Вінницький національний технічний університет
(повне найменування вищого навчального закладу)
Факультет інформаційних технологій та комп'ютерної інженерії
(назва факультету (відділення))
Кафедра програмного забезпечення
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«УДОСКОНАЛЕННЯ МЕТОДІВ ТА ЗАСОБІВ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОБРОБКИ КРИПТОВАЛЮТНИХ ТРАНЗАКЦІЙ»

Виконав: студент 2-го курсу, групи 2ПІ-22м
спеціальності 121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Мельник І.С.
(прізвище та ініціали)
Керівник: к.т.н., доцент кафедри ПЗ, Коваленко О.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
«15» грудня 2023 р.
Опонент: к.т.н. професор кафедри ЗІ, Кондратенко Н.Р.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
«15» грудня 2023 р.

Допущено до захисту
Завідувач кафедрою ПЗ
д.т.н., проф., Романюк О. Н.
«15» грудня 2023 р.

ВНТУ – 2023

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти другий (магістерський)
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
«19» вересня 2023 р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Мельник Ілля Сергійович

1. Тема роботи – «Удосконалення методів та засобів розробки програмного забезпечення для обробки криптовалютних транзакцій».

Керівник роботи: Коваленко Олена Олексіївна, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від «18» вересня 2023 р. № 247.

2. Строк подання студентом роботи 5 грудня 2023 р.

3. Вихідні дані до роботи: Процеси життевого циклу криптовалютні транзакції, принципи побудови централізованої архутектури, обробка даних в умовах багатопотоковості, взаємодія мікросервісів.

4. Зміст текстової частини: Розробка програмного забезпечення централізованої криптовалютної системи.

5. Перелік ілюстративного матеріалу: Графічний інтерфейс додатків, UML-діаграми внутрішньої структури програмної розробки, тестування компонентів.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Коваленко О.О., к.т.н., доцент кафедри ПЗ	19.09.2023	05.12.2023
5	Кавецький В. В. к.е.н, доцент. каф. ЕПВМ	18.11.2023	19.01.2023

7. Дата видачі завдання 19 вересня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної Роботи	Строк виконання етапів роботи	Примітка
1	Аналіз існуючих криптовалютних систем	20.09.23 – 01.10.23	вик
2	Побудова архутектури компонентів системи	02.10.23 – 02.11.23	вик
3	Аналіз і вибір інтрументів для розробки програмної системи	03.11.23 – 05.11.23	вик
4	Розробка сервісів системи	06.11.23 – 19.11.23	вик
5	Написання тестів для перевірки внутрішнії компонентіів системи	20.11.23 – 26.11.23	вик
6	E2E тестування криптовалютної системи	27.11.23 – 30.11.23	вик
7	Створення документації для кодової бази	31.11.23 – 02.12.23	вик
8	Розробка економічної частини	18.11.23 – 19.11.23	вик
9	Оформлення матеріалів до захисту МКР	02.12.23 – 04.12.23	вик

Студент _____ **Мельник І.С.**
(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ **Коваленко О. О.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Мельник І. С. Удосконалення методів та засобів створення програмного забезпечення для обробки криптовалютних транзакцій. Магістерська кваліфікаційно робота зі спеціальності 121 – Інженерія програмного забезпечення, освітня програма – 121 Інженерія програмного забезпечення. Вінниця: ВНТУ, 2023. 194 с.

На укр. мові. Бібліогр.: 39 назв; рис.: 66; табл. 14.

У магістерській кваліфікаційній роботі удосконалено підхід до розробки програмного забезпечення для обробки криптовалютних транзакцій, шляхом застосування централізації архітектури системи. Даний підхід вирішує проблему застосування криптовалютної системи як фінансової платформи для використання в умовах повсякденного життя. Також проведено роботу з вирішення типових проблем децентралізованих криптовалютних систем.

У результаті розроблено централізовану криптовалютну систему розділену на декілька сервісів з метою розвантнаження центрального вузла в умовах надвиского навантаження. Модифіковано систему обробки транзакції, для збільшення швидкості процесу з можливістю збереження рівня безпеки. Удосконалено систему роботи з транзакціями та їх пошуку з використанням гнучких фільтрів. Створений програмний продукт розроблемно на платформі .NET.

Ключові слова: криптовалюта, хешування, централізація, блокчейн.

ANNOTATION

Melnyk I.S. Improvement of methods and tools of creating software for processing cryptocurrency transactions. Master's thesis on the specialty 121 - Software engineering, educational program - 121 Software engineering. Vinnytsia: VNTU, 2023. 194 p.

In Ukrainian speech Bibliography: 39 titles; Fig.: 66; table 14.

In the master's qualification work, the approach to the development of software for processing cryptocurrency transactions has been improved, by applying the centralization of the system architecture. This approach solves the problem of using the cryptocurrency system as a financial platform for use in everyday life. Work was also carried out to solve typical problems of decentralized cryptocurrency systems.

As a result, a centralized cryptocurrency system divided into several services was developed in order to offload the central node in conditions of excessive load. The transaction processing system has been modified to increase the speed of the process while maintaining the security level. The system of working with transactions and their search using flexible filters has been improved. The created software product is developed on the .NET platform.

Keywords: cryptocurrency, hashing, centralization, blockchain.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Дослідження та розробка централізованих криптовалютних систем все ще актуальні з кількох причин, незважаючи на те, що останніми роками акцент в індустрії криптовалют змістився у бік децентралізації. Ось кілька основних причин, чому централізовані криптовалюти залишаються предметом інтересу досліджень та розробок:

1. Зручність використання. Централізовані криптовалюти можуть забезпечувати більш простий і зручний досвід користувача. Вони можуть бути більш легкими у використанні для тих, хто тільки починає знайомитися з криптовалютами, і не потребує глибокого розуміння технічних аспектів.

2. Висока продуктивність. Централізовані криптовалюти можуть забезпечувати більш високу продуктивність та масштабованість. Вони не стикаються з проблемами, пов'язаними з масштабуванням, які можуть виникнути у децентралізованих мережах.

3. Легалізаційні та регуляторні питання. У багатьох країнах існують законодавчі та регуляторні норми, які потребують централізованого контролю для дотримання законів. Централізовані криптовалюти можуть краще відповідати цим нормам, що робить їх більш придатними для деяких юрисдикцій.

4. Можливості інновацій. Централізовані криптовалюти можуть надавати основу для експериментів та інновацій у галузі фінансових технологій (FinTech). Вони можуть стати майданчиком для впровадження нових технологій та розробок у галузі блокчейну та криптовалют.

5. Стабільність та передбачуваність. В умовах централізації управління можна забезпечити більш високу стабільність і передбачуваність у функціонуванні криптовалюти. Це може бути важливим аспектом для тих, хто цінує фінансову стабільність.

6. Експерименти з консенсус-механізмами. Централізовані криптовалюти також надають можливість експериментувати з різними

консенсус-механізмами, тим самим даючи розробникам досліджувати та порівнювати ефективність різних підходів.

7. Інтеграція з традиційними фінансовими системами. Централізовані криптовалюти можуть легше інтегруватися з традиційними фінансовими системами та банківськими послугами, що забезпечує ширший доступ та взаємодію з існуючою фінансовою інфраструктурою.

Хоча децентралізовані криптовалюти, такі як Bitcoin та Ethereum, привернули більше уваги та забезпечили унікальні переваги, централізовані криптовалюти залишаються затребуваними у тих випадках, коли зручність, продуктивність та дотримання законодавства відіграють ключову роль.

Тому актуальною є розробка програмних додатків для емісії та переказу криптовалюти, що дозволить гарантувати безперервність та незалежність процесу емісії і обробки транзакцій

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є підвищення стабільності платіжної системи шляхом розробки власної централізованої криптовалюти, що дозволяє гарантувати безперервність та незалежність процесу емісії й обробки транзакцій.

Основними задачами дослідження є:

– провести аналіз існуючих методів і засобів розробки програмного забезпечення для обробки криптовалютних транзакцій;

– удосконалити метод використання цетралізованого підходу до створення криптовалютної системи;

– розробити програмні компоненти для централізованої обробки криптовалютних тразакцій;

– провести експериментальні дослідження розроблених програмних засобів.

Об'єкт дослідження – процеси розробки та функціонування

криптовалютної системи.

Предмет дослідження – методи створення та функціонування криптовалютної системи та засоби обробки криптовалютних транзакцій.

Методи дослідження. У процесі досліджень використовувались методи дослідження:

– методи моделювання та проєктування – для побудови архітектури для створення криптовалютної системи;

– теорія алгоритмів та методи візуалізації для визначення процесів взаємодії програмних сервісів між собою; здійснення хешування інформації для оптимізації даних; обробки даних при роботі з багатопотоковими системами; формування блоків даних для їх структуризації; теорія захисту інформації – для запровадження методів зберігання конфіденційних даних для захисту інформації; методи тестування для перевірки працездатності системи.

Наукова новизна отриманих результатів.

1. Вдосконалено функціонал кріптовалютної системи, який, на відміну від існуючих, надає можливість змінювати параметри формування блоків даних для збереження інформації за рахунок оптимізації умов безпеки у періоди високого навантаження, що дозволяє підвищити рівень адаптивності системи в залежності від навантаження мережі.

2. Вдосконалено систему пошуку криптовалютних транзакцій, яка, на відміну від існуючих, дозволяє використовувати гнучкі фільтри, що підвищує швидкість та збільшує глибину пошуку.

3. Запропоновано модель використання інтегрованої системи доступу до електронного гаманця, яка, на відміну від існуючих, має розширений функціонал з можливістю відновлення доступу при втраті облікових даних.

Практична цінність отриманих результатів. Практична цінність полягає в запровадженні програмного забезпечення криптовалюної системи, яке може бути використано як платформа для створення швидких та безпечних фінансових операцій.

Особистий внесок здобувача. Усі наукові результати, викладені у

магістерській кваліфікаційній роботі, отримані автором особисто. У друкованих працях, опублікованих у співавторстві, автору належить модель життєвого циклу транзакції.

Апробація матеріалів магістерської кваліфікаційної роботи. Основні положення магістерської кваліфікаційної роботи доповідалися та обговорювалися на LI науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженерії (2022) ВНТУ [1].

Публікації. Основні результати дослідження опубліковані в науковій роботі – в тезах доповіді на LI науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженерії (2022) ВНТУ.

1 АНАЛІЗ СТАНУ ПИТАННЯ РОЗРОБКИ ДОДАТКІВ ДЛЯ ОБРОБКИ КРИПТОВАЛЮТНИХ ТРАНЗАКЦІЙ

1.1 Аналіз стану питання розробки криптовалютної системи

Технологія блокчейн [2], що використовується під час емісії криптовалют, дозволяє створювати необмежену кількість одиниць цифрової валюти. Емісія, як і сама структура криптовалюти [3], напряму залежить від розміру мережі, тобто кількості клієнтів які приймають участь. Чим більша мережа, тим більш активною буде процес емісії і гарантія працездатності та надійності системи в цілому.

У багатьох європейських країнах розглядається можливість випуску «державної» криптовалюти.

Всі види криптовалюти застосовують відкритий алгоритм генерації, тому підключення до загальної мережі будь-якої є суто добровільною справою. Передача інформації про електронний гаманець абсолютно безпечна, жодних персональних даних при цьому не розголошується. Однак це є гальмівним фактором для визнання монет криптовалюти державою, адже законодавство вимагає повного контролю за рухом грошових коштів у країні, а криптовалюта не передбачає цього.

Коли користувач розуміє, що таке криптовалюта і як нею користуватися, варто звернути увагу на ряд властивих цій платіжній системі недоліків. Вони спільні для більшості онлайн-ресурсів і легко обходяться при належній увазі безпеки свого комп'ютера.

Криптовалюти схильні до тих самих ризиків, що й онлайн-рахунки з реальними грошима:

- Збереження облікових даних. Мається на увазі небезпека як втрата так і крадіжка облікових даних. Друге загрожує втратою всіх коштів на гаманці.
- Висока волатильність. На зміні курсу криптовалюти можна заробити, так і зазнати втрат. Все залежить від коливань курсу після покупки

«монет», чи власник відразу витрачатиме валюту, чи прагнутиме створити накопичення.

- Велика можливість появи обмежувальних законів. У всіх країнах державні банки, як головний регулятор фінансової системи, прагнуть контролю за всією грошовою масою країни. У деяких країнах вже висловлюється ініціатива обмежити кількість криптовалюти, яка буде доступна фізичним особам, а у деяких взагалі повна заборона.

Майнінг втрачає популярність через різке підвищення складності обчислень алгоритму та постійного зростання вимог до апаратного забезпечення комп'ютерів, що використовуються для розрахунків. Вигідність покупки залежить від поточного курсу та зміни найближчим часом. Поки існує тенденція його постійного зростання, тимчасові коригування у бік зменшення ціни криптовалют незначні, але тим не менш Bitcoin та аналоги поки що залишаються привабливим інструментом для інвестування.

На фоні втрати популярності майнінгу частіше розглядаються питанні централізованих криптовалют, що можна вважати досить вагомою підставою для розвитку нової системи.

1.2 Аналіз методів для планування архітектури криптовалютної системи

В загальному існує 2 методи для планування архітектури криптовалютної системи – централізована криптовалюта [4] та децентралізовна криптовалюта [5].

Централізовані криптовалюти визначаються наявністю центрального органу чи сутності, яка контролює емісію, обіг та транзакції. Такий орган може бути корпорацією, фінансовою установою чи урядовим регулятором.

Централізовані криптовалюти випускаються та управляються центральним органом. Цей орган має повний контроль над обсягом монет, які виходять в обіг, і може втручатися в економіку криптовалюти за потреби. Залежно від політики конкретного централізованого проекту, рівень приватності та анонімності може варіюватися. У більшості випадків, органи, які контролюють централізовані криптовалюти, можуть встановлювати правила щодо ідентифікації користувачів.

Централізовані криптовалюти можуть бути менш вразливими до атак, оскільки контроль за системою зосереджений у руках обмеженого числа осіб чи організацій. Однак це може також створювати центральну точку атаки для зловмисників. Органи, що стоять за централізованими криптовалютами, можуть впливати на їхню політику, регулювання та рішення. Це включає в себе можливість зміни правил та умов використання. Прозорість централізованих криптовалют може бути обмеженою, оскільки вони можуть вирішувати питання конфіденційності та доступу до інформації. Конфіденційність та регулювання можуть бути визначені правилами, які встановлює центральний орган. Деякі централізовані криптовалюти вимагають ідентифікації користувачів. Централізовані криптовалюти можуть бути більш ефективними у відношенні швидкості та масштабованості, оскільки кількість учасників обмежена, і немає необхідності досягати консенсусу в розподіленій мережі. У порівнянні з децентралізованими криптовалютами, централізовані мають меншу ступінь децентралізації, іноді навіть втрачаючи деякі переваги, такі як анонімність та невпливовість.

В загальному централізований підхід до створення криптовалюти криптовалюти може бути застосований в рамках традиційних фінансових структур з інвестицією у швидкість, ефективності та більшу контрольованость на відміну від децентралізованого.

Децентралізовані криптовалюти – це цифрові валюти, які функціонують на основі технології блокчейн та не мають центрального органу чи контролю. Основні риси децентралізованих криптовалют включають розподілений характер, відсутність посередників, анонімність та автономію.

Децентралізовані криптовалюти базуються на технології блокчейн, яка є розподіленою базою даних, в якій транзакції зберігаються у блоках, з'єднаних ланцюгом. Децентралізовані криптовалюти не мають центрального органу чи владаря. Рішення приймаються за допомогою протоколів консенсусу, таких як Proof-of-Work або Proof-of-Stake. Мережа складається з розподілених вузлів, кожен з яких зберігає копію блокчейну та бере участь у процесі консенсусу.

Відсутність одного центрального пункту атаки забезпечує безпеку мережі. Децентралізовані криптовалюти дозволяють користувачам самостійно управляти своїм капіталом, проводити транзакції та участь у голосуваннях за важливі зміни у мережі. Багато децентралізованих криптовалют прагнуть забезпечити анонімність та приватність користувачів шляхом використання різних технік, таких як конфіденційні транзакції або анонімні адреси.

Децентралізовані криптовалюти використовують різні технічні протоколи для забезпечення безпеки та працездатності. Учасники мережі, відомі як майнери або вузли, мають рівні права та беруть участь у консенсусі для підтвердження транзакцій. Деякі криптовалюти дозволяють створювати децентралізовані

Розвиток децентралізованих криптовалют здійснюється через процес узгодженого оновлення, де учасники голосують за запропоновані зміни в мережі.

Децентралізовані криптовалюти представляють собою новий підхід до фінансів та економіки, забезпечуючи користувачам більший контроль, безпеку та анонімність у порівнянні з традиційними фінансовими системами.

1.3 Порівняльний аналіз аналогів

Bitcoin [6] є першою та найбільш відомою криптовалютою у світі, представленою в 2009 році під псевдонімом Сатосі Накамото. Це цифрова валюта, яка дозволяє користувачам виконувати електронні транзакції через Інтернет без потреби посередників, таких як банки чи урядові органи.

Рисунок 1.1 – Графічне зображення логотипу Bitcoin

Біткоїн працює на основі технології блокчейн, що забезпечує децентралізовану систему управління та безпеку транзакцій. Ключові аспекти біткоїна включають обмежену емісію (лише 21 мільйон біткоїнів), процес майнінгу для створення нових монет та анонімність користувачів, забезпечена криптографією. Вартість біткоїна визначається ринковою попитом та пропозицією, і його ціна може змінюватися величезними обсягами.

Технічні параметри:

- Консенсус-механізм – Proof-of-Work (PoW) з використанням алгоритму SHA-256.
- Швидкість транзакцій – низька (близько 7 транзакцій у секунду).
- Вартість транзакцій – варіюється, але комісії можуть бути високими в період пікової активності.
- Маштабування – обмежене, проблеми зі збільшенням розміру блоку.
- Смарт-контракти – обмежена підтримка смарт-контрактів.
- Приватність – відносно низький рівень анонімності; транзакції відкриті і трассовані.
- Тип зберігання дани – повний вузол зберігає повну історію всіх транзакцій.
- Алгоритм хешировання – SHA-256.
- Розробка та суспільство – сильне суспільство розробників, але іноді виникають суперечки.
- Сумісність – відносно обмежена сумісність, більше можна зустріти порівняння, як «цифрове золото».

Примітні особливості:

- Bitcoin був першою криптовалютою, створеної в 2009 році під псевдонімом Сатосі Накамото.
- Основна ціль – надати децентралізоване сховище грошових цінностей.
- Обмежена загальна пропозиція в 21 мільйон біткойнів.

Ethereum [7] – це відкрита блокчейн-платформа, що дозволяє розробникам створювати та розгортати розумні контракти (smart contracts) і децентралізовані додатки (DApps). Платформа була запущена у 2015 році Віталієм Бутеріном і стала важливим кроком у розвитку блокчейн-технологій, особливо у сфері децентралізації та автоматизації угод.

Рисунок 1.2 – Графічне зображення логотипу Ethereum

Основна ідея Ethereum полягає в тому, щоб створити універсальну платформу для розробки розумних контрактів, що можуть взаємодіяти з іншими контрактами та додатками. Ефір (ETH) є власною криптовалютою Ethereum і використовується для оплати транзакцій та винагороди майнерам.

Ethereum відрізняється від Bitcoin, оскільки його блокчейн спроектований для підтримки розумних контрактів, що відкриває широкий спектр можливостей для використання технології блокчейн у різних галузях, включаючи фінанси, логістику, медицину та багато інших. Ethereum є однією з найбільш використовуваних та впливових блокчейн-платформ у світі.

Технічні параметри:

- Консенсус-механізм – перехід з PoW на PoS в процесі.
- Швидкість транзакцій – середня (близько 30 транзакцій у секунду).

- Вартість транзакцій – комісія може бути високою в період навантаження на мережу.
- Масштабування – обмежене, у планах впровадження шардингу і PoS для підвищення можливостей до масштабування.
- Смарт-контракти – лідер в смарт-контрактах з підтримкою Solidity.
- Приватність – рівень приватності обмежений, але працюють над поліпшенням.
- Тип зберігання даних – повний вузол зберігає всі дані про стан контрактів.
- Алгоритм хешування – Keccak (SHA-3).
- Розробка та суспільство – активне та інноваційне суспільство розробників, але рішення масштабних проблем тривале.
- Сумісність – токени Ethereum ERC-20 популярні для токенізації активів.

Litecoin [8] – це криптовалюта, яка була випущена у 2011 році Чарльзом Лі як конкурент Bitcoin. Litecoin є відкритим програмним забезпеченням, і, так само як і Bitcoin, використовує технологію блокчейн для забезпечення децентралізованих та безпечних транзакцій.

Рисунок 1.3 – Графічне зображення логотипу Litecoin

Однією з основних цілей Litecoin було зменшення часу обробки блоків та полегшення майнінгу порівняно з Bitcoin. Litecoin використовує алгоритм Proof-

of-Work, подібний до Bitcoin, але із деякими відмінностями, такими як скорочений час обробки блоків і застосування іншого хеш-алгоритму Scrypt.

Litecoin часто описують як "цифрове срібло" порівняно з "цифровим золотом" у випадку з Bitcoin. Він має власну криптовалюту під назвою LTC, яка використовується для проведення транзакцій та винагороди майнерам. Ця криптовалюта набула певної популярності в криптовалютному середовищі і використовується для різних цілей, включаючи онлайн-платежі та перекази.

Технічні параметри:

- Консенсус-механізм – PoW з використанням Scrypt.
- Швидкість транзакцій – висока (близько 56 транзакцій у секунду).
- Вартість транзакцій – зазвичай низькі комісії, що роблять Litecoin привабливим для мікротранзакцій.
- Масштабування – відносно хороше масштабування завдяки більш швидкому процесу створення нових блоків.
- Смарт-контракти – обмежена підтримка смарт-контрактів.
- Конфіденційність – низький рівень анонімності, але реалізовано оновлення з фокусом конфіденційності (MimbleWimble).
- Тип зберігання даних – повний вузол зберігає всі транзакції.
- Алгоритм хеширования – Scrypt.
- Розробка та суспільство – невелике, але активне суспільство.
- Сумісність – часто використовується як тестова площадка для нових технологій.

В загальному – Bitcoin є основним цифровим золотом, призначенем для накопичення грошових цінностей, але стикається з обмеженою масштабністю та високими комісійними. Ethereum – лідер в області смарт-контрактів, активного співтовариства, але з масштабними проблемами. Litecoin – швидка і дешева альтернатива для мікротранзакцій з фокусом на конфіденційності.

Враховуючи основні переваги та недоліки аналогів власна розробка матиме наступні технічні параметри:

- Консенсус-механізм – PoW з використанням SHA-512.
- Швидкість транзакцій – теоретично необмежена (залежність лише від технічних характеристик сервера обробки транзакцій).
- Вартість транзакцій – відсутність комісії.
- Масштабування – наявність механізму гнучкої конфігурації параметрів системи в залежності від поточного навантаження на мережу.
- Смарт-контракти – відсутня підтримка.
- Конфіденційність – достатньо високий рівень за рахунок мінімального використання персональних даних.
- Тип зберігання даних – централізований блокчейн.
- Алгоритм хешировання – SHA-512.
- Сумісність– можливість інтеграції з іншими сервісами через API.

Згідно результатів порівняння наявних на ринку криптовалютних платформ та в загальному питання криптовалюти як платіжного сервісу – отримані результати демонструють на доцільність розробки власної системи. Основна ідея централізації дозволить використовнувати нову платформу у державній та повсякденній сфері використання. Влана розробка також надасть можливість уникнути фактору надлишковості сторонніх сервісів для взаємодії с користувацьким гаманцем, оскілки система пропонує власний інтегрований сервіс для захищеної роботи.

1.4 Постановка задач дослідженяя

Проаналізувавши питання розробки криптовалютних систем, було визначено завдання, які необхідно виконати для розробки програмного рішення:

– розробити модель централізованої криптовалютної системи;

– розробити базу даних для зберігання інформації системи;

– розробити сервіс для обробки та валідації даних про криптовалютні транзакції;

– розробити сервіс для роботи з електронним гаманцем користувача;

– розробити сервіс для підтримки централізованого блокчейшну;

– розробити технічну документацію програмного рішення криптовалютної системи;

– провести тестування модулів системи з використання Unit-тестів;

– провести E2E-тестування криптовалютної системи.

1.5 Висновки

У першому розділі було детально досліджено сучасний стан криптовалютних систем. Зокрема, проведено порівняльний аналіз відомих аналогів, таких як Bitcoin, Ethereum та Litecoin. Порівняння виявило, що розглянуті криптовалюти отримали широку популярність завдяки інноваційності своєї концепції та функціонують як піонерські системи у цьому галузі.

Аналіз переваг і недоліків існуючих криптовалютних систем дозволив визначити, що вони, будучи децентралізованими, мають досить обмежені можливості для звичайного використання у повсякденних фінінсових справах та потенційної ролі як основної державної валюти. Це обумовлено їх структурою та спонукає до ідеї розробки власної централізованої криптовалюти, яка має потенціал вирішити ці обмеження.

На підставі отриманих результатів було прийнято рішення розробити власне програмне рішення, орієнтоване на централізовану архітектуру. Це рішення покликане надати новий рівень ефективності та можливостей для використання у реальному житті, у той час як збереження централізованого контролю над системою забезпечить її стійкість та надійність.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОБРОБКИ КРИПТОВАЛЮТНИХ ТРАНЗАКЦІЙ

2.1 Аналіз архітектурних підходів до створення програмної системи

Створення архітектури додатку [9] – це ключовий етап у процесі розробки, і правильно обрана архітектура може значно полегшити підтримку, розширення та вдосконалення програмного продукту. Існує кілька підходів до створення архітектури додатку, і вибір конкретного підходу залежить від конкретних вимог створюваного програмного продукту. Розглянемо основні підходи до побудови архітктури програмної системи:

1. Монолітна архітектура – усі компоненти додатку об'єднані в одному цілому, зазвичай у вигляді єдиної кодової бази та монолітного додатку.

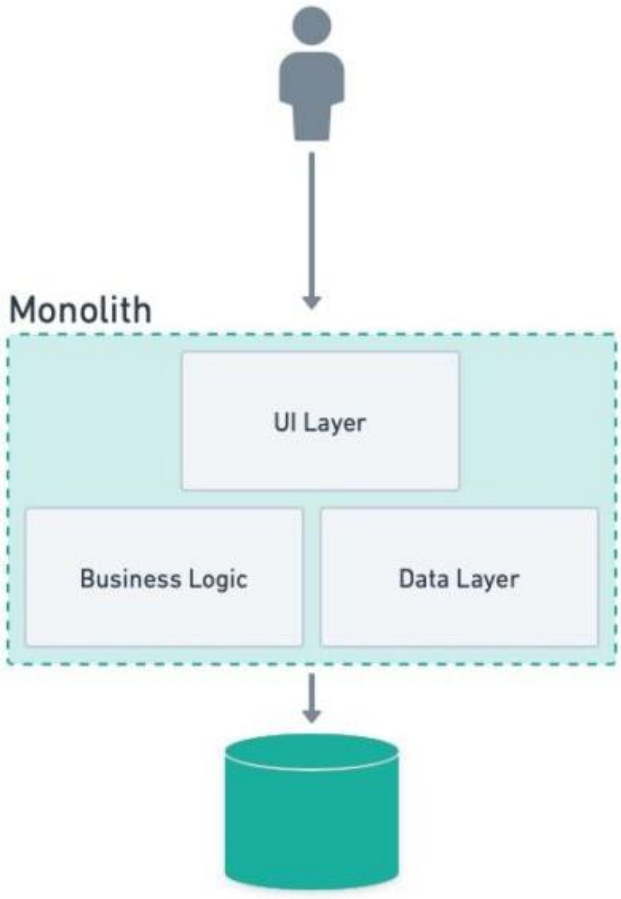

Рисунок 2.1 – Загальна схема монолітної архітектури

Переваги:

- Простота в розробці та розгортанні.
- Зниження складності в управлінні.

Недоліки:

- Обмежена масштабованість.
- Важкість у підтримці при зростанні обсягів та розширенні функціоналу.

2. Мікросервісна архітектура – додаток розділяється на невеликі, автономні мікросервіси, які працюють над конкретною функціональністю.

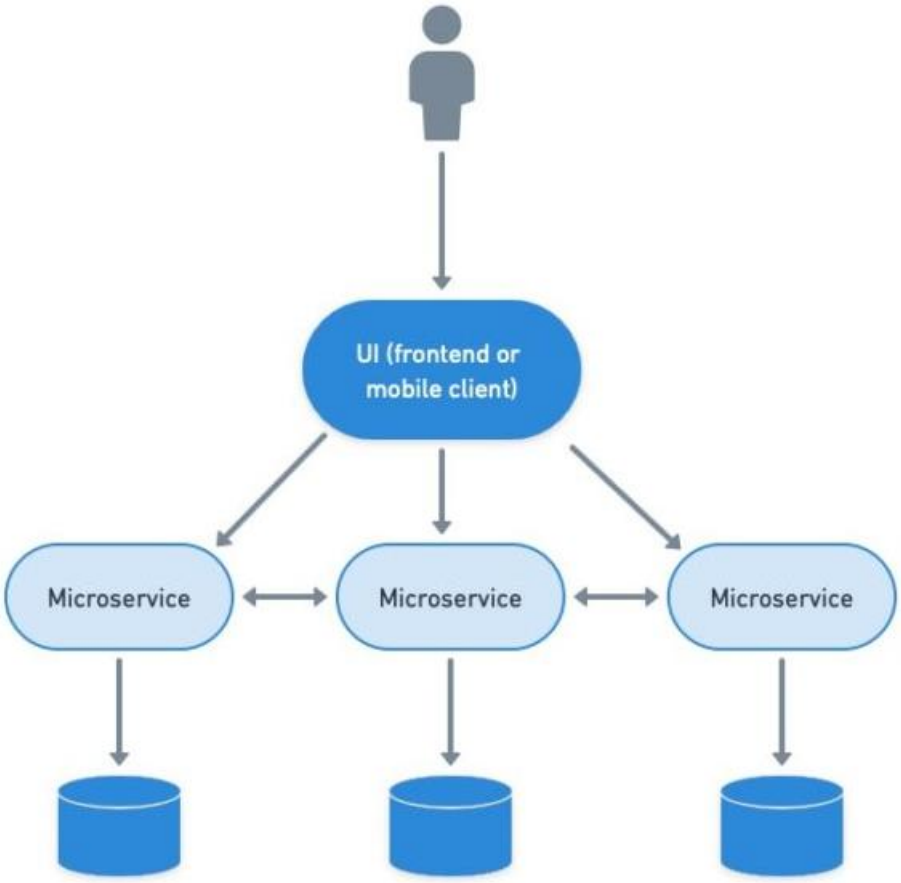

Рисунок 2.2 – Загальна схема мікросервісної архітектури

Переваги:

- Легкість у масштабуванні та підтримці.

- Незалежність мікросервісів дозволяє різним командам працювати паралельно.

Недоліки:

- Складніша оркестрація та управління мікросервісами.
- Збільшення витрат на інфраструктуру.

3. Event-Driven архітектура – компоненти обмінюються подіями та реагують на зміни стану через підписку на події.

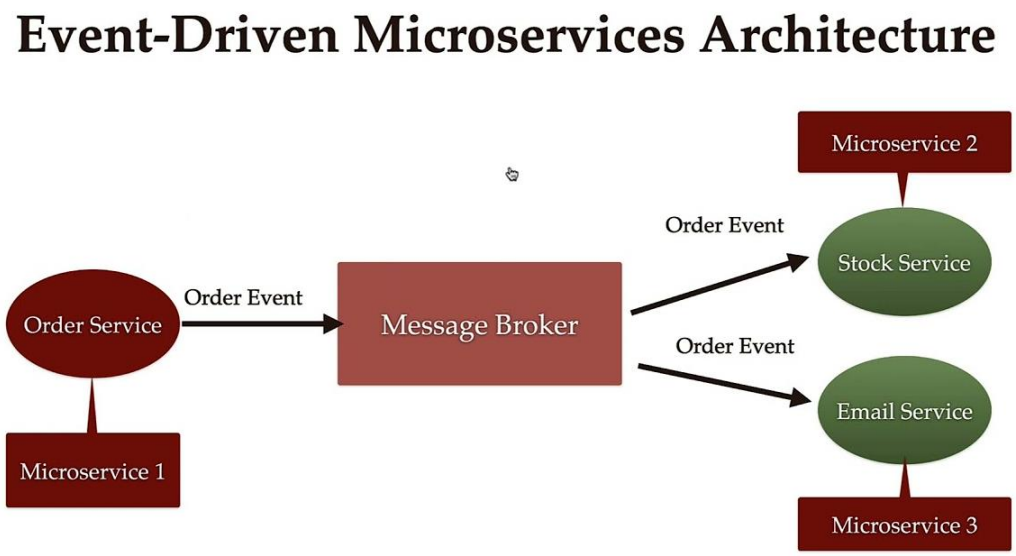

Рисунок 2.3 – Загальна схема Event-Driven архітектури

Переваги:

- Різні компоненти можуть бути розгорнуті та оновлювані окремо.
- Легка інтеграція нових функцій.

Недоліки:

- Система може стати важкою для розуміння через асинхронну природу подій.
- Важкість відлагодження та трасування.

4. Шарова архітектура – додаток розділяється на логічні шари (представлення, бізнес-логіка, доступ до даних), де кожен шар відповідає за конкретний аспект функціоналу.

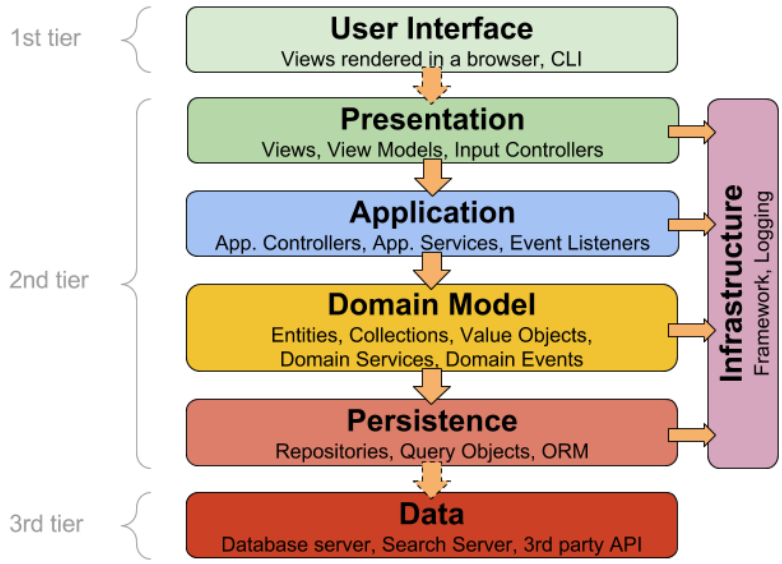

Рисунок 2.4 – Загальна схема шарової архітектури

Переваги:

- Легка модульність та підтримка.
- Забезпечує відокремленість концепцій.

Недоліки:

- Може призвести до зависимості між шарами.
- Важкість у підтримці при значних змінах.

1. Компонентна архітектура – додаток складається з незалежних компонентів, які можуть бути розроблені та підтримувані окремо.

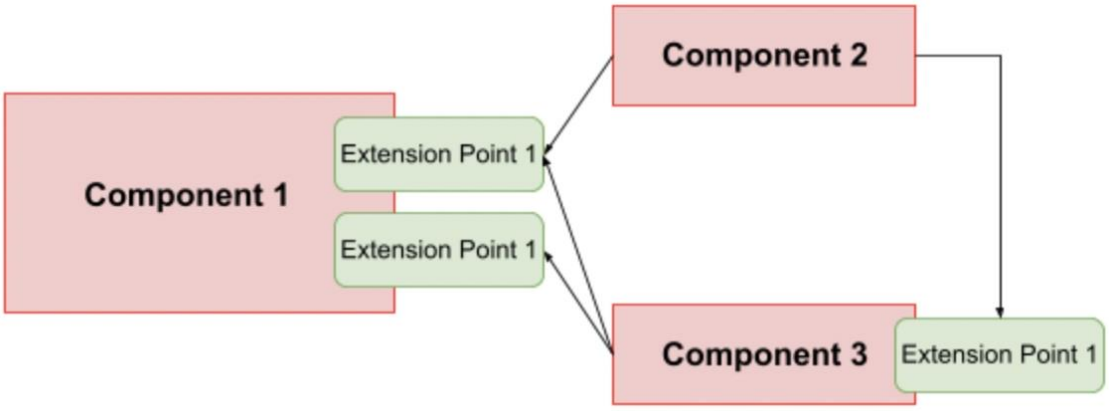

Рисунок 2.5 – Загальна схема компонентної архітектури

Переваги:

- Легкість у рефакторингу та заміні компонентів.
- Зниження витрат на розробку завдяки використанню готових компонентів.

Недоліки:

- Збільшення складності в управлінні та координації компонентами.
- Важкість у збереженні консистентності між компонентами.

В загальному кожен архітектурний підхід має свої переваги та недоліки в порівнянні з іншими. Ці підходи можна комбінувати або адаптувати відповідно до конкретних вимог проектного рішення та його контексту, отримуючи від комбінації найбільший виграш при побудові архітектурного рішення. Важливо враховувати, що не існує універсального рішення, і правильний вибір архітектури буде залежати від конкретних умов та потреб системи. І завдяки гнучкості поєднання архітектурних підходів можна нівелювати певні недоліки підходів, якщо використовувати їх окремо.

Для розробки криптовалютної системи буде використано комбінацію з мікросервсного, шарового та Event-Driven підходів до побудови архітектури програмної системи. Архітектуру найвищого рівня буде описано за допомогую мікросервісного підходу. Шаровий підхід в свою чергу визначати внутрішню будову мікросервісів. Та взаємодію критичних вузлів мікросервісів окрім використаня класичних server-to-server API запитів визначатиме Event-Driven підхід.

2.3 Загальна архітектура системи

Враховуючи взятого за основу мікросервісну архітектуру визачемо та склдемо загальну схему сервісів та потенційних шляхів взаємодії. На рисунку 2.6 продемонстровано загальну карту сервісів, з яких складатиметься криптовалютна система.

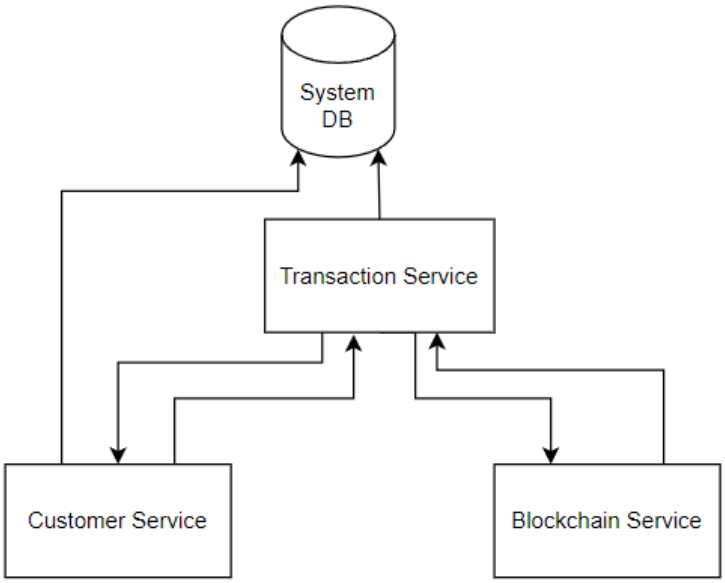

Рисунок 2.6 – Загальна схема мікросервісів криптовалютної системи

Як видно зі схеми, криптовалютна система має 4 компоненти:

1. DB – сховище даних, де буде зберігатись вся необхідна інформація про криптовалютну систему.

2. Transaction Service – центральний сервіс обробки криптовалютних транзакцій, валідації інформаційних вузлів системи та забезпеченні збереження перевірених даних у сховищі.

3. Customer Service – інтегрований сервіс для взаємодії користувачів системи з криптовалютним гаманцем, де користувач вводячи обікові дані від влсного гаманця може отрмати доступ до інформаціїї по існуючим транзакціям, що вже пройшли верифікацію та збережені у сховищі або ще знаходяться у процесі перевірки та збереження.

4. Blockchain Service – сервіс для забезпечення цілісності та безпеки криптовалютних транзакцій шляхом шляхом формування зв’язку між криптовалютними транзакціями.

Також згідно схеми маємо наступну варіацію ваємодії сервісів:

1. Customer Service => Transaction Service – варіація взаємодії сервіса для користувачів та центрального сервіса обробки транзакцій передбачає отримання поточних даних по криптовалютному гаманцю коричтувача на запит

та можливість ініціювати створення нової транзакції з відстеженням статусу у режимі реального часу.

2. Customer Service => DB – варіація взаємодії сервіса користувачів передбачає отримання даних з метою підтвердження валідності обілкових даних користувача для надання доступу до системи з урахуванням наявних прав та статусу акаунту або збережння даних при створенні нового облікового запису для користувача.

3. Blockchain Service => Transaction Service – варіація взаємодії сервіса для забезпечення цілісності та безпеки криптовалютних транзакцій і центрального сервіса транзакцій з метою отримання певної кількості поточних неопрацьованих транзакцій та формування цілісної структури даних для подальшого збереження.

4. Transaction Service => Customer Service – варіація взаємодії центрального сервіса транзакцій та сервісу користувачів полягає у підтримці процесу отримання даних у режимі реального часу без необхідності виконання запиту для отримання поточних оновлених вручну, що в свою чергу підвищує зручність та своєчасність отримання інформації про криптовалютні транзакцію по певному електроному гаманцю.

5. Transaction Service => Blockchain Service – варіація взаємодії центрального сервіса транзакцій та сервіса для забезпечення цілісності та безпеки криптовалютних транзакцій полягає у перевірці працездатності критично-важливої інфраструктури системи, щоб у випадку тимчасової втрати зв’язку з одним із вузлів перелаштувати маршрути взаємодії до доступних вузів сервісу.

6. Transaction Service => DB – варіація взаємодії центрального сервіса транзакцій та сховища даних полягає у збереженні валідної та закодованої інформації про криптовалютні транзакції та електронні гаманці користувачів.

Розглянемо внутрішню структуру кожного з сервісів більш детально. На рисунку 2.7 проілюстровано принцип побудови внутрішніх шарів мікросервісів.

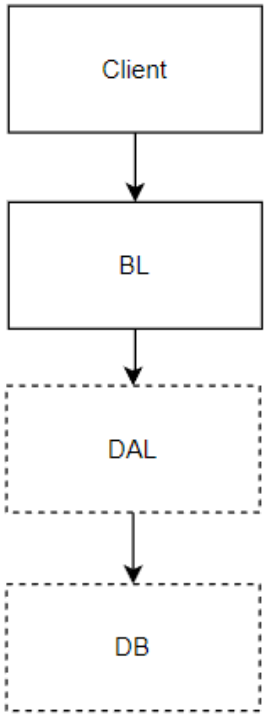

Рисунок 2.7 – Внутрішня шарова структура сервісів

Як видно зі схеми, мікросервіси мають 2 обов’язкові шари та 2 опіціональні, оскільки не всі сервіси взаємодіють зі сховищем даних, а тому не повинні мати надлишковукову інфраструктуру.

Шар «Client» – відповідає за надання доступу до можнливостей сервісу стороннім користувачам (у даному випадку іншим сервісам) через створення API запиту, та отримання стандартизованої відповіді про результат відпрацювання функціональних вузлів пов’язаних з конкретним API-методом.

Шар «BL» – або іншими словами «бізнес-логіка» – описує ключові функціональні можливості сервісу. Саме тут описані всі алгоритми та паттерни поведінки сервіса в залежності від різних умов. Цей шар має доступ до більш низькорівневих шарів, щоб відоокремити код клієнта від інших частин сервісу, а тому дуже часто є проміжним компонентом системи, якщо потрібно виконувати server-to-server запити між кількома сервісами.

Шар «DAL» – відповідє за доступ до даних, коли необхідно щось зберегти або отримати. Тут описується логіка і методи взаємодії зі сховищем та маршрутизація між сховизами у випадку, якщо у системі функіонує декілька баз

даних для прикладу.

Шар «DB» – це конкретна системна імплементація взаємодії сервісу зі сховищем, що є для кожного сховище окремою.

В загальному мікросервсна архітектура з шаровим внутрішнім поділом забезпечує чітку та просту структуру системи, яку легко підтримувати та розширювати.

Маючи загальні схеми внутрішньої архітектури сервісів розглянему більш детально складові системи.

Почнемо з шару доступу до даних, що є ключовою ланкую між сховищем даних та програмною системою. На рисунку 2.8 зображено класову схему шару доступу до даних.

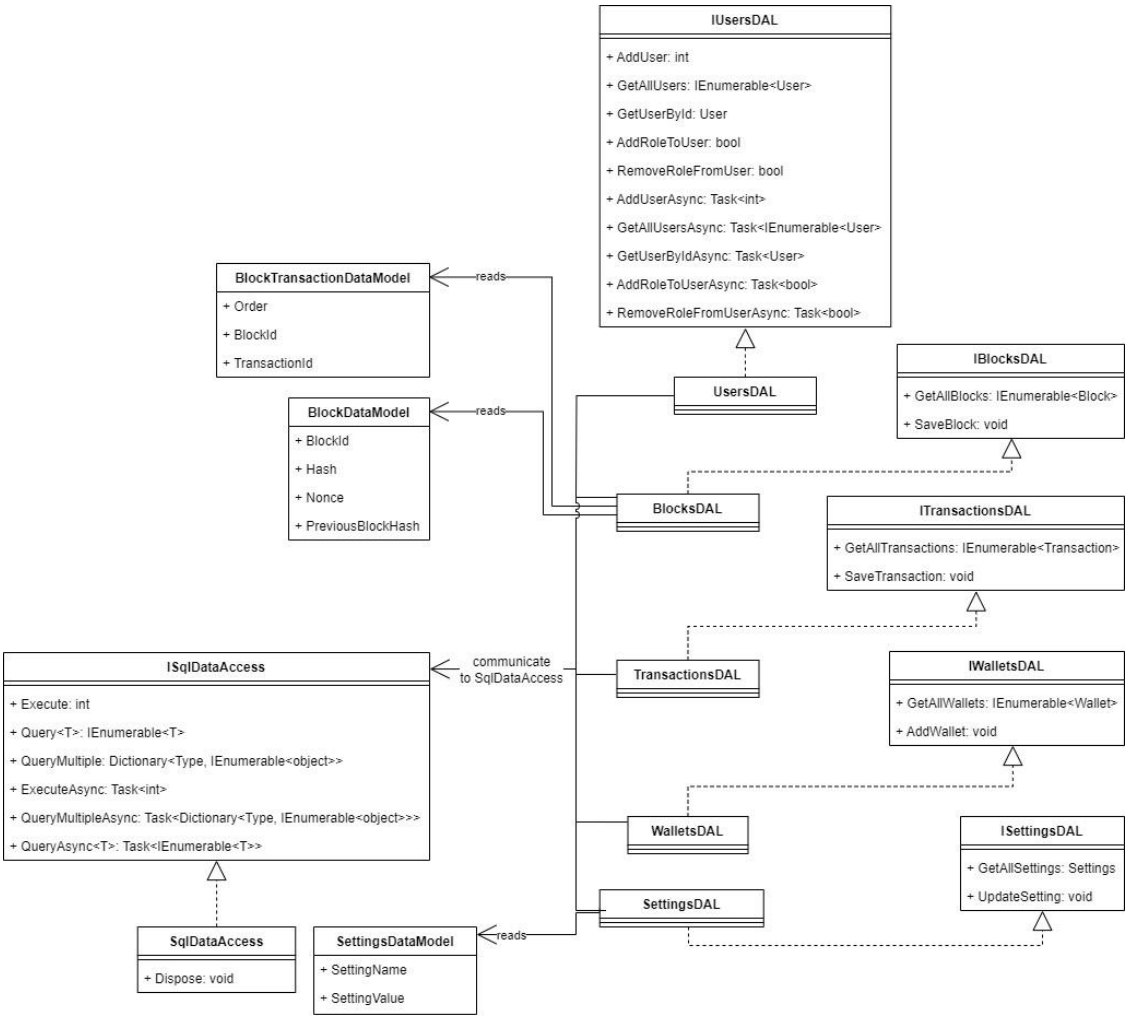

Рисунок 2.8 – Шар доступу до даних

Кожен об'єкт шару доступу до даних відповідає за конкретну сутність у базі даних та виконує перетворення даних, отриманих через запит, у модель, з якою може взаємодіяти система. Деякі моделі можуть бути складними та включати кілька сутностей з бази даних, наприклад, модель Block, що представляє собою комплексний об'єкт із списком транзакцій. Оскільки дані для цієї моделі зберігаються у різних сутностях, а також в одній спеціальній сутності для зберігання та гарантування порядку транзакцій у блоку, потрібні відповідні моделі на рівні програмного додатку, що представляють ці сутності з бази даних. Ці моделі потім використовуються для формування зрозумілої системі єдиної моделі.

Кожен DAL клас також має відповідний інтерфейс, який можуть використовувати вищорівневі модулі системи, необов'язково знаючи, як саме дані отримуються з бази даних.

У загальному, шар доступу до даних містить наступні модулі:

- SettingsDAL – модуль забезпечує доступ до зчитування та збереження даних про налаштування системи через інтерфейс ISettingsDAL.
- WalletsDAL – модуль надає доступ до зчитування та збереження даних про електронні гаманці через інтерфейс IWalletsDAL.
- BlocksDAL – модуль забезпечує доступ до зчитування та збереження даних про блоки транзакцій через інтерфейс IBlocksDAL.
- TransactionsDAL – модуль надає доступ до зчитування та збереження даних про транзакції через інтерфейс ITransactionsDAL.
- UsersDAL – модуль забезпечує доступ до зчитування та збереження даних про користувачів через інтерфейс IUsersDAL.

На найнижчому рівні реалізації прямого доступу до бази даних використовується клас SqlDataAccess та відповідний інтерфейс ISqlDataAccess. Цей рівень дозволяє взаємодіяти з базою даних із застосуванням DAL класів, забезпечуючи їхню коректну інтеграцію у систему.

На рисунку 2.9 зображено класову схему сервісу користувачів для взаємодії з електронним гаманцем.

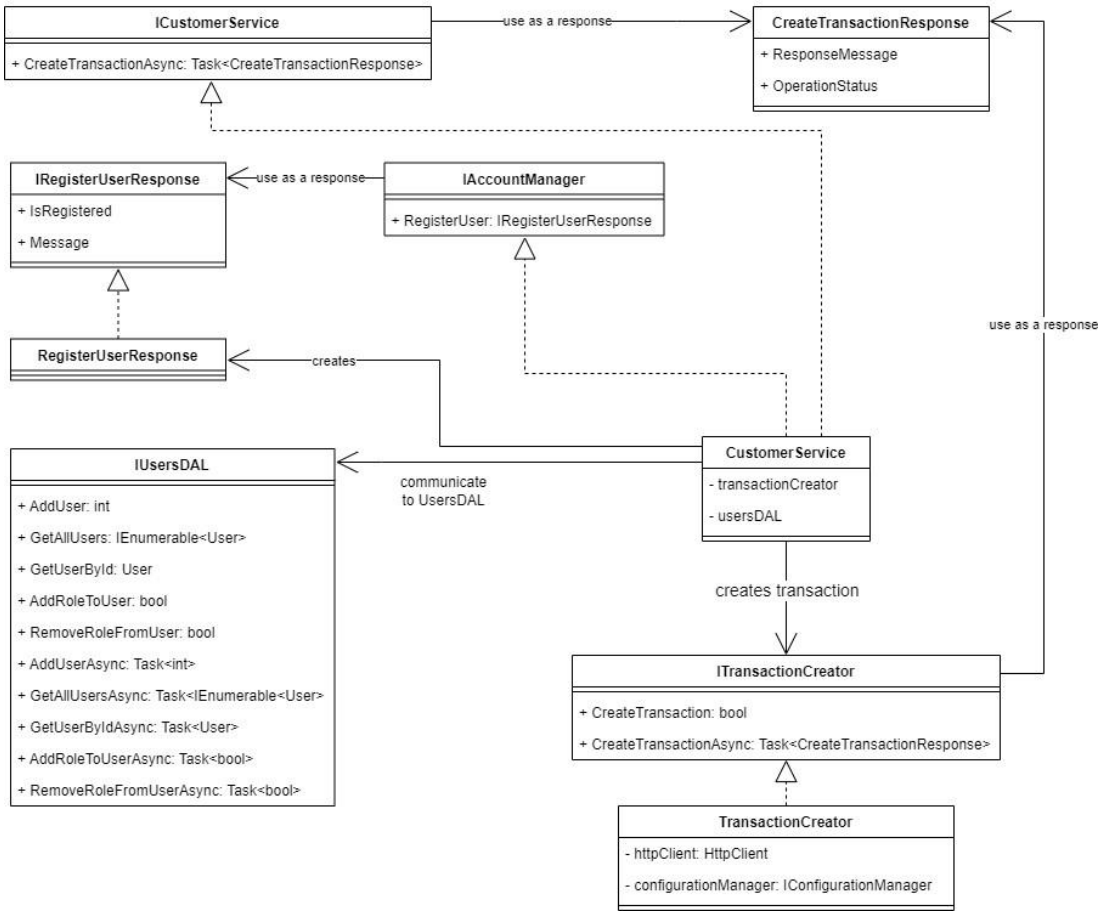

Рисунок 2.9 – Сервіс користувачів для взаємодії з електронним гаманцем

Основна функція сервісу полягає в створенні ефективного механізму взаємодії між кінцевим користувачем і системою. Сервіс складається з веб-клієнта, доступного через веб-браузер для зручного користування, та серверної частини, яка відповідає за важливі аспекти взаємодії з користувачем, такі як реєстрація, авторизація, аутентифікація та пересилання запитів із веб-клієнта до TransactionService API.

Однією з ключових мет цього сервісу є надання користувачам можливості ефективної реєстрації та управління аутентифікацією і авторизацією. Це відкриває двері до перегляду власного гаманця та можливості створення нових транзакцій.

На етапі реєстрації користувачеві надається можливість легко та безпечно вступити до системи. Далі, після успішної аутентифікації, відкривається

можливість авторизації, що дозволяє користувачеві здійснювати дії в системі, такі як перегляд інформації про власний гаманець та створення нових транзакцій.

Важливий етап цього процесу – перенаправлення запитів із веб-клієнта до TransactionService API, забезпечуючи зручний та безпечний обмін даними між користувачем та сервісом.

Отже, мета сервісу полягає в створенні дружелюного та ефективного середовища для користувачів, яке дозволяє їм легко реєструватися, аутентифікуватися, авторизовуватися та виконувати різноманітні дії у системі, розширюючи можливості управління своїм гаманцем та здійсненням транзакцій.

На рисунку 2.10 зображено класову схему сервісу для забезпечення цілісності та безпеки криптовалютних транзакцій

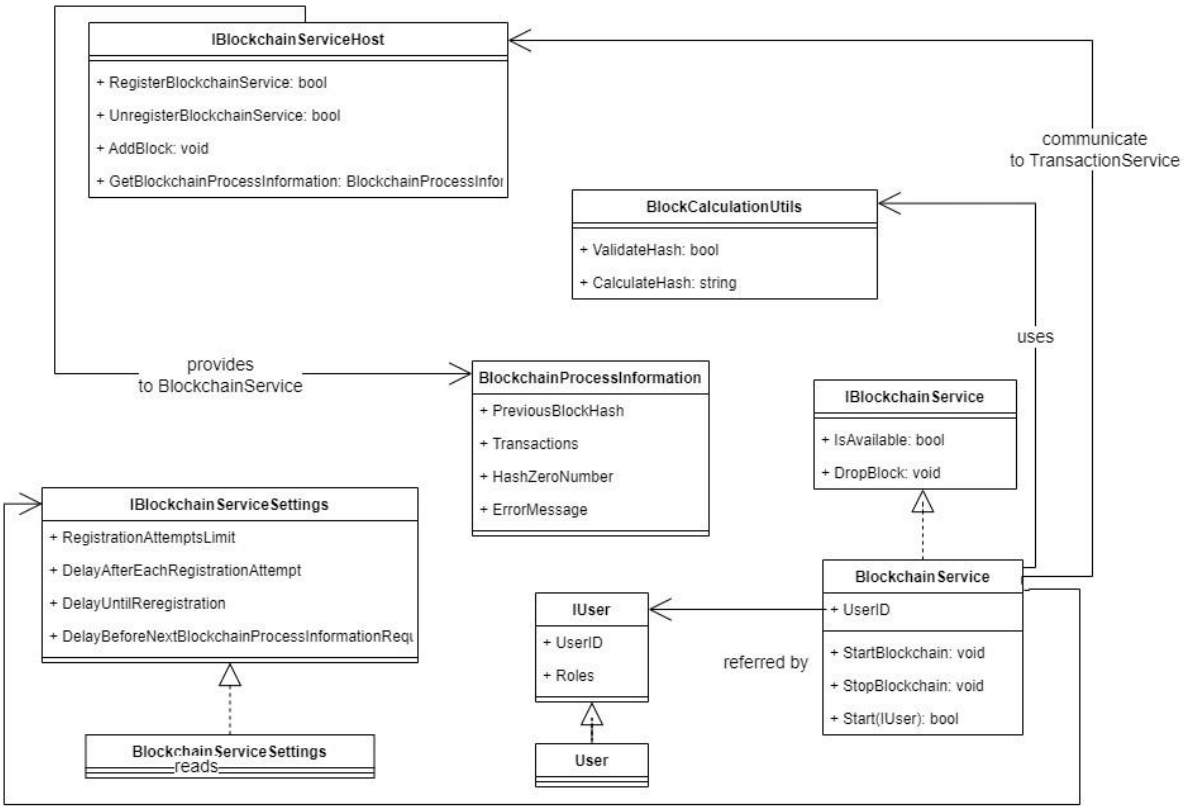

Рисунок 2.10 – Сервіс для забезпечення цілісності та безпеки криптовалютних транзакцій

Сервіс для забезпечення цілісності та безпеки криптовалютних транзакцій

працює з криптовалютними транзакціями після їх успішного створення. Сервіс блокчейну отримує від сервісу обробки транзакцій групу тразакцій фіксованого розміру, визначеного гнучкими налаштуваннями. Після чого ініціюється процес обрахунку хешу транзакцій з метою формування хешованого блоку транзакцій, який буде задовольняти умову безпеки. Сервіс представляє собою додаток, що може бути розгорнутий не лише у единому екземплярії, а мати декілька екземплярів, кожен з яких приймати участь у обрахуку, за рахунок цього зростає швидкість обробки транзакцій, оскільки кожен з сервісів має можливість швидше обрахувати хеш таким чином, щоб умова безпеки була виконана.

На рисунку 2.11 зображено класову схему центрального сервісу обробки транзакцій

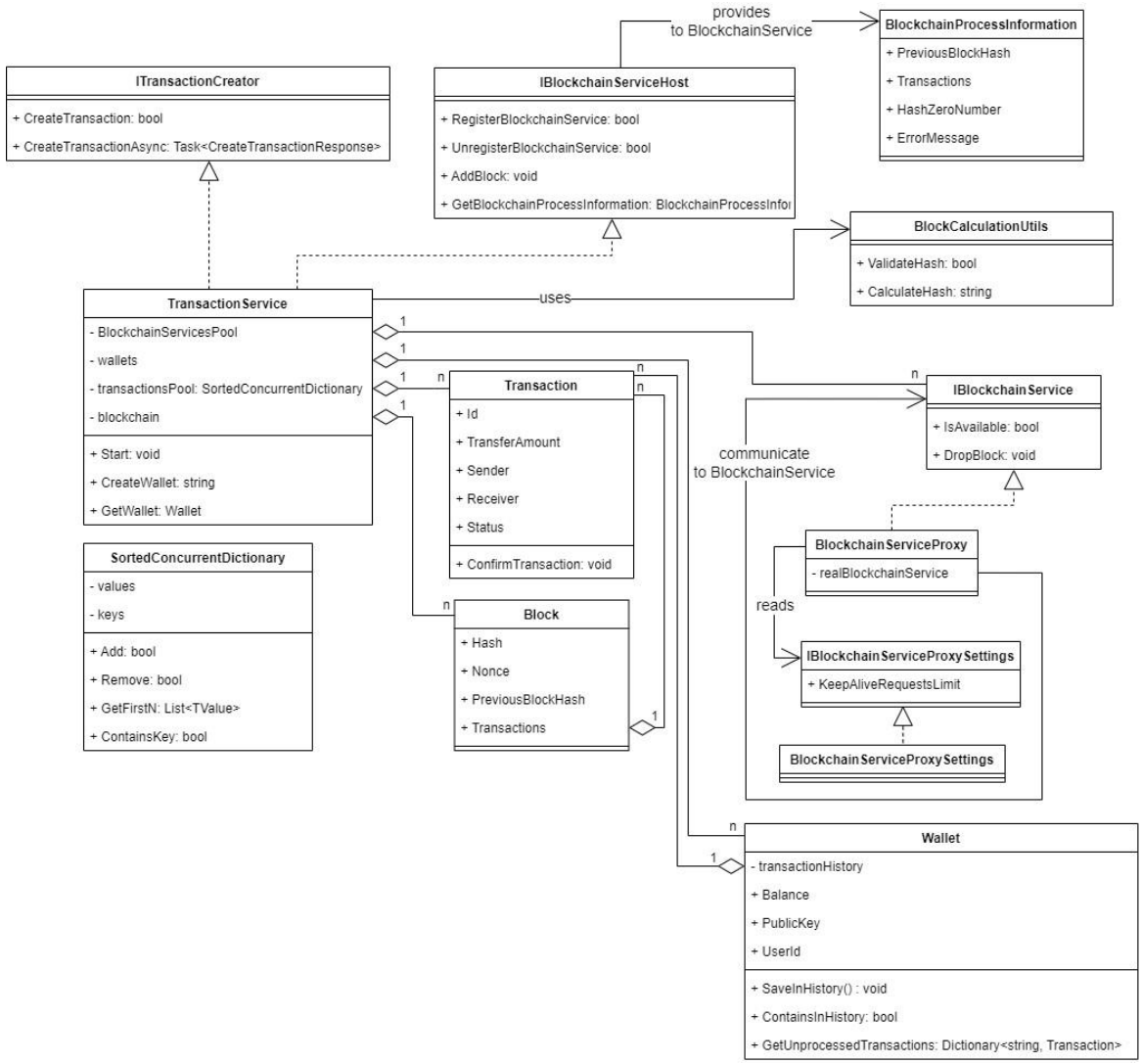

Рисунок 2.11 – Центральний сервіс обробки транзакцій

Сервіс обробки транзакцій відіграє стратегічну роль у повному життєвому циклі системи. Вже на етапі ініціалізації сервіс виконує загальну валідацію всіх наявних даних у базі даних, забезпечуючи їхню коректність. Протягом роботи системи, він продовжує валідувати дані, які підлягають збереженню у базі. Через безпосередній доступ до Data Access Layer (DAL) маємо можливість ефективно зберігати нові транзакції, блоки та інші структури у базі даних.

