НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)

“__” _____2022 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Інженерія програмного забезпечення комп’ютерних систем”

спеціальності 121 “Інженерія програмного забезпечення”

на тему: Спосіб побудови блокчейн системи

Виконав : студент __4__ курсу, групи _ІП-83_
(шифр групи)

_____Черевач Анатолій Миколайович_____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____асистент Волокита І. М._____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) професор, д.т.н., Сімоненко В. П._____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2022 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Інженерія програмного забезпечення комп’ютерних систем”

спеціальності 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2022 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Черевача Анатолія Миколайовича

1. Тема проєкту *Спосіб побудови блокчейн системи*
керівник проєкту Волокита Іван Миколайович, асистент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 10 червня 2022 року №1033-с

2. Термін здачі студентом закінченого 10 червня 2022 року

3. Вихідні дані до проєкту *технічна документація, теоретичні дані.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)

Розділ 1. Огляд блокчейн систем.

Розділ 2. Огляд технологій для розробки системи.

Розділ 3. Розробка та тестування системи.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) структурна схема системи, функціональна схема (діаграма класів), алгоритм дій програмного забезпечення.

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Сімоненко В. П.		

7. Дата видачі завдання «16» лютого 2022 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*10.01.2022-15.02.2022*	
2.	*Вивчення та аналіз завдання*	*15.02.2022-15.03.2022*	
3.	*Розробка архітектури та загальної структури системи*	*24.04.2022-07.05.2022*	
4.	*Програмна реалізація системи*	*07.05.2022-12.05.2022*	
5.	*Оформлення пояснювальної записки*	*12.05.2022-15.05.2022*	
6.	*Захист програмного продукту*	*20.05.2022*	
7.	*Передзахист*	*11.06.2022*	
8.	*Захист*	*25.06.2022*	

Студент-дипломник _____ Анатолій ЧЕРЕВАЧ
(підпис)

Керівник проєкту _____ Іван ВОЛОКИТА
(підпис)

АНОТАЦІЯ

У даній роботі було детально розглянуто блокчейн системи, принципи їх роботи, складові та алгоритми, що використовуються. На основі аналізу двох існуючих систем, було обрано рішення і технології, які лягли в основу створення власної блокчейн системи. В результаті роботи було розроблено блокчейн систему побудовану на основі P2P мережі. Розроблена програма дає можливість встановлювати зв’язок між учасниками мережі, здійснювати транзакції, синхронізувати дані між вузлами мережі та отримувати інформацію про ці дані. Програмний продукт був розроблений на мові JavaScript.

Ключові слова: блокчейн, P2P мережа, хешування, JavaScript.

ANNOTATION

In this project for a Bachelor's Degree, blockchain systems, algorithms and basic principles of their work were considered in detail. Based on the analysis of two existing systems, there were selected solutions and technologies that formed the basis for creating a blockchain system. As a result, a blockchain system based on the P2P network was created. The developed program allows users to establish communication between the participants, make transactions, synchronize the data between nodes and receive information about this data. The software product was developed in JavaScript.

Keywords: blockchain, P2P network, hashing, JavaScript

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпляра	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467200.002 ТЗ***	Спосіб побудови блокчейн	3		
			системи			
			Технічне завдання			
	А4	***ІАЛЦ.467200.003 ПЗ***	Спосіб побудови блокчейн	52		
			системи			
			Пояснювальна записка			
	А4	***ІАЛЦ.467200.004 Д1***	Спосіб побудови блокчейн	1		
			системи			
			Структурна схема			
			системи			
	А4	***ІАЛЦ.4672008.005 Д2***	Спосіб побудови блокчейн	1		
			системи			
			Функціональна схема			
			(діаграма класів)			
	А4	***ІАЛЦ.467200.006 Д3***	Спосіб побудови блокчейн	1		
			системи			
			оптимізації			
			Алгоритм дій програмного			
			забезпечення			
	А4	***ІАЛЦ.467200.007 Д4***	Спосіб побудови блокчейн	6		
			системи			
			Текст програмного коду			

Зм	*Лист*	*№ докум.*	*Підп*	*Дата*	***ІАЛЦ.467200.001 ОА***		
Розроб		Черевач А.М.			*Спосіб побудови блокчейн системи* Опис альбому	Літ.	Аркуш 1 / Аркушів 1
Перев		Волокита І.М.					
						НТУУ "КПІ" ФІОТ ІП-83	

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Спосіб побудови блокчейн системи*»

Київ – 2022

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку блокчейн системи, а також на подальшу підтримку та вдосконалення розробленої системи.

Областю застосування даної системи є багатокористувацькі застосунки для здійснення транзакцій, передачі і зберігання даних.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної системи є завдання для виконання роботи кваліфікаційно-освітнього рівня «бакалавр інженерії програмного забезпечення», який був затверджений факультетом “Інформатики та обчислювальної техніки” кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою та призначенням даної роботи є реалізація способу побудови блокчейн системи.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки даного дипломного проекту є офіційні документації, публікації та статті в мережі Інтернет на дану тему, науково-технічна література.

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Розроблена система має виконувати такі вимоги:

- Можливість простого і інтуїтивно-зрозумілого використання системи.

- Надійність і безпечність даних в системі.
- Можливість користувачам власноруч задавати початкові характеристики системи.
- Необхідна кількість інформації для коректного використання системи.

5.2. Вимоги до програмного забезпечення

- ОС Windows, Mac чи Linux.
- Node.js версії 16.0.0 або вище

5.3. Вимоги до апаратної частини

- ЦП не менше ніж Intel® Pentium ® G5400.
- ROM не менше ніж 16 ГБ.
- RAM не менше ніж 2 ГБ.

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	10.01.2022-15.02.2022
Вивчення та аналіз завдання	15.05.2022-15.03.2022
Розробка архітектури та загальної структури системи	24.04.2022-07.05.2022
Програмна реалізація системи	07.05.2022-12.05.2022
Виправлення помилок	12.05.2022-16.05.2022
Оформлення пояснювальної записки	17.05.2022

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Спосіб побудови блокчейн системи*»

Київ – 2022

ЗМІСТ

ВСТУП

Впродовж останніх 15 років технологія блокчейн виросла з ідеї серед вузького кола ентузіастів до одного з наймасштабніших напрямків в сфері інформаційних технологій [1]. Найбільш поширене застосування блокчейн знайшов у фінансовому секторі. Технологія передбачає зберігання транзакцій у блоках, які згодом додаються в кінець ланцюга без подальшої можливості модифікації та реплікуються на всіх вузлах мережі. Оскільки блокчейн надає можливість здійснювати безпечні, швидкі, доступні переводи коштів між користувачами мережі і обробку транзакцій без участі централізованого керуючого органу, він став основою для виникнення революційного платіжного засобу у вигляді криптовалюти [2]. Ринок цифрових валют розвивається стрімкими темпами. У 2019 році максимальна капіталізація ринку склала близько $350 млрд, а в 2021 році вона перевищила $3 трлн, більше половини з яких склали два лідери: Bitcoin та Ethereum [3].

