Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська дипломна робота

на тему: «Розробка програмних додатків для емісії та переказу криптовалюти»

Виконав: студент 3 курсу
групи 1ПІ-19мс2
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Мельник І.С.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ Катєльніков Д.І.
(прізвище та ініціали)

Рецензент: к.т.н., ст. вик. КН Озеранський В.С.
(прізвище та ініціали)

Допущено до захисту
Зав. кафедри _____
«____» _____ 2022 р.

ВНТУ – 2022

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

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
"_25_" __березня__2022 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Мельнику Іллі Сергійовичу

1. Тема роботи – «Розробка програмних додатків для емісії та переказу криптовалюти»

Керівник роботи: Катєльніков Денис Іванович, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 24 березня 2022 року № 66.

2. Строк подання студентом роботи 13 червня 2022 року

3. Вихідні дані до роботи: середовище розробки Visual Studio 2022, мова розробки C#, операційна система – Windows 10, система управління базою даних MS SQL Server, архітектура системи для емісії криптовалюти та взаємодії із електронними гаманцями.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз та постановка задач; розробка архітектури та алгоритмів програмного додатка; розробка програмного додатку; тестування додатку, висновки; список використаних джерел, додатки, графічна частина.

5. Перелік графічного матеріалу: графічний інтерфейс клієнтських додатків системи; модель життєвого циклу транзакції; структура системи; тестування системи.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-8	Катєльніков Д. І., к.т.н., доцент кафедри ПЗ		

7. Дата видачі завдання _____25 березня 2022 року_____

КАЛЕНДАРНИЙ ПЛАН

з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
	Аналіз стану криптовалют та вибір чітких цілей для поставленої задачі	26.03.2022 – 06.04.2022	Вик.
	Розробка архітектури системи	07.04.2021 – 14.05.2021	Вик.
	Розробка сервісу для роботи з транзакціями	15.05.2021 – 20.05.2021	Вик.
	Розробка сервісу для роботи з електронним гаманцем користувача	25.05.2021 – 31.05.2021	Вик.
	Розробка сервісу для емісії криптовалюти	6.06.2022 – 10.06.2022	Вик.

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

Керівник бакалаврської дипломної роботи _____ (підпис) Катєльніков Д.І. (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота складається зі 202 сторінок формату А4, на яких є 61 рисунок, 1 таблиця, список використаних джерел містить 23 найменування.

У бакалаврській дипломній роботі проведено детальний аналіз принципів роботи криптовалюти.

Запропоновано ідею централізованої криптовалюти, що дозволить використовувати її як офіційну платіжну одиницю у повсякденному житті за рахунок швидкого та адаптивного до навантажень процесу обробки транзакцій, що є критичним у щоденному використанні, а механізми роботи криптовалюти забезпечать цілісність на безпечність операцій.

Також запропоновано створення нового типу колекцій для збереження даних у додатку, що розширює базові можливості існуючих колекцій. Мова йде про колекцію що представляє собою пару ключ-значення та зберігає дані у порядку додання.

Розроблено алгоритми та програмні додатки для забезпечення функціонування життєвого циклу криптовалютної системи.

Отримані в бакалаврській дипломній роботі результати можна використати для виконання швидких та безпечних грошових переказів.

Ключові слова: криптовалюта, блок, транзакція, майнінг.

ABSTRACT

The bachelor's thesis consists of 202 A4 pages, which have 61 pictures, 1 table, a list of sources used contains 23 items.

In the bachelor's thesis a detailed analysis of the principles of cryptocurrency.

The idea of a centralized cryptocurrency has been proposed, which will allow it to be used as an official unit of payment in everyday life at the state level, due to the fast and load-responsive process of the transaction processing, which is critical in everyday use.

It is also proposed to create a new type of collection to store data in the application, which expands the basic capabilities of existing collections. It is a collection that is a key-value pair and stores data in the order of addition.

Algorithms and software applications have been developed to ensure the functioning of the cryptocurrency system life cycle.

The results obtained in the bachelor's thesis can be used to make fast and secure money transfers.

Key words: cryptocurrency, block, transaction, mining.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. На сьогоднішній день цифровізація зачіпає навіть найвіддаленіші куточки світу. І час, коли світ повністю перейде на цифрову валюту не так далеко. Вже зараз існує чимало варіантів реалізації криптовалют. Але все ж це тільки початок, тому реалізація екосистеми для функціонування криптовалюти є актуальним питанням.

Поняття "криптовалюта" [1] відноситься до цифрових видів валюти. Її створюють та передають за допомогою криптографічних методів, переважно на базі технології блокчейн. "Монети" випускаються спочатку в електронному вигляді буквально за рахунок проведення математичних обчислень. Простими словами, криптовалюта – це штучна платіжна система, що прирівнюється до справжніх грошей, що має офіційний курс [2].

Термін «криптовалюта» вперше став обговорюватися 2011 року, починаючи з публікації журналу Forbes. З тих пір назва міцно закріпилась і застосовується щодо одиниць валюти, що не мають виразу у вигляді паперових банкнот або монет з металу. Такий вид грошей існує виключно у цифровому просторі [3].

Перспективи розвитку криптовалют пов'язані з прогресом інтернет-технологій.

На відміну від інших електронних платіжних систем, криптовалюта спочатку з'являється без участі реальних грошей. Щоб стати власником певної суми монет, цілком достатньо підключитися до сервісу їх створення, стати учасником єдиної мережі майнінгу і дочекатися свого «заробітку». У цьому полягає ключова відмінність криптовалюти від реальних грошей.

Будь-який тип криптовалюти немає офіційного статусу як платіжного матеріалу (але деякі з країн вже визнали Bitcoin). Типові властивості як децентралізація привабливі для користувачів. Власник будь-якої з існуючих криптовалют не прив'язаний до будь-якої географічної точки, держави чи

політичного устрою. Незважаючи на прив'язку курсу до реальних грошей на зразок долара США або євро, цифрові гроші «цінні самі собою».

Віртуальні гроші набули популярності з таких причин:

- Висока поширеність, універсальність. Гаманець легко створити на будь-якому комп'ютері, смартфоні чи планшеті на різних операційних системах.
- Простота, відкритість розрахункових операцій. Повна історія вхідних та вихідних транзакцій зберігається без обмежень у часі.
- Кожен вузол системи генерації криптовалюти рівноправний, єдиного центру немає, що унеможливлює блокування гаманців, скасування та контролю платежів.
- Максимальна анонімність збільшує незалежність платіжної системи. При здійсненні платежів можна вказувати адресу, номер рахунку за бажанням власника гаманця, з якого здійснюється платіж.

Транзакції, що проводяться, захищаються криптографічним методом. Без передачі блоку із спеціальним перевірочним кодом підтвердити фінансову операцію не вдасться. Якщо ж це зроблено, ніхто не зможе скасувати переказ грошей, що виключає шахрайство при оплаті криптовалютою товарів/послуг.

Завдяки високій надійності електронних гаманців, що захищаються закритим ключем, криптовалюта може використовуватися для створення заощаджень. 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 бакалаврської дипломної роботи є підвищення стабільності платіжної системи шляхом розробки

власної централізованої криптовалюти, що дозволяє гарантувати безперервність та незалежність процесу емісії й обробки транзакцій.

Основними задачами роботи є:

– розробити моделі платіжної системи;
– розробити базу даних для зберігання інформації системи;
– розробити сервіс для обробки та валідації даних про криптовалюту;
– розробити сервіс для створення транзакцій та перегляду історії транзакцій клієнтів;
– розробити сервіс для емісії криптовалюти;
– провести тестування програмних модулів системи.

Об’єкт дослідження – процеси емісії та переказу криптовалюти.

Предмет дослідження – механізми функціонування і засоби розробки криптовалюти.

Методи дослідження. У процесі досліджень використовувались методи дослідження:

– методи побудови архітектури для створення архітектури ; системи
– методи хешування інформації для оптимізації даних;
– методи формування блоків даних для структуризації даних;
– методи зберігання конфіденційних даних для захисту інформації;
– методи прикладної теорії інформації і теорії алгоритмів для розробки алгоритмів і програмного забезпечення.
– методи тестування для перевірки працездатності системи..

Наукова новизна отриманих результатів.

- Подальшого розвитку отримала централізована модель функціонування криптовалюти, яка, на відміну від існуючих, є повністю автономною і не залежить від кількості користувачів, підключених до мережі, що, в свою чергу, дозволяє використовувати сервіс без потенційного ризику втрат власних заощаджень.

- Подальшого розвитку отримала модель збереження даних, яка, на відміну від існуючих, є потокобезпечною й орієнтованою на збереження даних типу ключ-значення у сортованому вигляді в порядку додавання, що дозволяє збільшити продуктивність та зручність роботи.

Практична цінність отриманих результатів. Практична цінність полягає у кінцевій реалізації криптовалюти, що в подальшому розвитку може бути використана як платформа для створення швидких та безпечних грошових переказів.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській дипломній роботі, отримані автором особисто. У науковій роботі [4], опублікованій у співавторстві, автору належить модель життєвого циклу транзакції.

Апробація матеріалів бакалаврської дипломної роботи. Основні положення бакалаврської дипломної роботи доповідалися та обговорювалися на LI науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженерії (2022) ВНТУ.

Публікації. Основні результати дослідження опубліковані в науковій роботі [4] – в тезах доповіді на LI науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженерії (2022) ВНТУ.

Аналіз. У пояснювальній записці до бакалаврської дипломної роботи було розглянуто 4 розділи та було використано 23 літературних джерела.

1 ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз стану криптовалют

Технологія блокчейн, що використовується під час емісії криптовалют, дозволяє створювати необмежену кількість одиниць цифрової валюти. Емісія, як і сама структура криптовалюти, 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 їй 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.

Рекомендується загострити увагу на наступні моменти:

- Збереження пароля. Несе небезпека як його 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.

1.2 Порівняльний аналіз аналогів

Популярність прийнято вважати за величиною курсу до долару США, тому розлянемо найбільші криптовалютні системи:

- Bitcoin [5]. Перший варіант криптовалюти, з'явився у 2009 році. На рисунку 1.1 зображено логотип Bitcoin.

Рисунок 1.1 – Логотип Bitcoin

- Ethereum [6]. Створення розпочалося у 2015 році. Технологія використовується для реєстрації різноманітних угод, передачі ресурсів, авторських прав, тощо. На рисунку 1.2 зображено логотип Ethereum.

Рисунок 1.2 – Логотип Ethereum

- Litecoin [7]. Вважається похідною від біткоїну. Запуск генерації відбувся у 2011 р. На рисунку 1.2 зображено логотип Litecoin.

Рисунок 1.3 – Логотип Litecoin

Список легко продовжити - Primecoin, Peercoin, DogeCoin, Namecoin і т.д., але визнається лише невелика кількість варіантів цифрової валюти. Інші часто відносять до електронних фінансових пірамід, здатних припинити існування будь-якої миті. Такий ризик існує для всіх типів криптовалют. На

практиці «зникнути» може навіть біткоїн (поки що немає законодавчо прийнятого способу врегулювання подібних ситуацій). В загальному кожна з криптовалют має спільні риси та ідеї. Єдиною різницею є алгоритми хешування, обробки даних, сферах застосування (але це більше про саму технологію блокчейн ніж про криптовалюту як таку) та правила роботи.

Результати порівняння аналогів наведено в таблиці 1.1.

Таблиця 1.1 – Порівняльні характеристика криптовалютних систем

Критерії	Bitcoin	Ethereum	Litecoin	Власна розробка
Децентралізація	+	+	+	-
Необмежена емісія	-	+	-	+
Можливість гнучкої зміни параметрів системи в залежності від поточного навантаження	-	-	-	+
Наявність власного інструменту для роботи з гаманцем	-	-	-	+
Загальна оцінка	25%	50%	25%	75%

Згідно результатів порівняння найвідоміших криптовалютних систем отримані результати вказують на доцільність розробки власної системи. Основним недоліком, якщо порівнювати з ідеями криптовалют є централізація, але в свою чергу це дозволить запровадити систему у державній та повсякденній сфері. Також власна розробка може покрити

фактор надлишковості сервісів для взаємодії с користувацьким гаманцем, оскілки система пропонує власний єдиний сервіс для захищеної роботи.

1.3 Постановка задач для розробки криптовалюти

Проаналізувавши переваги та недоліки існуючих криптовалют, було сформульовано такі задачі бакалаврської дипломної роботи:

- розробити моделі платіжної системи;
- розробити базу даних для зберігання інформації системи;
- розробити сервіс для обробки та валідації даних про криптовалюту;
- розробити сервіс для створення транзакцій та перегляду історії транзакцій клієнтів;
- розробити сервіс для емісії криптовалюти;
- провести тестування програмних модулів системи.

1.4 Висновки

У першому розділі було розглянуто поточний стан криптовалютних систем. Також порівняно відомі аналоги, такі як Bitcoin, Ethereum та Litecoin. Результат порівняння виявив, що досліджувані криптовалютні отримали свою популярність за рахунок новизни самої ідеє та на правах першостворених систем. Проаналізувавши переваги та недоліки існуючих криптовалют, було виявлено, що вони не мають місця для використання у повсякденному житті та на правах державною валюти в якості основного платіжного матеріалу, оскільки всі є децентралізованими. Таким чином було доведено доцільність розробки власного програмного рішення. На основі отриманих результатів було сформовано рішення для розробки власної централізованої криптовалюти.

2 РОЗРОБКА МОДЕЛЕЙ СИСТЕМИ І ПРОЦЕСІВ ЇЇ ФУНКЦІОНУВАННЯ

2.1 Розробка централізованої моделі криптовалюти та база даних

Ідея централізації полягає у наданні доступу до системи через єдиний шлюз та абстрагування від учасників криптовалютної мережі, оскільки для роботи централізованої мережі немає необхідності щоб користувачі підключались у якості хостів та утримували локальну копію всіх даних, що може мати досить великий розмір.

На рисунку 2.1 зображено загальну централізовану модель криптовалютної системи.

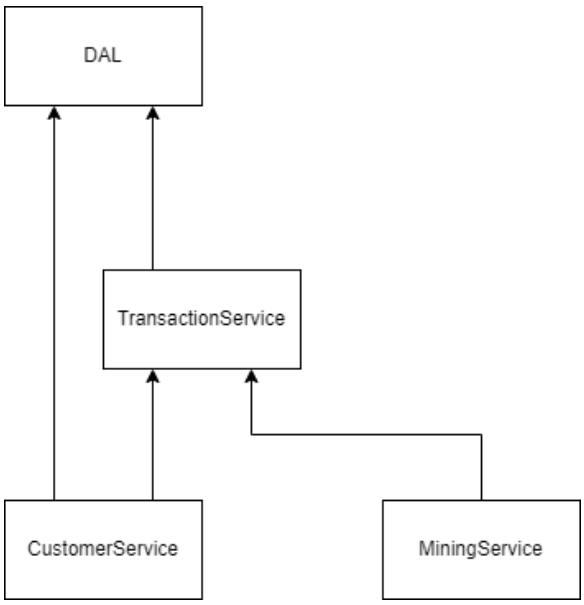

Рисунок 2.1 – Загальна централізована модель криптовалютної системи

- DAL (Data Access Layer) – шар доступу до даних, відповідає за комунікацію з джерелом даних, у даному випадку база даних MS SQL, за допомогою micro-ORM Dapper.
- TransactionService – сервіс транзакцій, веб-додаток на стороні сервера для обробки даних із клієнтських програм і забезпечення збереження даних у MS SQL за допомогою DAL.

- CustomerService – сервіс для користувачів, веб-клієнт для управління гаманцями користувачів, включаючи перегляд історії транзакцій, баланс, створення нових транзакцій.
- MiningService – консольний клієнт для емісії криптовалюти, а саме для обчислення хешів нових транзакцій і відправлення результатів на сервер для перевірки та зберігання.

Розглянемо кожен з сервісів більш детально. На рисунку 2.2 зображено класову діаграму шару доступу до даних.

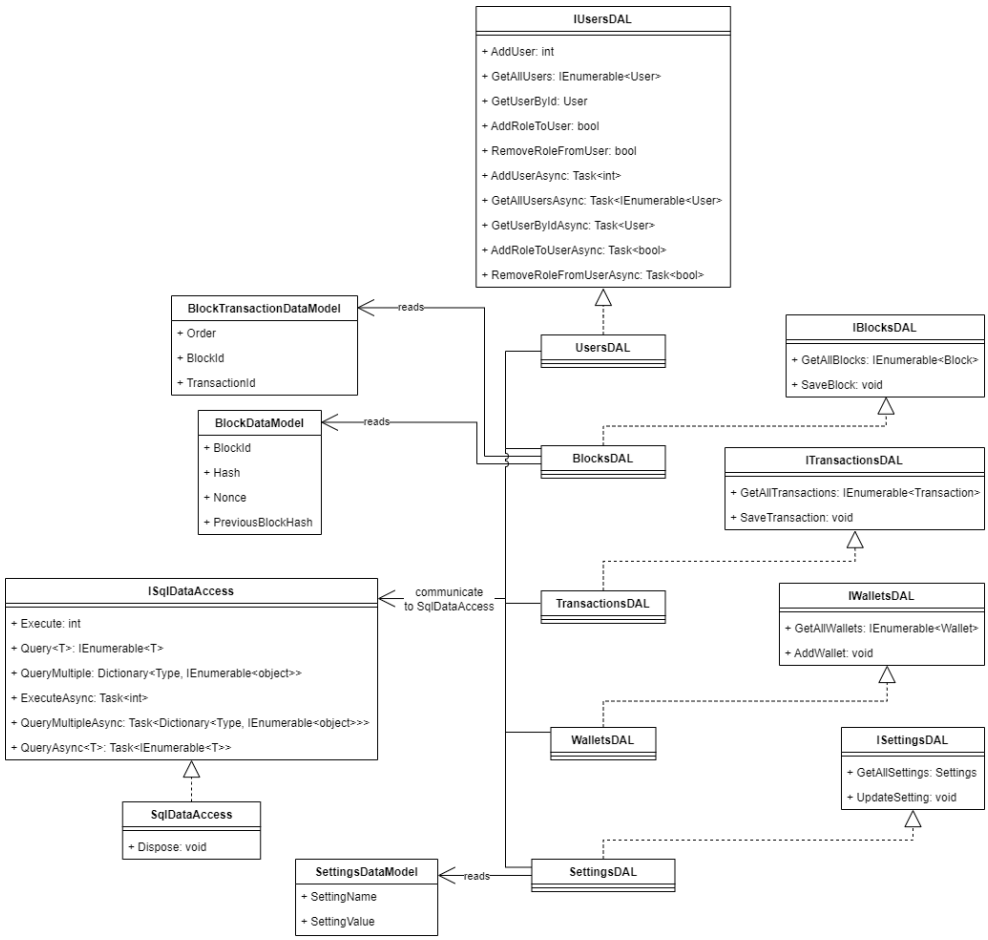

Рисунок 2.2 – Діаграма класів для шару доступу до даних

Кожен DAL клас відповідає за конкретну сутність у баз даних і конвертує дані отримані через запит до моделі, з якою може працювати система. Деякі моделі можуть складатись із кількох сутностей у базі даних, як наприклад Block, що представляє собою комплексний об’єкт, який містить

список інших об’єктів – транзакцій. Оскільки дані для такої моделі зберігають окремі сутності, і також одна спеціальна сутність для зберігання та гарантування порядку транзакцій у блоці, необхідно мати на стороні програмного додатку відповідні моделі, які будуть представляти сутності з бази. І далі ці моделі використовуються для формування однієї моделі, зрозумілої системі.

Також для кожного DAL класу є інтерфейс, який можуть використовувати більш високорівневі модулі системи, без необхідності знати про те, яким саме чином дані отримують із бази.

В загальному DAL представлено наступними класами:

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

На найнижчому рівні знаходиться безпосередньо доступ до бази даних, за який відповідає клас SqlDataAccess та інтерфейс використання для DAL класів ISqlDataAccess.

Для роботи із базою даних використано micro-ORM Dapper. Це інструмент, що дозволяє виконувати запити до бази даних, та за допомогою механізму мапінгу співставляє отримані дані з моделями що надає сервіс для виконання запиту. Також оскільки Dapper – це micro-ORM, на відміну від EnityFramework для прикладу, що являє собою повноцінну ORM, це значно впливає на підвищення продуктивності та швидкості виконання запитів.

Dapper пропонує зручний інструмент для виклику збережених процедуру вигляді всього трьох методів:

- Query – виконує зчитування однієї таблиці через виклик збереженої процедури та повертає результат.
- QueryMultiple – виконує зчитування декількох таблиці через виклик збереженої процедури та повертає результат.
- Execute – виконує операцію із сутністю у базі даних через збережену процедуру без очікування повернення даних. Зазвичай використовується для збереження, оновлення або видалення даних.

Для роботи із базою достатньо викликати один із описаних методів, вказати назву збереженої процедури та вказати додаткові параметри, що є опціональним пунктом.

Для збереження даних використовується база даних MS SQL. На рисунку 2.3 зображено ER-діаграму для бази даних.

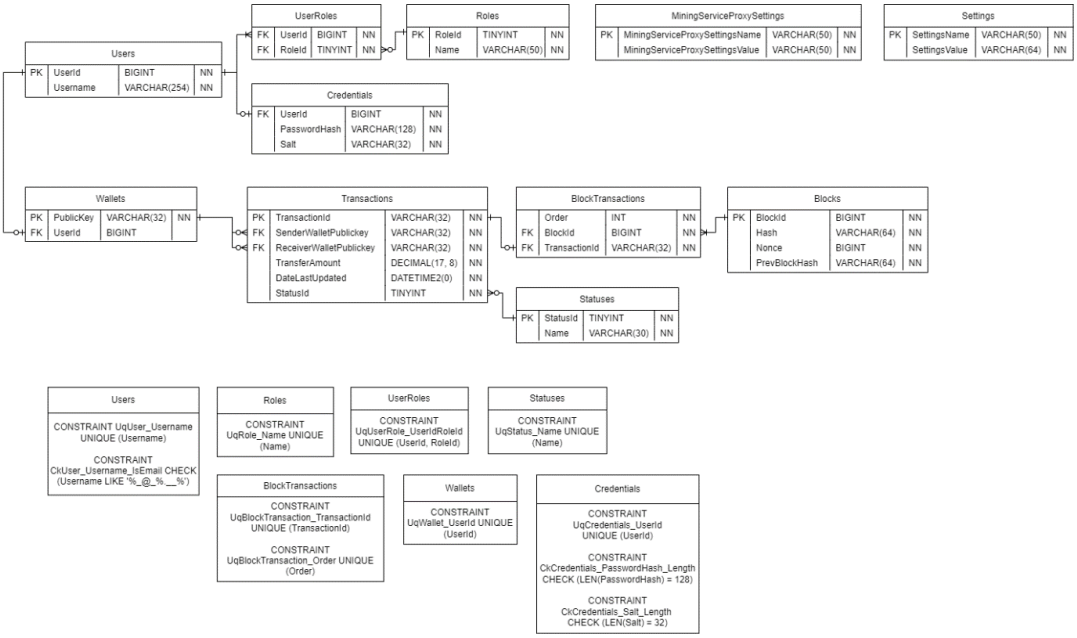

Рисунок 2.3 – ER-діаграма для бази даних

На ER-діаграмі зображено основні сутності системи. Найбільш важливими з яких є наступні:

- Transaction – це найменша одиниця в рамках системи, яка представляє собою криптовалютний переказ з одного гаманця на інший.
- Wallet – гаманець користувача, який виступає в ролі контейнера для транзакцій, що належать відповідному користувачеві.
- Block – це специфічна хешована структура, яка мітить в собі набір підтверджених транзакцій та забезпечує цілісність всіх транзакцій.

Для доступу до даних, що збережені у базі, використовуються збережені процедури – це іншими словами SQL запит, який зберігається як функція на стороні бази даних, і замість створення кожного разу окремого запиту, просто виконується виклик процедури, що виконує заздалегідь збережений запит.

На рисунку 2.4 зображено діаграму класів для сервісу транзакцій.

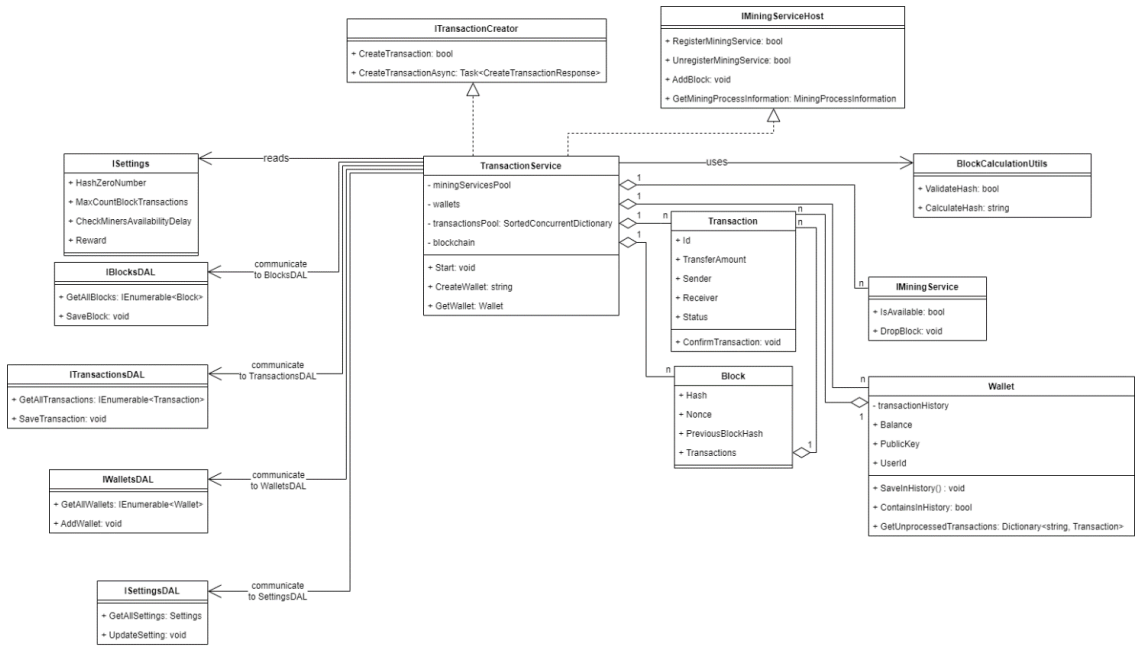

Рисунок 2.4 – Діаграма класів для сервісу транзакцій (TransactionService)

Сервіс транзакцій виконує ключову роль у життєвому циклі системи. Оскільки сервіс виконує початкову валідацію всіх даних наявних у базі даних на старті. Також у процесі роботи валідує всі дані, що повинні бути збережені у базі. Безпосередньо через DAL має можливість зберігати нові

транзакції, блоки, тощо у базі. Бізнес-логіка описана у сервісі транзакцій використовується у API, що викликається із зовнішніх сервісів.

Сервіс оперує такими моделями як:

- Transaction – основна одиниця системи, яка зберігає основну інформацію про умовні одиниці криптовалюти.
- Wallet – структура даних, яка зберігає право власності на транзакцію та дозволяє формувати баланс користувача з наявних транзакцій.
- Block – структура даних, яка забезпечує цілісність транзакцій та є ключовою у процесі підтвердження транзакції.

Має також наступні структури

- Wallets – колекція яка локально зберігає гаманці системи для максимально швидкого доступу.
- Blockchain – односпрямована черга, яка зберігає ланцюг блоків.
- PoolOfUnprocessedTransactions – власноруч створена структура даних, що представляє собою сортований словник, який зберігає дані у порядку додавання.

На рисунку 2.5 зображено сервіс для роботи з електронним гаманцем.