Бізнес-логіка, що реалізована у сервісі транзакцій, стає доступною через API, яке може бути викликане зовнішніми сервісами. Це дозволяє іншим системам взаємодіяти з ключовими функціональностями системи, реалізованими в сервісі транзакцій.

Сервіс операціює з різними моделями даних, такими як:

- Транзакція – Основна модель даних системи, що містить інформацію криптовалюту. Транзакція може бути описана як контракт для передачі права власності певної кількості криптовалютних одиниць. між двома електронними гаманцями на основу публічного ключа кожного з них
- Електронний гаманець – Структура даних, що визначає право власності на криптовалютну транзакцію та дозволяє формувати баланс користувача на основі наявної історії транзакцій.
- Блок – Структура даних, що забезпечує цілісність транзакцій та відіграє ключову роль у процесі підтвердження транзакції та її подальшого зберігання у системі.

Сервіс також використовує різні структури даних, такі як:

- Список гаманців – Колекція, яка локально у пам'яті зберігає всі електронні гаманці системи для швидкого доступу.
- Блокчейн – або ланцюг блоків – Односпрямована черга, що містить послідовність хешованих блоків транзакцій. Кожен блок містить фіксовану кількість транзакцій, що визначається системини налаштуваннями. Кожна транзакція включаючи всю інформацію про учасників та загальну суму транзакції приймає учать в обрахунку хешу блока.

- Пул необроблених транзакцій Спеціальна структура даних, що представляє собою сортований потокобезпечний словник та зберігає дані в порядку їхнього додавання бля безпечного проміжного зберігання транзакцій та гарантії обробки у порядку черги.

2.4 Принцип функціонування криптовалютної транзакції

В основі функціонування криптовалюної системи, як в загальному і будь-якої іншої фінансової установи, лежить транзакція. У віпадку з криптовалютою – транзакція виступає в ролі контракту між двома електроннимим гаманцями з метою, передаючи у права власності певну кількість електорнних монет з одного електронного сховища до іншого. Для створення та збереження криптовалюти використовуються дві основні структури: транзакція і блок. Транзакція є ключовою одиницею всієї системи. У контексті системи криптовалюти, інформація про криптовалютні одиниці не зберігається відкрито, але представлена у вигляді взаємодії кількох структур, де транзакція є основною та найпростішою одиницею.

Транзакції, щоб отримати підтвердження від системи, спочатку потрапляють до сховища валідатора – пул неопрацьованих транзакцій. У порядку черги з цього сховища відбирається певна кількість транзакцій для формування блоку. Транзакції з цього блоку хешуються та зберігаються в базі даних у захищеному вигляді. Ці блоки утворюють ланцюг даних, де кожен новий блок створюється, використовуючи дані з попереднього ланцюга, що забезпечує цілісність системи та захист від неправомірного втручання.

На життєвому циклі транзакції, який представлено на моделі послідовностей (рис. 2.12), показані основні етапи, такі як емісія (реєстрація певної кількості одиниць у системі), потрапляння до електронного гаманця користувача та використання криптовалюти у переказі на інший гаманець. Ці процеси ілюструють шлях транзакції від її створення до підтвердження системою, дозволяючи користувачам взаємодіяти з криптовалютною системою у зручний та безпечний спосіб.

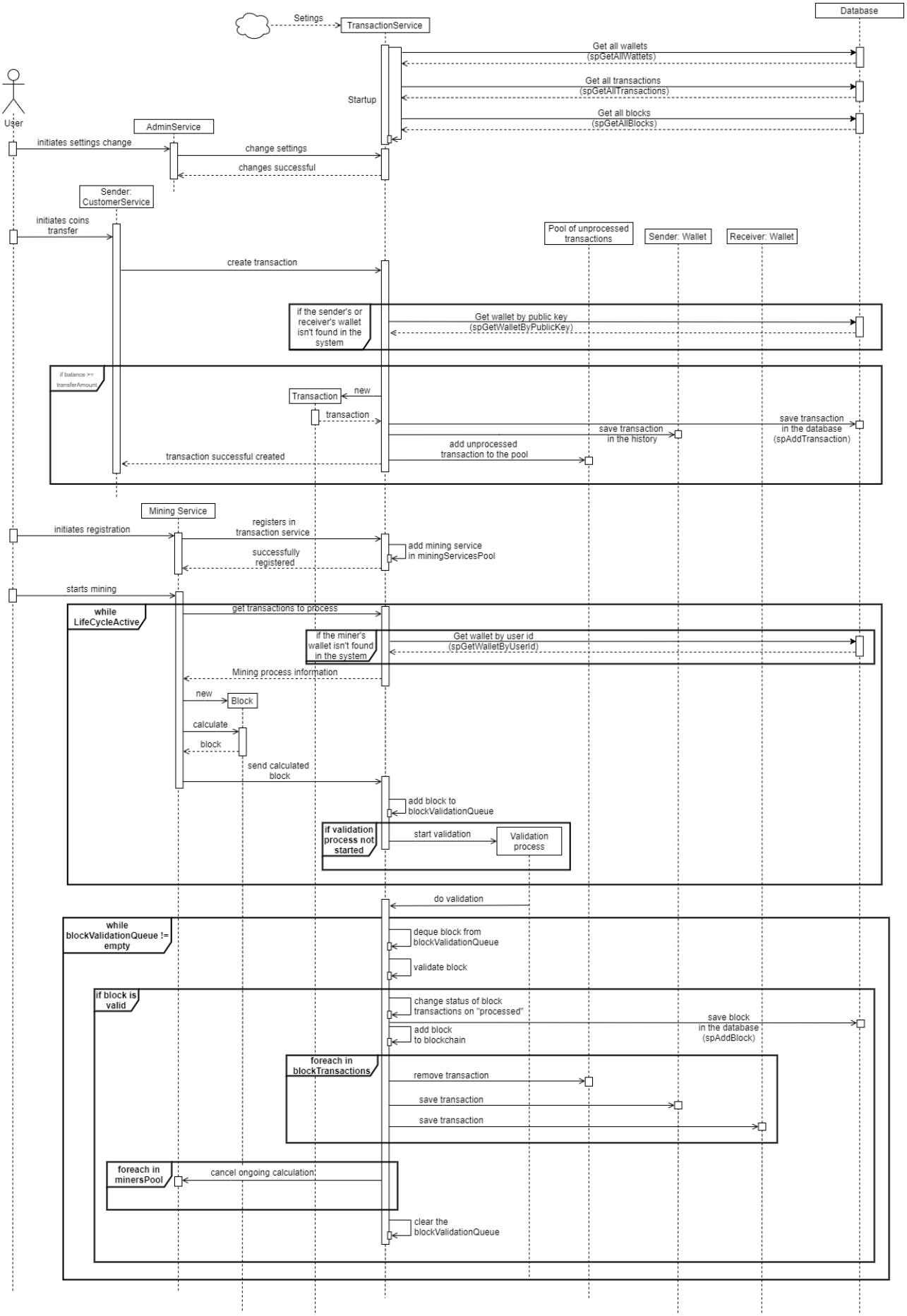

Рисунок 2.12 – Модель послідовностей функціонування системи

2.5 Висновки

У другому розділі було розглянуто архутектурні підходи для розробки програмного забезпечення з обробки криптовалютних транзакцій. На основі аналізу основних підходів для побудови архітектури програмної системи було запропоновано використання комбінованого архітектурного підходу на основі мікросервсного, шарового та Event-Driven підходів до побудови архітектури програмної системи. Архітектуру найвищого рівня описано на основі мікросервісної архітектури. Внутрішня складова кожного з мікросервісів описана шаровим архітектруним підходом. Взаємодію критичних вузлів мікросервісів окрім використаня класичних server-to-server API запитів визначатиме Event-Driven підхід.

Також розглянуто детальну класову схему кожного з мікросервісів. Класова схема визначає структуру внутрішніх копонентів та шляхи взаємодії між ними, що надає загальне уявлення системи та спрощує подальшу програмну реалізацію криптовалютної системи.

Як було зазначено в основі будь-якох криптовалютної системи лежить транзакція. Отож на основі цього було сформовано послідовніть життевого циклу транзакції, від ініціації створення користувачем через сервіс взаємодії з електронним гаманцем, до обробки сервісом транзакцій з подальшим хешуванням та збереженням у сховищі даних. На основі схеми послідовностей можна отримати загальне уявлення про порядок внутрішнього процесу у криптовалютній системі, що дає змногу бачаити весь процес від початку до кінця, що дозволить полегшити процес розробки.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КРИПТОВАЛЮТНОЇ СИСТЕМИ

3.1 Обгрунтування вибору інструментів для розробки

3.1.1 Вибір мови програмування для реалізації бізнес-логіки та серверної частини програмної системи

На сучасному етапі розвитку інформаційних технологій існує невичерпний арсенал загальнопризначених мов програмування, кожна з яких займає визначену нішу в різноманітних сферах технологічного простору. Одним із прикладів лідерства в конкретній галузі є C++ [10], який беззаперечно виступає абсолютним лідером у сфері ігрової індустрії. У випадку розробки веб-сервісів, зокрема back-end розробки, Java [11] та C# [12] виходять на передовий план як найпопулярніші мови програмування. Для розробки під платформу Windows надзвичайно популярним стає C#, тоді як Java залишається вибором для розробки під іншими операційними системами.

З'явлення .Net Core [13] і подальший розвиток у вигляді .Net 7 дозволяє мові C# розширити свою нішу і охопити кросплатформенний сегмент. Основні переваги C# включають приємний та зручний синтаксис, а також функціональну базову бібліотеку, що забезпечує необхідні інструменти для швидкої розробки.

Розглядаючи абсолютне лідерство платформи .Net для розробки під Windows та обіцянки кросплатформенного розвитку, ми вибрали мову C# для розробки серверної частини системи, бізнес-логіки та додатку для емісії криптовалюти. Це вибір, здійснений на фундаменті перспектив та потужностей .Net екосистеми, відображає стратегічний підхід до створення високоефективного та масштабованого рішення.

3.1.2 Вибір середовища розробки

Visual Studio [14] – це неоспоримий лідер у сфері розробки додатків з використанням мови програмування C#. Це інтегроване середовище розробки надає один із найкращих інструментів для автодоповнення коду під час розробки,

завдяки системі IntelliSense, яка в останніх версіях зазнала значних покращень в напрямку передбачення того, що слід написати. Використання Visual Studio дозволяє розробникам зосередитися на самому процесі написання коду, забезпечуючи продуктивне та комфортне робоче середовище.

Окрім того, Visual Studio пропонує зручні інструменти для тестування, дебагінгу та публікації додатків, допомагаючи вдосконалити процес розробки та підтримувати високий рівень якості програмного продукту. Вбудований інструмент для інсталяції пакетів NuGet дозволяє ефективно керувати сторонніми пакетами, які використовує додаток, спрощуючи процес управління залежностями та забезпечуючи легкість внесення змін у конфігурацію проекту. Такий підхід до розробки дозволяє максимально оптимізувати час та ресурси, що витрачаються на створення високоякісного програмного забезпечення.

3.1.3 Вибір системи управління базою даних

Для забезпечення ефективного збереження даних у системі ми визначилися з використанням бази даних MS SQL Server [15]. Ця реляційна база даних надає можливість створювати зв'язки між різними сутностями та підтримувати цілісність усіх даних, забезпечуючи високий рівень організації та структури в базі даних.

Вибір MS SQL Server обумовлений не лише його реляційною природою, але й тісною інтеграцією з платформою .Net. Цей ключовий фактор стає важливим у контексті обраних інструментів для розробки системи, сприяючи гармонійній взаємодії та забезпечуючи однорідний розвиток програмного продукту.

Крім того, MS SQL Server відомий своєю високою продуктивністю та надійністю. Ці характеристики роблять його оптимальним вибором для системи, де важлива швидкодія та стійкість в роботі бази даних. Такий підхід сприяє оптимізації роботи системи та забезпечує безперебійну роботу зберігання та обробки інформації.

3.1.4 Вибір інструментів для роботи з базою даних

Існують два основні варіанти взаємодії з базою даних: виклик збережених

процедур [15] та використання прямих SQL-запитів [17]. Давайте детальніше розглянемо кожен із цих варіантів.

1. Виклик збережених процедур:

Збережені процедури представляють собою набір інструкцій для взаємодії з базою даних, який обгорнуто в єдину логічну одиницю. Це забезпечує структурованість та узгодженість в роботі з базою даних, спрощуючи процес розробки та підтримки. Виклик збережених процедур дозволяє використовувати підготовлені та оптимізовані запити, що може призвести до покращення продуктивності системи.

2. Виклик прямих SQL-запитів:

В інших випадках, коли потрібно виконати конкретний SQL-запит, розробники можуть використовувати прямі SQL-запити. Це дає більший контроль над діями з базою даних, але може призвести до меншої читабельності коду та більшого обсягу написаного коду.

Обираючи між цими варіантами, потрібно враховувати потреби проекту, його масштаб та вимоги до продуктивності, щоб забезпечити ефективну та оптимізовану взаємодію з базою даних.

Переваги використання збережених процедур:

1. Зростання продуктивності за рахунок кешування – Збережена процедура будує план виконання та кешує його, що призводить до покращення продуктивності через швидше виконання.

2. Вирішення проблеми SQL-ін'єкції – Використання параметрів в одному об'єкті дозволяє уникнути SQL-ін'єкцій та підвищує безпеку системи.

3. Зменшення використаного трафіку – Виклик збереженої процедури передається як виклик функції з параметрами, зменшуючи обсяг передаваного трафіку.

4. Можливість налаштування доступу – Доступ до збережених процедур може бути обмежений для різних користувачів, що забезпечує гнучкість у керуванні правами користувачів.

5. Повторне використання коду – Зміни в збереженій процедурі не вимагають модифікацій у точках виклику, що полегшує підтримку та розвиток.

6. Відокремлення запитів від бізнес-логіки Запити у збережених процедурах існують окремо від бізнес-логіки додатку, полегшуючи розподіл обов'язків.

7. Можливість повертати більше декіклька резлтьтів на один виклик – Збережені процедури можуть повертати більше одного значення через зовнішні параметри.

8. Немає необхідності перекомпілювати весь проект – Зміни у збереженій процедурі не вимагають перекомпіляції всього проекту, що спрощує розвиток системи.

9. Простіше оптимізувати – Збережені процедури можуть бути ефективно оптимізовані для поліпшення продуктивності.

Недоліки використання збережених процедур:

1. Ускладнений контроль версій – Управління версіями може бути складнішим, оскільки код збережених процедур потрібно десь окремо зберігати, щоб мати доступ до нього поза базою даних.

Прямі SQL-запити представляють собою використання SQL-коду, який передається у вигляді тексту у виклику до бази даних. Цей метод дозволяє розробникам докладно контролювати структуру SQL-запитів та маніпулювати ними на пряму. Давайте розглянемо деякі переваги та недоліки використання прямих SQL-запитів.

Переваги використання прямих SQL-запитів:

1. Гнучкість та контроль – Розробник має повний контроль над створенням SQL-запитів, що дозволяє робити їх більш гнучкими та оптимізованими за конкретними потребами проекту.

2. Простота в розробці – Прямі SQL-запити можуть бути простими для використання та розробки, особливо якщо є необхідність в швидкому реагуванні на вимоги проекту.

Недоліки використання прямих SQL-запитів:

1. Вразливість до SQL-ін'єкцій – Якщо не використовувати параметризовані запити, є ризик вразливості до SQL-ін'єкцій, що може викликати проблеми з безпекою.

2. Складніше відлагодження – У порівнянні з використанням збережених процедур, відлагодження та тестування прямих SQL-запитів може бути складнішим.

3. Менше оптимізації – Прямі SQL-запити можуть бути менш оптимізованими, оскільки база даних не може підготувати та закешувати план виконання як у випадку збережених процедур.

4. Зростання трафіку – запит надсилається як текст з усією логікою, що може сягати чималого розміру.

5. Запити є частиною кодової бази – Це ускладнює підтримку з розширенням програмної системи.

6. Кожна зміна запиту вимагає додаткової перекомпіляції проетку – Це призводить до надлишкових витрат часу.

Отже використання збережених процедур може сприяти покращенню продуктивності, безпеки та гнучкості вашого додатку. Є можливість ефективно керувати доступом до бази даних для різних користувачів, уникнути SQL-ін'єкцій, а також забезпечити простоту відлагодження та тестування.

Додатково, використання збережених процедур може полегшити роботу з базою даних у великих та складних проектах, оскільки вони дозволяють можливість розділити логіку бази даних від логіки додатку.

Для взаємодії з базою даних та виклику збережених процедур у даному випадку вирішено обрати micro-ORM Dapper. Розглянемо деякі ключові аспекти, які вплинули на це рішення:

1. Продуктивність – Dapper славиться своєю високою продуктивністю. Він пропонує ефективний спосіб взаємодії з базою даних та оптимізовані запити, що дозволяє забезпечити швидкий доступ до даних.

2. Простота використання – Dapper спрощує взаємодію з базою даних, забезпечуючи лаконічний та зрозумілий синтаксис. Це особливо важливо для швидкої розробки та обслуговування коду.

3. Підтримка збережених процедур – Dapper надає зручні інструменти для виклику збережених процедур. Це важливо для структуризації та використання бізнес-логіки бази даних безпосередньо в коді додатку.

4. Гнучкість – Використання Dapper дозволяє розробникам зберігати контроль над SQL-запитами, що використовуються в додатку, та оптимізувати їх за необхідності.

5. Низькорівневі функції та загальна продуктивність – Dapper, як micro-ORM, надає певні низькорівневі функції, що дозволяють здійснювати контроль над виконанням SQL-запитів, що може бути важливим для певних вимог проекту.

Обираючи Dapper для роботи з базою даних – обирається інструмент, який поєднує у собі гнучкість та продуктивність, що є важливим для ефективної розробки програмної системи.

3.1.5 Вибір інструментів для розробки веб-клієнта для роботи з електронним гаманцем

При створенні браузерного клієнта в Asp.Net [18], вирішено використовувати двигун рендерингу Razor [19]. Цей інструмент пропонує зручний спосіб створення сторінок на стороні сервера, об'єднуючи синтаксис HTML та C#.

Основні переваги використання Razor:

1. Змішаний синтаксис – Razor дозволяє зручно комбінувати HTML [20] та C# прямо на сторінці. Це спрощує використання моделей, написаних мовою C#, безпосередньо в HTML-коді.

2. Допоміжні інструменти – Asp.Net надає допоміжні інструменти, які прискорюють виконання запитів до контролера. Це дозволяє зручно взаємодіяти з серверною частиною додатку та спрощує розробку.

3. Зручність та швидкість – Створення сторінок за допомогою Razor є ефективним та зручним процесом. Сполучення HTML та C# дозволяє швидко розробляти та підтримувати код.

Для дизайну сторінки використовано популярну CSS-бібліотеку Bootstrap [21]. Bootstrap не лише надає стиль та естетичний вигляд, але й забезпечує адаптивність інтерфейсу.

Основні характеристики використання Bootstrap:

1. Адаптивний дизайн – Bootstrap дозволяє легко створювати адаптивні інтерфейси, що коректно виглядають на різних пристроях та розмірах екранів.

2. Швидка розмітка – З використанням готових елементів Bootstrap можна швидко втілити базовий дизайн без додаткового написання CSS-коду.

3. Розширені компоненти – Bootstrap має багатий набір готових компонентів, що дозволяє легко додавати елементи, такі як кнопки, навігація, таблиці та інші.

Використання Razor та Bootstrap у поєднанні дозволяє забезпечити не тільки функціональний, але й стильний та адаптивний користувацький інтерфейс для браузерного клієнта вашого Asp.Net додатку.

3.2 Реалізація модулів для виконання запитів до бази даних

Приймаючи до уваги розроблену архітектуру, DAL надає зручний функціонал для бізнес-логіки інших сервісів у вигляді інтерфейсу для виклику відповідних методів. Кожен з інтерфейсів має власну реалізацію, що викликає методи класу, який реалізує базову логіку використовуючи Dapper.

Основним класом є SqlDataAcess, що пропонує 3 методи для виклику бази даних:

- Query – запит на вибірку даних, що повертає дані лише однієї існуючою, або сформованої у результаті об’єднань таблиці.

- QueryMultiple – дозволяє повернути результат одразу із кількох таблиць.

- Execute – виконує виклик збереженої процедури для викнання змін у таблицях бази даних та не очікує повернення будь-яких даних.

Кожен з цих методів використовується більш високорівневими модулями для надання зручного інтерфейсу використання. Розглянемо приклади використання у високорівневих модулях кожного з цих методів.

Для прикладу метод, що зберігає нового користувача у системі при реєстрації отримує в якості параметра обрікові дані коритсувача та формує виклик збереженої процедри через Query метод. Загальна блок-схема методу збереження користувача продемонствроано на рисунку 3.1.

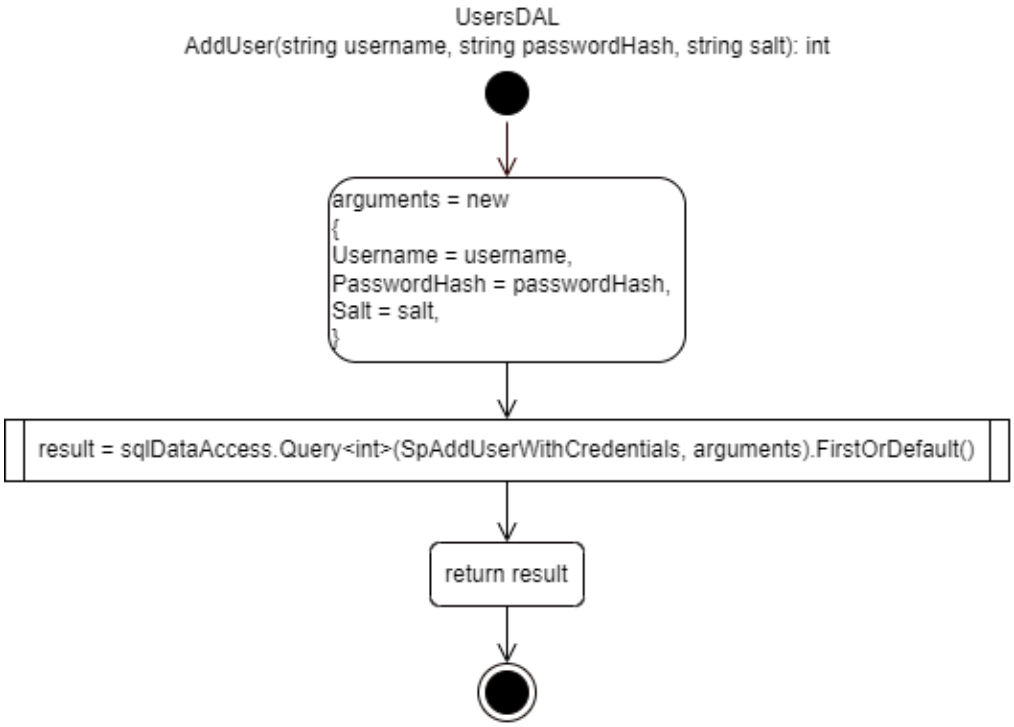

Рисунок 3.1 – Блок-схема алгоритму збереження нового користувача у системі

Інший приклад – отримання облікових даних користувача по вказаній електронній адресі. У цьому приклді для викліку використовується метод QueryMultiple для отримання даних з кількох таблиць. На рисунку 3.2 продемонстрвоано блок-схему алгоритму для отримання облікових даних користувача.

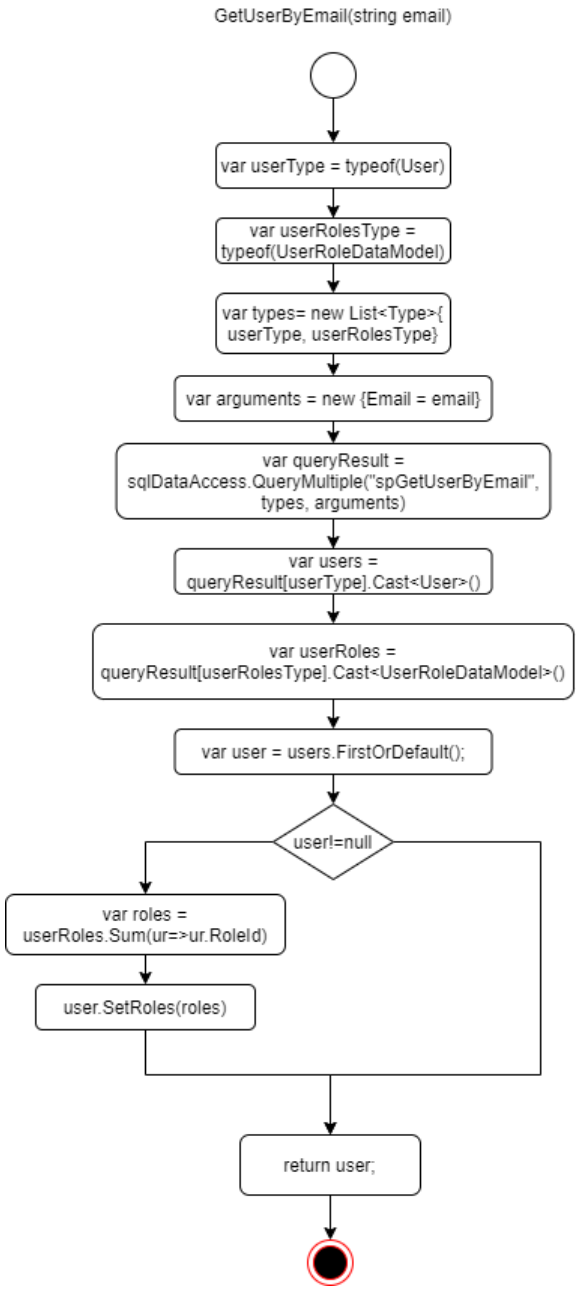

Рисунок 3.2 – Блок-схема алгоритму для отримання облікових даних існуючого користувача.

3.3 Розробка сервісу обробки криптовалютних транзакцій

TransactionService є одним із ключових сервісів у системі, функції якого включають в себе валідацію та обробку даних, що надходять від клієнтських сервісів через API. Для забезпечення доступу до логіки, сервіс надає API, побудоване на ASP .NET WebAPI Core [22] з використанням архітектури REST [23].

1. BlochainController – Дозволяє додавати нові блоки до системи, які отримуються в результаті обчислень у клієнтському додатку BlockchainService.
2. BlockcahinServiceController – Надає інформацію для обчислення нового блоку за запитом від клієнтського додатку BlockchainService.
3. TransactionController – Дозволяє створювати нові транзакції за запитом від клієнтського веб-додатку CustomerService.
4. WalletController – Надає доступ до інформації про користувацькі електронні гаманці за запитом від клієнтського веб-додатку CustomerService.

Кожен контролер функціонує як посередник для відповідного типу запиту. Кожен метод контролера викликає відповідний шар бізнес-логіки, що розподілений між різними модулями. Це підходить систему більш гнучкою та простою у використанні.

Оскільки .NET Core додатки володіють вбудованим DI Container, можливість змінювати використовувані модулі в контролерах (і не лише) стає дуже простою. Достатньо змінити лише 1 стрічку коду, яка відповідає за реєстрацію модуля у DI Container.

Система організована так, що виклики бізнес-логіки у контролерах є лише викликами методів, що розподілені між різними модулями. Це полегшує розробку, підтримку та масштабування системи, роблячи її більш гнучкою та адаптивною до змін у бізнес-логіці.

3.3.1 Розробка контролера для роботи з блоками

Контролер Blockchain має простий метод AddBlock, який приймає новий блок у якості параметра. Ця функція відповідає за додавання нових блоків до системи, які отримуються в результаті обчислень у клієнтському додатку BlockchainService.

Метод AddBlock є HTTP POST-методом, що дозволяє відправляти блоки у тілі запиту в захищеному форматі через розширений протокол HTTPS, який підтримує шифрування для забезпечення безпеки передачі даних.

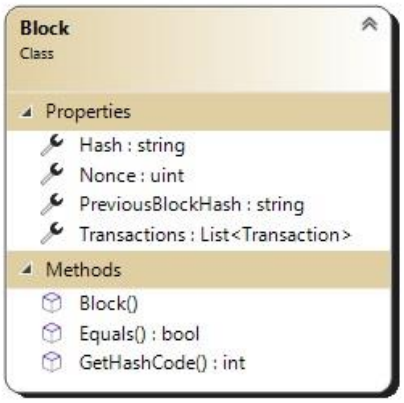

Рисунок 3.7 – Структура моделі Block

Сам метод має просту структуру, розбиту на два логічні блоки:

1. Виклик асинхронного методу бізнес-логіки:

- Викликає асинхронний метод бізнес-логіки, який представлений інтерфейсом `IBlockchainServiceHost`.
- В даному випадку, цей метод викликає метод AddBlockAsync, який реалізований у модулі TransactionService.
- Забезпечує асинхронний виклик та виконання бізнес-логіки.

2. Повернення результату в уніфікованому форматі:

- Повертає результат операції у стандартизованому форматі, представленому моделлю `ResponseResult`.
- Модель містить інформацію про статус операції та повідомлення, що може містити інформацію про виконану операцію.

Метод AddBlock ініціює додавання блоку до черги та запускає процес валідації, якщо черга була пустою перед викликом. Для забезпечення безпеки та уникнення виникнення виключень при одночасних додаваннях блоків використовується потокобезпечна колекція ConcurrentQueue. Це гарантує, що декілька запитів може оброблятися одночасно без конфліктів та збереження цілісності даних в черзі.

Для ефективного управління можливою повторною ініціацією процесу валідації через одночасні виклики у кількох потоках методу AddBlock, який був описаний раніше, використовується патерн Double-Checked Locking.

Принцип роботи цього патерну ілюструє діаграма на рисунку 3.10. Назва патерну вказує на те, що для ініціації процесу валідації необхідно пройти дві перевірки та заблокувати виконання інших потоків через монітор. Обидві перевірки мають впевнитися, що процес валідації ще не був розпочатий.

Ключовий момент полягає в тому, що перед другою перевіркою необхідно заблокувати інші потоки. Таким чином, коли починається друга перевірка, умови будуть відхиляти ініціативу розпочати інший паралельний процес валідації, і інші потоки будуть просто проходити через іншу гілку без блокування.

Цей підхід гарантує безпеку виконання процесу валідації при одночасних викликах та забезпечує ефективне управління ресурсами.

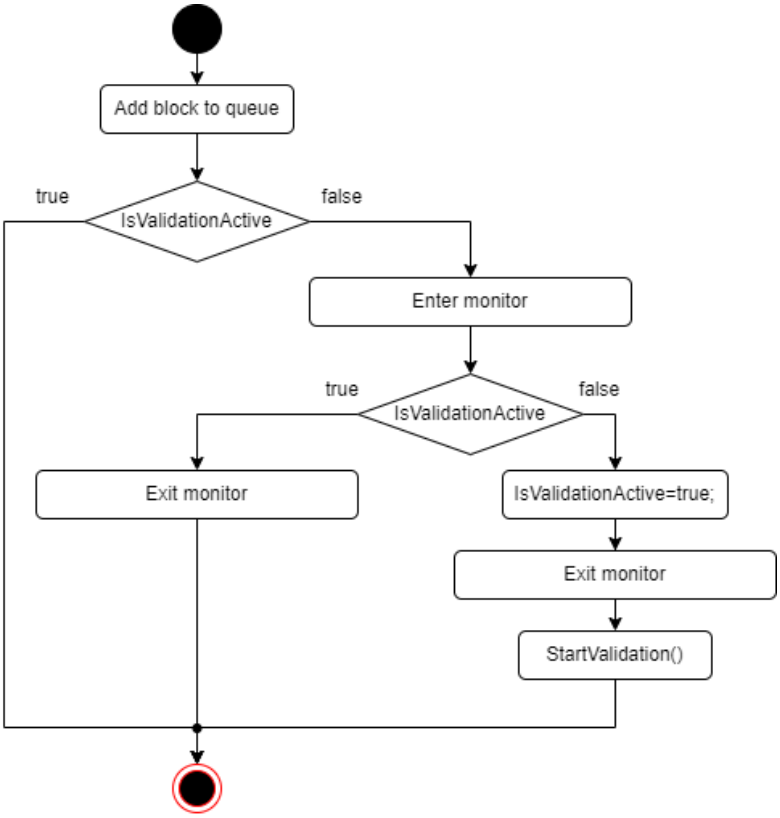

Рисунок 3.10 – Використання Double-checked Locking паттерну при додаванні нового блоку

Наступним кроком у функціонуванні системи є початок процесу валідації, де відбувається обробка черги блоків, і система шукає валідний блок серед тих, що присутні у черзі.

Процес валідації запускається з витягування одного блоку з черги колекції. Для цього блоку перевіряються всі його поля на відповідність умовам, при яких блок може бути вважений валідним. Це включає перевірку наявності коректного хешу, правильність транзакцій та інші умови, залежно від особливостей конкретної системи.

Алгоритм цього процесу валідації блоку ілюструється на рисунку 3.11, де кроки кожної ітерації валідації детально розглядаються.

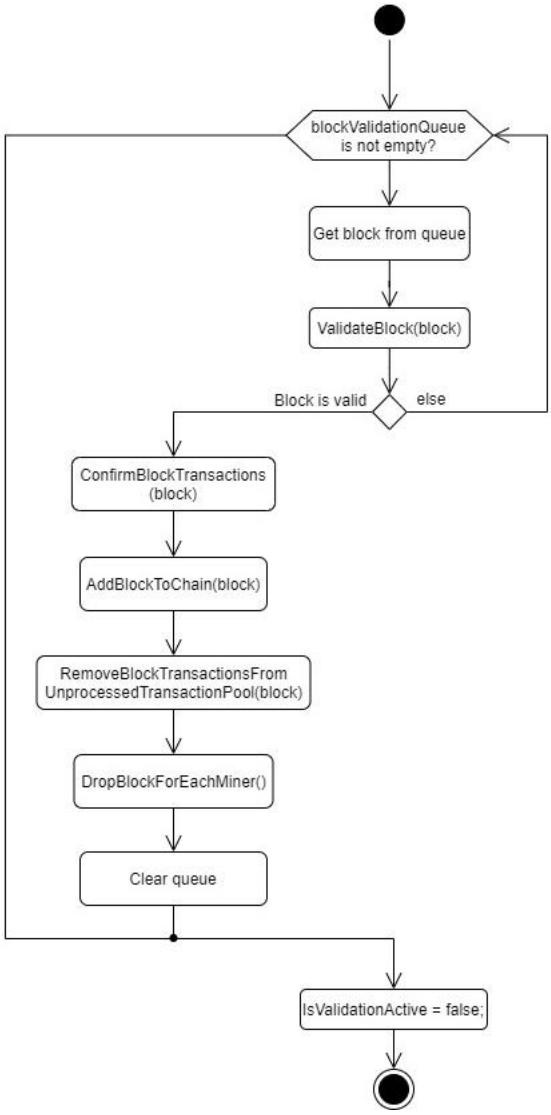

Рисунок 3.11 – Діаграма активності для процесу валідації черги блоків

Цей етап є ключовим у забезпеченні цілісності та безпеки системи, оскільки лише валідні блоки отримують можливість бути доданими до ланцюжка блоків.

Перш за все, кожен блок повинен пройти ретельний процес валідації, що включає перевірку всіх даних, з яких складається блок. Цей процес виконується наступним чином:

- Перевірка поля хешу останнього збереженого блок. Поле, що містить хеш останнього блока в системі, піддаватиметься увазі наступним чином. Якщо значення цього поля не збігається із фактичним хешем останнього блока у системі, то блок вважатиметься не валідним. Навіть якщо інші поля блока задовольняють усі умови, він буде виключений із подальшого процесу валідації. Ця перевірка гарантує коректність ланцюжка та узгодженість хешів між блоками.
- Перевірка правильності обрахунку хешу. Другий критерій валідації передбачає порівняння збереженого в блоку хешу із значенням, яке система отримує після повторного обчислення хешу. Для цього використовуються всі поля блоку. Якщо значення хешу не збігається, блок вважається не валідним. Ця перевірка гарантує, що дані в блоку не були спотворені і що хеш був правильно обчислений.
- Перевірка формату хешу. Ця умова валідації передбачає перевірку формату хешу, який повинен задовольняти певний набір символів. Загалом, це конкретна послідовність символів, з якої повинен складатися хеш блока. Якщо хеш не відповідає цьому формату, блок вважається не валідним. Ця перевірка забезпечує додатковий рівень безпеки та визначає конкретні вимоги до хешу блока для вважання його валідним..
- Валідація транзакцій. У цьому етапі проводиться валідація транзакцій, яка включає два типи транзакцій – зазвичай це системні транзакції та транзакції, які знаходяться у пулі необроблених транзакцій. Кожна невикористана транзакція тимчасово розміщується у спеціальній колекції, відомій як «Пул неопрацьованих транзакцій». Під час валідації блока перевіряється, чи містить він лише транзакції, які знаходяться у пулі. Єдиним винятком є системна транзакція, яка є результатом емісії криптовалюти. Оскільки блок може містити два типи транзакцій, їх валідація проводиться окремо для кожного типу, забезпечуючи коректність та стабільність системи:

- Валідація системної транзакції. Кожен блок обов'язково повинен містити системну транзакцію, інакше він не може вважатися валідним. Під час валідації системної транзакції застосовуються додаткові умови. По-перше, системна транзакція повинна бути створена з системного гаманця, до якого неможливо отримати ручний доступ. По-друге, системна транзакція повинна бути адресована гаманцеві, власник якого створив новий блок за допомогою BlockchainService. Алгоритм валідації системної транзакції наведено на рисунку 3.12, де крок за кроком показано процес перевірки відповідності системної транзакції умовам для валідності блока.

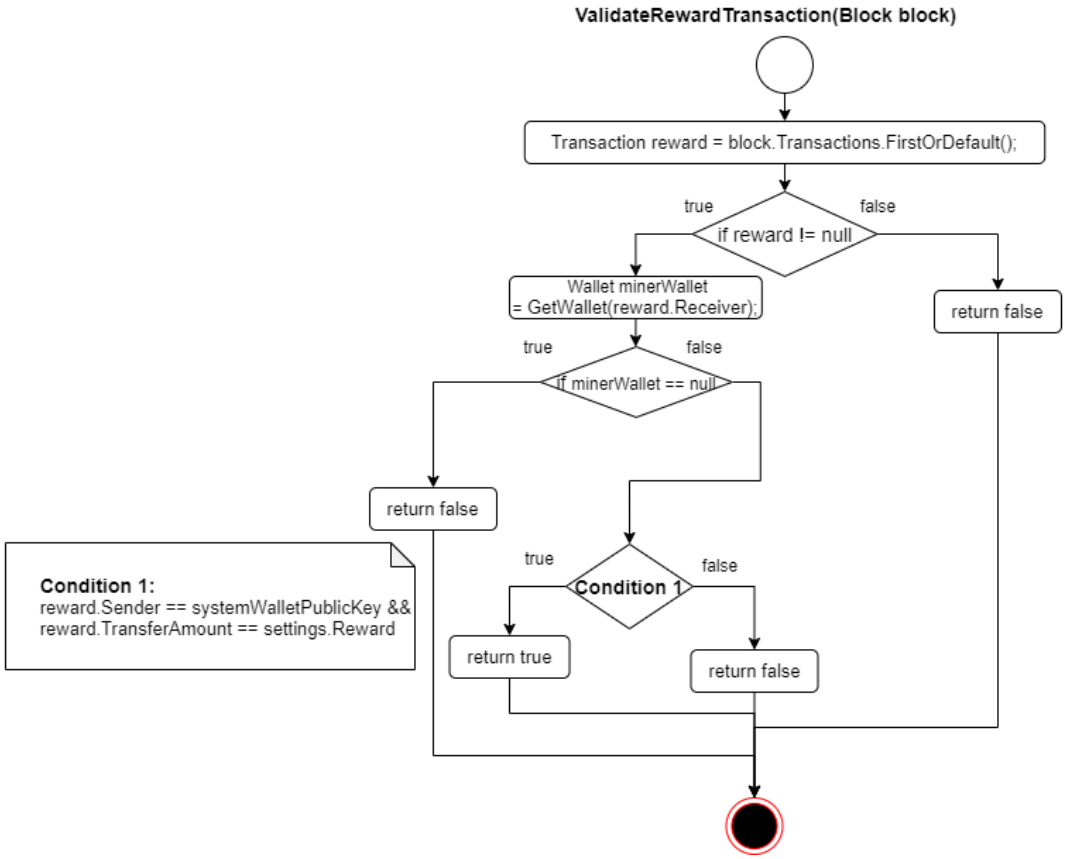

Рисунок 3.12 – Валідація системної транзакції

- Валідація звичайних транзакцій здійснюється іншим шляхом. Під час валідації перевіряється цілісність даних, тобто, чи не було змінено транзакції, чи не було додано дублікатів і т.д. Основним умовою є те, щоб транзакції блоку перебували у пулі на момент валідації. Алгоритм валідації

транзакцій наведено на рисунку 3.13, де ілюструється послідовність перевірок для забезпечення правильності та валідності звичайних транзакцій.

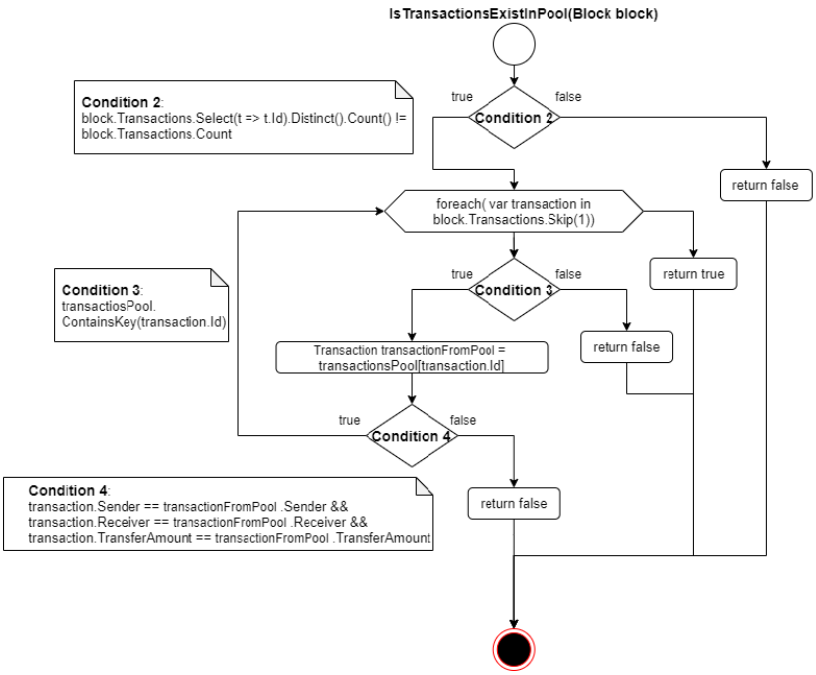

Рисунок 3.13 – Діаграма активності для валідації транзакцій

Якщо блок успішно пройшов всі етапи валідації, це свідчить про його валідність, і його слід додати до ланцюга блоків та зберегти у базі даних. Для цього блоку система підтверджує всі транзакції та видаляє їх з пулу. Усі зареєстровані сервіси майнингу отримують сповіщення для оновлення даних, при цьому перераховується хеш блоку. Останнім кроком є очищення черги, оскільки після додання нового блоку всі інші блоки в черзі не зможуть пройти валідацію хешу попереднього блоку. Таким чином, для залишених блоків у черзі валідація вже не має сенсу.

3.3.2 Розробка контролера для роботи з сервісом блокчейну

Контролер, як описано на початку, відповідає за отримання даних для обрахунку нового блоку. Цей контролер також має лише один метод. У порівнянні з попереднім, тип цього методу - GET, оскільки інформація, яка надається з

запитом, не є критичною і не має впливу на працездатність системи в цілому. Логічна структура методу не має відмінностей від описаного попереднього прикладу, тому перейдемо одразу до бізнес-логіки, за яку також відповідає інтерфейс IBlockchainServiceHost.

На цей раз розглянемо лише один метод GetMiningProcessInformation, який формує і повертає необхідні дані для нових обрахунків. На рисунку 3.16 наведено діаграму процесу формування даних для обрахунку нового блоку.

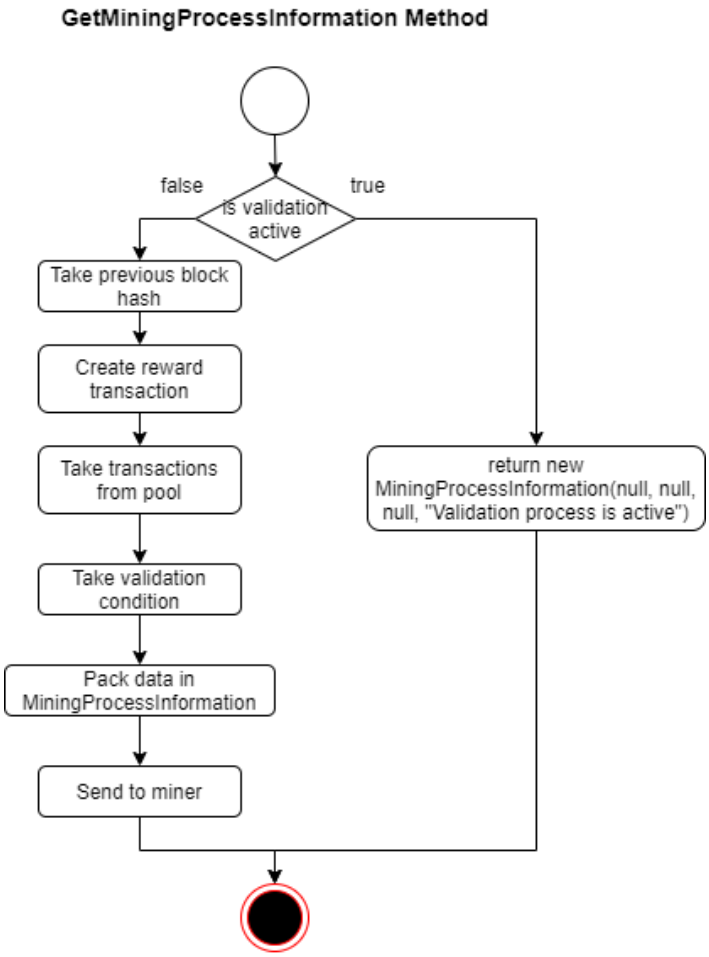

Рисунок 3.16 – Блок-схема методу GetMiningProcessInformation

Узагальнюючи, як видно на діаграмі, метод повинен спочатку перевірити, чи на даний момент не відбувається валідація блоків. У разі позитивної відповіді, він поверне повідомлення про те, що валідація в процесі, і рекомендує виконати повторний запит після завершення. Це обумовлено тим, що під час валідації ймовірно додавання нового блоку у систему, що призведе до оновлення інформації

для обчислення нових блоків. Враховуючи, що обчислення блоків із застарілими даними неефективне, чекати завершення валідації є більш раціональним вибором.

Коли валідація завершиться, або якщо її не розпочато, метод має отримати поточний хеш останнього доданого до системи блоку. Це значення використовується як зв'язуючий елемент для кожного блоку, що дозволяє сформувати логічну послідовність. Наступним етапом є створення системної транзакції, що виступає роль емісії криптовалюти. Отримувач транзакції визначається за допомогою ідентифікатора користувача, переданого у запиті. Потім формується список транзакцій, які очікують обробки у відповідному сховищі. Кількість транзакцій у блоці зазвичай обмежена, але може бути розширена за необхідності, змінюючи значення у налаштуваннях. Завершальним етапом є встановлення умов валідності хешу нового блоку та упакування сформованих даних у модель для подальшого використання.

3.3.3 Розробка контролера для роботи з транзакціями

Описаний раніше контролер відповідає за обробку запитів, спрямованих на створення транзакцій. Під час створення транзакції метод очікує надходження таких параметрів, як публічний ключ відправника, публічний ключ отримувача та сума переказу.

Вхідні параметри методу для створення транзакції:

1. Публічний ключ відправника – це унікальний ідентифікатор, який вказує на гаманець або адресу відправника транзакції.

2. Публічний ключ отримувача – аналогічно, це ідентифікатор гаманця або адреса отримувача, куди буде направлено переказану суму.

3. Сума переказу – кількість криптовалюти або іншого активу, яку відправник хоче переказати отримувачу.

Цей метод визначає логіку обробки переданих параметрів, зокрема виконує валідацію та інші перевірки, які необхідні для створення коректної транзакції. Після обробки запиту та створення транзакції, метод може повертати відповідь, що містить інформацію про статус операції, таку як успішність створення транзакції

або виявлені помилки.

Отримані дані контролер передає далі у бізнес-логіку, де проводиться валідація даних. Під час цього етапу перевіряється можливість створення транзакції. Для успішного проходження валідації обов'язково вказати існуючі публічні ключі гаманців, а також переказувана сума не повинна перевищувати наявний баланс у відправника.

Якщо введені дані вказані правильно, транзакція створюється та зберігається у гаманці відправника як необроблена. Крім того, транзакція додається до спеціального сховища, так званого "пулу необроблених транзакцій", де кожна транзакція очікує на підтвердження. Це дозволяє відслідковувати та опрацьовувати транзакції під час подальшого оброблення.

Загальна схема взаємодії класів і сутностей наведено на рисунку 3.17

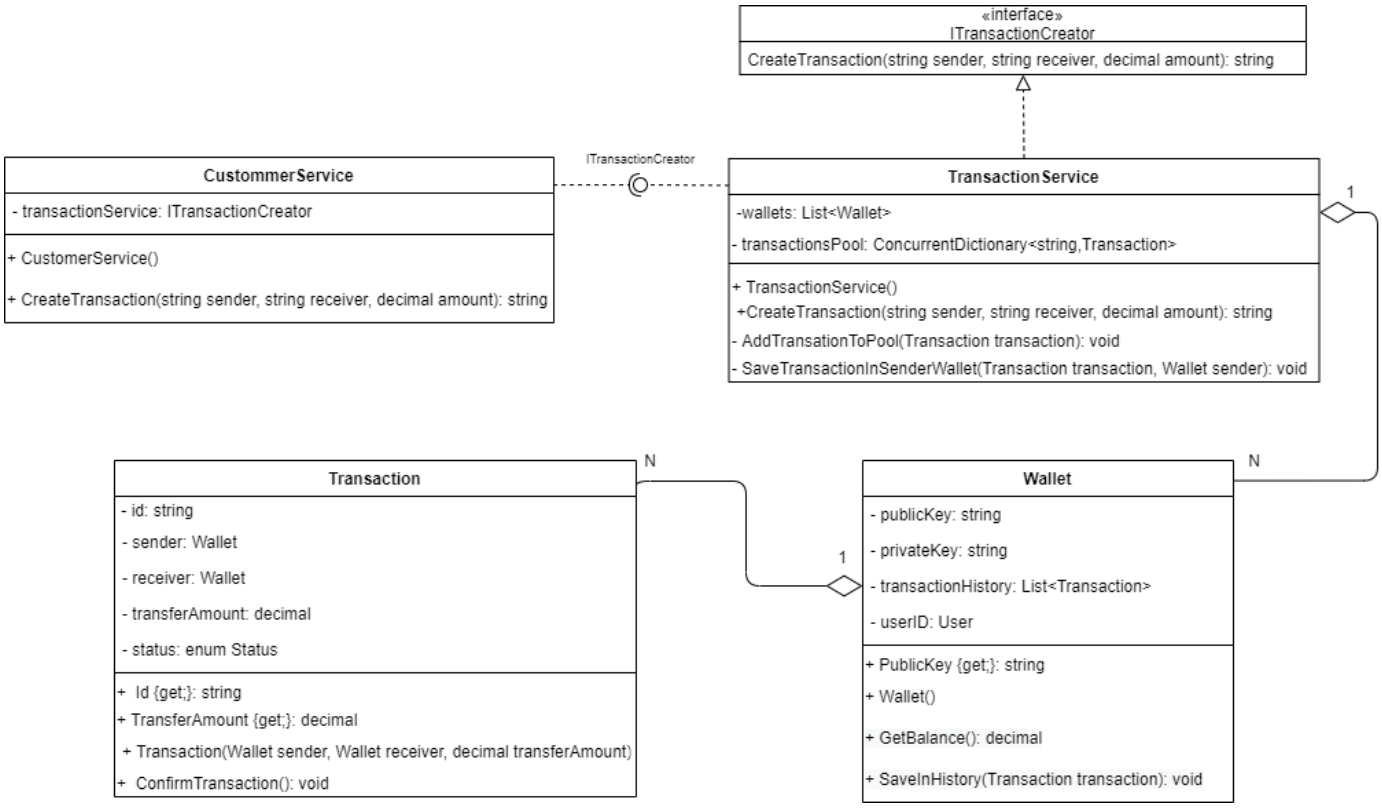

Рисунок 3.17 – Схема класів та їх взаємодія для сторення транзакціїї

3.3.4 Розробка контролера для роботи з електронними гаманцями

Цей контролер відповідає за обробку запитів, пов'язаних із роботою гаманця.

На відміну від попередніх контролерів, тут є декілька методів, які будуть описані нижче.

Метод бізнес-логіки, який викликається методом контролера, повинен спочатку спробувати отримати гаманець із сховища. Якщо виявляється, що гаманець відсутній, метод виконує спробу створити гаманець та зберегти його в базі даних. Зазвичай метод працює безперебійно, але для непередбачених сценаріїв створено додаткові перевірки, результати яких описуються у відповідних файлах про виявлені помилки.

Метод GetTransaction – це метод, що повинен повернути лише вказану транзакцію за її ідентифікатором.

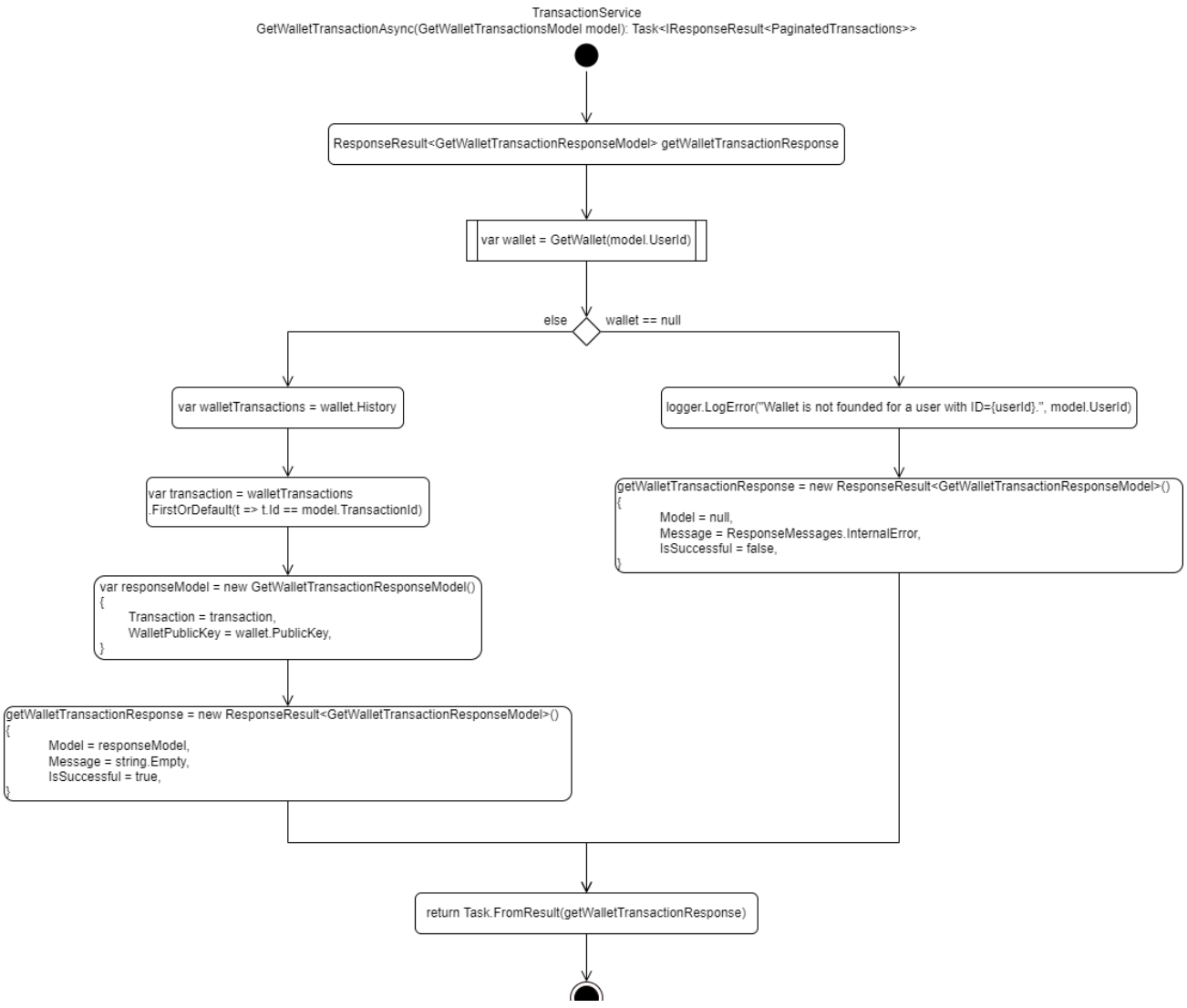

Рисунок 3.18 – Блок-схема методу для отримання транзакцій

Метод GetWallet викликає бізнес-логіку, яка створює гаманець, якщо для вказаного користувача гаманець відсутній, або повертає вже існуючий. Бізнес-логіка для отримання балансу працює за схожим сценарієм, як і отримання гаманця, але відмінність полягає в тому, що метод не створює новий гаманець, якщо для вказаного користувача гаманець не існує. Для існуючого гаманця метод отримає його із сховища та викличе окремий метод, який підрахує баланс із наявних транзакцій та поверне результат.

Метод GetBalance – це метод, що повинен повернути поточний баланс гаманця. Як і кожен метод будь-якого контролера, також він викликає бізнес-логіку, описану інтерфейсом IWalletProvider. Бізнес-логіка, яку викликає контролер, приймає до уваги лише транзакції гаманця користувача. І якщо цей гаманець містить вказану транзакцію – метод її поверне, в іншому випадку відповідь на запит буде пустою.

Метод GetTransactions – це метод, що повинен повернути всі транзакції гаманця у подрібненому форматі. Враховуючи, що на сторінці, де будуть відображені транзакції, буде використовуватись лише частина з них, метод бізнес-логіки за запитом повертає лише "сторінку" транзакцій, яку вказав користувач. Також окрім певної сторінки користувач може змогу задати певні фільтри, які також будуть враховані для формування кінцевого результату, наприклад, тип транзакції, період часу або лише певний гаманець, з якого або на який транзакція була створена, тощо. Реалізація методу бізнес-логіки наведено у додатку.

3.4 Розробка сервісу для роботи з електронним гаманцем

Основна мета цього сервісу – надавати авторизованим та аутентифікованим користувачам зручний доступ до реального часу інформації про їхні електронні гаманці, а також можливість створення нових транзакцій за умови наявності достатнього балансу.

Крім логіки обробки даних, сервіс також пропонує користувацький інтерфейс, через який користувач може ініціювати різні запити до системи, отримувати дані або створювати транзакції.

Сервіс включає наступні функціональні можливості, для яких створено користувацький інтерфейс та описано серверну частину, що безпосередньо обробляє всі користувацькі дії:

1. Реєстрація нових користувачів. Дозволяє користувачам створити обліковий запис, вказавши необхідні особисті дані та отримати доступ до сервісу.

2. Аутентифікація та авторизація існуючих користувачів. Забезпечує захищений вхід у систему для зареєстрованих користувачів, а також визначає їхні права доступу.

3. Користувацький інтерфейс для відображення та фільтрації даних про електронний гаманець. Надає зручний інтерфейс для перегляду та управління інформацією про гаманці, з можливістю використання різноманітних фільтрів.

4. Користувацький інтерфейс для створення транзакцій. Дозволяє користувачам ініціювати нові транзакції, вказуючи необхідні дані та отримуючи підтвердження їх виконання.

Цей сервіс спрямований на надання користувачам зручного та безпечного інструменту для взаємодії з їхніми електронними гаманцями, забезпечуючи широкий спектр можливостей через інтуїтивно зрозумілий інтерфейс.

