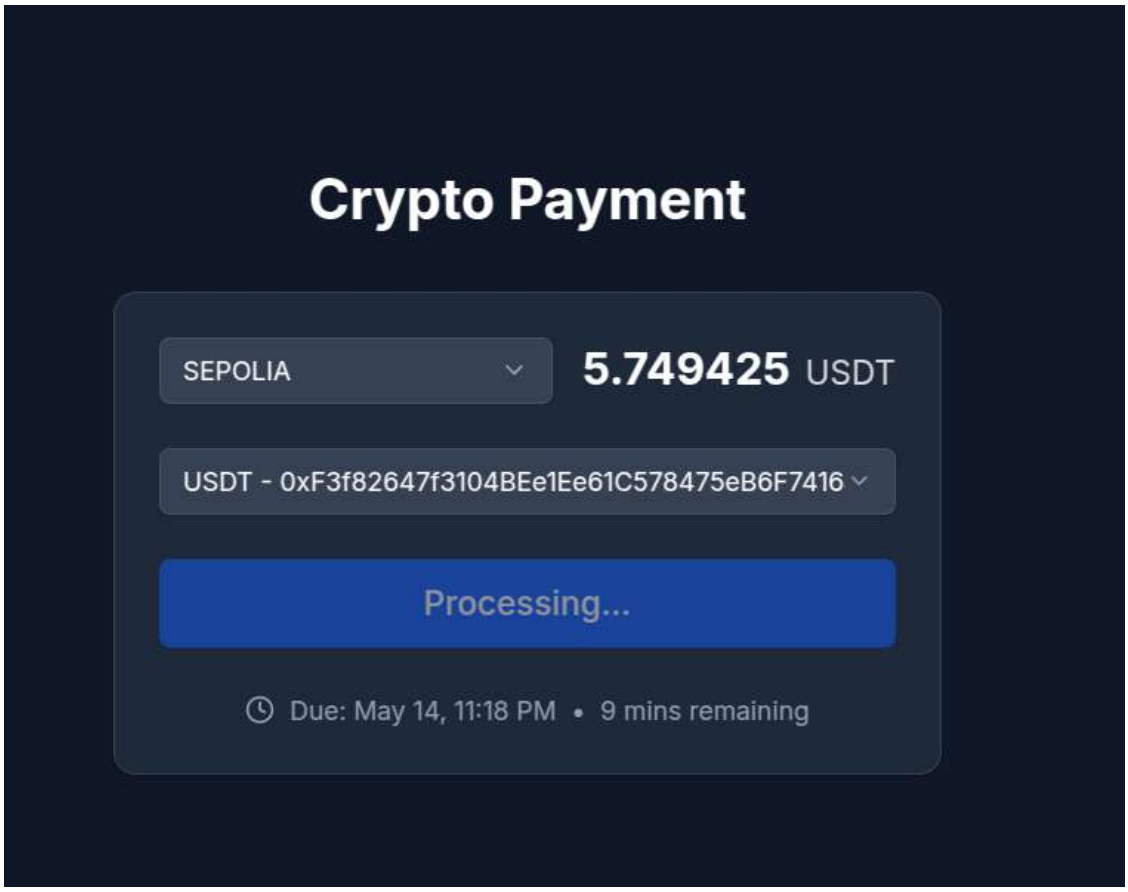

Рис. 3.5.6. Сторінка платежу з обраними іншою мережею та токеном

У даному випадку можна помітити, що відбулась конвертація USDC в USDT. Курси цих токенів майже однакові, тому сума оплати відмінна. Головне - ми можемо обрати інший токен, та будь-яку із доступних мереж. Проведемо оплату в мережі Sepolia з токеном USDT.