Проте ефективне застосування блокчейну не обмежується лише фінансовими сервісами та цифровими активами. Він знаходить своє місце у все більшій кількості сфер. За допомогою блокчейну можна зробити більш прозорими, підняти на якісно вищий рівень, наприклад, такі процеси:

- управління даними і розподіленими реєстрами;
- реєстрація та ідентифікація користувачів;
- зберігання і обробка персональних даних;
- прозоре голосування без можливості підробки;
- реєстрація авторства і права власності;
- відстеження постачання та походження товарів і сировини;
- благодійна діяльність;

Метою написання дипломного проєкту є:

- Дослідження предметної області блокчейн систем
- Огляд та порівняння існуючих рішень
- Розробка власної блокчейн системи
- Опис результатів і підведення підсумків

Пояснювальна записка складається зі вступу, трьох розділів, висновків до кожного з розділів і загального висновку.

РОЗДІЛ 1. ОГЛЯД БЛОКЧЕЙН СИСТЕМ

1.1 Різновиди систем

Перед дослідженням безпосередньо блокчейн систем необхідно визначити види існуючих систем. За критерієм централізації виділяється три основних вида систем: централізовані, децентралізовані та розподілені [4]. Вони лежать в основі розвитку та еволюції мереж, фінансових систем, компаній, додатків та веб-сервісів.

Хоча всі види систем можуть функціонувати ефективно, деякі з них за своєю конструкцією більш стабільні та безпечні, ніж інші. Системи можуть бути дуже маленькими, з'єднуючи між собою лише кілька пристроїв та вузьке коло користувачів. Або вони можуть бути масштабними і охоплювати країни та континенти. У будь-якому випадку вони стикаються з одними й тими самими проблемами: відмовостійкість, витрати на обслуговування та масштабованість.

1.1.1 Централізовані системи

У централізованій системі, проілюстрованій рисунком 1.1, всі користувачі підключені до центрального власника мережі або сервера [4]. Центральний власник зберігає дані, до яких можуть отримати доступ інші користувачі та інформацію про користувачів. Інформація про користувача може включати профілі користувачів, контент користувача та безліч інших даних. Централізована система проста в установці та може швидко розвиватися.

Проте такий вид системи має важливе обмеження. Якщо сервер виходить з ладу, система перестає працювати належним чином і користувачі не можуть отримати доступ до даних.

Оскільки централізована система потребує центрального власника, для підключення всіх інших користувачів та пристроїв, доступність мережі залежить від цього власника. Також очевидними є проблеми безпеки, які виникають, коли всі дані про користувачів зберігає один власник. Відповідно централізовані системи більше не являються переважним вибором багатьох організацій.

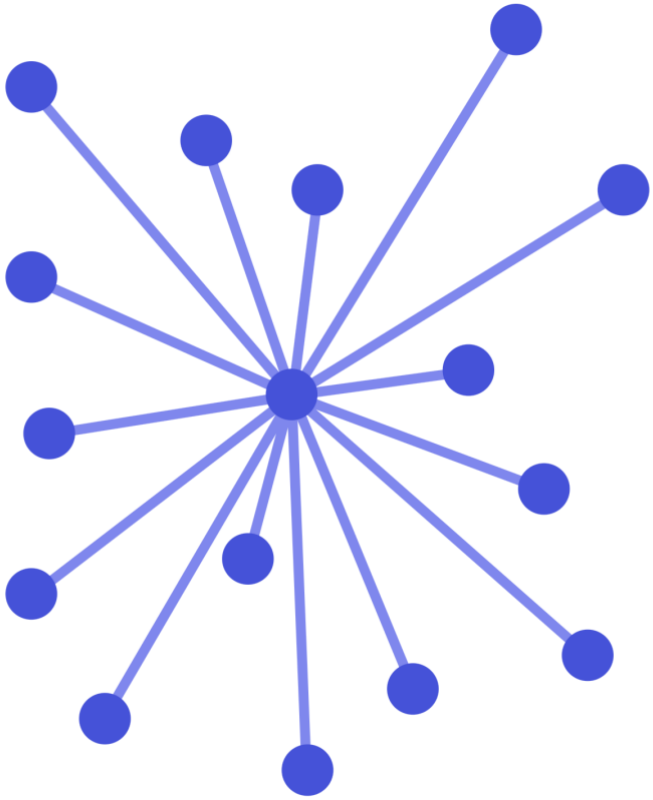


Рис. 1.1 – Централізована система

Переваги централізованої системи:

- Єдиний центр управління рішеннями та даними;
- Просте розгортання;
- Швидка розробка;
- Доступність в обслуговуванні;

Недоліки централізованої системи:

- Загроза атаки сервера, у результаті якої система повністю дестабілізується;
- Підвищені ризики безпеки та конфіденційності для користувачів;
- Непрозорість;

1.1.2 Децентралізовані системи

Як випливає з назви, децентралізовані системи не мають єдиного центрального власника . Натомість вони складаються з кількох власників, кожен з яких зазвичай зберігає копію ресурсів, до яких користувачі можуть отримати доступ [4]. Приклад систем изображено на рисунку 1.2

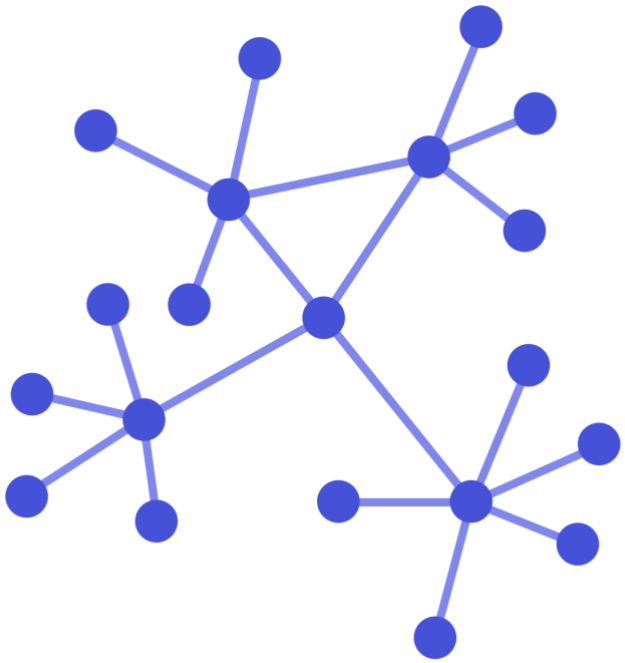


Рис 1.2 –Децентралізована система

Децентралізована система може бути так само вразлива до збоїв, як і централізована. Однак за своєю конструкцією система більш стійка до несправностей. Це пов'язано з тим, що при виході з ладу одного чи кількох центральних серверів інші продовжують працювати і обслуговувати

користувачів. Ресурси залишаються активними, якщо хоча б один із центральних серверів продовжує працювати. Зазвичай це означає, що власники системи можуть ремонтувати несправні сервери та вирішувати інші проблеми, в той час як сама система продовжує працювати у звичайному режимі.