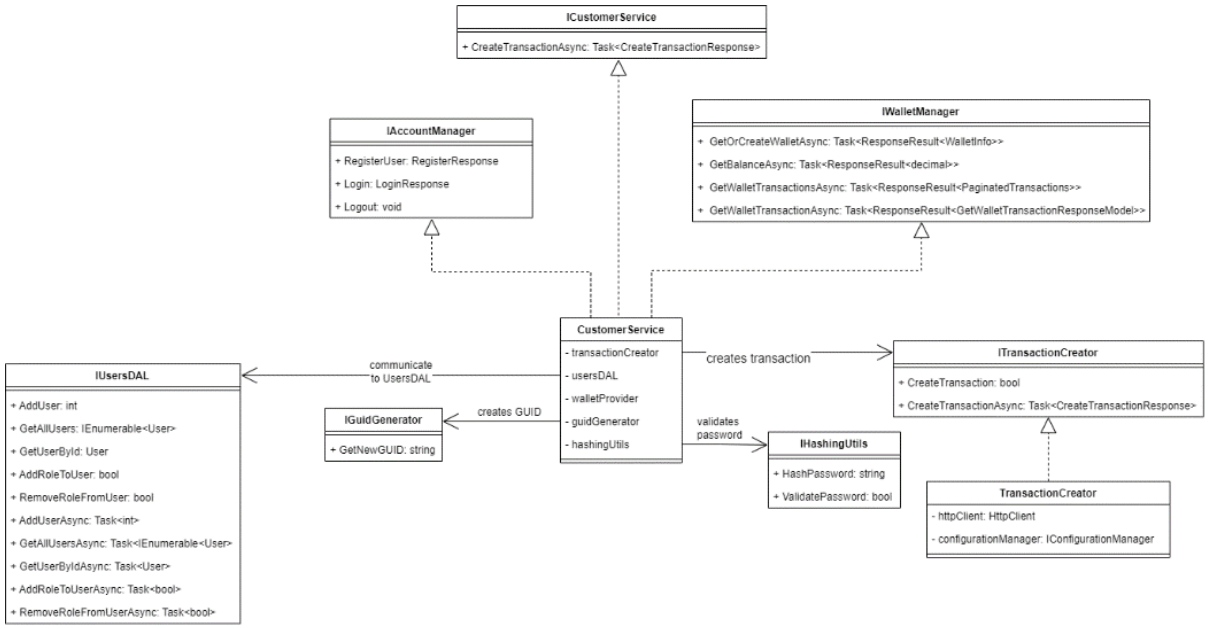

Рисунок 2.5 – Діаграма класів сервісу для роботи з електронним гаманцем (CustomerService)

Основна задача сервісу – це забезпечення можливості взаємодії кінцевого користувача із системою. Сервіс складається із веб-клієнта, доступного користувачам із веб-браузера та сервера який відповідає за роботу із користувачем в цілому, тобто реєстрація, авторизації та аутентифікація, та перенаправлення запитів із веб-клієнта до TransactionService API.

Сервіс в першу чергу повинен надати користувачу можливість для реєстрації, аутентифікації та авторизації, що в подальшому відкриє доступ до перегляду інформації про власний гаманець та дозволить створити нові транзакції.

На рисунку 2.6 зображено сервісс для емісії криптовалюти.

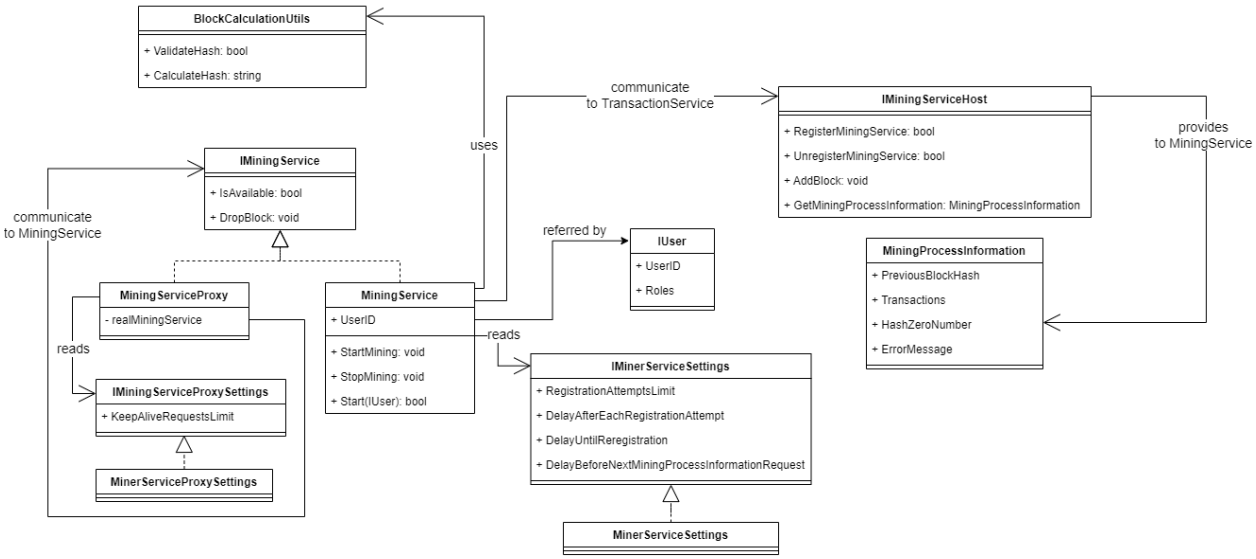

Рисунок 2.6 – Діаграма класів для сервісу емісії криптовалюти (MiningService)

Консольний клієнт забезпечує життєдіяльність транзакції та системи в цілому, оскільки створення нових блоків та емісія криптовалюти є головною задачею сервісу.

Сервіс отримує данні від центрального сервісу транзакцій для обрахунку нових блоків. Після чого отримані результати повинен повернути

до сервісу транзакцій для валідації і в подальшому очікувати нові дані для нових обрахунків.

Процес обрахунку хеша блоку полягає у тому, що необхідно знайти такий хеш, який починався би зі значення, що вимагає сервер. Пошуку такого значення використовується спеціальне поле блока, значення якого додається до обрахунку хеша. Процес майнінгу означає знаходження цього значення при якому хеш блока буде відповідати умові.

2.2 Розробка потокобезпечної моделі збереження даних, сортованої у порядку додавання у вигляді ключ-значення

Важливою частиною сервісу транзакцій є «пул необроблених транзакцій» – колекція, де транзакції тимчасово зберігаються після створення і чекають на підтвердження. Головними вимогами до колекції є потокобезпечність, оскільки зовнішні виклики до колекції відбуваються у різних потоках і часто вони можуть конкурувати між собою, що призведе до генерації виключень у звичайних колекціях. Також колекція повинна надавати зручний та швидкий доступ до даних. Існуючі реалізації не надають таких можливостей, тому необхідно створювати власну реалізацію потокобезпечної колекції з швидким доступом до даних, що зберігаються у порядку додавання. На рисунку 2.7 зображено модель збереження даних у вигляді діаграми класів колекції.

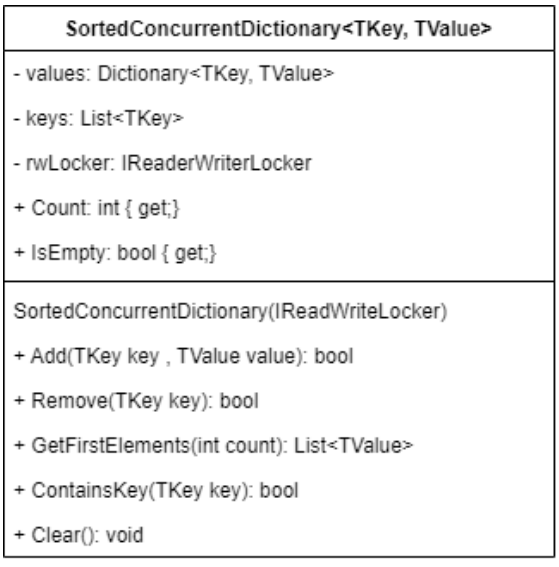

Рисунок 2.7 – Потокобезпечна модель збереження даних у порядку додавання у вигляді діаграми класів колекції

Для роботи з пулом необроблених транзакцій достатньо лише наведених методів – Add, Remove, GetFirstElements, ContainsKey, але в плані використання для інших цілей колекція спроектована як уніфікована, оскільки для ключів та значень зазначені Generic-типи, що дозволяє використовувати будь-які типи даних для збереження, і для розширення функціоналу можливо розширення функціоналу колекції за допомогою реалізації стандартних інтерфейсів колекцій, для прикладу такі як ICollection, IDictionary, IEnumerable, IList, тощо.

Для забезпечення потокобезпечності застосовано ReadWriteLocker, функціонал якого дозволяє створювати області які в момент використання можуть бути доступні лише для роботи одним потоком. Для прикладу, якщо один з потоків виконує операцію запису (додавання нового елементу або видалення існуючого елементу), тобто будь-що, що змінює вміст колекції, інші потоки блокуються до завершення виконання операції, оскільки при читанні даних з колекції потік може звернутись до елементу, що вже не існує, або може виникнути зчитування помилкових даних у результаті додання нових елементів. Для операції читання обмежень по кількості потоків немає до тих пір поки не почнеться операція запису, оскільки зчитування даних по-замовчуванню вважається потокобезпечною операцією, що не змінює вміст колекції.

Для збереження даних в порядку додавання в середині використовується список для збереження ключів. Для збереження даних використовується словник, для швидкого доступу до значення по ключу. Поєднання цих колекцій дозволяє отримати перевагу в зберіганні даних у порядку додання (що не притаманне словникам) та швидкого доступу до даних (що не притаманне спискам). Та в комбінації з механізмом управління потоками ReadWriteLocker отпримаємо потокобезпечну модель для збереження даних у сортованому вигляді зі швидким доступом до даних, що

цілком задовольняє вимого до збереження новостворених транзакцій в пулі необроблених транзакцій.

2.3 Розробка процесів функціонування системи

На рисунку 2.8 зображено модель послідовностей функціонування системи.

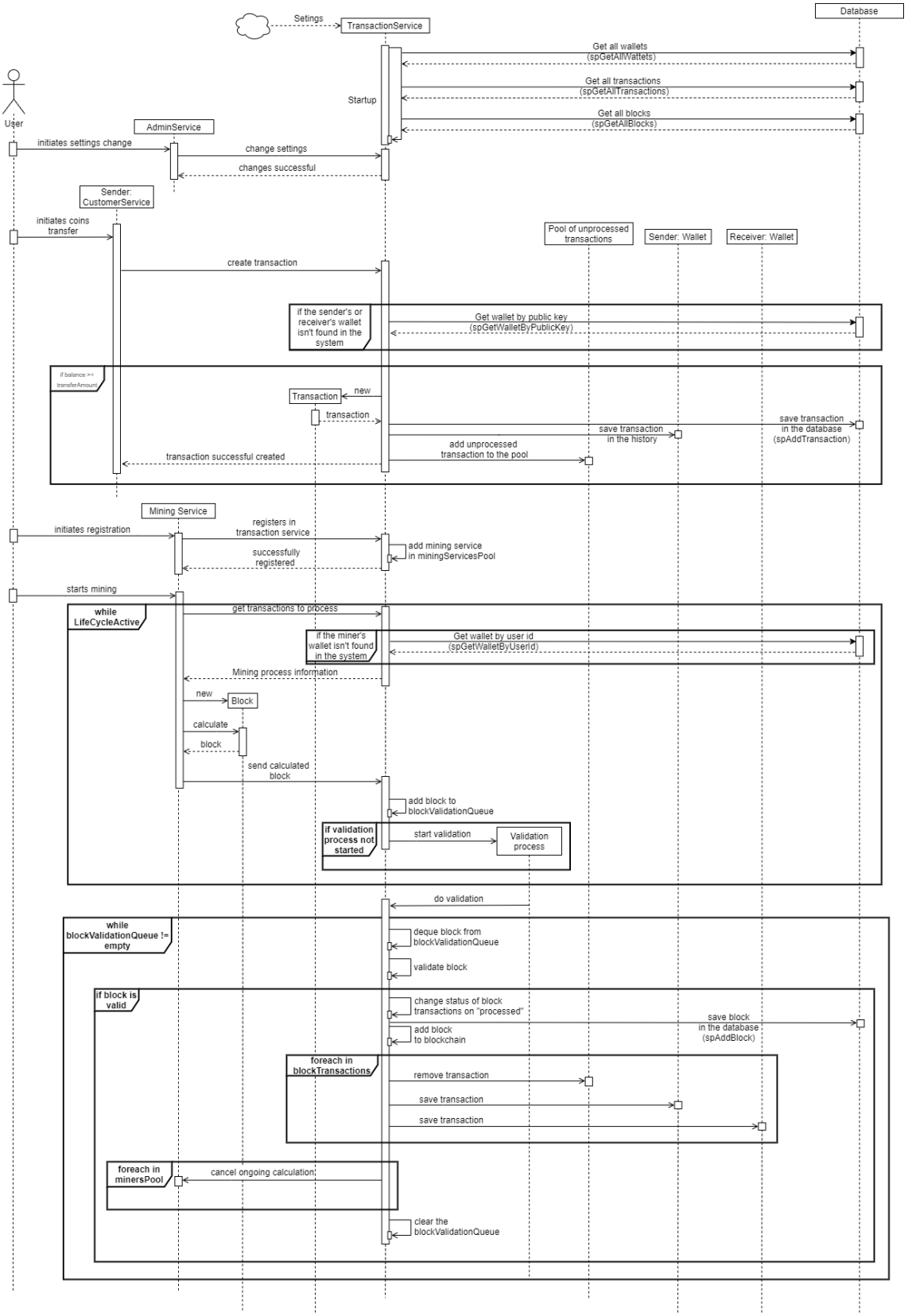

Рисунок 2.8 – Модель послідовностей функціонування системи

Для формування та збереження криптовалюти використовуються дві важливі структури: транзакція і блок. Транзакція виступає ключовою одиницею всієї системи, оскільки криптовалюта у системі не зберігається у

відкритому вигляді, як просте поле у програмному коді, а існує у вигляді взаємозв’язку кількох структур, найпростішою з яких є сама транзакція. Транзакції для затвердження системою спочатку потрапляють у тимчасове сховище очікування. З цього сховища час від часу відбирається певна кількість транзакцій, яка бере учать у емісії криптовалюти. Під час емісії формується блок з відібраних транзакцій. Транзакції з цього блоку підлягають хешуванню та подальшому збереженню у базі даних у захищеному вигляді. Ці блоки формують ланцюг, у якому кожен блок створюється, використовуючи дані із попереднього ланцюга. Це забезпечує цілісність системи та захист від зміни даних.

У моделі (рис. 2.8) продемонстровано загальний життєвий цикл транзакції, а саме: емісію (реєстрацію певної кількості одиниць у системі), потрапляння до електронного гаманця користувача, використання криптовалюти у переказі на інший гаманець – процеси, які проходить транзакція від створення до її затвердження системою.

2.4 Висновки

У другому розділі розроблено централізовану модель криптовалюти та базу даних системи. Розроблено потокобезпечну модель збереження даних, сортовану у порядку додавання у вигляді ключ-значення. Розглянуто процеси функціонування системи.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ КРИПТОВАЛЮТНОЇ СИСТЕМИ

3.1 Варіантний аналіз і обґрунтування вибору інструментів для розробки

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

На сьогоднішній день існує безліч мов програмування загального призначення. І кожен з цих інструментів займає певну нішу у різноманітних напрямках. Наприклад C++ абсолютний лідер у ігровій індустрії, для розробки веб-сервісів, в часності back-end розробка, найпопулярнішими мовами є Java та C#. У випадку для розробки під платформу Windows абсолютним лідером є саме C#, натомість Java використовується для розробки під інші операційні системи. Але з появою .Net Core і подальшим розвитком у .Net 5/6 C# починає займати і кросплатформенний сегмент. C# пропонує приємний та зручний синтаксис, функціональну базову бібліотеку, що має всі необхідні інструменті для швидкої розробки.

Отже приймаючи до уваги абсолютне лідерство платформи .Net [8] для розробки під Windows та перспективи розвитку кросплаформенності було обрано мову C# для розробки серверної частини системи, бізнес-логіки та додатку для емісії криптовалюти.

3.1.2 Аналіз середовища розробки

Visual Studio [9] є безперечним лідером для розробки додатків мовою C#. IDE пропонує один з найкращих інструментів для доповнення коду в ході розробки IntelliSense, що в останніх версіях зазнав значних покращень в плані передбачення того, що повинно бути написано. Також Visual Studio пропонує досить зручні інструменти для тестування, дебагінгу та публікації додатку.

Вбудований інструмент для інсталяції пакетів Nuget дозволяє досить зручно управляти сторонніми пакетами, які повинні бути використані додатком.

3.1.3 Аналіз бази даних

Для збереження даних системи було обрано базу даних MS SQL Server [10]. База даних є реляційною, що дозволяє створити зв’язки між сутностями ті підтримувати цілісність всіх даних.

MS SQL Server відома тісною інтеграцією із платформою .Net що є ключовим фактором у виборі, опираючись на обрані інструменти для розробки системи. Також SQL Server пропонує досить високу продуктивність у роботі та безвідмовність.

3.1.4 Аналіз інструментів для роботи з базою даних

Існує 2 основних варіанти роботи з базую даних – це виклик збережених процедур [11], та виклик прямих запитів. Отже розглянемо ці варіанти.

Збережені процедури, це певний набір дій з базою даних завернутий у єдину процедуру.

Переваги:

1. Продуктивність через кешування – збережена процедура будує план виконання, перевіряє синтаксис тільки при першому виклику і при кожному наступному виклику забирає його з кешу.

2. Менший шанс SQL-ін'єкції – перед відправкою запиту ми зв'язуємо виклик збереженої процедури з параметрами в одному об'єкті.

3. Зменшення використаного трафіку – запит надсилається просто як виклик функції з параметрами.

4. Можливість налаштувати доступ до збережених процедур для різних користувачів – для кожного користувача ми можемо обмежити доступ до

процедур, тим самим обмежуючи можливості користувача в системі, щоб запобігти небажаним діям.

5. Повторне використання коду – зміни в процедурі не потребують змін у точках виклику.

6. Запити існують окремо від бізнес-логіки додатку.

7. Можливість повертати більше ніж одне значення через зовнішні параметри.

8. Немає необхідності перекомпілювати весь проект після змін у збереженій процедурі.

9. Простіше оптимізувати.

Недоліки:

1. Дещо складніший контроль версій, оскільки код збережених процедур необхідно десь окреме зберігати окрім бази даних, щоб мати доступ. При зміні збереженого файлу з процедурами необхідно внести відповідні зміни до бази даних.

Прямі запити – це просто SQL-код, що передається у якості параметра до виклику до бази даних у вигляді тексту.

Переваги:

1. Простіше з контролем версій, оскільки запити являються частиною логіки додатку.

Недоліки:

1. Без кешування – для кожного виклику створюється план виконання, перевіряється синтаксис.

2. Легше ввести деякий код SQL у запит. Запит надсилається у вигляді рядка, і простіше ввести деякий код SQL через параметри.

3. Використовується більше трафіку – запит надсилається у вигляді рядка з усією логікою, яка може бути досить значною.

4. Усі користувачі можуть робити будь-які дії із доступними таблицями – оскільки користувач не має обмежень щодо роботи з доступною таблицею.

5. Повторюваний код – деякі запити можна використовувати в кількох місцях, і якщо ми хочемо змінити якусь логіку в нашому запиті, ми повинні змінити її скрізь.

6. Запити це частина бізнес-логіки додатку.

7. Запит не може повернути більше одного параметра.

8. Після змін у запиті ми повинні перекомпілювати весь проект.

9. Складніше оптимізувати.

Отже з переважною більшістю переваг і не дивлячись на деякі додаткові маніпуляції для підтримки можливості контролю версій вирішено використовувати як інструмент для роботи з базою даних саме збережені процедури.

Для виклику збережених процедур необхідно використовувати додатковий інструмент на стороні бізнес-логіки. В загальному існує безліч ORM для доступу до бази даних, але розглянемо основні. Основою для всіх ORM на платформі .NET є технологія ADO .NET. Вона надає базові можливості для роботи з базою даних і є найшвидшою в плані продуктивності, але недоліком є ускладнений процес розробки, оскільки всю комунікацію необхідно налаштовувати вручну. Для спрощення роботи є інші варіанти, такі як EnityFramework та Dapper [12]. EntityFramework є повноцінною ORM, що дозволяє вносити зміни в базу даних напряму. Але недоліком, що дозволяє зручно використовувати цей інструмент, є найнижча продуктивність. Наприклад зручним варіантом використання є можливістю писати LINQ-запити [13] на мові C#, які будуть перетворені у SQL-запити. Але це вимагає додаткових навантажень на систему. Для порівняння Dapper не дозволяє використати LINQ для створення запиту, але натомість пропонує зручні інструменти для виклику збережених процедур або прямих запитів.

Якщо порівняти із іншими інструментами Dapper являє собою певну золоту середину, пропонуючи не занадто низькорівневі функціональні можливості та має досить гарну продуктивність.

Отже опираючись на ці фактори вирішено використовувати для зв’яку з базою даних micro-ORM Dapper.

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

Для створення браузерного клієнта Asp. Net [14] пропонує Razor – це двигун для рендерингу сторінок на стороні сервера. Сторінки створюються з використанням змішаного синтаксису HTML та C#, що дозволяє зручно напряму використовувати моделі представлені у системи, написані мовою C#. Також наявні допоміжні інструменти для швидкого виконання запитів до контролеру, що спрощує та пришвидшує розробку.

Для дизайну сторінки використано популярну CSS-бібліотеку Bootstrap [15], що дозволяє досить швидку створити зручний та адаптивний користувацький інтерфейс.

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. На рисунку 3.1 зображено реалізацію методу.

```
public IEnumerable<T> Query<T>(string procedure, object arguments = null)
{
    IEnumerable<T> entities;

    entities = connection.Query<T>(procedure, arguments, commandType:
CommandType.StoredProcedure);
```

Рисунок 3.1 – Реалізація методу Query

- QueryMultiple – Дозволяє повернути результат одразу із кількох таблиць. На рисунку 3.2 зображено реалізацію методу.

```
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;
    }
```

Рисунок 3.2 – Реалізація методу QueryMultiple

- Execute – виконує виклик збереженої процедури для викнання змін у таблицях бази даних та не очікує повернення будь-яких даних. На рисунку 3.3 зображено реалізацію методу.

```
public int Execute(string procedure, object arguments = null)
    {
        return connection.Execute(procedure, arguments, commandType:
CommandType.StoredProcedure);
    }
```

Рисунок 3.3 – Реалізація методу Execute

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

Метод, що зберігає транзакцію у базу не очікує, що якісь дані повернуться після виконання запиту, тому використовується метод Execute. Для виклику формується об’єкт з заповненими полями відповідно до параметрів збереженої процедури і передається у якості параметру, як і назва

збереженої процедури. На рисунку 3.4 зображено реалізацію методу SaveTransaction.

```
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(SpAddTransaction, arguments);
}
```

Рисунок 3.4 – Реалізація методу SaveTransaction

Метод, що зберігає повертає список гаманців у базу очікує на дані які повернуться після виконання запиту, тому використовується метод Query. У метод GetAllWallets передаються у якості параметра всі транзакції, якими потрібно заповнити гаманці, оскільки транзакції зберігаються як окрема сутність. На рисунку 3.5 зображено реалізацію методу.

```
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);
        }
```

Рисунок 3.5 – Реалізація методу GetAllWallets

Метод, що зберігає повертає список блоків у базу очікує на дані які повернуться з декількох таблиць після виконання запиту, тому використовується метод QueryMultiple. У метод GetAllBlocks також

передаються у якості параметра всі транзакції, якими потрібно заповнити блоки згідно порядку. На рисунку 3.6 зображено реалізацію методу.

```
public IEnumerable<Block> GetAllBlocks(IEnumerable<Transaction> transactions)
{
    var blockDataModelType = typeof(BlockDataModel);
    var blockTransactionDataModelType = typeof(BlockTransactionDataModel);
    var types = new List<Type>
    {
        blockDataModelType,
        blockTransactionDataModelType,
    };

    var queryResult = sqlDataAccess.QueryMultiple(SpGetAllBlocks, types);

    var blocksDB = queryResult[blockDataModelType].Cast<BlockDataModel>();
    var blockTransactionsDB = queryResult[blockTransactionDataModelType]
        .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(transactionsDB[transactionDB.TransactionId]);
        }

        blocks.Add(new Block(blockDB.PrevBlockHash, blockTransactions) { Hash = blockDB.Hash, Nonce
= (uint)blockDB.Nonce });
    }

    return blocks;
}
```

Рисунок 3.6 – Реалізація методу GetAllBlocks

3.3 Розробка сервісу транзакцій

Один із основних сервісів системи – це TransactionService, основна ціль якого це валідація та обробка даних, отриманих через запит від клієнтських сервісів до API [16]. Для доступу до логіки сервіс надає API створене за допомогою ASP .NET WebAPI Core на основі REST [17]. Для кожної категорії процесів API має окремий контролер для обробки вхідних запитів. В загальному основними контролерами є наступні:

- BlochainController – надає можливість додати новий блок до системи отриманий в результаті обрахунку у клієнтському додатку MiningService.

- MiningServiceController – надає інформацію для обрахунку нового блоку по запиту від клієнтського додатку MiningService.
- TransactionController – надає можливість створити нову транзакцію по запиту від клієнтського веб-додатку CustomerService.
- WalletController – надає доступ до інформації про користувацькі електронні гаманці по запиту від клієнтського веб-додатку CustomerService.

Кожен контролер слугує певним ретранслятором запиту. Кожен метод контролера лише виклика бізнес-логіку описану у різних модулях, що робить систему більш гнучкою у розробці та простою у використанні. Оскільки .NET Core додатки мають вбудований DI Container, це дозволяє змінювати модулі, які повинні бути використані у контролерах (і не тільки), просто змінивши 1 стрічку коду, яка відповідає за реєстрацію модуля у DI Container.

Розглянемо більш детально, що заходиться поза викликами бізнес-логіки у контролерах.

3.3.1 Розробка контролера для роботи з блоками

Контролер має лише один простий метод AddBlock та приймає у якості параметра новий блок. На рисунку 3.7 зображено класову діаграму моделі Block.

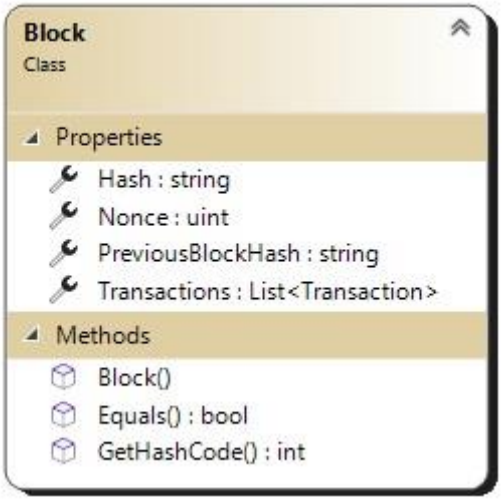

Рисунок 3.7 – Сигнатура класу Block

На рисунку 3.8 зображено реалізацію методу AddBlock.

```
[HttpPost]
public async Task<ResponseResult> AddBlock(Block block)
{
    await miningServiceHost.AddBlockAsync(block);
    return new ResponseResult()
    {
        IsSuccessful = true,
        Message = null,
    };
}
```

Рисунок 3.8 – Реалізація методу AddBlock

Як видно з реалізації AddBlock є POST методом. Це дозволяє відправляти блоки у тілі запиту в захищеному форматі при використанні розширеного протоколу HTTPS, що підтримує шифрування.

Сам метод має лише два логічних блоки – це виклик асинхронного методу бізнес-логіки, яка представлена інтерфейсом IMiningServiceHost, та повернення результату в уніфікованому форматі, представленому моделлю ResponseResult, яка має інформацію про статус операції, та деяке повідомлення, що може містити інформацію про операцію.

Як було згадано вище метод контролера викликає відповідну бізнес-логіку. У цьому випадку це метод AddBlockAsync, що імплементовано у модулі TransactionService. Асинхронний метод викликає інший внутрішній синхронний метод AddBlock. На рисунку 3.9 зображено реалізацію методу.

```
public void AddBlock(Block block)
{
    blockValidationQueue.Enqueue(block);

    logger.LogInformation("Block {@block} prepared for validation.", block);
    if (!isValidationActive)
    {
        locker.Enter(startValidationLocker);
        if (!isValidationActive)
        {
            isValidationActive = true;
            locker.Exit(startValidationLocker);
            StartValidation();
        }
        else
        {
            locker.Exit(startValidationLocker);
        }
    }
}
```

Рисунок 3.9 – Реалізація методу AddBlock

Головною задачею методу є додання блоку до черги та ініціація процесу валідації, якщо черга була пустою перед викликом. Черга створена за допомогою потокобезпечної колекції ConcurrentQueue для запобігання генерації виключень, оскільки одночасно може виконуватись декілька запитів на додання блоку.