3.4.1 Реєстрація, аутентифікація та авторизація користувачів

Механізм аутентифікації ґрунтується на використанні CookieAuthentication [24], забезпечуючи ефективну взаємодію між веб-клієнтом та сервером. Основна архітектура веб-додатка, відповідно до специфікації OWIN [25] (Open Web Interface for .NET), включає такі компоненти:

Хост – це процес, що виконує серверний додаток. Його основне завдання - запустити додаток та завантажити компоненти OWIN.

Сервер – це додаток, який відповідає за реалізацію специфікації OWIN. Основне призначення - виконання запитів від клієнта та їх обробка через конвеєр OWIN.

Middleware – це проміжні компоненти структури, через які проходять обробку запити від клієнтського додатку.

Клієнт – це сам додаток, через який користувачі взаємодіють із системою.

Для забезпечення доступу до акаунту викликається метод бізнес-логіки Login. Цей метод виконує ряд перевірок для введених користувачем даних, приймаючи рішення щодо надання доступу до акаунту. Робочий алгоритм методу наведено на рисунку 3.26, де визначено кроки перевірки та прийняття рішення на разунок авторизації користувача.

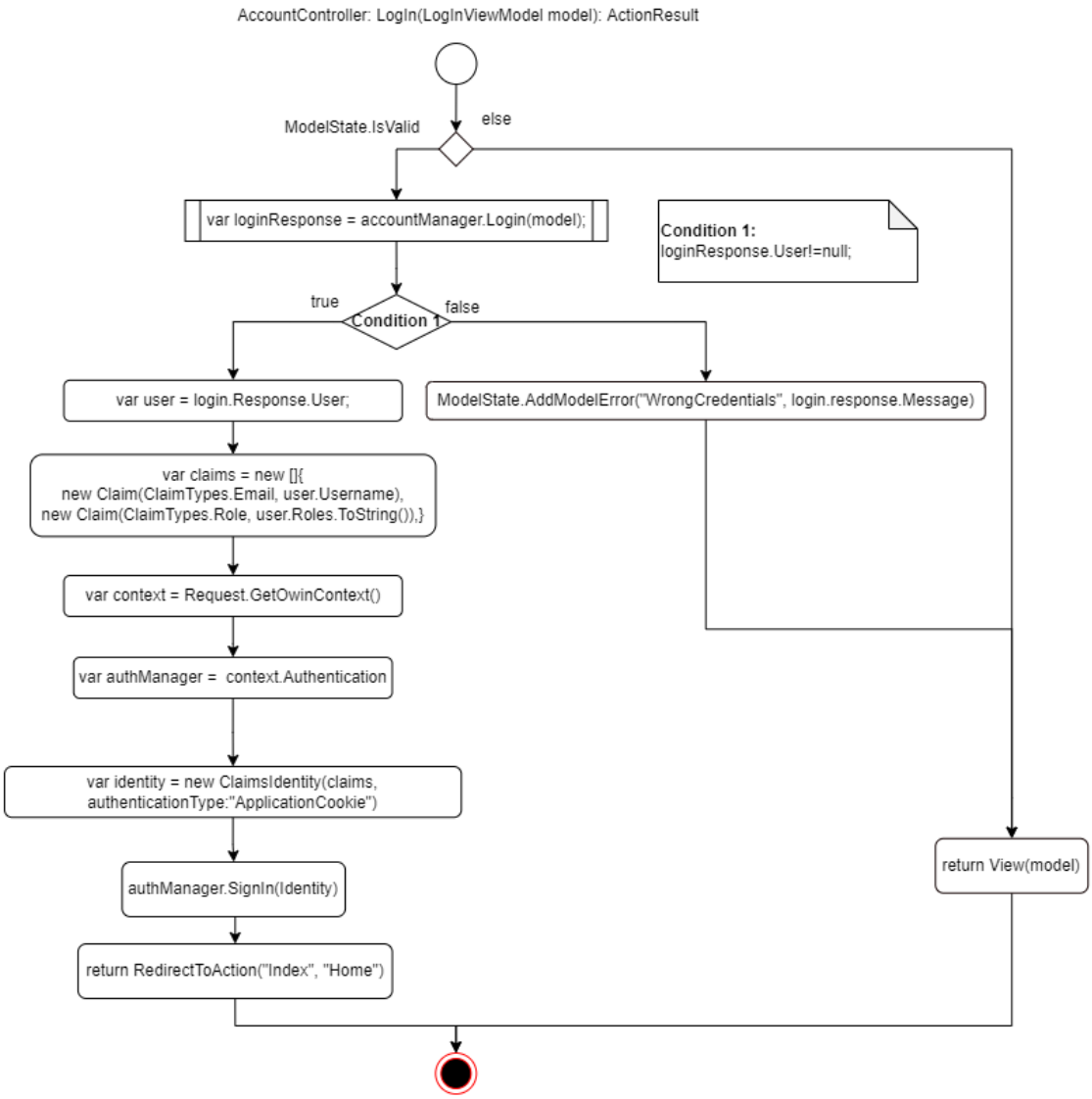

Рисунок 3.26 – Діаграма активності для методу аутентифікації користувача

Центральним елементом даного методу є виклик SingIn з AuthenticationManager, що складова OWIN. Перед викликом цього методу, ми передаємо Identity, яке містить інформацію про користувача, включаючи його

ідентифікатор та ролі, захищені Claims. Ці ролі визначають рівень доступу користувача до системи та надають можливості виконання конкретних операцій.

3.4.2 Механізм реєстрації користувача

Система має чітку архітектурну структуру для механізму реєстрації, яка включає кілька шарів для ефективного взаємодії та обробки даних. Основні шари включають:

1. Data source (Джерело даних) – використовується MS SQL Server та збережені процедури для зв’язку. Цей рівень є базою, де зберігаються всі дані про користувачів.
2. Data access layer (Рівень доступу до даних) – використовується SqlDataAccess (Dapper), щоб забезпечити ефективний зв'язок між програмою і базою даних чи іншим джерелом даних. Цей шар дозволяє звертатися до даних та виконувати різноманітні операції.
3. Stores (Сховища) – UsersDAL виступає в якості інтерфейсу для зв’язку з рівнем доступу до даних. Цей рівень забезпечує структурований спосіб взаємодії з даними та перенаправляє запити на рівень доступу до даних.
4. Managers (Менеджери) – CustomerService.BL виконує обробку та внесення змін у базу даних через так звані сховища. Цей рівень отримує дані від програми, взаємодіє з рівнем доступу до даних та виконує різноманітні бізнес-операції.
5. Asp.Net Applications (Додаток Asp.Net) – CustomerService.WebApplication виступає кінцевою точкою, яка обробляє запити від користувача та перенаправляє їх менеджеру. Цей рівень взаємодіє із зовнішнім інтерфейсом, при цьому забезпечуючє доступ користувачів до механізму реєстрації.

Алгоритм реєстрації нового користувача у системі представлений на рисунку 3.28, відображаючи послідовність операцій та взаємодію між різними шарами системи.

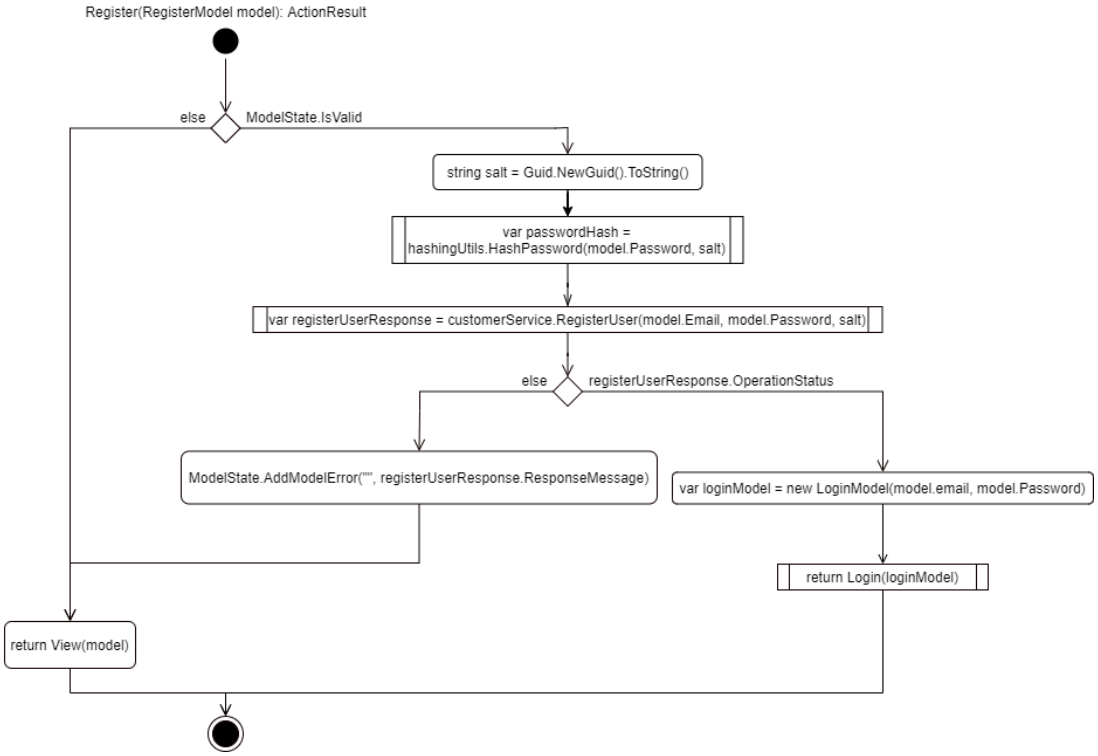

Рисунок 3.28 – Діаграма активності для механізму реєстрації користувача

3.4.3 Механізм хешування

Перед збереженням у базі даних важлива інформація, яка визначає можливість доступу користувача до системи, проходить через процес надзвичайно потужного захисту. Захист реалізується шляхом хешування критичних даних і використання вдосконаленого алгоритму SHA-512 [26]. Цей алгоритм перетворює введені користувачем дані в надійний 128-значний хеш, надійно захищаючи конфіденційні дані.

З метою додаткового підвищення безпеки кожен пароль обогащується унікальним значенням, відомим як "сіль". Це значення використовується при формуванні хешу пароля і також зберігається у базі даних. Додатковий шар безпеки введено за допомогою константи, яка зберігається прямо у коді системи і залишається недоступною для потенційних зловмисників. Ці два додаткові елементи ускладнюють завдання зловмисників у відтворенні правильного пароля, навіть якщо вони мають доступ до хешу.

Формула для обчислення хешу пароля виглядає наступним чином:

Хеш паролю = SHA512({"пароль"} + {"сіль"} + {"константа"})

Цей підхід робить вкрай важким знаходження правильного пароля методом перебору, оскільки навіть знання хеша майже не дає можливості відновлення вихідного пароля. Алгоритми для хешування пароля подані на рисунку 3.29, який визначає процеси, які гарантують надійний захист важливих користувацьких даних.

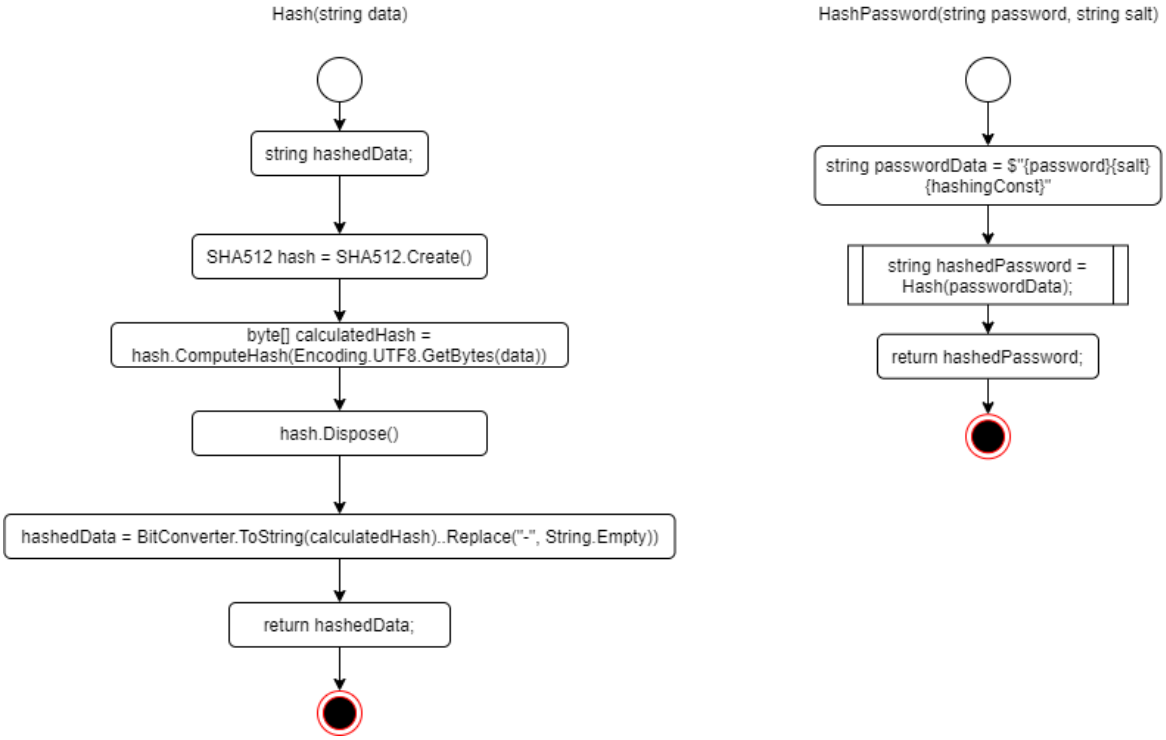

Рисунок 3.29 – Діаграма активності для процесу хешування

Процес хешування паролю відбувається на серверній стороні системи. Для клієнта, важливим є не саме хешування, і це обумовлено певними обставинами.

Ми можемо обробляти пароль як хеш ще до того, як передати його серверу. Такий підхід передбачає, що наша система отримує від користувача не реальний пароль, а його хеш, і подальша робота вже відбувається із збереженим хешем, а не з оригінальним паролем. Це також спрощує завдання зловмисникам у відновленні вихідного пароля, але це стратегічно необгрунтовано. Зловмисник, який перехопить хеш, зможе використовувати його для автентифікації в системі, оскільки сервер використовує хеш як пароль, інгороджуючи фактичний пароль від користувача.

Щодо шифрування на стороні клієнта, цей підхід також є неефективним. Код JavaScript доступний для перегляду і відкритий, тож будь-хто може вивчити алгоритм шифрування. Знайшовши алгоритм, можна використовувати його для розшифрування пароля, що суттєво підірве його захищеність.

Найбільш безпечним способом забезпечення захищеності з'єднання між клієнтом і сервером є використання протоколу SSL [27] (Secure Sockets Layer). Це робить дані, які передаються, недоступними для читання та змін у випадку їх перехоплення, надаючи ефективний захист від потенційних загроз.

3.4.4 Функціональні можливості системи

В загальному інструменти веб-розробки надають можливості створення вигляду головної сторінки електронного гаманця, яка відображає ключові функціональні можливості користувача. Головна мета цієї сторінки - надати швидкий та інтуїтивно зрозумілий доступ до операцій з транзакціями та перегляду стану гаманця.

Спроектований інтерфейс дозволяє користувачеві здійснювати наступні дії:

1. Створення транзакції – кнопка "Create transaction" відкриває вікно, де користувач може вказати отримувача, суму транзакції та інші деталі перед відправленням.
2. Перегляд гаманця – розділ "Wallet" відображає основні фінансові показники, такі як загальний баланс, список транзакцій та інші важливі дані.
3. Статистика та історія транзакцій – таблиці статистики надають користувачеві зрозумілу інформацію про його фінансові операції.
4. Фільтри та пошук – вбудовані гнучкі фільтри дозволяють швидко знаходити конкретні транзакції за різними параметрами, такими як дата, отримувач, статус або сума.

Цей інтерфейс має за мету поєднати простоту використання з багатофункціональністю, забезпечуючи користувачеві зручні та ефективні інструменти для управління своїми фінансами.

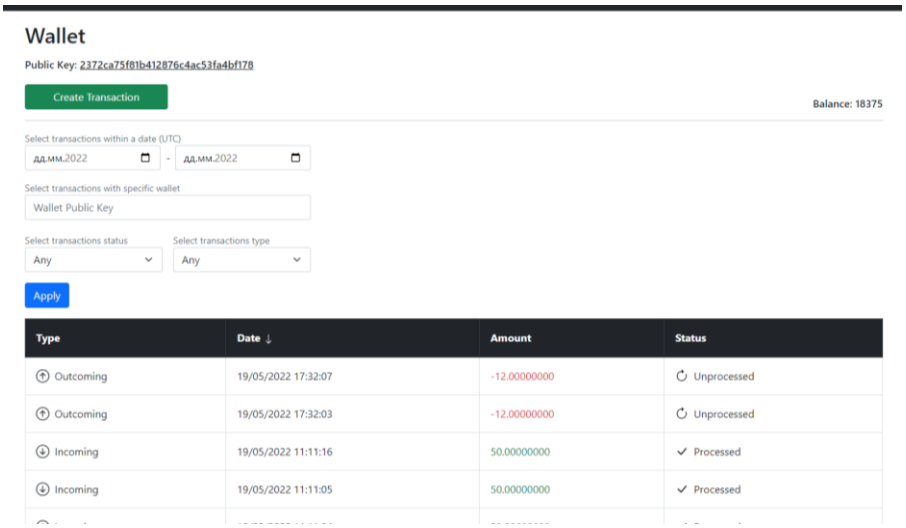

Рисунок 3.30 – Зовнішній вигляд головної сторінки гаманця

Користувачам доступно функціональне меню для здійснення переказу криптовалюти на інші гаманці. Процедура надзвичайно проста – необхідно вказати публічний ключ гаманця отримувача та суму переказу. Додатково, система автоматично враховує актуальний баланс, що відображений на сторінці, і попереджає користувача у разі введення некоректних даних.

У випадку, якщо вказаний публічний ключ гаманця отримувача недійсний або сума переказу перевищує актуальний баланс, система оперативно виведе спеціальне повідомлення про помилку. Це забезпечує користувача інформацією про необхідні корекції та дозволяє уникнути неправильних транзакцій.

На рисунку 3.31 зображено сторінку створення транзакції, де можна побачити інтуїтивно зрозумілі поля для введення даних, що сприяє легкості взаємодії користувача з системою та надає зручний інтерфейс для здійснення переказів.

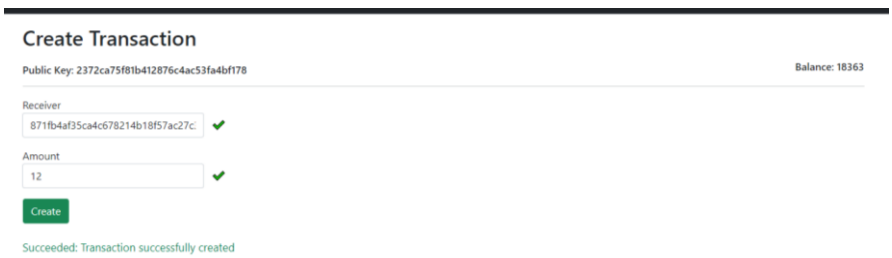

Рисунок 3.31 – Зовнішній вигляд сторінки створення нової транзакції

Однією з ключових особливостей нашого клієнтського додатку є можливість моніторингу змін у гаманці в режимі реального часу. Це означає, що якщо хтось створить транзакцію для нашого гаманця, список транзакцій автоматично оновиться без перезавантаження сторінки та без потреби ініціювати запит на оновлення даних вручну. Реалізацію цієї функціональності забезпечує бібліотека SignalR [28], яка базується на протоколі зв'язку WebSocket [29].

За допомогою WebSocket встановлюється двостороннє підключення між клієнтом та сервером, що дозволяє взаємодіяти та обмінюватися даними в режимі реального часу. Якщо на сервері відбуваються зміни у гаманцях користувачів, SignalR надсилає повідомлення відповідним користувачам, які мають активну сесію. Відповідно до цього, клієнтський додаток виконує асинхронний запит за допомогою AJAX [30] до сервера, щоб отримати актуальні дані. Після отримання повідомлення від сервера, клієнтський додаток автоматично актуалізує відповідні дані без перезавантаження веб-сторінки. Цей підхід забезпечує миттєву реакцію на зміни та покращує загальний досвід використання додатку для користувачів, що без сумніву покращує користувацький досвід та робить процес оновлення безперервним.

3.5 Розробка сервісу для забезпечення цілісності та безпеки криптовалютних транзакцій

Сервіс відіграє критичну роль у забезпеченні життєдіяльності транзакцій та системи в цілому, оскільки відповідає за створення нових блоків та емісію криптовалюти - головні завдання сервісу. Для взаємодії з системою клієнт спочатку підписується на сервіс транзакцій, щоб мати можливість отримання актуальних даних для обрахунку нового блоку. Після успішної підписки ініціюється процес обрахунку – ключового етапу, що визначає подальшу стійкість та функціонування системи. На рисунку 3.32 зображено алгоритм процесу обрахунку блоку, який включає в себе складні обчислення та створення нового блоку для включення його в ланцюг блоків.

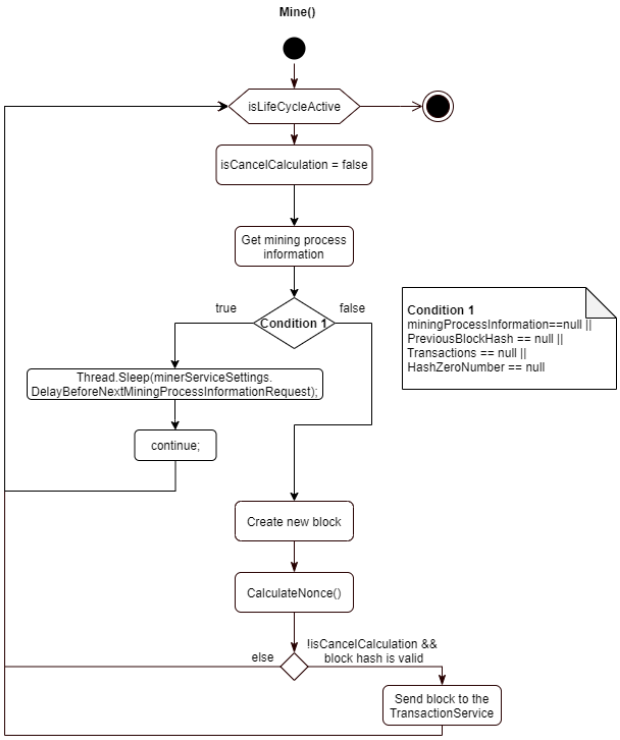

Рисунок 3.32 – Діаграма активності для процесу обрахунку блоку

Загальний алгоритм роботи сервісу такий:

- Під час процесу обрахунку кожного блока блокчейн сервіс ініціює запит до сервісу транзакцій щоб отримати групу транзакцій для створення нового блоку. Якщо в цей момент сервіс транзакцій проводить валідацію поточних блоків отриманих від іншихрозгорнутих сервісів блокчейну, отримання даних для нового блоку тимчасово призупиняється до завершення процедури валідації. Це гарантує, що дані, які будуть включені в новий блок, пройшли валідацію та відповідають усім встановленим правилам та стандартам системи. Отримавши коректні дані для блоку ініціюється процес обрахунку хешу для блока. Для цього до всіх даних, що містить блок ведеться пошук певного значення, яке потрібно додати, щоб отримати хеш, який задовольнить умову – певне шістнадцяткове значення з якого хеш повинен починатись. На рисунку 3.33 наведено алгоритм обрахунку хешу блока.

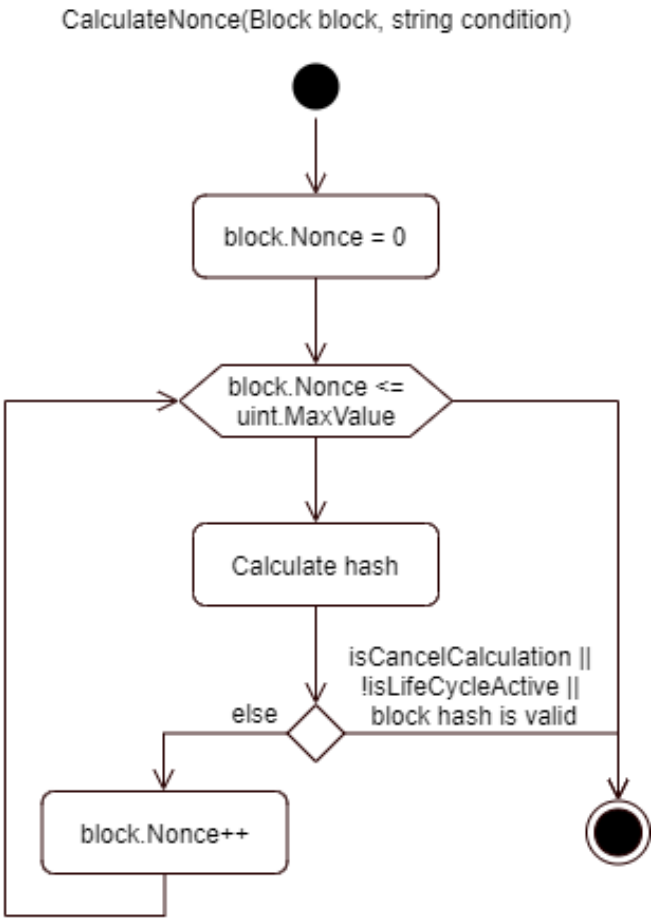

Рисунок 3.33 – Діаграма активності для процесу хешування блоку

- Після успішного знаходження хешу для блока, клієнтський додаток ініціює запит до сервера, передаючи обчислений блок для проведення валідації. Цей крок є важливим етапом, оскільки він забезпечує перевірку правильності обчисленого хешу та відповідність нового блоку усім вимогам системи.Після відправлення запиту процес циклічно продовжується для кожного блоку.

Процес обчислення блоків повністю автономний та не вимагає активної участі користувача, за винятком випадків, коли необхідно завершити роботу додатку. На рисунку 3.34 відображено вікно додатку з активним процесом обчислення хеша блока. У цьому режимі програма самостійно взаємодіє з сервісом транзакцій, отримуючи та оброблюючи дані для нового блоку. Під час роботи програма періодично відображає прогрес процесу, забезпечуючи користувачеві візуальний звіт про виконання завдань.

Рисунок 3.34 – Зовнішній вигляд консольного додатку

Як вказано на зображенні, додаток звертається до сервера для отримання необхідних даних, таких як хеш останнього доданого блоку, інформація про транзакції та умова хешування (в даному випадку, хеш повинен починатися з «00000»). Ці дані є ключовими для процесу обчислення хеша, який відображається у полі "Calculated hash".

З метою підтримки взаємодії із сервісом транзакцій, на консольному клієнті було розгорнуто WebAPI. Під час запуску додатку на початкових етапах створюється хост, для якого вказується локальна IP-адреса. Цей механізм дозволяє забезпечити зворотній зв’язок із сервером транзакцій та управляти обміном даними для ефективного обчислення та обробки нових блоків у системі.

В якості зворотнього зв’язку сервіс транзакцій час від часу створює два типи запитів всім підключеним клієнтам:

- Кожен клієнтський додаток має вбудований механізм для перевірки своєї працездатності. Якщо клієнт отримує запит на перевірку працездатності від сервера, він негайно відповідає, підтверджуючи свою активність і інформуючи, що він готовий обраховувати нові блоки. У випадку, якщо сервер не отримує відповіді протягом визначеного часового інтервалу, він розглядає цей клієнт як недоступний і автоматично відключає його. Однак, коли клієнт повертається до роботи, він може ініціювати підключення, відправляючи додатковий запит на підключення до сервера. Цей механізм дозволяє забезпечити ефективне використання ресурсів та підтримувати стабільну роботу системи.
- Коли в системі додається новий валідний блок, сервер надсилає запит на відміну поточного обчислення блоку кожному клієнту. Це необхідно, оскільки

дані, які використовуються для обчислення блоків, вже застаріли і неактуальні. Запит на відміну поточного обчислення дозволяє клієнтам припинити обчислення на застарілих даних і оновити їх, отримавши нові актуальні дані від сервера. Цей механізм забезпечує синхронізацію між сервером і всіма клієнтами, щоб уникнути можливих конфліктів та забезпечити правильність обчислень у системі.

3.6 Висновки

У третьому розділі, присвяченому розробці програмних модулів централізовнаї криптовалюти, звернуто увагу на ключові інструменти, що визначають основну бізнес-логіку та серверну частину проекту. Вибір був зроблений на користь платформи .Net та мови програмування C#, яка гарантує ефективність та гнучкість розробки. Для створення програмного забезпечення обрано середовище розробки Visual Studio, визначаючи його як оптимальний інструментарій для проектів, базованих на .NET.

У контексті вибору бази даних віддана перевага MS SQL Server, враховуючи його тісну інтеграцію з платформою .NET та високий рівень безпеки. Micro-ORM Dapper обраний для взаємодії з базою даних за його зручність та продуктивність при роботі з збереженими процедурами.

У сфері веб-розробки визначено використання ASP .NET, яке надає зручні інструменти для роботи з даними. Для створення користувацького інтерфейсу вибрана CSS-бібліотека Bootstrap, що дозволяє ефективно створювати сучасний та адаптивний дизайн.

Розглянуті також модулі для комунікації, які передбачають розробку базових методів виклику створених процедур та високорівневих модулів для використання іншими сервісами.

Описано процеси, що відбуваються поза API, яку викликають клієнтські сервіси. Зазначено роботу сервісу транзакцій, включаючи їх створення, зберігання та підтвердження. Також надано опис високорівневих процесів бізнес-логіки, в яких транзакції беруть участь.

У результаті роботи над сервісом для користувачів виникла веб-додаток,

який після реєстрації дозволяє користувачеві отримати доступ до свого гаманця. Користувач може переглядати свої транзакції в режимі реального часу та створювати нові, за умови наявності достатнього балансу.

Результатом роботи над сервісом для емісії криптовалюти став консольний додаток, який підтримує життєвий цикл системи. Даний додаток підтверджує неопрацьовані транзакції, упаковуючи їх у блоки, які хешуються за спеціальним методом для подальшого зберігання.

4 ТЕСТУВАННЯ КРИПТОВАЛЮТНОЇ СИСТЕМИ

4.1 Юніт-тестування бізнес-логіки

Для тестування всіх модулів системи було застосовано підхід Unit-testing [31] в комбінації з підходом TDD [32]. Платформа .Net пропонує вбудований інструмент для тестування MS Test. В загальному підхід до тестування кожного модуля окремо передбачає незалежність від зовнішніх викликів. Для слідування цьому підходу використано бібліотеку Nsubstitute [33], що дозволяє імітувати виклики інтерфейсів, розташовані у методах. Розглянемо детальніше на прикладі тесту для випадку успішного створення електронного гаманця.

Кожен з тестів слідує патерну AAA [34], або ж Arrange, Act, Assert (Підготування, Дія, Перевірка). У прикладі на етапі Arrange ми створюємо інфрастурктуру для виклику метода. Для цього необхідно створити об’єкт у класі якого описано метод. Для прикладу візьмемо клас TransactionService, який має низку залежностей, які передаються у якості параметра до конструктора класу. Оскільки кожна із залежностей представлена інтерфейсом ми можемо сторити цю залежність за допомогою NSubstitute. NSubstitute дозволяє створювати імітації об'єктів, які можуть підміняти реальні об'єкти в тестах. Це особливо корисно при тестуванні, коли потрібно ізолювати частину коду від реальних об'єктів або залежностей. NSubstitute може створювати імітації як для інтерфейсів, так і для класів. Це робить його універсальним інструментом для тестування різних типів об'єктів. NSubstitute надає можливість перевіряти, чи були викликані певні методи на імітованих об'єктах, які передаються в методи тестованого коду. Можливість визначати, з якими параметрами був викликаний певний метод на імітованому об'єкті. Задання повернення значень для методів імітованих об'єктів для визначення очікуваного поведінки. NSubstitute має досить лаконічний і зрозумілий синтаксис, що полегшує використання в тестовому коді. Легко інтегрується у проекти .NET через NuGet-пакети і не вимагає великої кількості додаткового налаштування. Використання NSubstitute дозволяє писати ефективні тести, ізолюючи той чи інший фрагмент коду від залежностей та перевіряючи його поведінку. Такий підхід

допомагає забезпечити стабільність та надійність програмного забезпечення.

Nsubstitute звісно значно спрощує тестування коду, але при створенні бізнес-логіки чимало кодової бази є недоступною для прямого виклику з-під тесту, для прикладу – приватні методи. Для тестування таких методів викуористовується підхід через створення пакетів, видимих лише для тестового проекту. Таким чином для кожного класу стоврюється інтерфейс з дублікатами приватних методів. При імплементації інтерфейсу створюються методи-дублікати приватних, видимі лише для тестових проектів, і недосстпуних для інших. У методах-дублікатах, що повністю повторюють сигнатуру приватного методу, виконуєтсья виклик цього самого приватного методу, що дає можливість створити юніт-тест для тестування закритого участку коду.

В загальному для тестування бізнес-логіки було написано 518 юніт-тестів. У поєднанні з різними наборами тестових вхідних даних та очікуваних результатів (мається на увазі використання одного тестового методу для тестування різних ділянок коду в залежності від вхідних даних) було отримано 799 тестових сценаріїв.

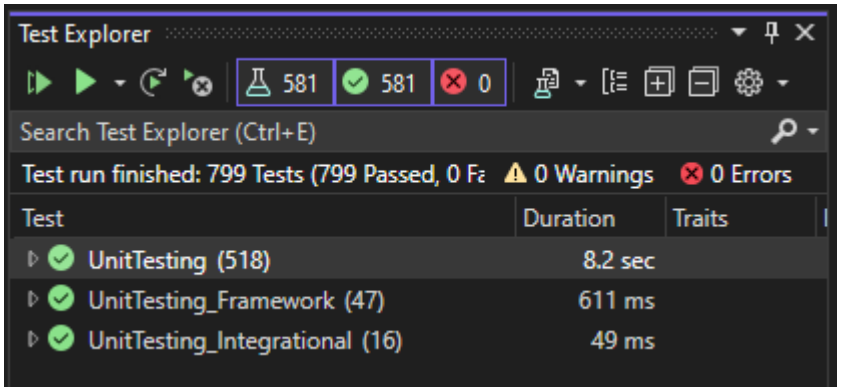

Рисунок 4.1 – Результат запуску юніт-тестів

4.2 Тестування веб-додатку для роботи з електронним гаманцем

Для валідації та тестування функціональності веб-системи використовувався хостинг на платформі IIS (Internet Information Services). Після успішного конфігурування додатку за вказаними налаштуваннями, ми здобули можливість отримати доступ до нашого додатку через визначену адресу. При першому відкритті додатку користувача автоматично перенаправляє на початкову сторінку з

привітальним повідомленням.

На цій початковій сторінці у нас відкривається можливість здійснити реєстрацію для нових користувачів або авторизуватися у системі, якщо у нас вже є власний обліковий запис. Цей етап дозволяє виконати базове тестування реєстраційного та авторизаційного функціоналу системи та переконатися в правильній роботі механізмів введення та валідації облікових даних.

Під час переходу до процесу реєстрації відкривається спеціальна сторінка, на якій користувачеві пропонується ввести свою електронну пошту та створити безпечний пароль. На зображенні 4.2 відображено інтерфейс сторінки, призначеної для реєстрації нового користувача.

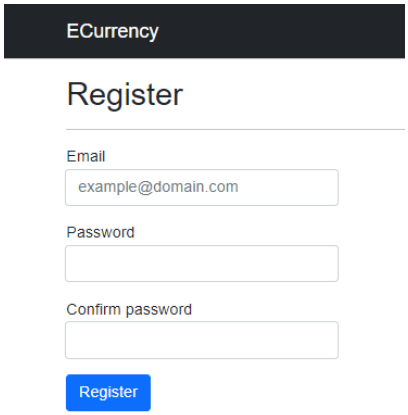

Рисунок 4.2 – Сторінка реєстрації

Цей етап процесу надає користувачеві можливість створити особистий обліковий запис у системі, використовуючи унікальну електронну адресу та надійний пароль. Інтерфейс сторінки реєстрації відображено так, щоб зробити процес якнайзручнішим та інтуїтивно зрозумілим для користувача, забезпечуючи йому необхідні інструменти для введення та перевірки облікових даних.

При введенні неправильних або невалідних даних система негайно повідомляє користувача про помилки, які виникли в кожному конкретному полі. На ілюстрації 4.3 відображено сторінку реєстрації, де введені дані не відповідають вимогам системи.

Цей інтерфейс призначений для активного спілкування з користувачем та

вказівки йому на конкретні помилки у введених даних. Кожне поле, що містить невірні або неприйнятні значення, відзначається відповідною помилковою позначкою, щоб користувач міг оперативно виправити введені ним дані та завершити процес реєстрації успішно.

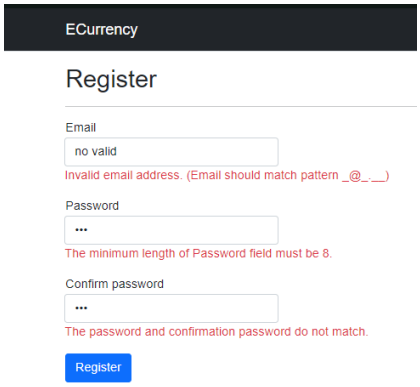

Рисунок 4.3 – Сторінка реєстрації при введенні некоректних даних

При повторному вході в систему зареєстровані користувачі мають можливість використовувати сторінку аутентифікації, де вони повинні ввести свою електронну пошту та пароль. На зображенні 4.4 представлена сторінка аутентифікації користувача, яка надає можливість зручного та безпечного входу в особистий обліковий запис.

Цей інтерфейс дозволяє користувачам швидко та ефективно ввійти в систему, використовуючи свої реєстраційні дані. Введені електронна пошта та пароль перевіряються на відповідність системним записам, а надійний механізм аутентифікації гарантує безпеку особистих даних користувача.

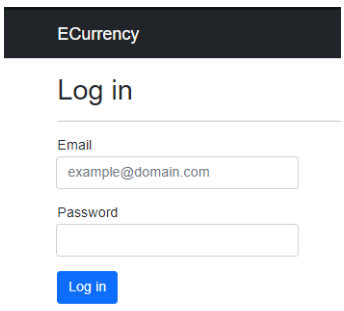

Рисунок 4.4 – Сторінка входу в акаунт

У випадку, коли користувач вводить неправильні облікові дані, система повідомляє, що введеного користувача не існує. Це попередження служить сигналом, що можливо, була допущена помилка в електронній пошті або паролі. Користувачу рекомендується перевірити правильність введених даних та виправити будь-які помилки.

На рисунку 4.5 представлена сторінка аутентифікації користувача під час введення недійсних облікових даних. Це інтуїтивно зрозуміле повідомлення надає користувачеві необхідну інформацію для подальших кроків та підтримує відмінний взаєморозуміння між користувачем і системою.

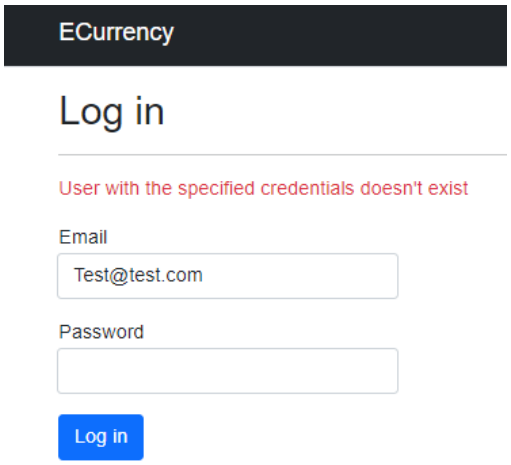

Рисунок 4.5 – Сторінка входу в акаунт при введенні неправильних даних

Після успішної аутентифікації користувача в системі відбувається автоматичне перенаправлення на основну сторінку його електронного гаманця. На цій сторінці користувач має можливість переглядати всі транзакції, які пов'язані з його акаунтом. Додатково, він може використовувати функцію фільтрації транзакцій для швидкого пошуку та вивчення конкретних транзакцій за певними параметрами.

Безпосередньо на головній сторінці гаманця користувач має зручний доступ до функцій створення нової транзакції або перегляду деталей конкретної транзакції, введенням її ідентифікатора у відповідне поле. Загальний вигляд головної сторінки електронного гаманця ілюстровано на рисунку 4.6.

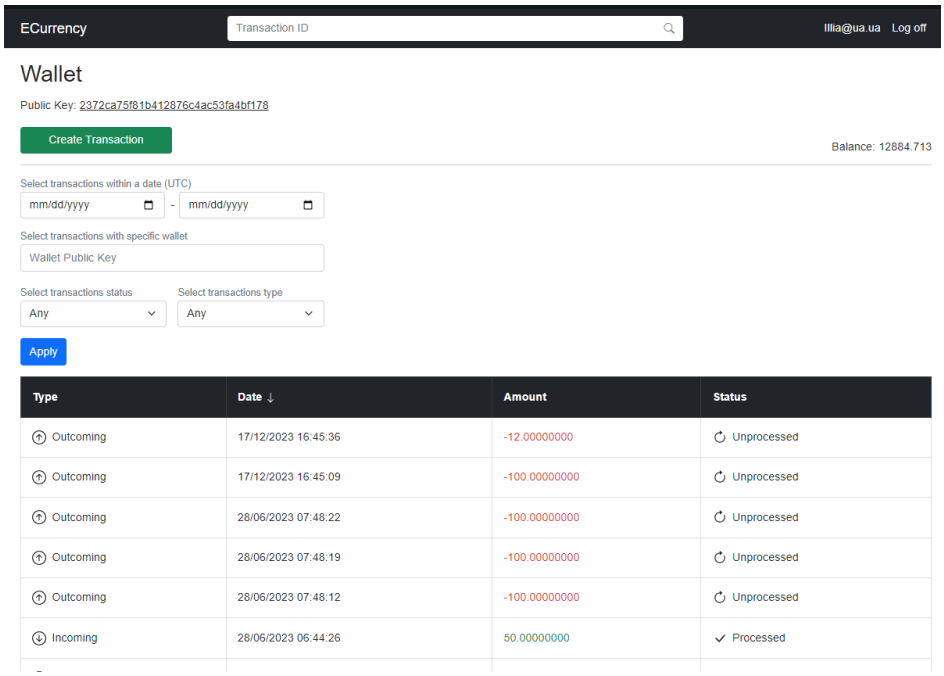

Рисунок 4.6 – Сторінка електронного гаманця користувача

На сторінці створення транзакції користувачеві стає доступними два основні поля для введення необхідної інформації: публічного ключа гаманця отримувача та суми переказу. Перед відправленням транзакції система виконує перевірку введених даних на валідність.

У випадку, якщо користувач введе дані невірно або пропустить яке-небудь обов'язкове поле, система виведе відповідне повідомлення про помилку, підказуючи, які саме дані потрібно виправити або доповнити. Це забезпечить користувачеві зручний інтерфейс для створення та відправлення транзакцій, а також допоможе уникнути непорозумінь та помилок під час введення інформації. Рисунок 4.7 ілюструє вигляд сторінки створення транзакції в системі.

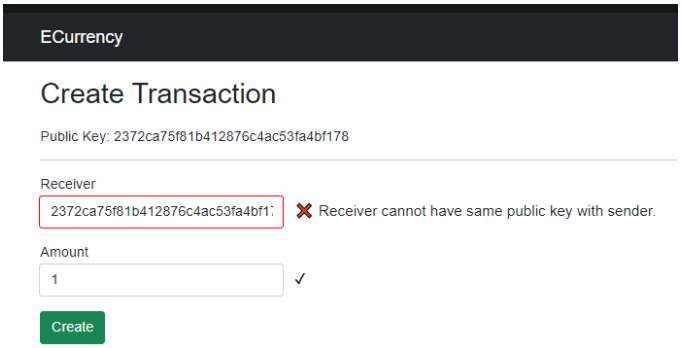

Рисунок 4.7 – Сторінка створення транзакція на неправильний гаманець

В разі успішного виконання переказу користувач отримає спеціальне повідомлення про успішно завершену операцію. Це повідомлення підтверджує, що транзакція була успішно створена та відправлена до обробки системою. Такий інформаційний підсумок надає користувачеві впевненість у тому, що його дії в системі пройшли успішно.

На рисунку 4.8 представлено зображення сторінки створення транзакції, де користувач ввів коректні дані, і відобразилося повідомлення про успішний переказ. Це дозволяє користувачеві легко візуалізувати і підтвердити успіх своєї дії у системі.

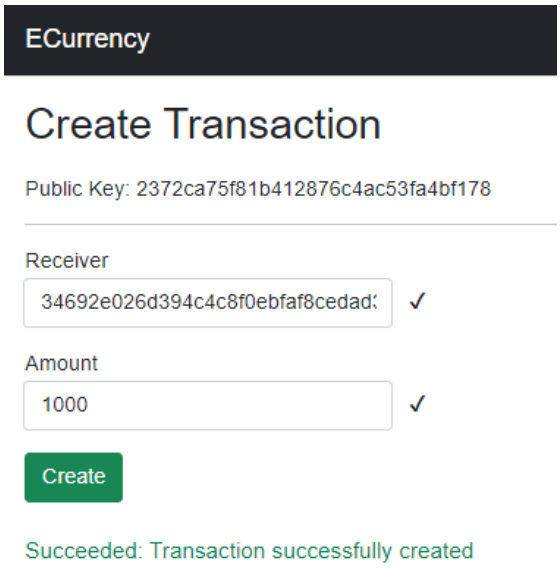

Рисунок 4.8 – Сторінка створення транзакція з правильними даними

При використанні функції фільтрації транзакцій за їхнім статусом, користувач отримає список необроблених транзакцій, серед яких буде відображена і новостворена транзакція. На рисунку 4.9 відображено процес фільтрації транзакцій за їхнім поточним статусом.

Завдяки цій функціональності користувач може швидко і зручно визначити статус та переглянути докладну інформацію про обрані транзакції відповідно до власних потреб.

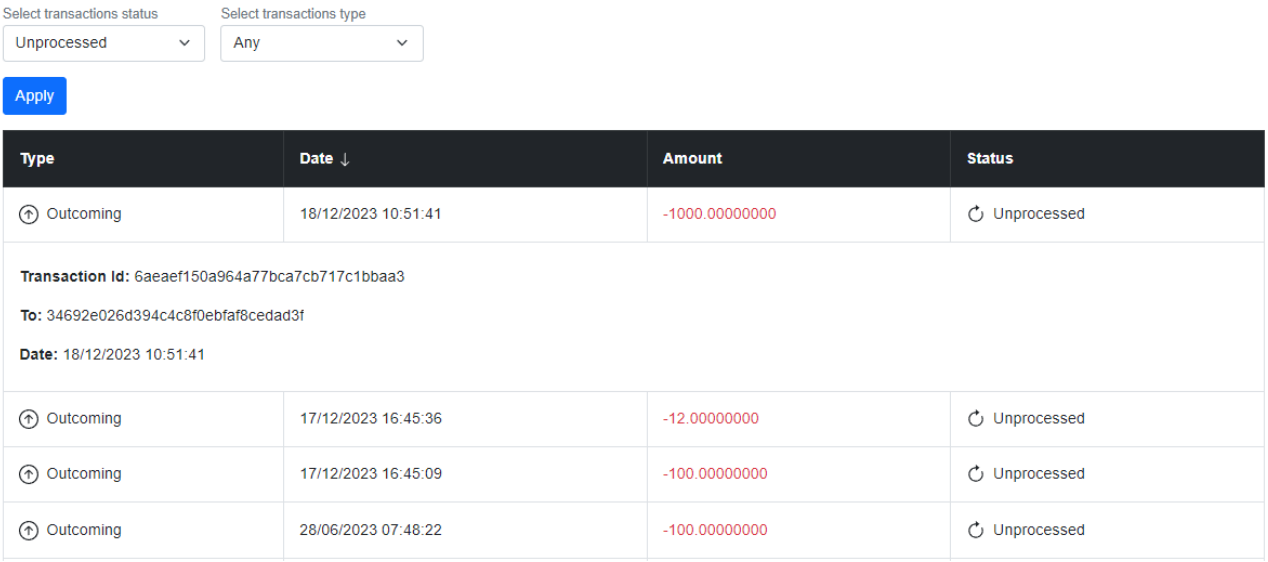

Рисунок 4.9 – Фільтрація транзакцій по статусу

При необхідності знаходження конкретної транзакції користувач може використовувати ідентифікатор і ввести його у відповідне поле в шапці сторінки.

На рисунку 4.10 наведено зображення сторінки з детальною інформацією про транзакцію.

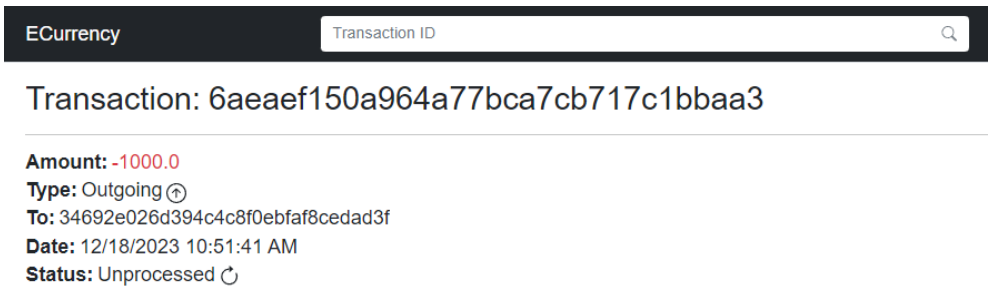

Рисунок 4.10 – Перегляд конкретної транзакції по її ідентифікатору

Після введення ідентифікатора користувач буде автоматично перенаправлений на окрему сторінку, де доступна детальна інформація про зазначену транзакцію, якщо така транзакція існує у системі.

Це включає в себе всі важливі дані, пов'язані з транзакцією, такі як відправник, отримувач, сума переказу, час створення, та інші важливі атрибути. Такий детальний перегляд дозволяє користувачам отримати повну картину про вибрану транзакцію.

4.3 Тестування API

Postman [35] – це інструмент для розробки API, який надає розробникам, тестувальникам та іншим спеціалістам в області програмування набір зручних інструментів для створення, тестування та автоматизації взаємодії з API. Використання Postman полегшує розробку та управління API, забезпечуючи швидку, ефективну та надійну роботу.

Postman дозволяє розробникам легко створювати HTTP-запити для взаємодії з різними ендпоінтами API. Зручний користувацький інтерфейс дозволяє вам обирати тип запиту, додавати параметри, встановлювати заголовки та відправляти запити для перевірки реакції сервера.

Postman дозволяє організовувати запити у колекції, що полегшує управління проектами та тестовими наборами. Запити можна зберігати, архівувати та шарити з іншими членами команди.

Postman надає вбудовані можливості тестування для перевірки відповідей сервера. За допомогою JavaScript можна створювати автоматизовані тести та перевіряти коректність відповідей, статуси кодів, заголовки тощо.

Postman забезпечує зручний інтерфейс для зберігання середовищ, змінних та інших налаштувань проекту. Це дозволяє розробникам легко переключатися між різними середовищами (наприклад, тестовий та продакшн) без повторного налаштування кожного запиту.

Postman надає можливість генерувати колекції та документацію для API. Це дозволяє легко ділитися з іншими членами команди або надавати інформацію стороннім розробникам.

Postman забезпечує зручні інструменти для редагування та зберігання історії змін. Це важливо для відстеження та управління змінами в запитах та колекціях.

За допомогою Postman Cloud, можна синхронізувати дані між різними пристроями та обліковими записами, що полегшує спільну роботу в команді.

Postman надає засоби для роботи з HTTPS, автентифікації та інші засоби безпеки для забезпечення надійності та безпеки взаємодії з API.

Використання Postman значно полегшує розробку та тестування API,

забезпечуючи розширені інструменти для автоматизації, організації та аналізу. Його зручний інтерфейс та багатий функціонал роблять його невід'ємною частиною процесу розробки програмного забезпечення.

Отож протестуємо основні запити. На рисунку 4.11 зображено результат запиту для створення транзакції. Для запиту було обрано повністю валідні дані – інсуючі електронні гаманці та наявну кількість монет на гаманці відправника. У відповіді бачимо, що запит успішно виконано та транзакція створена.

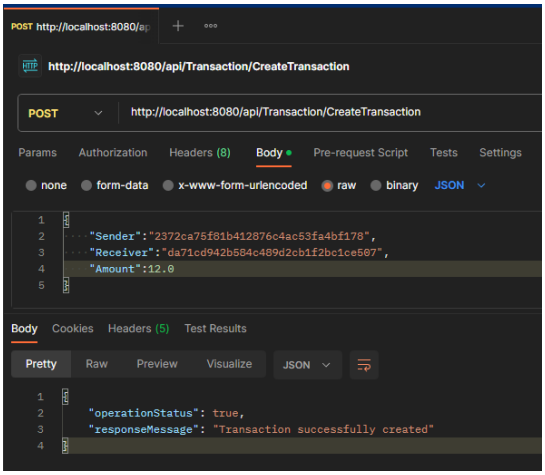

Рисунок 4.11 – Результат виконання запиту для створення транзакції

Також перевіримо декілька інших сценарії для цього запиту. На рисунку 4.12 зображено результат виконання запиту для створення транзакції, що був відхилений через недостатню кількість монет на електронному рахунку.

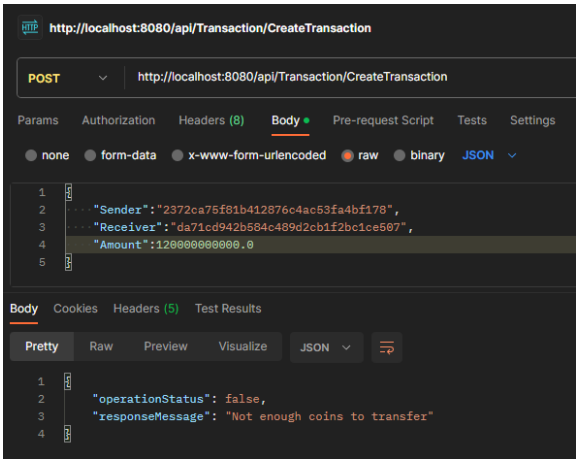

Рисунок 4.12 – Результат виконання запиту

Інший випадок – спроба створити транзвкція на рахунок, який не інує. Очевидний результат – транзакція не була створена. На рисунку 4.13 зображено результат виконання запиту.

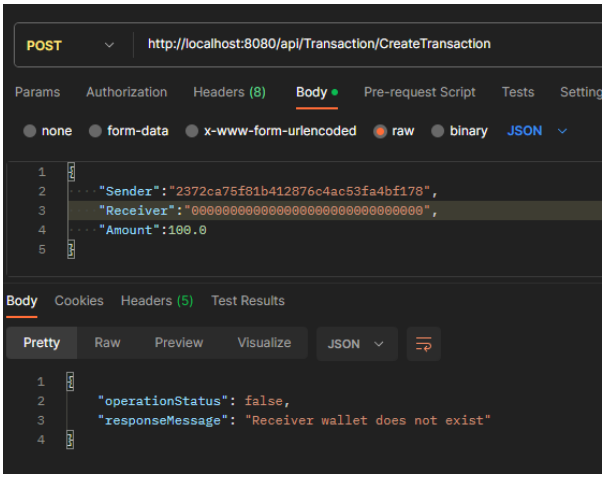

Рисунок 4.13 – Спроба створити транзакцію на неіснуючий рахунок

Це були приклади POST запитів. Тепер розглянемо кілька варіантів GET запитів.

Для прикладу запит на отримання поточного балансу користувача. Як параметр передаємо ідентифікатор користувача у системі та у відповідь при умові, що користувач увійшов до системи під своїм обліковим записом відсилаємо поточний баланс.

На рисунку 4.14 зображено виконання запиту.

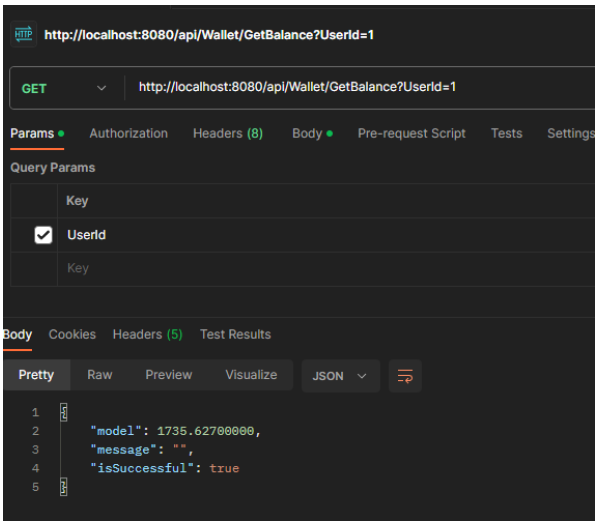

Рисунок 4.14 – Результат виконання запиту для отримання поточного балансу

Для прикладу – важливого GET запиту – отримання інформації про транзакцію. В якості параметру до запиту передається ідентифікатор транзакції. У відповідь, при умові, що транзакція існує у системі під вказаним ідентифікатором, надсилається вичерпна інформація – публічні ключі відправника та отримувача, кількість монет у транзакції, дата останнього оновлення та поточний статус. На рисунку 4.15 зображено результат виконання запиту.

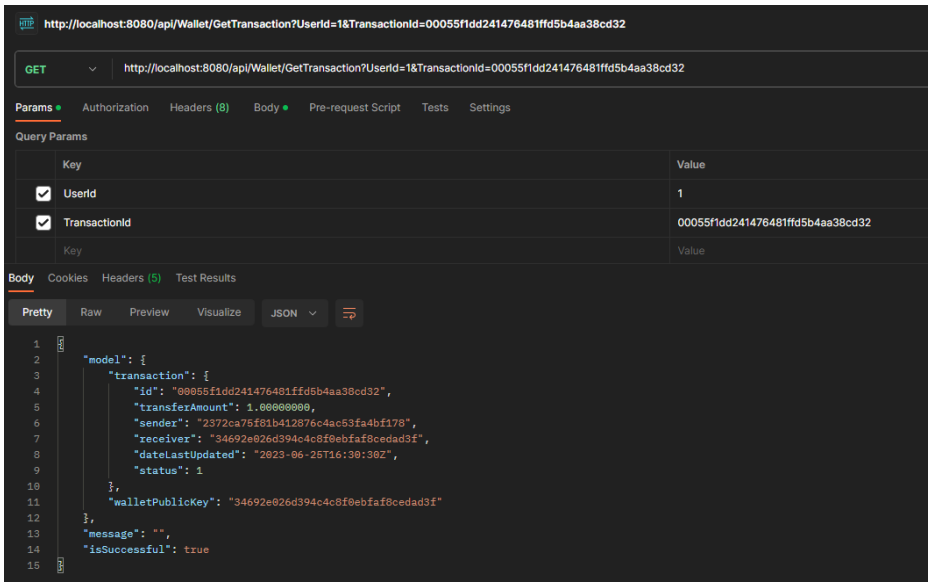

Рисунок 4.15 – Результат виконання запиту для отримання інформації про транзакцію

4.4 API інтрефейс з використанням Swagger та документація коду з використанням DocFX

Swagger [36] – це інструмент для розробки та документування API, який спрощує створення, тестування та взаємодію з веб-службами. Його популярність у світі розробки API пояснюється зручністю автоматизації процесу генерації документації, створення клієнтських бібліотек та тестування API. Давайте розглянемо ключові аспекти Swagger.

Swagger основано на специфікації OpenAPI (раніше відомої як Swagger Specification). OpenAPI – це стандарт опису веб-служб, який визначає формат для опису RESTful API. Swagger допомагає автоматизувати процеси створення та документування API за допомогою інструментів, які генерують документацію, код

та тести.

Ключові складові Swagger:

Swagger Editor – це візуальний редактор, який дозволяє розробникам створювати, редагувати та переглядати OpenAPI-специфікації у форматі YAML або JSON. Редактор забезпечує інтерактивний інтерфейс для визначення ресурсів, методів та параметрів API.

Swagger UI – це інтерактивне веб-середовище для взаємодії з API на основі OpenAPI-специфікацій. Воно автоматично генерує візуальну документацію API, яка дозволяє розробникам тестувати різні ендпоінти та переглядати приклади запитів та відповідей.

Swagger Codegen – це інструмент командного рядка для генерації клієнтського коду, серверного коду та навіть конфігураційних файлів для різних мов програмування на основі OpenAPI-специфікацій. Це дозволяє автоматизувати процес створення клієнтів та серверів для вашого API.

Для інтеграції Swagger з C# та платформою ASP.NET Core часто використовується бібліотека Swashbuckle.AspNetCore. Вона автоматично генерує OpenAPI-специфікації на основі атрибутів та коментарів коду. Додавши Swashbuckle до вашого проекту, ви можете легко створити та оновлювати документацію API під час розробки.

Розробники можуть використовувати атрибути та коментарі коду для докладного опису ресурсів, методів, параметрів та відповідей. Ця інформація використовується Swashbuckle для автоматичної генерації OpenAPI-специфікацій.

Swagger дозволяє генерувати чітку та інтерактивну документацію API автоматично. Це полегшує розробку та робить API більш доступним для інших команд.

Завдяки Swagger UI, розробники можуть тестувати різні частини API прямо в браузері. Це допомагає в їхній взаємодії з API та виявленні помилок.

Swagger Codegen дозволяє автоматично генерувати клієнтський код для взаємодії з API, що прискорює розробку.

Swagger є потужним інструментом для розробки та документування API,

дозволяючи розробникам зосередитися на створенні якісних веб-служб. Завдяки його інструментам та інтеграції з C# та ASP.NET Core, він став необхідним елементом для будь-якого проекту, що використовує API для взаємодії з клієнтами та іншими службами.