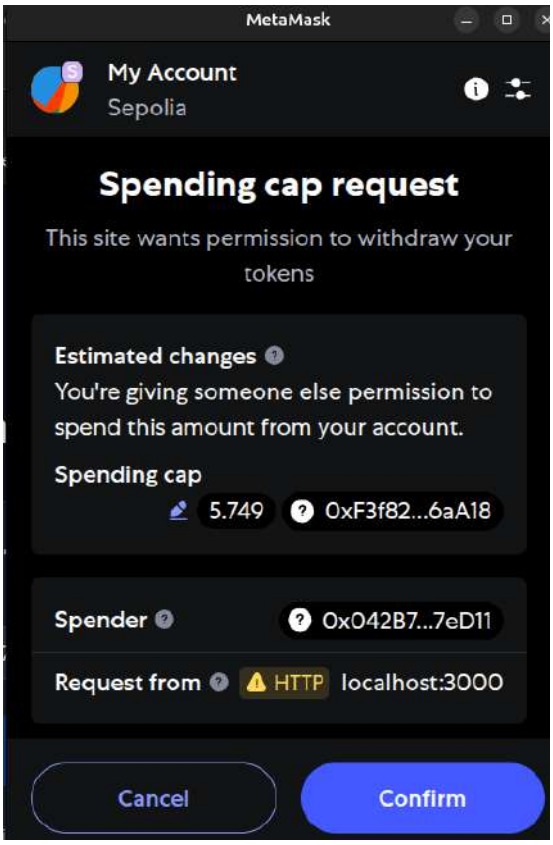

Рис. 3.5.7. Approve транзакція перед deposit транзакцією

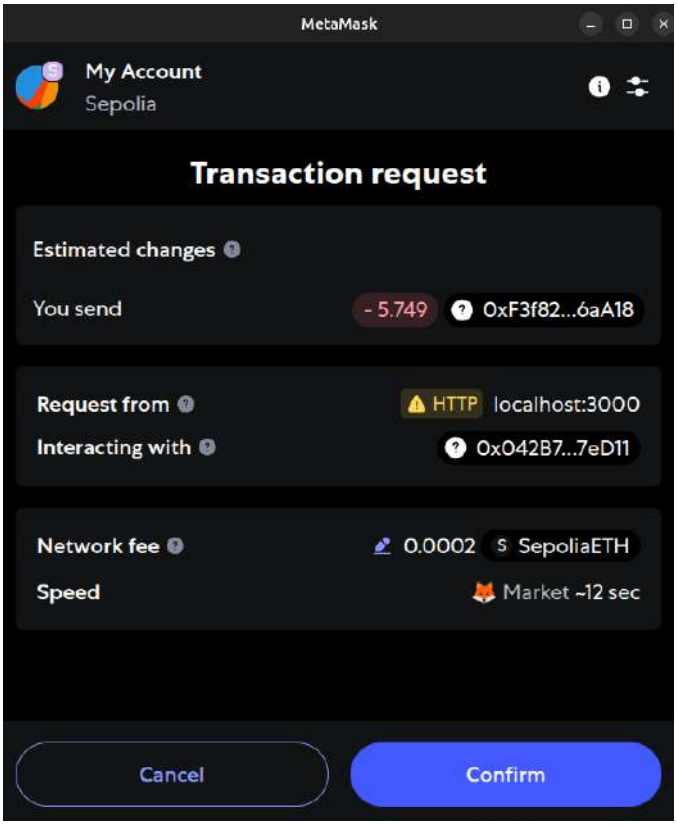

Рис. 3.5.8. Deposit транзакція

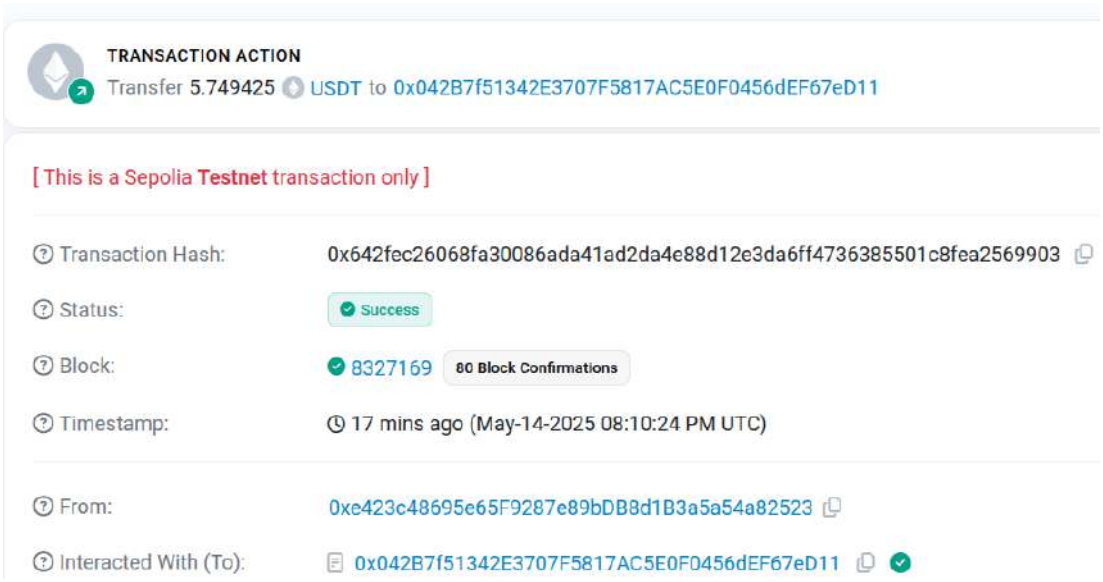

Рис. 3.5.9. Успішно проведена deposit транзакція

Рис. 3.5.10. Результат успішного відпрацювання сервісу сповіщень

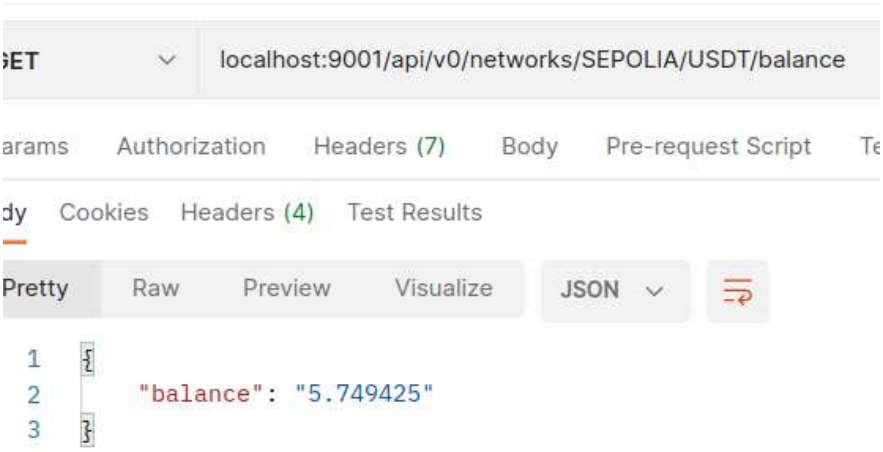

Рис. 3.5.11. Результат запиту перевірки балансу контракту з приватного API продавця після успішного депозиту покупця

На даному етапі можна продемонструвати функцію withdraw для продавця, щоб вивести зароблені кошти. Для цього продавець через приватний графічний інтерфейс має ініціювати транзакцію виводу. API payment сервісу виконає усі необхідні дії для підготовки параметрів, в тому числі і створення підпису.