Також для запобігання можливої повторної ініціації процесу валідації, через одночасний виклик у кількох потоках методу, що описано вище, застосовано патерн Double-checked locking. На рисунку 3.10 наведено діаграму принципу роботи методу AddBlock, що описано вище. Виходячи з назви патерну зрозуміло, що для того ініціювати валідацію необхідно пройти дві перевірки та заблокувати виконання інших потоків через монітор. Обидві перевірки мають переконатись, що процес валідаці ще не був розпочатий. Але важливий момент перед другою перевіркою необхідно заблокувати інші потоки, таким чином коли почнеться перевірка умови будуть відхиляти ініціативу розпочати інший, паралельний процес валідації та інші потоки просто підуть через іншу гілку.

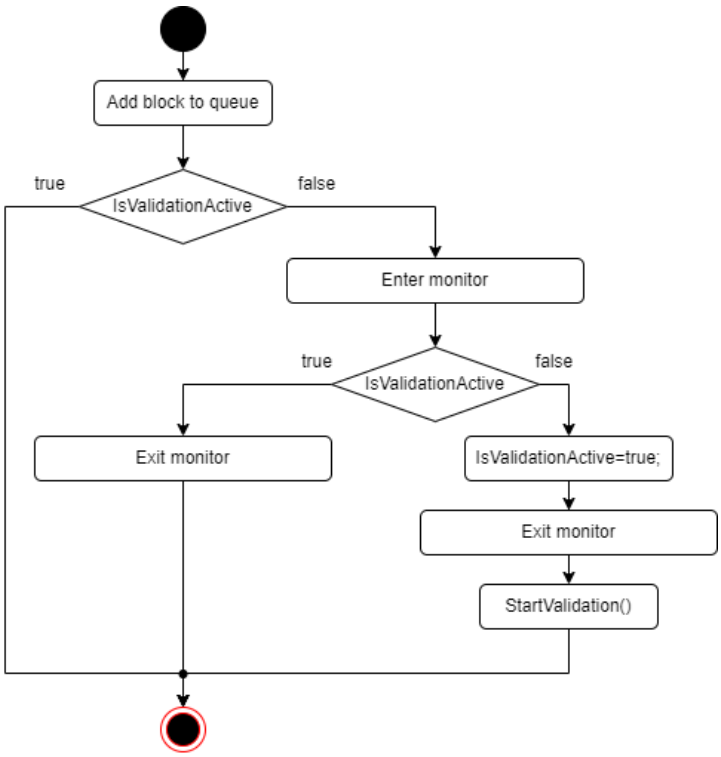

Рисунок 3.10 – Double-checked locking на прикладі AddBlock методу

Наступний етап – початок валідації. Тут відбувається процес обробки черги блоків, у пошуках валідного блоку серед присутніх. У порядку черги з колекції витягується 1 блок для якого перевіряються всі поля на рахунок заданих умов при яких блок може вважатись валідним. На рисунку 3.11 зображено алгоритм процесу валідації блоку.

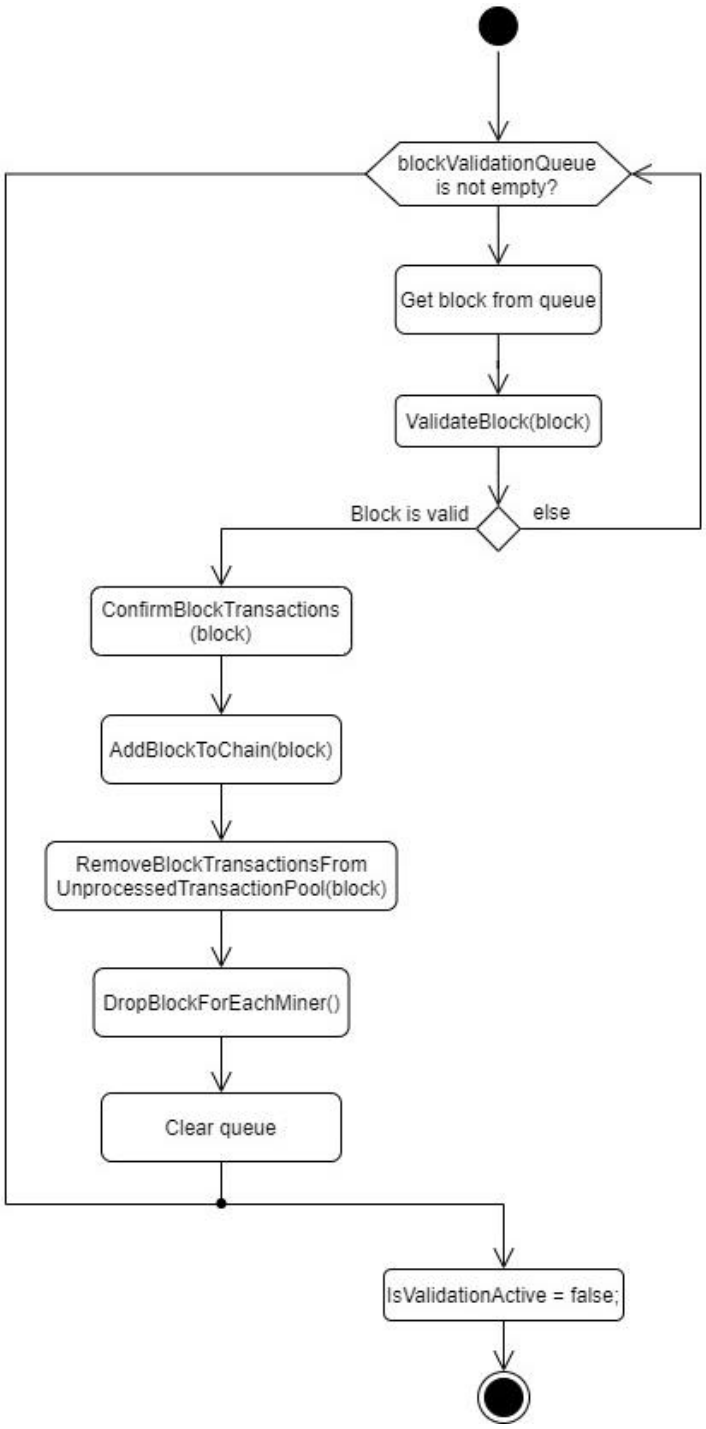

Рисунок 3.11 – Діаграма активності для процесу валідації черги блоків

В першу чергу цей блок повинен пройти процес валідації. А саме перевірка всіх даних, з яких складається блок. Цей процес виконується наступним чином:

- Перевірка поля що зберігає хеш останнього збереженого у системі блока. Якщо це поле не співпадає з реальним хешом останнього збереженого блока у системі, то блок є не валідним і виключається із подальшого процесу валідації, навіть якщо інші поля задовольняють усі умови.
- Перевірка правильності обрахунку хешу. Поле що зберігає хеш блока повинно збігтися зі значенням яке система отримує з повторного обрахунку хешу [18], використовуючи всі поля блоку. Якщо хеш не співпадає – блок є не валідним.
- Перевірка формату хешу який задовольнить умову валідації, в загальному це певний набір символів, з якого повинен починатись хеш блока. Якщо ця умова не була задоволена і хеш не починається з певного набору символів, то блок не є валідним.
- Валідація транзакцій. Кожна необроблена транзакція тимчасово розташовується у спеціальний колекції – «Пул необроблених транзакцій». І кожен блок повинен містити лише транзакції, що під час валідації знаходяться у пулі. Єдиним виключенням є системна транзакція яка є результатом емісії криптовалюти. Оскільки блок містить 2 типи транзакцій, тому їх валідація відбувається окремо:
    - Валідація системної транзакції. Кожен блок обов’язково повинен містити систему транзакцію, в іншому випадку блок не може вважатись валідним. Також додатковими умовами є що системна транзакція повинна бути створена з системного гаманця, до якого неможливо отримати ручний доступ. І системна транзакція повинна бути адресована гаманцеві, власник якого створив новий блок,

використовуючи MiningService. На рисунку 3.12 зображено алгоритм валідації системной транзакції.

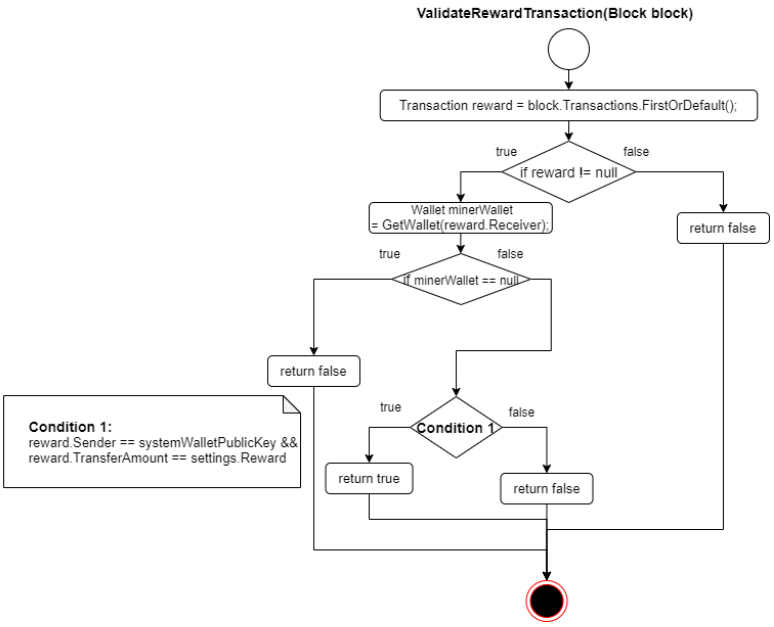

Рисунок 3.12 – Валідація системної транзакції

o Валідація звичайних транзакцій з пулу виконується іншим шляхом. В першу чергу під час валідації перевіряється цілісність даних, тобто чи не було змінено транзакції, чи не було додано дублікатів, тощо. Головне щоб транзакції блоку знаходились у пулі на момент валідації. На рисунку 3.13 зображено алгоритм валідації транзакцій.

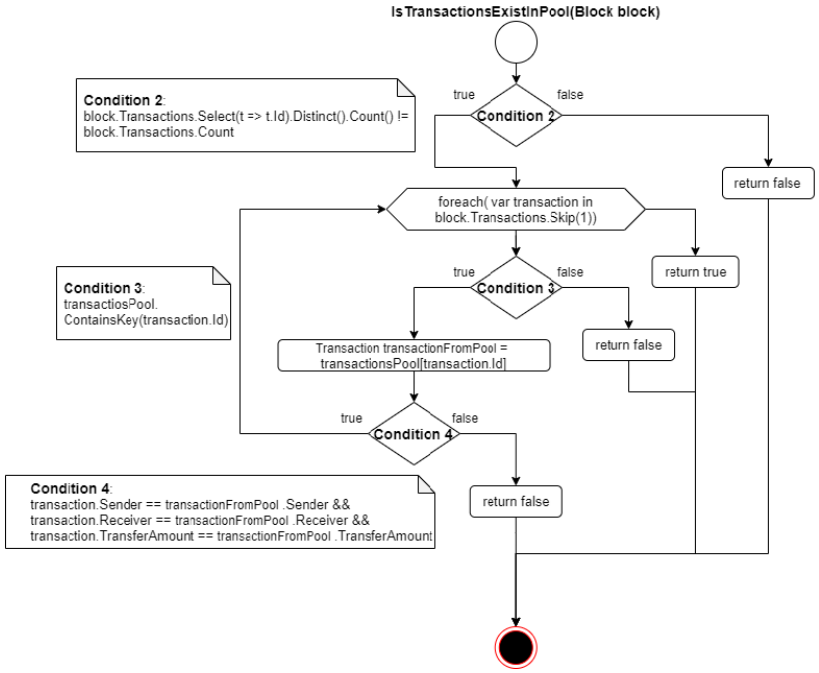

Рисунок 3.13 – Діаграма активності для валідації транзакцій

Реалізація вище описаних кроків методу для валідації блоку наведено нижче на рисунку 3.14.

```
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.CalculateHash(block.Transactions);
    string currentBlockHash = blockCalculationUtils.CalculateHash(previous.Hash, transactionsHash,
block.Nonce);
    if (previous.Hash == block.PreviousBlockHash
      && block.Hash == currentBlockHash
      && blockCalculationUtils.ValidateHash(block.Hash, settings.HashZeroNumber)
      && ValidateBlockTransactions(block)){
      return true;
    }

    return false;
  }
```

Рисунок 3.14 – Реалізація методу ValidateBlock

Якщо блок пройшов всі етапи валідації, це означає що блок є валідним і повинен бути доданий ко ланцюга блоків та збережений у базі даних. Для цього блока підтверджуються системою всі транзакції та видаляються з пулу. Для всіх зареєстрованих сервісів майнингу ініціюється виклик події для оновлення даних, для яких обраховується хеш блоку. Та останнім кроком є очищення черги, оскільки після додання нового блоку всі інші блоки з черги не зможуть пройти валідацію хешу попереднього блоку, тому для блоків що залишились валідація не є раціональною.

3.3.2 Розробка контролера для роботи з сервісом для емісії криптовалюти

Як було описано на початку цей контролер відповідає за отримання даних для обрахунку нового блоку. Контролер також має лише один метод, але у порівнянні з попереднім, тип цього методу GET, оскільки інформація яка надається із запитом не є критичною, та не має впливу на працездатність системи в цілому. Реалізацію методу контроллера наведено на рисунку 3.15.

```
    [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,
        };
}
```

Рисунок 3.15 – Реалізація методу GetMiningProcessInformation

Логічна структура методу не має відмінностей від описаного попереднього прикладу, тому перейдемо одразу до бізнес-логіки, за яку також відповідає інтерфейс IMiningServiceHost.

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

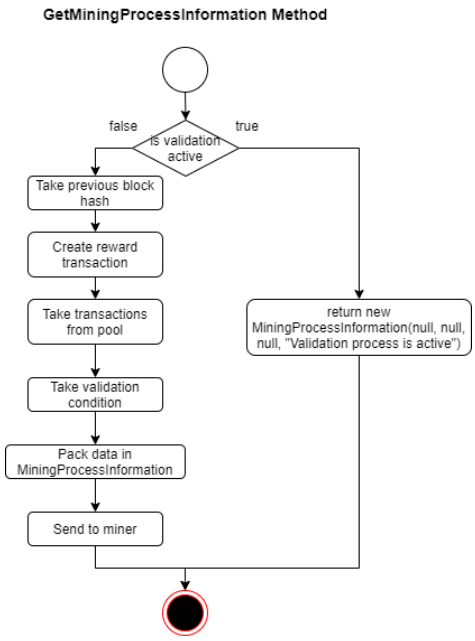

Рисунок 3.16 – Діаграма активності для методу GetMininProcessInformation

В загальному, як і наведено на діаграмі, необхідно перевірити чи не проходить в поточний час валідація блоків. Якщо так – метод поверне повідомлення про те, що зараз проходить валідація і необхідно виконати повторний запит після завершення, оскільки під час валідації швидше за все буде додано новий блок у систему і інформація для обрахунку нових блоків оновиться. Адже обрахунок блоків з неактуальними даними не є раціональним.

Коли валідація завершилась, або ж навіть не розпочато, метод повинен отримати поточний хеш останнього доданого до системи блоку, адже це значення використовується в якості зв’язуючого для кожного блоку, що дозволяє сформувати логічну послідовність. Наступним кроком є створення системної транзакції, яка виконує роль емісії криптовалюти. Отримувач транзакції виявляється за допомогою ідентифікатора користувача, відправленого із запитом. Після чого формується список транзакцій, які

чекають опрацювання у відповідному сховищі. Кількість транзакцій у блоці обмежена по замовчуванню, але може бути розширена зміною значення у налаштуваннях. Останнім кроком є задання умови валідності хешу нового блоку та запакування сформований даних у модель.

3.3.3 Розробка контролера для роботи з транзакціями

Як було описано на початку цей контролер відповідає за обробку запитів для створення транзакцій. Для створення транзакції метод очікує такі параметри, як публічні ключі відправника, отримувача та сума переказу. Реалізація методу контролера наведено на рисунку 3.17.

```
[HttpPost]
public async Task<CreateTransactionResponse> CreateTransaction(CreateTransactionModel data)
{
    var result = await transactionCreator.CreateTransactionAsync(data.Sender, data.Receiver,
data.Amount);
    return result;
}
```

Рисунок 3.17 – Реалізація методу CreateTransaction

Отримані дані контролер передає далі у бізнес-логіку, де проходить валідація даних під час якої виявляється чи може транзакція бути створена. Для успішного проходження валідації необхідно вказати існуючі публічні ключі гаманців, також сума переказу не повинна перевищувати наявний баланс у відправника. Якщо дані вказано правильно – транзакція створюється та зберігається у гаманці відправника як необроблена. Також транзакція потрапляє у спеціальне сховище – «пул необроблених транзакцій», де кожна транзакція очікує підтвердження. Реалізація методу наведено на рисунку 3.18.

```
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;
}
```

Рисунок 3.18 – Реалізація методу CreateTransaction

3.3.4 Розробка контролера для роботи з електронними гаманцями

Як було описано на початку цей контролер відповідає за обробку запитів для роботи з гаманцем. На відміну від попередніх контролерів тут наявні декілька методів, які описано нижче.

Метод бізнес-логіки, яку викликає метод контролера повинен в першу чергу зробити спробу отримати гаманець із сховища. Якщо ж виявляється, що гаманець відсутній метод виконує спробу створити гаманець та зберегти у базі даних. У переважній більшості випадків метод повинен спрацювати безвідмовно. Але для виняткових ситуацій, коли щось йде по

незапланованому сценарію створені додаткові перевірки, після яких описуються у окремих файлах дані про виявлену помилку. Реалізація методу контролера наведено на рисунку 3.19.

```
public async Task<ResponseResult<WalletInfo>> GetOrCreateWalletAsync(GetWalletModel model){
        var wallet = GetWallet(model.UserId);
        if (wallet == null){
            wallet = new Wallet(guidGenerator.GetNewGUID(), model.UserId);
            try{
                if (walletsDAL.AddWallet(wallet)){
                    wallets.TryAdd(wallet.PublicKey, 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.InternalError,
                IsSuccessful = false,
            };
        }

        return await Task.FromResult(getWalletResponse);
    }
```

Рисунок 3.19 – Реалізація методу GetOrCreateWalletAsync

GetWallet – виконує виклик бізнес-логіки, яка створює гаманець, якщо для вказаного користувача гаманець відсутній, або ж повертає вже існуючий. Реалізація методу контролера наведено на рисунку 3.20.

```
        [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;
        }
```

Рисунок 3.20 – Реалізація методу GetWallet

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. Реалізація методу наведено на рисунку 3.21.

```
public async Task<ResponseResult<decimal>> GetBalanceAsync(GetWalletBalanceModel 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.InternalError,
            IsSuccessful = false,
        };
    }
```

Рисунок 3.21 – Реалізація методу GetBalanceAsync

GetBalance – метод, що повинен повернути поточний баланс гаманця. Як і кожен метод будь-якого контролера також викликає бізнес-логіку, описану інтерфейсом IWalletProvider. Реалізацію методу наведено на рисунку 3.22

```
        [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;
        }
```

Рисунок 3.22 – Реалізація методу GetBalance

Бізнес-логіка, яку викликає контролер приймає до уваги лише транзакції гаманця користувач. І якщо цей гаманець містить вказану транзакцію – метод її поверне, в іншому випадку відповідь на запит буде пустою. Реалізацію методу наведено на рисунку 3.23.

```
  public async Task<ResponseResult<GetWalletTransactionResponseModel>>
GetWalletTransactionAsync(GetWalletTransactionModel model)
    {
        ResponseResult<GetWalletTransactionResponseModel> 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<GetWalletTransactionResponseModel>()
            {
                Model = null,
                Message = ResponseMessages.InternalError,
                IsSuccessful = false,
            };
        }
        else
        {
            var walletTransactions = wallet.History;
            var transaction = walletTransactions.FirstOrDefault(t => t.Id == model.TransactionId);
            var responseModel = new GetWalletTransactionResponseModel()
            {
                Transaction = transaction,
                WalletPublicKey = wallet.PublicKey,
            };
            getWalletTransactionResponse = new ResponseResult<GetWalletTransactionResponseModel>()
            {
                Model = responseModel,
                Message = string.Empty,
                IsSuccessful = true,
            };
        }

        return await Task.FromResult(getWalletTransactionResponse);
         }
```

Рисунок 3.23 – Реалізація методу GetWalletTransactionAsync

GetTransaction – метод що повинен повернути лише вказану транзакцію за її ідентифікатором. Реалізацію методу контролера наведено на рисунку 3.24.

```
    [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;
    }
```

Рисунок 3.24 – Реалізація методу GetTransaction

GetTransactions – метод, що повинен повернути всі транзакції гаманця у подрібненому форматі. Мається на увазі, що на сторінці, де будуть відображені транзакції, буде використовуватись лише частина з них. Реалізація методу контролера наведено на рисунку 3.25.

```
[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;
}
```

Рисунок 3.25 – Реалізація методу GetTransactions

Оскільки гаманці можуть мати досить велику кількість транзакцій, для кінцевого користувача більш зручним варіантом перегляду є можливість мати так названі сторінки, де відображується лише певна кількість. Саме тому метод бізнес-логіки за запитом повертає лише «сторінку» транзакцій, яку вказав користувач. Також окрім певної сторінки користувач має 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 ціль сервісу для користувачів – це надання доступу авторизованим та аутентифікованим користувачам до інформації про їх гаманці у реальному часі, та можливість створювати нові транзакції при умові достатнього балансу.

Окрім логіки обробки даних сервіс також надає користувацький інтерфейс через взаємодією з яким користувач має можливість ініціювати

різні запити до системи, для отримання даних, або створення транзакцій для прикладу.

Сервіс пропонує наступні функціональні можливості, для який створено UI та описано серверну частину яка безпосередньо опрацьовує всі дії, що виконуються користувачем:

- Реєстрація для нових користувачів.
- Аутентифікація та авторизація існуючих користувачів.
- UI для відображення та фільтрація даних про електронний гаманець у зручній формі.
- UI для створення транзакцій.

Розглянемо більш детально кожну функціональну можливість.

3.4.1 Реєстрація, аутентифікація та авторизація користувачів

Аутентифікація виконується на основі такого формату, як CookieAuthentication [19]. Функціонування самого механізму забезпечується за допомогою специфікації OWIN (Open Web Interface for .NET), що визначає взаємодію веб-клієнта із веб-сервером. Згідно цієї специфікації архітектура будується наступним чином – веб-додаток складається з:

- Хост – процес, що виконує серверний додаток. Основна задача – це запустити додаток та завантажити компоненти OWIN.
- Сервер – додаток, що реалізує специфікацію OWIN. Основна задача – це виконання запитів від клієнта та їх обробка через конвеєр OWIN.
- Middleware – це компоненти, через які запити від клієнта проходять обробку.
- Клієнт – безпосередньо сам додаток через який користувачі взаємодіють із системою.

Для отримання доступу до акаунту викликається метод бізнес-логіки Login. Що виконує ряд перевірок для введених користувачем даних, згідно

яких приймається рішення щодо надання доступу до акаунту. Алгоритм роботи методу наведено на рисунку 3.26.

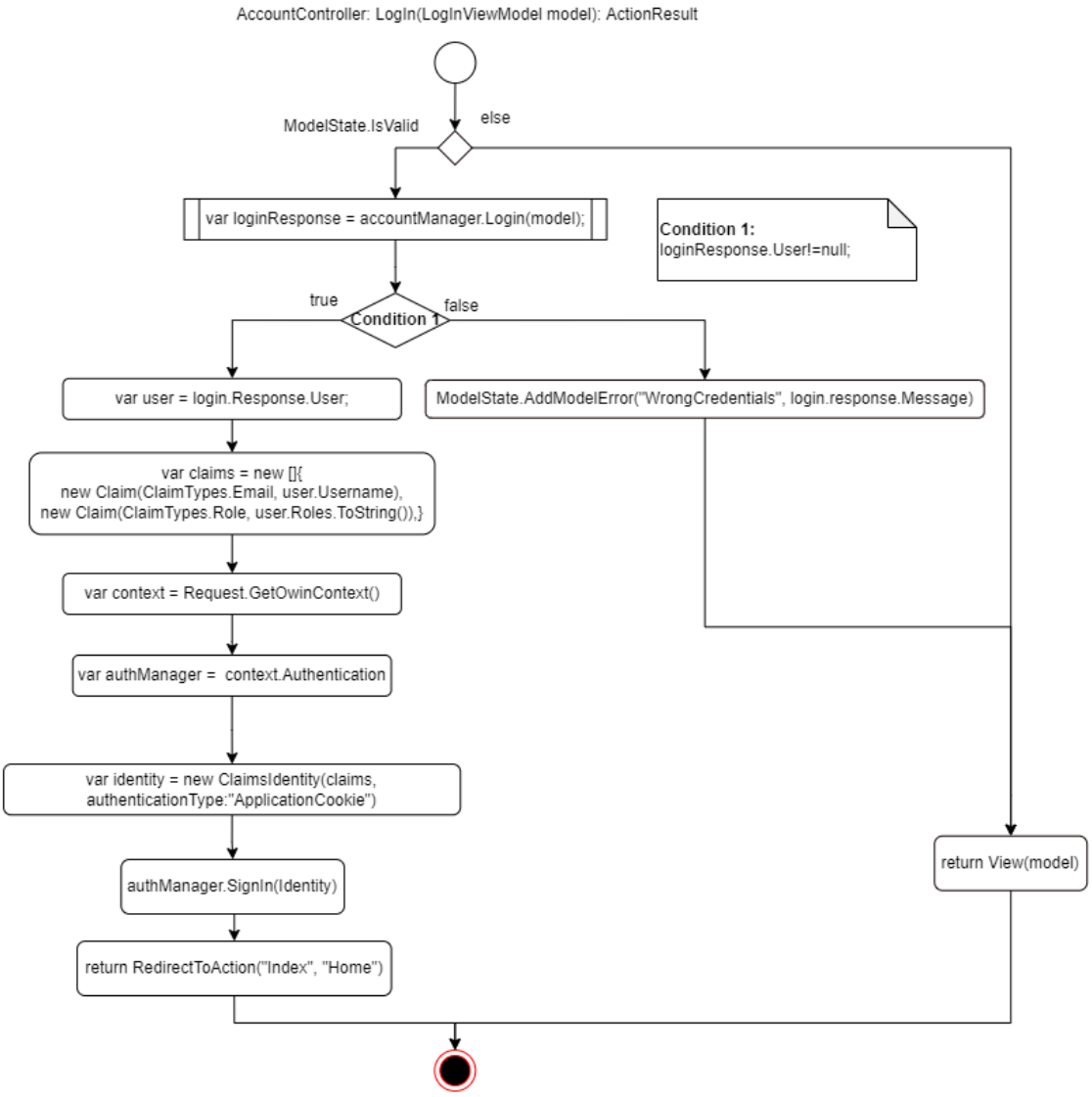

Рисунок 3.26 – Діаграма активності для методу аутентифікації користувача

Бізнес-логіка викликається через метод контролера, реалізацію якого наведено на рисунку 3.27.

```
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(identity);
                logger.LogInformation("User with ID={@userId} logged in.",
AuthenticationManager.User.Identity.GetUserId());
                return RedirectToAction("Index", "Home");
            }
            ModelState.AddModelError("WrongCredentials", loginResponse.Message);
        }
        return View(model);
    }
```

Рисунок 3.27 – Реалізація методу Login

Головною частиною методу є виклик методу SingIn з AuthenticationManager, що є частиною OWIN. До цього методу мі передаємо Identity, що містить інформацію про користувача зашиту у Claims, в часності це ідентифікатор користувача та його ролі, що є визначними для надання доступу до системи.

3.4.2 Механізм реєстрації користувача

Згідно архітектури, що реалізується в системі для механізму аутентифікації для реалізації механізму реєстрації визначено такі шари:

1. Data source – MS SQL Server та збережені процедури для зв’язку – це рівень, на якому зберігаються дані.

2. Data access layer – SqlDataAccess (Dapper) – цей рівень відповідає за зв’язок між базою даних (або іншим джерелом даних) і програмою.

3. Stores – UsersDAL – Цей рівень забезпечує інтерфейс для зв’язку з рівнем доступу до даних

4. Managers – CustomerService.BL – Цей рівень отримує дату від програми для обробки та внесення деяких змін у базу даних через магазини.

5. Asp.Net Applications – CustomerService.WebApplication – цей рівень є кінцевою точкою, яка обробляє запити від користувача та перенаправляє менеджеру.

На рисунку 3.28 наведено алгоритм реєстрації користувача.