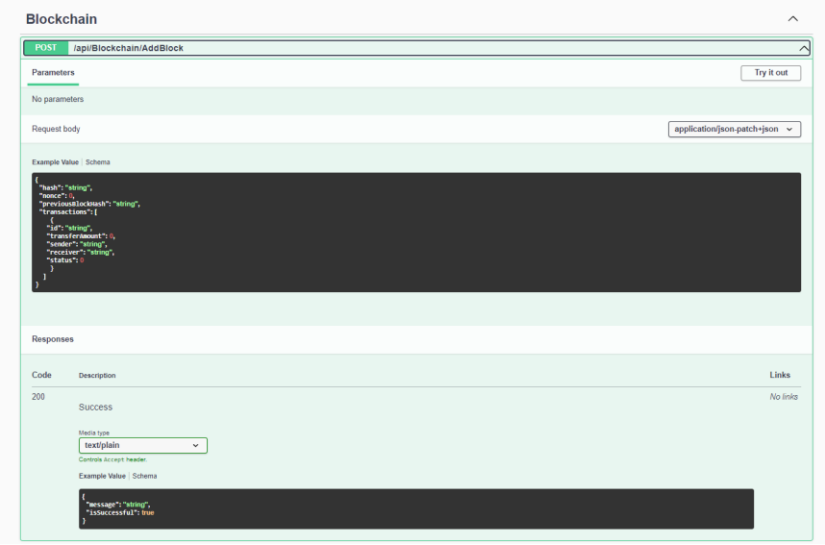

Рисунок 4.16 – Зовнішній вигляд інтерфейсу для виконання запитів до сервісу

DocFX [37] – це відкрите інструментарій для створення документації з програмного коду, особливо для проектів на платформі .NET. Він дозволяє генерувати високоякісну та добре оформлену документацію, яка може бути використана для опису API, бібліотек, фреймворків та інших проектів програмного забезпечення.

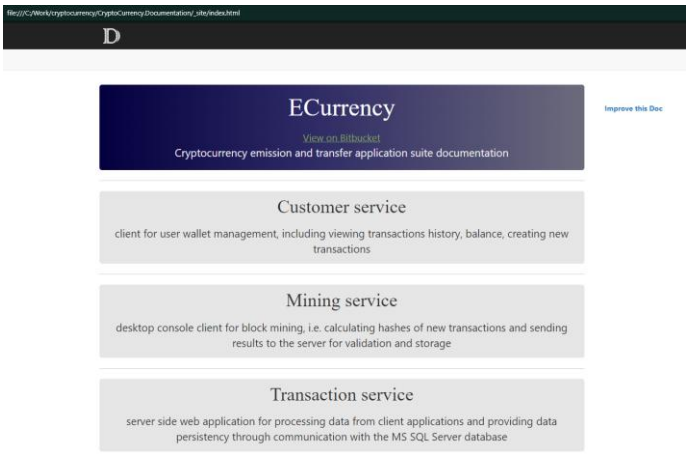

Рисунок 4.17 – Головна сторінка після генерації документації

DocFX використовує Markdown для написання документації. Це спрощує створення читабельного та легко редагованого тексту.

Підтримка мультиязичності дозволяє створювати документацію для проектів, розроблених для різних регіонів та мов.

Докоментування коду .NET, написаного в стандартному форматі XML коментарів, дозволяє автоматично генерувати частину документації з програмного коду.

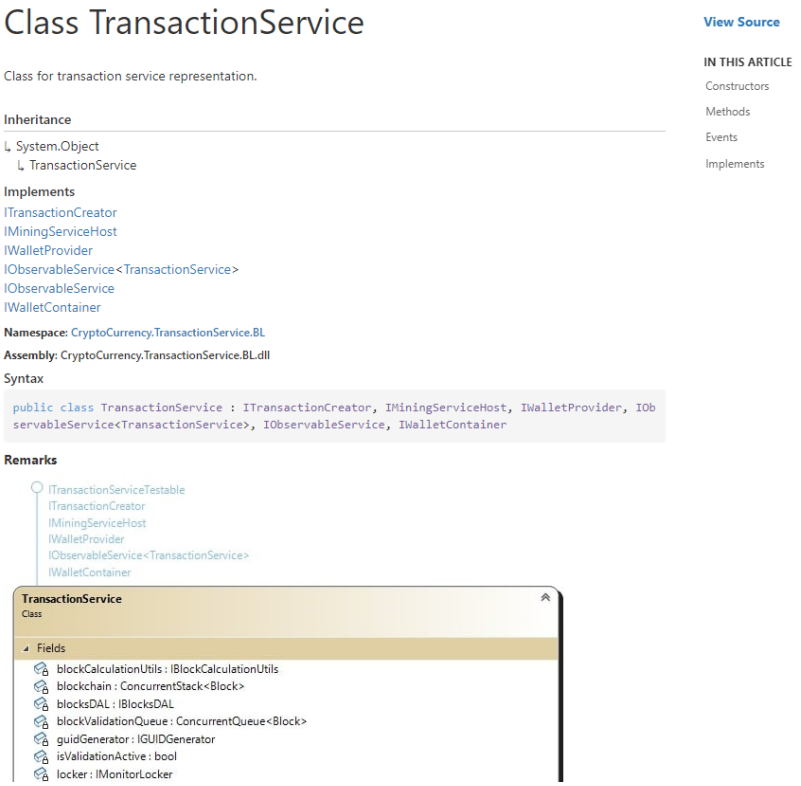

Рисунок 4.18 – Згенерована документація на основі коду

Можливість інтеграції з Visual Studio дозволяє генерувати документацію безпосередньо під час розробки.

DocFX постачається з рядом вбудованих шаблонів та стилів для форматування та оформлення документації.

Можливість налаштування різних параметрів, таких як вихідний формат, вигляд документації та інші, дозволяє розробникам вибирати налаштування відповідно до власних потреб.

DocFX дозволяє об'єднувати документацію з різних джерел та генерувати її

у різні формати, такі як HTML, PDF, або в інших вихідних форматах.

Режим розробки дозволяє переглядати та тестувати документацію локально перед публікацією.

DocFX добре підходить для великих та складних проектів, де потрібна докладна та структурована документація. Він може бути використаний для проектів на платформі .NET, а також для інших мов та технологій за умови належного налаштування та використання відповідних плагінів.

4.5 Висновки

У четвертому розділі було описано принципи тестування криптовалютної системи. Для тестування функціональних вузлів системи було використано Unit-тестування з використанням mock-фреймворку Nsubstitute. Для тестування приватних методів було затосовано підхід через створення пакетів, видимих лише для тестового проекту. Таким чином для кожного класу, що підлягає тестуванню створено інтерфейс з дублікатами приватних методів. При імплементації інтерфейсу методи-дублікати видимі лише для тестових проектів, і недосстпуні для інших. У методах-дублікатах, що повністю повторюють сигнатуру приватного методу, виконано виклик самого приватного методу, що дало можливість створити юніт-тест для тестування закритого участку коду.

В загальному для тестування бізнес-логіки було написано 518 юніт-тестів. У поєднанні з різними наборами тестових вхідних даних та очікуваних результатів (мається на увазі використання одного тестового методу для тестування різних ділянок коду в залежності від вхідних даних) було отримано 799 тестових сценаріїв.

Також для програмної системи було застосовано автоматичну генерацію документації з використанням DocFX, що дозволило отримати описову базу коду.

Для опису API було використано Swagger, що надало зручний інтерфейс для роботи з серверними запитами.

5 ЕКОНОМІЧНА ЧАСТИНА

Науково-технічна розробка має право на існування та впровадження, якщо вона відповідає вимогам часу, як в напрямку науково-технічного прогресу та і в плані економіки. Тому для науково-дослідної роботи необхідно оцінювати економічну ефективність результатів виконаної роботи.

Магістерська кваліфікаційна робота з розробки та дослідження «Удосконалення методів та засобів розробки програмного забезпечення для обробки криптовалютних транзакцій» відноситься до науково-технічних робіт, які орієнтовані на виведення на ринок (або рішення про виведення науково-технічної розробки на ринок може бути прийнято у процесі проведення самої роботи), тобто коли відбувається так звана комерціалізація науково-технічної розробки. Цей напрямок є пріоритетним, оскільки результатами розробки можуть користуватися інші споживачі, отримуючи при цьому певний економічний ефект. Але для цього потрібно знайти потенційного інвестора, який би взявся за реалізацію цього проекту і переконати його в економічній доцільності такого кроку.

Для наведеного випадку нами мають бути виконані такі етапи робіт:

1. проведено комерційний аудит науково-технічної розробки, тобто встановлення її науково-технічного рівня та комерційного потенціалу;
2. розраховано витрати на здійснення науково-технічної розробки;
3. розрахована економічна ефективність науково-технічної розробки у випадку її впровадження і комерціалізації потенційним інвестором і проведено обґрунтування економічної доцільності комерціалізації потенційним інвестором.

5.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою «Удосконалення методів та засобів розробки програмного забезпечення для обробки криптовалютних транзакцій» є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної

діяльності.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями, наведеними в таблиці 5.1 [38].

Таблиця 5.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено працездатність продукту в реальних умовах
Ринкові переваги (недоліки)					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в	Технічні та споживчі влас-тивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в	Технічні та споживчі властивості продукту значно кращі, ніж в
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає

Продовження таблиці 5.1

Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуютьс я у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь- які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки потрібно звести до таблиці.

Для проведення технологічного аудиту було залучено 3-х незалежних експертів Вінницького національного технічного університету кафедри програмного забезпечення: Войтко Вікторія Володимирівна (1), Рейда Олександр

Миколайович, Ракитянська Ганна Борисівна (3).

Таблиця 5.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт		
	1	2	3
	Бали:		
1. Технічна здійсненність концепції	4	3	4
2. Ринкові переваги (наявність аналогів)	3	3	3
3. Ринкові переваги (ціна продукту)	3	3	2
4. Ринкові переваги (технічні властивості)	3	2	2
5. Ринкові переваги (експлуатаційні витрати)	2	2	2
6. Ринкові перспективи (розмір ринку)	2	2	2
7. Ринкові перспективи (конкуренція)	2	2	2
8. Практична здійсненність (наявність фахівців)	4	4	4
9. Практична здійсненність (наявність фінансів)	2	3	2
10. Практична здійсненність (необхідність нових матеріалів)	2	2	2
11. Практична здійсненність (термін реалізації)	3	4	4
12. Практична здійсненність (розробка документів)	4	4	4
Сума балів	34	34	33
Середньоарифметична сума балів $СБ_c$	33,7		

За результатами розрахунків, наведених в таблиці 5.2, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в табл. 5.3.

Таблиця 5.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ ,розрахована на основі висновків	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вище середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Удосконалення методів та засобів розробки програмного забезпечення для обробки криптовалютних транзакцій» становить 33,7 бала, що, відповідно до таблиці 4.3, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

5.2 Оцінювання рівня новизни розробки

Виводячи на ринок новинку виробник вважає, що тієї новизни, якою наділена нова розробка є достатньо для того, щоб вона була сприйнята споживачем як нова. Але це не завжди так, в силу того, що споживач і виробник неоднозначно визначать її рівень новизни. Тому доцільним є визначення рівня новизни розробки отриманої в результаті досліджень за темою «Удосконалення методів та засобів розробки програмного забезпечення для обробки криптовалютних транзакцій».

Саме визначення рівня і ступеня інтегральної новизни є найбільш актуальним, оскільки її рівень визначає ступінь однакового позитивного сприйняття новизни розробки як виробником, так і споживачем, а отже і ринком в цілому, а це, у свою чергу, є гарантією того, що новинка знайде своє місце на ринку, користуватиметься попитом у споживачів і забезпечить відшкодування витрат, зазнаних товаровиробником під час розроблення та виробництва технічної розробки [39].

Рівень новизни нової продукції розраховуємо експертним методом шляхом протиставлення нової продукції та її аналогів, що існують в даний час на ринку, за

чинниками що визначають її значення, в системі «краще-гірше». Рівень новизни встановлюємо відносно рівня аналога (або продукту, що досить близький до аналога).

Для визначення *i*-го виду новизни, застосуємо чинники, які впливають на її рівень. Кожен чинник *i*-го виду новизни розраховуємо в балах. Більша кількість набраних балів свідчить про більший рівень новизни. Для оцінювання рівня новизни використаємо думки експертів, які встановлюють визначені бали відповідним чинникам. Бал відповідності проставляється в діапазоні від (-5 –значно гірше аналога до +5 – значно краще аналога). Результати попереднього оцінювання зведемо до відповідного листа оцінювання (таблиця 5.4).

Таблиця 5.4 – Лист оцінювання рівня новизни експертами

Види та чинники		Бали та експерти		
		Войтко В.В	Рейда О.М.	Ракитянскька Г.Б
1		*2*	*3*	*4*
Споживча новизна	Питома вага 0,26	Максимальний бал $Б_{i\ MAX}$		25
1. Зміна поведінкових звичок споживача		2	3	3
2. Ступінь задоволення потреб і запитів		4	4	4
3. Спосіб задоволення потреби		2	2	2
4. Формування нової потреби		4	3	3
5. Формування нового споживача		0	0	0
Середній бал експертів $Б_{i\ отр}$		12		
Товарна новизна	Питома вага 0,21	Максимальний бал $Б_{i\ MAX}$		30
1. Параметричні зміни показників продукції				

Продовження таблиці 5.4

Види та чинники		Бали та експерти		
		Войтко В.В	Рейда О.М.	Ракитянскька Г.Б
1		*2*	*3*	*4*
1.1. Якісні		3	3	4
1.2. Технічні		3	4	3
1.3. Економічні		3	3	3
1.4. Сервісні		4	4	4
2. Якість продукції по відношенню до конкурентів		3	3	3
3. Функціональні зміни		4	4	4
Середній бал експертів $Б_{i\ отр}$		21		
Виробнича новизна	Питома вага 0,014	Максимальний бал $Б_{i\ MAX}$		25
1. Рівень унікальності товару для підприємства		5	5	5
2. Рівень унікальності для галузі		3	3	3
3. Рівень унікальності товару для країни		1	1	1
4. Зміна виробничої системи		4	4	4
5. Відносно існуючого асортименту		3	2	3
Середній бал експертів $Б_{i\ отр}$		16		
Прогресивна новизна	Питома вага 0,2	Максимальний бал $Б_{i\ MAX}$		25
1. Зміна технології виготовлення		4	4	4
2. Рівень застосування нових компонентів і матеріалів		0	0	0
3. Зміна технологічного принципу дії виробу		1	2	1

Продовження таблиці 5.4

Види та чинники		Бали та експерти		
		Войтко В.В	Рейда О.М.	Ракитянскька Г.Б
1		*2*	*3*	*4*
4. Зміна конструктивного виконання		3	3	3
5. Рівень застосування інновацій		2	2	2
Середній бал експертів $Б_{i\ отр}$		10		
Ринкова новизна	Питома вага 0,1	Максимальний бал $Б_{i\ MAX}$		20
1. Новий виріб на новому ринку		0	0	0
2. Новий виріб на відомому ринку		4	4	4
3. Модернізований виріб		3	3	3
4. Нова модель		1	1	1
Середній бал експертів $Б_{i\ отр}$		8		
Екологічна новизна	Питома вага 0,035	Максимальний бал $Б_{i\ MAX}$		20
1. Рівень екологічної чистоти технології виробництва		5	5	5
2. Рівень впровадження мало- та безвідходних технологій		5	5	5
3. Рівень екологічно небезпечних режимів експлуатації продукції		5	5	5
4. Рівень забруднення навколишнього середовища		5	5	5
Середній бал експертів $Б_{i\ отр}$		20		
Соціальна новизна	Питома вага 0,036	Максимальний бал $Б_{i\ MAX}$		20

Продовження таблиці 5.4

Види та чинники		Бали та експерти		
		Войтко В.В	Рейда О.М.	Ракитянскька Г.Б
1		*2*	*3*	*4*
1. Використання нового товару приводить до покращення стану здоров'я нації		0	0	0
2. Використання нового товару приводить до зростання доходів населення		0	0	0
3. Виробництво нового товару приводить до збільшення (зменшення) кількості робочих місць на підприємстві		4	5	4
4. Виробництво нового товару приводить до підвищення кваліфікації персоналу		3	4	3
Середній бал експертів $Б_{i\ отр}$		8		
Маркетингова новизна	Питома вага 0,145	Максимальний бал $Б_{i\ MAX}$		20
1. Нові методи маркетингових досліджень		0	0	0
2. Вживання нових стратегій сегментації ринку		3	3	3
3. Вибір нової маркетингової стратегії обхвату і розвитку цільового сегмента		1	1	1
4. Побудова нових каналів збуту		2	1	1
Середній бал експертів $Б_{i\ отр}$		5		

Значення *i*-го виду новизни розрахуємо за формулою:

$$I_i = \frac{Б_{i\,отр}}{Б_{i\,MAX}}, \tag{5.1}$$

де $Б_{i\ отр}$ – отримана кількість балів за шкалою оцінок чинників, що визначають *i*-й вид новизни;

$Б_{i\ MAX}$ – максимальна кількість балів за *i*-м видом новизни.

Загальний рівень інтегральної новизни розраховуємо шляхом перемноження отриманого значення *i*-го виду новизни на її вагомість, причому вагомість *i*-го виду новизни визначаємо експертним методом, за формулою:

$$N_{інт} = \sum_{i}^{n} W_i \cdot I_i\ , \tag{5.2}$$

де $N_{інт}$ – рівень інтегральної (сукупної) новизни;

W_i – вагомість (питома вага) *i*-го виду новизни;

n – загальна кількість видів новизни.

$$N_{інт} = (0{,}26\cdot 12/25)+(0{,}21\cdot 21/30)+(0{,}014\cdot 16/25)+(0{,}2\cdot 10/25)+(0{,}1\cdot 8/20)+$$
$$(0{,}035\cdot 20/20)+(0{,}036\cdot 8/20)+(0{,}145\cdot 5/20) = 0{,}488.$$

Отримане значення інтегрального рівня новизни зіставляємо зі шкалою, що наведена в таблиці 5.5.

Таблиця 5.5 – Рівні новизни нового товару та їхня характеристика

Рівні новизни товару	Значення інтегральної новизни	Характеристика товару	Вид нового товару
Найвища	1,00	Абсолютно новий товар	Новий товар, що наділений ознаками інноваційності (інноваційний товар)
Висока	0,8…0,99	Товар, який не має аналогів	

Продовження таблиці 5.5

Значуща	0,6…0,79	Принципова зміна споживчих властивостей товару	
Достатня	0,4…0,59	Принципова технологічна модифікація товару	
Незначна	0,2…0,39	Кардинальна зміна параметрів	Новий товар
Помилкова	0,00…0,19	Малоістотна модифікація	

Згідно таблиці 5.5 розробка відповідає рівню при значенні інтегральної новизни 0,488 - достатня новизна; за характеристикою: принципова технологічна модифікація товару; вид розробки - новий товар, що наділений ознаками інноваційності (інноваційний товар).

5.3 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему «Удосконалення методів та засобів розробки програмного забезпечення для обробки криптовалютних транзакцій», під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

5.3.1 Витрати на оплату праці

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці.

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою:

$$З_о = \sum_{i=1}^{k} \frac{М_{ni} \cdot t_i}{Т_р}, \quad (5.3)$$

де k – кількість посад дослідників залучених до процесу досліджень;

$М_{ni}$ – місячний посадовий оклад конкретного дослідника, грн;

t_i – число днів роботи конкретного дослідника, дн.;

$Т_р$ – середнє число робочих днів в місяці, $Т_р$=22 дні.

$$З_о = 22400{,}00 \cdot 32 / 22 = 32581{,}82 \; грн.$$

Проведені розрахунки зведемо до таблиці.

Таблиця 5.6 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник проекту	22400,00	1018,18	32	32581,82
Інженер-дослідник (інженер розробник програмного забезпечення)	20100,00	913,64	30	27409,09
Консультант (фахівець з дослідження генерації та прогнозування криптовалют)	25000,00	1136,36	11	12500,00
Технік	8050,00	365,91	28	10245,45
Всього				82736,36

Основна заробітна плата робітників:

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР на тему «Удосконалення методів та засобів розробки програмного забезпечення для обробки криптовалютних транзакцій» розраховуємо за формулою:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i , \quad (5.4)$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{зм}} , \quad (5.5)$$

де $М_М$ – розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо $Мм$=6700,00 грн;

$К_і$ – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду;

$К_с$ – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об’єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

$T_р$ – середнє число робочих днів в місяці, приблизно $T_р$ = 22 дн;

$t_{зм}$ – тривалість зміни, год.

C_1 = 6700,00 · 1,35 · 1,35 / (22 · 8) = 69,38 *грн.*

$З_{р1}$ = 69,38 · 8,00 = 555,03 *грн.*

Таблиця 5.7 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
Розміщення комп'ютерного обладнання	8,00	3	1,35	69,38	555,03
Інсталяція програмного забезпечення для розробки програмного забезпечення для обробки криптовалютних транзакцій	10,00	5	1,70	87,37	873,66
Підготовка автоматизованого робочого місця розробника ПЗ	5,50	3	1,35	69,38	381,59
Компіляція програмних блоків	14,00	5	1,70	87,37	1223,13
Налагодження програмних блоків	9,50	5	1,70	87,37	829,98
Тестування системи	12,00	2	1,10	56,53	678,38
Всього					4541,77

Додаткова заробітна плата дослідників та робітників

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної

заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (5.6)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 12%.

$З_{дод}$ = (82736,36 + 4541,77) · 12 / 100% = 10473,38 грн.

5.3.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} \quad (5.7)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймаємо 22%.

Зн = (82736,36 + 4541,77 + 10473,38) · 22 / 100% = 21505,33 грн.

5.3.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень за темою «Удосконалення методів та засобів розробки програмного забезпечення для обробки криптовалютних транзакцій».

Витрати на матеріали (*М*), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$М = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j - \sum_{j=1}^{n} В_j \cdot Ц_{вj}, \quad (5.8)$$

де $Н_j$ – норма витрат матеріалу *j*-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу *j*-го найменування, грн/кг;

$К_j$ – коефіцієнт транспортних витрат, ($К_j$ = *1,1 ... 1,15*);

$В_j$ – маса відходів *j*-го найменування, кг;

$Ц_{вj}$ – вартість відходів *j*-го найменування, грн/кг.

$М_1$ = 3 · 225,00 · 1,02 - 0 · 0 = 688,50 грн.

Проведені розрахунки зведемо до таблиці.

Таблиця 5.8 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за 1 кг, грн	Норма витрат, кг	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Офісний папір Ultra Plus	225,00	3	0	0	688,50
Папір для записів Light A5	110,00	4	0	0	448,80
Органайзер офісний	210,00	3	0	0	642,60
Канцелярське приладдя (набір офісного працівника)	186,00	3	0	0	569,16
Картридж для принтера EPSON LaserJet Pro M102w з Wi-Fi (G3Q35A)	2620,00	1	0	0	2672,40
Диск оптичний NewOPtice CD-RW	28,00	1	0	0	28,56
USB-пам'ять Kingston 64 GB	229,00	1	0	0	233,58
Тека для паперів	102,00	4	0	0	416,16
Всього					5699,76

5.3.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі *($К_в$)*, які використовують при проведенні НДР на тему «Удосконалення методів та засобів розробки програмного забезпечення для обробки криптовалютних транзакцій», розраховуємо, згідно з їхньою номенклатурою, за формулою:

$$К_в = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j \quad (5.9)$$

де $Н_j$ – кількість комплектуючих j-го виду, шт.;

$Ц_j$ – покупна ціна комплектуючих j-го виду, грн;

$К_j$ – коефіцієнт транспортних витрат, ($К_j = 1,1 \ldots 1,15$).

$К_в = 1 \cdot 2119,00 \cdot 1,02 = 2161,38$ *грн.*

Проведені розрахунки зведемо до таблиці.

Таблиця 5.9 – Витрати на комплектуючі

Найменування комплектуючих	Кількість, шт.	Ціна за штуку, грн	Сума, грн
Зовнішній жорсткий диск 2.5" 1TB Seagate (STJL1000400)	1	2119,00	2161,38
Відеокарта ASUS GeForce RTX4060 8Gb DUAL OC (DUAL-RTX4060-O8G)	2	14989,00	30577,56
Кулер до відеокарти TTC-SC 07 TZ Titan	2	874,00	1782,96
Всього			34521,90

5.3.5 Спецустаткування для наукових (експериментальних) робіт

До статті «Спецустаткування для наукових (експериментальних) робіт» належать витрати на виготовлення та придбання спецустаткування необхідного для проведення досліджень, також витрати на їх проектування, виготовлення,

транспортування, монтаж та встановлення.

Балансову вартість спецустаткування розраховуємо за формулою:

$$B_{спец} = \sum_{i=1}^{k} Ц_i \cdot C_{пр.i} \cdot K_i \ , \qquad (5.10)$$

де $Ц_i$ – ціна придбання одиниці спецустаткування даного виду, марки, грн;

$C_{пр.i}$ –кількість одиниць устаткування відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує доставку, монтаж, налагодження устаткування тощо, (K_i = *1,10…1,12);*

k – кількість найменувань устаткування.

$B_{спец}$ = *14063,00 · 1 · 1,02 = 14344,26 грн.*

Отримані результати зведемо до таблиці:

Таблиця 5.10 – Витрати на придбання спецустаткування по кожному виду

Найменування устаткування	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Серверна платформа Supermicro CSE-732D4-668B	1	14063,00	14344,26
Маршрутизатор TP-Link ARCHER-MR200	1	3899,00	3976,98
Комп'ютер Vinga Wolverine A5003 (I5M16G3060T.A5003)	1	44320,00	45206,40
Всього			63527,64

5.3.6 Програмне забезпечення для наукових (експериментальних) робіт

До статті «Програмне забезпечення для наукових (експериментальних) робіт» належать витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення, (програм, алгоритмів, баз даних) необхідних для проведення досліджень, також витрати на їх проектування, формування та

встановлення.

Балансову вартість програмного забезпечення розраховуємо за формулою:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i \ , \qquad (5.11)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу даного виду, грн;

$C_{прг.i}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, ($K_i = 1,10…1,12$);

k – кількість найменувань програмних засобів.

$B_{прг} = 5320,00 \cdot 1 \cdot 1,02 = 5426,40$ *грн.*

Отримані результати зведемо до таблиці:

Таблиця 5.11 – Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Пакет забезпечення мови розробки	1	5320,00	5426,40
Середовище розробки WebStorm 2022.3	1	4450,00	4539,00
Фреймворк Angular v13	1	3750,00	3825,00
Абонентна плата доступу до мережі Internet (2 місяці)	1	510,00	520,20
Всього			14310,60

5.3.7 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12}, \quad (5.12)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$A_{обл} = (7280,00 \cdot 2) / (2 \cdot 12) = 606,67$ *грн.*

Проведені розрахунки зведемо до таблиці.

Таблиця 5.12 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Обладнання доступу до мережі Internet	7280,00	2	2	606,67
Обладнання виводу інформації Лазерний принтер EPSON LaserJet Pro M102w з Wi-Fi (G3Q35A)	7560,00	4	2	315,00

Продовження таблиці 5.12

Робоче місце інженера-дослідника спеціалізоване	8320,00	5	2	277,33
Офісна оргтехніка	9900,00	5	2	330,00
Приміщення лабораторії досліджень	440000,00	25	2	2933,33
ПК проектувальника систем	32500,00	3	2	1805,56
Всього				6267,89

5.3.8 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію (B_e) розраховуємо за формулою:

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_e \cdot K_{вni}}{\eta_i}, \qquad (5.13)$$

де W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо Ц$_e$ = 7,50 грн;

$K_{вni}$ – коефіцієнт, що враховує використання потужності, $K_{вni} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$B_e = 0,75 \cdot 200,0 \cdot 7,50 \cdot 0,95 / 0,97 = 1125,00$ *грн.*

Проведені розрахунки зведемо до таблиці.

Таблиця 5.13 – Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Серверна платформа Supermicro CSE-732D4-668B	0,75	200,0	1125,00
Маршрутизатор TP-Link ARCHER-MR200	0,04	200,0	60,00
Комп'ютер Vinga Wolverine A5003 (I5M16G3060T.A5003)	0,42	200,0	630,00
Обладнання доступу до мережі Internet	0,08	240,0	144,00
Обладнання виводу інформації Лазерний принтер EPSON LaserJet Pro M102w з Wi-Fi (G3Q35A)	0,32	3,3	7,92
Робоче місце інженера-дослідника спеціалізоване	0,10	240,0	180,00
Офісна оргтехніка	0,40	1,5	4,50
ПК проектувальника систем	0,25	240,0	450,00
Всього			2601,42

5.3.9 Службові відрядження

До статті «Службові відрядження» дослідної роботи на тему «Удосконалення методів та засобів розробки програмного забезпечення для обробки криптовалютних транзакцій» належать витрати на відрядження штатних

працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов'язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з'їзди, конференції, наради, пов'язані з виконанням конкретних досліджень.

Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{св} = (З_о + З_р) \cdot \frac{Н_{св}}{100\%}, \qquad (5.14)$$

де $Н_{св}$ – норма нарахування за статтею «Службові відрядження», приймемо $Н_{св}$ = 20%.

$В_{св}$ = (82736,36 + 4541,77) · 20 / 100% = 17455,63 грн.

5.3.10 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» розраховуємо як 30…45% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{сп} = (З_о + З_р) \cdot \frac{Н_{сп}}{100\%}, \qquad (5.15)$$

де $Н_{сп}$ – норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації», приймемо $Н_{сп}$= 30%.

$В_{сп}$ = (82736,36 + 4541,77) · 30 / 100% = 26183,44 грн.

5.3.11 Інші витрати

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$І_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \qquad (5.16)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 60%.

$І_в$ = (82736,36 + 4541,77) · 60 / 100% = 52366,88 грн.

5.3.12 Накладні (загальновиробничі) витрати

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов'язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов'язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов'язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$В_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (5.17)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 100%.

$В_{нзв}$ = (82736,36 + 4541,77) · 100 / 100% = 87278,14 грн.

Витрати на проведення науково-дослідної роботи на тему «Удосконалення методів та засобів розробки програмного забезпечення для обробки криптовалютних транзакцій» розраховуємо як суму всіх попередніх статей витрат за формулою:

$$В_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв}. \quad (5.18)$$

$В_{заг}$= 82736,36 +4541,77 +10473,38 +21505,33 +5699,76 +34521,90 +63527,64 + 14310,60 + 6267,89 +2601,42 +17455,63 +26183,44 +52366,88 +87278,14 = =429470,14 грн.

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою:

$$ЗВ = \frac{В_{заг}}{\eta}, \quad (5.19)$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η=0,9.

$$ЗВ = 429470,14 / 0,9 = 477189,04 \text{ грн.}$$

5.4 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку.

Результати дослідження проведені за темою «Удосконалення методів та засобів розробки програмного забезпечення для обробки криптовалютних транзакцій» передбачають комерціалізацію протягом 4-х років реалізації на ринку і відповідають напряму «*Розробка чи суттєве вдосконалення програмного засобу (програмного забезпечення, програмного продукту) для використання масовим споживачем».*

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

ΔN – збільшення кількості споживачів продукту, у періоди часу, що аналізуються, від покращення його певних характеристик;

Таблиця 5.14 – Прогноз збільшення кількості споживачів

Показник	1-й рік	2-й рік	3-й рік	4-й рік
Збільшення кількості споживачів, осіб	1000	2000	2500	1500

N – кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки, приймемо 15000 осіб;

$Ц_б$ – вартість програмного продукту у році до впровадження результатів

розробки, приймемо 6500,00 грн;

$\pm\Delta Ц_o$ – зміна вартості програмного продукту від впровадження результатів науково-технічної розробки, приймемо 644,50 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 4-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою:

$$\Delta П_i = (\pm\Delta Ц_o \cdot N + Ц_o \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \quad (5.20)$$

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2023 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0,8333$;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту). Приймемо $\rho = 42\%$;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2023 році $\vartheta = 18\%$;

Збільшення чистого прибутку 1-го року:

$\Delta П_1 = (644,50 \cdot 15000,00 + 7144,50 \cdot 1000) \cdot 0,83 \cdot 0,42 \cdot (1-0,18/100\%) = 4805743,82$ грн.

Збільшення чистого прибутку 2-го року:

$\Delta П_2 = (644,50 \cdot 15000,00 + 7144,50 \cdot 3000) \cdot 0,83 \cdot 0,42 \cdot (1-0,18/100\%) = 8890283,05$ грн.

Збільшення чистого прибутку 3-го року:

$\Delta П_3 = (644,50 \cdot 15000,00 + 7144,50 \cdot 5500) \cdot 0,83 \cdot 0,42 \cdot (1-0,18/100\%) = 13995957,09$ грн.

Збільшення чистого прибутку 4-го року:

$\Delta П_4 = (644,50 \cdot 15000,00 + 7144,50 \cdot 7000) \cdot 0,83 \cdot 0,42 \cdot (1-0,18/100\%) = 17059361,51$

грн.

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \quad (5.21)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ =0,2;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$$ПП = 4805743,82/(1+0,2)^1+8890283,05/(1+0,2)^2+13995957,09/(1+0,2)^3+$$
$$+17059361,51/(1+0,2)^4=4004786,52+6173807,68+8099512,20+8226929,74=$$
$$=26505036,14 \text{ грн.}$$

Величина початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки:

$$PV = k_{інв} \cdot ЗВ, \quad (5.22)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$=2;

ЗВ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 477189,04 грн.

$PV = k_{інв} \cdot ЗВ = 2 \cdot 477189{,}04 = 954378{,}09$ грн.

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV \tag{5.23}$$

де $ПП$ – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 26505036,14 грн;

PV – теперішня вартість початкових інвестицій, 954378,09 грн.

$E_{абс} = ПП - PV = 26505036{,}14 - 954378{,}09 = 25550658{,}05$ грн.

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1, \tag{5.24}$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, 25550658,05 грн;

PV – теперішня вартість початкових інвестицій, 954378,09 грн;

$T_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 4 роки.

$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1 = (1+25550658{,}05/954378{,}09)^{1/4} = 1{,}30.$

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$:

$$\tau_{мін} = d + f, \tag{5.25}$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2023 році в Україні d =0,09;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,33.

$\tau_{мін}$ = 0,09+0,33 = 0,42 < 1,30 свідчить про те, що внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою «Удосконалення методів та засобів розробки програмного забезпечення для обробки криптовалютних транзакцій» доцільно.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_в}, \tag{5.26}$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$T_{ок}$ = 1 / 1,30 = 0,77 р.

$T_{ок}$ < 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

5.5 Висновки

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Удосконалення методів та засобів розробки програмного забезпечення для обробки криптовалютних транзакцій» становить 33,7 бала, що, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

Прогнозування витрат на виконання науково-дослідної роботи по кожній з статей витрат складе 477 189,04 грн. Загальна ж величина витрат на виконання та впровадження результатів даної НДР буде складати 477 189,04 грн.

Також термін окупності становить 0,77 р., що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою «Удосконалення методів та засобів розробки програмного забезпечення для обробки криптовалютних транзакцій».

ВИСНОВОК

У магістерської кваліфікаційної роботі по удосконаленню методів та засобів розробки програмного забезпечення для обробки криптовалютних транзакцій було проаналізовано поточний стан криптовалютних систем. Було розглянуто основні аналоги криптовалют, такі, як Bitcoin, Ethereum, Litecoin. Було визначено їхні недоліки та порівняно з власним програмним продуктом. Беручи до уваги результати порівняння було встановлено основні задачі магістерської кваліфікаційної роботи.

Було розглянуто архутектурні підходи для розробки програмного забезпечення з обробки криптовалютних транзакцій. На основі аналізу основних підходів для побудови архітектури програмної системи було запропоновано використання комбінованого архітектурного підходу на основі мікросервсного, шарового та Event-Driven підходів до побудови архітектури програмної системи.

Згідно архітектурного планування систему було розділено на декілька сервісів:

1. Transaction Service – центральний сервіс обробки криптовалютних транзакцій, валідації інформаційних вузлів системи та забезпеченні збереження перевірених даних у сховищі.
2. Customer Service – інтегрований сервіс для взаємодії користувачів системи з криптовалютним гаманцем, де користувач вводячи обікові дані від влсного гаманця може отрмати доступ до інформаціїї по існуючим транзакціям, що вже пройшли верифікацію та збережені у сховищі або ще знаходяться у процесі перевірки та збереження.
3. Blockchain Service – сервіс для забезпечення цілісності та безпеки криптовалютних транзакцій шляхом шляхом формування зв’язку між криптовалютними транзакціями.

Було сформовано послідовніть життевого циклу транзакції, від ініціації створення користувачем через сервіс взаємодії з електронним гаманцем, до обробки сервісом транзакцій з подальшим хешуванням та збереженням у сховищі

даних. Розроблено підхід до хешування даних, з використання алгоритмів хешування. Розроблено алгоритми забезпечення секретності даних, пов’язаних з обліковими записами користувачів.

За рахунок розробленої архітектруи було досягнуто використання цетралізованого підходу до створення криптовалютної системи за рахунок створення.

Розробювана криптовалютна система надає можливість змінювати параметри формування блоків даних для збереження інформації за рахунок оптимізації умов безпеки у періоди високого навантаження, що дозволяє підвищити рівень адаптивності системи в залежності від навантаження мережі.

Сервіс користувачів визначено як інтегровану систему доступу до електронного гаманця, яка, на відміну від існуючих, має розширений функціонал з можливістю відновлення доступу при втраті облікових даних та має вдосконалену систему пошуку криптовалютних транзакцій, яка, на відміну від існуючих, дозволяє використовувати гнучкі фільтри, що підвищує швидкість та збільшує глибину пошуку.

Тестування модулів системи за допомогою Unit-тести довело працездатність кожного з них. В загальному було написано 518 тестів та на основі них створено 799 тестових сценаріїв. Також проведено E2E тестування сервісів, що довело працездатність системи.

Для кодової бази було створено описову документацію використовуючи DocFX, де надано про кожен з функціональних вузлів системи та візувальзовано в графічному форматі сигнатуру кожного модуля. Для серверної API частини було згенеровано користувацький інтерфейс за допомогою Swagger для більш зручного доступу до відритих API-методів

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Черноволик Г.О. Розробка екосистеми для емісії та переказу криптовалюти / Г.О. Черноволик., С.В. Бевз, С.М. Бурбело, В.В. Войтко, І..С.Мельник // Матеріали LI Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2022) ВНТУ, Факультет інформаційних технологій та комп'ютерної інженерії, Секція програмного забезпечення . [Інтернет]. [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2022/paper/view/15743/13210

2. Blockchain [Електронний ресурс] – Режим доступу до ресурсу: https://www.investopedia.com/terms/b/blockchain.asp

3. Криптовалюта [Електронний ресурс] – Режим доступу до ресурсу: https://alpari.com/en/beginner/glossary/cryptocurrency/

4. Centralized cryptocurrency [Електронний ресурс] – Режим доступу до ресурсу: https://cointelegraph.com/learn/centralized-vs-decentralized-crypto-exchanges

5. Decentralized cryptocurrency [Електронний ресурс] – Режим доступу до ресурсу: https://www.sciencedirect.com/science/article/abs/pii/S092911992300007X

6. Bitcoin [Електронний ресурс] – Режим доступу до ресурсу: https://bitcoin.org/uk/resources

7. Ethereum [Електронний ресурс] – Режим доступу до ресурсу: https://ethereum.org/en/what-is-ethereum/

8. Litecoin [Електронний ресурс] – Режим доступу до ресурсу: https://dev.litecoin.org

9. Arhitecture aproaches [Електронний ресурс] – Режим доступу до ресурсу:https://www.indeed.com/career-advice/career-development/what-is-application-architecture

10. C++ [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/cpp/cpp/cpp-language-reference?view=msvc-170

11. Java [Електронний ресурс] – Режим доступу до ресурсу: https://dev.java

12. C# Fundamentals - C# 10 and .NET 6 using Visual Studio 2022: Course in a book : навч. посіб. / Adam Seebeck – unQbd, 2021. – 168 ст.

13. .NET CORE – https://learn.microsoft.com/en-us/dotnet/core/introduction

14. Visual Studio [Електронний ресурс] – Режим доступу до ресурсу: https://visualstudio.microsoft.com/vs/

15. MS SQL Server [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver16

16. Stored procedures [Електронний ресурс] – Режим доступу до ресурсу: https://www.youtube.com/watch?v=Sggdhot-MoM

17. Direct Queries [Електронний ресурс] – Режим доступу до ресурсу: https://skypoint.ai/blog/using-direct-query-a-tale-from-the-trenches/

18. ASP .Net Core [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-6.0

19. Razor [Електронний ресурс] – Режим доступу до ресурсу: https://www.udemy.com/course/aspnet-core-razor-pages-net-6/

20. HTML [Електронний ресурс] – Режим доступу до ресурсу: https://www.udemy.com/course/html5-oz/

21. Bootstrap [Електронний ресурс] – Режим доступу до ресурсу: https://getbootstrap.com/docs/5.0/getting-started/introduction/

22. WebAPI [Електронний ресурс] – Режим доступу до ресурсу: https://www.youtube.com/watch?v=vN9NRqv7xmY

23. REST [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/rest-api-introduction/

24. Cookie Authentication in Net [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/aspnet/core/security/authentication/cookie

25. OWIN [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/owin?view=aspnetcore-8.0

26. Hashing in .NET [Електронний ресурс] – Режим доступу до ресурсу: https://jintechflow.wordpress.com/2021/06/28/hash-algorithm-in-net-core/

27. SSL [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/what-is/ssl-certificate/

28. SignalR [Електронний ресурс] – Режим доступу до ресурсу: https://ably.com/topic/signalr-deep-dive

29. Web sockets [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

30. AJAX [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/xml/ajax_xmlhttprequest_send.asp

31. MS Test [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/core/testing/unit-testing-with-mstest

32. TDD [Електронний ресурс] – Режим доступу до ресурсу: https://www.guru99.com/test-driven-development.html

33. NSubstitute [Електронний ресурс] – Режим доступу до ресурсу: https://nsubstitute.github.io/help/getting-started/

34. AAA pattern [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@pjbgf/title-testing-code-ocd-and-the-aaa-pattern-df453975ab80

35. Postman [Електронний ресурс] – Режим доступу до ресурсу: https://www.postman.com

36. Swagger [Електронний ресурс] – Режим доступу до ресурсу: https://swagger.io

37. DocFX [Електронний ресурс] – Режим доступу до ресурсу: https://dotnet.github.io/docfx/

38. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. – Вінниця : ВНТУ, 2021. – 42 с.

39. Кавецький В. В. Економічне обґрунтування інноваційних рішень: практикум / В. В. Кавецький, В. О. Козловський, І. В. Причепа – Вінниця : ВНТУ, 2016. – 113 с.

121

Додаток А
(обов'язковий)

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедрою ПЗ
к.т.н., проф.
_____ О. Н. Романюк
"20" вересня 2023 р.

Технічне завдання
на магістерську кваліфікаційну роботу
«Удосконалення методів та засобів розробки програмного забезпечення для обробки криптовалютних транзакцій»
за спеціальністю 121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:
_____ к.т.н., доцент кафедри ПЗ, Коваленко О.О.
"20" вересня 2023 р.
Виконав:
_____ студент гр. 2ПІ-22м Мельник І.С.
"20" вересня 2023 р.

Вінниця – 2023 року

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Удосконалення методів та засобів розробки програмного забезпечення для обробки криптовалютних транзакцій».

Галузь застосування – електронні фінансові операції.

2. Підстава для розробки.

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на МКР та наказ № 247 від «18» вересня 2023 р. ректора ВНТУ про закріплення тем МКР.

3. Мета та призначення розробки.

Метою роботи є підвищення стабільності платіжної системи шляхом розробки власної централізованої криптовалюти, що дозволяє гарантувати безперервність та незалежність процесу емісії й обробки транзакцій.

Основними задачами дослідження є:

– провести аналіз існуючих методів і засобів розробки програмного забезпечення для обробки криптовалютних транзакцій;

– удосконалити метод використанян цетралізованого підходу до створення криптовалютної системи;

– розробити програмні компоненти для централізованої обробки криптовалютних тразакцій;

– провести експериментальні дослідження розроблених програмних засобів.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись МКР:

1. Digital currency [Електронний ресурс] – Режим доступу до ресурсу: https://www.thalesgroup.com/en/markets/digital-identity-and-security/banking-payment/digital-currency

2. Cryptocurrency [Електронний ресурс] – Режим доступу до ресурсу: https://alpari.com/en/beginner/glossary/cryptocurrency/

3. C# Fundamentals – C# 10 and .NET 6 using Visual Studio 2022: Course in

a book : навч. посіб. / Adam Seebeck – unQbd, 2021. – 168 ст.

4. Hashing in .NET [Електронний ресурс] – Режим доступу до ресурсу: https://jintechflow.wordpress.com/2021/06/28/hash-algorithm-in-net-core/

5. SignalR [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr

6. REST [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/rest-api-introduction

7. Cookie Authentication in Net [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/aspnet/core/security/authentication/cookie

8. TDD [Електронний ресурс] – Режим доступу до ресурсу: https://www.guru99.com/test-driven-development.html

9. MS SQL Server [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver16

5. Технічні вимоги

- середовище розробки – Microsoft Visual Studio 2022;
- мова програмування – C#;
- допоміжні засоби – ASP.NET, JQuery, SignalR;

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред’являється по закінченню робіт:

1. Пояснювальна записка до МКР;
2. Технічне завдання;
3. Лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів магістерської кваліфікаційної Роботи	Строк виконання етапів роботи
1	Аналіз існуючих криптовалютних систем	20.09.23 – 01.10.23
2	Побудова архутектури компонентів системи	02.10.23 – 02.11.23
3	Аналіз і вибір інтрументів для розробки програмної системи	03.11.23 – 05.11.23
4	Розробка сервісів системи	06.11.23 – 19.11.23
5	Написання тестів для перевірки внутрішнії компонентіів системи	20.11.23 – 26.11.23
6	E2E тестування криптовалютної системи	27.11.23 – 30.11.23
7	Створення документації для кодової бази	31.11.23 – 02.12.23
8	Розробка економічної частини	18.11.23 –19.11.23
9	Оформлення матеріалів до захисту МКР	02.12.23 – 04.12.23

10. Порядок контролю та прийняття

Порядок контролю і приймання роботи регламентується відповідними документами ВНТУ і державними стандартами.

125

Додаток Б

(обов'язковий)

ПРОТОКОЛ ПЕРЕВІРКИ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Удосконалення методів та засобів розробки програмного забезпечення для обробки криптовалютних транзакцій.

Тип роботи: магістерська кваліфікаційна робота

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: Коваленко О. О.

Оригінальність	89,4 %
Схожість	10.6 %

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи Мельник І. С.

Керівник роботи Коваленко О. О.

Додаток В

(довідковий)
Лістинг програмного коду криптовалютної системи

TransactionService.cs

```
using System;
using
System.Collections.Concurrent;
using
System.Collections.Generic;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using
CryptoCurrency.Common.BL;
using
CryptoCurrency.Common.Inter
faces;
using
CryptoCurrency.Common.Mod
els;
using
CryptoCurrency.DAL.Interface
s;
using
CryptoCurrency.TransactionSer
vice.Interfaces;
using
CryptoCurrency.TransactionSer
vice.Models;
using
Microsoft.Extensions.Logging;

namespace
CryptoCurrency.TransactionSer
vice.BL
{
   /// <summary>
   /// Class for transaction
service representation.
   /// </summary>
   /// <remarks>
   ///
[![Flowchart](../images/TS.jpg)
](../images/TS.jpg)
   /// </remarks>
   public partial class
TransactionService :
ITransactionCreator,
IMiningServiceHost,
IWalletProvider,
IObservableService<Transactio
nService>, IWalletContainer
   {
      private const int
TotalCoinsLimit = 44500000;
      private readonly long?
systemWalletId = null;

      /// <summary>
      /// Stores the pool of
unprocessed transactions.
      /// </summary>
      private readonly
SortedConcurrentDictionary<st
ring, Transaction>
transactionsPool;

      /// <summary>
      /// Stores the chain of valid
blocks.
      /// </summary>
      private readonly
ConcurrentStack<Block>
blockchain;

      /// <summary>
      /// Queue of blocks, which
waiting their validation.
      /// </summary>
      private readonly
ConcurrentQueue<Block>
blockValidationQueue;
      private readonly ISettings
settings;
      private readonly
IBlockCalculationUtils
blockCalculationUtils;
      private readonly object
startValidationLocker = new ();
      private readonly
IMonitorLocker locker;
      private readonly
IGUIDGenerator
guidGenerator;
      private readonly
ITransactionsDAL
transactionsDAL;
      private readonly
IWalletsDAL walletsDAL;
      private readonly
IBlocksDAL blocksDAL;
      private readonly
ILogger<TransactionService>
logger;
      private readonly string
systemWalletPublicKey =
"0000000000000000000000000
00000000";
      private bool
isValidationActive = false;

      /// <summary>
      /// Stores the users wallet
list.
      /// </summary>
      private
ConcurrentDictionary<string,
Wallet> wallets;

      /// <summary>
      /// Initializes a new
instance of the <see
cref="TransactionService"/>
class.
      /// </summary>
      /// <param
name="settings">Transaction
service settings.</param>
      /// <param
name="locker">Locker for
adding miners blocks.</param>
      /// <param
name="calculationsUtils">Utils
for block hash calculation and
validation.</param>
      /// <param
name="guidGenerator">Unique
ID generator.</param>
      /// <param
name="readerWriterLocker">R
ead/Write locker for sorted
concurrent dictionary.</param>
      /// <param
name="transactionsDAL">Data
access layer for
transactions.</param>
      /// <param
name="walletsDAL">Data
access layer for
wallets.</param>
      /// <param
name="blocksDAL">Data
access layer for
blocks.</param>
      /// <param
```

```
name="logger">Interface for logging.</param>
        public TransactionService(ISettings settings, IMonitorLocker locker, IBlockCalculationUtils calculationsUtils, IGUIDGenerator guidGenerator, IReaderWriterLocker readerWriterLocker, ITransactionsDAL transactionsDAL, IWalletsDAL walletsDAL, IBlocksDAL blocksDAL, ILogger<TransactionService> logger)
        {
            wallets = new ConcurrentDictionary<string, Wallet>();
            transactionsPool = new SortedConcurrentDictionary<string, Transaction>(readerWriterLocker);
            blockchain = new ConcurrentStack<Block>();
            blockValidationQueue = new ConcurrentQueue<Block>();
            this.settings = settings;
            this.locker = locker;
            blockCalculationUtils = calculationsUtils;
            this.guidGenerator = guidGenerator;
            this.transactionsDAL = transactionsDAL;
            this.walletsDAL = walletsDAL;
            this.blocksDAL = blocksDAL;
            this.logger = logger;
        }

        /// <inheritdoc cref="IObservableService.Notify"/>
        public event NotifyDelegate Notify;

        /// <summary>
        /// Starts transaction service.
        /// </summary>
        public void Start()
        {
            var dbTransactions = transactionsDAL.GetAllTransactions();

            RecreateBlockchain(dbTransactions);

            RecreateWallets(dbTransactions);

            RecreateTransactionsPool(dbTransactions);
            CreateSystemWallet();
        }

        /// <inheritdoc cref="ITransactionCreator.CreateTransaction(string, string, decimal, out string)"/>
        public bool CreateTransaction(string senderPublicKey, string receiverPublicKey, decimal amount, out string response)
        {
            if (senderPublicKey == receiverPublicKey)
            {
                response = TransactionServiceOperationResultMessages.SenderAndReciverAreSame;
                return false;
            }

            if (!wallets.ContainsKey(senderPublicKey))
            {
                response = TransactionServiceOperationResultMessages.SenderDoesNotExist;
                return false;
            }

            Wallet sender = wallets[senderPublicKey];
            if (!wallets.ContainsKey(receiverPublicKey))
            {
                response = TransactionServiceOperationResultMessages.ReceiverDoesNotExist;
                return false;
            }

            if (sender.Balance < amount)
            {
                response = TransactionServiceOperationResultMessages.NotEnoughCoins;
                return false;
            }

            if (amount <= 0)
            {
                response = TransactionServiceOperationResultMessages.InvalidTransaferAmount;
                return false;
            }

            var transaction = new Transaction(senderPublicKey, receiverPublicKey, amount, guidGenerator.GetNewGUID());

            transactionsDAL.SaveTransaction(transaction);

            sender.SaveInHistory(transaction);

            AddTransactionToPool(transaction);


            logger.LogInformation("Transaction {@transaction} created successfully.", transaction);
            response = TransactionServiceOperationResultMessages.TransactionSuccessfullyCreated;
            return true;
        }

        /// <inheritdoc cref="ITransactionCreator.CreateTransactionAsync(string, string, decimal)"/>
        public Task<CreateTransactionResponse> CreateTransactionAsync(string sender, string receiver, decimal amount)
```

```
        {
            var operationResult =
CreateTransaction(sender,
receiver, amount, out string
response);
            var result = new
CreateTransactionResponse(op
erationResult, response);
            return
Task.FromResult(result);
        }

        /// <summary>
        /// Creates new wallet and
adds it to the Wallets list and to
the database.
        /// </summary>
        /// <param
name="userId">Wallet owner
id.</param>
        /// <returns>
        /// 'public key' of new
wallet, if creating and adding to
wallets list successful;
        /// empty string, if adding
new wallet to wallets list failed.
        /// </returns>
        public virtual string
CreateWallet(long? userId)
        {
            var wallet = new
Wallet(guidGenerator.GetNew
GUID(), userId);
            if
(wallets.TryAdd(wallet.PublicK
ey, wallet))
            {

walletsDAL.AddWallet(wallet)
;
                return
wallet.PublicKey;
            }

            return string.Empty;
        }

        /// <summary>
        /// Gets wallet from wallets
list.
        /// </summary>
        /// <param
name="identificator">Wallets
public key.</param>
        /// <returns>
        /// Wallet, if it exist by
public key;
        /// null, if wallet not exist
by public key.
        /// </returns>
        public Wallet
GetWallet(string identificator)
        {
            if
(wallets.ContainsKey(identifica
tor))
            {
                return
wallets[identificator];
            }

            return null;
        }

        /// <inheritdoc
cref="IWalletContainer.GetWal
letByUserId(long)"/>
        public
IObservableService<Wallet>
GetWalletByUserId(long
userId)
        {
            return
wallets.Values.FirstOrDefault(
wallet => wallet.UserId ==
userId);
        }

        /// <inheritdoc
cref="IMiningServiceHost.Add
Block(Block)"/>
        public void
AddBlock(Block block)
        {

blockValidationQueue.Enqueue
(block);


logger.LogInformation("Block
{@block} prepared for
validation.", block);
            if (!isValidationActive)
            {

locker.Enter(startValidationLoc
ker);
                if
(!isValidationActive)
                {
                    isValidationActive
= true;

locker.Exit(startValidationLock
er);
                    StartValidation();
                }
                else
                {

locker.Exit(startValidationLock
er);
                }
            }
        }

        /// <inheritdoc
cref="IMiningServiceHost.Add
BlockAsync(Block)"/>
        public Task
AddBlockAsync(Block block)
        {
            AddBlock(block);
            return
Task.CompletedTask;
        }

        /// <inheritdoc
cref="IMiningServiceHost.Get
MiningProcessInformation(lon
g)"/>
        public
MiningProcessInformation
GetMiningProcessInformation(l
ong userId)
        {
            string errorMessage;
            try
            {
                if
(isValidationActive)
                {
                    errorMessage =
TransactionServiceOperationRe
sultMessages.ValidationProcess
IsActive;
                    return new
MiningProcessInformation(null
, null, null, errorMessage);
                }

                if
(!blockchain.TryPeek(out
Block previousBlock))
                {
                    previousBlock =
new Block(string.Empty, new
List<Transaction>())
                    {
                        Hash =
string.Empty,
```

```
            };
        }

        var transactions =
new List<Transaction>();

        var minersWallet =
wallets.FirstOrDefault(item =>
item.Value.UserId ==
userId).Value;
        if (minersWallet ==
null)
        {
            errorMessage =
TransactionServiceOperationRe
sultMessages.WalletIsNotFoun
d;
            return new
MiningProcessInformation(null
, null, null, errorMessage);
        }

        var reward = new
Transaction(systemWalletPubli
cKey, minersWallet.PublicKey,
settings.Reward,
guidGenerator.GetNewGUID())
;

transactions.Add(reward);

        int transactionLimit =
settings.MaxCountBlockTransa
ctions - 1;


transactions.AddRange(transact
ionsPool.GetFirstElements(tran
sactionLimit));

        errorMessage =
TransactionServiceOperationRe
sultMessages.MiningProcessInf
ormationPreparedSuccessfully;
        var mpi = new
MiningProcessInformation(prev
iousBlock.Hash, transactions,
settings.HashZeroNumber,
errorMessage);


logger.LogInformation("Mining
process information {@mpi}
successfully built.", mpi);
        return mpi;
      }
      catch (Exception ex)
      {

logger.LogWarning(ex,
"Something went wrong when
trying to get mining process
information");
        errorMessage =
ex.Message;
        return new
MiningProcessInformation(null
, null, null, errorMessage);
      }
    }

    /// <inheritdoc
cref="IMiningServiceHost.Get
MiningProcessInformationAsy
nc(long)"/>
    public
Task<MiningProcessInformatio
n>
GetMiningProcessInformation
Async(long userId)
    {
      var mpi =
GetMiningProcessInformation(
userId);
      return
Task.FromResult(mpi);
    }

    /// <inheritdoc
cref="IWalletProvider.GetOrCr
eateWalletAsync(GetWalletMo
del)"/>
    public async
Task<ResponseResult<WalletI
nfo>>
GetOrCreateWalletAsync(Get
WalletModel model)
    {
      var wallet =
GetWallet(model.UserId);
      if (wallet == null)
      {
        wallet = new
Wallet(guidGenerator.GetNew
GUID(), model.UserId);
        try
        {
          if
(walletsDAL.AddWallet(wallet
))
          {

wallets.TryAdd(wallet.PublicK
ey, wallet);

logger.LogInformation("A new
wallet with
PublicKey={walletPublicKey}
for user with ID={userId} is
successfully created.",
wallet.PublicKey,
model.UserId);
          }
          else
          {

logger.LogError("AddWallet
didn't make any changes in DB,
and didn't throw an
exception.");
            wallet = null;
          }
        }
        catch (Exception ex)
        {

logger.LogError(ex,
"Something went wrong during
creating new wallet for user
with ID={userId}",
model.UserId);
          wallet = null;
        }
      }


ResponseResult<WalletInfo>
getWalletResponse;
      if (wallet != null)
      {
        getWalletResponse =
new
ResponseResult<WalletInfo>
        {
          Model = wallet,
          Message =
string.Empty,
          IsSuccessful = true,
        };
      }
      else
      {
        getWalletResponse =
new
ResponseResult<WalletInfo>
        {
          Model = null,
          Message =
ResponseMessages.InternalErro
r,
          IsSuccessful =
```

```
false,
            };
        }

        return await
Task.FromResult(getWalletRes
ponse);
    }

    /// <inheritdoc
cref="IWalletProvider.GetBala
nceAsync(GetWalletBalanceM
odel)"/>
    public async
Task<ResponseResult<decimal
>>
GetBalanceAsync(GetWalletBa
lanceModel model)
    {
        var wallet =
GetWallet(model.UserId);

ResponseResult<decimal>
getWalletResponse;
        if (wallet != null)
        {
            getWalletResponse =
new ResponseResult<decimal>
            {
                Model =
wallet.Balance,
                Message =
string.Empty,
                IsSuccessful = true,
            };
        }
        else
        {
            getWalletResponse =
new ResponseResult<decimal>
            {
                Model = default,
                Message =
ResponseMessages.InternalErro
r,
                IsSuccessful =
false,
            };
        }

        return await
Task.FromResult(getWalletRes
ponse);
    }

    /// <inheritdoc
cref="IWalletProvider.GetWall
etTransactionsAsync(GetWallet
TransactionsModel)"/>
    public async
Task<ResponseResult<Paginat
edTransactions>>
GetWalletTransactionsAsync(G
etWalletTransactionsModel
model)
    {

ResponseResult<PaginatedTran
sactions>
getWalletTransactionsResponse
;
        var wallet =
GetWallet(model.UserId);
        if (wallet == null)
        {

logger.LogError("Wallet is not
found for a user with
ID={userId}.", model.UserId);

getWalletTransactionsResponse
= new
ResponseResult<PaginatedTran
sactions>
            {
                Model = null,
                Message =
ResponseMessages.InternalErro
r,
                IsSuccessful =
false,
            };
        }
        else
        {
            var pageInfo = new
PageInfo();
            var
walletTransactions =
wallet.History;

PaginatedTransactions
paginatedTransactions;
            if
(!walletTransactions.Any())
            {

paginatedTransactions = new
PaginatedTransactions()
                {
                    Transactions =
walletTransactions,
                    PageInfo =
pageInfo,
                };
            }
            else
            {
                if
(!string.IsNullOrWhiteSpace(m
odel.Wallet))
                {

walletTransactions =
walletTransactions.Where(t =>
t.Sender == model.Wallet ||
t.Receiver == model.Wallet);
                }

                if
(model.DateFrom != null)
                {

walletTransactions =
walletTransactions.Where(t =>
t.DateLastUpdated >=
model.DateFrom);
                }

                if (model.DateTo
!= null)
                {

walletTransactions =
walletTransactions.Where(t =>
t.DateLastUpdated <
model.DateTo.Value.AddDays(
1));
                }

                if (model.Type !=
TransactionsTypeFilter.Any)
                {
                    if (model.Type
==
TransactionsTypeFilter.Incomi
ng)
                    {

walletTransactions =
walletTransactions.Where(t =>
t.Receiver ==
wallet.PublicKey);
                    }
                    else if
(model.Type ==
TransactionsTypeFilter.Outgoin
g)
                    {

walletTransactions =
```

```
walletTransactions.Where(t =>
t.Sender == wallet.PublicKey);
                }
            }

            if (model.Status !=
TransactionsStatusFilter.Any)
            {
                if (model.Status
==
TransactionsStatusFilter.Unpro
cessed)
                {

walletTransactions =
walletTransactions.Where(t =>
t.Status ==
Status.Unprocessed);
                }
                else if
(model.Status ==
TransactionsStatusFilter.Proces
sed)
                {

walletTransactions =
walletTransactions.Where(t =>
t.Status == Status.Processed);
                }
            }

            if (model.Sort ==
TransactionsSort.DateDesc)
            {

walletTransactions =
walletTransactions.OrderByDes
cending(t =>
t.DateLastUpdated);
            }
            else if (model.Sort
== TransactionsSort.DateAsc)
            {

walletTransactions =
walletTransactions.OrderBy(t
=> t.DateLastUpdated);
            }
            else if (model.Sort
==
TransactionsSort.AmountDesc)
            {
                var
receivedTransactions =
walletTransactions.Where(t =>
t.Receiver ==
wallet.PublicKey).OrderByDes
cending(t =>
t.TransferAmount);
                var
sendedTransactions =
walletTransactions.Where(t =>
t.Sender ==
wallet.PublicKey).OrderBy(t
=> t.TransferAmount);

walletTransactions =
receivedTransactions.Concat(se
ndedTransactions);
            }
            else if (model.Sort
==
TransactionsSort.AmountAsc)
            {
                var
receivedTransactions =
walletTransactions.Where(t =>
t.Receiver ==
wallet.PublicKey).OrderBy(t
=> t.TransferAmount);
                var
sendedTransactions =
walletTransactions.Where(t =>
t.Sender ==
wallet.PublicKey).OrderByDes
cending(t =>
t.TransferAmount);

walletTransactions =
sendedTransactions.Concat(rec
eivedTransactions);
            }

            if
(walletTransactions.Any())
            {

pageInfo.TotalItems =
walletTransactions.Count();
                if (model.Page
>= 1)
                {
                    if
(model.Page >
pageInfo.TotalPages)
                    {

pageInfo.PageNumber =
pageInfo.TotalPages;
                    }
                    else
                    {

pageInfo.PageNumber =
model.Page;
                    }
                }


walletTransactions =
walletTransactions

.Skip((pageInfo.PageNumber -
1) * pageInfo.PageSize)

.Take(pageInfo.PageSize);
            }


paginatedTransactions = new
PaginatedTransactions()
            {
                Transactions =
walletTransactions,
                PageInfo =
pageInfo,
            };
        }


getWalletTransactionsResponse
= new
ResponseResult<PaginatedTran
sactions>
            {
                Model =
paginatedTransactions,
                Message =
string.Empty,
                IsSuccessful = true,
            };
        }

        return await
Task.FromResult(getWalletTra
nsactionsResponse);
    }

    /// <inheritdoc
cref="IWalletProvider.GetWall
etTransactionAsync(GetWallet
TransactionModel)"/>
    public async
Task<ResponseResult<GetWall
etTransactionResponseModel>
>
GetWalletTransactionAsync(Ge
tWalletTransactionModel
model)
    {
```

```
ResponseResult<GetWalletTra
nsactionResponseModel>
getWalletTransactionResponse;
        var wallet =
GetWallet(model.UserId);
        if (wallet == null)
        {

logger.LogError("Wallet is not
found for a user with
ID={userId}.", model.UserId);

getWalletTransactionResponse
= new
ResponseResult<GetWalletTra
nsactionResponseModel>()
            {
                Model = null,
                Message =
ResponseMessages.InternalErro
r,
                IsSuccessful =
false,
            };
        }
        else
        {
            var
walletTransactions =
wallet.History;
            var transaction =
walletTransactions.FirstOrDefa
ult(t => t.Id ==
model.TransactionId);
            var responseModel =
new
GetWalletTransactionResponse
Model()
            {
                Transaction =
transaction,
                WalletPublicKey =
wallet.PublicKey,
            };

getWalletTransactionResponse
= new
ResponseResult<GetWalletTra
nsactionResponseModel>()
            {
                Model =
responseModel,
                Message =
string.Empty,
                IsSuccessful = true,
            };
        }
            return await
Task.FromResult(getWalletTra
nsactionResponse);
        }

        /// <summary>
        /// Starts validation process
after adding first block in
empty queue.
        /// </summary>
        protected virtual void
StartValidation()
        {
            new
Thread(DoValidation).Start();
        }

        /// <summary>
        /// Change status of
transaction in block at
'processed'.
        /// </summary>
        /// <param
name="block">Valid
block.</param>
        protected virtual void
ConfirmBlockTransactions(Blo
ck block)
        {
            foreach (var transaction
in block.Transactions)
            {

transaction.ConfirmTransaction
();
            }
        }

        /// <summary>
        /// Adds new valid block to
the chain.
        /// </summary>
        /// <param
name="block">Valid
block.</param>
        protected virtual void
AddBlockToChain(Block
block)
        {
            blockchain.Push(block);


logger.LogInformation("Block
{@block} added to the
blockchain.", block);
        }

        /// <summary>
        /// Remove valid block
transactions from transactions
pool of transaction service,
        /// and notifying sender
and receiver wallets about
completed transactions.
        /// </summary>
        /// <param
name="block">Valid
block.</param>
        protected virtual void
RemoveBlockTransactionsFro
mTransactionsPool(Block
block)
        {
            foreach (var
blockTransaction in
block.Transactions)
            {
                try
                {

transactionsPool.Remove(block
Transaction.Id);
                }
                catch
(InvalidOperationException ex)
                {

logger.LogError(ex,
"Transaction {@transaction}
exists only in one collection",
blockTransaction);
                }


NotifySenderAndReceiverAbou
tCompletedTransaction(blockTr
ansaction);
            }
        }

        /// <summary>
        /// Saves completed
transactions in sender and
receiver wallets.
        /// </summary>
        /// <param
name="transaction">Transactio
n from valid block.</param>
        protected virtual void
NotifySenderAndReceiverAbou
tCompletedTransaction(Transac
tion transaction)
        {
```

```
        Wallet senderWallet = GetWallet(transaction.Sender);
        Wallet receiverWallet = GetWallet(transaction.Receiver);

senderWallet.SaveInHistory(transaction);

receiverWallet.SaveInHistory(transaction);
    }

    /// <summary>
    /// Validates blocks queue.
    /// </summary>
    protected virtual void DoValidation()
    {
        while (!blockValidationQueue.IsEmpty)
        {
            if (blockValidationQueue.TryDequeue(out Block block))
            {
                if (ValidateBlock(block))
                {

ConfirmBlockTransactions(block);


SaveBlockInDB(block);


AddBlockToChain(block);


RemoveBlockTransactionsFromTransactionsPool(block);


Notify?.Invoke(new ObserverNotificationArgs() { EventName = "DropBlock" });


blockValidationQueue.Clear();

                    break;
                }
            }
        }

        isValidationActive = false;
    }

    /// <summary>
    /// Gets blocks from database and recreates blockchain.
    /// </summary>
    /// <param name="dbTransactions">Transactions to add in blocks.</param>
    protected virtual void RecreateBlockchain(IEnumerable<Transaction> dbTransactions)
    {
        blockchain.Clear();

        var dbBlocks = blocksDAL.GetAllBlocks(dbTransactions);
        foreach (var block in dbBlocks)
        {

AddBlockToChain(block);
        }

        ValidateBlockchain();
    }

    /// <summary>
    /// Gets wallets from database and recreates wallets collection.
    /// </summary>
    /// <param name="dbTransactions">Transactions to add in wallets.</param>
    protected virtual void RecreateWallets(IEnumerable<Transaction> dbTransactions)
    {
        wallets.Clear();

        var dbWallets = walletsDAL.GetAllWallets(dbTransactions);

        foreach (var wallet in dbWallets)
        {

wallets.TryAdd(wallet.PublicKey, wallet);
        }
    }

    /// <summary>
    /// Puts all unprocessed transactions from a database to the transactions poll.
    /// </summary>
    /// <param name="dbTransactions">Transactions to add in transactions poll.</param>
    protected virtual void RecreateTransactionsPool(IEnumerable<Transaction> dbTransactions)
    {

transactionsPool.Clear();

        dbTransactions = dbTransactions
            .Where(t => t.Status == Status.Unprocessed)
            .OrderBy(t => t.DateLastUpdated);

        foreach (var transaction in dbTransactions)
        {

transactionsPool.Add(transaction.Id, transaction);
        }
    }

    /// <summary>
    /// Creates a system wallet for generating miner reward transactions.
    /// </summary>
    protected virtual void CreateSystemWallet()
    {
        if (!wallets.Any(item => item.Value.UserId == systemWalletId))
        {
            var wallet = new Wallet(systemWalletPublicKey, systemWalletId);

wallets.TryAdd(wallet.PublicKey, wallet);
            var coinBase = new Transaction("air",
```

```
systemWalletPublicKey,
TotalCoinsLimit,
guidGenerator.GetNewGUID())
;

coinBase.ConfirmTransaction()
;

wallet.SaveInHistory(coinBase)
;
            }
        }

        /// <summary>
        /// Gets wallet from wallets
collection.
        /// </summary>
        /// <param
name="userId">Identifier of the
wallet`s owner.</param>
        /// <returns>
        /// Wallet, if it has
corresponding user;
        /// otherwise - null.
        /// </returns>
        private Wallet
GetWallet(long userId)
        {
            return
wallets.Values.FirstOrDefault(
wallet => wallet.UserId ==
userId);
        }

        /// <summary>
        /// Saves valid block in the
database.
        /// </summary>
        /// <param
name="block">Valid block
prepared to save.</param>
        private void
SaveBlockInDB(Block block)
        {
            Transaction reward =
block.Transactions.First();
            string transactions =
string.Join(',',
block.Transactions.Skip(1)
                .Select(transaction =>
                {
                    return
transaction.Id;
                }));

blocksDAL.SaveBlock(block.H
ash, block.Nonce,
block.PreviousBlockHash,
reward.Id, reward.Sender,
reward.Receiver,
reward.TransferAmount,
transactions);
        }

        /// <summary>
        /// Validates blockchain.
        /// </summary>
        /// <exception
cref="ApplicationException">
Block in blockchain has failed
validation.</exception>
        private void
ValidateBlockchain()
        {
            string
previousBlockHash =
string.Empty;
            foreach (var block in
blockchain.Reverse())
            {
                string
transactionsHash =
blockCalculationUtils.Calculate
Hash(block.Transactions);
                string
currentBlockHash =
blockCalculationUtils.Calculate
Hash(previousBlockHash,
transactionsHash,
block.Nonce);
                if (currentBlockHash
== block.Hash
                    &&
previousBlockHash ==
block.PreviousBlockHash)
                {

previousBlockHash =
currentBlockHash;
                }
                else
                {

logger.LogCritical("Block
{@block} failed validation.",
block);
                    throw new
ApplicationException($"Block
with hash [{block.Hash}] failed
startup validation");
                }
            }
        }

        /// <summary>
        /// Validates block.
        /// </summary>
        /// <param
name="block">block for
validation.</param>
        /// <returns>true, if block
is valid; false, if block isn't
valid.</returns>
        private bool
ValidateBlock(Block block)
        {
            if
(!blockchain.TryPeek(out
Block previous))
            {
                previous = new
Block(string.Empty, new
List<Transaction>())
                {
                    Hash =
string.Empty,
                };
            }

            string transactionsHash
=
blockCalculationUtils.Calculate
Hash(block.Transactions);
            string currentBlockHash
=
blockCalculationUtils.Calculate
Hash(previous.Hash,
transactionsHash,
block.Nonce);

            if (previous.Hash ==
block.PreviousBlockHash
                && block.Hash ==
currentBlockHash
                &&
blockCalculationUtils.Validate
Hash(block.Hash,
settings.HashZeroNumber)
                &&
ValidateBlockTransactions(blo
ck))
            {
                return true;
            }

            return false;
        }

        /// <summary>
        /// Checks if all
transactions exist in the pool.
```

```
        /// </summary>
        /// <returns>true - if all transactions exist in the pool; false - if one or more transaction not exist in the pool.</returns>
        private bool ValidateBlockTransactions(Block block)
        {
            return ValidateRewardTransaction(block) && IsTransactionsExistInPool(block);
        }

        private bool IsTransactionsExistInPool(Block block)
        {
            if (block.Transactions.Select(t => t.Id).Distinct().Count() != block.Transactions.Count)
            {
                return false;
            }

            foreach (var transactionInBlock in block.Transactions.Skip(1))
            {
                Transaction transactionFromPool;

                try
                {
                    if (!transactionsPool.ContainsKey(transactionInBlock.Id))
                    {
                        return false;
                    }

                    transactionFromPool = transactionsPool[transactionInBlock.Id];
                }
                catch (InvalidOperationException ex)
                {

                    logger.LogError(ex, "Transaction {@transaction} exists only in one collection", transactionInBlock);
                    return false;
                }

                if (!transactionFromPool.Equals(transactionInBlock))
                {
                    return false;
                }
            }

            return true;
        }

        private bool ValidateRewardTransaction(Block block)
        {
            Transaction reward = block.Transactions.FirstOrDefault();
            if (reward == null)
            {
                return false;
            }

            Wallet minerWallet = GetWallet(reward.Receiver);
            if (minerWallet == null)
            {
                return false;
            }

            if (reward.Sender == systemWalletPublicKey && reward.TransferAmount == settings.Reward)
            {
                return true;
            }

            return false;
        }

        /// <summary>
        /// Adds created transaction to the pool of unprocessed transactions.
        /// </summary>
        /// <param name="transaction">Created transaction.</param>
        private void AddTransactionToPool(Transaction transaction)
        {
            if (transaction != null)
            {
                try
                {

                    transactionsPool.Add(transaction.Id, transaction);
                }
                catch (InvalidOperationException ex)
                {

                    logger.LogError(ex, "Transaction {@transaction} exists only in one collection", transaction);
                }
            }
        }
    }
}
```