Збої сервера в децентралізованій системі можуть вплинути на продуктивність та обмежити доступ до деяких даних. Але за критерієм загального часу безвідмовної роботи ця система значно краща за централізовану. Ще однією перевагою цієї конструкції є те, що час доступу до даних часто менший. Це є наслідком того, що власники можуть створювати вузли у різних регіонах, відповідно до більш або менш високої активності користувачів.

Однак децентралізовані системи, схильні до тих же ризиків безпеки та конфіденційності користувачів, що й централізовані системи.

Хоча їхня відмовостійкість вища, за це доводиться платити. Підтримка децентралізованої системи зазвичай дорожча.

Переваги децентралізованої системи:

- Менша ймовірність відмови, ніж у централізованої системи
- Швидший доступ користувачів до даних
- Дозволяє створити більш різноманітну та гнучку систему

Недоліки децентралізованої системи:

- Вищі витрати на обслуговування;
- Непостійна продуктивність у разі неправильної оптимізації;

1.1.3 Розподілені системи

Розподілена система схожа на децентралізовану в тому, що вона не має єдиного центрального власника. Головна відмінність полягає в тому, що в розподіленій системі власниками є всі вузли (рисунок 1.3).

Розподілена система дозволяє користувачам ділити володіння даними [4]. Апаратні та програмні ресурси також розподіляються між користувачами, що в деяких випадках може покращити продуктивність системи.

Розподілена система захищена від незалежної відмови компонентів, що може значно покращити час безвідмовної роботи.

Розподілені системи розвивалися внаслідок обмежень інших систем. У зв'язку зі зростаючими проблемами безпеки, зберігання даних та конфіденційності, а також постійною потребою у підвищенні продуктивності, розподілені системи стають природним вибором для багатьох організацій. Тому не дивно, що технології, які використовують розподілену систему, насамперед блокчейн, змінюють багато галузей.

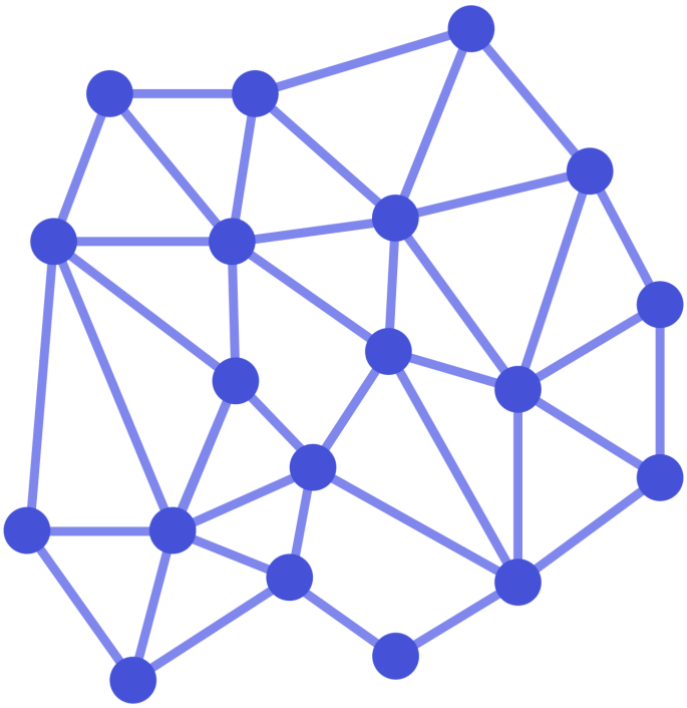


Рис 1.3 – Розподілена система

Переваги розподіленої системи:

- Відмовостійка;
- Прозора та безпечна;
- Сприяє спільному використанню ресурсів;
- Надзвичайно масштабована;

Недоліки розподіленої системи:

- Складніше розгорнути;
- Вищі витрати на обслуговування;

Підсумовуючи, централізовані системи допомогли розростатися першим мережам. Децентралізовані системи, менш схильні до збоїв і з більш швидким доступу, значно покращили старі системи і широко використовуються сьогодні. Однак тільки розподілені системи ефективно перерозподіляють ресурси і права по всій мережі. Таким чином, вони не тільки надзвичайно стійкі до збоїв, але й прозоріші за інші системи. Тому саме їх використання буде поширюватись у найближчі роки

1.2 Блокчейн

1.2.1 Концепція блокчейну

Як видно з назви (з англ. block - блок, chain - ланцюг), блокчейн — це ланцюжок блоків, які містять певну інформацію. Ця технологія була спочатку описана в 1991 році групою дослідників і була призначена для позначення часових міток цифрових документів, щоб їх було неможливо встановити заднім числом або підробити їх. Однак розповсюдження використання блокчейн не отримав, допоки анонімний розробник під псевдонімом Сатоші Накамото не адаптував його в 2009 році для створення криптовалюти Bitcoin [1].

Блокчейн — це незмінний цифровий реєстр даних, який дублюється і розподіляється по вузлах підключенних до однорангової мережі блокчейну. Така форма зберігання інформації дозволяє лише додатвати нові дані та не дозволяє проводити редагування, видалення, або шахрайство в будь-якому вигляді. Кожна транзакція у реєстрі підтверджена цифровим підписом автора, такий механізм захищає її від підробок. При цьому, транзакція, створена на

одному вузлі, є видимою для всіх інших вузлів. Таким чином учасники мережі можуть її перевірити і прийняти, або відхилити в залежності від результату перевірки.

Рисунок 1.4 демонструє структуру ланцюга блоків та їх склад. Кожен блок в блокчейні містить хеш самого блоку, хеш попереднього блоку та деякі дані [5]. Лише генезис блок не має хеша попереднього блоку, оскільки він є першим в ланцюгу. Дані, що зберігаються всередині блоку, залежать від типу блокчейну. Блокчейн Bitcoin, наприклад, зберігає масив транзакцій, кожна з яких містить інформацію про відправника, одержувача і кількість відправлених коштів.