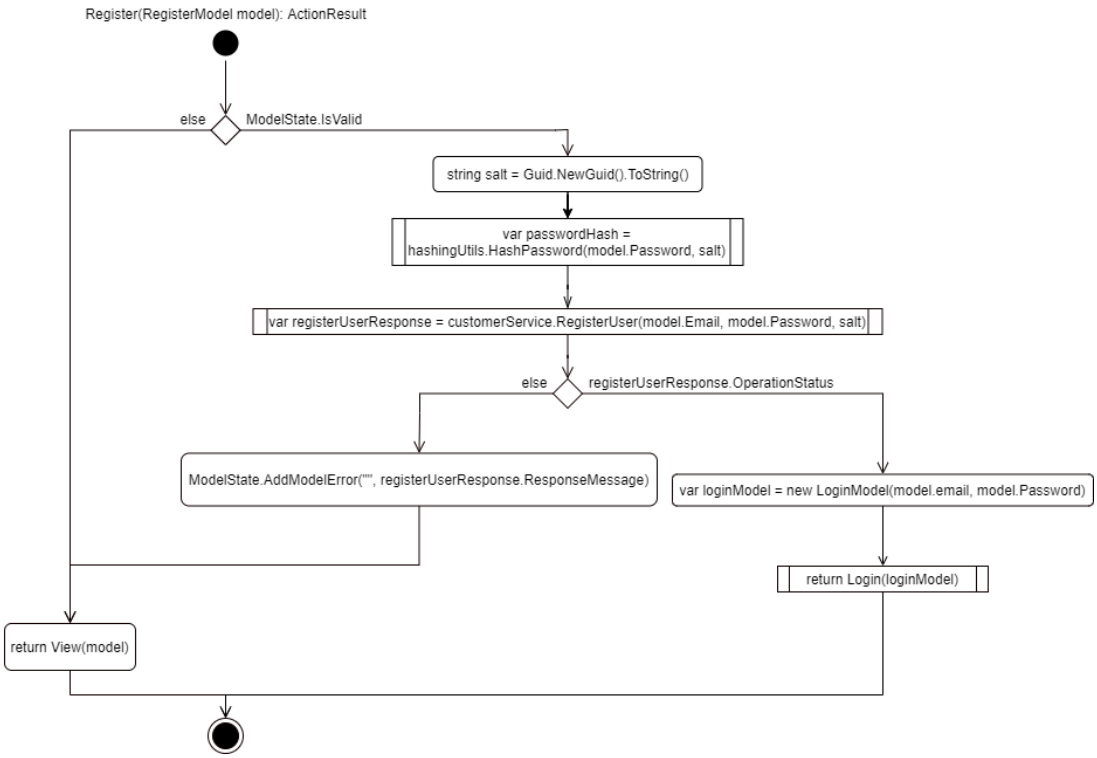

Рисунок 3.28 – Діаграма активності для механізму реєстрації користувача

3.4.3 Механізм хешування

Перед збереженням у базі даних критично важлива інформація, яка надає можливість користувачеві отримати доступ до системи повинна бути хешована. У якості алгоритму використовується відомий SHA-512, що перетворює вказані користувачем дані у 128-значний хеш.

Для більшої безпеки кожен пароль збагачується певним значенням, так звана «сіль», яка використовується для формули хешування і також зберігається у базі даних. І останній додатковий шар безпеки забезпечує певна константа, яка зберігається безпосередньо у коді і є недоступною для зловмисників. Ці дві речі ускладнять можливість знайти правильний пароль, введений користувачем.

Для обрахунку хеша пароля використовується наступна формула:

PasswordHash = sha512(“{пароль}{salt}{constant}”)

Завдяки цьому знайти правильний пароль методом підбору, знаючи хеш майже не є можливим. На рисунку 3.29 наведено алгоритми для хешування пароля.

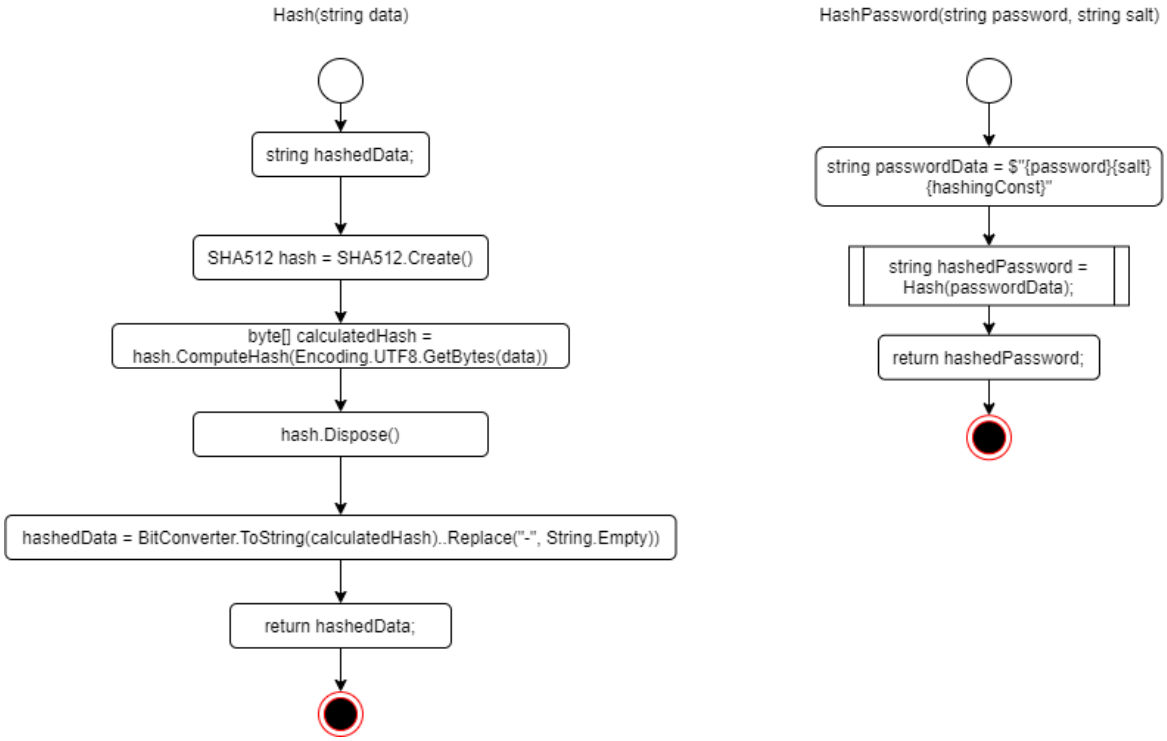

Рисунок 3.29 – Діаграма активності для процесу хешування

Хешування паролю відбувається на стороні сервера. Для сторони клієнта хешування не є досить важливим кроком і на те є певні причини.

Перш ніж надати пароль серверу, ми також можемо його хешувати. Це означає, що наша система отримує пароль як хеш і працюватиме з хешем замість реального пароля. І це також ускладнить можливість знайти правильний пароль, введений користувачем. Але це не має сенсу, тому що якщо зловмисник перехопить запит і знайде хеш пароля, він може використовувати цей хеш для зв’язку з системою, оскільки наша система використовуватиме хеш як пароль замість пароля, введеного користувачем.

Інший спосіб використання шифрування також не має сенсу, оскільки код JS є відкритим вихідним кодом на стороні клієнта, і кожен може

переглянути його та знайти алгоритм шифрування, а потім знайти використовуючи відповідний алгоритм дешифрування отримати пароль.

Найкращий спосіб зробити зв'язок між клієнтом і сервером безпечним – це застосування SSL, що робить дані запиту закритими для змін і читання у випадку перехоплення.

3.4.4 Функціональні можливості системи

Не зважаючи на важливість роботи з користувачем не менш важливою частиною клієнтського сервісу є надання доступу до функціональних можливостей системи, а саме створення транзакцій та перегляд власного гаманця. На рисунку 3.30 наведено головну сторінку електронного гаманця.

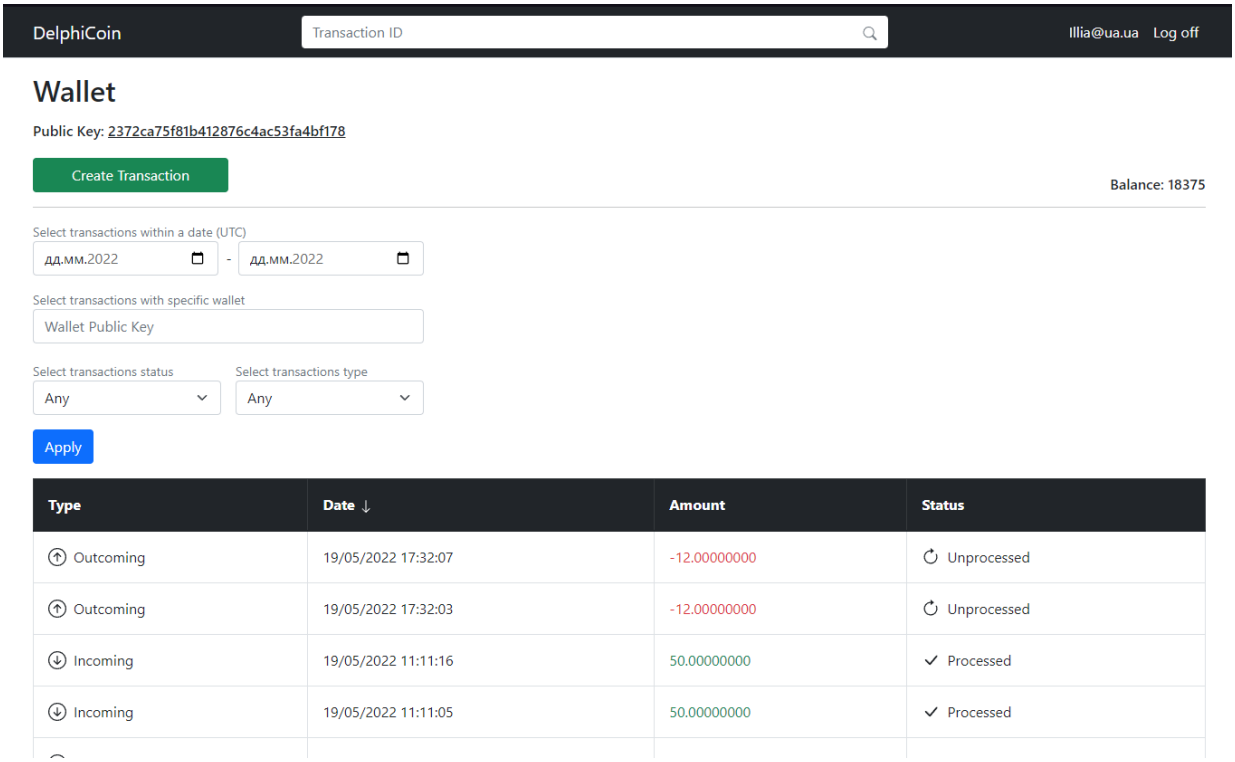

Рисунок 3.30 – Зовнішній вигляд головної сторінки гаманця

Веб-додаток пропонує кінцевому користувачеві зручну веб-сторінку, де можливо переглянути поточний баланс та всі транзакції, що були створені користувачем або для користувача. Також присутня можливість виконати фільтрацію по всім можливим параметрам транзакції, а саме публічний ключ

гаманця на який або з якого надійшла транзакція, діапазон дат коли була останній раз оновлена транзакція (створена або підтверджена), поточний статус, та напрям.

Також користувачі мають можливість переказати певну кількість криптовалюти на інші гаманці. Для цього необхідно ввести публічний ключ гаманця отримувача та суму, яка не повинна перевищувати баланс, відображений на сторінці. У випадку якщо введені некоректні дані для отримувача або суми переказу буде виведено відповідне повідомлення. На рисунку 3.31 наведено зображення сторінки створення транзакції.

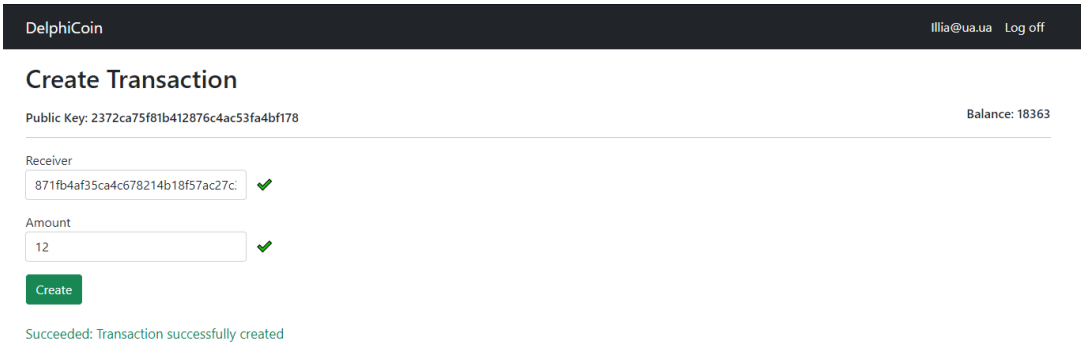

Рисунок 3.31 – Зовнішній вигляд сторінки створення нової транзакції

Головною особливістю клієнтського додатку є моніторинг змін у гаманці в режимі реального часу. Тобто якщо хтось створює транзакцію для нашого гаманця, то список транзакцій буде автоматично оновлено без перезавантаження сторінки та необхідності власноруч ініціювати запит на оновлення даних. Функціональна можливість моніторингу у режимі реального часу забезпечується за допомоги бібліотеки SignalR [20], в основі якої лежить протокол зв'язку WebSocket. Завдяки цьому між клієнтом та сервером створюється підключення, яке можливо використовувати для відправки запитів у двох напрямках. Це дозволяє на стороні сервера при будь-яких змінах у гаманцях користувачів відправляти повідомлення про це відповідним користувачам, які в цей момент мають активну сесію. Після чого, клієнтський додаток може певним чином відреагувати на це

повідомлення. В нашому випадку клієнтський додаток виконує асинхронний запит за допомогою AJAX до сервера та актуалізую дані без необхідності перезавантажити сторінку у браузері, що відтворює ці дані.

3.5 Розробка сервісу для емісії криптовалюти

Консольний клієнт забезпечує життєдіяльність транзакції та системи в цілому, оскільки створення нових блоків та емісія криптовалюти є головною задачею сервісу.

Для початку роботи клієнт підписується на сервіс транзакцій для можливості отримання актуальних даних для обрахунку блоку після чого ініціюється процес «майнінгу». На рисунку 3.32 наведено алгоритм процесу майнінгу.

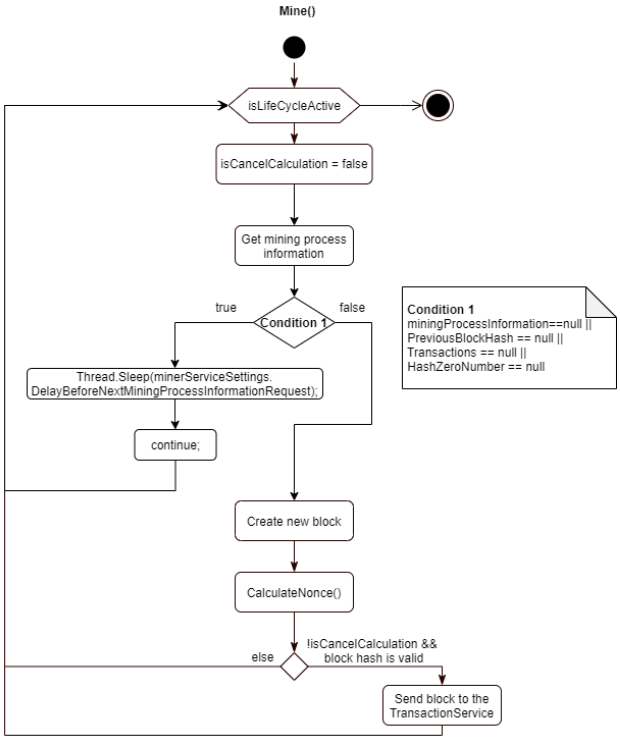

Рисунок 3.32 – Діаграма активності для процесу майнингу

Загальний алгоритм роботи сервісу такий:

- Під час «майнингу» клієнт робить запит до сервісу транзакцій для отримання даних для нового блоку. Якщо сервіс транзакцій в цей момент

виконує валідацію, отримання даних для блоку призупиняється до завершення валідації.

- 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.

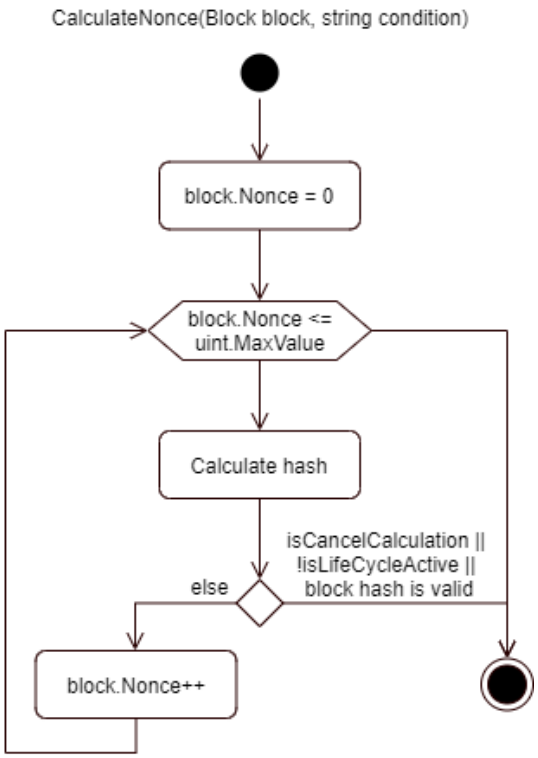

Рисунок 3.33 – Діаграма активності для процесу хешування блоку

- Після успішного знаходження хешу для блока, виконується запит до сервера у якому пересилається обрахований блок для валідації.
- Після відправлення запиту процес циклічно продовжується для кожного блоку.

Процес обрахунку блоків є повністю автономним і не потребує втручання користувача окрім випадку коли є потреба завершити роботу

додатку. На рисунку 3.34 наведено вікно додатку з запущеним процесом обрахунку хеша блока.

Рисунок 3.34 – Зовнішній вигляд консольного додатку

Як видно з рисунку додаток від сервера певний перелік даних, серед яких хеш останнього блоку доданого до системи, транзакції, та умова хешування (в даному випадку хеш повинен починатись із «00000»). Ці дані використовуються у процесі обрахунку після чого ми отримуємо хеш, як виведено у полі «Calculated hash».

Для можливості підтримувати зворотній зв’язок із сервісом транзакцій було розгорнуто WebAPI на консольному клієнті [21]. Для цього на початкових етапах запуску додатку створюється хост, для якого вказується локальна IP-адреса. Реалізація методу створення хоста наведено на рисунку3.35.

```
public static IHostBuilder CreateHostBuilder(IConfiguration configuration) =>
   Host.CreateDefaultBuilder()
      .UseSerilog()
      .ConfigureWebHostDefaults(webBuilder =>
      {
         webBuilder.UseUrls(GetLocalUrl(configuration));
         webBuilder.UseStartup<Startup>();
               }).ConfigureServices(services
   => services.AddHostedService<BackgroundComandHandler>());
```

Рисунок 3.35 – Реалізація методу CreateHostBuilder

В якості зворотнього зв’язку сервіс транзакцій відправляє два типи запиту всім підключеним клієнтам:

- Запит на перевірку працездатності. Якщо клієнт отримав цей запит, він просто відправляє відповідь, що зараз є активним і обраховує блоки. Сервер у випадку, якщо не отримує відповіді, вважає, що клієнт не є доступним і відключає його. Коли клієнт повернеться до роботи – просто відправить додатковий запит на підключення. Реалізацію методу контролера, що викликається сервером наведено на рисунку 3.36.

```
[HttpPost]
public ResponseResult IsAvailable(ObserverAddress observerAddress)
{
    return observableServiceProxyControllerBL.IsAvailable(observerAddress);
}
```

Рисунок 3.36 – Реалізація методу контролера для перевірки працездатності клієнта

- Запит на відміну поточного обрахунку блока. У випадку коли було додано новий валідний блок до системи сервер відправляє кожному клієнту запит на відміну поточного обрахунку, оскільки дані, з якими обраховуються блоки вже не актуальні. Реалізацію методу контролера, що викликається сервером наведено на рисунку 3.37.

```
[HttpPost]
public void Notify(ObserverNotifyInfo observerNotifyInfo)
{
    observableServiceProxyControllerBL.Notify(observerNotifyInfo);
}
```

Рисунок 3.37 – Реалізація методу контролера для відміни поточного обрахунку блока

Бізнес-логіка, що відповідає за відміну поточного обрахунку наведена нижче. У операторі Switch просто викликається метод DropBlock(), що і скидує поточний обрахунок, зміною спеціального прапора, значення якого активно відстежується під час обрахунку. Реалізацію методу для обробки події наведено на рисунку 3.38.

```
internal void OnNotify(ObserverNotificationArgs args)
    {
        switch (args.EventName)
        {
            case "DropBlock":
                {
                    DropBlock();
                    break;
                }

            default:
                break;
        }
}
```

Рисунок 3.38– Реалізація методу бізнес-логіки, що виступає в ролі узагальненого обробника події відміни обрахунку блока

3.6 Висновки

У третьому розділі було коротко розглянуто основні інструменти для розробки власної криптовалюти, а саме для розробки основної бізнес-логіки та серверної частини обрано платформу .Net та мову C#. Опираючись на мову програмування, було обрано середовище розробки Visual Studio. Опираючись на тісну інтеграцію з платформою .NET, було обрано базу даних MS SQL Server. Для взаємодії із базою даних було обрано micro-ORM Dapper за її зручність та продуктивність роботи із збереженими процедурами. Для роботи із веб клієнтом було обрано розробку за допомогою ASP .NET, що пропонує зручні інструменти при роботі із даними. Для створення користувацького інтерфейсу було обрано CSS-бібліотеку Bootstap.

Було розглянуто розробку модулів для комунікації, що передбачали створення базових методів виклику створених процедур та створення вискорівневих модулів для використання іншими сервісами.

Було детально розглянуто процеси які відбуваються поза API яку викликають клієнтські сервіси. Наведено опис та фрагменти імплементації для сервісу транзакцій, що включає в себе безпосередньо роботу з транзакціями, їх створення, зберігання та підтвердження. Також описано роботу високорівневих процесів бізнес-логіки в яких транзакція приймає участь.

Результатом роботи над сервісом для користувачів є веб-додаток, що дозволяє користувачеві після реєстрації отримати доступ до свого гаманця. Користувач має змогу переглядати свою транзакції у режимі реального часу та створювати нові за умові достатнього балансу.

Результатом роботи над сервісом для емісії криптовалюти є консольний додаток, що підтримує життєвий цикл системи, підтверджуючи неопрацьовані транзакції, шляхом упакування їх у блоки, які хешуються спеціальним способом для подальшого зберігання.

4 ТЕСТУВАННЯ СИСТЕМИ

4.1 Юніт-тестування бізнес-логіки

Для тестування всіх модулів системи було застосовано підхід Unit-testing в комбінації з підходом TDD [22]. Платформа .Net пропонує вбудований інструмент для тестування MS Test [23]. В загальному підхід до тестування кожного модуля окремо передбачає незалежність від зовнішніх викликів. Для слідування цьому підходу використано бібліотеку NSubstitute [24], 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, 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. В даному випадку це 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. Цей інструмент динамічно створює об’єкт по шаблону, описаному інтерфейсом. Об’єкт створений через NSubstitute обмежений лише інтерфейсом і не може бути приведений до інших типів, як у випадку використання реальних об’єктів. Але це і не потрібно. Натомість NSubstitute пропонує широкий спектр можливостей, наприклад можна відстежити чи було викликано певний метод, кількість викликів, параметри, з якими було викликано, виконати певну дію під час виклику, запрограмувати результат, який повинен повернути метод, тощо.

Отже для тесту було створено імітації залежностей, з якими і створено об’єкт TransactionService для якого викликається метод CreateWallet на етапі Act. Після чого ми отримуємо результат і на етапі Assert виконуємо необхідні перевірки, а саме чи результат є коректним і не рівним null, і також на одній із залежностей за допомогою можливостей NSubstitute перевіряємо чи біло

викликано метод додавання нового гаманця до бази даних через AddWallet метод IWalletsDAL інтерфейсу. На рисунку 4.1 зображено реалізацію опи саного вище тесту.

```
public void TestWalletCreated()
{
    // arrange
    var settings = Substitute.For<ISettings>();
    var locker = Substitute.For<IMonitorLocker>();
    var blockCalculationUtils = Substitute.For<IBlockCalculationUtils>();
    var guidGenerator = new GUIDGenerator();
    var rwLocker = Substitute.For<IReaderWriterLocker>();
    var transactionsDAL = Substitute.For<ITransactionsDAL>();
    var walletsDAL = Substitute.For<IWalletsDAL>();
    walletsDAL.AddWallet(Arg.Any<Wallet>()).Returns(true);
    var blocksDAL = Substitute.For<IBlocksDAL>();
    var logger = Substitute.For<ILogger<TransactionService>>();
    var transactionService = new TransactionService(settings, locker, blockCalculationUtils, guidGenerator,
rwLocker, transactionsDAL, walletsDAL, blocksDAL, logger);
    string walletPK;

    // act
    walletPK = transactionService.CreateWallet(1);
    ITransactionServiceTestable transactionServiceTestable = transactionService;
    var wallets = transactionServiceTestable.Wallets;

    // assert
    Assert.AreNotEqual(walletPK, string.Empty);
    Assert.IsTrue(wallets.ContainsKey(walletPK));

    walletsDAL.Received().AddWallet(wallets[walletPK]);
```

Рисунок 4.1 – Реалізація тесту для методу створення гаманця у випадку успішного створення

Розглянемо інший приклад для тестування методу відправки GET-запиту по вказаному шляху. В цьому випадку тест повинен виконати перевірку параметрів під час використання SendAsync методу. Для цього замінюється базова імплементація на виконання анонімного методу який виконує низку перевірок, а саме шлях по якому необхідно виконати запит у випадку використання реального об’єкту, тип запиту, заголовки, а саме HMAC заголовок, що дозволяє забезпечити цілісність запиту, та в кінці сам вміст запиту, тобто дані, що передаються. На рисунку 4.2 зображено реалізацію описаного вище тесту.

```
[TestMethod]
public async Task SendAsyncTest_GetMethod()
{
    var hmacHashingUtils = Substitute.For<IHMACHashingUtils>();
    hmacHashingUtils.GetHMAC(Arg.Any<string>()).Returns("hmac");
    var httpClient = Substitute.For<HttpClient>();
    var hmacHttpClient = new HMACHttpClientWrapper(hmacHashingUtils, httpClient);

    var queryString = "?parameter=1";
    string uri = "http://localhost/api/getwallet" + queryString;
    httpClient.When(a => a.SendAsync(Arg.Any<HttpRequestMessage>(), Arg.Any<CancellationToken>()))
        .Do(param =>
        {
            Assert.AreEqual(uri, param.ArgAt<HttpRequestMessage>(0).RequestUri.ToString());
            Assert.AreEqual(HttpMethod.Get, param.ArgAt<HttpRequestMessage>(0).Method);
            Assert.IsTrue(param.ArgAt<HttpRequestMessage>(0).Headers.Contains("HMAC"));
            Assert.AreEqual("hmac",
param.ArgAt<HttpRequestMessage>(0).Headers.GetValues("HMAC").First());
            Assert.IsNull(param.ArgAt<HttpRequestMessage>(0).Content);
        });
    var result = await hmacHttpClient.GetAsync(uri);

    await httpClient.Received().SendAsync(Arg.Any<HttpRequestMessage>(),
Arg.Any<CancellationToken>());
    hmacHashingUtils.Received().GetHMAC(queryString);
```

Рисунок 4.2 – Реалізація тесту для методу відправки GET-запиту у випадку успішного виконання

4.2 Тестування веб-додатку для роботи з електронним гаманцем

Для тестування веб-системи було використано хостинг додатку на IIS [25]. Після чого ми за вказаною при налаштуваннях адресою маємо можливість отримати доступ до нашого додатку. При відкритті додатку в перший раз нас буде переправлено на початкову сторінку із привітанням. Тут ми маємо можливість зареєструватись або увійти у систему, якщо вже маємо власний акаунт. На рисунку 4.3 зображено початкову сторінку сервісу для роботи з електронним гаманцем.