Cryptocurrency.TransactionService.Interfaces

IMiningServiceHost.cs

```
using System.Threading.Tasks;
using CryptoCurrency.Common.Models;

namespace CryptoCurrency.TransactionService.Interfaces
{
    /// <summary>
    /// An interface for communication with transaction service for miner service.
    /// </summary>
    /// <remarks>
    /// [![Flowchart](../images/TS_IMiningServiceHost.jpg)](../images/TS_IMiningServiceHost.jpg)
    /// </remarks>
    public interface IMiningServiceHost
    {
        /// <summary>
        /// Adds block to the queue for validation.
        /// </summary>
        /// <param name="block">block for validation.</param>
        void AddBlock(Block block);

        /// <summary>
        /// Gets data, needed to block`s calculation.
        /// </summary>
        /// <param name="userId">user`s id, needed to
```

```
reward transaction
creation.</param>
        /// <returns>Object with data,
needed to block`s
calculation.</returns>
        MiningProcessInformation
GetMiningProcessInformation(long
userId);

        /// <summary>
        /// Adds block to the queue for
validation asynchronously.
        /// </summary>
        /// <param
name="block">block for
validation.</param>
        /// <returns>A <see
cref="Task"/> representing the
asynchronous operation.</returns>
        Task AddBlockAsync(Block
block);

        /// <summary>
        /// Gets data, needed to block`s
calculation asynchronously.
        /// </summary>
        /// <param
name="userId">user`s id, needed to
reward transaction
creation.</param>
        /// <returns>Object with data,
needed to block`s
calculation.</returns>

Task<MiningProcessInformation>
GetMiningProcessInformationAsync
(long userId);
    }
}
```

ITransactionCreator.cs

```
using System.Threading.Tasks;
using
CryptoCurrency.Common.Models;

namespace
CryptoCurrency.TransactionService.
Interfaces
{
    /// <summary>
    /// An interface for
communication with transaction
service for customer service.
    /// </summary>
    /// <remarks>
    ///
[![Flowchart](../images/TS_ITransac
tionCreator.jpg)](../images/TS_ITran
sactionCreator.jpg)
    /// </remarks>
    public interface
ITransactionCreator
    {
        /// <summary>
        /// Method for creating
transaction.
        /// </summary>
        /// <param
name="sender">Sender`s wallet
public key.</param>
        /// <param
name="receiver">Receiver`s wallet
public key.</param>
        /// <param
name="amount">How many coins
are transferred from sender to
receiver.</param>
        /// <param
name="response">Message about
transaction creation status.</param>
        /// <returns>true, if transaction
successful created; false, if
transaction creation failed.</returns>
        bool CreateTransaction(string
sender, string receiver, decimal
amount, out string response);

        /// <summary>
        /// Method for creating
transaction asynchronously.
        /// </summary>
        /// <param
name="sender">Sender`s wallet
public key.</param>
        /// <param
name="receiver">Receiver`s wallet
public key.</param>
        /// <param
name="amount">How many coins
are transferred from sender to
receiver.</param>
        /// <returns>JSON object with
response message and operation
status(true or false).</returns>

Task<CreateTransactionResponse>
CreateTransactionAsync(string
sender, string receiver, decimal
amount);
    }
}
```

IWalletContainer.cs

```
using CryptoCurrency.Common.BL;
using
CryptoCurrency.Common.Interfaces
;

namespace
CryptoCurrency.TransactionService.
Interfaces
{
    /// <summary>
    /// An interface for providing
wallets from storage.
    /// </summary>
    /// <remarks>
    ///
[![Flowchart](../images/TS_IWallet
Container.jpg)](../images/TS_IWalle
tContainer.jpg)
    /// </remarks>
    public interface IWalletContainer
    {
        /// <summary>
        /// Gets wallet from the
collection.
        /// </summary>
        /// <param
name="userId">Wallet owner's
identifier.</param>
        /// <returns>Wallet if it exists;
otherwise - null.</returns>
        IObservableService<Wallet>
GetWalletByUserId(long userId);
    }
}
```

IWalletProvider.cs

```
using System.Threading.Tasks;
using
CryptoCurrency.Common.Models;

namespace
CryptoCurrency.TransactionService.
Interfaces
{
    /// <summary>
    /// An interface to perform
operations with a wallets.
    /// </summary>
    /// <remarks>
    ///
[![Flowchart](../images/TS_IWalletP
rovider.jpg)](../images/TS_IWalletPr
ovider.jpg)
    /// </remarks>
    public interface IWalletProvider
    {
        /// <summary>
        /// Provides functionality to get
or create wallet.
        /// </summary>
        /// <param
name="model">Model with
information to perform get or create
wallet operation.</param>
        /// <returns>The task object
representing the asynchronous
operation with <see
cref="WalletInfo"/> object as a
generic type in <see
cref="ResponseResult{T}"/>.</retur
ns>

Task<ResponseResult<WalletInfo>>
GetOrCreateWalletAsync(GetWallet
Model model);

        /// <summary>
        /// Gets wallet balance.
        /// </summary>
        /// <param name="model">Data
for getting wallet balance.</param>
        /// <returns>Object with wallet
balance and information about
operation.</returns>

Task<ResponseResult<decimal>>
GetBalanceAsync(GetWalletBalance
Model model);
```

```
        /// <summary>
        /// Provides functionality to get paginated wallet transactions.
        /// </summary>
        /// <param name="model">Model with information to perform get wallet transactions operation.</param>
        /// <returns>The task object representing the asynchronous operation with <see cref="PaginatedTransactions"/> object as a generic type in <see cref="ResponseResult{T}"/>.</returns>

Task<ResponseResult<PaginatedTransactions>> GetWalletTransactionsAsync(GetWalletTransactionsModel model);

        /// <summary>
        /// Provides functionality to get wallet transaction.
        /// </summary>
        /// <param name="model">Data for getting wallet transaction.</param>
        /// <returns>The task object representing the asynchronous operation with <see cref="GetWalletTransactionResponseModel"/> object as a generic type in <see cref="ResponseResult{T}"/>.</returns>

Task<ResponseResult<GetWalletTransactionResponseModel>> GetWalletTransactionAsync(GetWalletTransactionModel model);
    }
}
```

Cryptocurrency.TransactionService.Models

Settings.cs

```
using CryptoCurrency.TransactionService.Interfaces;

namespace CryptoCurrency.TransactionService.Models
{
    /// <summary>
    /// Class for settings representation.
    /// </summary>
    /// <remarks>
    /// [![Flowchart](../images/TS_Settings.jpg)](../images/TS_Settings.jpg)
    /// </remarks>
    public partial class Settings : ISettings
    {
        private const string DefaultHashZeroNumber = "00000";
        private const int DefaultReward = 50;
        private const int DefaultMaxCountBlockTransactions = 250;

        /// <summary>
        /// Initializes a new instance of the <see cref="Settings"/> class with default values.
        /// </summary>
        public Settings()
        {
            HashZeroNumber = DefaultHashZeroNumber;
            Reward = DefaultReward;
            MaxCountBlockTransactions = DefaultMaxCountBlockTransactions;
        }

        /// <inheritdoc cref="ISettings.HashZeroNumber"/>
        public string HashZeroNumber { get; set; }

        /// <inheritdoc cref="ISettings.Reward"/>
        public decimal Reward { get; set; }

        /// <inheritdoc cref="ISettings.MaxCountBlockTransactions"/>
        public int MaxCountBlockTransactions { get; set; }
    }
}
```

Cryptocurrency.TransactionService.WebAPI

BlockchainController.cs

```
using System.Threading.Tasks;
using CryptoCurrency.Common.Models;
using CryptoCurrency.TransactionService.Interfaces;
using Microsoft.AspNetCore.Mvc;

namespace CryptoCurrency.TransactionService.WebAPI.Controllers
{
    /// <summary>
    /// Provides functionality to HTTP methods by next address [api/blockchain].
    /// </summary>
    public class BlockchainController : CustomApiController
    {
        private readonly IMiningServiceHost miningServiceHost;

        /// <summary>
        /// Initializes a new instance of the <see cref="BlockchainController"/> class.
        /// </summary>
        /// <param name="miningServiceHost">The instance of TransactionService.</param>
        public BlockchainController(IMiningServiceHost miningServiceHost)
        {
            this.miningServiceHost = miningServiceHost;
        }

        /// <summary>
        /// Allows user to send post request to add block.
        /// </summary>
        /// <param name="block">The block.</param>
        /// <returns>A <see cref="Task"/> representing the asynchronous operation.</returns>
        [HttpPost]
        public async Task<ResponseResult> AddBlock(Block block)
        {
            await miningServiceHost.AddBlockAsync(block);
            return new ResponseResult()
            {
                IsSuccessful = true,
                Message = null,
            };
        }
    }
}
```

MiningServiceController.cs

```
using System.Threading.Tasks;
using CryptoCurrency.Common.Models;
using CryptoCurrency.TransactionService.Interfaces;
using Microsoft.AspNetCore.Mvc;

namespace CryptoCurrency.TransactionService.WebAPI.Controllers
{
    /// <summary>
    /// Provides functionality to HTTP methods by next address [api/miningservice].
    /// </summary>
```

```
    public class MiningServiceController : CustomApiController
    {
        private readonly IMiningServiceHost miningServiceHost;

        /// <summary>
        /// Initializes a new instance of the <see cref="MiningServiceController"/> class.
        /// </summary>
        /// <param name="miningServiceHost">The instance of TransactionService.</param>
        public MiningServiceController(IMiningServiceHost miningServiceHost)
        {
            this.miningServiceHost = miningServiceHost;
        }

        /// <summary>
        /// Allows user to send get request to get mining process information.
        /// </summary>
        /// <param name="model">Model with user`s identifier.</param>
        /// <returns>Object with data, needed to block`s calculation.</returns>
        [HttpGet]
        public async Task<ResponseResult<MiningProcessInformation>> GetMiningProcessInformation([FromQuery]GetMiningProcessInformationModel model)
        {
            var result = await miningServiceHost.GetMiningProcessInformationAsync(model.UserId);
            return new ResponseResult<MiningProcessInformation>()
            {
                Model = result,
                IsSuccessful = true,
                Message = null,
            };
        }
    }
}
```

TransactionController.cs

```
using System.Threading.Tasks;
using CryptoCurrency.Common.Models;
using CryptoCurrency.TransactionService.Interfaces;
using CryptoCurrency.TransactionService.WebAPI.Filters;
using Microsoft.AspNetCore.Mvc;

namespace CryptoCurrency.TransactionService.WebAPI.Controllers
{
    /// <summary>
    /// Provides functionality to HTTP methods by next address [api/transaction].
    /// </summary>
    [HMACValidationFilter]
    public class TransactionController : CustomApiController
    {
        private readonly ITransactionCreator transactionCreator;

        /// <summary>
        /// Initializes a new instance of the <see cref="TransactionController"/> class.
        /// </summary>
        /// <param name="transactionCreator">The instance of TransactionService.</param>
        public TransactionController(ITransactionCreator transactionCreator)
        {
            this.transactionCreator = transactionCreator;
        }

        /// <summary>
        /// Allows user to send post request to create transaction.
        /// </summary>
        /// <param name="data">JSON data.</param>
        /// <returns>Result of CreateTransaction method on TransactionService.</returns>
        [HttpPost]
        public async Task<CreateTransactionResponse> CreateTransaction(CreateTransactionModel data)
        {
            var result = await transactionCreator.CreateTransactionAsync(data.Sender, data.Receiver, data.Amount);
            return result;
        }
    }
}
```

WalletController.cs

```
using System.Threading.Tasks;
using CryptoCurrency.Common.Models;
using CryptoCurrency.TransactionService.Interfaces;
using CryptoCurrency.TransactionService.WebAPI.Filters;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;

namespace CryptoCurrency.TransactionService.WebAPI.Controllers
{
    /// <summary>
    /// Provides functionality to HTTP methods by next address [api/wallet].
    /// </summary>
    [HMACValidationFilter]
    public class WalletController : CustomApiController
    {
        private readonly IWalletProvider walletProvider;
        private readonly ILogger<WalletController> logger;

        /// <summary>
        /// Initializes a new instance of the <see cref="WalletController"/> class.
        /// </summary>
        /// <param name="walletProvider">The instance of TransactionService.</param>
        /// <param name="logger">Logger.</param>
        public WalletController(IWalletProvider walletProvider, ILogger<WalletController> logger)
        {
            this.walletProvider = walletProvider;
            this.logger = logger;
        }

        /// <summary>
        /// Allows user to send post request to get a wallet.
        /// </summary>
        /// <param name="getWalletModel">Model with information to perform get wallet operation.</param>
        /// <returns>Result of GetWalletAsync method on TransactionService.</returns>
        [HttpPost]
        public async Task<ResponseResult<WalletInfo>> GetWallet(GetWalletModel getWalletModel)
        {
```

```
logger.LogInformation("Received a request to get a wallet for user with ID={userId}.", getWalletModel.UserId);
            var getWalletResponse = await walletProvider.GetOrCreateWalletAsync(getWalletModel);
            return getWalletResponse;
        }

        /// <summary>
        /// Allows user to send post request to get a wallet balance.
        /// </summary>
        /// <param name="getWalletBalanceModel">Model with information to perform get wallet balance operation.</param>
        /// <returns>Result of GetBalanceAsync method on TransactionService.</returns>
        [HttpGet]
        public async Task<ResponseResult<decimal>> GetBalance([FromQuery] GetWalletBalanceModel getWalletBalanceModel)
        {

logger.LogInformation("Received a request to get a wallet balance for user with ID={userId}.", getWalletBalanceModel.UserId);
            var getWalletResponse = await walletProvider.GetBalanceAsync(getWalletBalanceModel);
            return getWalletResponse;
        }

        /// <summary>
        /// Allows user to send get request to get paginated transactions for a wallet.
        /// </summary>
        /// <param name="model">Model with information to perform get transactions for a wallet operation.</param>
        /// <returns>Result of GetWalletTransactionsAsync method on TransactionService.</returns>
        [HttpGet]
        public async Task<ResponseResult<PaginatedTransactions>> GetTransactions([FromQuery] GetWalletTransactionsModel model)
        {

logger.LogInformation("Received a request to get transactions for a user with ID={userId}.", model.UserId);
            var getWalletTransactionsResponse = await walletProvider.GetWalletTransactionsAsync(model);
            return getWalletTransactionsResponse;
        }

        /// <summary>
        /// Allows user to send get request to get transaction for a wallet.
        /// </summary>
        /// <param name="model">Model with information to perform get transaction for a wallet operation.</param>
        /// <returns>Result of GetWalletTransactionAsync method on TransactionService.</returns>
        [HttpGet]
        public async Task<ResponseResult<GetWalletTransactionResponseModel>> GetTransaction([FromQuery] GetWalletTransactionModel model)
        {

logger.LogInformation("Received a request to get transaction={transactionId} for a user with ID={userId}.", model.TransactionId, model.UserId);
            var getWalletTransactionResponse = await walletProvider.GetWalletTransactionAsync(model);
            return getWalletTransactionResponse;
        }
    }
}
```

HMACValidationFilterAttribute.cs

```
using System;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using CryptoCurrency.Common.Interfaces;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Filters;
using Microsoft.Extensions.DependencyInjection;

namespace CryptoCurrency.TransactionService.WebAPI.Filters
{
    /// <summary>
    /// A Filer for authorizing the request before performing API action.
    /// </summary>

    [AttributeUsage(AttributeTargets.Method | AttributeTargets.Class)]
    public class HMACValidationFilterAttribute : Attribute, IAsyncAuthorizationFilter
    {
        /// <inheritdoc cref="IAsyncAuthorizationFilter.OnAuthorizationAsync(AuthorizationFilterContext)"/>
        public async Task OnAuthorizationAsync(AuthorizationFilterContext context)
        {
            var hashingUtils = context.HttpContext.RequestServices.GetService<IHMACHashingUtils>();
            var query = context.HttpContext.Request.QueryString;
            var content = string.Empty;
            var request = context.HttpContext.Request;
            request.EnableBuffering();
            request.Body.Position = 0;
            using (var stream = new StreamReader(request.Body, Encoding.UTF8, true, 1024, leaveOpen: true))
            {
                content = await stream.ReadToEndAsync();
            }

            request.Body.Position = 0;
            var data = content + query;
            var hmac = hashingUtils.GetHMAC(data);
            if (!request.Headers.TryGetValue("HMAC", out var hmacHeader) || hmacHeader.FirstOrDefault() != hmac)
            {

context.HttpContext.Response.StatusCode = 403;
                context.Result = new EmptyResult();
            }
        }
    }
}
```

Program.cs

```
using System;
using System.Diagnostics.CodeAnalysis;
```

```
using System.IO;
using CryptoCurrency.Common.BL;
using
Microsoft.AspNetCore.Hosting;
using
Microsoft.Extensions.Configuration;
using
Microsoft.Extensions.DependencyIn
jection;
using Microsoft.Extensions.Hosting;
using Serilog;

namespace
CryptoCurrency.TransactionService.
WebAPI
{
   /// <summary>
   /// Encapsulates the application's
main entry point.
   /// </summary>
   [ExcludeFromCodeCoverage]
   public static class Program
   {
      /// <summary>
      /// The application's entry point.
      /// </summary>
      /// <param name="args">An
array of console line
arguments.</param>
      public static void Main(string[]
args)
      {
         var config = new
ConfigurationBuilder()

.SetBasePath(Directory.GetCurrentD
irectory())

.AddJsonFile("appsettings.json")
            .Build();

         Log.Logger = new
LoggerConfiguration()

.ReadFrom.Configuration(config)
            .CreateLogger();

         try
         {

Log.Information("Application
Starting");
            CreateHostBuilder(args)
               .Build()
               .StartServices()
               .Run();
         }
         catch (Exception ex)
         {
            Log.Fatal(ex, "Application
failed startup.");
         }
         finally
         {
            Log.CloseAndFlush();
         }
      }

      /// <summary>
      /// Creates Host Builder.
      /// </summary>
      /// <param name="args">An
array of console line
arguments.</param>
      /// <returns>The initialized
IHostBuilder.</returns>
      public static IHostBuilder
CreateHostBuilder(string[] args) =>

Host.CreateDefaultBuilder(args)
            .UseSerilog()

.ConfigureWebHostDefaults(webBui
lder =>
            {

webBuilder.UseStartup<Startup>();
            });

      /// <summary>
      /// Starts TransactionService
before running web application.
      /// </summary>
      /// <param name="host">IHost
to modify.</param>
      /// <returns>IHost with active
TransactionService.</returns>
      public static IHost
StartServices(this IHost host)
      {
         var ts =
host.Services.GetService<Transactio
nService.BL.TransactionService>();
         var observersManager =
host.Services.GetService<Observers
Manager>();
         ts.Start();
         observersManager.Start();
         return host;
      }
   }
}
```

Startap.cs

```
using
System.Diagnostics.CodeAnalysis;
using CryptoCurrency.Common.BL;
using
CryptoCurrency.Common.Interfaces
;
using
CryptoCurrency.Common.Models;
using CryptoCurrency.DAL;
using
CryptoCurrency.DAL.Interfaces;
using
CryptoCurrency.TransactionService.
BL;
using
CryptoCurrency.TransactionService.
Interfaces;
using
CryptoCurrency.TransactionService.
WebAPI.Filters;
using
CryptoCurrency.TransactionService.
WebAPI.Hubs;
using
Microsoft.AspNetCore.Builder;
using
Microsoft.AspNetCore.Hosting;
using
Microsoft.Extensions.DependencyIn
jection;
using Microsoft.Extensions.Hosting;
using
Swashbuckle.AspNetCore.Swagger
UI;

namespace
CryptoCurrency.TransactionService.
WebAPI
{
   /// <summary>
   /// Provides the startup methods
for the web application.
   /// </summary>
   [ExcludeFromCodeCoverage]
   public class Startup
   {
      /// <summary>
      /// Adds services to the web
application container.
      /// </summary>
      /// <param
name="services">Collection of
service descriptors.</param>
      public static void
ConfigureServices(IServiceCollectio
n services)
      {
         services.AddSignalR();

services.AddControllers().AddNewt
onsoftJson();

services.AddSwaggerGen(options
=>
options.OperationFilter<HMACHea
derParameterOperationFilter>());

services.AddHttpClient<IHttpClient
Wrapper, HttpClientWrapper>();

services.AddSingleton<IConfigurati
onManager,
ConfigurationManager>()

.AddSingleton<ISqlDataAccess,
SqlDataAccess>()

.AddSingleton<IHMACHashingUtil
s, HashingUtils>()

.AddSingleton<ISettingsDAL,
SettingsDAL>()

.AddSingleton<ISettings>(sp =>
sp.GetRequiredService<ISettingsDA
```

```
L>().GetAllSettings())

.AddTransient<IMonitorLocker,
Locker>()

.AddSingleton<IBlockCalculationUt
ils, BlockCalculationUtils>()

.AddSingleton<IGUIDGenerator,
GUIDGenerator>()

.AddTransient<IReaderWriterLocker
, ReaderWriterLock>()

.AddSingleton<ITransactionsDAL,
TransactionsDAL>()

.AddSingleton<IWalletsDAL,
WalletsDAL>()

.AddSingleton<IBlocksDAL,
BlocksDAL>()

.AddSingleton<BL.TransactionServi
ce>()

.AddSingleton<ITransactionCreator
>(sp =>
sp.GetRequiredService<BL.Transact
ionService>())

.AddSingleton<IMiningServiceHost
>(sp =>
sp.GetRequiredService<BL.Transact
ionService>())

.AddSingleton<IWalletProvider>(sp
=>
sp.GetRequiredService<BL.Transact
ionService>())

.AddSingleton<IWalletContainer>(s
p =>
sp.GetRequiredService<BL.Transact
ionService>())

.AddSingleton<IObservableService<
BL.TransactionService>>(sp =>
sp.GetRequiredService<Transaction
Service.BL.TransactionService>())

.AddTransient<IHttpHelper,
HttpHelper>()

.AddTransient<ObserverSettings>()

.AddTransient<IObserverWebProxy,
ObserverWebProxy>()

.AddTransient<ObserversManagerSe
ttings>()

.AddSingleton<ObserversManager>(
)

.AddSingleton<IObserversManager>
(sp =>
sp.GetRequiredService<ObserversM
anager>())

.AddSingleton<IGroupConnections<
WalletObserverHub>,
GroupConnections<WalletObserver
Hub>>()

.AddSingleton<IObservableEventHa
ndler<TransactionServiceObserverH
ub>,
TransactionServiceSignalrEventHan
dler>()

.AddSingleton<IObservableEventHa
ndler<WalletObserverHub>,
WalletSignalrEventHandler>();
        }

        /// <summary>
        /// Request processing pipeline
with a sequence of middleware
components.
        /// </summary>
        /// <param name="app">Class
to configure an application's request
pipeline.</param>
        /// <param name="env">Web
hosting environment an application
is running in.</param>
        public static void
Configure(IApplicationBuilder app,
IWebHostEnvironment env)
        {
            app.UseSwagger();
            app.UseSwaggerUI(options
=>
            {

options.SwaggerEndpoint("/swagger
/v1/swagger.json", "TS OpenAPI
doc");
                options.RoutePrefix =
string.Empty;

options.DocExpansion(DocExpansio
n.None);
            });

            if (env.IsDevelopment())
            {

app.UseDeveloperExceptionPage();
            }

            app.UseRouting();

            app.UseEndpoints(endpoints
=>
            {

endpoints.MapControllers();

endpoints.MapHub<TransactionServ
iceObserverHub>("/transactionservic
eobserverhub");

endpoints.MapHub<WalletObserver
Hub>("/walletobserverhub");
            });
        }
    }
}
```

**Cryptocurrency.CustomerSer
vice.BL**

CustomerService.cs

```
using System;
using
System.Collections.Generic;
using
System.Runtime.CompilerServi
ces;
using System.Security.Claims;
using System.Threading.Tasks;
using
CryptoCurrency.Common.Inter
faces;
using
CryptoCurrency.Common.Mod
els;
using
CryptoCurrency.CustomerServi
ce.Interfaces;
using
CryptoCurrency.DAL.Interface
s;
using
CryptoCurrency.TransactionSer
vice.Interfaces;
using
Microsoft.Extensions.Logging;

[assembly:
InternalsVisibleTo("UnitTestin
g")]
[assembly:
InternalsVisibleTo("UnitTestin
g_Framework")]

namespace
CryptoCurrency.CustomerServi
ce.BL
{
    /// <summary>
    /// Class for customer service
representation.
    /// </summary>
    /// <remarks>
    ///
[![Flowchart](../images/CS.jpg)
](../images/CS.jpg)
    /// </remarks>
    public class CustomerService
```

```
: ICustomerService,
IAccountManager,
IWalletManager,
IUsersManager
    {
        private const string
WalletServiceObservableName
= "WalletService";

        /// <summary>
        /// Interface for interaction
with the Transaction Service.
        /// </summary>
        private readonly
ITransactionCreator
transactionCreator;
        private readonly
IWalletProvider walletProvider;
        private readonly
IUsersDAL usersDAL;
        private readonly
ILogger<CustomerService>
logger;
        private readonly
IHashingUtils hashingUtils;
        private readonly
IGUIDGenerator
guidGenerator;
        private readonly
IObservableServiceProvider
observableServiceProvider;
        private readonly
ISessionHandler
sessionHandler;
        private readonly
IWalletChangedEventHandler
walletChangedHandler;

        /// <summary>
        /// Initializes a new
instance of the <see
cref="CustomerService"/>
class.
        /// </summary>
        /// <param
name="transactionCreator">Ins
tance of Transaction Service for
transaction creation.</param>
        /// <param
name="walletProvider">Instanc
e of Transaction Service for
work with wallet.</param>
        /// <param
name="usersDAL">Data
access layer for users.</param>
        /// <param
name="logger">Interface for
logging.</param>
        /// <param
name="hashingUtils">Utils for
creating hashed
strings.</param>
        /// <param
name="guidGenerator">Unique
ID generator.</param>
        /// <param
name="observableServiceProvi
der">Instance to get observable
instances to observe an
events.</param>
        /// <param
name="sessionHandler">A
session handler to work with
user sessions.</param>
        /// <param
name="walletChangedEventHa
ndler">Event handler.</param>
        public
CustomerService(IWalletProvid
er walletProvider,
ITransactionCreator
transactionCreator, IUsersDAL
usersDAL,
ILogger<CustomerService>
logger, IHashingUtils
hashingUtils, IGUIDGenerator
guidGenerator,
IObservableServiceProvider
observableServiceProvider,
ISessionHandler
sessionHandler,
IWalletChangedEventHandler
walletChangedEventHandler)
        {
            this.walletProvider =
walletProvider;
            this.transactionCreator
= transactionCreator;
            this.usersDAL =
usersDAL;
            this.logger = logger;
            this.hashingUtils =
hashingUtils;
            this.guidGenerator =
guidGenerator;

this.observableServiceProvider
= observableServiceProvider;
            this.sessionHandler =
sessionHandler;

this.walletChangedHandler =
walletChangedEventHandler;
        }

        /// <inheritdoc
cref="IUsersManager.AddRole
ToUser(long, byte)"/>
        public bool
AddRoleToUser(long userId,
byte roleId)
        {
            try
            {
                return
usersDAL.AddRoleToUser(use
rId, roleId);
            }
            catch
(ArgumentException ex)
            {

logger.LogWarning(ex.Messag
e);
                return false;
            }
            catch (Exception ex)
            {
                logger.LogError(ex,
"Something went wrong in
attempt to add role to user");
                return false;
            }
        }

        /// <summary>
        /// Method for creating
transaction.
        /// </summary>
        /// <param
name="sender">Sender`s
wallet public key.</param>
        /// <param
name="receiver">Receiver`s
wallet public key.</param>
        /// <param
name="amount">How many
coins are transferred from
sender to receiver.</param>
        /// <param
name="response">Message
about transaction creation
status.</param>
        /// <returns>true, if
transaction successfully
created; false, if transaction
creation failed.</returns>
        public bool
CreateTransaction(string
sender, string receiver, decimal
amount, out string response)
```

```
        {
            return
transactionCreator.CreateTrans
action(sender, receiver, amount,
out response);
        }

        /// <inheritdoc
cref="ICustomerService.Create
TransactionAsync(string,
string, decimal)"/>
        public async
Task<CreateTransactionRespon
se>
CreateTransactionAsync(string
sender, string receiver, decimal
amount)
        {
            return await
transactionCreator.CreateTrans
actionAsync(sender, receiver,
amount);
        }

        /// <inheritdoc
cref="IUsersManager.GetAllUs
ers()"/>
        public
IEnumerable<User>
GetAllUsers()
        {
            try
            {
                return
usersDAL.GetAllUsers();
            }
            catch (Exception ex)
            {
                logger.LogError(ex,
ex.Message);
                return
Array.Empty<User>();
            }
        }

        /// <inheritdoc
cref="IWalletManager.GetBala
nceAsync(GetWalletBalanceM
odel)"/>
        public async
Task<ResponseResult<decimal
>>
GetBalanceAsync(GetWalletBa
lanceModel model)
        {
            return await
walletProvider.GetBalanceAsy
nc(model);
        }

        /// <inheritdoc
cref="IWalletManager.GetOrCr
eateWalletAsync(GetWalletMo
del)"/>
        public async
Task<ResponseResult<WalletI
nfo>>
GetOrCreateWalletAsync(Get
WalletModel model)
        {
            var responseResult =
await
walletProvider.GetOrCreateWa
lletAsync(model);
            return responseResult;
        }

        /// <inheritdoc
cref="IWalletManager.GetWall
etTransactionAsync(GetWallet
TransactionModel)"/>
        public async
Task<ResponseResult<GetWall
etTransactionResponseModel>
>
GetWalletTransactionAsync(Ge
tWalletTransactionModel
model)
        {
            var responseResult =
await
walletProvider.GetWalletTrans
actionAsync(model);
            return responseResult;
        }

        /// <inheritdoc
cref="IWalletManager.GetWall
etTransactionsAsync(GetWallet
TransactionsModel)"/>
        public async
Task<ResponseResult<Paginat
edTransactions>>
GetWalletTransactionsAsync(G
etWalletTransactionsModel
model)
        {
            var responseResult =
await
walletProvider.GetWalletTrans
actionsAsync(model);
            return responseResult;
        }

        /// <inheritdoc
cref="IAccountManager.Login(
string, string)"/>
        public LoginResponse
Login(string email, string
password)
        {
            var user =
usersDAL.GetUserByEmail(em
ail);
            if (user != null)
            {
                if
(hashingUtils.ValidatePassword
(password, user.Salt,
user.PasswordHash))
                {
                    if
(user.Roles.HasFlag(Roles.Cust
omer))
                    {
                        if
(sessionHandler.GetSessionsCo
unt(user.UserId.ToString()) ==
0)
                        {
                            var
parameters = new
Dictionary<string, string> { {
"userid", $"{user.UserId}" } };
                            var
walletService =
observableServiceProvider.Get(
WalletServiceObservableName,
parameters);

walletService.Notify +=
OnNotify;
                        }


sessionHandler.AddSession(use
r.UserId.ToString());
                        var claims =
new[]
                        {
                        new
UserClaim(ClaimTypes.NameI
dentifier,
user.UserId.ToString()),
                        new
UserClaim(ClaimTypes.Email,
user.Username),
                        new
UserClaim(ClaimTypes.Role,
user.Roles.ToString()),
                        };
```

```
                return new
LoginResponse(claims,
string.Empty);
            }

            return new
LoginResponse(null,
ResponseMessages.AccountBlo
cked);
        }
    }

    return new
LoginResponse(null,
ResponseMessages.WrongCred
entials);
  }

  /// <inheritdoc
cref="IAccountManager.Logou
t(string)"/>
  public void Logout(string
userId)
  {
    var userSessionsCount
=
sessionHandler.GetSessionsCo
unt(userId);
    if (userSessionsCount >
0)
    {
      if (userSessionsCount
== 1)
      {
        var walletService =
observableServiceProvider.Get(
WalletServiceObservableName,
new Dictionary<string, string>
{ { "userid", $"{userId}" } });

walletService.Notify -=
OnNotify;
      }


sessionHandler.RemoveSession
(userId);
    }
  }

  /// <inheritdoc
cref="IAccountManager.Regist
erUser(string, string)"/>
  public
RegisterUserResponse
RegisterUser(string username,
string password)
  {
    RegisterUserResponse
registerUserResponse;

    try
    {

logger.LogInformation("Receiv
ed request to register user.");
      var salt =
guidGenerator.GetNewGUID();
      var passwordHash =
hashingUtils.HashPassword(pas
sword, salt);
      var userId =
usersDAL.AddUser(username,
passwordHash, salt);
      if (userId > 0)
      {

logger.LogInformation("New
user with ID={@userId}
successfully added to DB",
userId);

registerUserResponse = new
RegisterUserResponse(true,
ResponseMessages.Successfull
yRegistered);
      }
      else
      {

logger.LogError("AddUser
didn't make any changes in DB,
and didn't throw an exception");

registerUserResponse = new
RegisterUserResponse(false,
ResponseMessages.InternalErro
r);
      }
    }
    catch (Exception ex)
    {
      if
(ex.Message.Contains("UqUser
_Username"))
      {

registerUserResponse = new
RegisterUserResponse(false,
ResponseMessages.UserAlread
yExists);
      }
      else if
(ex.Message.Contains("CkUser
_Username_IsEmail") ||
ex.Message.Contains("column
'Username'. Truncated"))
      {

registerUserResponse = new
RegisterUserResponse(false,
ResponseMessages.InvalidEma
il);
      }
      else if
(ex.Message.Contains("CkCred
entials_Salt_Length") ||
ex.Message.Contains("column
'Salt'. Truncated"))
      {

logger.LogError("AddUser
violated salt length constraint in
DB.");

registerUserResponse = new
RegisterUserResponse(false,
ResponseMessages.InternalErro
r);
      }
      else if
(ex.Message.Contains("CkCred
entials_PasswordHash_Length"
) ||
ex.Message.Contains("column
'PasswordHash'. Truncated"))
      {

logger.LogError("AddUser
violated password length
constraint in DB.");

registerUserResponse = new
RegisterUserResponse(false,
ResponseMessages.InternalErro
r);
      }
      else
      {

logger.LogError(ex,
"Something went wrong in
attempt to register user");

registerUserResponse = new
RegisterUserResponse(false,
ResponseMessages.InternalErro
r);
      }
    }
```

```
            return
registerUserResponse;
        }

        /// <inheritdoc
cref="IUsersManager.Remove
RoleFromUser(long, byte)"/>
        public bool
RemoveRoleFromUser(long
userId, byte roleId)
        {
            try
            {
                return
usersDAL.RemoveRoleFromUs
er(userId, roleId);
            }
            catch
(ArgumentException ex)
            {

logger.LogWarning(ex.Messag
e);
                return false;
            }
            catch (Exception ex)
            {
                logger.LogError(ex,
"Something went wrong in
attempt to remove role from
user");
                return false;
            }
        }

        /// <summary>
        /// Event handler.
        /// </summary>
        /// <param
name="args">Additional
parameters with event
name.</param>
        internal void
OnNotify(ObserverNotification
Args args)
        {
            switch
(args.EventName)
            {
                case
"WalletChanged":
                    {

logger.LogInformation("Receiv
ed event with arguments:
{@args}", args);
                        if
(args.EventArgs != null
                            &&
args.EventArgs.TryGetValue("
userid", out string userId))
                        {

walletChangedHandler.Handle(
userId);
                        }
                        else
                        {

logger.LogWarning("{@event
Name}: Invalid event
arguments - {@args}",
args.EventName,
args.EventArgs);
                        }

                        break;
                    }

                default:

logger.LogInformation("Attem
pt to raise non-existent event:
{@args}", args);
                    break;
            }
        }
    }
```

**Cryptocurrency.CustomerSer
vice.Interfaces**

ICustomerService.cs

```
using System.Threading.Tasks;
using
CryptoCurrency.Common.Models;

namespace
CryptoCurrency.CustomerService.In
terfaces
{
    /// <summary>
    /// Interface for communication
with CustomerService.
    /// </summary>
    /// <remarks>
    ///
[![Flowchart](../images/CS_ICustom
erService.jpg)](../images/CS_ICusto
merService.jpg)
    /// </remarks>
    public interface ICustomerService
    {
        /// <summary>
        /// Method for creating
transaction asynchronously.
        /// </summary>
        /// <param
name="sender">Sender`s wallet
public key.</param>
        /// <param
name="receiver">Receiver`s wallet
public key.</param>
        /// <param
name="amount">How many coins
are transferred from sender to
receiver.</param>
        /// <returns>JSON object with
response message and operation
status(true or false).</returns>

        Task<CreateTransactionResponse>
CreateTransactionAsync(string
sender, string receiver, decimal
amount);
    }
}
```

IWalletManager.cs

```
using System.Threading.Tasks;
using
CryptoCurrency.Common.Models;

namespace
CryptoCurrency.CustomerService.In
terfaces
{
    /// <summary>
    /// Interface for communicating
with CustomerService.
    /// </summary>
    /// <remarks>
    ///
[![Flowchart](../images/CS_IWallet
Manager.jpg)](../images/CS_IWallet
Manager.jpg)
    /// </remarks>
    public interface IWalletManager
    {
        /// <summary>
        /// Gets wallet if it exist or
creates if wallet doesn't exist.
        /// </summary>
        /// <param name="model">Data
for getting wallet.</param>
        /// <returns>Object with short
wallet info and information about
operation.</returns>

        Task<ResponseResult<WalletInfo>>
GetOrCreateWalletAsync(GetWallet
Model model);

        /// <summary>
        /// Gets wallet balance.
        /// </summary>
        /// <param name="model">Data
for getting wallet balance.</param>
        /// <returns>Object with wallet
balance and information about
operation.</returns>

        Task<ResponseResult<decimal>>
GetBalanceAsync(GetWalletBalance
```

```
Model model);

        /// <summary>
        /// Gets paginated wallet
transactions.
        /// </summary>
        /// <param name="model">Data
for getting wallet
transactions.</param>
        /// <returns>Object with
wallet's owner user id and
sorting/filtering
information.</returns>

Task<ResponseResult<PaginatedTra
nsactions>>
GetWalletTransactionsAsync(GetW
alletTransactionsModel model);

        /// <summary>
        /// Gets wallet transaction.
        /// </summary>
        /// <param name="model">Data
for getting wallet
transaction.</param>
        /// <returns>Object with
wallet's owner user id and
transaction id.</returns>

Task<ResponseResult<GetWalletTr
ansactionResponseModel>>
GetWalletTransactionAsync(GetWal
letTransactionModel model);
    }
}
```

**Cryptocurrency.CustomerSer
vice.WebApplication**

AccountManagerController.cs

```
using
System.Diagnostics.CodeAnalysis;
using System.Web;
using System.Web.Mvc;
using Microsoft.Owin.Security;

namespace
CryptoCurrency.CustomerService.W
ebApplication.Controllers
{
    /// <summary>
    /// Extends default Controller base
class with Authentication manager
which provides object from OWIN
context.
    /// </summary>
    /// <remarks>
    ///
[![Flowchart](../images/CS.jpg)](../i
mages/CS.jpg)
    /// </remarks>
    public class
AccountManagerController :
Controller
    {
        private IAuthenticationManager
authenticationManager;

        /// <summary>
        /// Gets or sets authentication
manager.
        /// </summary>
        [ExcludeFromCodeCoverage]
        internal
IAuthenticationManager
AuthenticationManager
        {
            get
            {
                return
authenticationManager ??
Request.GetOwinContext().Authenti
cation;
            }

            set
            {
                authenticationManager =
value;
            }
        }
    }
}
```

AccountController.cs

```
using
System.Runtime.CompilerServices;
using System.Security.Claims;
using System.Web.Mvc;
using
CryptoCurrency.Common.Interfaces
;
using
CryptoCurrency.Common.Models;
using
CryptoCurrency.CustomerService.W
ebApplication.Models;
using Microsoft.AspNet.Identity;
using Microsoft.Extensions.Logging;

[assembly:
InternalsVisibleTo("UnitTesting_Fra
mework")]

namespace
CryptoCurrency.CustomerService.W
ebApplication.Controllers
{
    /// <summary>
    /// Controller to handle the
processes of account actions.
    /// </summary>
    /// <remarks>
    ///
[![Flowchart](../images/CS.jpg)](../i
mages/CS.jpg)
    /// </remarks>
    public class AccountController :
AccountManagerController
    {
        private readonly
IAccountManager accountManager;
        private readonly
ILogger<AccountController>
logger;

        /// <summary>
        /// Initializes a new instance of
the <see cref="AccountController"/>
class.
        /// </summary>
        /// <param
name="accountManager">The
instance of account
manager.</param>
        /// <param
name="logger">Logger.</param>
        public
AccountController(IAccountManage
r accountManager,
ILogger<AccountController>
logger)
        {
            this.accountManager =
accountManager;
            this.logger = logger;
        }

        /// <summary>
        /// Returns page for registration.
        /// </summary>
        /// <returns>View.</returns>
        public ActionResult Register()
        {
            return View();
        }

        /// <summary>
        /// Allows user to send post
request to register in the system.
        /// </summary>
        /// <param
name="model">Model filled with
user data for registration.</param>
        ///
<returns>RedirectToRouteResult
with home/index path if user
successfully registered; otherwise -
ViewResult.</returns>
        [HttpPost]
        public ActionResult
Register(RegisterModel model)
        {
            if (ModelState.IsValid)
            {
                var registerUserResponse
=
accountManager.RegisterUser(mode
l.Email, model.Password);
                if
(registerUserResponse.IsRegistered)
                {
                    var loginModel = new
LoginModel()
                    {
                        Email = model.Email,
                        Password =
model.Password,
                    };
```

```
                return
Login(loginModel);
            }


ModelState.AddModelError("Failed
RegisterOperation",
registerUserResponse.Message);
        }

        return View(model);
    }

    /// <summary>
    /// Returns page to log in.
    /// </summary>
    /// <returns>View.</returns>
    public ActionResult Login()
    {
        return View();
    }

    /// <summary>
    /// Allows user to send post
request to log in the system.
    /// </summary>
    /// <param
name="model">Model filled with
user data for log in.</param>
    ///
<returns>RedirectToRouteResult
with home/index path if user
successfully logged in; otherwise -
ViewResult.</returns>
    [HttpPost]
    public virtual ActionResult
Login(LoginModel model)
    {
        if (ModelState.IsValid)
        {
            var loginResponse =
accountManager.Login(model.Email
, model.Password);
            if
(loginResponse.UserClaims != null)
            {
                var claims =
loginResponse.UserClaims;
                var identity = new
ClaimsIdentity(claims,
authenticationType:
"ApplicationCookie");


AuthenticationManager.SignIn(ident
ity);

logger.LogInformation("User with
ID={@userId} logged in.",
AuthenticationManager.User.Identit
y.GetUserId());

                return
RedirectToAction("Index",
"Home");
            }


ModelState.AddModelError("Accou
ntInfo", loginResponse.Message);
        }

        return View(model);
    }

    /// <summary>
    /// Allows miner to send post
request to log in the system.
    /// </summary>
    /// <param
name="userinfo">Model filled with
miner data for log in.</param>
    /// <returns>Response with
collection of claims which contains
information about user(meaning id
and roles).</returns>
    [HttpPost]
    public JsonResult
MinerLogin(UserInfoModel
userinfo)
    {

ResponseResult<LoginResponse>
result;
        if (userinfo != null)
        {
            var loginResponse =
accountManager.Login(userinfo.Em
ail, userinfo.Password);
            if
(loginResponse.UserClaims != null)
            {
                result = new
ResponseResult<LoginResponse>()
                {
                    Model =
loginResponse,
                    IsSuccessful = true,
                    Message =
string.Empty,
                };
                return Json(result);
            }

            result = new
ResponseResult<LoginResponse>()
            {
                Model = default,
                IsSuccessful = false,
                Message =
loginResponse.Message,
            };
            return Json(result);
        }

        result = new
ResponseResult<LoginResponse>()
        {
            Model = default,
            IsSuccessful = false,
            Message =
ResponseMessages.UnreadableUserI
nfo,
        };
        return Json(result);
    }

    /// <summary>
    /// Allows user to send post
request to log out from the system.
    /// </summary>
    ///
<returns>RedirectToRouteResult
with Account/Login path.</returns>
    [HttpPost]
    [Authorize]
    public ActionResult Logout()
    {
        var userId =
AuthenticationManager.User.Identit
y.GetUserId();

accountManager.Logout(userId);

AuthenticationManager.SignOut(aut
henticationTypes:
"ApplicationCookie");
        logger.LogInformation("User
with ID={@userId} logged off.",
userId);
        return
RedirectToAction("Login");
    }
  }
}
```

HomeController.cs

```
using System.Web.Mvc;

namespace
CryptoCurrency.CustomerService.W
ebApplication.Controllers
{
  /// <summary>
  /// Represents controller for
default pages.
  /// </summary>
  /// <remarks>
  ///
[![Flowchart](../images/CS.jpg)](../i
mages/CS.jpg)
  /// </remarks>
  public class HomeController :
Controller
  {
    /// <summary>
    /// Returns main page of
application.
    /// </summary>
    /// <returns>View.</returns>
    public ActionResult Index()
    {
      if (Request.IsAuthenticated)
      {
        return
RedirectToAction("Index",
"Wallet");
      }
```

```
        return View();
      }
   }
}
```

TransactionControler.cs

```
using System;
using System.Threading.Tasks;
using System.Web.Mvc;
using
CryptoCurrency.Common.Models;
using
CryptoCurrency.CustomerService.In
terfaces;
using Microsoft.AspNet.Identity;
using Microsoft.Extensions.Logging;

namespace
CryptoCurrency.CustomerService.W
ebApplication.Controllers
{
   /// <summary>
   /// Controller to handle the process
of creating a transaction.
   /// </summary>
   /// <remarks>
   ///
[![Flowchart](../images/CS.jpg)](../i
mages/CS.jpg)
   /// </remarks>
   [Authorize]
   public class
TransactionsController :
AccountManagerController
   {
      private readonly
ICustomerService customerService;
      private readonly
IWalletManager walletManager;
      private readonly
ILogger<TransactionsController>
logger;

      /// <summary>
      /// Initializes a new instance of
the <see
cref="TransactionsController"/>
class.
      /// </summary>
      /// <param
name="customerService">Interface
for communication with
CustomerService
transactions.</param>
      /// <param
name="walletManager">Interface
for getting wallet.</param>
      /// <param
name="logger">Logger.</param>
      public
TransactionsController(IWalletMana
ger walletManager,
ICustomerService customerService,
ILogger<TransactionsController>
logger)
      {
         this.walletManager =
walletManager;
         this.customerService =
customerService;
         this.logger = logger;
      }

      /// <summary>
      /// Returns page for creating
transaction.
      /// </summary>
      /// <returns>View.</returns>
      [HttpGet]
      public async
Task<ActionResult>
CreateTransaction()
      {
         var userId =
Convert.ToInt64(AuthenticationMan
ager.User.Identity.GetUserId());
         var responseResult = await
walletManager.GetOrCreateWalletA
sync(new GetWalletModel() {
UserId = userId });
         return View(responseResult);
      }

      /// <summary>
      /// Allows user to send post
request to create transaction.
      /// </summary>
      /// <param
name="transaction">Transaction
filled with data provided by user
through the UI.</param>
      /// <returns>Result of creating a
transaction in JSON
format.</returns>
      [HttpPost]
      public async Task<JsonResult>
CreateTransaction(CreateTransactio
nModel transaction)
      {

logger.LogInformation("Started
request for creating transaction
{@transaction}", transaction);
         var response = await
customerService.CreateTransaction
Async(transaction.Sender,
transaction.Receiver,
transaction.Amount);
         if (response == null)
         {

logger.LogWarning("Request
received no response");
            response = new
CreateTransactionResponse(false,
ResponseMessages.NoResponseFro
mServer);
         }

         return Json(response);
      }
   }
}
```