Рис. 3.5.12. Сторінка виводу токенів з контракту

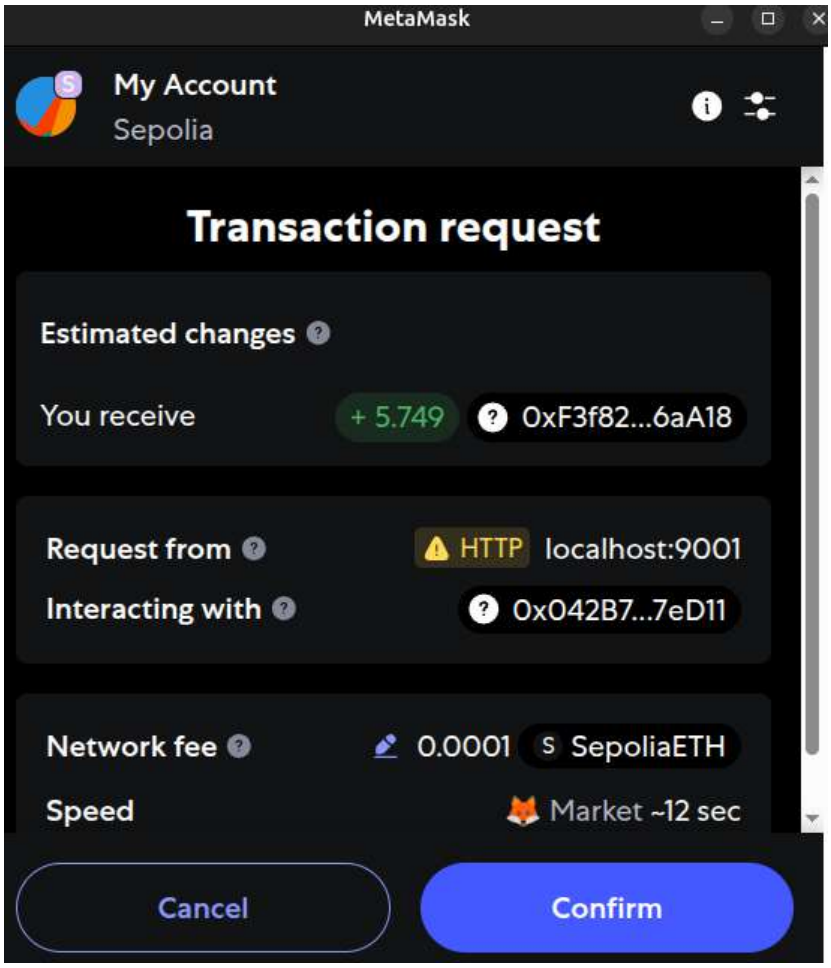

Рис. 3.5.13. Withdraw транзакція перед підписом у гаманці

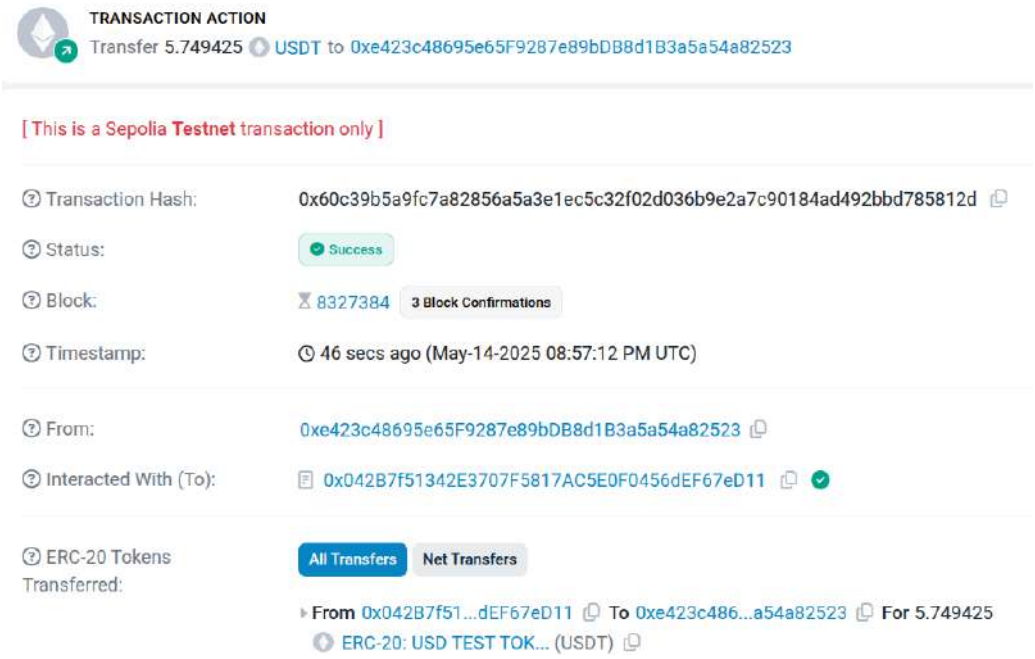

Рис. 3.5.16. Успішно проведена withdraw транзакція на блокчейні

3.6 Висновки

У даному розділі було розглянуто архітектуру платформи та реалізацію її складових. Наведено використані технології та підходи, продемонстровано ключові рішення.