Рисунок 4.3 – Початкова сторінка

При переході до реєстрації нам відриється сторінка де ми маємо ввести електронну пошту та придумати пароль. На рисунку 4.4 зображено сторінку реєстрації користувача.

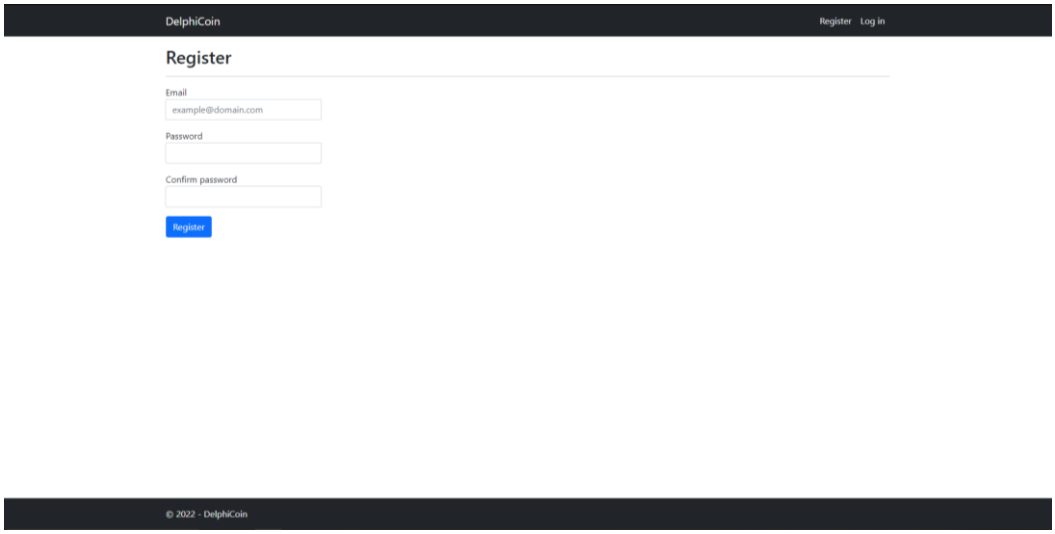

Рисунок 4.4 – Сторінка реєстрації

У випадку якщо користувач вводить дані що не вважається системою валідними, кожне поле повідомить про помилку введення. На рисунку 4.5 зображено сторінку реєстрації користувача при введенні невалідних даних.

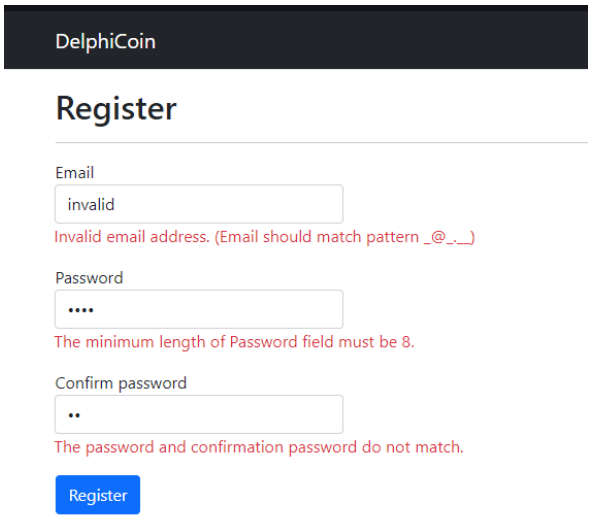

Рисунок 4.5 – Сторінка реєстрації при введенні некоректних даних

У випадку коли користувач вже зареєстрований у системі необхідно використати іншу сторінку, де потрібно ввести дані від власного акаунту. На рисунку 4.6 зображено сторінку аутентифікації користувача.

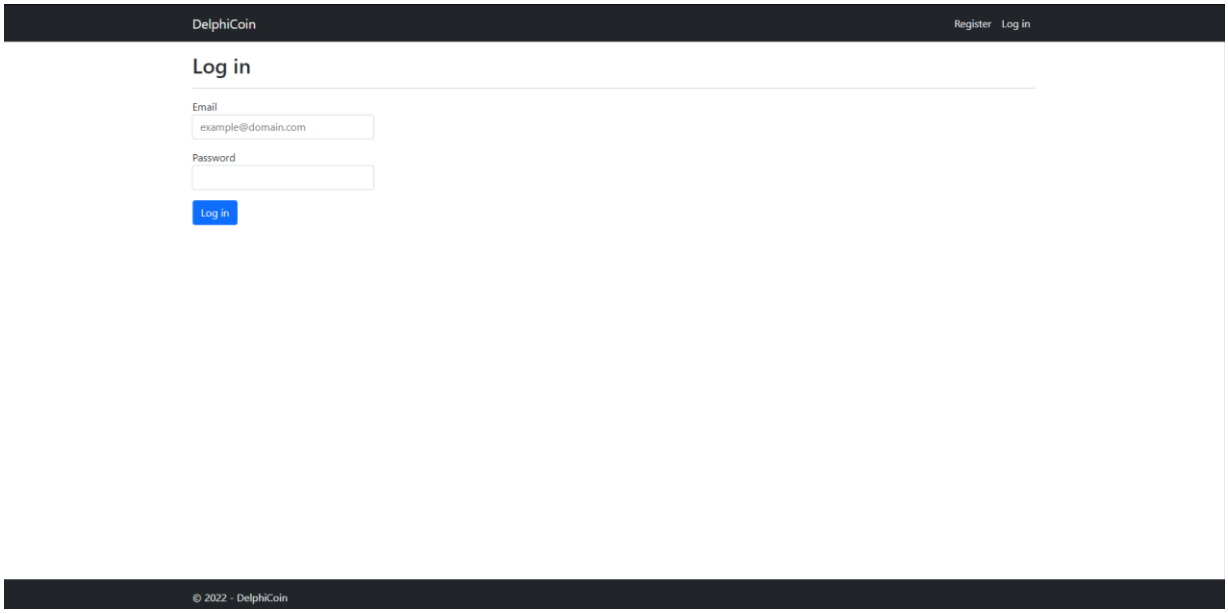

Рисунок 4.6 – Сторінка входу в акаунт

У випадку якщо користувач вводить дані що є помилковими система повідомить що такого користувача не існує і це означає, що користувач помилився і повинен більш уважно вводити дані. На рисунку 4.7 зображено сторінку аутентифікації користувача при введенні недійсних облікових даних.

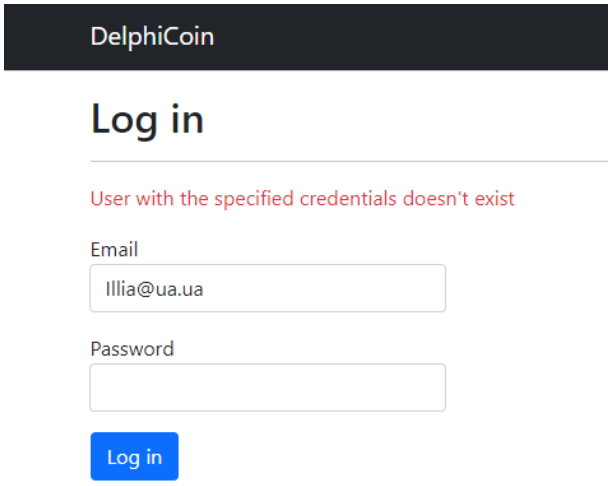

Рисунок 4.7 – Сторінка входу в акаунт при введенні неправильних даних

При введенні коректних даних від акаунту користувача буде перенаправлено на сторінку його гаманця, де він має можливість переглянути всі транзакції, відфільтрувати їх, або перейти до перегляду конкретної транзакції ввівши її ідентифікатор у відповідне поле, або перейти до створення нової транзакції. На рисунку 4.8 зображено головну сторінку електронного гаманця.

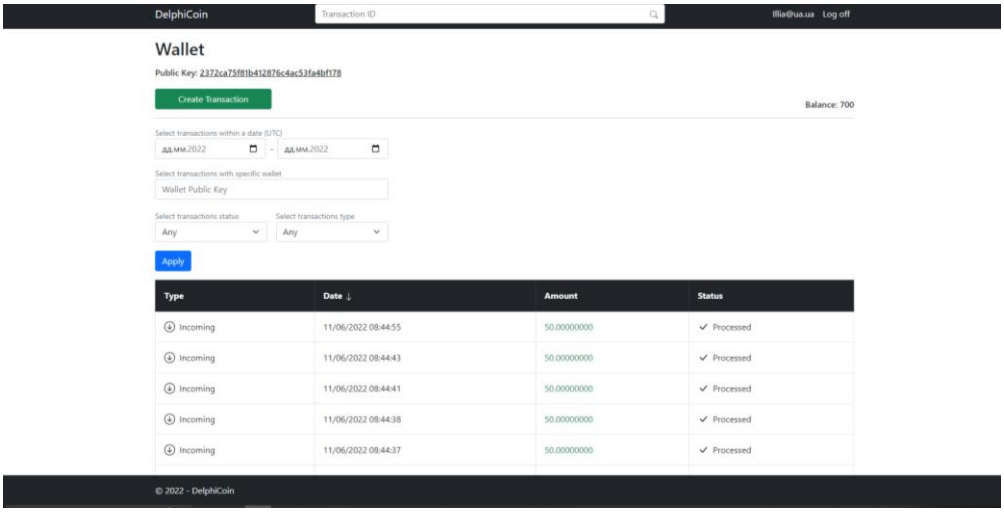

Рисунок 4.8 – Сторінка електронного гаманця користувача

При переході до створення транзакції користувачеві доступно 2 поля, для введення гаманця отримувача та суми переказу. Якщо дані введено помилково, буду виведено повідомлення про це. На рисунку 4.9 зображено сторінку створення транзакції.

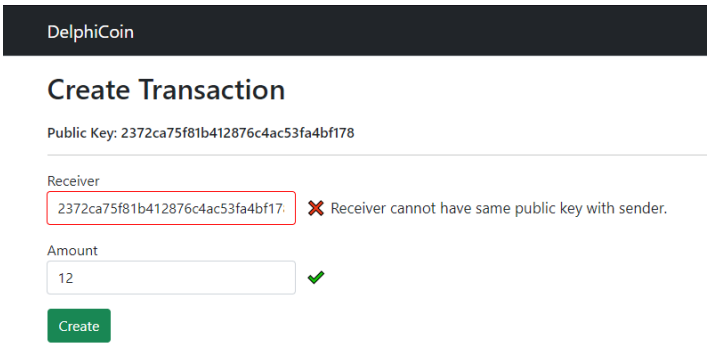

Рисунок 4.9 – Сторінка створення транзакція на неправильний гаманець

При успішному переказі отримаємо відповідне повідомлення про успіх операції. На рисунку 4.10 зображено сторінку створення транзакції при введені коректних даних.

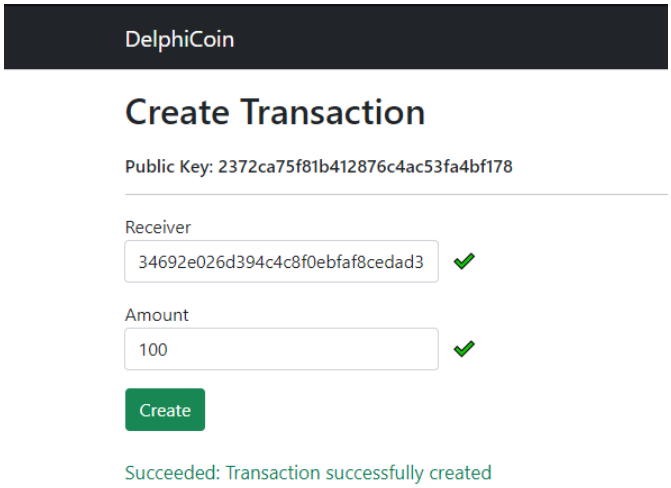

Рисунок 4.10 – Сторінка створення транзакція з правильними даними

При спробі відфільтрувати дані по статусу отримаємо лише одну неопрацьовану, щойно створену транзакцію. На рисунку 4.11 зображено фільтрацію транзакцій по статусу.

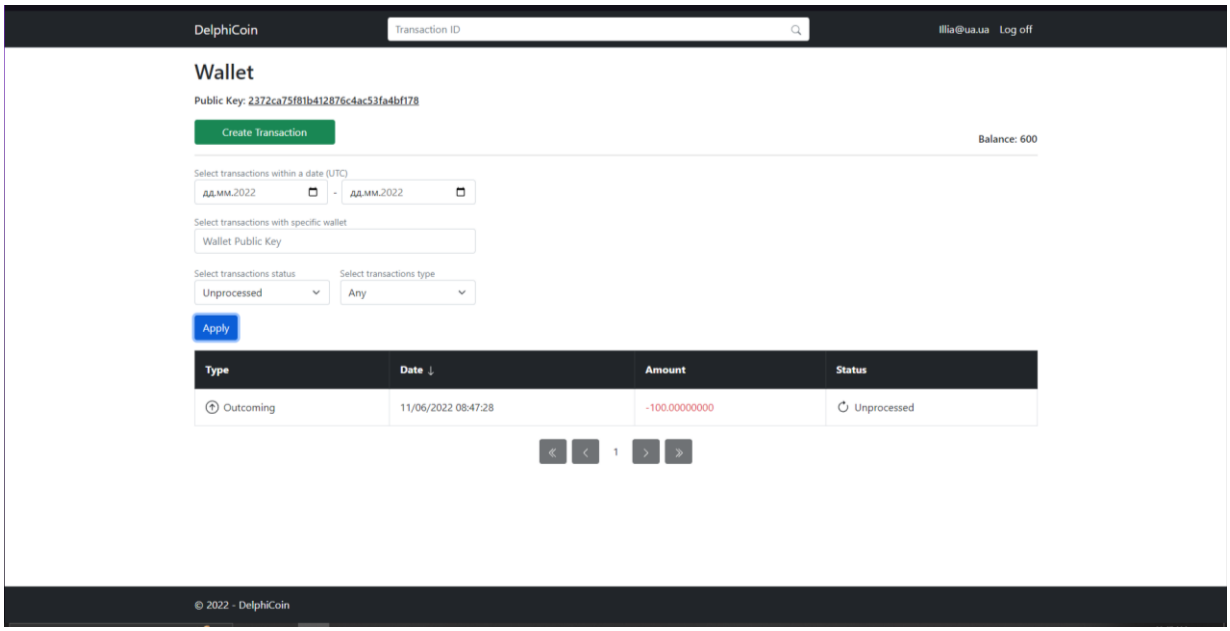

Рисунок 4.11 – Фільтрація транзакцій по статусу

Для пошуку транзакції вводимо ідентифікатор у відповідне поле у шапці сторінки та на буде перенаправлено на окрему сторінку, де виведено всю детальну інформацію про транзакцію, якщо вона існує. На рисунку 4.12 зображено сторінку з детальною інформацією про транзакцію.

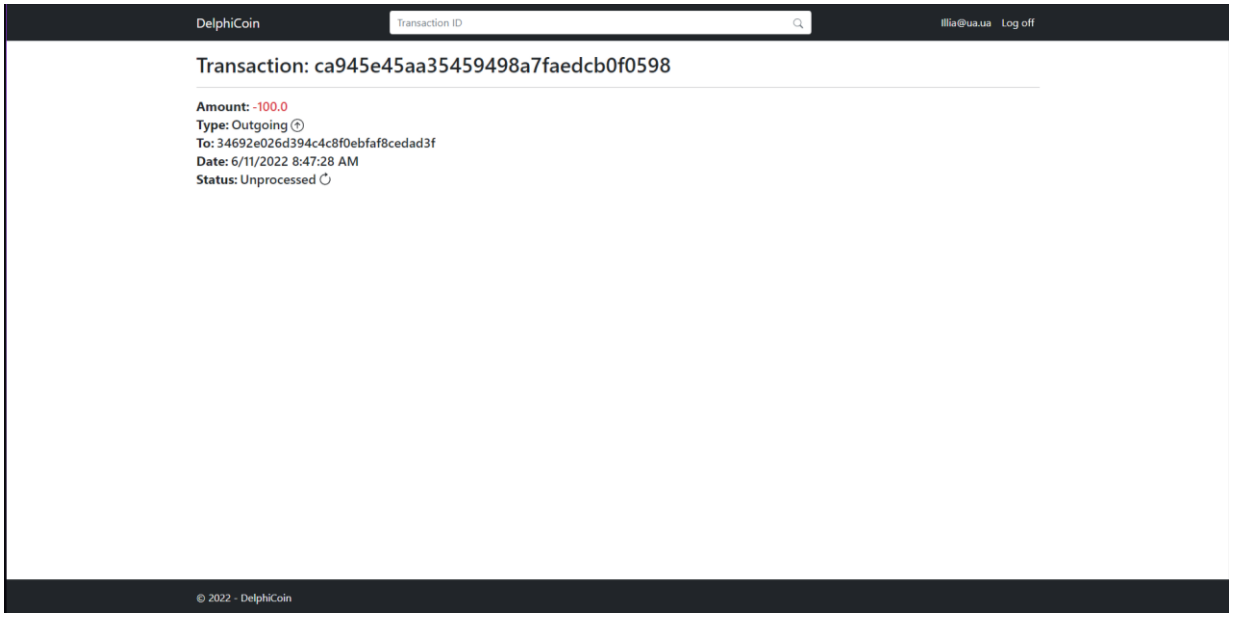

Рисунок 4.12 – Перегляд конкретної транзакції по її ідентифікатору

4.3 Розробка інструкції користувача сервісу для емісії криптовалюти

Перший запуск:

1. Перейдіть до папки з встановленою програмою
2. Відкрийте файл «appsettings.json»
3. Знайдіть розділ «Інформація про користувача»
4. Введіть свій власний UserID замість #
5. Збережіть зміни та закрийте файл.
6. Програма готова до запуску.
7. Просто запустіть файл .exe, і програма прочитає ваш ідентифікатор користувача з файлу .config

Другий та наступні запуски – просто запустіть файл .exe, і програма прочитає ваш ідентифікатор користувача з файлу .config

Важливі «гарячі клавіші»:

- Ctrl+Q – завершує процес майнінгу та закриває програму.
- Ctrl+D – перериває процес майнінгу.
- Ctrl+E – продовжує перерваний процес майнінгу.

Щодо помилок – найпопулярніші помилки стосуються проблеми з конфігураційним файлом.

Найкраще рішення:

1. Перейдіть до офіційного дистриб'ютора програми
2. Завантажте конфігураційний файл
3. Замініть новий файл наявним у кореневій папці програми.
4. Повторіть інструкцію «Перший запуск».

Важливо: якщо ви помітили виняток, не пов’язаний з файлом конфігурації, зробіть скріншот і зв’яжіться зі службою підтримки.

4.4 Висновки

У четвертому розділі було розглянуто юніт-тестування бізнес-логіки додатку. Розглянуто інструмент NSubstitute, що полегшує процес написання юніт-тестів. Проведено тестування сервісу для роботи з електронним гаманцем, розміщеного на IIS. Написано інструкцію користувача по роботі з сервісом для емісії криптовалюти.

ВИСНОВКИ

У бакалаврській дипломній роботі було розроблено власну реалізацію централізованої криптовалюти. Для розробки було використано середовище програмної розробки Microsoft Visual Studio 2022.

Під час виконання бакалаврської дипломної роботи було проаналізовано поточний стан криптовалют. Було розглянуто основні аналоги криптовалют, як Bitcoin, Ethereum, Litecoin. Було визначено їхні недоліки та порівняно з власним програмним продуктом. Беручи до уваги результати порівняння було встановлено основні задачі бакалаврської дипломної роботи.

Під час аналізу технологій розробки було обґрунтовано вибір платформи .Net та мови C# для розробки бізнес-логіки та серверної частини. Для розробки веб-додатку для роботи з електронним гаманцем було використано платформу Asp .Net Framework. Для створення користувацького інтерфейсу було застосовано CSS-бібліотеку Bootstrap. Для збереження даних було обрано MS SQL Server через тісний зв’язок із платформою .Net. Для доступу у до бази даних було обрано micro-ORM Dapper, базуючись на аналізі підходів для роботи із БД.

У бакалаврській дипломній роботі було зроблено:

– розроблено моделі платіжної системи;
– розроблено базу даних для зберігання інформації системи;
– розроблено сервіс для обробки та валідації даних про криптовалюту;
– розроблено сервіс для створення транзакцій та перегляду історії транзакцій клієнтів;
– розроблено сервіс для емісії криптовалюти;
– проведено тестування програмних модулів системи.

Було розроблено модель життєвого циклу транзакції, що функціонує у системі. Розроблено підхід до хешування даних, з використання алгоритмів

хешування. Розроблено алгоритми забезпечення секретності даних, пов’язаних з обліковими записами користувачів.

Тестування системи через юніт-тести довело працездатність кожного модуля. В загальному було написано 566 тестів. Також проведено інтеграційне тестування сервісів, що довело працездатність системи. Також створено інструкцію користувача по роботі з додатком для емісії криптовалюти.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

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

2. Payment types [Електронний ресурс] – Режим доступу до ресурсу: https://tfig.unece.org/contents/payments-types.htm

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

4. 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

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

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

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

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

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

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

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

12. Dapper [Електронний ресурс] – Режим доступу до ресурсу: https://www.learndapper.com

13. LINQ [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/csharp/linq/

14. 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

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

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

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

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

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

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

21. Self-hosting [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/aspnet/web-api/overview/older-versions/self-host-a-web-api

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

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

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

25. IIS [Електронний ресурс] – Режим доступу до ресурсу: https://www.iis.net/overview

ДОДАТКИ

Додаток А – Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ

д.т.н., проф. О. Н. Романюк

31 березня 2022 р.

Технічне завдання

на бакалаврську дипломну роботу «Розробка програмних додатків для емісії та переказу криптовалюти»

за спеціальністю

121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:

_____ доцент, к.т.н. Катєльніков Д.І

31 березня 2022 р.

Виконав:

_____студент гр. 1ПІ-19мс2 І.С. Мельник

31 березня 2022 р.

Вінниця – 2022 року

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка програмних додатків для емісії та переказу криптовалюти».

Галузь застосування – веб-технології.

2. Підстава для розробки.

Завдання на роботу, яке затверджене на засіданні кафедри програмного забезпечення – протокол № 12 від « 07 » лютого 2022 р.

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..

Призначення роботи – можливості застосування у повсякденному житті у якості платіжного ресурсу.

4. Вихідні дані для проведення НДР

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

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

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

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

5. Технічні вимоги

Вхідні дані – введення даних користувачем для створення транзакції; вихідні дані – оновлення списку транзакцій з подальшою зміною статусу піля опрацювання.

6. Конструктивні вимоги

Конструкція програми повинна відповідати структурним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

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

– пояснювальна записка до бакалаврської дипломної роботи;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

Додаток Б – Протокол перевірки на плагіат

ПРОТОКОЛ
ПЕРЕВІРКИ БАКАЛАВРСЬКОЇ ДИПЛОМНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: «Розробка програмних додатків для емісії та переказу криптовалюти»

Тип роботи: БДР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: к.т.н., доц. каф. ПЗ Катєльніков Д.І.

Оригінальність	98,2%
Схожість	1,8%

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 Ілля Сергійович

Керівник роботи _____ Катєльніков Денис Іванович

Додаток В – Лістинг програми

Cryptocurrency.TransactionService.BL

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.Interfaces;
using CryptoCurrency.Common.Models;
using CryptoCurrency.DAL.Interfaces;
using CryptoCurrency.TransactionService.Interfaces;
using CryptoCurrency.TransactionService.Models;
using Microsoft.Extensions.Logging;

namespace CryptoCurrency.TransactionService.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<TransactionService>, IWalletContainer
   {
      private const int TotalCoinsLimit = 44500000;
      private readonly long? systemWalletId = null;

      /// <summary>
      /// Stores the pool of unprocessed transactions.
      /// </summary>
      private readonly SortedConcurrentDictionary<string, 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 = "00000000000000000000000000000000";
        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">Read/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(operationResult, 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.GetNewGUID(), userId);
    if (wallets.TryAdd(wallet.PublicKey, 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(identificator))
    {
        return wallets[identificator];
    }

    return null;
}

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

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

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

```
                locker.Enter(startValidationLocker);
                if (!isValidationActive)
                {
                    isValidationActive = true;
                    locker.Exit(startValidationLocker);
                    StartValidation();
                }
                else
                {
                    locker.Exit(startValidationLocker);
                }
            }
        }

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

        /// <inheritdoc cref="IMiningServiceHost.GetMiningProcessInformation(long)"/>
        public MiningProcessInformation GetMiningProcessInformation(long userId)
        {
            string errorMessage;
            try
            {
                if (isValidationActive)
                {
                    errorMessage                                                    =
TransactionServiceOperationResultMessages.ValidationProcessIsActive;
                    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 = TransactionServiceOperationResultMessages.WalletIsNotFound;
                    return new MiningProcessInformation(null, null, null, errorMessage);
                }

                var reward = new Transaction(systemWalletPublicKey, minersWallet.PublicKey,
settings.Reward, guidGenerator.GetNewGUID());
                transactions.Add(reward);
```

```
                int transactionLimit = settings.MaxCountBlockTransactions - 1;

                transactions.AddRange(transactionsPool.GetFirstElements(transactionLimit));

                errorMessage = TransactionServiceOperationResultMessages.MiningProcessInformationPreparedSuccessfully;
                var mpi = new MiningProcessInformation(previousBlock.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.GetMiningProcessInformationAsync(long)"/>
        public Task<MiningProcessInformation> GetMiningProcessInformationAsync(long userId)
        {
            var mpi = GetMiningProcessInformation(userId);
            return Task.FromResult(mpi);
        }

        /// <inheritdoc cref="IWalletProvider.GetOrCreateWalletAsync(GetWalletModel)"/>
        public async Task<ResponseResult<WalletInfo>> GetOrCreateWalletAsync(GetWalletModel model)
        {
            var wallet = GetWallet(model.UserId);
            if (wallet == null)
            {
                wallet = new Wallet(guidGenerator.GetNewGUID(), model.UserId);
                try
                {
                    if (walletsDAL.AddWallet(wallet))
                    {
                        wallets.TryAdd(wallet.PublicKey, 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.InternalError,
                IsSuccessful = false,
            };
        }

        return await Task.FromResult(getWalletResponse);
    }

    /// <inheritdoc cref="IWalletProvider.GetBalanceAsync(GetWalletBalanceModel)"/>
    public async Task<ResponseResult<decimal>> GetBalanceAsync(GetWalletBalanceModel
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.InternalError,
                IsSuccessful = false,
            };
        }

        return await Task.FromResult(getWalletResponse);
    }
```

```
        /// <inheritdoc cref="IWalletProvider.GetWalletTransactionsAsync(GetWalletTransactionsModel)"/>
        public async Task<ResponseResult<PaginatedTransactions>> GetWalletTransactionsAsync(GetWalletTransactionsModel model)
        {
            ResponseResult<PaginatedTransactions> 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<PaginatedTransactions>
                {
                    Model = null,
                    Message = ResponseMessages.InternalError,
                    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(model.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.Incoming)
                        {
                            walletTransactions = walletTransactions.Where(t => t.Receiver == wallet.PublicKey);
```

```
                }
                else if (model.Type == TransactionsTypeFilter.Outgoing)
                {
                    walletTransactions = walletTransactions.Where(t => t.Sender == wallet.PublicKey);
                }
            }

            if (model.Status != TransactionsStatusFilter.Any)
            {
                if (model.Status == TransactionsStatusFilter.Unprocessed)
                {
                    walletTransactions = walletTransactions.Where(t => t.Status == Status.Unprocessed);
                }
                else if (model.Status == TransactionsStatusFilter.Processed)
                {
                    walletTransactions = walletTransactions.Where(t => t.Status == Status.Processed);
                }
            }

            if (model.Sort == TransactionsSort.DateDesc)
            {
                walletTransactions = walletTransactions.OrderByDescending(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).OrderByDescending(t => t.TransferAmount);
                var sendedTransactions = walletTransactions.Where(t => t.Sender == wallet.PublicKey).OrderBy(t => t.TransferAmount);
                walletTransactions = receivedTransactions.Concat(sendedTransactions);
            }
            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).OrderByDescending(t => t.TransferAmount);
                walletTransactions = sendedTransactions.Concat(receivedTransactions);
            }

            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<PaginatedTransactions>
        {
            Model = paginatedTransactions,
            Message = string.Empty,
            IsSuccessful = true,
        };
    }

    return await Task.FromResult(getWalletTransactionsResponse);
}

/// <inheritdoc cref="IWalletProvider.GetWalletTransactionAsync(GetWalletTransactionModel)"/>
public async Task<ResponseResult<GetWalletTransactionResponseModel>> GetWalletTransactionAsync(GetWalletTransactionModel model)
{
    ResponseResult<GetWalletTransactionResponseModel> 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<GetWalletTransactionResponseModel>()
        {
            Model = null,
            Message = ResponseMessages.InternalError,
            IsSuccessful = false,
        };
    }
    else
    {
        var walletTransactions = wallet.History;
        var transaction = walletTransactions.FirstOrDefault(t => t.Id == model.TransactionId);
        var responseModel = new GetWalletTransactionResponseModel()
        {
            Transaction = transaction,
            WalletPublicKey = wallet.PublicKey,
```

```
                };
                getWalletTransactionResponse = new
ResponseResult<GetWalletTransactionResponseModel>()
                {
                    Model = responseModel,
                    Message = string.Empty,
                    IsSuccessful = true,
                };
            }

            return await Task.FromResult(getWalletTransactionResponse);
        }

        /// <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(Block 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 RemoveBlockTransactionsFromTransactionsPool(Block block)
        {
            foreach (var blockTransaction in block.Transactions)
            {
                try
                {
                    transactionsPool.Remove(blockTransaction.Id);
```