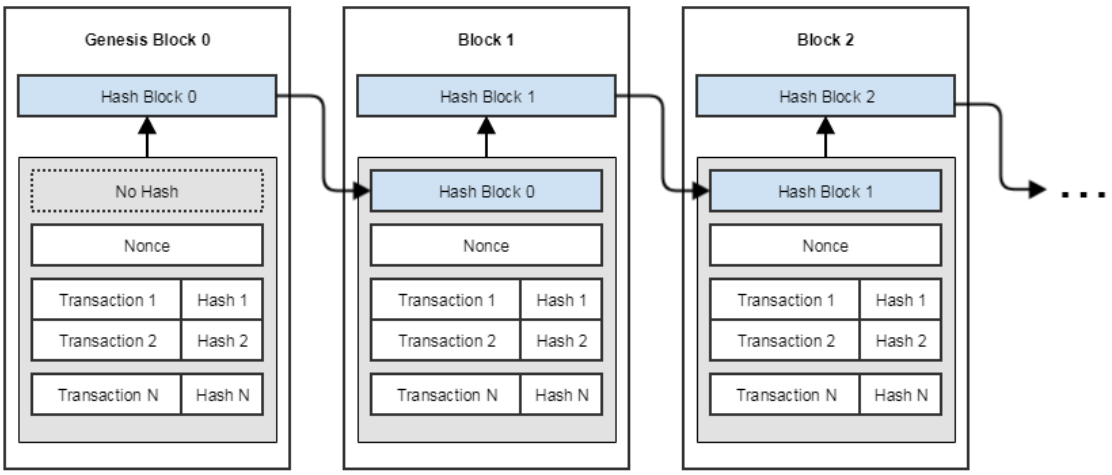


Рис 1.4 - Структура ланцюга блоків

Хеш блоку можна порівняти з відбитком пальця. Хеш ідентифікує блок, весь його вміст і завжди є унікальним, як відбиток пальця. Після створення блоку всі його дані передаються до хеш-функції яка повертає строку символів фіксованої довжини. Зміна будь-яких даних всередині блоку призведе до зміни хеша.

Хеш попереднього блоку є свого роду посиланням на попередню ланку ланцюга блоків. Таким чином, зміна одного блоку зробить усі наступні блоки недійсними. Але використання хешів недостатньо для запобігання фальсифікації. Сучасні обчислювальні пристрої можуть перебирати сотні тисяч

хешів за секунду. Це означає, що зловмисник можете втрутитися в блок і перерахувати всі хеші інших блоків, щоб зробити свій блокчейн дійсним.

Для запобігання використанню цієї вразливості, блокчейни реалізують те, що називається доказом роботи (Proof-of-Work). Proof-of-Work – це механізм, який уповільнює створення нових блоків. У випадку з Біткоінами обчислення необхідного доказу роботи та додавання нового блоку до ланцюжка займає близько 10 хвилин. Цей механізм ускладнює підробку блоків, тому що в разі зміни одного блоку, необхідно перерахувати підтвердження роботи для всіх наступних блоків.

Безпека блокчейну забезпечується використанням хешування та механізму підтвердження роботи. Але є ще один спосіб убезпечити блокчейни, а саме розповсюдження. Замість того, щоб центральний орган керував ланцюгом, блокчейн використовує мережу P2P, до якої може приєднатися будь-хто (за умови, що блокчейн є загальнодоступним). Коли новий учасник приєднується до цієї мережі, він стає вузлом і отримує повну копію блокчейну. Потім цей вузол може використовувати копію блокчейну для перевірки дійсності. Коли учасник блокчейну створює новий блок, цей блок надсилається всім вузлам мережі. Потім кожен вузол перевіряє блок, щоб переконатися, що він не був підроблений. Після підтвердження, кожен вузол додає цей блок у свій власний блокчейн. Вузли в мережі в кінцевому підсумку досягають консенсусу: вони домовляються про те, які блоки дійсні, а які ні. Блоки, які були змінені, будуть відхилені іншими вузлами мережі.

Таким чином, щоб успішно підробити блокчейн, потрібно змінити всі блоки в ланцюжку, повторити обчислення доказу роботи для кожного з блоків, наступних за зміненим блоком, і взяти під контроль понад 50% вузлів однорангової мережі. Тільки в такому випадку підроблений блок буде прийнято всіма іншими учасниками блокчейну. Але така атака потребує велику кількість обчислювальних ресурсів і обладнання, особливо це стосується великих мереж. Витрати на досягнення необхідної кількості ресурсів перевищують

потенціальну вигоду шахрайства, тому такі атаки в реальності є економічно недоцільними.

1.2.2 Структура блока

Основною структурною одиницею блокчейна є блок. Дані в блоках можуть відрізнятися в залежності від потреб та особливостей реалізації конкретного блокчейну [5]. Кожен блок складається з двох головних частин: заголовка (Head) і тіла (Payload). На рисунку 1.5 зображено базовий приклад структури блока: заголовок і його поля сірим кольором, а тіло і його складові помаранчевим.