WalletController.cs

```
using System;
using System.Threading.Tasks;
using System.Web.Mvc;
using
CryptoCurrency.Common.Models;
using
CryptoCurrency.CustomerService.In
terfaces;
using Microsoft.AspNet.Identity;

namespace
CryptoCurrency.CustomerService.W
ebApplication.Controllers
{
   /// <summary>
   /// Controller to work with user's
wallet.
   /// </summary>
   /// <remarks>
   ///
[![Flowchart](../images/CS.jpg)](../i
mages/CS.jpg)
   /// </remarks>
   [Authorize]
   public class WalletController :
AccountManagerController
   {
      private readonly
IWalletManager walletManager;

      /// <summary>
      /// Initializes a new instance of
the <see cref="WalletController"/>
class.
      /// </summary>
      /// <param
name="walletManager">Interface
for communication with
CustomerService.</param>
      public
WalletController(IWalletManager
walletManager)
      {
         this.walletManager =
walletManager;
      }

      /// <summary>
      /// Returns wallet's home page.
      /// </summary>
      /// <returns>View.</returns>
      [HttpGet]
      public async
Task<ActionResult> Index()
      {
         var userId =
Convert.ToInt64(AuthenticationMan
ager.User.Identity.GetUserId());
         var responseResult = await
walletManager.GetOrCreateWalletA
sync(new GetWalletModel() {
UserId = userId });
         return View(responseResult);
      }
```

```
        /// <summary>
        /// Gets wallet's balance.
        /// </summary>
        /// <returns>Balance.</returns>
        [HttpGet]
        public async Task<JsonResult>
GetBalance()
        {
            var userId =
Convert.ToInt64(AuthenticationMan
ager.User.Identity.GetUserId());
            var responseResult = await
walletManager.GetBalanceAsync(ne
w GetWalletBalanceModel() {
UserId = userId });
            return Json(responseResult,
JsonRequestBehavior.AllowGet);
        }

        /// <summary>
        /// Allows user to send request
to get wallet's filtered transactions.
        /// </summary>
        /// <param
name="model">Model filled with
filtering information.</param>
        /// <returns>Result of get
transactions in JSON
format.</returns>
        [HttpGet]
        public async Task<JsonResult>
GetTransactions(GetWalletTransacti
onsModel model)
        {
            var userId =
Convert.ToInt64(AuthenticationMan
ager.User.Identity.GetUserId());
            model.UserId = userId;
            var responseResult = await
walletManager.GetWalletTransactio
nsAsync(model);
            return Json(responseResult,
JsonRequestBehavior.AllowGet);
        }

        /// <summary>
        /// Allows user to send request
to get page with specific transaction.
        /// </summary>
        /// <param
name="model">Model filled with
transaction ID.</param>
        /// <returns>View.</returns>
        [HttpGet]
        public async
Task<ActionResult>
GetTransaction(GetWalletTransactio
nModel model)
        {
            var userId =
Convert.ToInt64(AuthenticationMan
ager.User.Identity.GetUserId());
            model.UserId = userId;
            var responseResult = await
walletManager.GetWalletTransactio
nAsync(model);
            return View(responseResult);
        }
    }
}
```

WalletsHub.cs

```
using System;
using System.Linq;
using System.Security.Claims;
using System.Threading.Tasks;
using Microsoft.AspNet.SignalR;
using
Microsoft.AspNet.SignalR.Hubs;
using Microsoft.Extensions.Logging;

namespace
CryptoCurrency.CustomerService.W
ebApplication.Hubs
{
    /// <inheritdoc cref="Hub"/>
    /// <remarks>
    ///
[![Flowchart](../images/CS.jpg)](../i
mages/CS.jpg)
    /// </remarks>
    [HubName("wallets")]
    public class WalletsHub : Hub
    {
        private readonly
ILogger<WalletsHub> logger;

        /// <summary>
        /// Initializes a new instance of
the <see cref="WalletsHub"/> class.
        /// </summary>
        /// <param
name="logger">Logger.</param>
        public
WalletsHub(ILogger<WalletsHub>
logger)
        {
            this.logger = logger;
        }

        /// <inheritdoc
cref="HubBase.OnConnected"/>
        public override Task
OnConnected()
        {
            try
            {
                var connectionId =
Context.ConnectionId;
                var user = Context.User as
ClaimsPrincipal;
                var userId =
user.Claims.First(c => c.Type ==
ClaimTypes.NameIdentifier).Value;
                Groups.Add(connectionId,
userId);


logger.LogInformation("Client
{@id} connected",
Context.ConnectionId);
                return
base.OnConnected();
            }
            catch (Exception ex)
            {
                logger.LogError(ex,
"Exception occured during
connecting to hub");
                return
Task.CompletedTask;
            }
        }

        /// <inheritdoc
cref="HubBase.OnDisconnected(bo
ol)"/>
        public override Task
OnDisconnected(bool stopCalled)
        {
            try
            {
                var connectionId =
Context.ConnectionId;
                var user = Context.User as
ClaimsPrincipal;
                var userId =
user.Claims.First(c => c.Type ==
ClaimTypes.NameIdentifier).Value;

Groups.Remove(connectionId,
userId);

logger.LogInformation("Client
{@id} disconnected",
Context.ConnectionId);
                return
base.OnConnected();
            }
            catch (Exception ex)
            {
                logger.LogError(ex,
"Exception occured during
disconnecting from hub");
                return
Task.CompletedTask;
            }
        }
    }
}
```

WalletProvider.cs

```
using System.Threading.Tasks;
using
CryptoCurrency.Common.Interfaces
;
using
CryptoCurrency.Common.Models;
using
CryptoCurrency.TransactionService.
Interfaces;
using Microsoft.Extensions.Logging;

namespace
CryptoCurrency.CustomerService.W
ebApplication.Models
{
    /// <summary>
    /// Provides functionality to
```

```
perform operations with wallets.
    /// </summary>
    /// <remarks>
    ///
[![Flowchart](../images/CS.jpg)](../images/CS.jpg)
    /// </remarks>
    public class WalletProvider :
IWalletProvider
    {
        private const string
GetWalletEndpointName =
"GetWalletEndpoint";
        private const string
GetWalletBalanceEndpointName =
"GetWalletBalanceEndpoint";
        private const string
GetWalletTransactionEndpointName
= "GetWalletTransactionEndpoint";
        private const string
GetWalletTransactionsEndpointNam
e =
"GetWalletTransactionsEndpoint";
        private readonly IHttpHelper
httpClient;
        private readonly
IEndpointConfiguration
configurationManager;
        private readonly
ILogger<WalletProvider> logger;

        /// <summary>
        /// Initializes a new instance of
the <see cref="WalletProvider"/>
class.
        /// </summary>
        /// <param
name="httpClient">HTTP client for
sending requests.</param>
        /// <param
name="configurationManager">Inte
rface for interaction with
configuration file.</param>
        /// <param
name="logger">Logger.</param>
        public
WalletProvider(IHttpHelper
httpClient, IEndpointConfiguration
configurationManager,
ILogger<WalletProvider> logger)
        {
            this.httpClient = httpClient;
            this.configurationManager =
configurationManager;
            this.logger = logger;
        }

        /// <inheritdoc
cref="IWalletProvider.GetBalanceA
sync(GetWalletBalanceModel)"/>
        public async
Task<ResponseResult<decimal>>
GetBalanceAsync(GetWalletBalance
Model model)
        {
            logger.LogInformation("Sending a
request to get a wallet balance for
user with ID={userId}.",
model.UserId);
            var endpoint =
configurationManager.GetEndpoint(
GetWalletBalanceEndpointName);
            var
getWalletBalanceResponse = await
httpClient.GetAsync<GetWalletBala
nceModel, decimal>(model,
endpoint);
            return
getWalletBalanceResponse;
        }

        /// <inheritdoc
cref="IWalletProvider.GetOrCreate
WalletAsync(GetWalletModel)"/>
        public async
Task<ResponseResult<WalletInfo>>
GetOrCreateWalletAsync(GetWallet
Model model)
        {

logger.LogInformation("Sending a
request to get a wallet for user with
ID={userId}.", model.UserId);
            var endpoint =
configurationManager.GetEndpoint(
GetWalletEndpointName);
            var getWalletResponse =
await
httpClient.PostAsync<GetWalletMo
del, WalletInfo>(model, endpoint);
            return getWalletResponse;
        }

        /// <inheritdoc
cref="IWalletProvider.GetWalletTra
nsactionAsync(GetWalletTransactio
nModel)"/>
        public async
Task<ResponseResult<GetWalletTr
ansactionResponseModel>>
GetWalletTransactionAsync(GetWal
letTransactionModel model)
        {

logger.LogInformation("Sending a
request to get a
transaction={transactionId} for a
user with ID={userId}.",
model.TransactionId, model.UserId);
            var endpoint =
configurationManager.GetEndpoint(
GetWalletTransactionEndpointName
);
            var
getWalletTransactionResponse =
await
httpClient.GetAsync<GetWalletTran
sactionModel,
GetWalletTransactionResponseMod
el>(model, endpoint);
            return
getWalletTransactionResponse;
        }

        /// <inheritdoc
cref="IWalletProvider.GetWalletTra
nsactionsAsync(GetWalletTransacti
onsModel)"/>
        public async
Task<ResponseResult<PaginatedTra
nsactions>>
GetWalletTransactionsAsync(GetW
alletTransactionsModel model)
        {

logger.LogInformation("Sending a
request to get a transactions for a
user with ID={userId}.",
model.UserId);
            var endpoint =
configurationManager.GetEndpoint(
GetWalletTransactionsEndpointNam
e);
            var
getWalletTransactionsResponse =
await
httpClient.GetAsync<GetWalletTran
sactionsModel,
PaginatedTransactions>(model,
endpoint);
            return
getWalletTransactionsResponse;
        }
    }
}
```

TransactionCreator.cs

```
using System;
using
System.Diagnostics.CodeAnalysis;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;
using
CryptoCurrency.Common.Interfaces
;
using
CryptoCurrency.Common.Models;
using
CryptoCurrency.TransactionService.
Interfaces;
using Microsoft.Extensions.Logging;
using Newtonsoft.Json;

namespace
CryptoCurrency.CustomerService.W
ebApplication.Models
{
    /// <summary>
    /// Provides functionality for
transaction creating.
    /// </summary>
    /// <remarks>
    ///
[![Flowchart](../images/CS.jpg)](../images/CS.jpg)
    /// </remarks>
    public class TransactionCreator :
```

```
ITransactionCreator
    {
        private const string
CreateTransactionEndpointName =
"CreateTransactionEndpoint";
        private readonly
IHttpClientWrapper httpClient;
        private readonly
IEndpointConfiguration
configurationManager;
        private readonly
ILogger<TransactionCreator>
logger;

        /// <summary>
        /// Initializes a new instance of
the <see
cref="TransactionCreator"/> class.
        /// </summary>
        /// <param
name="httpClient">Http client for
sending requests.</param>
        /// <param
name="configurationManager">Inte
rface for interaction with
configuration file.</param>
        /// <param
name="logger">Logger.</param>
        public
TransactionCreator(IHttpClientWrap
per httpClient,
IEndpointConfiguration
configurationManager,
ILogger<TransactionCreator>
logger)
        {
            this.httpClient = httpClient;
            this.configurationManager =
configurationManager;
            this.logger = logger;
        }

        /// <inheritdoc
cref="ITransactionCreator.CreateTra
nsaction(string, string, decimal, out
string)"/>
        [ExcludeFromCodeCoverage]
        public bool
CreateTransaction(string sender,
string receiver, decimal amount, out
string response)
        {
            response = $"{sender}-
>{receiver} - {amount}";
            return true;
        }

        /// <inheritdoc
cref="ITransactionCreator.CreateTra
nsactionAsync(string, string,
decimal)"/>
        public async
Task<CreateTransactionResponse>
CreateTransactionAsync(string
sender, string receiver, decimal
amount)
        {
            var transaction = new
CreateTransactionModel
            {
                Sender = sender,
                Receiver = receiver,
                Amount = amount,
            };

            var jsonData =
JsonConvert.SerializeObject(transact
ion);
            var content = new
StringContent(jsonData,
Encoding.UTF8, "application/json");
            var result = await
SendPostAsync(CreateTransactionE
ndpointName, content);
            return result;
        }

        private async
Task<CreateTransactionResponse>
SendPostAsync(string
endpointName, StringContent
content)
        {
            CreateTransactionResponse
result;
            try
            {
                var endpoint =
configurationManager.GetEndpoint(
endpointName);
                var requestResponse =
await
httpClient.PostAsync(endpoint,
content);
                var requestResult = await
requestResponse.Content.ReadAsStr
ingAsync();
                result =
JsonConvert.DeserializeObject<Crea
teTransactionResponse>(requestRes
ult);
            }
            catch (Exception ex)
            {
                logger.LogError(ex,
"Request cannot be sent");
                result = new
CreateTransactionResponse(false,
ResponseMessages.NoResponseFro
mServer);
            }

            return result;
        }
    }
}
```

create-transaction.js

```
function onAmountInput() {
    var amount =
document.getElementById("amount"
).value.replace(/[^0-9.]/, '')
document.getElementById("amount"
).value = amount;
}

function createTransaction() {

document.getElementById("createTr
ansaction").blur();

document.getElementById("respons
e").innerHTML = "";
    let sender =
document.getElementById('wallet-
pk').innerHTML;
    let receiver =
document.getElementById('receiver'
).value;
    let amount =
document.getElementById('amount')
.value;
    let senderValidationResult =
validateSender(sender);
    let receiverValidationResult =
validateReceiver(receiver, sender);
    let amountValidationResult =
validateAmount(amount);
    if (!senderValidationResult) {

document.getElementById("respons
e").style.color = 'red';

document.getElementById("respons
e").innerHTML = "Request cannot
be completed. Please, contact our
support team to provide additional
details.";
    }
    else {
        if (senderValidationResult &&
receiverValidationResult &&
amountValidationResult) {
            $.ajax({
                url:
"/Transactions/CreateTransaction",
                contentType:
'application/json; charset=utf-8',
                data: JSON.stringify({
                    Sender:
sender.toLowerCase(),
                    Receiver:
receiver.toLowerCase(),
                    Amount: amount
                }),
                method: 'post',
                dataType: 'json',
                success: function (data) {
                    if (data.OperationStatus)
{

document.getElementById("respons
e").style.color = '#037f51';

document.getElementById("respons
e").innerHTML = "Succeeded: " +
data.ResponseMessage;
                }
```

```
                else {

document.getElementById("respons
e").style.color = 'red';

document.getElementById("respons
e").innerHTML = "Failed: " +
data.ResponseMessage;
                }
            }
        })
    }
  }
}

function validateSender(sender) {
  let regex = /[a-fA-F0-9]$/;

  if (regex.test(sender) &&
sender.length == 32) {
    return true;
  }
  return false;
}

function validateReceiver(receiver,
sender) {
  if (receiver == sender) {
    printMessage("receiver",
"Receiver cannot have same public
key with sender.", true);
    return false;
  }
  return validatePublicKey(receiver,
"receiver")
}

function
validatePublicKey(publicKey,
owner) {
  let regex = /[a-fA-F0-9]$/;

  if (regex.test(publicKey)) {
    printMessage(owner, "", false);
    let pkLength =
publicKey.length;
    if (pkLength != 32) {
      printMessage(owner, "Public
key must be 32 characters long (now
it's " + pkLength + ")", true);
      return false;
    }
    return true;
  }
  else {
    printMessage(owner, "Invalid
input (acceptable characters a-f,
numbers from 0 to 9)", true);
    return false;
  }
}

function validateAmount(amount) {
  if (isNaN(amount)) {
    printMessage("amount",
"Invalid input", true);
    return false;
  }

  if (amount <= 0) {
    printMessage("amount",
"Amount cannot be less or equal 0",
true);
    return false;
  }

  if (amount.includes('.')) {
    let fractionalPart =
amount.substring(amount.indexOf("."
) + 1);
    let integerPart =
amount.substring(0,
amount.indexOf("."));

    let fractionalLimit = 8;
    let integerLimit = 8;

    if (integerPart.length >
integerLimit) {
      printMessage("amount",
"Amout should be less than 1" +
'0'.repeat(integerLimit), true);
      return false;
    }
    if (fractionalPart.length >
fractionalLimit) {
      printMessage("amount",
"Length of amount fractional part
should be less or equal " +
fractionalLimit, true);
      return false;
    }
  }
  else {
    let limit = 8;
    if (amount.length > limit) {
      printMessage("amount",
"Amout should be less than 1" +
'0'.repeat(limit), true);
      return false;
    }
  }

  printMessage("amount", "", false);
  return true;
}

function printMessage(owner,
message, isError) {
  if (isError) {

document.getElementById(owner +
"Message").innerHTML =
"&#10060 " + message;

document.getElementById(owner).st
yle.borderColor = 'red';
  }
  else {

document.getElementById(owner +
"Message").innerHTML =
"&#10004";

document.getElementById(owner).st
yle.borderColor = "";
  }
}
```

create-transacction-signalr.js

```
import "../Scripts/jquery.signalR-
2.4.3.min.js";
import "../signalr/hubs";

$(function () {
  var connection =
$.hubConnection();
  var wallets =
connection.createHubProxy('wallets'
);

  wallets.on('refreshData', function
() {
    getBalance();
  });

  connection.start();
})
```

wallet-balance.js

```
$(document).ready(getBalance());

function getBalance() {
  $.ajax({
    url: "/Wallet/GetBalance",
    contentType: 'application/json;
charset=utf-8',
    method: 'get',
    dataType: 'json',
    success: function (data) {
      if (data.IsSuccessful) {

document.getElementById("balance
").innerHTML = data.Model;
        }
        else {

document.getElementById("balance
").innerHTML = "";
        }
    }
  })
}
```

wallet-index-signalr.js

```
import "../Scripts/jquery.signalR-
2.4.3.min.js";
import "../signalr/hubs";

$(function () {
  var connection =
$.hubConnection();
  var wallets =
connection.createHubProxy('wallets'
);

  wallets.on('refreshData',
function () {

getPaginatedTransactionsAjax()
```

```
;
        getBalance();
    });

    connection.start();
})
```

wallet-index.js

```
'use strict'

$(document).ready(function ()
{
    limitDateInputs();

getPaginatedTransactionsAjax()
;
})

function limitDateInputs() {
    let today = new Date();
    let dd = today.getDate();
    let mm = today.getMonth() +
1;
    let yyyy =
today.getFullYear();

    if (dd < 10) {
        dd = '0' + dd;
    }

    if (mm < 10) {
        mm = '0' + mm;
    }

    today = yyyy + '-' + mm + '-'
+ dd;
    $('[type="date"]').attr('max',
today);
}

$('[type="date"]').on('blur',
function() {
    let filterDate = $(this);

    if (filterDate.val() <
filterDate.attr('min')) {

filterDate.val(filterDate.attr('mi
n'))
    }
    else if (filterDate.val() >
filterDate.attr('max')) {

filterDate.val(filterDate.attr('ma
x'))
    }
})

$('#filter-date-from').on('blur',
function( ){
    let filterDateFrom = $(this);
    let filterDateTo = $('#filter-
date-to');

    if (filterDateTo.val() &&
filterDateFrom.val() >
filterDateTo.val()) {

filterDateTo.val(filterDateFrom
.val());
    }

    filterDateTo.attr('min',
filterDateFrom.val());
  })

$('#filters-apply-btn').on('click',
function() {
    let queryParams = new
URLSearchParams(window.loc
ation.search);
    let filterDateFrom =
document.getElementById('filte
r-date-from');
    let filterDateTo =
document.getElementById('filte
r-date-to');
    let filterWallet =
document.getElementById('filte
r-wallet');
    let filterStatus =
document.getElementById('filte
r-status');
    let filterType =
document.getElementById('filte
r-type');

    if (filterDateFrom.value &&
filterDateFrom.value !==
filterDateFrom.min) {

queryParams.set(filterDateFro
m.name, filterDateFrom.value);
    } else {

queryParams.delete(filterDateF
rom.name);
    }

    if (filterDateTo.value &&
filterDateTo.value !==
filterDateTo.max) {

queryParams.set(filterDateTo.n
ame, filterDateTo.value);
    } else {

queryParams.delete(filterDateT
o.name);
    }

    if (filterWallet.value) {

queryParams.set(filterWallet.na
me, filterWallet.value);
    } else {

queryParams.delete(filterWallet
.name);
    }

    if (filterStatus.value == 'any')
{

queryParams.delete(filterStatus.
name);
    } else {

queryParams.set(filterStatus.na
me, filterStatus.value);
    }

    if (filterType.value == 'any')
{

queryParams.delete(filterType.
name);
    } else {

queryParams.set(filterType.nam
e, filterType.value);
    }

    history.replaceState(null,
null, '?' +
queryParams.toString());

getPaginatedTransactionsAjax()
})

$('.pagination-
button').on('click',
function(event) {
    let pageValue =
event.currentTarget.value;
    let queryParams = new
URLSearchParams(window.loc
ation.search);
    if (pageValue == '1') {

queryParams.delete('page');
    } else {
```

```
        queryParams.set('page',
pageValue);
    }
    history.replaceState(null,
null, '?' +
queryParams.toString());

getPaginatedTransactionsAjax()
;
})

$('.transactions-
container').on('click',
'.expandable', function() {
    let neighborClass =
$(this).next();
    if
(neighborClass.hasClass('d-
none')) {

neighborClass.removeClass('d-
none');
    } else {

neighborClass.addClass('d-
none');
    }
})

function parseDate(date) {
    return new
Date(parseInt(date.match(/\d+/)
[0]));
}

function
getDateMounthYear(date) {
    let year =
date.getUTCFullYear();
    let month =
date.getUTCMonth() + 1;
    month = month < 10 ? '0' +
month : month;
    let day = date.getUTCDate();
    day = day < 10 ? '0' + day :
day;
    return
`${day}/${month}/${year}`;
}

function
getHourMinuteSecond(date) {
    let hour =
date.getUTCHours();
    hour = hour < 10 ? '0' + hour
: hour;
    let minute =
date.getUTCMinutes();
    minute = minute < 10 ? '0' +
minute : minute;
    let second =
date.getUTCSeconds();
    second = second < 10 ? '0' +
second : second;
    return
`${hour}:${minute}:${second}
`;
}

function
getTransactionTypeHtml(transa
ctionSender, walletPK) {
    if (transactionSender ==
walletPK) {
        return `<div class="d-flex
align-items-center justify-
content-center justify-content-
md-start">
                    <svg
xmlns="http://www.w3.org/200
0/svg" width="20" height="20"
fill="currentColor" class="bi
bi-arrow-up-circle"
viewBox="0 0 16 16">
                        <path fill-
rule="evenodd" d="M1 8a7 7 0
1 0 14 0A7 7 0 0 0 1 8zm15
0A8 8 0 1 1 0 8a8 8 0 0 1 16
0zm-7.5 3.5a.5.5 0 0 1-1
0V5.707L5.354 7.854a.5.5 0 1
1-.708-.708l3-3a.5.5 0 0 1 .708
0l3 3a.5.5 0 0 1-.708.708L8.5
5.707V11.5z" />
                    </svg>
                    <div class="ms-2
d-none d-md-
block">Outcoming</div>
                </div>`;
    }

    return `<div class="d-flex
align-items-center justify-
content-center justify-content-
md-start">
                <svg
xmlns="http://www.w3.org/200
0/svg" width="20" height="20"
fill="currentColor" class="bi
bi-arrow-down-circle"
viewBox="0 0 16 16">
                    <path fill-
rule="evenodd" d="M1 8a7 7 0
1 0 14 0A7 7 0 0 0 1 8zm15
0A8 8 0 1 1 0 8a8 8 0 0 1 16
0zM8.5 4.5a.5.5 0 0 0-1
0v5.793L5.354 8.146a.5.5 0 1
0-.708.708l3 3a.5.5 0 0 0 .708
0l3-3a.5.5 0 0 0-.708-.708L8.5
10.293V4.5z" />
                </svg>
                <div class="ms-2 d-
none d-md-
block">Incoming</div>
            </div>`;
}

function
getTransactionAmountHtml(tra
nsactionSender, walletPK,
transactionAmount) {
    if (transactionSender ==
walletPK) {
        return `<div class="d-flex
align-items-center justify-
content-center justify-content-
md-start text-danger">
                    -
${transactionAmount.toFixed(8
)}
                </div>`;
    }

    return `<div class="d-flex
align-items-center justify-
content-center justify-content-
md-start text-success">

${transactionAmount.toFixed(8
)}
            </div>`;
}

function
getTransactionDirectionHtml(tr
ansactionSender,
transactionReceiver, walletPK)
{
    if (transactionSender ==
walletPK) {
        return `<p class="fw-bold
my-2 text-nowrap">To:</p>
                 
                <p
class="transaction-wallet-value
my-2 text-truncate">

${transactionReceiver}
                </p>`;
    }

    return `<p class="fw-bold
```

```
my-2 text-nowrap">From:</p>
         
        <p class="transaction-wallet-value my-2 text-truncate">
            ${transactionSender}
        </p>`;
}

function getTransactionStatusHtml(transactionStatus) {
    let processedStatus = 1;
    let unprocessedStatus = 0;
    if (transactionStatus == processedStatus) {
        return `<div class="d-flex align-items-center justify-content-center justify-content-md-start">
                <svg xmlns="http://www.w3.org/2000/svg" width="20" height="20" fill="currentColor" class="bi bi-check-lg" viewBox="0 0 16 16">
                    <path d="M12.736 3.97a.733.733 0 0 1 1.047 0c.286.289.29.756.01 1.05L7.88 12.01a.733.733 0 0 1-1.065.02L3.217 8.384a.757.757 0 0 1 0-1.06.733.733 0 0 1 1.047 0l3.052 3.093 5.4-6.425a.247.247 0 0 1 .02-.022Z" />
                </svg>
                <div class="ms-2 d-none d-md-block">Processed</div>
            </div>`;
    } else if (transactionStatus == unprocessedStatus) {
        return `<div class="d-flex align-items-center justify-content-center justify-content-md-start">
                <svg xmlns="http://www.w3.org/2000/svg" width="20" height="20" fill="currentColor" class="bi bi-arrow-clockwise" viewBox="0 0 16 16">
                    <path fill-rule="evenodd" d="M8 3a5 5 0 1 0 4.546 2.914.5.5 0 0 1 .908-.417A6 6 0 1 1 8 2v1z" />
                    <path d="M8 4.466V.534a.25.25 0 0 1 .41-.192l2.36 1.966c.12.1.12.284 0 .384L8.41 4.658A.25.25 0 0 1 8 4.466z" />
                </svg>
                <div class="ms-2 d-none d-md-block">Unprocessed</div>
            </div>`;
    }
}

function getPaginatedTransactionsAjax() {
    let queryParams = new URLSearchParams(window.location.search).toString();
    let walletPK = document.getElementById('wallet-pk').innerHTML;

    $.ajax({
        url: '/Wallet/GetTransactions',
        contentType: 'application/json; charset=utf-8',
        data: queryParams,
        dataType: 'json',
        success: function (data) {
            let tableBody = '';

            $.each(data.Model.Transactions, function (index, transaction) {
                let transactionDate = parseDate(transaction.DateLastUpdated);
                let transactionDMY = getDateMounthYear(transactionDate);
                let transactionHMS = getHourMinuteSecond(transactionDate)

                let transactionTypeHtml = getTransactionTypeHtml(transaction.Sender, walletPK);
                let transactionAmountHtml = getTransactionAmountHtml(transaction.Sender, walletPK, transaction.TransferAmount);
                let transactionDirectionHtml = getTransactionDirectionHtml(transaction.Sender, transaction.Receiver, walletPK);
                let transactionStatusHtml = getTransactionStatusHtml(transaction.Status);

                tableBody += `<tr class="pointer tr-hover expandable">

<td>${transactionTypeHtml}</td>
                    <td>
                        <div class="d-flex align-items-center justify-content-center justify-content-md-start">

<div>${transactionDMY}</div>
                            <div class="ms-1 d-none d-md-block">${transactionHMS}</div>
                        </div>
                    </td>

<td>${transactionAmountHtml}</td>

<td>${transactionStatusHtml}</td>
                </tr>
                <tr class="d-none">
                    <td colspan="4">
                        <div class="transaction-id d-flex">
                            <p class="fw-bold my-2 text-nowrap">Transaction Id:</p>
                             
                            <p class="transaction-id-value my-2 text-truncate">${transaction.Id}</p>
                        </div>
                        <div class="transaction-wallet d-flex">${transactionDirectionHtml}</div>
                        <div class="transaction-date d-flex">
                            <p class="fw-
```

```
bold my-2 text-
nowrap">Date:</p>
                         
                        <p
class="transaction-date-value
my-2 text-
truncate">${transactionDMY}
${transactionHMS}</p>
                    </div>
                </td>
            </tr>`;
        })

        $('table
tbody').html(tableBody);

        if
(data.Model.PageInfo.HasPrevi
ous) {
            $('#first-
page').removeClass('disabled');
            $('#prev-
page').removeClass('disabled');
        } else {
            $('#first-
page').addClass('disabled');
            $('#prev-
page').addClass('disabled');
        }

        if
(data.Model.PageInfo.HasNext)
{
            $('#next-
page').removeClass('disabled');
            $('#last-
page').removeClass('disabled');
        } else {
            $('#next-
page').addClass('disabled');
            $('#last-
page').addClass('disabled');
        }

        $('#prev-
page').val(data.Model.PageInfo.
PageNumber - 1);
        $('#current-
page').html(data.Model.PageInf
o.PageNumber);
        $('#next-
page').val(data.Model.PageInfo.
PageNumber + 1);
        $('#last-
page').val(data.Model.PageInfo.
TotalPages);
    }
    });
}
```

Login.cshtml

```
@model
CryptoCurrency.CustomerService.W
ebApplication.Models.LoginModel

@{
    ViewBag.Title = "Log in";
}
<h2>@ViewBag.Title</h2>
<hr />
<p>@Html.ValidationMessage("Ac
countInfo", new { @class = "text-
danger" })</p>
@using (Html.BeginForm("Login",
"Account", FormMethod.Post, new {
@class = "form-horizontal", role =
"form" }))
{
    <div class="input-group mb-3">
        @Html.LabelFor(m =>
m.Email, new { @class = "col-12"
})
        <div class="col-12">
            @Html.TextBoxFor(m =>
m.Email, new { @class = "col-md-1
form-control", @placeholder =
"example@domain.com" })

@Html.ValidationMessageFor(m =>
m.Email, "", new { @class = "col-
md-8 text-danger" })
        </div>
    </div>
    <div class="input-group mb-3">
        @Html.LabelFor(m =>
m.Password, new { @class = "col-
12" })
        <div class="col-12">
            @Html.PasswordFor(m =>
m.Password, new { @class = "col-
md-1 form-control" })

@Html.ValidationMessageFor(m =>
m.Password, "", new { @class =
"col-md-8 text-danger" })
        </div>
    </div>
    <div class="input-group mb-3">
        <div class="col-12">
            <input type="submit"
class="btn btn-primary" value="Log
in" />
        </div>
    </div>

}

@section scripts {

@Scripts.Render("~/bundles/jqueryv
al")
}
```

Register.cshtml

```
@model
CryptoCurrency.CustomerService.W
ebApplication.Models.RegisterMode
l
@{
    ViewBag.Title = "Register";
}

<h2>@ViewBag.Title</h2>
@using
(Html.BeginForm("Register",
"Account", FormMethod.Post, new {
@class = "form-horizontal", role =
"form" }))
{
    <hr />
    <div class="input-group mb-3">
        @Html.LabelFor(m =>
m.Email, new { @class = "col-12"
})
        <div class="col-12">
            @Html.TextBoxFor(m =>
m.Email, new { @class = "col-md-1
form-control", @placeholder =
"example@domain.com" })

@Html.ValidationMessageFor(m =>
m.Email, "", new { @class = "col-
md-8 text-danger" })
        </div>
    </div>
    <div class="input-group mb-3">
        @Html.LabelFor(m =>
m.Password, new { @class = "col-
12" })
        <div class="col-12">
            @Html.PasswordFor(m =>
m.Password, new { @class = "col-
md-1 form-control" })

@Html.ValidationMessageFor(m =>
m.Password, "", new { @class =
"col-md-8 text-danger" })
        </div>
    </div>
    <div class="input-group mb-3">
        @Html.LabelFor(m =>
m.ConfirmPassword, new { @class
= "col-12" })
        <div class="col-12">
            @Html.PasswordFor(m =>
m.ConfirmPassword, new { @class
= "col-md-1 form-control" })

@Html.ValidationMessageFor(m =>
m.ConfirmPassword, "", new {
@class = "col-md-8 text-danger" })
        </div>
    </div>
    <div class="input-group mb-3">
        <div class="col-12">
            <input type="submit"
class="btn btn-primary"
value="Register" />
        </div>
    </div>
```

```
@Html.ValidationMessage("FailedRegisterOperation", new { @class = "text-danger" })
}

@section scripts {

@Scripts.Render("~/bundles/jqueryval")
}
```

CreateTransaction.cshtml

```
@model CryptoCurrency.Common.Models.ResponseResult<CryptoCurrency.Common.Models.WalletInfo>
@{
    ViewBag.Title = "CreateTransaction";
}

@section css {
    <link rel="stylesheet" href="~/Content/CopyToClipboard.css">
    <link rel="stylesheet" href="~/Content/CreateTransaction.css">
}

@section hint{
    <div id="copyHint" class="fixed-top p-3 bg-secondary text-light text-truncate d-none"></div>
}

<div class="d-flex flex-column">
    <h2 class="mb-3">Create Transaction</h2>
    <div class="d-flex flex-column flex-sm-row justify-content-sm-between">
        <div class="d-flex align-items-sm-end">
            <h6 class="mb-0 d-none d-sm-block">Public Key: </h6>
            <h6 class="mb-0 d-block d-sm-none">PK: </h6>
             
            @if (Model.IsSuccessful)
            {
                <p id="wallet-pk" class="h6 mb-0 text-truncate text-dark pointer" title="Copy to clipboard">@Model.Model.PublicKey</p>
            }
            else
            {
                <p id="wallet-pk" class="h6 mb-0 text-truncate text-dark pointer" title="Copy to clipboard">@Model.Message</p>
            }
        </div>
        <div class="d-flex mb-0 mb-sm-0">
            <h6 class="mb-0">Balance: </h6>
             
            <h6 id="balance" class="mb-0"></h6>
        </div>
    </div>
</div>
<hr />
<div class="input-group mb-3">
    <label class="col-12">Receiver</label>
    <div class="col-12">
        <input type="text" class="form-control inline" autocomplete="off" id="receiver" />
        <div class="message inline" id="receiverMessage"></div>
    </div>
</div>
<div class="input-group mb-3">
    <label class="col-12">Amount</label>
    <div class="col-12">
        <input type="text" class="form-control inline" autocomplete="off" id="amount" oninput="onAmountInput()" />
        <div class="message inline" id="amountMessage"></div>
    </div>
</div>
<div>
    <button class="btn btn-success btn-create-transaction" id="createTransaction" onclick="createTransaction()">Create</button>
</div>
<br />
<div id="response"></div>

@section scripts {
    <script type="text/javascript" src="~/Scripts/copy-to-clipboard.js"></script>
    <script type="text/javascript" src="~/Scripts/create-transaction.js"></script>
    <script type="text/javascript" src="~/Scripts/wallet-balance.js"></script>
    <script type="module" src="~/Scripts/create-transaction-signalR.js"></script>
}
```

WalletIndex.cshtml

```
@model CryptoCurrency.Common.Models.ResponseResult<CryptoCurrency.Common.Models.WalletInfo>

@{
    ViewBag.Title = "Index";
}

@section css {
    <link rel="stylesheet" href="~/Content/CopyToClipboard.css">
    <link rel="stylesheet" href="~/Content/WalletIndex.css">
}

@section hint{
    <div id="copyHint" class="fixed-top p-3 bg-secondary text-light text-truncate d-none"></div>
}

@section searchForm{
    @{
        Html.RenderPartial("_SearchForm");
    }
}

<div class="d-flex flex-column">
    <h2 class="mb-3">Wallet</h2>
    <div class="d-flex mb-3">
        <h6 class="mb-0 d-none d-sm-block">Public Key: </h6>
        <h6 class="mb-0 d-block d-sm-none">PK: </h6>
         
        @if (Model.IsSuccessful)
        {
            <a id="wallet-pk" class="h6 mb-0 text-truncate text-dark pointer" title="Copy to clipboard">@Model.Model.PublicKey</a>
        }
        else
        {
            <p id="wallet-pk" class="h6 mb-0 text-truncate text-dark">@Model.Message</p>
        }
    </div>
    <div class="d-flex flex-column flex-sm-row-reverse justify-content-sm-between">
        <div class="d-flex mb-3 mb-sm-0 align-items-sm-end">
            <h6 class="mb-0">Balance: </h6>
             
            <h6 id="balance" class="mb-0"></h6>
        </div>
        <a class="btn btn-success col-12 col-sm-4 col-md-3 col-lg-2"
           href="/Transactions/CreateTransaction">Create Transaction</a>
    </div>
</div>
```

```
<hr />

<div class="col12 col-md-6 col-xl-4">
    <div class="form-text">Select transactions within a date (UTC)</div>
    <div class="input-group align-items-center mb-3">
        <input type="date" name="dateFrom" id="filter-date-from" class="form-control rounded mw-100" min="2022-01-01" />
        <div class="mx-2">-</div>
        <input type="date" name="dateTo" id="filter-date-to" class="form-control rounded mw-100" min="2022-01-01" />
    </div>
    <div class="form-text">Select transactions with specific wallet</div>
    <div class="input-group mb-3">
        <input type="search" name="wallet" id="filter-wallet" class="form-control rounded mw-100" placeholder="Wallet Public Key" />
    </div>
    <div class="d-flex flex-column flex-sm-row">
        <div class="w-100 me-sm-2">
            <div class="form-text">Select transactions status</div>
            <div class="input-group mb-3">
                <select class="form-select mw-100" id="filter-status" name="status">
                    @foreach (int value in Enum.GetValues(typeof(CryptoCurrency.Common.Models.TransactionsStatusFilter)))
                    {
                        var name = Enum.GetName(typeof(CryptoCurrency.Common.Models.TransactionsStatusFilter), value);
                        <option value="@value">@name</option>
                    }
                </select>
            </div>
        </div>
        <div class="w-100 ms-sm-2">
            <div class="form-text">Select transactions type</div>
            <div class="input-group mb-3">
                <select class="form-select mw-100" id="filter-type" name="type">
                    @foreach (int value in Enum.GetValues(typeof(CryptoCurrency.Common.Models.TransactionsTypeFilter)))
                    {
                        var name = Enum.GetName(typeof(CryptoCurrency.Common.Models.TransactionsTypeFilter), value);
                        <option value="@value">@name</option>
                    }
                </select>
            </div>
        </div>
    </div>
    <div class="mb-3">
        <button class="btn btn-primary" id="filters-apply-btn">Apply</button>
    </div>
</div>

<div class="transactions-container table-responsive">
    <table class="table table-bordered">
        <thead class="table-dark">
            <tr>
                <th class="align-middle">
                    <div class="d-flex justify-content-center justify-content-md-start">
                        Type
                    </div>
                </th>
                <th class="sorting pointer align-middle" id="date-sort" value="@CryptoCurrency.Common.Models.TransactionsSort.DateDesc">
                    <div class="d-flex align-items-center justify-content-center justify-content-md-start">
                        <div>Date</div>
                        <div class="mx-1 date-ico">
                            <svg xmlns="http://www.w3.org/2000/svg" width="16" height="16" fill="currentColor" class="bi bi-arrow-up sort-ico d-none" id="date-asc-ico" viewBox="0 0 16 16">
                                <path fill-rule="evenodd" d="M8 15a.5.5 0 0 0 .5-.5V2.707l3.146 3.147a.5.5 0 0 0 .708-.708l-4-4a.5.5 0 0 0-.708 0l-4 4a.5.5 0 1 0 .708.708L7.5 2.707V14.5a.5.5 0 0 0 .5.5z" />
                            </svg>
                            <svg xmlns="http://www.w3.org/2000/svg" width="16" height="16" fill="currentColor" class="bi bi-arrow-down sort-ico" id="date-desc-ico" viewBox="0 0 16 16">
                                <path fill-rule="evenodd" d="M8 1a.5.5 0 0 1 .5.5v11.793l3.146-3.147a.5.5 0 0 1 .708.708l-4 4a.5.5 0 0 1-.708 0l-4-4a.5.5 0 0 1 .708-.708L7.5 13.293V1.5A.5.5 0 0 1 8 1z" />
                            </svg>
                        </div>
                    </div>
                </th>
                <th class="sorting pointer align-middle" id="amount-sort" value="@CryptoCurrency.Common.Models.TransactionsSort.AmountDesc">
                    <div class="d-flex align-items-center justify-content-center justify-content-md-start">
                        <div>Amount</div>
                        <div class="mx-1 amount-ico">
                            <svg xmlns="http://www.w3.org/2000/svg" width="16" height="16" fill="currentColor" class="bi bi-arrow-up sort-ico d-none" id="amount-asc-ico" viewBox="0 0 16 16">
                                <path fill-rule="evenodd" d="M8 15a.5.5 0 0 0 .5-.5V2.707l3.146 3.147a.5.5 0 0 0 .708-.708l-4-4a.5.5 0 0 0-.708 0l-4 4a.5.5 0 1 0 .708.708L7.5 2.707V14.5a.5.5 0 0 0 .5.5z" />
                            </svg>
                            <svg xmlns="http://www.w3.org/2000/svg" width="16" height="16" fill="currentColor" class="bi bi-arrow-down sort-ico d-none" id="amount-desc-ico" viewBox="0 0 16 16">
                                <path fill-rule="evenodd" d="M8 1a.5.5 0 0 1 .5.5v11.793l3.146-3.147a.5.5 0 0 1 .708.708l-4 4a.5.5 0 0 1-.708 0l-4-4a.5.5 0 0 1 .708-.708L7.5 13.293V1.5A.5.5 0 0 1 8 1z" />
                            </svg>
                        </div>
                    </div>
                </th>
                <th class="align-middle">
                    <div class="d-flex justify-content-center justify-content-md-start">
                        Status
                    </div>
                </th>
            </tr>
        </thead>
        <tbody>
        </tbody>
    </table>
</div>

@{

Html.RenderPartial("_PaginationBut
```

```
tons");
}

@section scripts {
    <script src="~/Scripts/wallet-index.js"></script>
    <script src="~/Scripts/copy-to-clipboard.js"></script>
    <script src="~/Scripts/wallet-balance.js"></script>
    <script type="module" src="~/Scripts/wallet-index-signalR.js"></script>
    <script>
        $('.sorting').on('click', function () {
            let queryParams = new URLSearchParams(window.location.search);
            let sort = $(this);
            $('.sort-ico').addClass('d-none');

            if (sort.attr('value') == '@CryptoCurrency.Common.Models.TransactionsSort.DateDesc') {
                sort.attr('value', '@CryptoCurrency.Common.Models.TransactionsSort.DateAsc');
                $(`#date-asc-ico`).removeClass('d-none');
            } else if (sort.attr('value') == '@CryptoCurrency.Common.Models.TransactionsSort.DateAsc') {
                sort.attr('value', '@CryptoCurrency.Common.Models.TransactionsSort.DateDesc');
                $(`#date-desc-ico`).removeClass('d-none');
            } else if (sort.attr('value') == '@CryptoCurrency.Common.Models.TransactionsSort.AmountDesc') {
                sort.attr('value', '@CryptoCurrency.Common.Models.TransactionsSort.AmountAsc');
                $(`#amount-asc-ico`).removeClass('d-none');
            } else if (sort.attr('value') == '@CryptoCurrency.Common.Models.TransactionsSort.AmountAsc') {
                sort.attr('value', '@CryptoCurrency.Common.Models.TransactionsSort.AmountDesc');
                $(`#amount-desc-ico`).removeClass('d-none');
            }

            let sortValue = sort.attr('value');
            if (sortValue == '@CryptoCurrency.Common.Models.TransactionsSort.DateDesc') {
                queryParams.delete('sort');
            } else {
                queryParams.set('sort', sortValue);
            }

            history.replaceState(null, null, '?' + queryParams.toString());

            getPaginatedTransactionsAjax();
        })
    </script>
}
```

GetTransaction.cshtml

```
@model CryptoCurrency.Common.Models.ResponseResult<CryptoCurrency.Common.Models.GetWalletTransactionResponseModel>

@{
    ViewBag.Title = "GetTransaction";
}

@section searchForm{
    @{
        Html.RenderPartial("_SearchForm");
    }
}


@if (Model.IsSuccessful == false)
{
    <h2>@Model.Message</h2>
}
else if (Model.Model.Transaction == null)
{
    <h2>Can not find transaction with specified ID.</h2>
}
else
{
    <div class="d-flex mb-3">
        <h2 class="mb-0">Transaction</h2>
        <div class="h2 mb-0 d-none d-md-block">:</div>
        <div class="px-1"></div>
        <div class="h2 mb-0 d-block d-md-none">Details</div>
        <div class="h2 mb-0 d-none d-md-block">@Model.Model.Transaction.Id</div>
    </div>

    <hr />

    <div class="d-flex flex-column">
        <div class="d-flex mb-1">
            <div class="h5 m-0 fw-bold">Amount:</div>
            @if (@Model.Model.Transaction.Receiver == @Model.Model.WalletPublicKey)
            {
                <div class="h5 m-0 ms-1 text-success">

@Model.Model.Transaction.TransferAmount
                </div>
            }
            else if (@Model.Model.Transaction.Sender == @Model.Model.WalletPublicKey)
            {
                <div class="h5 m-0 ms-1 text-danger">
                    -
@Model.Model.Transaction.TransferAmount
                </div>
            }
        </div>
        <div class="d-flex mb-1">
            <div class="h5 m-0 fw-bold">Type:</div>
            <div class="d-flex align-items-end">
                @if (@Model.Model.Transaction.Receiver == @Model.Model.WalletPublicKey)
                {
                    <div class="h5 m-0 mx-1">@CryptoCurrency.Common.Models.TransactionsTypeFilter.Incoming</div>
                    <svg xmlns="http://www.w3.org/2000/svg" width="20" height="20" fill="currentColor" class="bi bi-arrow-down-circle" viewBox="0 0 16 16">
                        <path fill-rule="evenodd" d="M1 8a7 7 0 1 0 14 0A7 7 0 0 0 1 8zm15 0A8 8 0 1 1 0 8a8 8 0 0 1 16 0zM8.5 4.5a.5.5 0 0 0-1 0v5.793L5.354 8.146a.5.5 0 1 0-.708.708l3 3a.5.5 0 0 0 .708 0l3-3a.5.5 0 0 0-.708-.708L8.5 10.293V4.5z" />
                    </svg>
                }
                else if (@Model.Model.Transaction.Sender == @Model.Model.WalletPublicKey)
                {
                    <div class="h5 m-0 mx-1">@CryptoCurrency.Common.Models.TransactionsTypeFilter.Outgoing</div>
                    <svg xmlns="http://www.w3.org/2000/svg" width="20" height="20" fill="currentColor" class="bi bi-arrow-up-circle" viewBox="0 0 16
```

```
16">
                <path fill-
rule="evenodd" d="M1 8a7 7 0 1 0
14 0A7 7 0 0 0 1 8zm15 0A8 8 0 1 1
0 8a8 8 0 0 1 16 0zm-7.5 3.5a.5.5 0
0 1-1 0V5.707L5.354 7.854a.5.5 0 1
1-.708-.708l3-3a.5.5 0 0 1 .708 0l3
3a.5.5 0 0 1-.708.708L8.5
5.707V11.5z" />
            </svg>
        }
    </div>
</div>
<div class="d-flex d-md-none
mb-1">
    <div class="h5 m-0 fw-
bold">Id:</div>
    <div class="h5 m-0 ms-1
text-truncate">

@Model.Model.Transaction.Id
    </div>
</div>
<div class="d-flex mb-1">
    @if
(@Model.Model.Transaction.Receiv
er ==
@Model.Model.WalletPublicKey)
    {
        <div class="h5 m-0 fw-
bold">From:</div>
        <div class="h5 m-0 ms-1
text-truncate">

@Model.Model.Transaction.Sender
        </div>
    }
    else if
(@Model.Model.Transaction.Sender
==
@Model.Model.WalletPublicKey)
    {
        <div class="h5 m-0 fw-
bold">To:</div>
        <div class="h5 m-0 ms-1
text-truncate">

@Model.Model.Transaction.Receive
r
        </div>
    }
</div>
<div class="d-flex mb-1">
    <div class="h5 m-0 fw-
bold">Date:</div>
    <div class="h5 m-0 ms-1">

@Model.Model.Transaction.DateLa
stUpdated
    </div>
</div>
<div class="d-flex mb-1">
    <div class="h5 m-0 fw-
bold">Status:</div>
    <div class="d-flex align-
items-end">
        @if
(Model.Model.Transaction.Status ==
CryptoCurrency.Common.Models.St
atus.Unprocessed)
        {
            <div class="h5 m-0 mx-
1">Unprocessed</div>
            <svg
xmlns="http://www.w3.org/2000/sv
g" width="22" height="22"
fill="currentColor" class="bi bi-
arrow-clockwise" viewBox="1 0 16
16">
                <path fill-
rule="evenodd" d="M8 3a5 5 0 1 0
4.546 2.914.5.5 0 0 1 .908-.417A6 6
0 1 1 8 2v1z" />
                <path d="M8
4.466V.534a.25.25 0 0 1 .41-
.192l2.36 1.966c.12.1.12.284 0
.384L8.41 4.658A.25.25 0 0 1 8
4.466z" />
            </svg>
        }
        else if
(Model.Model.Transaction.Status ==
CryptoCurrency.Common.Models.St
atus.Processed)
        {
            <div class="h5 m-0 mx-
1">Processed</div>
            <svg
xmlns="http://www.w3.org/2000/sv
g" width="22" height="22"
fill="currentColor" class="bi bi-
check-lg" viewBox="2 0 16 16">
                <path d="M12.736
3.97a.733.733 0 0 1 1.047
0c.286.289.29.756.01 1.05L7.88
12.01a.733.733 0 0 1-
1.065.02L3.217 8.384a.757.757 0 0
1 0-1.06.733.733 0 0 1 1.047 0l3.052
3.093 5.4-6.425a.247.247 0 0 1 .02-
.022Z" />
            </svg>
        }
    </div>
</div>
</div>
}
<a href="/Wallet/Index" class="btn
btn-success mt-2 fs-5 d-sm-
none">Wallet</a>
```

Cryptocurrency.MiningServi ce.BL

MiningService.cs

```
using System.Threading;
using
CryptoCurrency.Common.Interfaces
;
using
CryptoCurrency.Common.Models;
using
CryptoCurrency.MiningService.Inter
faces;
using
CryptoCurrency.TransactionService.
Interfaces;
using Microsoft.Extensions.Logging;

namespace
CryptoCurrency.MiningService.BL
{
    /// <summary>
    /// Class for mining service
representation.
    /// </summary>
    /// <remarks>
    ///
[![Flowchart](../images/MS.jpg)](../i
mages/MS.jpg)
    /// </remarks>
    public partial class MiningService
: IMiningServiceLifecycle
    {
        private const string
TransactionServiceObservableName
= "TransactionService";
        private readonly
IObservableServiceProvider
observableServiceProvider;
        private readonly
IMiningServiceHost
transactionServiceInstance;
        private readonly
IMinerServiceSettings
minerServiceSettings;
        private readonly
IBlockCalculationUtils
blockCalculationUtils;
        private readonly
ILogger<MiningService> logger;
        private volatile bool
isLifeCycleActive;
        private volatile bool
isCancelCalculation;

        /// <summary>
        /// Initializes a new instance of
the <see cref="MiningService"/>
class.
        /// </summary>
        /// <param
name="miningServiceHost">Instanc
e of transaction service through
interface for mining
service.</param>
        /// <param
name="minerServiceSettings">Setti
ngs for miner service
registration.</param>
        /// <param
name="calculationsUtils">Utils for
block hash calculation and
validation.</param>
        /// <param
name="observableServiceProvider">
Instance of transaction service to
observe an events.</param>
        /// <param
```

```
name="logger">Logger.</param>
        public
MiningService(IMiningServiceHost
miningServiceHost,
IMinerServiceSettings
minerServiceSettings,
IBlockCalculationUtils
calculationsUtils,
IObservableServiceProvider
observableServiceProvider,
ILogger<MiningService> logger)
        {

this.observableServiceProvider =
observableServiceProvider;
            transactionServiceInstance =
miningServiceHost;
            this.minerServiceSettings =
minerServiceSettings;
            blockCalculationUtils =
calculationsUtils;
            this.logger = logger;
        }

        /// <summary>
        /// Gets a user id property of
mining service.
        /// </summary>
        public long UserId { get;
private set; }

        /// <inheritdoc
cref="IMiningServiceLifecycle.Start
Mining"/>
        public void StartMining()
        {
            if (!isLifeCycleActive)
            {
                isLifeCycleActive = true;
                new Thread(Mine).Start();
            }
        }

        /// <inheritdoc
cref="IMiningServiceLifecycle.Stop
Mining"/>
        public void StopMining()
        {
            isLifeCycleActive = false;
        }

        /// <inheritdoc
cref="IMiningServiceLifecycle.Start
(User)"/>
        public bool Start(User user)
        {
            if (IsUserMiner(user))
            {
                UserId = user.UserId;
                var transactionService =
observableServiceProvider.Get(Tran
sactionServiceObservableName);
                transactionService.Notify -
= OnNotify;
                transactionService.Notify
+= OnNotify;
                return true;
            }

            return false;
        }

        /// <inheritdoc
cref="IMiningServiceLifecycle.Stop
"/>
        public void Stop()
        {
            StopMining();
            var transactionService =
observableServiceProvider.Get(Tran
sactionServiceObservableName);
            transactionService.Notify -=
OnNotify;
        }

        /// <summary>
        /// Event handler.
        /// </summary>
        /// <param
name="args">Additional parameters
with event name.</param>
        internal void
OnNotify(ObserverNotificationArgs
args)
        {
            switch (args.EventName)
            {
                case "DropBlock":
                    {
                        DropBlock();
                        break;
                    }

                default:
                    break;
            }
        }

        /// <summary>
        /// Check a user for the miner
role.
        /// </summary>
        /// <param name="user">User
who is verified to be a
miner.</param>
        /// <returns>true if user has role
miner; otherwise, false.</returns>
        private static bool
IsUserMiner(User user)
        {
            return
user.Roles.HasFlag(Roles.Miner);
        }

        private void DropBlock()
        {
            isCancelCalculation = true;
        }

        /// <summary>
        /// Mining loop.
        /// </summary>
        private void Mine()
        {
            while (isLifeCycleActive)
            {
                isCancelCalculation =
false;

                var blockInfo =
transactionServiceInstance.GetMinin
gProcessInformation(UserId);
                if (blockInfo == null ||

blockInfo.PreviousBlockHash ==
null ||
                    blockInfo.Transactions
== null ||

blockInfo.HashZeroNumber == null)
                {

Thread.Sleep(minerServiceSettings.
DelayBeforeNextMiningProcessInfo
rmationRequest);
                    continue;
                }


logger.LogInformation("Data for
new block calculation: {@data}",
blockInfo);
                var block = new
Block(blockInfo.PreviousBlockHash
, blockInfo.Transactions);

                CalculateNonce(block,
blockInfo.HashZeroNumber);

logger.LogInformation("Current
block successfully calculated:
{@block}", block);

                if (!isCancelCalculation
&&
blockCalculationUtils.ValidateHash(
block.Hash,
blockInfo.HashZeroNumber))
                {

logger.LogInformation("Block
'{@hash}' has passed validation and
will be sent to the server.",
block.Hash);

transactionServiceInstance.AddBloc
k(block);
                }
            }
        }

        /// <summary>
        /// Calculates nonce for current
block.
        /// </summary>
        /// <param
name="block">Current block for
hash calculating.</param>
```

```
        /// <param
name="condition">The string that
the valid hash should start
with.</param>
        private void
CalculateNonce(Block block, string
condition)
        {
            string transactionsHash =
blockCalculationUtils.CalculateHash
(block.Transactions);

            for (; block.Nonce <
uint.MaxValue; block.Nonce++)
            {
                block.Hash =
blockCalculationUtils.CalculateHash
(block.PreviousBlockHash,
transactionsHash, block.Nonce);
                if (isCancelCalculation ||
!isLifeCycleActive ||
blockCalculationUtils.ValidateHash(
block.Hash, condition))
                {
                    break;
                }
            }
        }
    }
}
```

UserProvider.cs

```
using System;
using System.Linq;
using System.Security.Claims;
using System.Threading.Tasks;
using
CryptoCurrency.Common.Interfaces
;
using
CryptoCurrency.Common.Models;
using
CryptoCurrency.MiningService.Inter
faces;
using Microsoft.Extensions.Logging;

namespace
CryptoCurrency.MiningService.BL
{
    /// <summary>
    /// Provides users from source.
    /// </summary>
    public class UserProvider :
IUserProvider
    {
        private const string ServerName
= "CustomerService";
        private const string
LoginEndpointName =
"LoginEndpoint";
        private readonly
ILogger<UserProvider> logger;
        private readonly
IUserConfiguration
userConfiguration;
        private readonly
IEndpointConfigurationTemp
endpointConfiguration;
        private readonly IHttpHelper
httpHelper;

        /// <summary>
        /// Initializes a new instance of
the <see cref="UserProvider"/>
class.
        /// </summary>
        /// <param
name="logger">Logger.</param>
        /// <param
name="userConfiguration">Configu
ration manager for user
info.</param>
        /// <param
name="endpointConfiguration">Con
figuration manager for
endpoints.</param>
        /// <param
name="httpHelper">HTTP
client.</param>
        public
UserProvider(ILogger<UserProvider
> logger, IUserConfiguration
userConfiguration,
IEndpointConfigurationTemp
endpointConfiguration, IHttpHelper
httpHelper)
        {
            this.logger = logger;
            this.userConfiguration =
userConfiguration;
            this.endpointConfiguration =
endpointConfiguration;
            this.httpHelper = httpHelper;
        }

        /// <inheritdoc
cref="IUserProvider.GetUser"/>
        public async
Task<ResponseResult<User>>
GetUser()
        {
            UserInfoModel userInfo;
            try
            {
                userInfo =
userConfiguration.GetUserInfo();
            }
            catch (Exception ex)
            {
                logger.LogError(ex,
"Exception occurred during reading
configuration file");
                return new
ResponseResult<User>()
                {
                    Model = default,
                    IsSuccessful = false,
                    Message =
ResponseMessages.CannotReadUser
InfoSection,
                };
            }
            if (userInfo.IsConfigured())
            {
                var endpoint =
endpointConfiguration.GetEndpoint(
ServerName, LoginEndpointName);
                var
userDataRequestResponse = await
httpHelper.PostAsync<UserInfoMod
el, LoginResponse>(userInfo,
endpoint);
                var userData =
userDataRequestResponse.Model;
                if (userData != null)
                {
                    var roles =
userData.UserClaims.FirstOrDefault
(c => c.Type == ClaimTypes.Role);
                    var id =
userData.UserClaims.FirstOrDefault
(c => c.Type ==
ClaimTypes.NameIdentifier);
                    if (roles != null && id
!= null)
                    {
                        Roles userRoles =
default;
                        foreach (var role in
roles.Value.Split(','))
                        {
                            _ =
Enum.TryParse(role.Trim(), out
Roles userRole);
                            userRoles =
(Roles)((int)userRoles +
(int)userRole);
                        }

                        var userId =
Convert.ToInt64(id.Value);

                        var user = new
User(userId, userRoles);
                        return new
ResponseResult<User>()
                        {
                            Model = user,
                            IsSuccessful = true,
                            Message =
string.Empty,
                        };
                    }


logger.LogWarning("Request to the
server to get user data for login
returned wrong response");
                    return new
ResponseResult<User>()
                    {
                        Model = default,
                        IsSuccessful = false,
                        Message =
ResponseMessages.ServerErrorUrea
dableUserData,
                    };
                }
```

```
logger.LogWarning(userDataRequestResponse.Message);
            return new ResponseResult<User>()
            {
                Model = default,
                IsSuccessful = false,
                Message = userDataRequestResponse.Message,
            };
        }

        logger.LogWarning("Config file contains wrong user data.");
        return new ResponseResult<User>()
        {
            Model = default,
            IsSuccessful = false,
            Message = ResponseMessages.InvalidUserInfoSection,
        };
    }
  }
}
```