Було реалізовано алгоритм захисту від повторного використання параметрів на контракті та бекенді, наведено стратегії тестування, та приклади тестів з коду застосунку.

Було розглянуто повний цикл роботи платіжної платформи, проведено deploy, deposit та withdraw операції, продемонстровано коректність роботи сервісів та графічної частини.

Список використаної літератури

1. What is Stablecoin?: A Survey on Its Mechanism and Potential as Decentralized Payment Systems / K. Ito та ін. International Journal of Service and Knowledge Management. 2020. Т. 4, № 2. С. 71–86. URL: https://doi.org/10.52731/ijskm.v4.i2.574.
2. Chen D. B. Central Banks and Blockchains. Transforming Climate Finance and Green Investment with Blockchains. 2018. С. 201–216. URL: https://doi.org/10.1016/b978-0-12-814447-3.00015-x.
3. An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends / Z. Zheng та ін. 2017 IEEE International Congress on Big Data (BigData Congress), м. Honolulu, HI, USA, 25–30 черв. 2017 р. 2017. URL: https://doi.org/10.1109/bigdatacongress.2017.85.
4. Di Pierro M. What Is the Blockchain?. Computing in Science & Engineering. 2017. Т. 19, № 5. С. 92–95. URL: https://doi.org/10.1109/mcse.2017.3421554.
5. NIST C. The digital signature standard. Communications of the ACM. 1992. Т. 35, № 7. С. 36–40. URL: https://doi.org/10.1145/129902.129904.
6. Network Security and Cryptography / Pradeep Nayak та ін. International Journal of Advanced Research in Science, Communication and Technology. 2024. С. 185–187. URL: https://doi.org/10.48175/ijarsct-22829.
7. Johnson D., Menezes A., Vanstone S. The Elliptic Curve Digital Signature Algorithm (ECDSA). International Journal of Information Security. 2001. Т. 1, № 1. С. 36–63. URL: https://doi.org/10.1007/s102070100002.
8. Globale Zahlungslösungen für Web3 | Stripe. Online-Bezahldienst und Zahlungsdienstleister | Stripe. URL: https://stripe.com/use-cases/crypto.
9. Merchant Payment Gateway. CoinGate Cryptocurrency Payment API. URL: https://developer.coingate.com/reference/cryptocurrency-payment-api.
10. Development API | Payments API for your business | NOWPayments. NOWPayments. URL: https://nowpayments.io/api.
11. Kuzin E. Crypto Payment Gateway | Cryptocurrency & Blockchain Transactions. CoinsPaid. URL: https://coinspaid.com/crypto-payment-gateway/.
12. Saleh M., Gomaa H. Separation of concerns in software product line engineering. ACM SIGSOFT Software Engineering Notes. 2005. Т. 30, № 4. С. 1–5. URL: https://doi.org/10.1145/1082983.1083139.
13. gRPC. gRPC. URL: https://grpc.io/.
14. RabbitMQ Documentation | RabbitMQ. RabbitMQ: One broker to queue them all | RabbitMQ. URL: https://www.rabbitmq.com/docs.

15. Documentation - The Go Programming Language. The Go Programming Language. URL: https://go.dev/doc/.
16. Worsley J. C., Drake J. D. Practical PostgreSQL. O'Reilly Media, Incorporated, 2002.
17. Hardhat | Ethereum development environment for professionals by Nomic Foundation. Hardhat | Ethereum development environment for professionals by Nomic Foundation. URL: https://hardhat.org/.