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

                NotifySenderAndReceiverAboutCompletedTransaction(blockTransaction);
            }
        }

        /// <summary>
        /// Saves completed transactions in sender and receiver wallets.
        /// </summary>
        /// <param name="transaction">Transaction from valid block.</param>
        protected virtual void NotifySenderAndReceiverAboutCompletedTransaction(Transaction
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.Hash, 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.CalculateHash(block.Transactions);
                string currentBlockHash = blockCalculationUtils.CalculateHash(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.CalculateHash(block.Transactions);
            string currentBlockHash = blockCalculationUtils.CalculateHash(previous.Hash, transactionsHash, block.Nonce);

            if (previous.Hash == block.PreviousBlockHash
                && block.Hash == currentBlockHash
                && blockCalculationUtils.ValidateHash(block.Hash, settings.HashZeroNumber)
                && ValidateBlockTransactions(block))
            {
                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_ITransactionCreator.jpg)](../images/TS_ITransactionCreator.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_IWalletContainer.jpg)](../images/TS_IWalletContainer.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_IWalletProvider.jpg)](../images/TS_IWalletProvider.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}"/>.</returns>
        Task<ResponseResult<WalletInfo>> GetOrCreateWalletAsync(GetWalletModel 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(GetWalletBalanceModel 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.DependencyInjection;
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.GetCurrentDirectory())
                .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(webBuilder =>
                {
                    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<TransactionService.BL.TransactionService>();
            var observersManager = host.Services.GetService<ObserversManager>();
            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.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Swashbuckle.AspNetCore.SwaggerUI;

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(IServiceCollection services)
        {
            services.AddSignalR();
            services.AddControllers().AddNewtonsoftJson();
            services.AddSwaggerGen(options => options.OperationFilter<HMACHeaderParameterOperationFilter>());
            services.AddHttpClient<IHttpClientWrapper, HttpClientWrapper>();
            services.AddSingleton<IConfigurationManager, ConfigurationManager>()
                .AddSingleton<ISqlDataAccess, SqlDataAccess>()
                .AddSingleton<IHMACHashingUtils, HashingUtils>()
                .AddSingleton<ISettingsDAL, SettingsDAL>()
                .AddSingleton<ISettings>(sp => sp.GetRequiredService<ISettingsDAL>().GetAllSettings())
                .AddTransient<IMonitorLocker, Locker>()
                .AddSingleton<IBlockCalculationUtils, BlockCalculationUtils>()
                .AddSingleton<IGUIDGenerator, GUIDGenerator>()
                .AddTransient<IReaderWriterLocker, ReaderWriterLock>()
                .AddSingleton<ITransactionsDAL, TransactionsDAL>()
                .AddSingleton<IWalletsDAL, WalletsDAL>()
                .AddSingleton<IBlocksDAL, BlocksDAL>()
                .AddSingleton<BL.TransactionService>()
                .AddSingleton<ITransactionCreator>(sp => sp.GetRequiredService<BL.TransactionService>())
                .AddSingleton<IMiningServiceHost>(sp => sp.GetRequiredService<BL.TransactionService>())
                .AddSingleton<IWalletProvider>(sp => sp.GetRequiredService<BL.TransactionService>())
                .AddSingleton<IWalletContainer>(sp => sp.GetRequiredService<BL.TransactionService>())
                .AddSingleton<IObservableService<BL.TransactionService>>(sp =>
sp.GetRequiredService<TransactionService.BL.TransactionService>())
                .AddTransient<IHttpHelper, HttpHelper>()
                .AddTransient<ObserverSettings>()
                .AddTransient<IObserverWebProxy, ObserverWebProxy>()
                .AddTransient<ObserversManagerSettings>()
                .AddSingleton<ObserversManager>()
                .AddSingleton<IObserversManager>(sp => sp.GetRequiredService<ObserversManager>())
                .AddSingleton<IGroupConnections<WalletObserverHub>, GroupConnections<WalletObserverHub>>()
                .AddSingleton<IObservableEventHandler<TransactionServiceObserverHub>,
TransactionServiceSignalrEventHandler>()
                .AddSingleton<IObservableEventHandler<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(DocExpansion.None);
            });

            if (env.IsDevelopment())
            {
```

```
            app.UseDeveloperExceptionPage();
        }

        app.UseRouting();

        app.UseEndpoints(endpoints =>
        {
            endpoints.MapControllers();
            endpoints.MapHub<TransactionServiceObserverHub>("/transactionserviceobserverhub");
            endpoints.MapHub<WalletObserverHub>("/walletobserverhub");
        });
    }
  }
}
```

Cryptocurrency.CustomerService.BL

CustomerService.cs

```
using System;
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using System.Security.Claims;
using System.Threading.Tasks;
using CryptoCurrency.Common.Interfaces;
using CryptoCurrency.Common.Models;
using CryptoCurrency.CustomerService.Interfaces;
using CryptoCurrency.DAL.Interfaces;
using CryptoCurrency.TransactionService.Interfaces;
using Microsoft.Extensions.Logging;

[assembly: InternalsVisibleTo("UnitTesting")]
[assembly: InternalsVisibleTo("UnitTesting_Framework")]

namespace CryptoCurrency.CustomerService.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">Instance of Transaction Service for transaction creation.</param>
        /// <param name="walletProvider">Instance 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="observableServiceProvider">Instance to get observable instances to observe an events.</param>
        /// <param name="sessionHandler">A session handler to work with user sessions.</param>
        /// <param name="walletChangedEventHandler">Event handler.</param>
        public CustomerService(IWalletProvider 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.AddRoleToUser(long, byte)"/>
        public bool AddRoleToUser(long userId, byte roleId)
        {
            try
            {
                return usersDAL.AddRoleToUser(userId, roleId);
            }
            catch (ArgumentException ex)
            {
                logger.LogWarning(ex.Message);
                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.CreateTransaction(sender, receiver, amount, out response);
        }

        /// <inheritdoc cref="ICustomerService.CreateTransactionAsync(string, string, decimal)"/>
        public async Task<CreateTransactionResponse> CreateTransactionAsync(string sender, string receiver, decimal amount)
        {
            return await transactionCreator.CreateTransactionAsync(sender, receiver, amount);
        }

        /// <inheritdoc cref="IUsersManager.GetAllUsers()"/>
        public IEnumerable<User> GetAllUsers()
        {
            try
            {
                return usersDAL.GetAllUsers();
            }
            catch (Exception ex)
            {
                logger.LogError(ex, ex.Message);
                return Array.Empty<User>();
            }
        }

        /// <inheritdoc cref="IWalletManager.GetBalanceAsync(GetWalletBalanceModel)"/>
        public async Task<ResponseResult<decimal>> GetBalanceAsync(GetWalletBalanceModel model)
        {
            return await walletProvider.GetBalanceAsync(model);
        }

        /// <inheritdoc cref="IWalletManager.GetOrCreateWalletAsync(GetWalletModel)"/>
        public async Task<ResponseResult<WalletInfo>> GetOrCreateWalletAsync(GetWalletModel model)
        {
            var responseResult = await walletProvider.GetOrCreateWalletAsync(model);
            return responseResult;
        }

        /// <inheritdoc cref="IWalletManager.GetWalletTransactionAsync(GetWalletTransactionModel)"/>
        public async Task<ResponseResult<GetWalletTransactionResponseModel>> GetWalletTransactionAsync(GetWalletTransactionModel model)
        {
            var responseResult = await walletProvider.GetWalletTransactionAsync(model);
            return responseResult;
        }
```

```
        /// <inheritdoc cref="IWalletManager.GetWalletTransactionsAsync(GetWalletTransactionsModel)"/>
        public async Task<ResponseResult<PaginatedTransactions>> GetWalletTransactionsAsync(GetWalletTransactionsModel model)
        {
            var responseResult = await walletProvider.GetWalletTransactionsAsync(model);
            return responseResult;
        }

        /// <inheritdoc cref="IAccountManager.Login(string, string)"/>
        public LoginResponse Login(string email, string password)
        {
            var user = usersDAL.GetUserByEmail(email);
            if (user != null)
            {
                if (hashingUtils.ValidatePassword(password, user.Salt, user.PasswordHash))
                {
                    if (user.Roles.HasFlag(Roles.Customer))
                    {
                        if (sessionHandler.GetSessionsCount(user.UserId.ToString()) == 0)
                        {
                            var parameters = new Dictionary<string, string> { { "userid", $"{user.UserId}" } };
                            var walletService = observableServiceProvider.Get(WalletServiceObservableName, parameters);
                            walletService.Notify += OnNotify;
                        }

                        sessionHandler.AddSession(user.UserId.ToString());
                        var claims = new[]
                        {
                        new UserClaim(ClaimTypes.NameIdentifier, 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.AccountBlocked);
                }
            }

            return new LoginResponse(null, ResponseMessages.WrongCredentials);
        }

        /// <inheritdoc cref="IAccountManager.Logout(string)"/>
        public void Logout(string userId)
        {
            var userSessionsCount = sessionHandler.GetSessionsCount(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.RegisterUser(string, string)"/>
    public RegisterUserResponse RegisterUser(string username, string password)
    {
        RegisterUserResponse registerUserResponse;

        try
        {
            logger.LogInformation("Received request to register user.");
            var salt = guidGenerator.GetNewGUID();
            var passwordHash = hashingUtils.HashPassword(password, 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.SuccessfullyRegistered);
            }
            else
            {
                logger.LogError("AddUser didn't make any changes in DB, and didn't throw an
exception");
                registerUserResponse = new RegisterUserResponse(false,
ResponseMessages.InternalError);
            }
        }
        catch (Exception ex)
        {
            if (ex.Message.Contains("UqUser_Username"))
            {
                registerUserResponse = new RegisterUserResponse(false,
ResponseMessages.UserAlreadyExists);
            }
            else if (ex.Message.Contains("CkUser_Username_IsEmail") ||
ex.Message.Contains("column 'Username'. Truncated"))
            {
                registerUserResponse = new RegisterUserResponse(false,
ResponseMessages.InvalidEmail);
            }
            else if (ex.Message.Contains("CkCredentials_Salt_Length") ||
ex.Message.Contains("column 'Salt'. Truncated"))
            {
                logger.LogError("AddUser violated salt length constraint in DB.");
                registerUserResponse = new RegisterUserResponse(false,
ResponseMessages.InternalError);
            }
            else if (ex.Message.Contains("CkCredentials_PasswordHash_Length") ||
ex.Message.Contains("column 'PasswordHash'. Truncated"))
            {
                logger.LogError("AddUser violated password length constraint in DB.");
```

```
                registerUserResponse = new RegisterUserResponse(false,
ResponseMessages.InternalError);
            }
            else
            {
                logger.LogError(ex, "Something went wrong in attempt to register user");
                registerUserResponse = new RegisterUserResponse(false,
ResponseMessages.InternalError);
            }
        }

        return registerUserResponse;
    }

    /// <inheritdoc cref="IUsersManager.RemoveRoleFromUser(long, byte)"/>
    public bool RemoveRoleFromUser(long userId, byte roleId)
    {
        try
        {
            return usersDAL.RemoveRoleFromUser(userId, roleId);
        }
        catch (ArgumentException ex)
        {
            logger.LogWarning(ex.Message);
            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(ObserverNotificationArgs args)
    {
        switch (args.EventName)
        {
            case "WalletChanged":
                {
                    logger.LogInformation("Received event with arguments: {@args}", args);
                    if (args.EventArgs != null
                        && args.EventArgs.TryGetValue("userid", out string userId))
                    {
                        walletChangedHandler.Handle(userId);
                    }
                    else
                    {
                        logger.LogWarning("{@eventName}: Invalid event arguments - {@args}",
args.EventName, args.EventArgs);
                    }

                    break;
```

```
                }

            default:
                logger.LogInformation("Attempt to raise non-existent event: {@args}", args);
                break;
        }
    }
  }
}
```

Cryptocurrency.CustomerService.Interfaces

ICustomerService.cs

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

namespace CryptoCurrency.CustomerService.Interfaces
{
    /// <summary>
    /// Interface for communication with CustomerService.
    /// </summary>
    /// <remarks>
    /// [![Flowchart](../images/CS_ICustomerService.jpg)](../images/CS_ICustomerService.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.Interfaces
{
    /// <summary>
    /// Interface for communicating with CustomerService.
    /// </summary>
    /// <remarks>
    /// [![Flowchart](../images/CS_IWalletManager.jpg)](../images/CS_IWalletManager.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(GetWalletModel 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(GetWalletBalanceModel 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<PaginatedTransactions>> GetWalletTransactionsAsync(GetWalletTransactionsModel 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<GetWalletTransactionResponseModel>>
GetWalletTransactionAsync(GetWalletTransactionModel model);
    }
}
```

Cryptocurrency.CustomerService.WebApplication

AccountManagerController.cs

```
using System.Diagnostics.CodeAnalysis;
using System.Web;
using System.Web.Mvc;
using Microsoft.Owin.Security;

namespace CryptoCurrency.CustomerService.WebApplication.Controllers
{
    /// <summary>
    /// Extends default Controller base class with Authentication manager which provides object from OWIN context.
    /// </summary>
    /// <remarks>
    /// [![Flowchart](../images/CS.jpg)](../images/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().Authentication;
            }

            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.WebApplication.Models;
using Microsoft.AspNet.Identity;
using Microsoft.Extensions.Logging;

[assembly: InternalsVisibleTo("UnitTesting_Framework")]

namespace CryptoCurrency.CustomerService.WebApplication.Controllers
{
    /// <summary>
    /// Controller to handle the processes of account actions.
    /// </summary>
    /// <remarks>
    /// [![Flowchart](../images/CS.jpg)](../images/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(IAccountManager 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(model.Email, model.Password);
                if (registerUserResponse.IsRegistered)
                {
                    var loginModel = new LoginModel()
                    {
                        Email = model.Email,
                        Password = model.Password,
                    };
                    return Login(loginModel);
                }

                ModelState.AddModelError("FailedRegisterOperation", 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(identity);
                    logger.LogInformation("User with ID={@userId} logged in.",
AuthenticationManager.User.Identity.GetUserId());

                    return RedirectToAction("Index", "Home");
                }

                ModelState.AddModelError("AccountInfo", 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.Email, 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.UnreadableUserInfo,
        };
        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.Identity.GetUserId();
        accountManager.Logout(userId);
        AuthenticationManager.SignOut(authenticationTypes: "ApplicationCookie");
        logger.LogInformation("User with ID={@userId} logged off.", userId);
        return RedirectToAction("Login");
    }
  }
}
```

HomeController.cs

```
using System.Web.Mvc;

namespace CryptoCurrency.CustomerService.WebApplication.Controllers
{
  /// <summary>
  /// Represents controller for default pages.
  /// </summary>
  /// <remarks>
  /// [![Flowchart](../images/CS.jpg)](../images/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.Interfaces;
using Microsoft.AspNet.Identity;
using Microsoft.Extensions.Logging;

namespace CryptoCurrency.CustomerService.WebApplication.Controllers
{
    /// <summary>
    /// Controller to handle the process of creating a transaction.
    /// </summary>
    /// <remarks>
    /// [![Flowchart](../images/CS.jpg)](../images/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(IWalletManager 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(AuthenticationManager.User.Identity.GetUserId());
            var responseResult = await walletManager.GetOrCreateWalletAsync(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(CreateTransactionModel transaction)
        {
            logger.LogInformation("Started request for creating transaction {@transaction}", transaction);
            var response = await customerService.CreateTransactionAsync(transaction.Sender, transaction.Receiver,
transaction.Amount);
            if (response == null)
```

```
            {
                logger.LogWarning("Request received no response");
                response = new CreateTransactionResponse(false, ResponseMessages.NoResponseFromServer);
            }

            return Json(response);
        }
    }
}
```

WalletController.cs

```
using System;
using System.Threading.Tasks;
using System.Web.Mvc;
using CryptoCurrency.Common.Models;
using CryptoCurrency.CustomerService.Interfaces;
using Microsoft.AspNet.Identity;

namespace CryptoCurrency.CustomerService.WebApplication.Controllers
{
    /// <summary>
    /// Controller to work with user's wallet.
    /// </summary>
    /// <remarks>
    /// [![Flowchart](../images/CS.jpg)](../images/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(AuthenticationManager.User.Identity.GetUserId());
            var responseResult = await walletManager.GetOrCreateWalletAsync(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(AuthenticationManager.User.Identity.GetUserId());
            var responseResult = await walletManager.GetBalanceAsync(new 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(GetWalletTransactionsModel model)
        {
            var userId = Convert.ToInt64(AuthenticationManager.User.Identity.GetUserId());
            model.UserId = userId;
            var responseResult = await walletManager.GetWalletTransactionsAsync(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(GetWalletTransactionModel model)
        {
            var userId = Convert.ToInt64(AuthenticationManager.User.Identity.GetUserId());
            model.UserId = userId;
            var responseResult = await walletManager.GetWalletTransactionAsync(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.WebApplication.Hubs
{
    /// <inheritdoc cref="Hub"/>
    /// <remarks>
    /// [![Flowchart](../images/CS.jpg)](../images/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(bool)"/>
    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.WebApplication.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 GetWalletTransactionsEndpointName = "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">Interface 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.GetBalanceAsync(GetWalletBalanceModel)"/>
        public async Task<ResponseResult<decimal>> GetBalanceAsync(GetWalletBalanceModel 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<GetWalletBalanceModel, decimal>(model,
endpoint);
            return getWalletBalanceResponse;
        }

        /// <inheritdoc cref="IWalletProvider.GetOrCreateWalletAsync(GetWalletModel)"/>
        public async Task<ResponseResult<WalletInfo>> GetOrCreateWalletAsync(GetWalletModel 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<GetWalletModel, WalletInfo>(model, endpoint);
            return getWalletResponse;
        }

        /// <inheritdoc cref="IWalletProvider.GetWalletTransactionAsync(GetWalletTransactionModel)"/>
        public async Task<ResponseResult<GetWalletTransactionResponseModel>>
GetWalletTransactionAsync(GetWalletTransactionModel 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<GetWalletTransactionModel,
GetWalletTransactionResponseModel>(model, endpoint);
            return getWalletTransactionResponse;
        }

        /// <inheritdoc cref="IWalletProvider.GetWalletTransactionsAsync(GetWalletTransactionsModel)"/>
        public async Task<ResponseResult<PaginatedTransactions>>
GetWalletTransactionsAsync(GetWalletTransactionsModel model)
        {
            logger.LogInformation("Sending a request to get a transactions for a user with ID={userId}.", model.UserId);
            var endpoint = configurationManager.GetEndpoint(GetWalletTransactionsEndpointName);
            var getWalletTransactionsResponse = await httpClient.GetAsync<GetWalletTransactionsModel,
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.WebApplication.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">Interface for interaction with configuration file.</param>
        /// <param name="logger">Logger.</param>
        public TransactionCreator(IHttpClientWrapper httpClient, IEndpointConfiguration configurationManager,
ILogger<TransactionCreator> logger)
        {
            this.httpClient = httpClient;
            this.configurationManager = configurationManager;
            this.logger = logger;
        }

        /// <inheritdoc cref="ITransactionCreator.CreateTransaction(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.CreateTransactionAsync(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(transaction);
            var content = new StringContent(jsonData, Encoding.UTF8, "application/json");
            var result = await SendPostAsync(CreateTransactionEndpointName, 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.ReadAsStringAsync();
            result = JsonConvert.DeserializeObject<CreateTransactionResponse>(requestResult);
        }
        catch (Exception ex)
        {
            logger.LogError(ex, "Request cannot be sent");
            result = new CreateTransactionResponse(false, ResponseMessages.NoResponseFromServer);
        }

        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("createTransaction").blur();
  document.getElementById("response").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("response").style.color = 'red';
    document.getElementById("response").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("response").style.color = '#037f51';
            document.getElementById("response").innerHTML = "Succeeded: " + data.ResponseMessage;
          }
          else {
            document.getElementById("response").style.color = 'red';
            document.getElementById("response").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).style.borderColor = 'red';
    }
    else {
        document.getElementById(owner + "Message").innerHTML = "&#10004";
        document.getElementById(owner).style.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('min'))
            }
            else if (filterDate.val() > filterDate.attr('max')) {
                filterDate.val(filterDate.attr('max'))
```

```
    }
})

$('#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.location.search);
    let filterDateFrom = document.getElementById('filter-date-from');
    let filterDateTo = document.getElementById('filter-date-to');
    let filterWallet = document.getElementById('filter-wallet');
    let filterStatus = document.getElementById('filter-status');
    let filterType = document.getElementById('filter-type');

    if (filterDateFrom.value && filterDateFrom.value !== filterDateFrom.min) {
        queryParams.set(filterDateFrom.name, filterDateFrom.value);
    } else {
        queryParams.delete(filterDateFrom.name);
    }

    if (filterDateTo.value && filterDateTo.value !== filterDateTo.max) {
        queryParams.set(filterDateTo.name, filterDateTo.value);
    } else {
        queryParams.delete(filterDateTo.name);
    }

    if (filterWallet.value) {
        queryParams.set(filterWallet.name, filterWallet.value);
    } else {
        queryParams.delete(filterWallet.name);
```

```
            }

            if (filterStatus.value == 'any') {
                queryParams.delete(filterStatus.name);
            } else {
                queryParams.set(filterStatus.name, filterStatus.value);
            }

            if (filterType.value == 'any') {
                queryParams.delete(filterType.name);
            } else {
                queryParams.set(filterType.name, 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.location.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(transactionSender, 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/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 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/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>
                <div class="ms-2 d-none d-md-block">Incoming</div>
            </div>`;
}

function getTransactionAmountHtml(transactionSender, 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(transactionSender, 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.HasPrevious) {
            $('#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.PageInfo.PageNumber);
            $('#next-page').val(data.Model.PageInfo.PageNumber + 1);
            $('#last-page').val(data.Model.PageInfo.TotalPages);
        }
    });
}
```

Login.cshtml

```
@model CryptoCurrency.CustomerService.WebApplication.Models.LoginModel

@{
    ViewBag.Title = "Log in";
}
<h2>@ViewBag.Title</h2>
<hr />
<p>@Html.ValidationMessage("AccountInfo", 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/jqueryval")
            }
```

Register.cshtml

```
@model CryptoCurrency.CustomerService.WebApplication.Models.RegisterModel
@{
    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("_PaginationButtons");
}

@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.GetWalletTransactionResponseMo
del>

@{
    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.Receiver == @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.Receiver
                </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.DateLastUpdated
            </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.Status.Unprocessed)
                {
                    <div class="h5 m-0 mx-1">Unprocessed</div>
                    <svg xmlns="http://www.w3.org/2000/svg" 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.Status.Processed)
                {
                    <div class="h5 m-0 mx-1">Processed</div>
                    <svg xmlns="http://www.w3.org/2000/svg" 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.MiningService.BL

MiningService.cs

```
using System.Threading;
using CryptoCurrency.Common.Interfaces;
using CryptoCurrency.Common.Models;
using CryptoCurrency.MiningService.Interfaces;
using CryptoCurrency.TransactionService.Interfaces;
using Microsoft.Extensions.Logging;

namespace CryptoCurrency.MiningService.BL
{
    /// <summary>
    /// Class for mining service representation.
    /// </summary>
    /// <remarks>
    /// [![Flowchart](../images/MS.jpg)](../images/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">Instance of transaction service through interface for mining service.</param>
        /// <param name="minerServiceSettings">Settings 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.StartMining"/>
        public void StartMining()
        {
            if (!isLifeCycleActive)
            {
                isLifeCycleActive = true;
                new Thread(Mine).Start();
            }
        }

        /// <inheritdoc cref="IMiningServiceLifecycle.StopMining"/>
        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(TransactionServiceObservableName);
                transactionService.Notify -= OnNotify;
                transactionService.Notify += OnNotify;
                return true;
            }

            return false;
        }

        /// <inheritdoc cref="IMiningServiceLifecycle.Stop"/>
        public void Stop()
        {
            StopMining();
```

```
    var transactionService = observableServiceProvider.Get(TransactionServiceObservableName);
    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.GetMiningProcessInformation(UserId);
        if (blockInfo == null ||
            blockInfo.PreviousBlockHash == null ||
            blockInfo.Transactions == null ||
            blockInfo.HashZeroNumber == null)
        {
            Thread.Sleep(minerServiceSettings.DelayBeforeNextMiningProcessInformationRequest);
            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.AddBlock(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.Interfaces;
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">Configuration manager for user info.</param>
    /// <param name="endpointConfiguration">Configuration 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.CannotReadUserInfoSection,
                };
            }

            if (userInfo.IsConfigured())
            {
                var endpoint = endpointConfiguration.GetEndpoint(ServerName, LoginEndpointName);
                var userDataRequestResponse = await httpHelper.PostAsync<UserInfoModel, 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.ServerErrorUreadableUserData,
                    };
                }

                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.Interfaces
{
    /// <summary>
```

```
    /// An interface for running application.
    /// </summary>
    /// <remarks>
    /// [![Flowchart](../images/MS_IComandHandler.jpg)](../images/MS_IComandHandler.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.Interfaces
{
    /// <summary>
    /// An interface for command calls.
    /// </summary>
    /// <remarks>
    /// [![Flowchart](../images/MS_ILifecycleManager.jpg)](../images/MS_ILifecycleManager.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.Interfaces
{
    /// <summary>
    /// An interface responsible for displaying information.
```

```
    /// </summary>
    /// <remarks>
    /// [![Flowchart](../images/MS_IOutputManager.jpg)](../images/MS_IOutputManager.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(MiningProcessInformation mpi);
    }
}
```

Cryptocurrency.MiningService.Models

Command.cs

```
using System;

namespace CryptoCurrency.MiningService.Models
{
    /// <summary>
    /// A class represents command for user input.
    /// </summary>
    /// <remarks>
    /// [![Flowchart](../images/MS_Command.jpg)](../images/MS_Command.jpg)
    /// </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.Models
{
    /// <summary>
    /// Provides messages for mining client.
    /// </summary>
    /// <remarks>
    /// [![Flowchart](../images/MS_OutputMessages.jpg)](../images/MS_OutputMessages.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.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Serilog;

namespace CryptoCurrency.MiningService.Client
{
    /// <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.GetCurrentDirectory())
                .AddJsonFile("appsettings.json")
                .Build();
```

```
            Log.Logger = new LoggerConfiguration()
                .ReadFrom.Configuration(config)
                .CreateLogger();

            CreateHostBuilder(config).Build().Run();
        }

        /// <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(webBuilder =>
                {
                    webBuilder.UseUrls(GetLocalUrl(configuration));
                    webBuilder.UseStartup<Startup>();
                }).ConfigureServices(services => services.AddHostedService<BackgroundComandHandler>());

        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.Interfaces;
using CryptoCurrency.TransactionService.Interfaces;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.SignalR.Client;
using Microsoft.Extensions.DependencyInjection;

namespace CryptoCurrency.MiningService.Client
{
    /// <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(IServiceCollection services)
        {
            services.AddControllers();
            services.AddHttpClient<IHttpClientWrapper, HttpClientWrapper>();
            services.AddOptions();
            services.AddSingleton<ICommandHandler, 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(ILifecycleManager 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(keyInfo))
                {
                    commandsKey[keyInfo].Action.Invoke();
                }
            }
        }