LifecycleManager.cs

```
using System.Threading.Tasks;
using CryptoCurrency.Common.Interfaces;
using CryptoCurrency.Common.Models;
using CryptoCurrency.MiningService.Interfaces;
using CryptoCurrency.MiningService.Models;

namespace CryptoCurrency.MiningService.BL
{
  /// <summary>
  /// A class for managing mining service life cycle.
  /// </summary>
  /// <remarks>
  /// [![Flowchart](../images/MSLifecycleManager.jpg)](../images/MSLifecycleManager.jpg)
  /// </remarks>
  public partial class LifecycleManager : ILifecycleManager
  {
    private readonly IMiningServiceLifecycle miningServiceLifecycle;
    private readonly IUserProvider userProvider;
    private readonly IOutputManager outputManager;
    private readonly IApplicationManager applicationManager;

    /// <summary>
    /// Initializes a new instance of the <see cref="LifecycleManager"/> class.
    /// </summary>
    /// <param name="miningServiceLifecycle">Mining service instance.</param>
    /// <param name="userProvider">User provider.</param>
    /// <param name="outputManager">Output manager.</param>
    /// <param name="applicationManager">Program life cycle manager.</param>
    public LifecycleManager(IMiningServiceLifecycle miningServiceLifecycle, IUserProvider userProvider, IOutputManager outputManager, IApplicationManager applicationManager)
    {
      this.miningServiceLifecycle = miningServiceLifecycle;
      this.userProvider = userProvider;
      this.outputManager = outputManager;
      this.applicationManager = applicationManager;
    }

    /// <inheritdoc cref="ILifecycleManager.AppStop"/>
    public void AppStop()
    {
      miningServiceLifecycle.Stop();
      applicationManager.Exit();
    }

    /// <inheritdoc cref="ILifecycleManager.Run"/>
    public async Task Run()
    {
      var user = await GetUser();
      if (miningServiceLifecycle.Start(user))
      {
        StartMining();
      }
      else
      {
        outputManager.PrintMessage(OutputMessages.UserHasNotMinerRole);
        applicationManager.Exit();
      }
    }

    /// <inheritdoc cref="ILifecycleManager.StopMining"/>
    public void StopMining()
    {
      miningServiceLifecycle.StopMining();

      outputManager.PrintMessage(OutputMessages.MiningProcessInterrupted);
    }

    /// <inheritdoc cref="ILifecycleManager.StartMining"/>
    public void StartMining()
    {
      outputManager.PrintMessage(OutputMessages.MiningProcessStarted);

      miningServiceLifecycle.StartMining();
    }

    private async Task<User> GetUser()
    {
      var userResponse = await userProvider.GetUser();
      if (!userResponse.IsSuccessful)
      {
        outputManager.PrintError(userResponse.Message);
        applicationManager.Exit();
      }

      return userResponse.Model;
    }
  }
```

Cryptocurrency.MiningService.Interfaces

IApplicationManager.cs

```
namespace CryptoCurrency.MiningService.Interfaces
{
  /// <summary>
  /// Interface for managing application life cycle.
  /// </summary>
  /// <remarks>
  /// [![Flowchart](../images/MS_IApplicationManager.jpg)](../images/MS_IApplicationManager.jpg)
  /// </remarks>
  public interface
```

```
IApplicationManager
    {
        /// <summary>
        /// Closes application.
        /// </summary>
        void Exit();
    }
}
```

ICommandHandler.cs

```
namespace
CryptoCurrency.MiningService.Inter
faces
{
    /// <summary>
    /// An interface for running
application.
    /// </summary>
    /// <remarks>
    ///
[![Flowchart](../images/MS_IComan
dHandler.jpg)](../images/MS_IComa
ndHandler.jpg)
    /// </remarks>
    public interface
ICommandHandler
    {
        /// <summary>
        /// Initiates application start.
        /// </summary>
        void AppStart();

        /// <summary>
        /// Initiates application stop.
        /// </summary>
        void AppStop();
    }
}
```

ILifecycleManager.cs

```
using System.Threading.Tasks;

namespace
CryptoCurrency.MiningService.Inter
faces
{
    /// <summary>
    /// An interface for command
calls.
    /// </summary>
    /// <remarks>
    ///
[![Flowchart](../images/MS_ILifecy
cleManager.jpg)](../images/MS_ILif
ecycleManager.jpg)
    /// </remarks>
    public interface
ILifecycleManager
    {
        /// <summary>
        /// Initiates stopping mining
service.
        /// </summary>
        void AppStop();

        /// <summary>
        /// Initiates starting mining
process.
        /// </summary>
        void StartMining();

        /// <summary>
        /// Initiates stopping mining
process.
        /// </summary>
        void StopMining();

        /// <summary>
        /// Initiates starting mining
service life cycle.
        /// </summary
        /// <returns>Task</returns>
        Task Run();
    }
}
```

IOutputManager.cs

```
using
CryptoCurrency.Common.Models;

namespace
CryptoCurrency.MiningService.Inter
faces
{
    /// <summary>
    /// An interface responsible for
displaying information.
    /// </summary>
    /// <remarks>
    ///
[![Flowchart](../images/MS_IOutput
Manager.jpg)](../images/MS_IOutpu
tManager.jpg)
    /// </remarks>
    public interface IOutputManager
    {
        /// <summary>
        /// Displays information about
block.
        /// </summary>
        /// <param
name="block">Block.</param>
        void PrintBlock(Block block);

        /// <summary>
        /// Displays some message.
        /// </summary>
        /// <param
name="message">Text.</param>
        void PrintMessage(string
message);

        /// <summary>
        /// Displays error message.
        /// </summary>
        /// <param
name="error">Text.</param>
        void PrintError(string error);

        /// <summary>
        /// Displays mining process
information.
        /// </summary>
        /// <param
name="mpi">Mining process
information.</param>
        void
PrintMiningProcessInformation(Min
ingProcessInformation mpi);
    }
}
```

Cryptocurrency.MiningServi ce.Models

Command.cs

```
using System;

namespace
CryptoCurrency.MiningService.Mod
els
{
    /// <summary>
    /// A class represents command for
user input.
    /// </summary>
    /// <remarks>
    ///
[![Flowchart](../images/MS_Comma
nd.jpg)](../images/MS_Command.jp
g)
    /// </remarks>
    public class Command
    {
        /// <summary>
        /// Initializes a new instance of
the <see cref="Command"/> class.
        /// </summary>
        /// <param
name="description">Command
description.</param>
        /// <param
name="action">Command
action.</param>
        public Command(string
description, Action action)
        {
            Description = description;
            Action = action;
        }

        /// <summary>
        /// Gets command description.
        /// </summary>
        public string Description { get;
}

        /// <summary>
        /// Gets command action.
        /// </summary>
        public Action Action { get; }
    }
}
```

OutputMessages.cs

```
namespace
CryptoCurrency.MiningService.Mod
els
{
    /// <summary>
    /// Provides messages for mining
client.
```

```
    /// </summary>
    /// <remarks>
    ///
[![Flowchart](../images/MS_Output
Messages.jpg)](../images/MS_Outpu
tMessages.jpg)
    /// </remarks>
    public static class
OutputMessages
    {
        /// <summary>
        /// Message about current active
state of the mining process.
        /// </summary>
        public const string
MiningProcessStarted = "Mining
process started.";

        /// <summary>
        /// Message about current
inactive state of the mining process.
        /// </summary>
        public const string
MiningProcessInterrupted = "Mining
process interrupted. Press 'Ctrl+E' to
continue mining process.";

        /// <summary>
        /// Message for case when user
tries to start mining without Miner
role.
        /// </summary>
        public const string
UserHasNotMinerRole = "User has
not miner role. Make sure that
account you use has miner
permissions.";
    }
}
```

Cryptocurrency.MiningService.Client

Program.cs

```
using
System.Diagnostics.CodeAnalysis;
using System.IO;
using CryptoCurrency.Common.BL;
using
Microsoft.AspNetCore.Hosting;
using
Microsoft.Extensions.Configuration;
using
Microsoft.Extensions.DependencyIn
jection;
using Microsoft.Extensions.Hosting;
using Serilog;

namespace
CryptoCurrency.MiningService.Clie
nt
{
    /// <summary>
    /// Encapsulates the application's
main entry point.
    /// </summary>
    [ExcludeFromCodeCoverage]
    public static class Program
    {
        /// <summary>
        /// The application's entry point.
        /// </summary>
        public static void Main()
        {
            var config = new
ConfigurationBuilder()

.SetBasePath(Directory.GetCurrentD
irectory())

.AddJsonFile("appsettings.json")
                .Build();

            Log.Logger = new
LoggerConfiguration()

.ReadFrom.Configuration(config)
                .CreateLogger();


CreateHostBuilder(config).Build().R
un();
        }

        /// <summary>
        /// Creates Host Builder.
        /// </summary>
        /// <param
name="configuration">An interface
for reading config file..</param>
        /// <returns>The initialized
IHostBuilder.</returns>
        public static IHostBuilder
CreateHostBuilder(IConfiguration
configuration) =>
            Host.CreateDefaultBuilder()
                .UseSerilog()

.ConfigureWebHostDefaults(webBuil
der =>
                {

webBuilder.UseUrls(GetLocalUrl(co
nfiguration));

webBuilder.UseStartup<Startup>();

}).ConfigureServices(services =>
services.AddHostedService<Backgr
oundComandHandler>());

        private static string
GetLocalUrl(IConfiguration
configuration)
        {
            var netUtils = new
NetUtils(configuration);
            var localUrl =
netUtils.GetLocalEndpoint();
            return localUrl;
        }
    }
}
```

Startup.cs

```
using
System.Diagnostics.CodeAnalysis;
using CryptoCurrency.Common.BL;
using
CryptoCurrency.Common.Interfaces
;
using
CryptoCurrency.Common.Models;
using
CryptoCurrency.Common.Signalr;
using
CryptoCurrency.MiningService.BL;
using
CryptoCurrency.MiningService.Inter
faces;
using
CryptoCurrency.TransactionService.
Interfaces;
using
Microsoft.AspNetCore.Builder;
using
Microsoft.AspNetCore.SignalR.Clie
nt;
using
Microsoft.Extensions.DependencyIn
jection;

namespace
CryptoCurrency.MiningService.Clie
nt
{
    /// <summary>
    /// Provides the startup methods
for the web application.
    /// </summary>
    [ExcludeFromCodeCoverage]
    public class Startup
    {
        /// <summary>
        /// Adds services to the web
application container.
        /// </summary>
        /// <param
name="services">Collection of
service descriptors.</param>
        public void
ConfigureServices(IServiceCollectio
n services)
        {
            services.AddControllers();

services.AddHttpClient<IHttpClient
Wrapper, HttpClientWrapper>();
            services.AddOptions();

services.AddSingleton<ICommandH
andler, CommandHandler>()

.AddTransient<IHttpHelper,
HttpHelper>()

.AddTransient<IUserConfiguration,
ConfigurationManager>()

.AddTransient<IUserConfiguration,
```

```
ConfigurationManager>()

.AddTransient<IEndpointConfiguration, ConfigurationManager>()

.AddTransient<IEndpointConfigurationTemp, ConfigurationManager>()

.AddSingleton<IMiningServiceHost, MiningServiceHost>()

.AddSingleton<IApplicationManager, ApplicationManager>()

.AddSingleton<ILifecycleManager, LifecycleManager>()

.AddSingleton<IMiningServiceLifecycle, BL.MiningService>()

.AddTransient<IMinerServiceSettings, MinerServiceSettings>()

.AddSingleton<IBlockCalculationUtils, BlockCalculationUtils>()

.AddSingleton<IOutputManager, OutputManagerConsole>()

.AddSingleton<IUserProvider, UserProvider>()

.AddSingleton<IObservableServiceProvider, ObservableServiceSignalrProxyProvider>()

.AddSingleton<IObservablesManager<ObserverParameters, IObservableServiceSignalrProxy>, ObservablesManager<ObserverParameters, IObservableServiceSignalrProxy>>()
            .AddSingleton<INetUtils, NetUtils>()

.AddTransient<IObservableServiceWebProxy, ObservableServiceWebProxy>()

.AddTransient<IObservableServiceProxyController, ObservableServiceProxyControllerBL>()

.AddTransient<IHubConnectionBuilder, HubConnectionBuilder>()

.AddTransient<IObservableServiceSignalrProxy, ObservableServiceSignalrProxy>();
        }

        /// <summary>
        /// Request processing pipeline with a sequence of middleware components.
        /// </summary>
        /// <param name="app">Class to configure an application's request pipeline.</param>
        public void Configure(IApplicationBuilder app)
        {
            app.UseRouting();

            app.UseEndpoints(endpoints =>
            {

endpoints.MapControllers();
            });
        }
    }
}
```

BackgroundCommandHadler.cs

```
using System.Threading;
using System.Threading.Tasks;
using CryptoCurrency.MiningService.Interfaces;
using Microsoft.Extensions.Hosting;

namespace CryptoCurrency.MiningService.Client
{
    /// <summary>
    /// Allows to start service as a background.
    /// </summary>
    public class BackgroundComandHandler : IHostedService
    {
        private readonly ICommandHandler commandHandler;

        /// <summary>
        /// Initializes a new instance of the <see cref="BackgroundComandHandler"/> class.
        /// </summary>
        /// <param name="commandHandler">Handler for user inputs.</param>
        public BackgroundComandHandler(ICommandHandler commandHandler)
        {
            this.commandHandler = commandHandler;
        }

        /// <inheritdoc cref="IHostedService.StartAsync(CancellationToken)"/>
        public Task StartAsync(CancellationToken cancellationToken)
        {

commandHandler.AppStart();
            return Task.CompletedTask;
        }

        /// <inheritdoc cref="IHostedService.StopAsync(CancellationToken)"/>
        public Task StopAsync(CancellationToken cancellationToken)
        {
            commandHandler.AppStop();
            return Task.CompletedTask;
        }
}
```

ApplicationManager.cs

```
using System;
using System.Diagnostics.CodeAnalysis;
using CryptoCurrency.MiningService.Interfaces;

namespace CryptoCurrency.MiningService.Client
{
    /// <summary>
    /// Class for managing application life cycle.
    /// </summary>
    [ExcludeFromCodeCoverage]
    public class ApplicationManager : IApplicationManager
    {
        /// <inheritdoc cref="IApplicationManager.Exit"/>
        public void Exit()
        {
            Environment.Exit(0);
        }
    }
}
```

CommandHandler.cs

```
using System;
using System.Collections.Generic;
using System.Diagnostics.CodeAnalysis;
using CryptoCurrency.MiningService.Interfaces;
using CryptoCurrency.MiningService.Models;

namespace CryptoCurrency.MiningService.Client
{
    /// <summary>
    /// A class for handling user inputs and managing application behavior.
```

```
    /// </summary>
    public class CommandHandler :
ICommandHandler
    {
        private readonly
ILifecycleManager
lifecycleManager;
        private readonly
Dictionary<ConsoleKeyInfo,
Command> commandsKey;

        /// <summary>
        /// Initializes a new instance of
the <see cref="CommandHandler"/>
class.
        /// </summary>
        /// <param
name="lifecycleManager">Life
cycle manager.</param>
        public
CommandHandler(ILifecycleManag
er lifecycleManager)
        {
            this.lifecycleManager =
lifecycleManager;
            commandsKey = new
Dictionary<ConsoleKeyInfo,
Command>();
        }

        /// <inheritdoc
cref="ICommandHandler.AppStart"/
>
        [ExcludeFromCodeCoverage]
        public void AppStart()
        {
            FillCommandsDictionary();
            lifecycleManager.Run();

Console.TreatControlCAsInput =
true;
            while (true)
            {
                var keyInfo =
Console.ReadKey(true);
                if
(commandsKey.ContainsKey(keyInf
o))
                {

commandsKey[keyInfo].Action.Invo
ke();
                }
            }
        }

        /// <inheritdoc
cref="ICommandHandler.AppStop"/
>
        public void AppStop()
        {
            lifecycleManager.AppStop();
        }

        [ExcludeFromCodeCoverage]
        private void
FillCommandsDictionary()
        {
            commandsKey.Add(new
ConsoleKeyInfo((char)17,
ConsoleKey.Q, false, false, true),
new Command("Stops application",
lifecycleManager.AppStop));
            commandsKey.Add(new
ConsoleKeyInfo((char)4,
ConsoleKey.D, false, false, true),
new Command("Stops mining",
lifecycleManager.StopMining));
            commandsKey.Add(new
ConsoleKeyInfo((char)5,
ConsoleKey.E, false, false, true),
new Command("Starts mining",
lifecycleManager.StartMining));
        }
    }
}
```

OutputManagerConsole.cs

```
using System;
using
System.Diagnostics.CodeAnalysis;
using
CryptoCurrency.Common.Models;
using
CryptoCurrency.MiningService.Inter
faces;

namespace
CryptoCurrency.MiningService.Clie
nt
{
    /// <summary>
    /// A class responsible for
displaying data in the console UI.
    /// </summary>
    [ExcludeFromCodeCoverage]
    public class
OutputManagerConsole :
IOutputManager
    {
        /// <inheritdoc
cref="IOutputManager.PrintBlock(B
lock)"/>
        public void PrintBlock(Block
block)
        {
            Console.ForegroundColor =
ConsoleColor.Green;
            Console.Write("Calculated
block: ");
            Console.ForegroundColor =
ConsoleColor.White;

Console.WriteLine($"{block.Hash}"
);
            Console.WriteLine(new
string('-', Console.WindowWidth));
        }

        /// <inheritdoc
cref="IOutputManager.PrintMessage
(string)"/>
        public void PrintMessage(string
message)
        {
            Console.ForegroundColor =
ConsoleColor.Yellow;

Console.WriteLine(message);
            Console.ForegroundColor =
ConsoleColor.White;
        }

        /// <inheritdoc
cref="IOutputManager.PrintError(str
ing)"/>
        public void PrintError(string
error)
        {
            Console.ForegroundColor =
ConsoleColor.Red;
            Console.WriteLine(error);
            Console.ForegroundColor =
ConsoleColor.White;
        }

        /// <inheritdoc
cref="IOutputManager.PrintMiningP
rocessInformation(MiningProcessInf
ormation)"/>
        public void
PrintMiningProcessInformation(Min
ingProcessInformation mpi)
        {
            Console.ForegroundColor =
ConsoleColor.Green;
            Console.Write("MPI: ");
            Console.ForegroundColor =
ConsoleColor.White;

Console.WriteLine(DateTime.Now);

Console.WriteLine($"{mpi}");
        }
    }
}
```

TransactionServiceProxyContro
ller.cs

```
using
CryptoCurrency.Common.Interfaces
;
using
CryptoCurrency.Common.Models;
using Microsoft.AspNetCore.Mvc;

namespace
CryptoCurrency.MiningService.Clie
nt.Controllers
{
    /// <summary>
    /// Controller for receiving calls
from the server.
    /// </summary>

[Route("api/[controller]/[action]")]
    [ApiController]
    public class
TransactionServiceProxyController :
ControllerBase,
```

```
IObservableServiceProxyController
    {
        private readonly
IObservableServiceProxyController
observableServiceProxyControllerB
L;

        /// <summary>
        /// Initializes a new instance of
the <see
cref="TransactionServiceProxyContr
oller"/> class.
        /// </summary>
        /// <param
name="observableServiceProxyCont
rollerBL">BL of observable service
proxy controller.</param>
        public
TransactionServiceProxyController(I
ObservableServiceProxyController
observableServiceProxyControllerB
L)
        {

this.observableServiceProxyControll
erBL =
observableServiceProxyControllerB
L;
        }

        /// <inheritdoc
cref="IObservableServiceProxyCont
roller.IsAvailable(ObserverAddress)
"/>
        [HttpPost]
        public ResponseResult
IsAvailable(ObserverAddress
observerAddress)
        {
            return
observableServiceProxyControllerB
L.IsAvailable(observerAddress);
        }

        /// <inheritdoc
cref="IObservableServiceProxyCont
roller.Notify(ObserverNotifyInfo)"/>
        [HttpPost]
        public void
Notify(ObserverNotifyInfo
observerNotifyInfo)
        {

observableServiceProxyControllerB
L.Notify(observerNotifyInfo);
        }
    }
}
```

Cryptocurrency.DAL.BL

SqlDataAccess.cs

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
using
System.Diagnostics.CodeAnalysis;
using System.Threading.Tasks;
using Dapper;

namespace CryptoCurrency.DAL
{
    /// <summary>
    /// Provides a methods for
accessing database.
    /// </summary>
    /// <remarks>
    ///
[![Flowchart](../images/SqlDataAcce
ss.jpg)](../images/SqlDataAccess.jpg
)
    /// </remarks>
    public partial class SqlDataAccess
: ISqlDataAccess
    {
        private readonly
IConfigurationManager
configurationManager;
        private readonly
IDbConnection connection;
        private string connectionString;

        /// <summary>
        /// Initializes a new instance of
the <see cref="SqlDataAccess"/>
class.
        /// </summary>
        /// <param
name="configurationManager">Inte
rface for interaction with
configuration file.</param>
        /// <param
name="logger">Interface for
logging.</param>
        /// <exception
cref="ConfigurationErrorsException
">Thrown when <see
cref="IConfigurationManager"/>
cannot find configuration file or
connection string
inside.</exception>
        [ExcludeFromCodeCoverage]
        public
SqlDataAccess(IConfigurationMana
ger configurationManager)
        {
            this.configurationManager =
configurationManager;
            connectionString =
this.configurationManager.GetConn
ectionString();

            if (connectionString == null)
            {
                throw new
ConfigurationErrorsException("Conf
iguration file doesn't exist or it's not
contains connection string");
            }
            else
            {
                connection = new
SqlConnection(connectionString);
            }
        }

        /// <summary>
        /// Initializes a new instance of
the <see cref="SqlDataAccess"/>
class
        /// for testing communication
with database through Dapper.
        /// </summary>
        /// <param
name="connection">Interface
through which dapper will
communicate with
database.</param>
        public
SqlDataAccess(IDbConnection
connection)
        {
            this.connection = connection;
        }

        /// <inheritdoc
cref="ISqlDataAccess.Execute(strin
g, object)"/>
        public int Execute(string
procedure, object arguments = null)
        {
            return
connection.Execute(procedure,
arguments, commandType:
CommandType.StoredProcedure);
        }

        /// <inheritdoc
cref="ISqlDataAccess.Query{T}(stri
ng, object)"/>
        public IEnumerable<T>
Query<T>(string procedure, object
arguments = null)
        {
            IEnumerable<T> entities;

            entities =
connection.Query<T>(procedure,
arguments, commandType:
CommandType.StoredProcedure);

            return entities;
        }

        /// <inheritdoc
cref="ISqlDataAccess.QueryMultipl
e(string, IEnumerable{Type},
object)"/>
        public Dictionary<Type,
IEnumerable<object>>
QueryMultiple(string procedure,
IEnumerable<Type> types, object
arguments = null)
        {
            var entities = new
Dictionary<Type,
IEnumerable<object>>();
```

```
            var multi =
connection.QueryMultiple(procedure
, arguments, commandType:
CommandType.StoredProcedure);
            try
            {
                foreach (var item in types)
                {
                    entities.Add(item,
multi.Read(item));
                }
            }
            catch (Exception)
            {
                throw new
InvalidOperationException("Data
cannot be read!");
            }

            return entities;
        }

        /// <inheritdoc
cref="ISqlDataAccess.ExecuteAsyn
c(string, object)"/>
        public async Task<int>
ExecuteAsync(string procedure,
object arguments = null)
        {
            return await
connection.ExecuteAsync(procedure
, arguments, commandType:
CommandType.StoredProcedure);
        }

        /// <inheritdoc
cref="ISqlDataAccess.QueryAsync{
T}(string, object)"/>
        public async
Task<IEnumerable<T>>
QueryAsync<T>(string procedure,
object arguments = null)
        {
            IEnumerable<T> entities;

            entities = await
connection.QueryAsync<T>(proced
ure, arguments, commandType:
CommandType.StoredProcedure);

            return entities;
        }

        /// <inheritdoc
cref="ISqlDataAccess.QueryMultipl
eAsync(string, IEnumerable{Type},
object)"/>
        public async
Task<Dictionary<Type,
IEnumerable<object>>>
QueryMultipleAsync(string
procedure, IEnumerable<Type>
types, object arguments = null)
        {
            var entities = new
Dictionary<Type,
IEnumerable<object>>();

            var multi = await
connection.QueryMultipleAsync(pro
cedure, arguments, commandType:
CommandType.StoredProcedure);
            try
            {
                foreach (var item in types)
                {
                    entities.Add(item,
multi.Read(item));
                }
            }
            catch (Exception)
            {
                throw new
InvalidOperationException("Data
cannot be read!");
            }

            return entities;
        }

        /// <inheritdoc
cref="IDisposable.Dispose"/>
        public void Dispose()
        {
            Dispose(true);
            GC.SuppressFinalize(this);
        }

        /// <summary>
        /// Performs application-defined
tasks associated with freeing,
releasing, or resetting
        /// unmanaged resources.
        /// </summary>
        /// <param
name="disposing">Indicates
whether method has been invoked by
user or GC.</param>
        protected virtual void
Dispose(bool disposing)
        {
            if (connectionString != null)
            {
                if (disposing)
                {
                    connection.Dispose();
                }

                connectionString = null;
            }
        }
    }
}
```

BlocksDAL.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using
CryptoCurrency.Common.Models;
using
CryptoCurrency.DAL.Interfaces;
using CryptoCurrency.DAL.Models;

namespace CryptoCurrency.DAL
{
    /// <summary>
    /// Provides methods to access
blocks table in database.
    /// </summary>
    /// <remarks>
    ///
[![Flowchart](../images/BlocksDAL.
jpg)](../images/BlocksDAL.jpg)
    /// </remarks>
    public class BlocksDAL :
IBlocksDAL
    {
        private const string
SpAddBlock = "spAddBlock";
        private const string
SpGetAllBlocks =
"spGetAllBlocks";
        private readonly
ISqlDataAccess sqlDataAccess;

        /// <summary>
        /// Initializes a new instance of
the <see cref="BlocksDAL"/> class.
        /// </summary>
        /// <param
name="sqlDataAccess">Class with
methods for accessing
database.</param>
        public
BlocksDAL(ISqlDataAccess
sqlDataAccess)
        {
            this.sqlDataAccess =
sqlDataAccess;
        }

        /// <inheritdoc
cref="IBlocksDAL.GetAllBlocks(IE
numerable{Transaction})"/>
        public IEnumerable<Block>
GetAllBlocks(IEnumerable<Transac
tion> transactions)
        {
            var blockDataModelType =
typeof(BlockDataModel);
            var
blockTransactionDataModelType =
typeof(BlockTransactionDataModel)
;
            var types = new List<Type>
            {
                blockDataModelType,

blockTransactionDataModelType,
            };

            var queryResult =
sqlDataAccess.QueryMultiple(SpGet
AllBlocks, types);

            var blocksDB =
queryResult[blockDataModelType].
Cast<BlockDataModel>();
```

```
            var blockTransactionsDB =
queryResult[blockTransactionDataM
odelType]

.Cast<BlockTransactionDataModel>
()
                .OrderBy(a => a.BlockId)
                .ThenBy(a => a.Order);

            var blocks = new
List<Block>();
            var transactionsDB =
transactions.ToDictionary(key =>
key.Id);

            foreach (var blockDB in
blocksDB)
            {
                var blockTransactions =
new List<Transaction>();

                var
currentBlockTransactionsDB =
blockTransactionsDB.Where(a =>
a.BlockId == blockDB.BlockId);
                foreach (var transactionDB
in currentBlockTransactionsDB)
                {

blockTransactions.Add(transactions
DB[transactionDB.TransactionId]);
                }

                blocks.Add(new
Block(blockDB.PrevBlockHash,
blockTransactions) { Hash =
blockDB.Hash, Nonce =
(uint)blockDB.Nonce });
            }

            return blocks;
        }

        /// <inheritdoc
cref="IBlocksDAL.SaveBlock(string
, long, string, string, string, string,
decimal, string)"/>
        public void SaveBlock(string
hash, long nonce, string
previousBlockHash, string rewardId,
string rewardSender, string
rewardReceiver, decimal
rewardAmount, string
transactionsList)
        {
            object arguments = new
            {
                Hash = hash,
                Nonce = nonce,
                PreviousBlockHash =
previousBlockHash,
                RewardId = rewardId,
                RewardSender =
rewardSender,
                RewardReceiver =
rewardReceiver,
                RewardAmount =
rewardAmount,
                TransactionsList =
transactionsList,
            };

sqlDataAccess.Execute(SpAddBlock
, arguments);
        }
    }
}
```

SettingsDAL.cs

```
using System;
using System.Linq;
using
CryptoCurrency.DAL.Interfaces;
using CryptoCurrency.DAL.Models;
using
CryptoCurrency.TransactionService.
Models;

namespace CryptoCurrency.DAL
{
    /// <summary>
    /// Provides methods to access
settings table in database.
    /// </summary>
    /// <remarks>
    ///
[![Flowchart](../images/SettingsDAL
.jpg)](../images/SettingsDAL.jpg)
    /// </remarks>
    public class SettingsDAL :
ISettingsDAL
    {
        private const string
SpUpdateSetting =
"spUpdateSetting";
        private const string
SpGetSettings = "spGetSettings";

        private readonly
ISqlDataAccess sqlDataAccess;

        /// <summary>
        /// Initializes a new instance of
the <see cref="SettingsDAL"/>
class.
        /// </summary>
        /// <param
name="sqlDataAccess">Class with
methods for accessing
database.</param>
        public
SettingsDAL(ISqlDataAccess
sqlDataAccess)
        {
            this.sqlDataAccess =
sqlDataAccess;
        }

        /// <inheritdoc
cref="ISettingsDAL.GetAllSettings"
/>
        public Settings GetAllSettings()
        {
            var settingsDataModel =
sqlDataAccess.Query<SettingsData
Model>(SpGetSettings).ToDictionar
y(key => key.SettingName, value =>
value.SettingValue);
            var settings = new Settings();
            var settingType =
typeof(Settings);
            var settingsProperties =
settingType.GetProperties();
            foreach (var property in
settingsProperties)
            {

settingsDataModel.TryGetValue(pro
perty.Name, out string value);
                if (value != null)
                {

settingType.GetProperty(property.N
ame).SetValue(settings,
Convert.ChangeType(settingsDataM
odel[property.Name],
settingType.GetProperty(property.N
ame).PropertyType));
                }
            }

            return settings;
        }

        /// <inheritdoc
cref="ISettingsDAL.UpdateSetting(s
tring, string)"/>
        public void
UpdateSetting(string settingName,
string settingValue)
        {

sqlDataAccess.Execute(SpUpdateSet
ting, new { SettingName =
settingName, SettingValue =
settingValue });
        }
    }
}
```

TransactionsDAL.cs

```
using System.Collections.Generic;
using
CryptoCurrency.Common.Models;
using
CryptoCurrency.DAL.Interfaces;

namespace CryptoCurrency.DAL
{
    /// <summary>
    /// Provides methods to access
transactions table in database.
    /// </summary>
    /// <remarks>
    ///
[![Flowchart](../images/Transactions
DAL.jpg)](../images/TransactionsD
AL.jpg)
    /// </remarks>
    public class TransactionsDAL :
```

```
ITransactionsDAL
    {
        private const string
SpAddTransaction =
"spAddTransaction";
        private const string
SpGetAllTransactions =
"spGetAllTransactions";
        private readonly
ISqlDataAccess sqlDataAccess;

        /// <summary>
        /// Initializes a new instance of
the <see cref="TransactionsDAL"/>
class.
        /// </summary>
        /// <param
name="sqlDataAccess">Class with
methods for accessing
database.</param>
        public
TransactionsDAL(ISqlDataAccess
sqlDataAccess)
        {
            this.sqlDataAccess =
sqlDataAccess;
        }

        /// <inheritdoc
cref="ITransactionsDAL.GetAllTran
sactions"/>
        public
IEnumerable<Transaction>
GetAllTransactions()
        {
            var transactions =
sqlDataAccess.Query<Transaction>(
SpGetAllTransactions);
            return transactions;
        }

        /// <inheritdoc
cref="ITransactionsDAL.SaveTrans
action(Transaction)"/>
        public void
SaveTransaction(Transaction
transaction)
        {
            object arguments = new
            {
                TransactionId =
transaction.Id,
                SenderWalletPublicKey =
transaction.Sender,
                ReceiverWalletPublicKey
= transaction.Receiver,

transaction.TransferAmount,
                StatusId =
(byte)transaction.Status,
            };

sqlDataAccess.Execute(SpAddTrans
action, arguments);
        }
    }
}
```

UsersDAL.cs

```
using System;
using
System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using
CryptoCurrency.Common.Mod
els;
using
CryptoCurrency.DAL.Interface
s;
using
CryptoCurrency.DAL.Models;

namespace
CryptoCurrency.DAL
{
    /// <summary>
    /// Provides methods to
access users and user's roles
tables in database.
    /// </summary>
    /// <remarks>
    ///
[![Flowchart](../images/UsersD
AL.jpg)](../images/UsersDAL.j
pg)
    /// </remarks>
    public class UsersDAL :
IUsersDAL
    {
        private const string
SpAddUser = "spAddUser";
        private const string
SpAddUserWithCredentials =
"spAddUserWithCredentials";
        private const string
SpAddRoleToUser =
"spAddRoleToUser";
        private const string
SpRemoveRoleFromUser =
"spRemoveRoleFromUser";
        private const string
SpGetAllUsers =
"spGetAllUsers";
        private const string
SpGetUserById =
"spGetUserById";
        private const string
SpGetUserByEmail =
"spGetUserByEmail";

        private readonly
ISqlDataAccess sqlDataAccess;

        /// <summary>
        /// Initializes a new
instance of the <see
cref="UsersDAL"/> class.
        /// </summary>
        /// <param
name="sqlDataAccess">Class
with methods for accessing
database.</param>
        public
UsersDAL(ISqlDataAccess
sqlDataAccess)
        {
            this.sqlDataAccess =
sqlDataAccess;
        }

        /// <inheritdoc
cref="IUsersDAL.AddRoleToU
ser(long, byte)"/>
        public bool
AddRoleToUser(long userId,
byte roleId)
        {
            if
(Enum.IsDefined(typeof(Roles)
, (int)roleId))
            {
                var arguments = new
                {
                    UserId = userId,
                    RoleId = roleId,
                };

                var result =
sqlDataAccess.Execute(SpAdd
RoleToUser, arguments);

                if (result > 0)
                {
                    return true;
                }

                return false;
            }

            throw new
ArgumentException($"Role
with ID '{roleId}' not
defined.");
        }

        /// <inheritdoc
cref="IUsersDAL.AddUser(stri
ng)"/>
        public bool
```

```
AddUser(string username)
        {
            var arguments = new
            {
                Username =
username,
            };
            var result =
sqlDataAccess.Execute(SpAdd
User, arguments);
            if (result > 0)
            {
                return true;
            }

            return false;
        }

        /// <inheritdoc
cref="IUsersDAL.AddUser(stri
ng, string, string)"/>
        public int AddUser(string
username, string
passwordHash, string salt)
        {
            var arguments = new
            {
                Username =
username,
                PasswordHash =
passwordHash,
                Salt = salt,
            };
            var result =
sqlDataAccess.Query<int>(Sp
AddUserWithCredentials,
arguments).FirstOrDefault();
            return result;
        }

        /// <inheritdoc
cref="IUsersDAL.GetAllUsers"
/>
        public
IEnumerable<User>
GetAllUsers()
        {
            var userType =
typeof(User);
            var userRolesType =
typeof(UserRoleDataModel);
            var types = new
List<Type>
            {
                userType,
                userRolesType,
            };
            var queryReult =
sqlDataAccess.QueryMultiple(
SpGetAllUsers, types);
            var users =
queryReult[userType].Cast<Us
er>();
            var userRoles =
queryReult[userRolesType].Cas
t<UserRoleDataModel>();
            foreach (var user in
users)
            {
                var roles =
userRoles.Where(ur =>
ur.UserId ==
user.UserId).Sum(ur =>
ur.RoleId);
                user.SetRoles(roles);
            }

            return users;
        }

        /// <inheritdoc
cref="IUsersDAL.GetUserById
(long)"/>
        public User
GetUserById(long userId)
        {
            var userType =
typeof(User);
            var userRolesType =
typeof(UserRoleDataModel);
            var types = new
List<Type>
            {
                userType,
                userRolesType,
            };
            var arguments = new
            {
                UserId = userId,
            };
            var queryResult =
sqlDataAccess.QueryMultiple(
SpGetUserById, types,
arguments);

            var users =
queryResult[userType].Cast<U
ser>();
            var userRoles =
queryResult[userRolesType].Ca
st<UserRoleDataModel>();
            var user =
users.FirstOrDefault();
            if (user != null)
            {
                var roles =
userRoles.Sum(ur =>
ur.RoleId);
                user.SetRoles(roles);
            }

            return user;
        }

        /// <inheritdoc
cref="IUsersDAL.RemoveRole
FromUser(long, byte)"/>
        public bool
RemoveRoleFromUser(long
userId, byte roleId)
        {
            if
(Enum.IsDefined(typeof(Roles)
, (int)roleId))
            {
                var arguments = new
                {
                    UserId = userId,
                    RoleId = roleId,
                };
                var result =
sqlDataAccess.Execute(SpRem
oveRoleFromUser, arguments);
                if (result > 0)
                {
                    return true;
                }

                return false;
            }

            throw new
ArgumentException($"Role
with ID '{roleId}' not
defined.");
        }

        /// <inheritdoc
cref="IUsersDAL.AddRoleToU
serAsync(long, byte)"/>
        public async Task<bool>
AddRoleToUserAsync(long
userId, byte roleId)
        {
            if
(Enum.IsDefined(typeof(Roles)
, (int)roleId))
            {
                var arguments = new
                {
```

```
            UserId = userId,
            RoleId = roleId,
        };

        var result = await
sqlDataAccess.ExecuteAsync(S
pAddRoleToUser, arguments);

        if (result > 0)
        {
            return true;
        }

        return false;
    }

    throw new
ArgumentException($"Role
with ID '{roleId}' not
defined.");
}

/// <inheritdoc
cref="IUsersDAL.GetUserById
Async(long)"/>
public async Task<User>
GetUserByIdAsync(long
userId)
{
    var userType =
typeof(User);
    var userRolesType =
typeof(UserRoleDataModel);
    var types = new
List<Type>
    {
        userType,
        userRolesType,
    };
    var arguments = new
    {
        UserId = userId,
    };
    var queryResult = await
sqlDataAccess.QueryMultipleA
sync(SpGetUserById, types,
arguments);

    var users =
queryResult[userType].Cast<U
ser>();
    var userRoles =
queryResult[userRolesType].Ca
st<UserRoleDataModel>();
    var user =
users.FirstOrDefault();
    if (user != null)
    {
        var roles =
userRoles.Sum(ur =>
ur.RoleId);
        user.SetRoles(roles);
    }

    return user;
}

/// <inheritdoc
cref="IUsersDAL.RemoveRole
FromUserAsync(long, byte)"/>
public async Task<bool>
RemoveRoleFromUserAsync(l
ong userId, byte roleId)
{
    if
(Enum.IsDefined(typeof(Roles)
, (int)roleId))
    {
        var arguments = new
        {
            UserId = userId,
            RoleId = roleId,
        };
        var result = await
sqlDataAccess.ExecuteAsync(S
pRemoveRoleFromUser,
arguments);
        if (result > 0)
        {
            return true;
        }

        return false;
    }

    throw new
ArgumentException($"Role
with ID '{roleId}' not
defined.");
}

/// <inheritdoc
cref="IUsersDAL.AddUserAsy
nc(string)"/>
public async Task<bool>
AddUserAsync(string
username)
{
    var arguments = new
    {
        Username =
username,
    };
    var result = await
sqlDataAccess.ExecuteAsync(S
pAddUser, arguments);
    if (result > 0)
    {
        return true;
    }

    return false;
}

/// <inheritdoc
cref="IUsersDAL.AddUserAsy
nc(string, string, string)"/>
public async Task<int>
AddUserAsync(string
username, string
passwordHash, string salt)
{
    var arguments = new
    {
        Username =
username,
        PasswordHash =
passwordHash,
        Salt = salt,
    };
    var result = (await
sqlDataAccess.QueryAsync<int
>(SpAddUserWithCredentials,
arguments)).FirstOrDefault();
    return result;
}

/// <inheritdoc
cref="IUsersDAL.GetAllUsers
Async"/>
public async
Task<IEnumerable<User>>
GetAllUsersAsync()
{
    var userType =
typeof(User);
    var userRolesType =
typeof(UserRoleDataModel);
    var types = new
List<Type>
    {
        userType,
        userRolesType,
    };

    var queryReult = await
sqlDataAccess.QueryMultipleA
sync(SpGetAllUsers, types);
    var users =
queryReult[userType].Cast<Us
er>();
```

```
            var userRoles =
queryReult[userRolesType].Cas
t<UserRoleDataModel>();
            foreach (var user in
users)
            {
                var roles =
userRoles.Where(ur =>
ur.UserId ==
user.UserId).Sum(ur =>
ur.RoleId);
                user.SetRoles(roles);
            }

            return users;
        }

        /// <inheritdoc
cref="IUsersDAL.GetUserByE
mail(string)"/>
        public User
GetUserByEmail(string email)
        {
            var userType =
typeof(User);
            var userRolesType =
typeof(UserRoleDataModel);
            var types = new
List<Type>
            {
                userType,
                userRolesType,
            };
            var arguments = new
            {
                Email = email,
            };
            var queryResult =
sqlDataAccess.QueryMultiple(
SpGetUserByEmail, types,
arguments);

            var users =
queryResult[userType].Cast<U
ser>();
            var userRoles =
queryResult[userRolesType].Ca
st<UserRoleDataModel>();
            var user =
users.FirstOrDefault();
            if (user != null)
            {
                var roles =
userRoles.Sum(ur =>
ur.RoleId);
                user.SetRoles(roles);
            }

            return user;
        }

        /// <inheritdoc
cref="IUsersDAL.GetUserByE
mailAsync(string)"/>
        public async Task<User>
GetUserByEmailAsync(string
email)
        {
            var userType =
typeof(User);
            var userRolesType =
typeof(UserRoleDataModel);
            var types = new
List<Type>
            {
                userType,
                userRolesType,
            };
            var arguments = new
            {
                Email = email,
            };
            var queryResult = await
sqlDataAccess.QueryMultipleA
sync(SpGetUserByEmail,
types, arguments);

            var users =
queryResult[userType].Cast<U
ser>();
            var userRoles =
queryResult[userRolesType].Ca
st<UserRoleDataModel>();
            var user =
users.FirstOrDefault();
            if (user != null)
            {
                var roles =
userRoles.Sum(ur =>
ur.RoleId);
                user.SetRoles(roles);
            }

            return user;
        }
    }
}
```

WalletsDAL.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using CryptoCurrency.Common.BL;
using
CryptoCurrency.Common.Models;
using
CryptoCurrency.DAL.Interfaces;

namespace CryptoCurrency.DAL
{
    /// <summary>
    /// Provides methods to access
wallets table in database.
    /// </summary>
    /// <remarks>
    ///
[![Flowchart](../images/WalletsDAL
.jpg)](../images/WalletsDAL.jpg)
    /// </remarks>
    public class WalletsDAL :
IWalletsDAL
    {
        private const string
SpAddWallet = "spAddWallet";
        private const string
SpGetAllWallets =
"spGetAllWallets";
        private const string
SpGetWalletByUserId =
"spGetWalletByUserId";

        private readonly
ISqlDataAccess sqlDataAccess;

        /// <summary>
        /// Initializes a new instance of
the <see cref="WalletsDAL"/> class.
        /// </summary>
        /// <param
name="sqlDataAccess">Class with
methods for accessing
database.</param>
        public
WalletsDAL(ISqlDataAccess
sqlDataAccess)
        {
            this.sqlDataAccess =
sqlDataAccess;
        }

        /// <inheritdoc
cref="IWalletsDAL.AddWallet(Wall
et)"/>
        public bool AddWallet(Wallet
wallet)
        {
            var arguments = new
            {
                wallet.PublicKey,
                wallet.UserId,
            };
            var result =
sqlDataAccess.Execute(SpAddWalle
t, arguments);
            if (result > 0)
            {
                return true;
            }

            return false;
```

```
        }

        /// <inheritdoc
cref="IWalletsDAL.GetAllWallets(IEnumerable{Transaction})"/>
        public IEnumerable<Wallet>
GetAllWallets(IEnumerable<Transaction> transactions)
        {
            var wallets =
sqlDataAccess.Query<Wallet>(SpGetAllWallets);
            foreach (var wallet in
wallets)
            {
                var walletTransactions =
transactions
                    .Where(t => t.Sender ==
wallet.PublicKey || (t.Receiver ==
wallet.PublicKey && t.Status ==
Status.Processed))
                    .OrderBy(t =>
t.DateLastUpdated);
                foreach (var transaction in
walletTransactions)
                {

wallet.SaveInHistory(transaction);
                }
            }

            return wallets;
        }

        /// <inheritdoc
cref="IWalletsDAL.GetWalletByUserId(long?)"/>
        public Wallet
GetWalletByUserId(long? userId)
        {
            var walletDataModelType =
typeof(Wallet);
            var
transactionDataModelType =
typeof(Transaction);
            var types = new List<Type>
            {
                walletDataModelType,

transactionDataModelType,
            };
            var arguments = new
            {
                UserId = userId,
            };
            var queryResult =
sqlDataAccess.QueryMultiple(SpGetWalletByUserId, types, arguments);

            var wallets =
queryResult[walletDataModelType].Cast<Wallet>();
            var transactions =
queryResult[transactionDataModelType].Cast<Transaction>();
            var wallet =
wallets.FirstOrDefault();
            if (wallet != null)
            {
                foreach (var transaction in
transactions)
                {

wallet.SaveInHistory(transaction);
                }
            }

            return wallet;
        }

        /// <inheritdoc
cref="IWalletsDAL.GetWalletByUserIdAsync(long?)"/>
        public async Task<Wallet>
GetWalletByUserIdAsync(long?
userId)
        {
            var walletDataModelType =
typeof(Wallet);
            var
transactionDataModelType =
typeof(Transaction);
            var types = new List<Type>
            {
                walletDataModelType,

transactionDataModelType,
            };
            var arguments = new
            {
                UserId = userId,
            };
            var queryResult = await
sqlDataAccess.QueryMultipleAsync
(SpGetWalletByUserId, types,
arguments);

            var wallets =
queryResult[walletDataModelType].
Cast<Wallet>();
            var transactions =
queryResult[transactionDataModelT
ype].Cast<Transaction>();
            var wallet =
wallets.FirstOrDefault();
            if (wallet != null)
            {
                foreach (var transaction in
transactions)
                {

wallet.SaveInHistory(transaction);
                }
            }

            return wallet;
        }
    }
}
```

Cryptocurrency.DAL.Interfaces

ISqlDataAccess.cs

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace CryptoCurrency.DAL
{
    /// <summary>
    /// An interface for
SqlDataAccess.
    /// </summary>
    /// <remarks>
    ///
[![Flowchart](../images/ISqlDataAcc
ess.jpg)](../images/ISqlDataAccess.j
pg)
    /// </remarks>
    public interface ISqlDataAccess :
IDisposable
    {
        /// <summary>
        /// Execute parameterized SQL.
        /// </summary>
        /// <param
name="procedure">The stored
procedure name to execute.</param>
        /// <param
name="arguments">The arguments
which is needed for execution stored
procedure.</param>
        /// <returns>Count of affected
rows in database.</returns>
        int Execute(string procedure,
object arguments = null);

        /// <summary>
        /// Execute parameterized SQL,
returning the data typed as T.
        /// </summary>
        /// <typeparam name="T">The
type of results to
return.</typeparam>
        /// <param
name="procedure">The stored
procedure name to execute.</param>
        /// <param
name="arguments">The arguments
which is needed for execution stored
procedure.</param>
        /// <returns>A sequence of data
of the provided type.</returns>
        IEnumerable<T>
Query<T>(string procedure, object
arguments = null);

        /// <summary>
        /// Execute parameterized SQL
that returns multiple result sets, and
access each in turn.
        /// </summary>
        /// <param
name="procedure">The stored
procedure name to execute.</param>
        /// <param name="types">The
types of objects to return.</param>
        /// <param
```

```
name="arguments">The arguments which is needed for execution stored procedure.</param>
        /// <returns>Dictionary of IEnumerable objects with corresponding types.</returns>
        Dictionary<Type, IEnumerable<object>> QueryMultiple(string procedure, IEnumerable<Type> types, object arguments = null);

        /// <summary>
        /// Execute parameterized SQL asynchronously.
        /// </summary>
        /// <param name="procedure">The stored procedure name to execute.</param>
        /// <param name="arguments">The arguments which is needed for execution stored procedure.</param>
        /// <returns>Count of affected rows in database.</returns>
        Task<int> ExecuteAsync(string procedure, object arguments = null);

        /// <summary>
        /// Execute parameterized SQL asynchronously, returning the data typed as T.
        /// </summary>
        /// <typeparam name="T">The type of results to return.</typeparam>
        /// <param name="procedure">The stored procedure name to execute.</param>
        /// <param name="arguments">The arguments which is needed for execution stored procedure.</param>
        /// <returns>A sequence of data of the provided type.</returns>
        Task<IEnumerable<T>> QueryAsync<T>(string procedure, object arguments = null);

        /// <summary>
        /// Execute parameterized SQL asynchronously that returns multiple result sets, and access each in turn.
        /// </summary>
        /// <param name="procedure">The stored procedure name to execute.</param>
        /// <param name="types">The types of objects to return.</param>
        /// <param name="arguments">The arguments which is needed for execution stored procedure.</param>
        /// <returns>Dictionary of IEnumerable objects with corresponding types.</returns>
        Task<Dictionary<Type, IEnumerable<object>>> QueryMultipleAsync(string procedure, IEnumerable<Type> types, object arguments = null);
    }
}
```

IBlocksDAL.cs

```
using System.Collections.Generic;
using CryptoCurrency.Common.Models;

namespace CryptoCurrency.DAL.Interfaces
{
    /// <summary>
    /// An interface for communication with blocks DAL.
    /// </summary>
    /// <remarks>
    /// ![Flowchart](../images/IBlocksDAL.jpg)](../images/IBlocksDAL.jpg)
    /// </remarks>
    public interface IBlocksDAL
    {
        /// <summary>
        /// Gets all blocks from database.
        /// </summary>
        /// <param name="transactions">Transactions for recreating transactions list of each block.</param>
        /// <returns>Set of blocks converted to a system-readable type.</returns>
        IEnumerable<Block> GetAllBlocks(IEnumerable<Transaction> transactions);

        /// <summary>
        /// Adds new valid block to the database.
        /// </summary>
        /// <param name="hash">Hash of the new valid block.</param>
        /// <param name="nonce">Nonce of the new valid block.</param>
        /// <param name="previousBlockHash">Hash of last valid block in the system, used for calculating hash of a the new block.</param>
        /// <param name="rewardId">Id of reward transaction for miner who calculated this block.</param>
        /// <param name="rewardSender">System wallet public key.</param>
        /// <param name="rewardReceiver">Public key of the miner's wallet.</param>
        /// <param name="rewardAmount">Reward amount.</param>
        /// <param name="transactionsList">List of transaction IDs of this block, converted to single string.</param>
        void SaveBlock(string hash, long nonce, string previousBlockHash, string rewardId, string rewardSender, string rewardReceiver, decimal rewardAmount, string transactionsList);
    }
}
```

ISettingsDAL.cs

```
using CryptoCurrency.TransactionService.Models;

namespace CryptoCurrency.DAL.Interfaces
{
    /// <summary>
    /// An interface for SettingsDAL.
    /// </summary>
    /// <remarks>
    /// ![Flowchart](../images/ISettingsDAL.jpg)](../images/ISettingsDAL.jpg)
    /// </remarks>
    public interface ISettingsDAL
    {
        /// <summary>
        /// Gets current settings from database.
        /// </summary>
        /// <returns>Settings object with properties from database.</returns>
        Settings GetAllSettings();

        /// <summary>
        /// Updates setting value by name in database.
        /// </summary>
        /// <param name="settingName">Setting name in database.</param>
        /// <param name="settingValue">New setting value.</param>
        void UpdateSetting(string settingName, string settingValue);
    }
}
```

ITransactionsDAL.cs

```
using System.Collections.Generic;
using CryptoCurrency.Common.Models;

namespace CryptoCurrency.DAL.Interfaces
{
    /// <summary>
    /// An interface for
```

```
communication with transactions
DAL.
    /// </summary>
    /// <remarks>
    ///
[![Flowchart](../images/ITransaction
sDAL.jpg)](../images/ITransactions
DAL.jpg)
    /// </remarks>
    public interface
ITransactionsDAL
    {
        /// <summary>
        /// Gets all transactions from
database.
        /// </summary>
        /// <returns>Set of transactions
converted to a system-readable
type.</returns>
        IEnumerable<Transaction>
GetAllTransactions();

        /// <summary>
        /// Adds new transaction to the
database.
        /// </summary>
        /// <param
name="transaction">The transaction
prepared for save.</param>
        void
SaveTransaction(Transaction
transaction);
    }
}
```

IUsersDAL.cs

```
using System.Collections.Generic;
using System.Threading.Tasks;
using
CryptoCurrency.Common.Models;

namespace
CryptoCurrency.DAL.Interfaces
{
    /// <summary>
    /// An interface for
communication with users DAL.
    /// </summary>
    /// <remarks>
    ///
[![Flowchart](../images/IUsersDAL.j
pg)](../images/IUsersDAL.jpg)
    /// </remarks>
    public interface IUsersDAL
    {
        /// <summary>
        /// Adds new user to database.
        /// </summary>
        /// <param
name="username">User
name.</param>
        /// <returns>true - if user
created and added to database;
otherwise - false.</returns>
        bool AddUser(string
username);

        /// <summary>
        /// Adds new user with
password hash to database.
        /// </summary>
        /// <param
name="username">User
name.</param>
        /// <param
name="passwordHash">Hash of
password.</param>
        /// <param name="salt">Salt for
password.</param>
        /// <returns>User`s identifier:
>0 - if user created and added to
database; otherwise - 0.</returns>
        int AddUser(string username,
string passwordHash, string salt);

        /// <summary>
        /// Gets all users from database.
        /// </summary>
        /// <returns>Collection of
users.</returns>
        IEnumerable<User>
GetAllUsers();

        /// <summary>
        /// Gets user from database by
id.
        /// </summary>
        /// <param name="userId">User
identifier.</param>
        /// <returns>User, if exists in
database; otherwise - null.</returns>
        User GetUserById(long
userId);

        /// <summary>
        /// Gets user from database by
email.
        /// </summary>
        /// <param name="email">User
email.</param>
        /// <returns>User, if exists in
database; otherwise - null.</returns>
        User GetUserByEmail(string
email);

        /// <summary>
        /// Adds new role to user.
        /// </summary>
        /// <param name="userId">User
identifier.</param>
        /// <param name="roleId">Role
identifier.</param>
        /// <returns>true if role added to
user; otherwise false.</returns>
        bool AddRoleToUser(long
userId, byte roleId);

        /// <summary>
        /// Removes role to user.
        /// </summary>
        /// <param name="userId">User
identifier.</param>
        /// <param name="roleId">Role
identifier.</param>
        /// <returns>true if role
removed from user; otherwise
false.</returns>
        bool
RemoveRoleFromUser(long userId,
byte roleId);

        /// <summary>
        /// Adds new user to database
asynchronously.
        /// </summary>
        /// <param
name="username">User
name.</param>
        /// <returns>true - if user
created and added to database;
otherwise - false.</returns>
        Task<bool>
AddUserAsync(string username);

        /// <summary>
        /// Adds new user with
password hash to database
asynchronously.
        /// </summary>
        /// <param
name="username">User
name.</param>
        /// <param
name="passwordHash">Hash of
password.</param>
        /// <param name="salt">Salt for
password.</param>
        /// <returns>User`s identifier:
>0 - if user created and added to
database; otherwise - 0.</returns>
        Task<int>
AddUserAsync(string username,
string passwordHash, string salt);

        /// <summary>
        /// Gets all users from database
asynchronously.
        /// </summary>
        /// <returns>Collection of
users.</returns>
        Task<IEnumerable<User>>
GetAllUsersAsync();

        /// <summary>
        /// Gets user from database by
id asynchronously.
        /// </summary>
        /// <param name="userId">User
identifier.</param>
        /// <returns>User, if exists in
database; otherwise - null.</returns>
        Task<User>
GetUserByIdAsync(long userId);

        /// <summary>
        /// Gets user from database by
email asynchronously.
        /// </summary>
        /// <param name="email">User
```

```
email.</param>
        /// <returns>User, if exists in database; otherwise - null.</returns>
        Task<User> GetUserByEmailAsync(string email);

        /// <summary>
        /// Adds new role to user asynchronously.
        /// </summary>
        /// <param name="userId">User identifier.</param>
        /// <param name="roleId">Role identifier.</param>
        /// <returns>true if role added to user; otherwise false.</returns>
        Task<bool> AddRoleToUserAsync(long userId, byte roleId);

        /// <summary>
        /// Removes role to user asynchronously.
        /// </summary>
        /// <param name="userId">User identifier.</param>
        /// <param name="roleId">Role identifier.</param>
        /// <returns>true if role removed from user; otherwise false.</returns>
        Task<bool> RemoveRoleFromUserAsync(long userId, byte roleId);
    }
}
```

IWalletsDAL.cs

```
using System.Collections.Generic;
using System.Threading.Tasks;
using CryptoCurrency.Common.BL;
using CryptoCurrency.Common.Models;

namespace CryptoCurrency.DAL.Interfaces
{
    /// <summary>
    /// An interface for communication with wallets DAL.
    /// </summary>
    /// <remarks>
    /// [![Flowchart](../images/IWalletsDAL.jpg)](../images/IWalletsDAL.jpg)
    /// </remarks>
    public interface IWalletsDAL
    {
        /// <summary>
        /// Gets all wallets from database.
        /// </summary>
        /// <param name="transactions">Transactions for recreating wallet's history.</param>
        /// <returns>Set of wallets converted to a system-readable type.</returns>
        IEnumerable<Wallet> GetAllWallets(IEnumerable<Transaction> transactions);

        /// <summary>
        /// Adds new wallet to the database.
        /// </summary>
        /// <param name="wallet">New wallet.</param>
        /// <returns>true - if wallet created and added to database; otherwise - false.</returns>
        bool AddWallet(Wallet wallet);

        /// <summary>
        /// Gets specific wallet by user ID from database.
        /// </summary>
        /// <param name="userId">Wallet's owner.</param>
        /// <returns>Wallet owned by the user, if one exists.</returns>
        Wallet GetWalletByUserId(long? userId);

        /// <summary>
        /// Gets specific wallet by user ID from database asynchronously.
        /// </summary>
        /// <param name="userId">Wallet's owner.</param>
        /// <returns>Wallet owned by the user, if one exists.</returns>
        Task<Wallet> GetWalletByUserIdAsync(long? userId);
    }
}
```

Cryptocurrency.DAL.Models

BlockDataModel.cs

```
namespace CryptoCurrency.DAL.Models
{
    /// <summary>
    /// Class for representation of blocks table in the database.
    /// </summary>
    /// <remarks>
    /// [![Flowchart](../images/BlockDataModel.jpg)](../images/BlockDataModel.jpg)
    /// </remarks>
    public class BlockDataModel
    {
        /// <summary>
        /// Gets or sets block id.
        /// </summary>
        public int BlockId { get; set; }

        /// <summary>
        /// Gets or sets the block`s hash.
        /// </summary>
        public string Hash { get; set; }

        /// <summary>
        /// Gets or sets the block`s nonce.
        /// </summary>
        public long Nonce { get; set; }

        /// <summary>
        /// Gets or sets the previous block hash in the chain.
        /// </summary>
        public string PrevBlockHash { get; set; }
    }
}
```

BlockTransactionDataModel.cs

```
namespace CryptoCurrency.DAL.Models
{
    /// <summary>
    /// Class for representation of block transactions table in the database.
    /// </summary>
    /// <remarks>
    /// [![Flowchart](../images/BlockTransactionDataModel.jpg)](../images/BlockTransactionDataModel.jpg)
    /// </remarks>
    public class BlockTransactionDataModel
    {
        /// <summary>
        /// Gets or sets order in which transactions should be stored in block.
        /// </summary>
        public int Order { get; set; }

        /// <summary>
        /// Gets or sets block that owns transactions.
        /// </summary>
        public int BlockId { get; set; }

        /// <summary>
        /// Gets or sets transactions for blocks.
        /// </summary>
        public string TransactionId { get; set; }
    }
}
```

SettingsDataModel.cs

```
namespace CryptoCurrency.DAL.Models
{
    /// <summary>
    /// Class for representation of settings table in the database.
    /// </summary>
```

```
    /// <remarks>
    ///
[![Flowchart](../images/SettingsData
Model.jpg)](../images/SettingsData
Model.jpg)
    /// </remarks>
    public class SettingsDataModel
    {
        /// <summary>
        /// Gets or sets setting name
which represents property name in
the real setting model.
        /// </summary>
        public string SettingName {
get; set; }

        /// <summary>
        /// Gets or sets setting value
which represents property value in
the real setting model.
        /// </summary>
        public string SettingValue {
get; set; }
    }
}
```

UserRoleDataModel.cs

```
namespace
CryptoCurrency.DAL.Models
{
    /// <summary>
    /// Class for representation of user
roles table in the database.
    /// </summary>
    /// <remarks>
    ///
[![Flowchart](../images/UserRoleDat
aModel.jpg)](../images/UserRoleDat
aModel.jpg)
    /// </remarks>
    public class UserRoleDataModel
    {
        /// <summary>
        /// Gets or sets UserId.
        /// </summary>
        public long UserId { get; set; }

        /// <summary>
        /// Gets or sets RoleId.
        /// </summary>
        public byte RoleId { get; set; }
    }
}
```