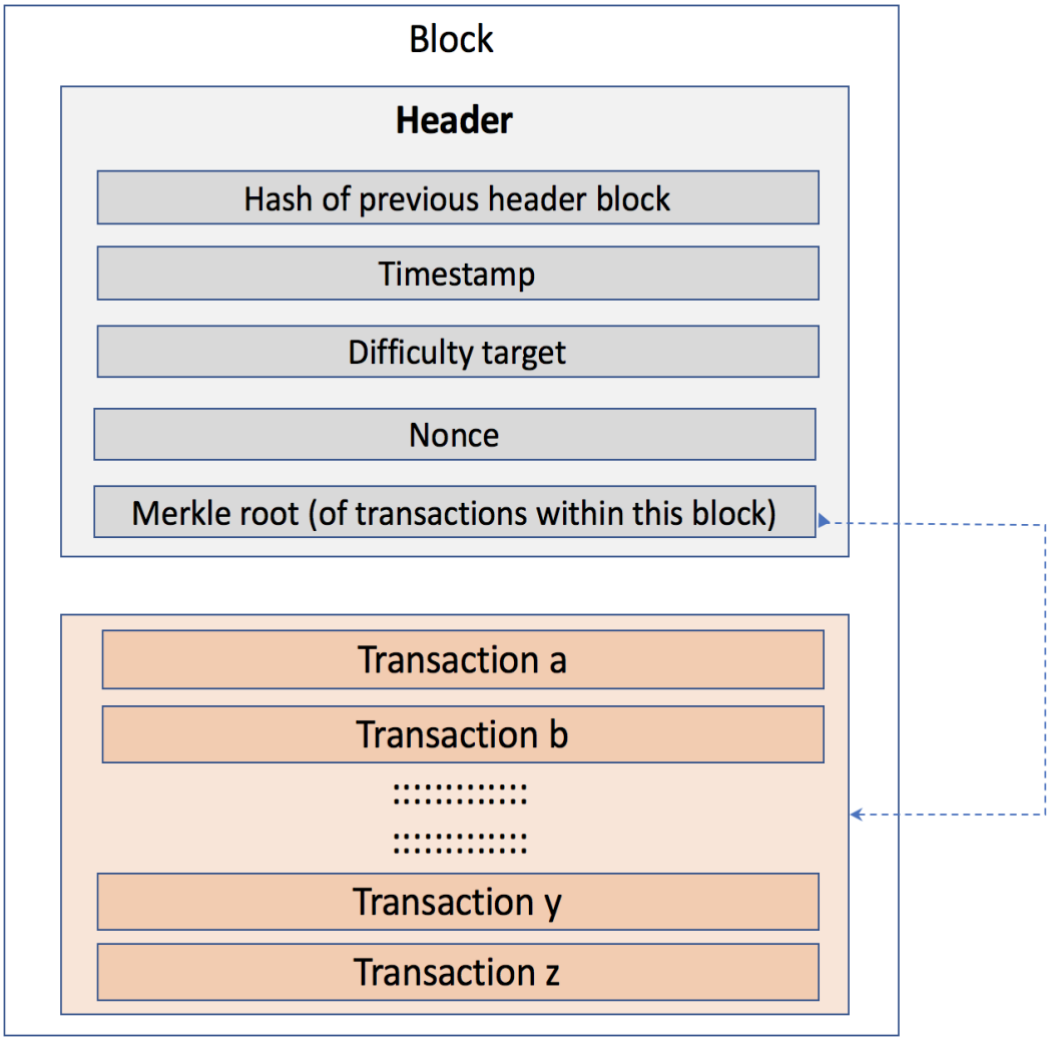


Рис 1.5 - Структура блока

Head містить інформацію, яка відповідає за стабільність, а також іммутабельність мережі.

У класичному блокчейні Head містить такі поля:

- Номер версії блоку
- Хеш блоку
- Хеш попереднього блоку
- Хеш всіх транзакцій в поточному блоку
- Часову мітку, коли цей блок був створений
- Bits і Nonce, які використовуються в майнінгу

Payload в класичному випадку містить список всіх транзакцій між учасниками мережі, які повинні бути збережені в цьому блоці. Транзакція в свою чергу складається з:

- Адреси відправника
- Адреси отримувача
- Суми переводу
- Підпису приватним ключем

1.3 Порівняння існуючих систем

1.3.1 Ethereum

Ethereum (рисунок 1.6) – це децентралізована блокчейн платформа, яка створює однорангову мережу, яка безпечно виконує та перевіряє код програми, який називається смарт-контрактами. В основі платформи лежить внутрішній токен Ether, який дозволяє користувачам здійснюватиторгівлю, заробляти відсотки на своїх інвестиціях шляхом стейкингу, використовувати та зберігати незамінні токени (NFT), торгувати криптовалютами, грати в ігри та брати участь у соціальних мережах [6].

У 2013 році були представлені смарт-контракти, які є автоматизованими незмінними операторами, що дозволяють створювати децентралізовані додатки (DApps).

Рис 1.6 - Логотип Ethereum

Різниця між Ethereum та Ether: Ether — це цифрова валюта, яка може використовуватися для фінансових операцій, інвестицій і як засіб збереження вартості. Ether зберігається та обмінюється в мережі блокчейн Ethereum. А зберігати дані або виконувати децентралізовані програми можна через мережу Ethereum. Люди можуть розміщувати програмне забезпечення на блокчейні Ethereum, а не на сервері, який належить і контролюється Google або Amazon, де лише одна компанія контролює дані. Оскільки немає єдиного органу, який би що-небудь регулював, споживачі мають повний контроль над своїми даними та повний доступ до програми.

Переваги Ethereum:

- Децентралізована архітектура Ethereum ефективно розподіляє дані та довіру між учасниками мережі, усуваючи потребу в центральному органі для керування системою та посередництва в транзакціях.

- Замість того, щоб створювати реалізацію блокчейну з нуля, організації можуть швидко створювати та адмініструвати приватні мережі блокчейну.
- Основна мережа Ethereum демонструє, що може функціонувати з сотнями вузлів і мільйонами користувачів.

  Більшість конкурентів бізнес-блокчейну керують мережами з значно меншою кількістю вузлів.
- Мережі, створені на Ethereum, можуть перевершувати загальнодоступну мережу та збільшуватись до сотень транзакцій в секунду або більше залежно від налаштування мережі, завдяки консенсусу Proof of Authority та індивідуальним обмеженням часу генерації блоків.

Недоліки Ethereum:

- Використання складної мови програмування. Незважаючи на те, що Ethereum використовує мову програмування, подібну до C++, Python і Java, вивчення Solidity, мови Ethereum, може бути складним. Однією з найбільш значущих проблем є нестача класів, зручних для початківців.
- Можливі проблеми з масштабуванням. На відміну від блокчейнів з єдиним призначенням, Ethereum має реєстр, платформу для смарт-контрактів тощо, які можуть призвести до помилок, збоїв та злому.

1.3.2 Waves

Waves (рисунок 1.7) — це багатоцільова блокчейн-платформа, яка підтримує багато способів використання, включаючи децентралізовані додатки (DApps) та смарт-контракти. З моменту запуску у Waves була одна мета: масове впровадження технології блокчейн у глобальному масштабі. Платформа Waves

схожа на Ethereum в тому сенсі, що це екосистема блокчейну, яку користувачі Waves можуть використовувати для запуску власних власних токенів і криптопроектів на базі існуючої криптовалюти [7]. Еквівалентом такої функції в Ethereum є ERC-20.

Рис 1.7 - Логотип Waves

Однією з унікальних особливостей, які відрізняють Waves від інших подібних проектів, таких як Ethereum, є можливість для користувачів створювати токени криптовалюти в інтерфейсі гаманця. Користувачі також можуть вільно обмінювати ці токени через вбудований децентралізований обмін. Особливо децентралізований обмін можна розглядати як ефективне рішення в галузі, яка все ще страждає від централізованих мереж і різних аферистів, які маніпулюють відчуттям безпеки, яке приносять деякі напівцентралізовані проекти.