        /// <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.Interfaces;

namespace CryptoCurrency.MiningService.Client
{
    /// <summary>
    /// A class responsible for displaying data in the console UI.
    /// </summary>
    [ExcludeFromCodeCoverage]
    public class OutputManagerConsole : IOutputManager
    {
        /// <inheritdoc cref="IOutputManager.PrintBlock(Block)"/>
        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(string)"/>
        public void PrintError(string error)
        {
            Console.ForegroundColor = ConsoleColor.Red;
            Console.WriteLine(error);
            Console.ForegroundColor = ConsoleColor.White;
        }

        /// <inheritdoc cref="IOutputManager.PrintMiningProcessInformation(MiningProcessInformation)"/>
        public void PrintMiningProcessInformation(MiningProcessInformation mpi)
        {
            Console.ForegroundColor = ConsoleColor.Green;
            Console.Write("MPI: ");
            Console.ForegroundColor = ConsoleColor.White;
            Console.WriteLine(DateTime.Now);
            Console.WriteLine($"{mpi}");
        }
    }
}
```

TransactionServiceProxyController.cs

```
using CryptoCurrency.Common.Interfaces;
using CryptoCurrency.Common.Models;
using Microsoft.AspNetCore.Mvc;

namespace CryptoCurrency.MiningService.Client.Controllers
{
    /// <summary>
    /// Controller for receiving calls from the server.
    /// </summary>
    [Route("api/[controller]/[action]")]
    [ApiController]
    public class TransactionServiceProxyController : ControllerBase, IObservableServiceProxyController
    {
        private readonly IObservableServiceProxyController observableServiceProxyControllerBL;

        /// <summary>
        /// Initializes a new instance of the <see cref="TransactionServiceProxyController"/> class.
        /// </summary>
        /// <param name="observableServiceProxyControllerBL">BL of observable service proxy controller.</param>
        public TransactionServiceProxyController(IObservableServiceProxyController
observableServiceProxyControllerBL)
        {
            this.observableServiceProxyControllerBL = observableServiceProxyControllerBL;
        }

        /// <inheritdoc cref="IObservableServiceProxyController.IsAvailable(ObserverAddress)"/>
        [HttpPost]
        public ResponseResult IsAvailable(ObserverAddress observerAddress)
        {
            return observableServiceProxyControllerBL.IsAvailable(observerAddress);
        }

        /// <inheritdoc cref="IObservableServiceProxyController.Notify(ObserverNotifyInfo)"/>
        [HttpPost]
        public void Notify(ObserverNotifyInfo observerNotifyInfo)
        {
            observableServiceProxyControllerBL.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/SqlDataAccess.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">Interface 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(IConfigurationManager configurationManager)
        {
            this.configurationManager = configurationManager;
            connectionString = this.configurationManager.GetConnectionString();

            if (connectionString == null)
            {
                throw new ConfigurationErrorsException("Configuration 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(string, object)"/>
        public int Execute(string procedure, object arguments = null)
        {
            return connection.Execute(procedure, arguments, commandType: CommandType.StoredProcedure);
        }

        /// <inheritdoc cref="ISqlDataAccess.Query{T}(string, 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.QueryMultiple(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.ExecuteAsync(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>(procedure, arguments, commandType:
CommandType.StoredProcedure);

            return entities;
        }

        /// <inheritdoc cref="ISqlDataAccess.QueryMultipleAsync(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(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="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(IEnumerable{Transaction})"/>
```

```
        public IEnumerable<Block> GetAllBlocks(IEnumerable<Transaction> transactions)
        {
            var blockDataModelType = typeof(BlockDataModel);
            var blockTransactionDataModelType = typeof(BlockTransactionDataModel);
            var types = new List<Type>
            {
                blockDataModelType,
                blockTransactionDataModelType,
            };

            var queryResult = sqlDataAccess.QueryMultiple(SpGetAllBlocks, types);

            var blocksDB = queryResult[blockDataModelType].Cast<BlockDataModel>();
            var blockTransactionsDB = queryResult[blockTransactionDataModelType]
                .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(transactionsDB[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<SettingsDataModel>(SpGetSettings).ToDictionary(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(property.Name, out string value);
                if (value != null)
                {
                    settingType.GetProperty(property.Name).SetValue(settings,
Convert.ChangeType(settingsDataModel[property.Name], settingType.GetProperty(property.Name).PropertyType));
                }
            }

            return settings;
        }

        /// <inheritdoc cref="ISettingsDAL.UpdateSetting(string, string)"/>
        public void UpdateSetting(string settingName, string settingValue)
        {
            sqlDataAccess.Execute(SpUpdateSetting, 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/TransactionsDAL.jpg)](../images/TransactionsDAL.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.GetAllTransactions"/>
        public IEnumerable<Transaction> GetAllTransactions()
        {
            var transactions = sqlDataAccess.Query<Transaction>(SpGetAllTransactions);
            return transactions;
        }