Cryptocurrency.Common.BL

BlockCalculationUtils.cs

```
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using
System.Security.Cryptography;
using System.Text;
using
CryptoCurrency.Common.Interfaces
;
using
CryptoCurrency.Common.Models;

namespace
CryptoCurrency.Common.BL
{
    /// <summary>
    /// Class for block calculation utils
representation.
    /// </summary>
    /// <remarks>
    ///
[![Flowchart](../images/BlockCalcul
ationUtils.jpg)](../images/BlockCalc
ulationUtils.jpg)
    /// </remarks>
    public class BlockCalculationUtils
: IBlockCalculationUtils
    {
        /// <inheritdoc
cref="IBlockCalculationUtils.Valida
teHash(string, string)"/>
        public bool ValidateHash(string
blockHash, string condition)
        {
            if (blockHash != null)
            {
                return
blockHash.StartsWith(condition);
            }

            return false;
        }

        /// <inheritdoc
cref="IBlockCalculationUtils.Calcul
ateHash(string, string, uint)"/>
        public string
CalculateHash(string
previousBlockHash, string
transactionsHash, uint nonce)
        {
            return
Calculate($"{previousBlockHash}{t
ransactionsHash}{nonce}");
        }

        /// <inheritdoc
cref="IBlockCalculationUtils.Calcul
ateHash(List{Transaction})"/>
        public string
CalculateHash(List<Transaction>
transactions)
        {
            return
Calculate(string.Join(string.Empty,
transactions.Select(
                transaction =>
                {
                    return
Calculate($"{transaction.Id}{transac
tion.Sender}{transaction.TransferA
mount.ToString("F8",
CultureInfo.InvariantCulture)}{trans
action.Receiver}");
                })));
        }

        private static string
Calculate(string data)
        {
            using (SHA256 hash =
SHA256.Create())
            {
                byte[] calculatedHash =
hash.ComputeHash(Encoding.UTF8.
GetBytes(data));
                return
System.BitConverter.ToString(calcu
latedHash).Replace("-",
string.Empty);
            }
        }
    }
}
```

Wallet.cs

```
using
System.Collections.Concurrent;
using System.Collections.Generic;
using System.Linq;
using
CryptoCurrency.Common.Interfaces
;
using
CryptoCurrency.Common.Models;
using Newtonsoft.Json;

namespace
CryptoCurrency.Common.BL
{
    /// <summary>
    /// Class for wallet representation.
    /// </summary>
    /// <remarks>
    ///
[![Flowchart](../images/Wallet.jpg)](
../images/Wallet.jpg)
    /// </remarks>
    public partial class Wallet :
WalletInfo,
IObservableService<Wallet>
    {
        private const string
WalletChangedEventName =
"WalletChanged";
        private readonly
ConcurrentDictionary<string,
Transaction> transactionHistory;

        /// <summary>
        /// Initializes a new instance of
the <see cref="Wallet"/> class with
data from database.
        /// </summary>
        /// <param
name="publicKey">Wallet's public
key.</param>
        /// <param
name="userId">Wallets
owner.</param>
        public Wallet(string publicKey,
long? userId)
            : base(publicKey, userId)
        {
```

```
        transactionHistory = new
ConcurrentDictionary<string,
Transaction>();
    }

    /// <inheritdoc
cref="IObservableService.Notify"/>
    public event NotifyDelegate
Notify;

    /// <summary>
    /// Gets the balance of wallet.
    /// </summary>
    [JsonIgnore]
    public decimal Balance
    {
        get
        {
            return
transactionHistory.Sum(a =>
a.Value.Sender == PublicKey ? -
a.Value.TransferAmount :
a.Value.TransferAmount);
        }
    }

    /// <summary>
    /// Gets transaction history.
    /// </summary>
    [JsonIgnore]
    public
IEnumerable<Transaction> History
    {
        get
        {
            return
transactionHistory.Values;
        }
    }

    /// <summary>
    /// Save transaction in the
wallet's transaction history.
    /// </summary>
    /// <param
name="transaction">Transaction to
save.</param>
    public void
SaveInHistory(Transaction
transaction)
    {
        if (transaction != null)
        {

transactionHistory[transaction.Id] =
transaction;
            var eventArgs = new
Dictionary<string, string> { {
"userid", UserId.ToString() } };
            var notificationArgs = new
ObserverNotificationArgs() {
EventName =
WalletChangedEventName,
EventArgs = eventArgs };

Notify?.Invoke(notificationArgs);
        }
    }

    /// <summary>
    /// Checks if the transaction
history contains the transaction.
    /// </summary>
    /// <param
name="id">Transaction
ID.</param>
    /// <returns>true if transaction
is found; else, false.</returns>
    public bool
ContainsInHistory(string id)
    {
        return
transactionHistory.ContainsKey(id);
    }
  }
}
```

HttpHelper.cs

```
  /// <summary>
  /// Class to provide functionality
for communication though HTTP.
  /// </summary>
  /// <remarks>
  ///
[![Flowchart](../images/HttpHelper.j
pg)](../images/HttpHelper.jpg)
  /// </remarks>
  public class HttpHelper :
IHttpHelper
  {
    private readonly
IHttpClientWrapper httpClient;
    private readonly
ILogger<HttpHelper> logger;

    /// <summary>
    /// Initializes a new instance of
the <see cref="HttpHelper"/> class.
    /// </summary>
    /// <param
name="httpClient">HTTP client for
sending requests.</param>
    /// <param
name="logger">Interface for
logging.</param>
    public
HttpHelper(IHttpClientWrapper
httpClient, ILogger<HttpHelper>
logger)
    {
      this.httpClient = httpClient;
      this.logger = logger;
    }

    /// <inheritdoc
cref="IHttpHelper.GetAsync{TRequ
estModel,
TResponseModel}(TRequestModel,
string)"/>
    public async
Task<ResponseResult<TResponseM
odel>> GetAsync<TRequestModel,
TResponseModel>(TRequestModel
model, string endpoint)
    {

ResponseResult<TResponseModel>
responseResult;
      try
      {
        var queryString =
GetQueryString(model);
        var requestResponse =
await httpClient.GetAsync(endpoint
+ queryString);
        var requestResult = await
requestResponse.EnsureSuccessStat
usCode().Content.ReadAsStringAsy
nc();
        responseResult =
JsonConvert.DeserializeObject<Res
ponseResult<TResponseModel>>(re
questResult);
      }
      catch (Exception e)
      {
        logger.LogError(e,
"Something went wrong during
HTTP request");
        responseResult = new
ResponseResult<TResponseModel>(
)
        {
          Message =
ResponseMessages.FailedRequest,
          IsSuccessful = false,
        };
      }

      return responseResult;
    }

    /// <inheritdoc
cref="IHttpHelper.PostAsync{TReq
uestModel,
TResponseModel}(TRequestModel,
string)"/>
    public async
Task<ResponseResult<TResponseM
odel>> PostAsync<TRequestModel,
TResponseModel>(TRequestModel
model, string endpoint)
    {

ResponseResult<TResponseModel>
responseResult;
      try
      {
        var jsonData =
JsonConvert.SerializeObject(model);
        var content = new
StringContent(jsonData,
Encoding.UTF8, "application/json");
        var requestResponse =
await
httpClient.PostAsync(endpoint,
content);
        var requestResult = await
requestResponse.EnsureSuccessStat
```

```
usCode().Content.ReadAsStringAsy
nc();
            responseResult =
JsonConvert.DeserializeObject<Res
ponseResult<TResponseModel>>(re
questResult);
        }
        catch (Exception e)
        {
            logger.LogError(e,
"Something went wrong during
HTTP request");
            responseResult = new
ResponseResult<TResponseModel>(
)
            {
                Message =
ResponseMessages.FailedRequest,
                IsSuccessful = false,
            };
        }

        return responseResult;
    }

    /// <inheritdoc
cref="IHttpHelper.PostAsync{TReq
uestModel}(TRequestModel,
string)"/>
    /// <exception
cref="HttpRequestException">The
request failed due to an underlying
issue such as network connectivity,
DNS failure, server certificate
validation or timeout.</exception>
    public async
Task<ResponseResult>
PostAsync<TRequestModel>(TRequ
estModel model, string endpoint)
    {
        ResponseResult
responseResult;
        try
        {
            var jsonData =
JsonConvert.SerializeObject(model);
            var content = new
StringContent(jsonData,
Encoding.UTF8, "application/json");
            var requestResponse =
await
httpClient.PostAsync(endpoint,
content);
            var requestResult = await
requestResponse.EnsureSuccessStat
usCode().Content.ReadAsStringAsy
nc();
            responseResult =
JsonConvert.DeserializeObject<Res
ponseResult>(requestResult);
        }
        catch (Exception ex)
        {
            if (ex.InnerException is
SocketException)
            {
                logger.LogWarning(ex.Message);
            }
            else
            {
                logger.LogError(ex,
"Something went wrong during
HTTP request");
            }

            responseResult = new
ResponseResult()
            {
                Message =
ResponseMessages.FailedRequest,
                IsSuccessful = false,
            };
        }

        return responseResult;
    }

    private string
GetQueryString<T>(T obj)
    {
        var properties =
typeof(T).GetProperties()
            .Where(p =>
p.GetValue(obj) != null)
            .Select(p => p.Name + "="
+
HttpUtility.UrlEncode(p.GetValue(o
bj).ToString()));

        return '?' + string.Join("&",
properties);
    }
  }
}
```

HttpClientWrapper.cs

```
using
System.Diagnostics.CodeAnalysis;
using System.Net.Http;
using System.Threading.Tasks;
using
CryptoCurrency.Common.Interfaces
;

namespace
CryptoCurrency.Common.BL
{
  /// <summary>
  /// A wrapper for HttpClient.
  /// </summary>
  /// <remarks>
  ///
[![Flowchart](../images/HttpClientW
rapper.jpg)](../images/HttpClientWra
pper.jpg)
  /// </remarks>
  [ExcludeFromCodeCoverage]
  public class HttpClientWrapper :
IHttpClientWrapper
  {
    private readonly HttpClient
httpClient;

    /// <summary>
    /// Initializes a new instance of
the <see
cref="HttpClientWrapper"/> class.
    /// </summary>
    /// <param
name="httpClient">An instance of
HttpClient to make requests through
Internet.</param>
    public
HttpClientWrapper(HttpClient
httpClient)
    {
      this.httpClient = httpClient;
    }

    /// <inheritdoc
cref="IHttpClientWrapper.GetAsync
(string)"/>
    public async
Task<HttpResponseMessage>
GetAsync(string requestUri)
    {
      return await
httpClient.GetAsync(requestUri);
    }

    /// <inheritdoc
cref="IHttpClientWrapper.PostAsyn
c(string, HttpContent)"/>
    public async
Task<HttpResponseMessage>
PostAsync(string requestUri,
HttpContent content)
    {
      return await
httpClient.PostAsync(requestUri,
content);
    }
  }
}
```

HMACHttpClientWrapper.cs

```
using System;
using System.Net.Http;
using System.Threading;
using System.Threading.Tasks;
using
CryptoCurrency.Common.Interfaces
;

namespace
CryptoCurrency.Common.BL
{
  /// <summary>
  /// A wrapper for HttpClient with
HMAC.
  /// </summary>
  /// <remarks>
  ///
[![Flowchart](../images/HMACHttp
ClientWrapper.jpg)](../images/HMA
CHttpClientWrapper.jpg)
  /// </remarks>
  public partial class
```

```
HMACHttpClientWrapper :
IHttpClientWrapper
   {
      private readonly
IHMACHashingUtils
hmacHashingUtils;
      private readonly HttpClient
httpClient;

      /// <summary>
      /// Initializes a new instance of
the <see
cref="HMACHttpClientWrapper"/>
class.
      /// </summary>
      /// <param
name="httpClient">An instance of
HttpClient to make requests through
Internet.</param>
      /// <param
name="hmacHashingUtils">Interfac
e for calculating HMAC.</param>
      public
HMACHttpClientWrapper(IHMAC
HashingUtils hmacHashingUtils,
HttpClient httpClient)
      {
         this.httpClient = httpClient;
         this.hmacHashingUtils =
hmacHashingUtils;
      }

      /// <inheritdoc
cref="IHttpClientWrapper.GetAsync
(string)"/>
      public async
Task<HttpResponseMessage>
GetAsync(string requestUri)
      {
         return await
SendAsync(requestUri,
HttpMethod.Get, null);
      }

      /// <inheritdoc
cref="IHttpClientWrapper.PostAsyn
c(string, HttpContent)"/>
      public async
Task<HttpResponseMessage>
PostAsync(string requestUri,
HttpContent content)
      {
         return await
SendAsync(requestUri,
HttpMethod.Post, content);
      }

      private async
Task<HttpResponseMessage>
SendAsync(string requestUri,
HttpMethod method, HttpContent
content)
      {
         var requestMessage = new
HttpRequestMessage()
         {
            Content = content,
            RequestUri = new
Uri(requestUri),
            Method = method,
         };
         string query =
ExtractQueryString(requestUri);
         string hmac;
         if (content != null)
         {
            hmac =
hmacHashingUtils.GetHMAC(await
content.ReadAsStringAsync() +
query);
         }
         else
         {
            hmac =
hmacHashingUtils.GetHMAC(query
);
         }


requestMessage.Headers.Add("HM
AC", hmac);
         return await
httpClient.SendAsync(requestMessa
ge, CancellationToken.None);
      }

      private string
ExtractQueryString(string uri)
      {
         if (uri.Contains("?"))
         {
            return
uri.Substring(uri.IndexOf("?"));
         }

         return string.Empty;
      }
   }
}
```

Cryptocurrency.Common.Models

Block.cs

```
using System.Collections.Generic;

namespace
CryptoCurrency.Common.Models
{
   /// <summary>
   /// Class for block representation.
   /// </summary>
   /// <remarks>
   ///
[![Flowchart](../images/Block.jpg)](.
./images/Block.jpg)
   /// </remarks>
   public class Block
   {
      /// <summary>
      /// Initializes a new instance of
the <see cref="Block"/> class.
      /// </summary>
      /// <param
name="previousBlockHash">Hash
of previous block in the
blockchain.</param>
      /// <param
name="transactions">List of
transactions from the pool of
unprocessed transactions.</param>
      /// <param
name="userId">Block
creator.</param>
      public Block(string
previousBlockHash,
List<Transaction> transactions)
      {
         Hash = string.Empty;
         PreviousBlockHash =
previousBlockHash;
         Transactions = transactions;
         Nonce = 0;
      }

      /// <summary>
      /// Gets or sets the block`s hash.
      /// </summary>
      public string Hash { get; set; }

      /// <summary>
      /// Gets or sets the block`s
nonce.
      /// </summary>
      public uint Nonce { get; set; }

      /// <summary>
      /// Gets the previous block hash
in the chain.
      /// </summary>
      public string
PreviousBlockHash { get; }

      /// <summary>
      /// Gets the block`s transactions
list.
      /// </summary>
      public List<Transaction>
Transactions { get; }

      /// <inheritdoc
cref="Equals(object)"/>
      public override bool
Equals(object obj)
      {
         if (obj == null || !(obj is
Block block))
         {
            return false;
         }

         if (block.Transactions.Count
!= this.Transactions.Count)
         {
            return false;
         }

         for (int i = 0; i <
block.Transactions.Count; i++)
```

```
        {
            if
(!Transactions[i].Equals(block.Trans
actions[i]))
            {
                return false;
            }
        }

        return
Hash.Equals(block.Hash)
            &&
Nonce.Equals(block.Nonce)
            &&
PreviousBlockHash.Equals(block.Pr
eviousBlockHash);
    }

    /// <inheritdoc
cref="GetHashCode"/>
    public override int
GetHashCode() =>
Hash.GetHashCode();
  }
}
```

Transaction.cs

```
using System;
using Newtonsoft.Json;

namespace
CryptoCurrency.Common.Models
{
  /// <summary>
  /// Class for transaction
representation.
  /// </summary>
  /// <remarks>
  ///
[![Flowchart](../images/Transaction.j
pg)](../images/Transaction.jpg)
  /// </remarks>
  public class Transaction
  {
    /// <summary>
    /// Initializes a new instance of
the <see cref="Transaction"/> class.
    /// </summary>
    /// <param
name="sender">Sender`s wallet
(transaction initiator).</param>
    /// <param
name="receiver">Receiver`s
wallet.</param>
    /// <param
name="transferAmount">How many
coins are transferred from sender to
receiver.</param>
    /// <param name="id">Unique
transaction ID.</param>
    public Transaction(string
sender, string receiver, decimal
transferAmount, string id)
    {
      Id = id;
      Sender = sender;
      Receiver = receiver;
      TransferAmount =
transferAmount;
      Status = Status.Unprocessed;
      DateLastUpdated =
DateTime.UtcNow;
    }

    /// <summary>
    /// Initializes a new instance of
the <see cref="Transaction"/> class
with data from database.
    /// </summary>
    /// <param
name="transactionId">Transaction
ID.</param>
    /// <param
name="senderWalletPublicKey">Se
nder wallet public key.</param>
    /// <param
name="receiverWalletPublicKey">R
eceiver wallet public key.</param>
    /// <param
name="transferAmount">Transfer
amount.</param>
    /// <param
name="dateLastUpdated">Date of
the last transaction update.</param>
    /// <param
name="statusId">Status of the
transaction.</param>
    public Transaction(string
transactionId, string
senderWalletPublicKey, string
receiverWalletPublicKey, decimal
transferAmount, DateTime
dateLastUpdated, byte statusId)
    {
      Id = transactionId;
      Sender =
senderWalletPublicKey;
      Receiver =
receiverWalletPublicKey;
      TransferAmount =
transferAmount;
      DateLastUpdated =
DateTime.SpecifyKind(dateLastUpd
ated, DateTimeKind.Utc);
      Status = (Status)statusId;
    }

    /// <summary>
    /// Initializes a new instance of
the <see cref="Transaction"/> class
for data transferring.
    /// </summary>
    /// <param
name="id">Transaction
ID.</param>
    /// <param
name="sender">Sender wallet
public key.</param>
    /// <param
name="receiver">Receiver wallet
public key.</param>
    /// <param
name="transferAmount">Transfer
amount.</param>
    /// <param
name="dateLastUpdated">Date of
the last transaction update.</param>
    /// <param
name="status">Status of the
transaction.</param>
    [JsonConstructor]
    public Transaction(string id,
string sender, string receiver,
decimal transferAmount, DateTime
dateLastUpdated, Status status)
    {
      Id = id;
      Sender = sender;
      Receiver = receiver;
      TransferAmount =
transferAmount;
      DateLastUpdated =
DateTime.SpecifyKind(dateLastUpd
ated, DateTimeKind.Utc);
      Status = status;
    }

    /// <summary>
    /// Gets the transactions id.
    /// </summary>
    public string Id { get; }

    /// <summary>
    /// Gets the transfer amount.
    /// </summary>
    public decimal TransferAmount
{ get; }

    /// <summary>
    /// Gets the sender`s wallet.
    /// </summary>
    public string Sender { get; }

    /// <summary>
    /// Gets the receiver`s wallet.
    /// </summary>
    public string Receiver { get; }

    /// <summary>
    /// Gets date of transaction last
update.
    /// </summary>
    public DateTime
DateLastUpdated { get; private set; }

    /// <summary>
    /// Gets the transaction status.
    /// </summary>
    public Status Status { get;
private set; }

    /// <summary>
    /// Changes transaction status to
'processed' after transaction is added
to a block in the blockchain.
    /// </summary>
    public void
ConfirmTransaction()
    {
```

```
            Status = Status.Processed;
            DateLastUpdated =
DateTime.UtcNow;
        }

        /// <inheritdoc
cref="Equals(object)"/>
        public override bool
Equals(object obj)
        {
            if (obj == null || !(obj is
Transaction transaction))
            {
                return false;
            }

            return
Id.Equals(transaction.Id)
                &&
Sender.Equals(transaction.Sender)
                &&
Receiver.Equals(transaction.Receive
r)
                &&
TransferAmount.Equals(transaction.
TransferAmount);
        }

        /// <inheritdoc
cref="GetHashCode"/>
        public override int
GetHashCode() =>
Id.GetHashCode();
    }
}
```

Status.cs

```
namespace
CryptoCurrency.Common.Models
{
    /// <summary>
    /// Transaction status.
    /// </summary>
    /// <remarks>
    ///
[![Flowchart](../images/Status.jpg)](.
./images/Status.jpg)
    /// </remarks>
    public enum Status
    {
        /// <summary>
        /// A transaction has a status of
"Unprocessed" until it is added to a
valid block in the blockchain.
        /// </summary>
        Unprocessed,

        /// <summary>
        /// A transaction has a status of
"Processed" after adding to a valid
block in the blockchain.
        /// </summary>
        Processed,
    }
}
```

User.cs

```
using Newtonsoft.Json;

namespace
CryptoCurrency.Common.Models
{
    /// <summary>
    /// Class for user representation.
    /// </summary>
    /// <remarks>
    ///
[![Flowchart](../images/User.jpg)](../
images/User.jpg)
    /// </remarks>
    public class User
    {
        /// <summary>
        /// Initializes a new instance of
the <see cref="User"/> class.
        /// </summary>
        /// <param name="userId">User
identifier.</param>
        /// <param name="roles">Roles
of user.</param>
        public User(long userId, Roles
roles)
        {
            UserId = userId;
            Roles = roles;
        }

        /// <summary>
        /// Initializes a new instance of
the <see cref="User"/> class.
        /// </summary>
        /// <param name="userId">User
identifier.</param>
        /// <param
name="username">User
name.</param>
        /// <param
name="passwordHash">Hash of
password.</param>
        /// <param name="salt">Salt for
password.</param>
        public User(long userId, string
username, string passwordHash,
string salt)
        {
            UserId = userId;
            Username = username;
            PasswordHash =
passwordHash;
            Salt = salt;
        }

        /// <summary>
        /// Gets or sets a user id
property.
        /// </summary>
        public string Username { get;
set; }

        /// <summary>
        /// Gets a user id property.
        /// </summary>
        public long UserId { get; }

        /// <summary>
        /// Gets a user roles property.
        /// </summary>
        public Roles Roles { get;
private set; }

        /// <summary>
        /// Gets a user password hash
property.
        /// </summary>
        [JsonIgnore]
        public string PasswordHash {
get; }

        /// <summary>
        /// Gets a user salt property.
        /// </summary>
        [JsonIgnore]
        public string Salt { get; }

        /// <summary>
        /// Adds new role to user.
        /// </summary>
        /// <param name="roleId">Role
identifier.</param>
        public void SetRoles(int roleId)
        {
            Roles += roleId;
        }
    }
}
```

Roles.cs

```
using System;

namespace
CryptoCurrency.Common.Models
{
    /// <summary>
    /// User roles.
    /// </summary>
    /// <remarks>
    ///
[![Flowchart](../images/Roles.jpg)](..
/images/Roles.jpg)
    /// </remarks>
    [Flags]
    public enum Roles
    {
        /// <summary>
        /// Customer user role.
        /// </summary>
        Customer = 1,

        /// <summary>
        /// Admin user role.
        /// </summary>
        Admin = 4,

        /// <summary>
        /// Miner user role.
        /// </summary>
        Miner = 2,
    }
```

Додаток Г

(обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

УДОСКОНАЛЕННЯ МЕТОДІВ ТА ЗАСОБІВ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОБРОБКИ КРИПТОВАЛЮТНИХ ТРАНЗАКЦІЙ

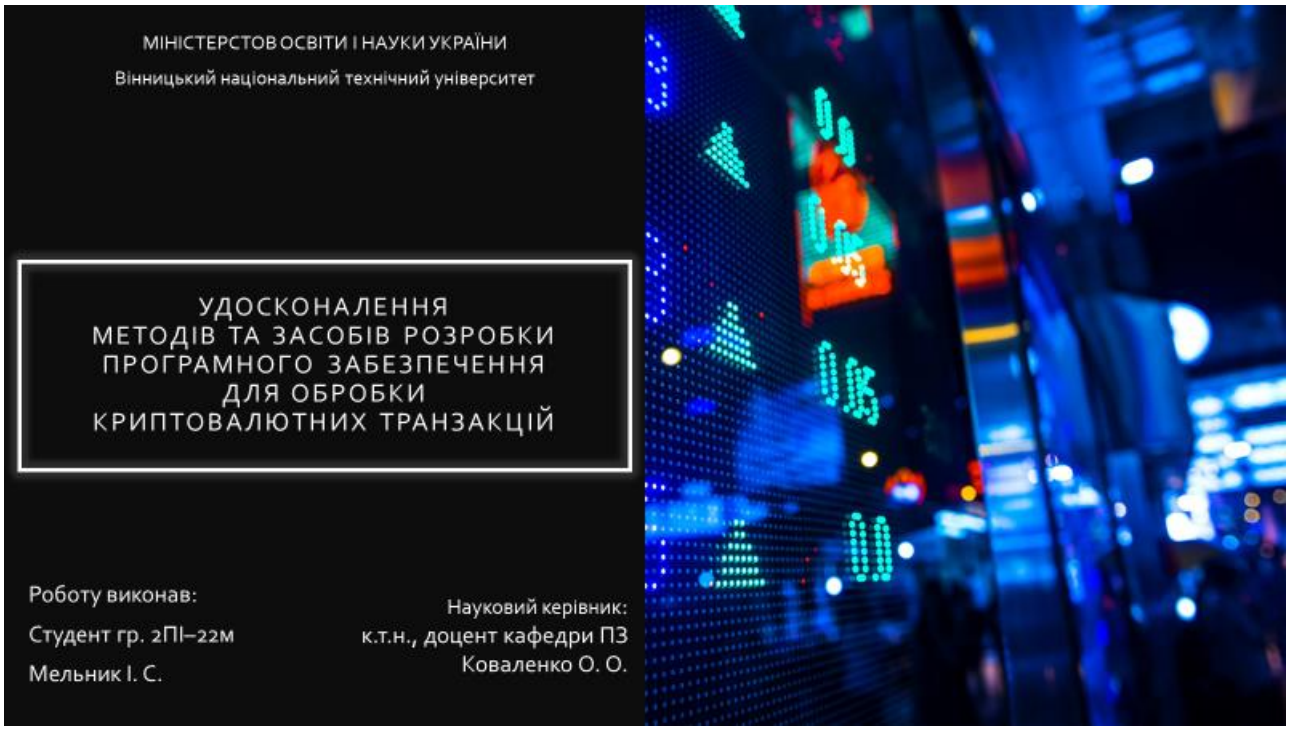

Рисунок Г.1 – Титульний слайд

Рисунок Г.2 – Мета, предмет і об’єкт роботи

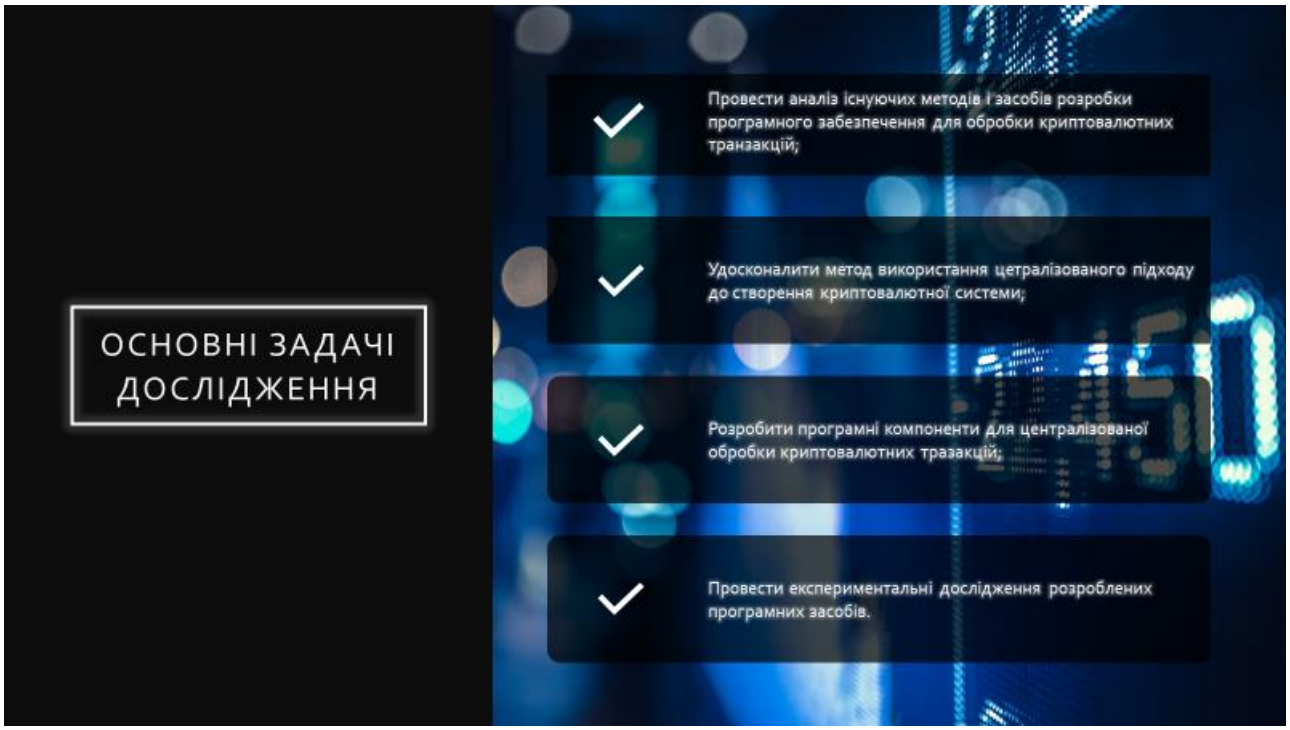

Рисунок Г.3 – Основні задачі дослідження

Рисунок Г.4 – Наукова новизна отриманих результатів

Рисунок Г.5 – Практичне значення

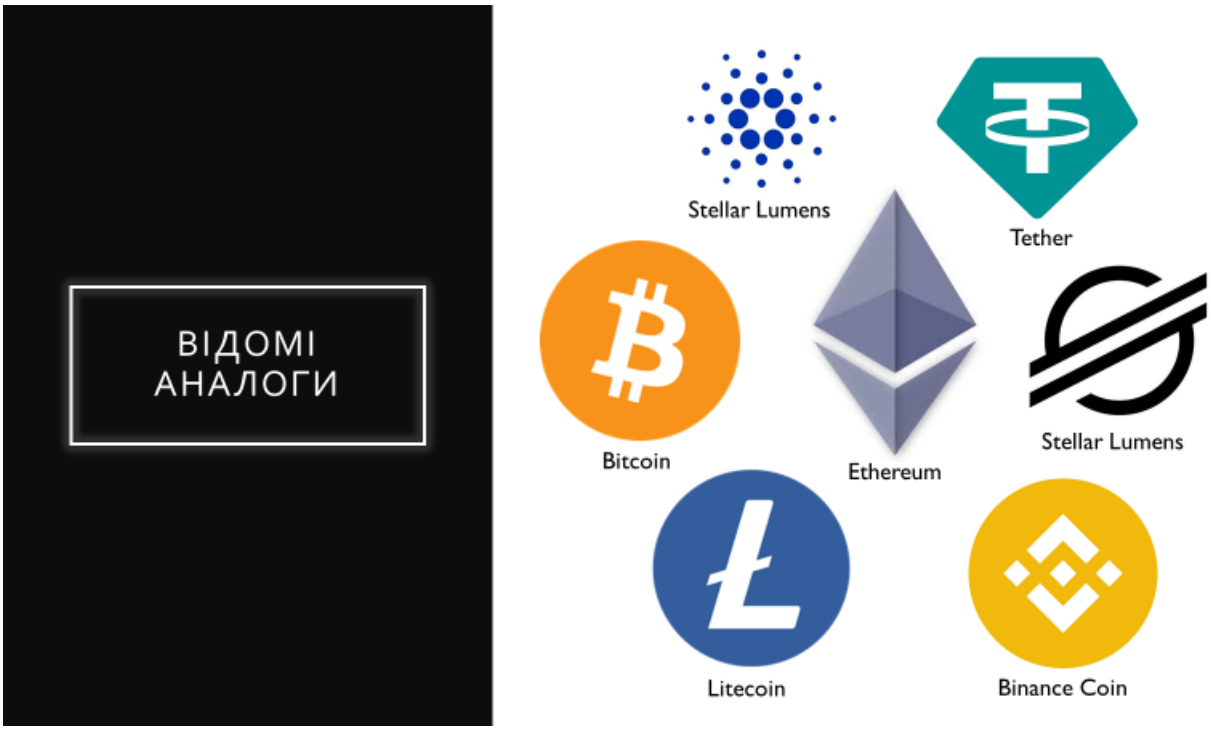

Рисунок Г.6 – Відомі аналоги

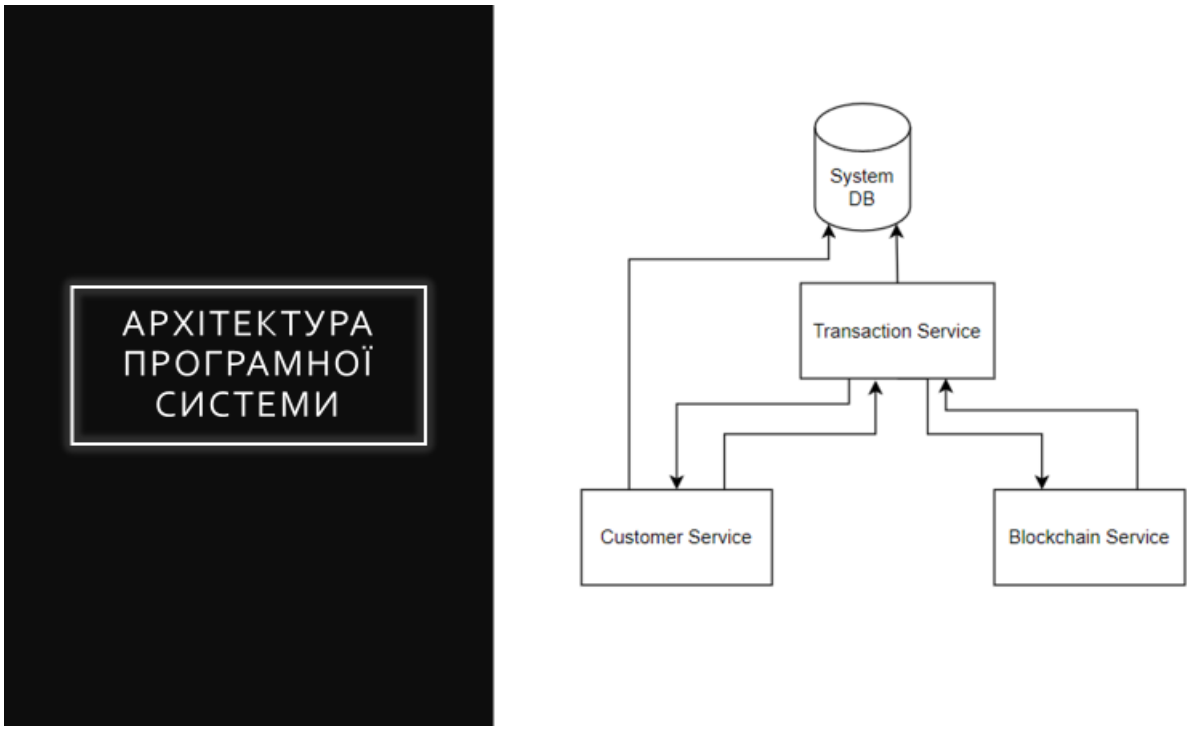

Рисунок Г.7 – Архітектура програмної системи

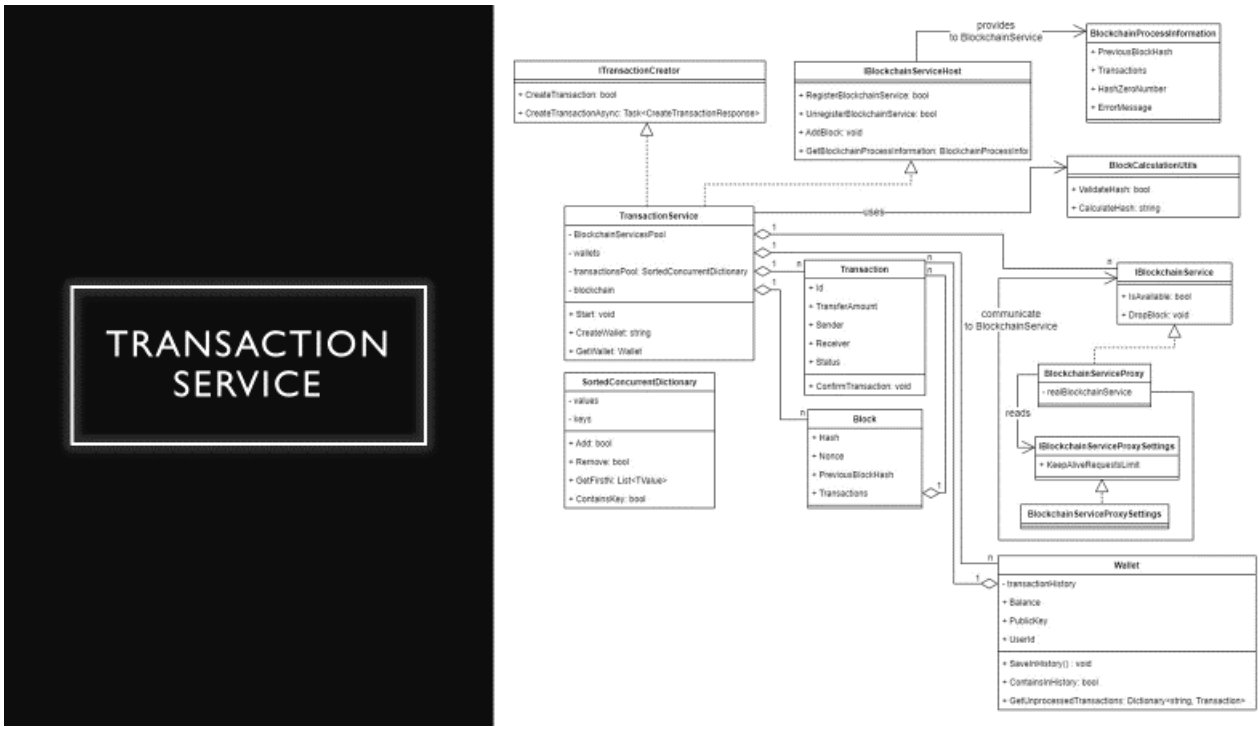

Рисунок Г.8 – Сервіс транзакцій

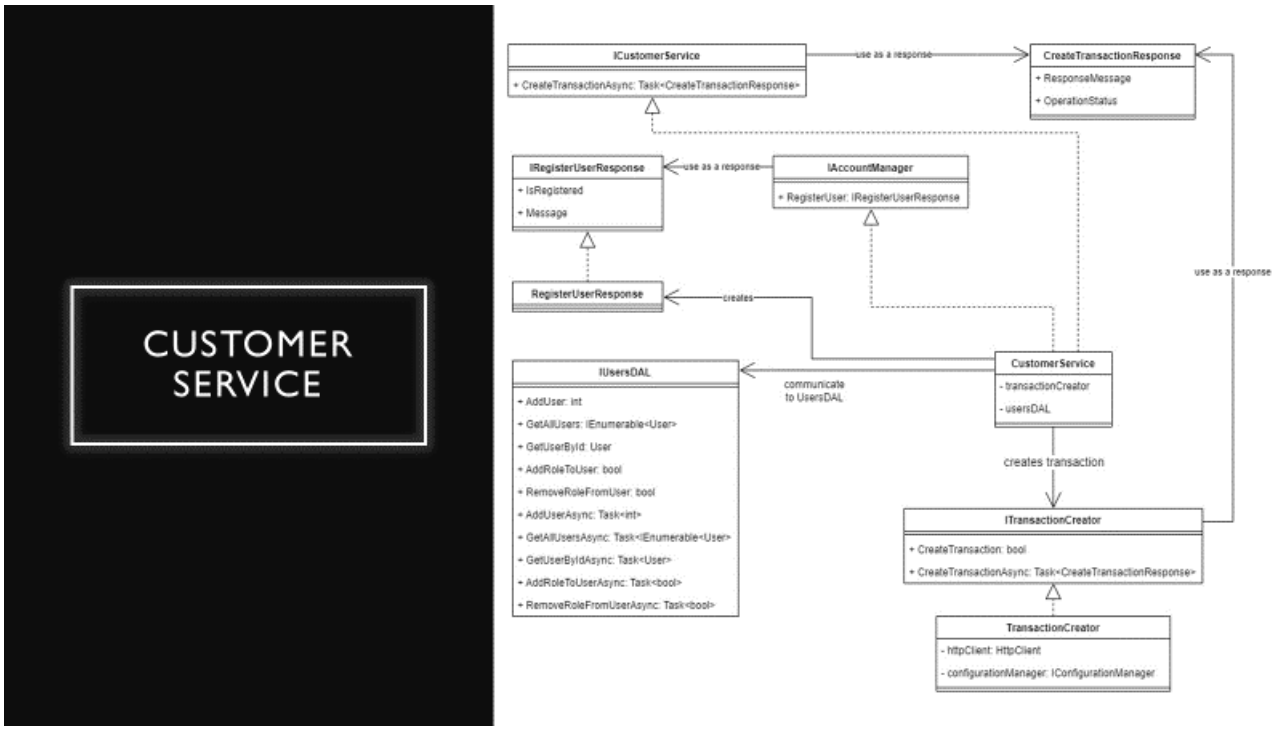

Рисунок Г.9 – Сервіс користувача

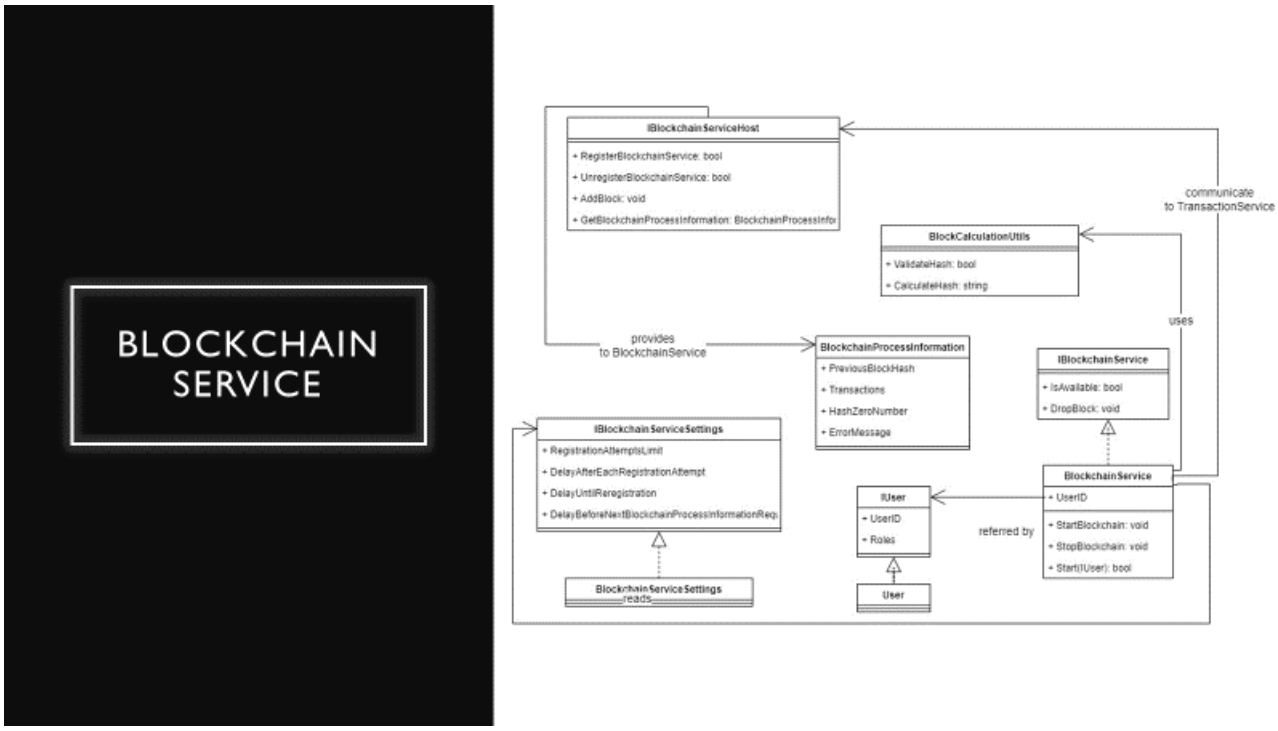

Рисунок Г.10 – Сервіс блокчейну

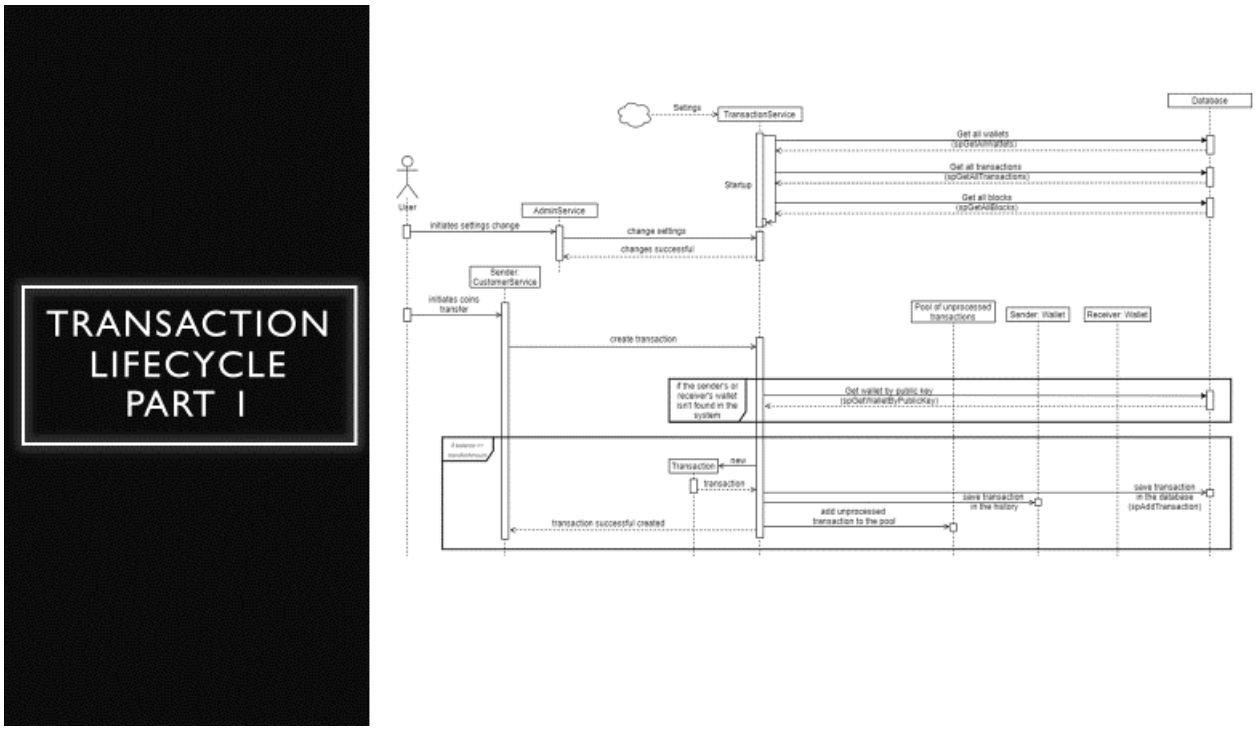

Рисунок Г.11 – Життевий цикл транзакції, частина 1

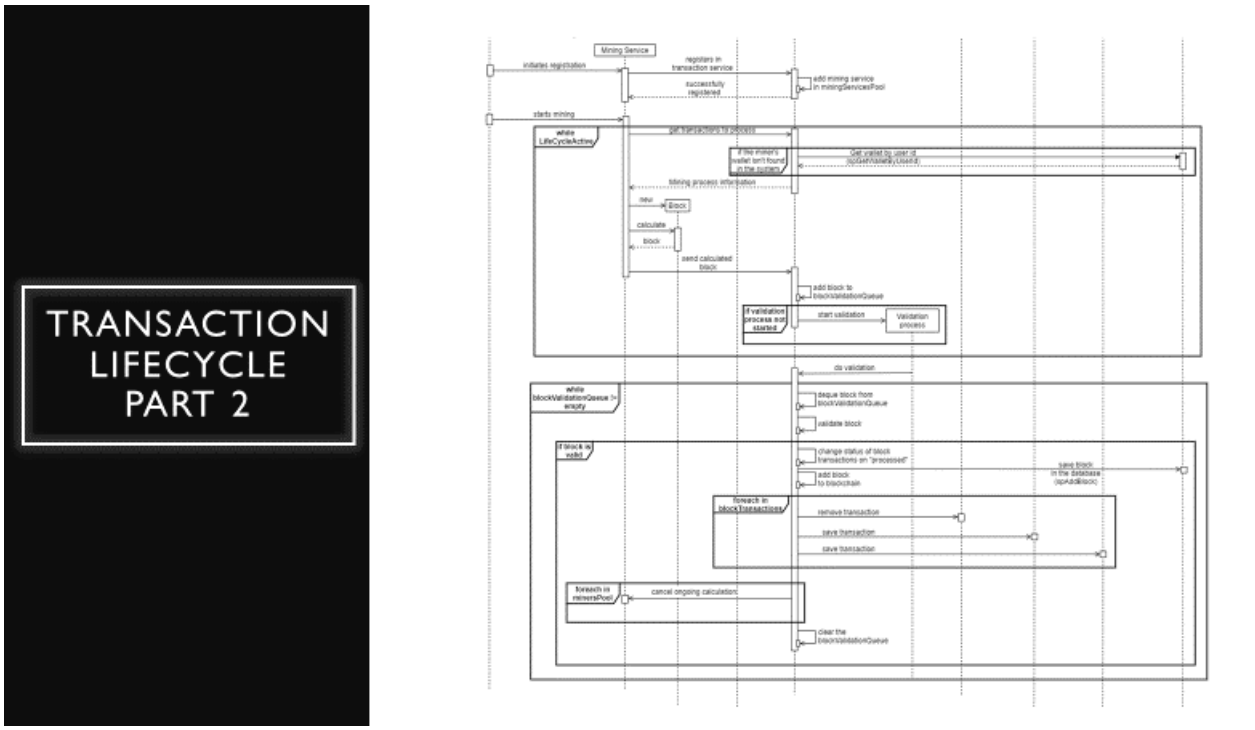

Рисунок Г.12 – Життевий цикл транзакції, частина 2

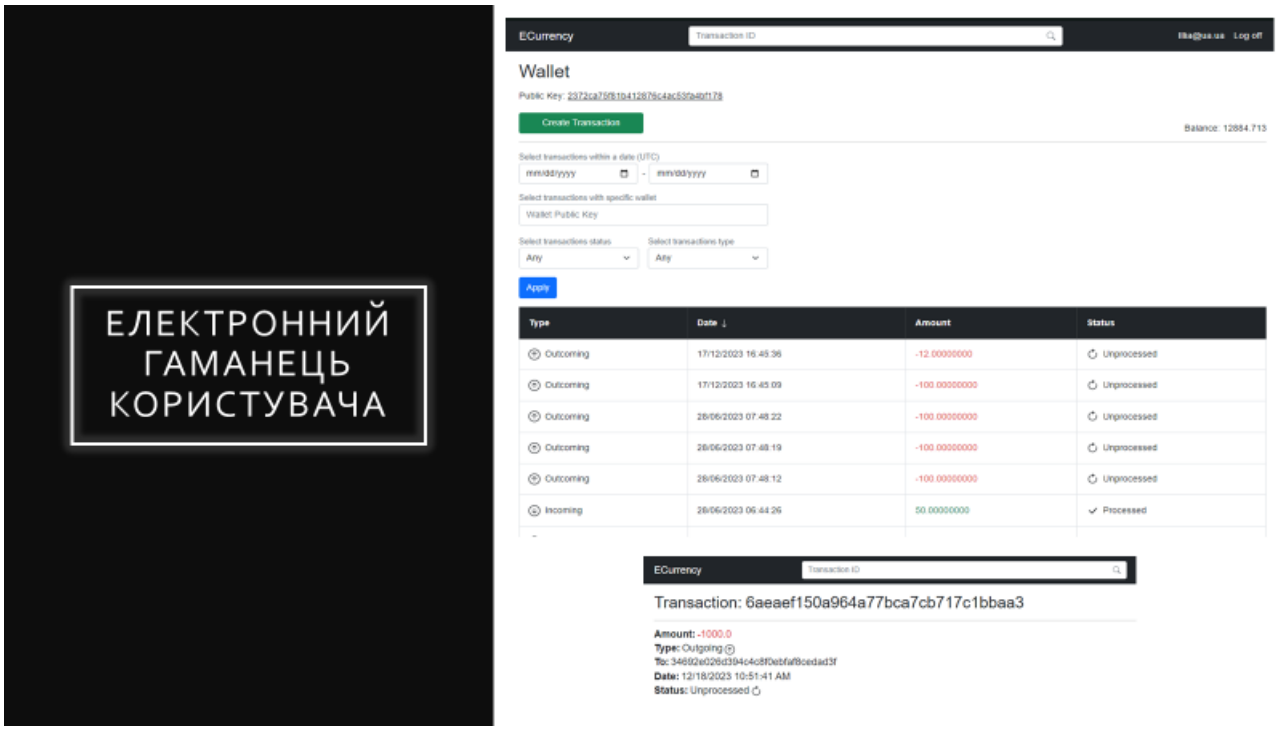

Рисунок Г.13 – Електронний гаманець користувача

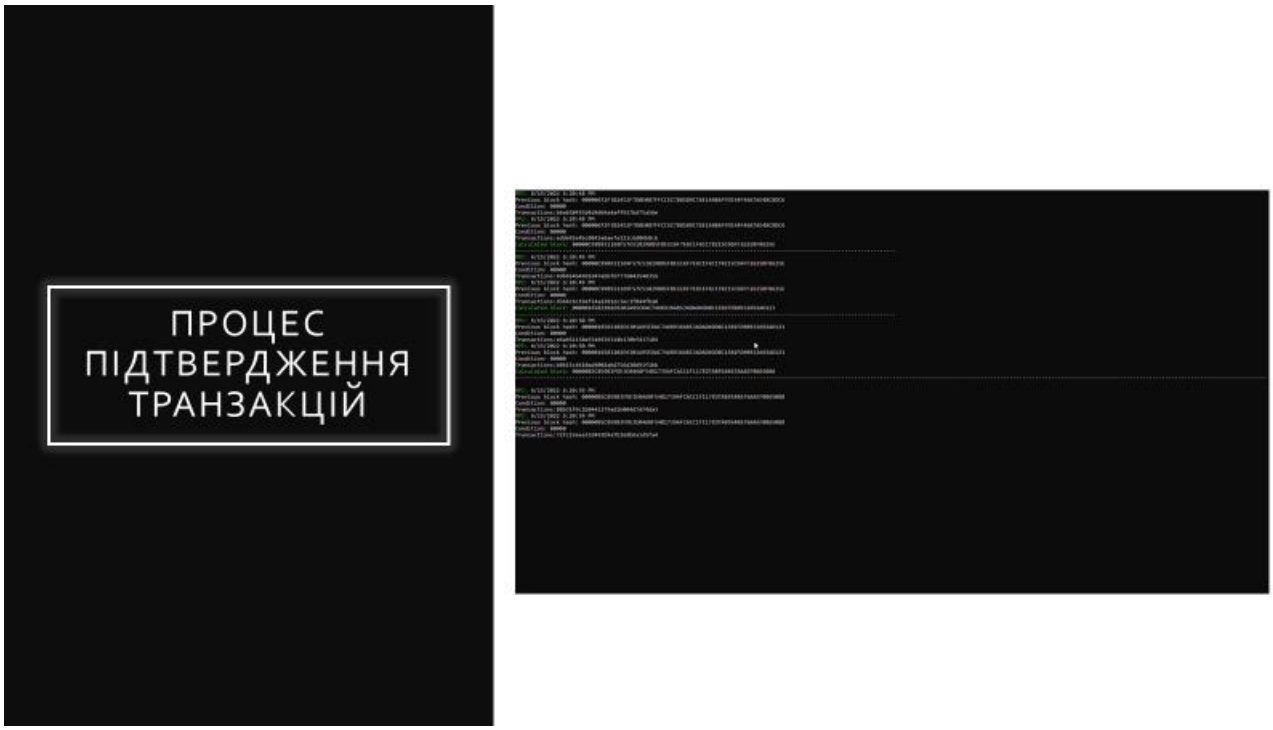

Рисунок Г.14 – Процес підтвердження транзакції

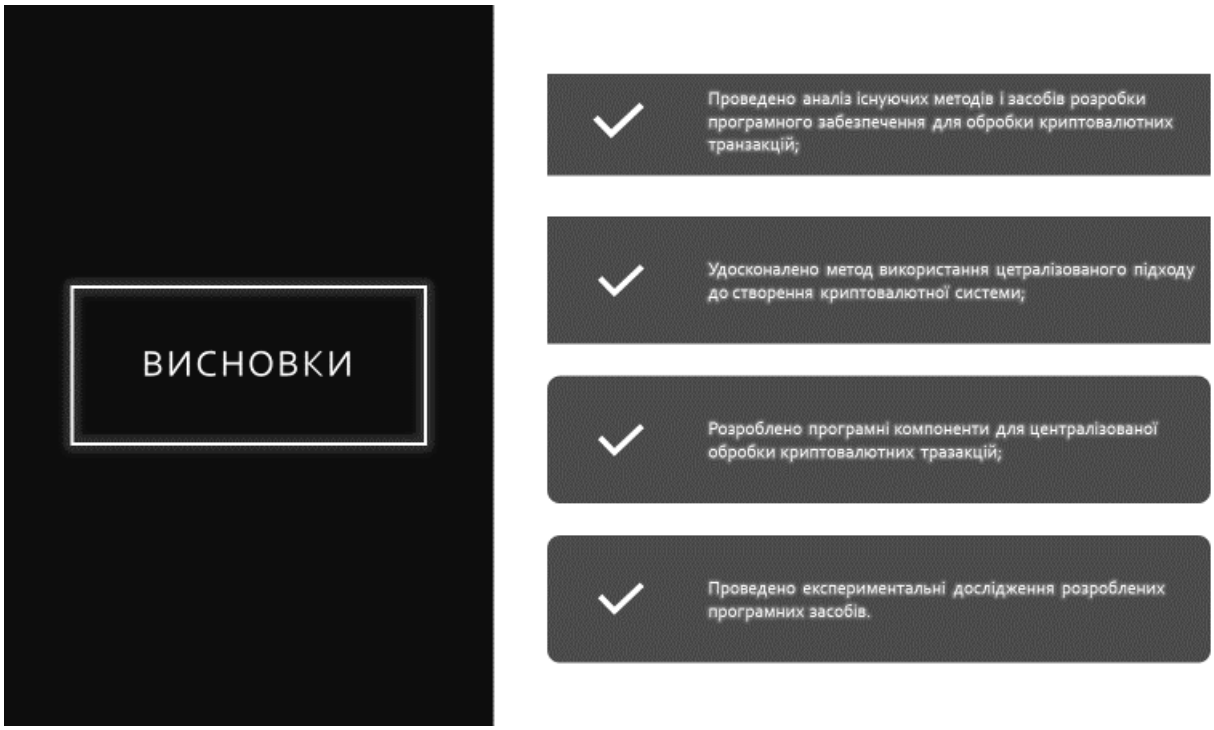

Рисунок Г.15 – Висновки

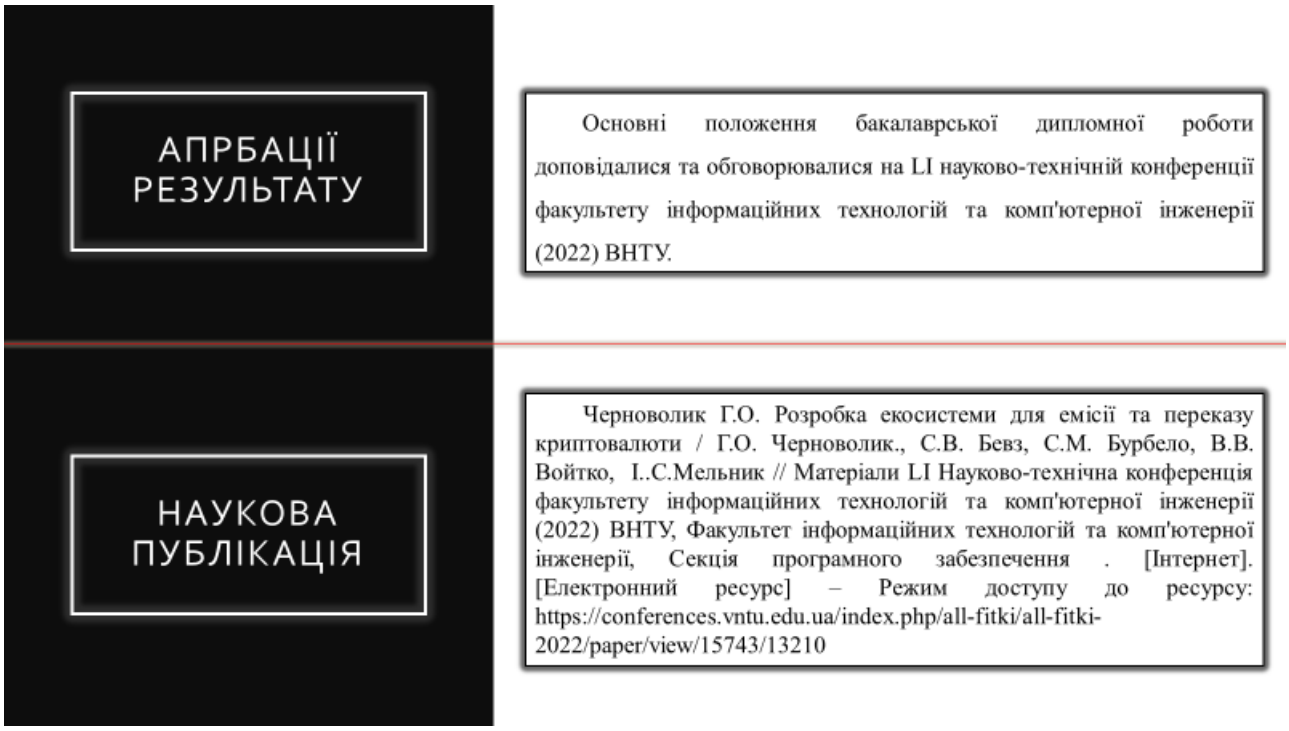

Рисунок Г.16 – Апробації та публікації

Рисунок Г.17 – Фінальний слайд