Блокчейн Waves спочатку був розроблений у два шари, причому перший «основний» рівень був централізованим і містив «повні вузли», які мають повний запис усіх транзакцій. Другий, «зовнішній» рівень децентралізований і складається з легких вузлів, які зберігають лише записи останніх транзакцій, які передаються повним вузлам для реєстрації. Якщо є розбіжності між історіями транзакцій, блокчейн Waves тимчасово розгалужується, доки не вдасться визначити правильну версію відповідно до більш довгої гілки.

Переваги ексосистеми Waves:

- Можливість створення власних токенів і додатків.

- Наявність децентралізованої біржі The Waves. Децентралізована біржа, яка зберігає гаманець Waves, має ряд конкурентних переваг серед інших криптовалютних проектів. Зокрема, надає можливість створення своїх токенів та торговлі ними.
- Waves менш шкідлива для екології. Більшість традиційних криптовалютних проектів досі покладаються на алгоритм Proof of Work для перевірки транзакцій на блокчейні. Для цього потрібні майнери і, великий обсяг витрат електроенергії та обладнання. Waves відійшов від Proof of Work і використовує систему Proof of Stake.

Недоліки Waves:

- Неповна децентралізація, оскільки блокчейн складається з основного централізованного рівня і зовнішнього децентралізованного.
- Висока конкуренція з боку Ethereum та інших платформ для створення децентралізованих додатків
- Waves все ще є платформою, що розвивається, що може змусити інвесторів відкладати фінансові зобов’язання.

ВИСНОВОК ДО РОЗДІЛУ 1

У першому розділі було проведено аналіз предметної області блокчейн систем. Спочатку були описані і порівняні існуючі види систем: централізовані, децентралізовані та розподіленні. Виявлені їх переваги та недоліки з точки зору надійності, безпеки, масштабованості та відмовостійкості. Визначено за якими саме параметрами пов’язані блокчейн та розподілені системи.

Наступним етапом було дослідження блокчейну як концепції. Було наведено визначення, історичний шлях розвитку технології, принципи на яких базується технологія. А саме, відкритість, прозорість, безпечність та іммутабельність даних. Далі були наведені деякі відомості щодо безпосередньо технічних та структурних складових технології. Описано і проілюстровано яким чином будується ланцюг блоків та за рахунок чого зумовлена неможливість його редагування. Також наведено детальну структуру базової одиниці блокчейна – блока, та структуру основних даних які зберігає класичний блокчейн – транзакції.

У якості прикладів існуючих систем розглянуто дві платформи: Ethereum і Waves. Було досліджено принципи їх роботи, можливості, переваги і недоліки.

РОЗДІЛ 2. ОГЛЯД ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ СИСТЕМИ

2.1 Архітектура мережі

Оскільки блокчейн система має на увазі постійну взаємодію між учасниками мережі задля швидкого обміну оновленнями у блокчейні і підтримки актуальності данних на кожному вузлі, вибір і побудова архітектури мережі є одним з головним аспектів при розробці. два основних архітектурних підходи до побудови мереж:

- клієнт-серверний
- одноранговий (Peer-to-peer)

Далі будуть розглянуті обидві моделі щоб дізнатися, про способи обміну даними між учасниками мережі, порівняти їх та зробити обґрунтований вибір враховучи особливості предметної області.

2.2.1 Клієнт-серверна архітектура

Клієнт-серверна архітектура побудови мережі, проілюстрована рисунком 2.1, включає багато клієнтів або робочих станцій, які підключені принаймні до одного центрального сервера. Більшість програм і даних встановлюється на безпосередньо на сервер. Коли клієнту необхідно отримати доступ до ресурсів і даних, він отримує доступ до них з сервера.

Сервер може мати різні каталоги даних доступні для запиту клієнтами, в залежності від прав користувача у мережі.

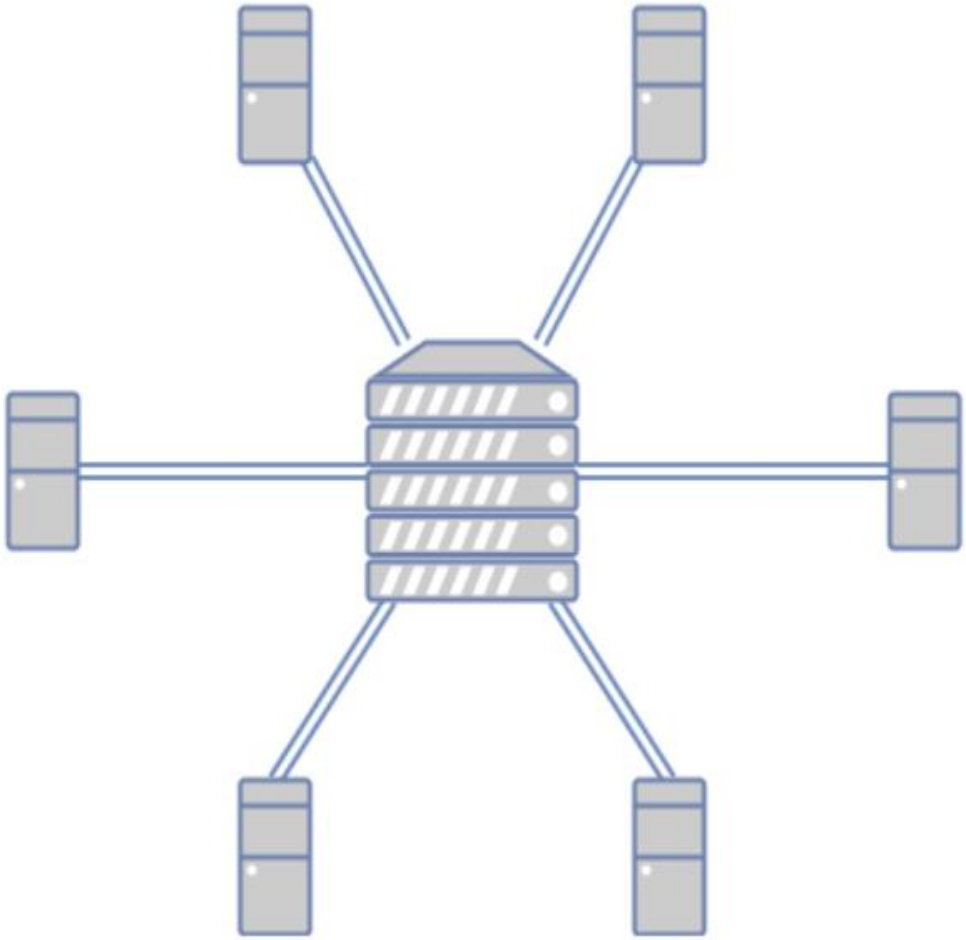


Рис 2.1 - Клієнт-серверна архітектура

Клієнт-серверні мережі мають більшу швидкість доступу, оскільки вони розроблені для підтримки багатьох клієнтів. Вся системна логіка та інформація приховані всередині сервера, що дозволяє знизити вимоги до продуктивності клієнтських пристроїв та забезпечити високу швидкість обробки даних [8]. Також це полегшує оновлення програмних додатків і файлів, оскільки вони зберігаються лише на одному пристрої. Проте, якзо сервер виходить з ладу, мережаґ перестає працювати належним чином і користувачі не можуть отримати доступ до даних.