        /// <inheritdoc cref="ITransactionsDAL.SaveTransaction(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(SpAddTransaction, arguments);
        }
    }
}
```

UsersDAL.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using CryptoCurrency.Common.Models;
using CryptoCurrency.DAL.Interfaces;
using CryptoCurrency.DAL.Models;

namespace CryptoCurrency.DAL
{
    /// <summary>
    /// Provides methods to access users and user's roles tables in database.
    /// </summary>
    /// <remarks>
    /// [![Flowchart](../images/UsersDAL.jpg)](../images/UsersDAL.jpg)
    /// </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.AddRoleToUser(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(SpAddRoleToUser, arguments);

        if (result > 0)
        {
            return true;
        }

        return false;
    }

    throw new ArgumentException($"Role with ID '{roleId}' not defined.");
}

/// <inheritdoc cref="IUsersDAL.AddUser(string)"/>
public bool AddUser(string username)
{
    var arguments = new
    {
        Username = username,
    };
    var result = sqlDataAccess.Execute(SpAddUser, arguments);
    if (result > 0)
    {
        return true;
    }
```

```
            return false;
        }

        /// <inheritdoc cref="IUsersDAL.AddUser(string, 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>(SpAddUserWithCredentials,
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<User>();
            var userRoles = queryReult[userRolesType].Cast<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<User>();
    var userRoles = queryResult[userRolesType].Cast<UserRoleDataModel>();
    var user = users.FirstOrDefault();
    if (user != null)
    {
        var roles = userRoles.Sum(ur => ur.RoleId);
        user.SetRoles(roles);
    }

    return user;
}

/// <inheritdoc cref="IUsersDAL.RemoveRoleFromUser(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(SpRemoveRoleFromUser, arguments);
        if (result > 0)
        {
            return true;
        }

        return false;
    }

    throw new ArgumentException($"Role with ID '{roleId}' not defined.");
}

/// <inheritdoc cref="IUsersDAL.AddRoleToUserAsync(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(SpAddRoleToUser, arguments);

        if (result > 0)
        {
            return true;
        }

        return false;
    }

    throw new ArgumentException($"Role with ID '{roleId}' not defined.");
```

```
        }

        /// <inheritdoc cref="IUsersDAL.GetUserByIdAsync(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.QueryMultipleAsync(SpGetUserById, types,
arguments);

            var users = queryResult[userType].Cast<User>();
            var userRoles = queryResult[userRolesType].Cast<UserRoleDataModel>();
            var user = users.FirstOrDefault();
            if (user != null)
            {
                var roles = userRoles.Sum(ur => ur.RoleId);
                user.SetRoles(roles);
            }

            return user;
        }

        /// <inheritdoc cref="IUsersDAL.RemoveRoleFromUserAsync(long, byte)"/>
        public async Task<bool> RemoveRoleFromUserAsync(long userId, byte roleId)
        {
            if (Enum.IsDefined(typeof(Roles), (int)roleId))
            {
                var arguments = new
                {
                    UserId = userId,
                    RoleId = roleId,
                };
                var result = await sqlDataAccess.ExecuteAsync(SpRemoveRoleFromUser, arguments);
                if (result > 0)
                {
                    return true;
                }

                return false;
            }

            throw new ArgumentException($"Role with ID '{roleId}' not defined.");
        }

        /// <inheritdoc cref="IUsersDAL.AddUserAsync(string)"/>
        public async Task<bool> AddUserAsync(string username)
        {
```

```
            var arguments = new
            {
                Username = username,
            };
            var result = await sqlDataAccess.ExecuteAsync(SpAddUser, arguments);
            if (result > 0)
            {
                return true;
            }

            return false;
        }

        /// <inheritdoc cref="IUsersDAL.AddUserAsync(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.GetAllUsersAsync"/>
        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.QueryMultipleAsync(SpGetAllUsers, types);
            var users = queryReult[userType].Cast<User>();
            var userRoles = queryReult[userRolesType].Cast<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.GetUserByEmail(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<User>();
            var userRoles = queryResult[userRolesType].Cast<UserRoleDataModel>();
            var user = users.FirstOrDefault();
            if (user != null)
            {
                var roles = userRoles.Sum(ur => ur.RoleId);
                user.SetRoles(roles);
            }

            return user;
        }

        /// <inheritdoc cref="IUsersDAL.GetUserByEmailAsync(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.QueryMultipleAsync(SpGetUserByEmail, types,
arguments);

            var users = queryResult[userType].Cast<User>();
            var userRoles = queryResult[userRolesType].Cast<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(Wallet)"/>
        public bool AddWallet(Wallet wallet)
        {
            var arguments = new
            {
                wallet.PublicKey,
                wallet.UserId,
            };
            var result = sqlDataAccess.Execute(SpAddWallet, 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[transactionDataModelType].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/ISqlDataAccess.jpg)](../images/ISqlDataAccess.jpg)
    /// </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/ITransactionsDAL.jpg)](../images/ITransactionsDAL.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.jpg)](../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/SettingsDataModel.jpg)](../images/SettingsDataModel.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/UserRoleDataModel.jpg)](../images/UserRoleDataModel.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/BlockCalculationUtils.jpg)](../images/BlockCalculationUtils.jpg)
    /// </remarks>
    public class BlockCalculationUtils : IBlockCalculationUtils
    {
        /// <inheritdoc cref="IBlockCalculationUtils.ValidateHash(string, string)"/>
        public bool ValidateHash(string blockHash, string condition)
        {
            if (blockHash != null)
            {
                return blockHash.StartsWith(condition);
            }

            return false;
        }

        /// <inheritdoc cref="IBlockCalculationUtils.CalculateHash(string, string, uint)"/>
        public string CalculateHash(string previousBlockHash, string transactionsHash, uint nonce)
        {
            return Calculate($"{previousBlockHash}{transactionsHash}{nonce}");
        }

        /// <inheritdoc cref="IBlockCalculationUtils.CalculateHash(List{Transaction})"/>
        public string CalculateHash(List<Transaction> transactions)
        {
            return Calculate(string.Join(string.Empty, transactions.Select(
                transaction =>
                {
                    return Calculate($"{transaction.Id}{transaction.Sender}{transaction.TransferAmount.ToString("F8",
CultureInfo.InvariantCulture)}{transaction.Receiver}");
                })));
        }

        private static string Calculate(string data)
        {
            using (SHA256 hash = SHA256.Create())
            {
                byte[] calculatedHash = hash.ComputeHash(Encoding.UTF8.GetBytes(data));
                return System.BitConverter.ToString(calculatedHash).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.jpg)](../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{TRequestModel, TResponseModel}(TRequestModel, string)"/>
        public async Task<ResponseResult<TResponseModel>> 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.EnsureSuccessStatusCode().Content.ReadAsStringAsync();
                responseResult = JsonConvert.DeserializeObject<ResponseResult<TResponseModel>>(requestResult);
            }
            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{TRequestModel, TResponseModel}(TRequestModel, string)"/>
        public async Task<ResponseResult<TResponseModel>> 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.EnsureSuccessStatusCode().Content.ReadAsStringAsync();
                responseResult = JsonConvert.DeserializeObject<ResponseResult<TResponseModel>>(requestResult);
            }
            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{TRequestModel}(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>(TRequestModel 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.EnsureSuccessStatusCode().Content.ReadAsStringAsync();
                responseResult = JsonConvert.DeserializeObject<ResponseResult>(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(obj).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/HttpClientWrapper.jpg)](../images/HttpClientWrapper.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.PostAsync(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/HMACHttpClientWrapper.jpg)](../images/HMACHttpClientWrapper.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">Interface for calculating HMAC.</param>
        public HMACHttpClientWrapper(IHMACHashingUtils 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.PostAsync(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("HMAC", hmac);
            return await httpClient.SendAsync(requestMessage, 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.Transactions[i]))
                {
                    return false;
                }
            }

            return Hash.Equals(block.Hash)
```

```
            && Nonce.Equals(block.Nonce)
            && PreviousBlockHash.Equals(block.PreviousBlockHash);
        }

        /// <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.jpg)](../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">Sender wallet public key.</param>
        /// <param name="receiverWalletPublicKey">Receiver 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(dateLastUpdated, 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(dateLastUpdated, 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.Receiver)
                && 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 – Ілюстративна частина

ІЛЮСТРАТИВНА ЧАСТИНА
РОЗРОБКА ПРОГРАМНИХ ДОДАТКІВ ДЛЯ ЕМІСІЇ ТА ПЕРЕКАЗУ КРИПТОВАЛЮТИ

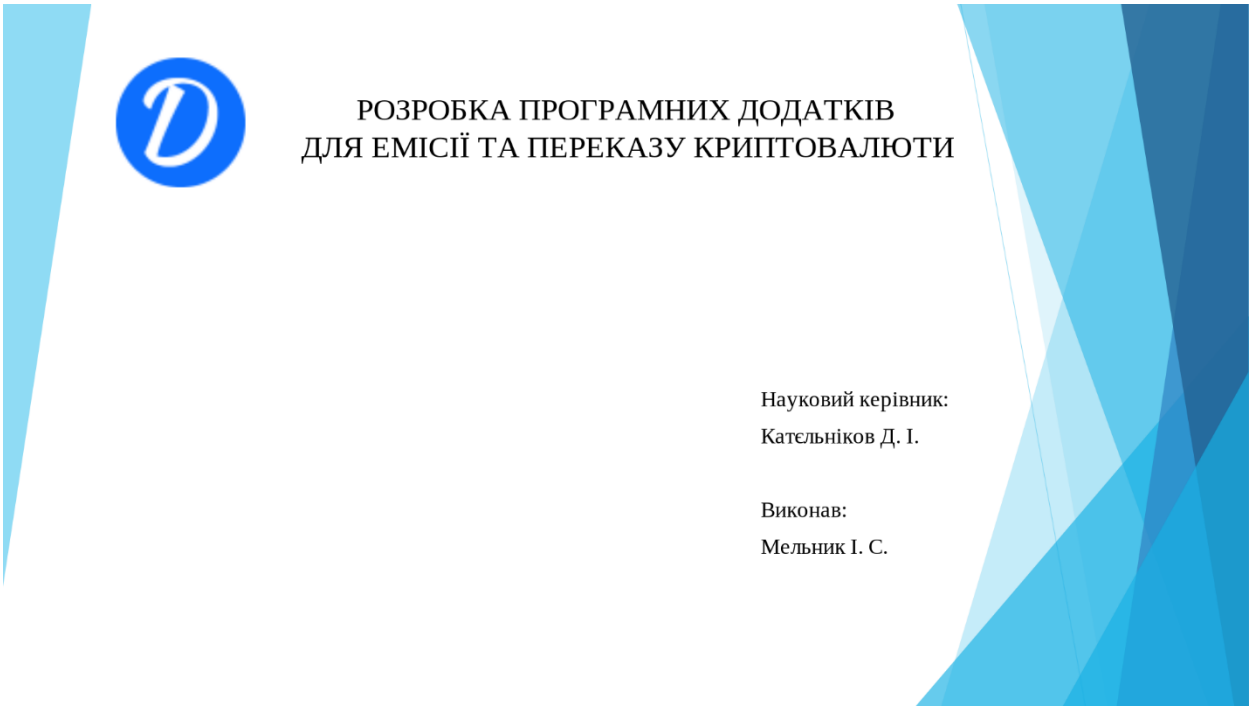

Рисунок Г.1 – Титульний слайд з назвою роботи

Рисунок Г.2 – Мета, об'єкт, премет та практична цінність дослідження

Рисунок Г.3 – Наукова новизна

Рисунок Г.4 – Основні завдання бакалаврської дипломної роботи

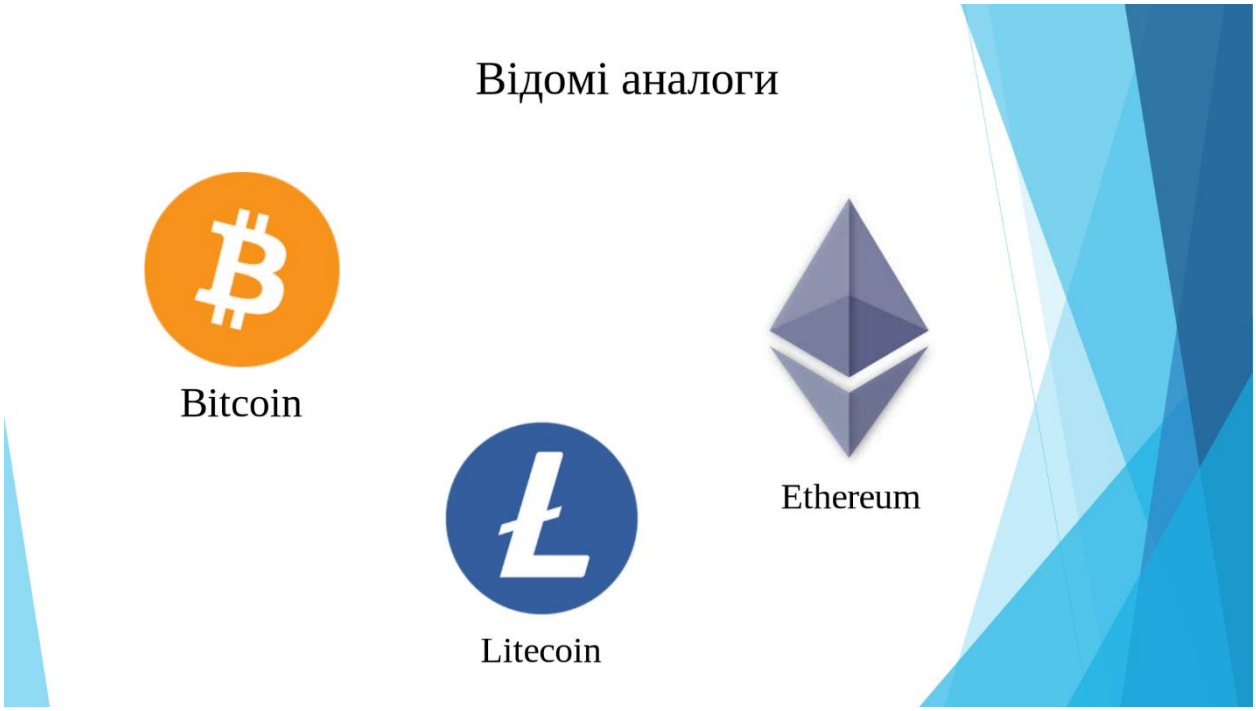

Рисунок Г.5 – Відомі аналоги

Порівняльний аналіз аналогів

Критерії	Bitcoin	Ethereum	Litecoin	Власна розробка
Децентралізація	+	+	+	-
Необмежена емісія	-	+	-	+
Можливість гнучкої зміни параметрів системи в залежності від поточного навантаження	-	-	-	+
Наявність власного інструменту для роботи з гаманцем	-	-	-	+

Рисунок Г.6 – Порівняльний аналіз аналогів

Рисунок Г.7 – Загальна централізована модель криптовалютної системи

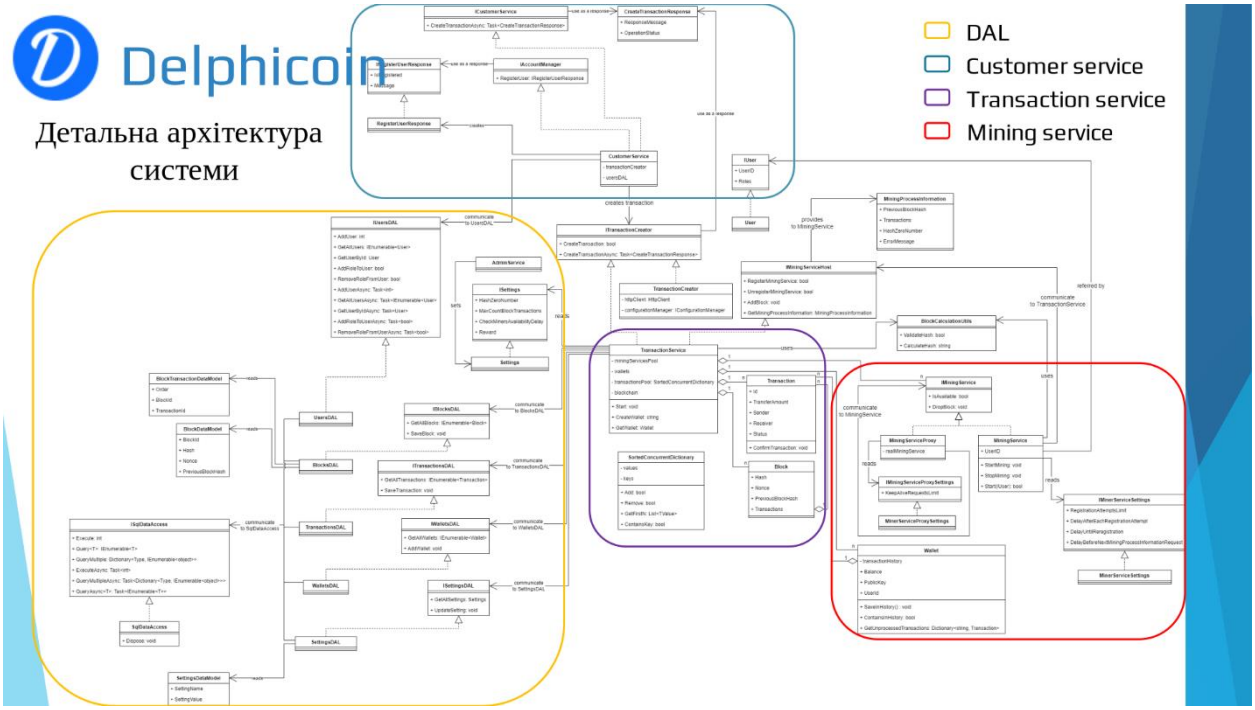

Рисунок Г.8 – Детальна архітектура системи

Delphicoin

Модель життєдіяльності системи

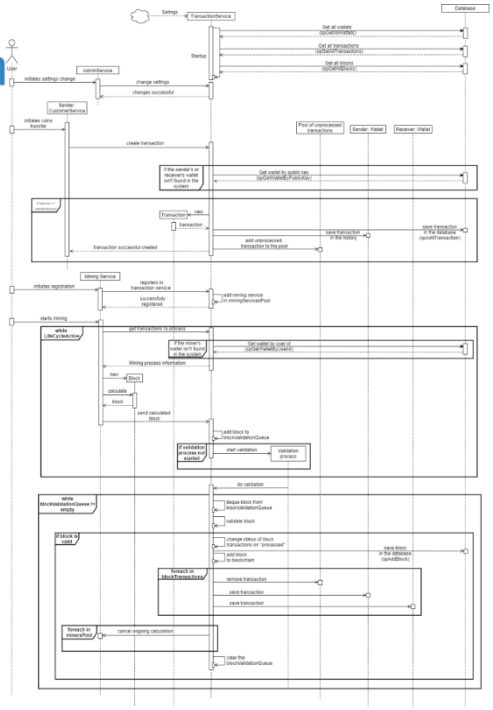

Рисунок Г.9 – Модель послідовностей функціонування системи

Delphicoin

Алгоритм процесу емісії криптовалюти

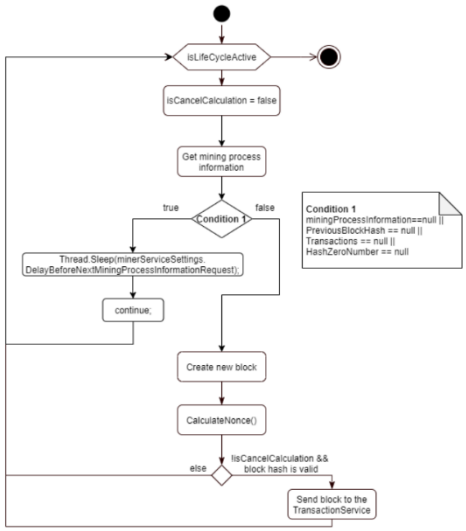

Рисунок Г.10 – Алгоритм процесу емісії криптовалюти

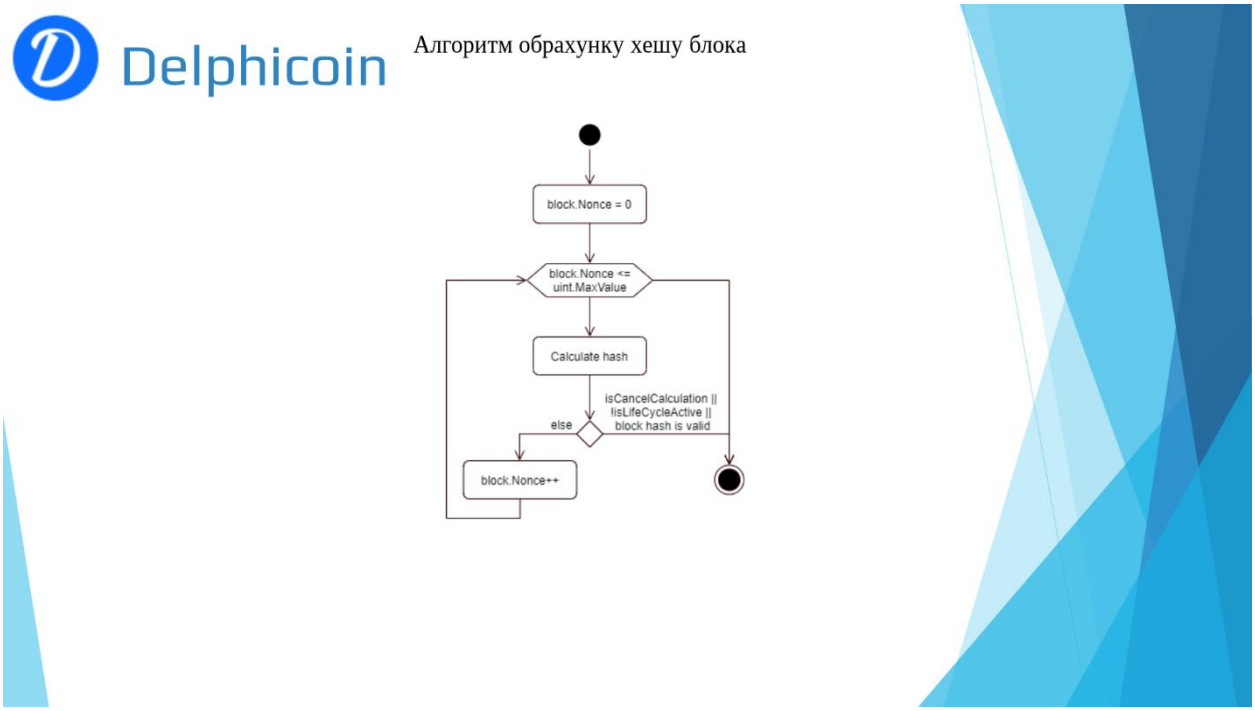

Рисунок Г.11 – Алгоритм обрахунку хешу блока

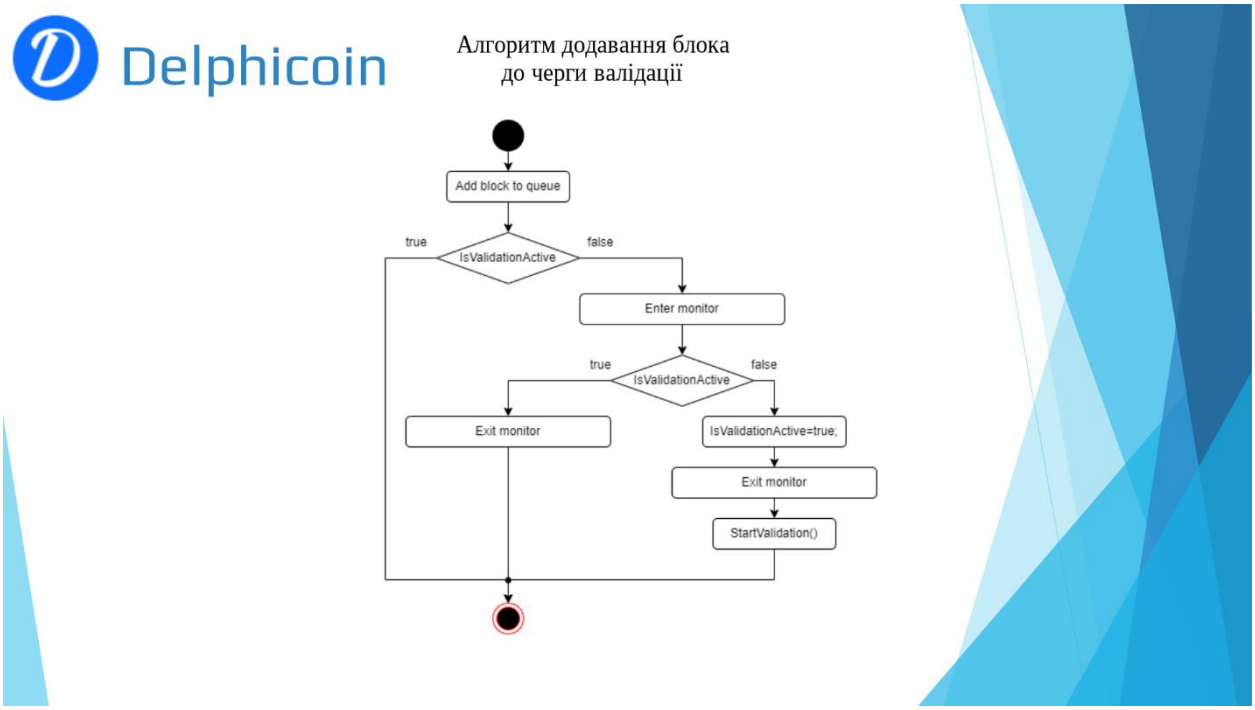

Рисунок Г.12 – Алгоритм додавання блоку до черги валідації

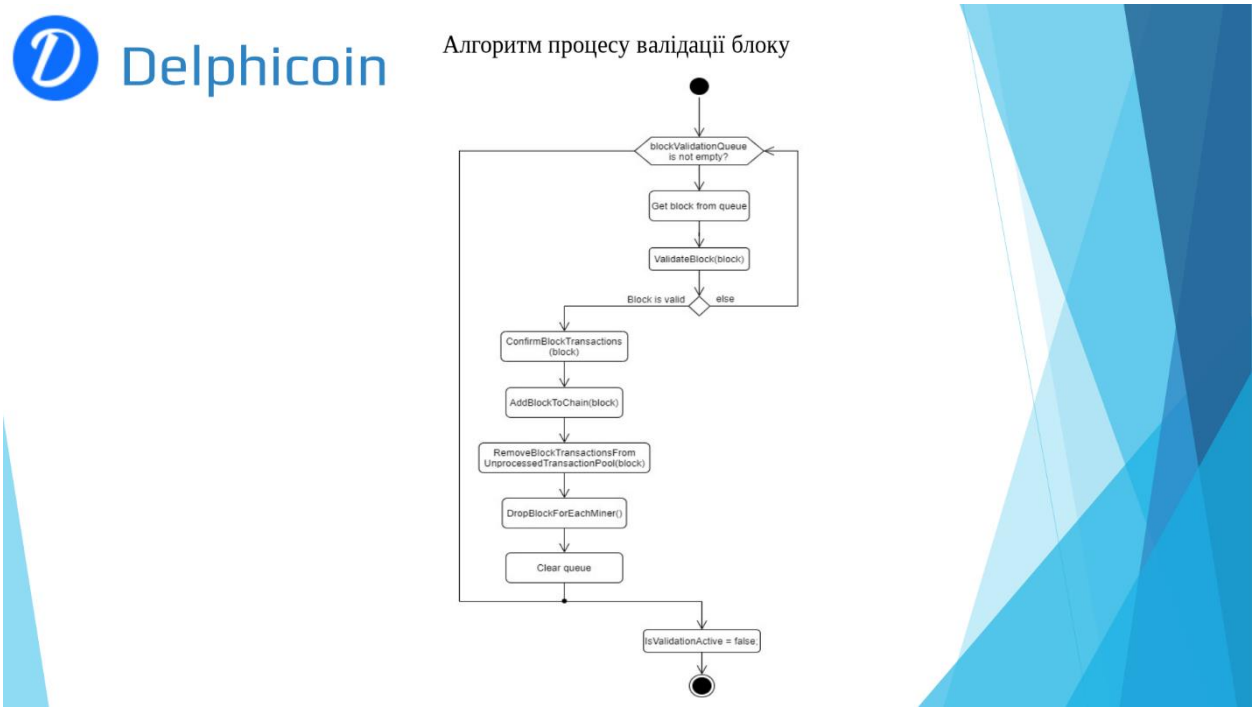

Рисунок Г.13 – Алгоритм процесу валідації блоку

Рисунок Г.14 – Тестування системи

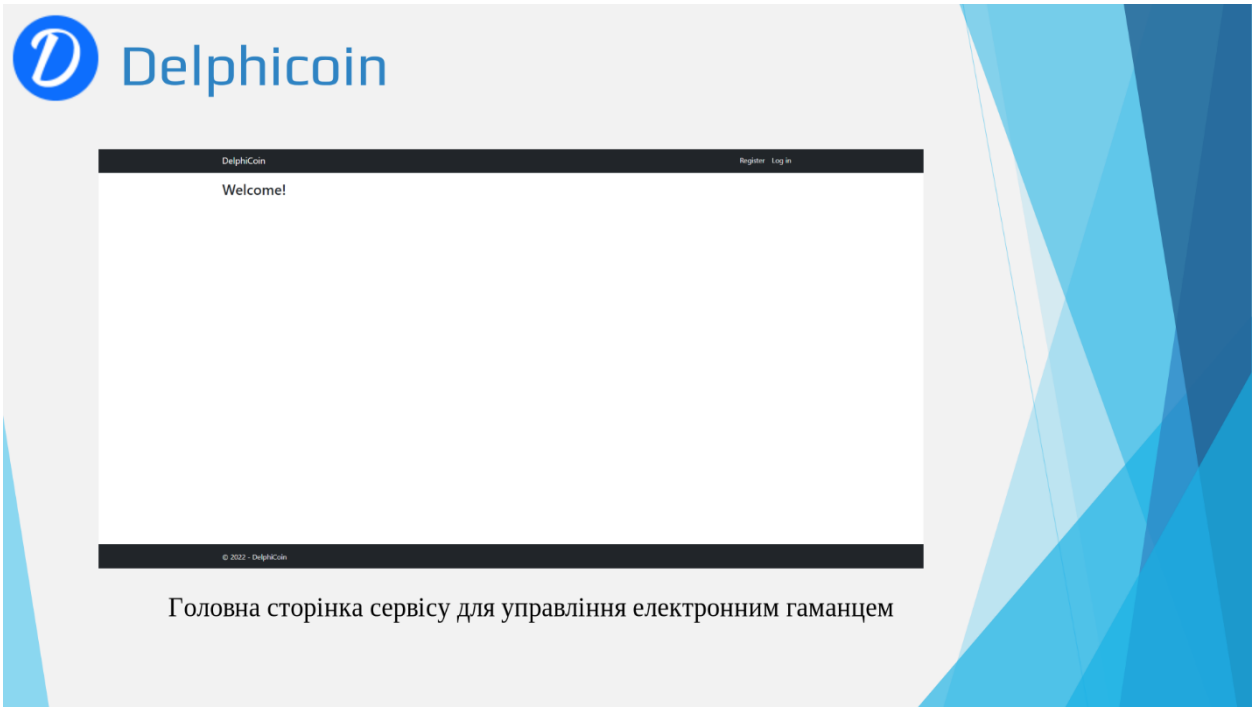

Рисунок Г.15 – Головна сторінка сервісу для роботи з електронним гаманцем

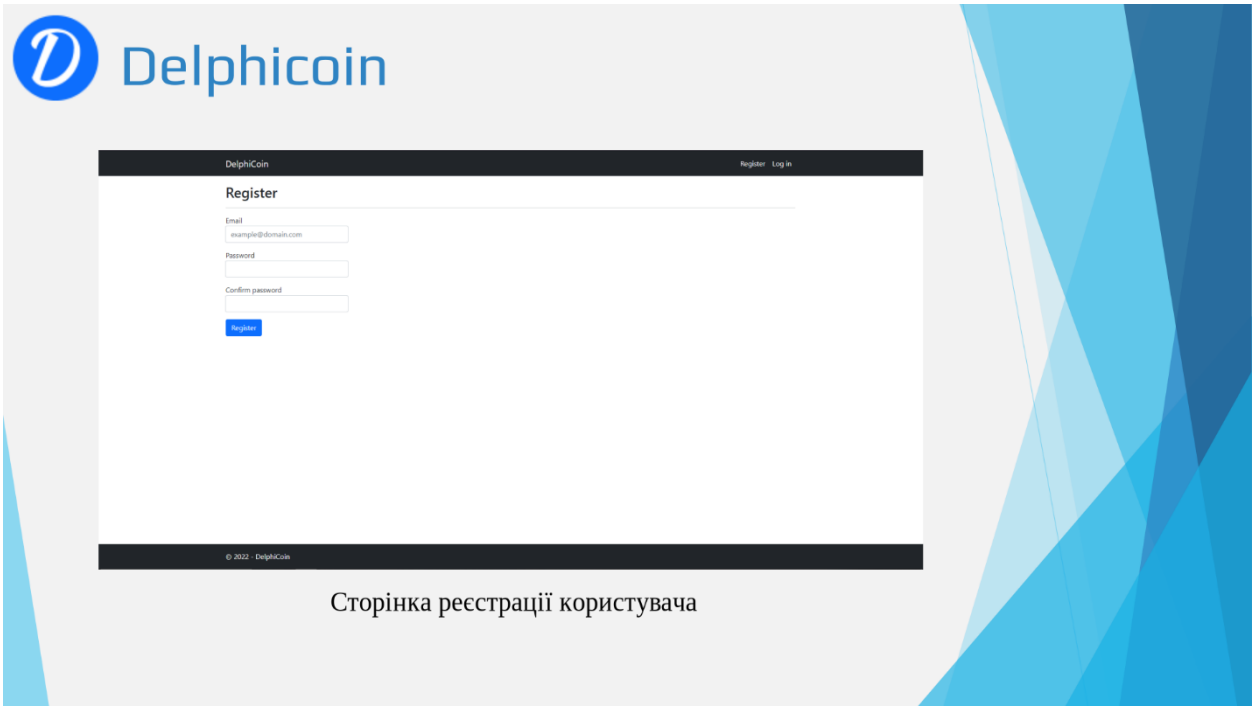

Рисунок Г.16 – Сторінка реєстрації користувача

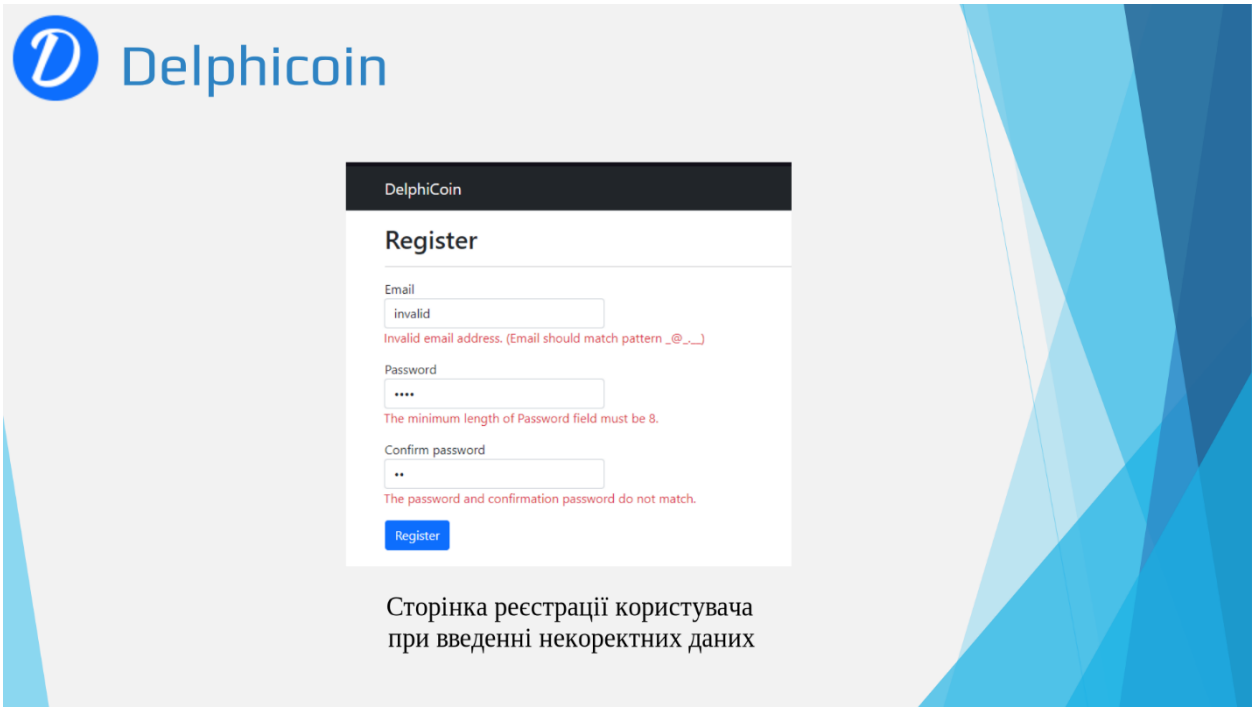

Рисунок Г.17 – Сторінка реєстрації користувача при введенні некоректних даних

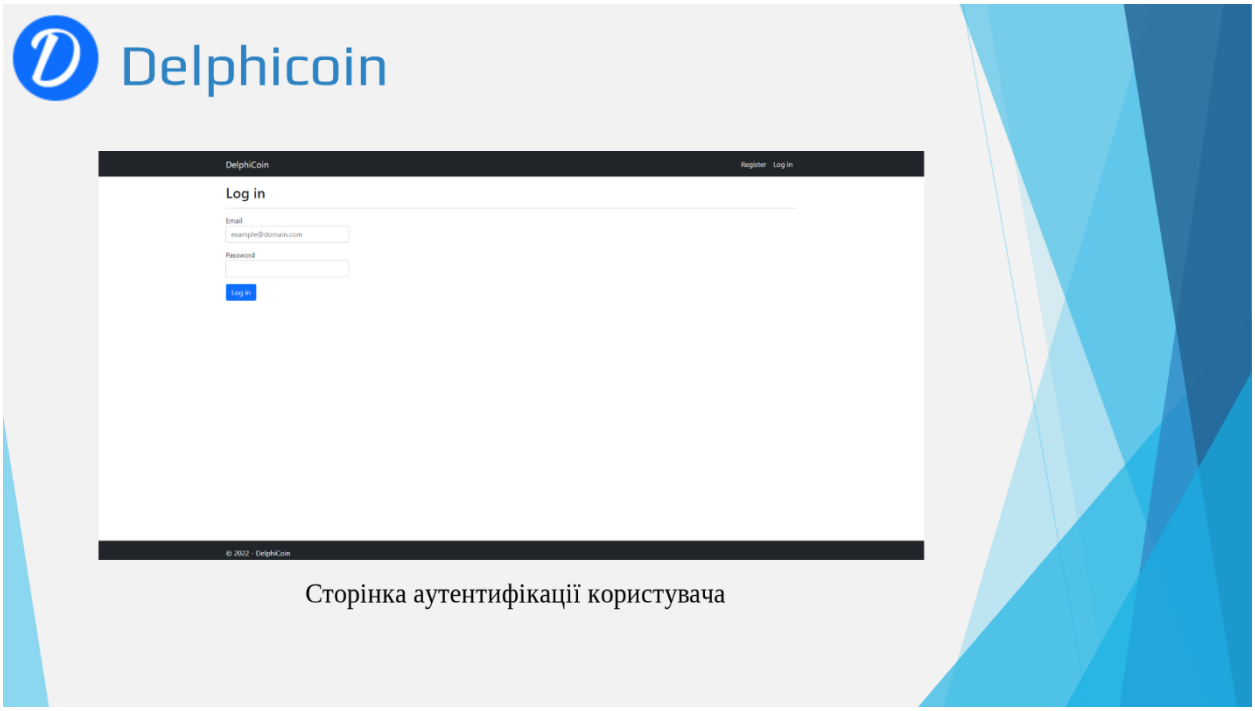

Рисунок Г.18 – Сторінка аутентифікації користувача

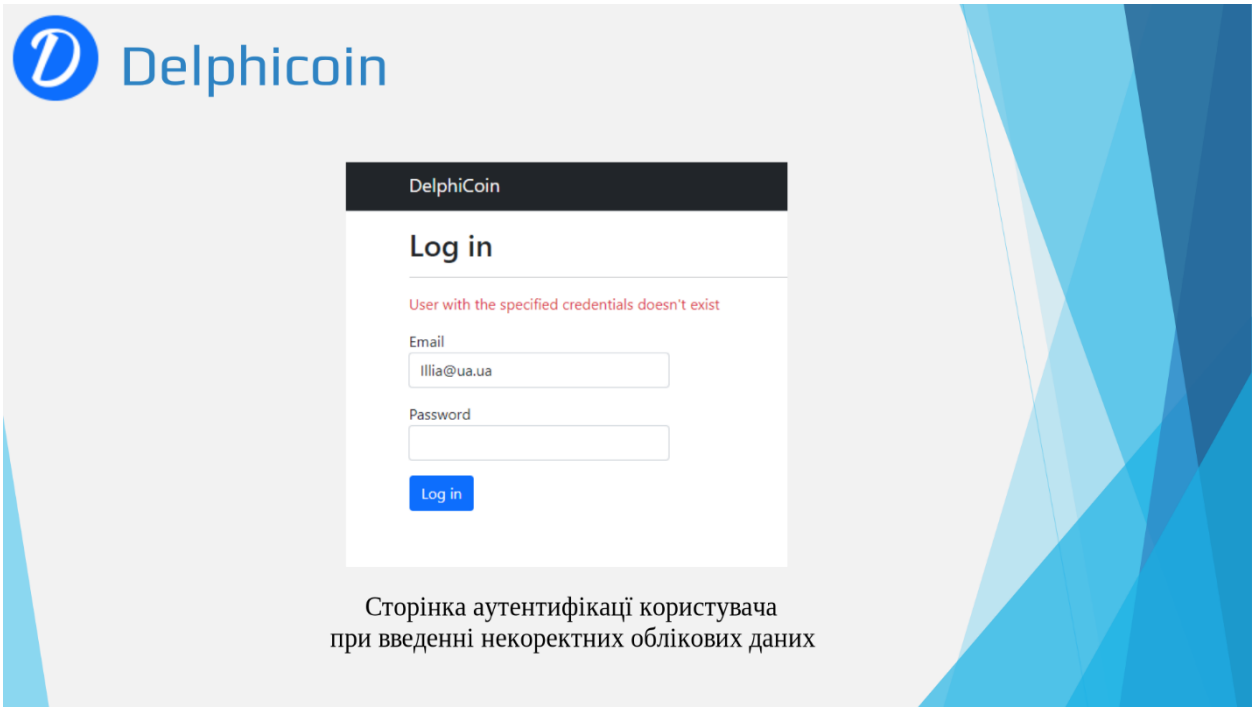

Рисунок Г.19 – Сторінка аутентифікації при введенні неправильних облікових даних

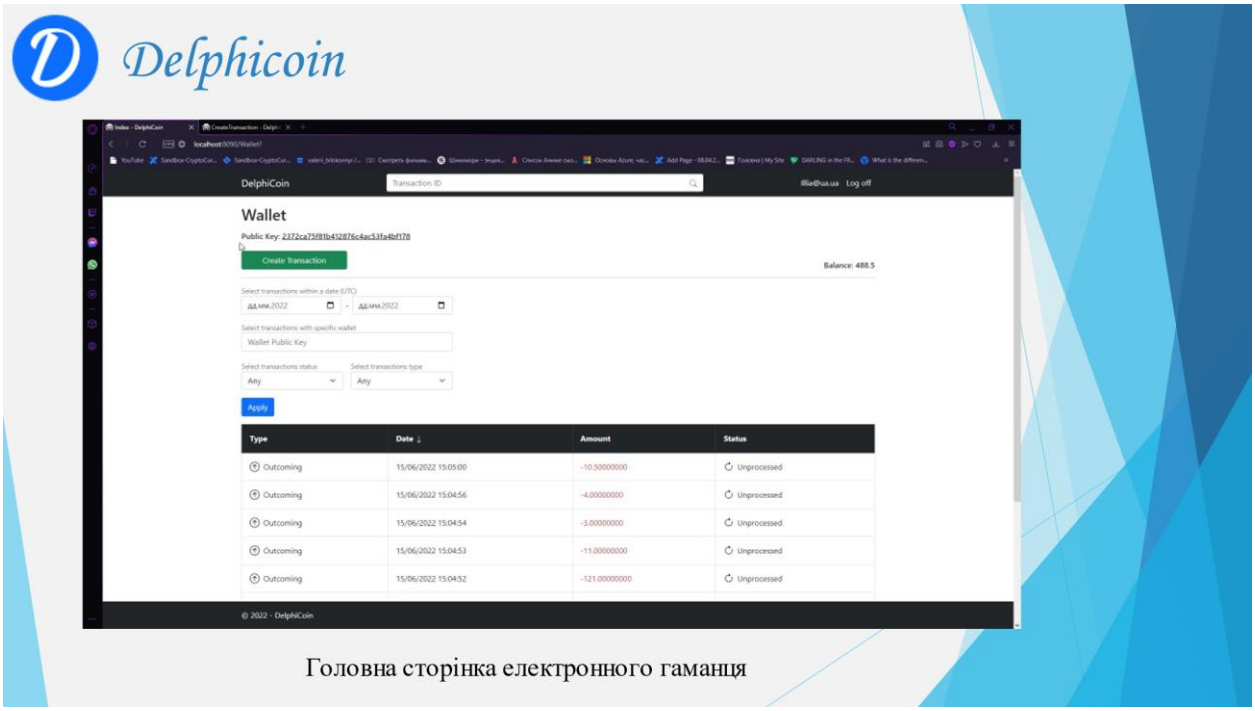

Рисунок Г.20 – Головна сторінка електронного гаманця

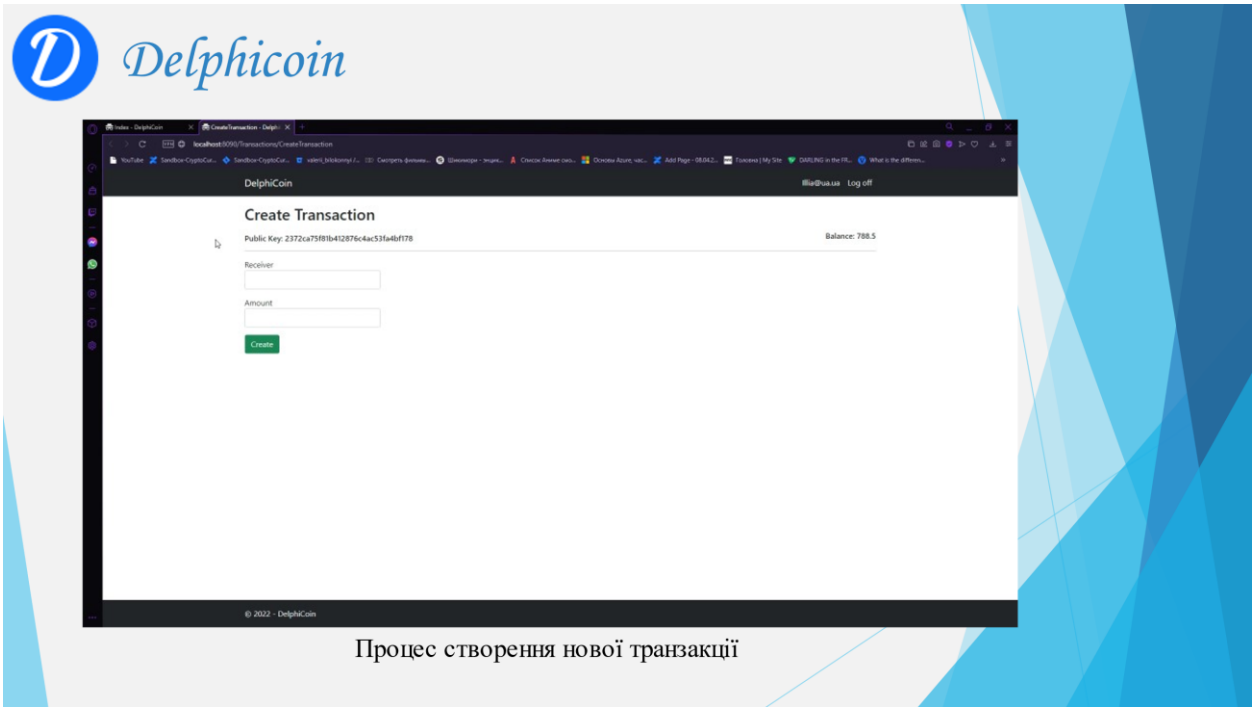

Рисунок Г.21 – Процес створення нової транзакції

Рисунок Г.22 – Сторінка з детальною інформацією про транзакцію

Рисунок Г.23 – Процес обрахунку нового блока сервісом для емісії криптовалюти

Рисунок Г.24 – Висновки

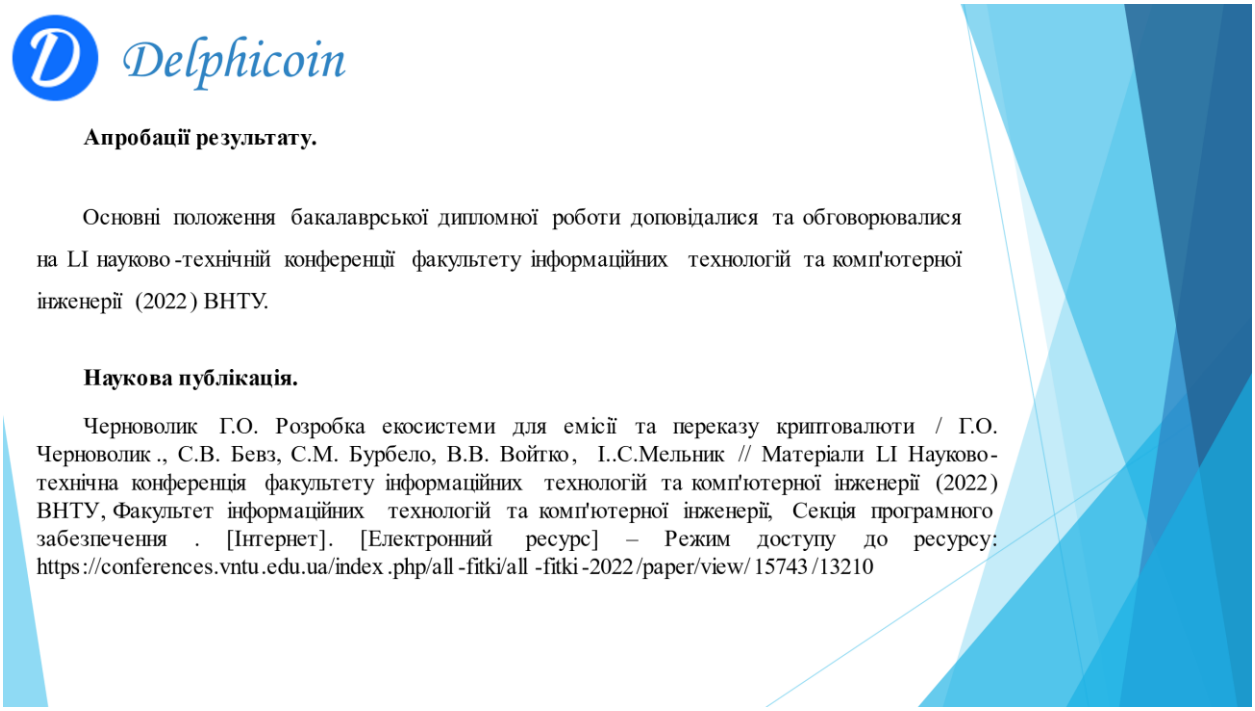

Рисунок Г.25 – Апробації та публікації