2.1.2 Peer-to-peer архітектура

На відміну від архітектури клієнт-сервера, однорангова організація мережі, ілюстрована рисунком 2.2, дозволяє зберігати працездатність за будь-якої кількості та будь-якого поєднання доступних вузлів. Кожен з вузлів може надсилати запити іншим на надання ресурсів у межах цієї мережі і таким чином виступати в ролі клієнта [8]. Будучи сервером, кожна машина повинна бути здатна обробляти запити від інших машин в мережі, відсилати те, що було запитано. Також машина повинна виконувати деякі допоміжні та адміністративні функції. Наприклад, зберігати список інших відомих машин-сусідів і підтримувати його актуальність.

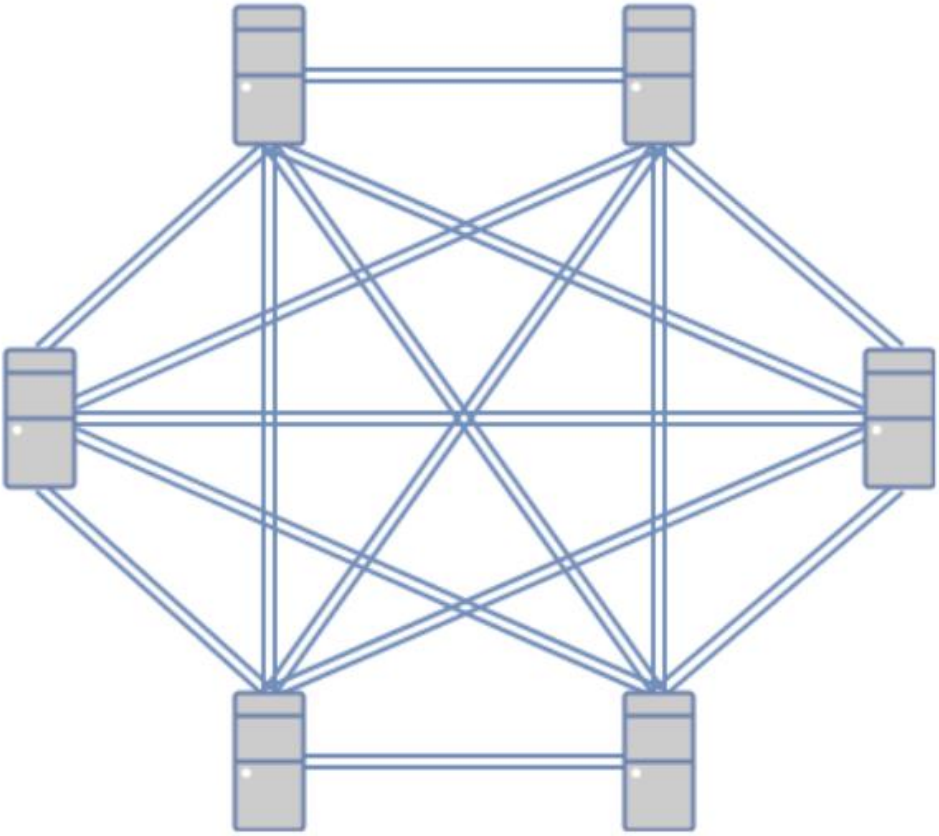


Рис 2.2 - Однорангова P2P архітектура

Оскільки блокчейн система має на увазі рівноправність користувачів та високу працездатність, доцільним вибором основи мережі, що розробляється, є однорангова P2P архітектура.

2.2 Мова програмування

У якості мови програмування системи було обрано мову JavaScript [9] та платформу Node.js [10], яка розширює можливості використання мови в тому числі для створення серверних застосунків, роботи з базами даних та бібліотеками. Перевага у виборі конкретних технологій зумовлена робочим інтересом автора, але обрані технології є об’єктивно рівними або кращими серед представлених

2.2.1 C++

C++ є однією з найпопулярніших мов програмування і вона стала основною мовою в індустрії блокчейну [11]. На додачу до своєї корисності у розробці блокчейну, вона також має ті ж принципи, що і блокчейн. Такі принципи включають поліморфізм, приховування даних, абстрагування та інкапсуляцію для запобігання зміни даних.

Перша в історії реалізація блокчейну, Bitcoin, спочатку була написана на C++. Це гарна мова програмування для блокчейну завдяки своїм розширеним можливостям багатопоточності та примітивному контролю над пам’яттю. Об’єктно-орієнтовані функції мови дають розробникам можливість пов’язувати дані та методи, призначені для їх обробки. Це схоже на те, як блокчейн використовує криптографічні ланцюги для зв’язування блоків разом.

Переваги:

- Висока швидкість

- Незалежність і наявність кількох платформ
- Статична типізація

Недоліки:

- Складний синтаксис і важке відлагодження
- Надлишковість коду
- Не підтримує garbage collection

Основні криптовалюти, які використовують C++:

- Bitcoin
- Ethereum
- Litecoin
- Dogecoin
- Stellar
- Ripple

2.2.2 Java

Java є об'єктно-орієнтованою, класовою і конкурентною мовою. Вона популярна серед розробників, оскільки її можна легко запускати на будь-якому комп’ютері, на якому встановлено середовище виконання Java (JRE).

Блокчейн-розробники віддають перевагу використанню Java через її високу мобільність [11]. Програми, написані на Java, можуть працювати майже на всіх обчислювальних пристроях, оскільки вони не залежать від специфічної архітектури системи. Замість неї вони використовують універсальну віртуальну машину Java для виконання коду.

Переваги:

- Об'єктно-орієнтоване програмування
- Простіша у розробці, ніж такі мови, як C++ та C
- Багато функціональних бібліотек
- Немає проблем із виділенням пам'яті

Недоліки:

- Для запуску потрібна віртуальна машина Java
- Повільніша, ніж такі мови, як C++

Використання Java в Blockchain:

- NEM (платформа однорангової криптовалюти)
- Блокчейн IBM
- Ethereum
- Контракти NEO
- BitcoinJ (реалізація біткойна на Java)
- Контракти Hyperledger

2.2.3 Phyton

Мова Python відома своєю простотою. У неї є велика активна спільнота, яка випустила такі бібліотеки, як NumPy, Pandas і SciPy, які використовуються для різних технічних застосунків у науці, математиці та інженерії.

Python підходить для обробки чисел, широко використовується для обробки, аналізу та візуалізації даних. Серед багатьох інших можливостей її можна використовувати в блокчейні для написання смарт-контрактів [11].

Переваги:

- Простий синтаксис
- Швидкий та масштабований
- Він має багато бібліотек і фреймворків
- Має велику спільноту

Недоліки:

- В основному використовується для обробки даних
- Деякі бібліотеки недостатньо задокументовані

Використання Python в Blockchain:

- Реалізація Ethereum (pyethereum)

- Смарт-контракти для Hyperledger
- Створення договорів для NEO

2.2.4 Javascript

Спочатку використання JavaScript [9] для розробки блокчейну викликало складності у розробників. Однак з появою середовища виконання JavaScript за межами браузера - Node.js [10] з'явилась можливість створювати блокчейн-додатки.

Великою перевагою використання JavaScript є те, що не потрібно турбуватися про інтеграцію, використовуючи його для розробки блокчейну. Це дозволяє повністю зосередити зусилля розробника на логіці програми.

JavasScript також стає все більш популярним у розробці блокчейну, оскільки він може обробляти асинхронний код. Ця можливість має вирішальне значення для блокчейну через можливість одночасного виконання мільйонів транзакцій. Асинхронна природа JavaScript дозволяє програмі виконувати кілька дій одночасно, покращуючи тим самим продуктивність програми та реагування програмного забезпечення.

Переваги:

- Мова асинхронного програмування
- Підтримує всі парадигми програмування
- В основі полягають прототипи

Недоліки:

- Знижує продуктивність при виконанні важких обчислювальних завдань
- Модель асинхронного програмування ускладнює підтримку коду

Використання JavaScript в Blockchain:

- Ethereum.js
- Web3.js

I Ethereum.js, і Web3.js допомагають підключити програму до смарт-контрактів і мережі Ethereum.

- NEO
- Raiden Network (високошвидкісна інфраструктура, побудована на основі Ethereum)

2.3 Алгоритм хешування

При розробці програмних компонентів блокчейн систем скрізь використовуються хеш-функції для хешування блоків, цифрові підписи для верифікації транзакцій, алгоритми для генерації пар ключів, адрес.

Для потреб хешування в системі обрано алгоритм SHA-256. Хеш-функція SHA-256 приймає вхідні дані випадкового розміру і повертає вихідний рядок фіксованого розміру, приклад на рисунку 2.3.

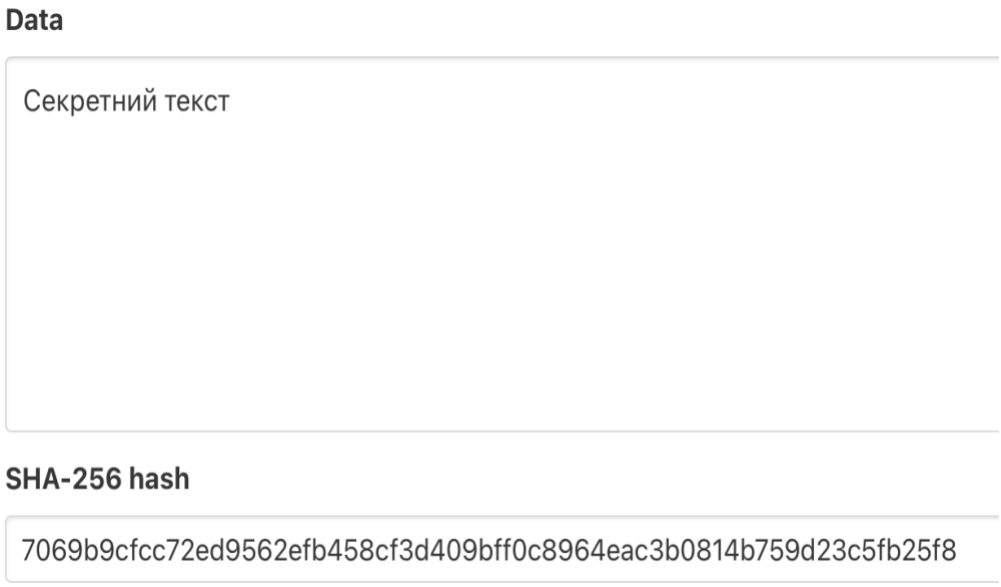


Рис 2.3 – Приклад хешування даних за допомогою SHA-256

Хеш-функції є потужними, оскільки вони «односторонні». Хеш-функція завжди видає однакове хеш-значення, якщо надається однакове вхідне значення, а складні алгоритми хешування використовуються для мінімізації ймовірності того, що більше ніж один вхід надасть однакове хеш-значення [12].

Це означає, що будь-хто може використовувати хеш-функцію для отримання результату, коли має вхідні дані. Однак неможливо використовувати вихідні дані хеш-функції для відновлення вхідних даних. Ця властивість робить її ідеальною для застосування в блокчейні.

Хеш-функція SHA-256 використовується зазвичай використовується у майнінгу для виконання доказу роботу (Proof-of-work) та створення адрес [12].

2.4 Бібліотеки

2.4.1 .Crypto

Crypto – це вбудований в Node.js модуль для роботи з криптографією. Він включає в себе широкий набір класів і функцій для хешування, шифрування, дешифрування, підписів та різноманітних перевірок [13]. На відміну від інших бібліотек Crypto не потребує попереднього встановлення та конфігурації. Crypto дозволяє створювати стійкі, детерміновані хеші фіксованої довжини. Для хешованих даних пароль не можна розшифрувати за допомогою заздалегідь визначеного ключа, на відміну від зашифрованих даних. Також за його допомогою можна генерувати і перевіряти цифрові підписи, та використовувати їх для підпису даних.

2.4.2 WS

Оскільки архітектурним способом побудови системи обрано однорангову P2P мережу, у якості способу комунікації між вузлами мережі обрано WebSocket. WebSocket - це протокол зв'язку, який забезпечує повнодуплексні канали зв'язку через єдине TCP-з'єднання, встановлене між вузлами за допомогою «класичного HTTP-механізму» рукостискання (зображено на

рисунку 2.4), реалізованого за допомогою заголовків запиту/відповіді [14]. Простими словами: між клієнтом і сервером існує постійне з'єднання, і обидві сторони можуть почати надсилати дані у будь-який момент. WS – популярна WebSocket бібліотека для Node.js, яка допомагає керувати процесами та комунікаціями за протоколом WebSocket [15].

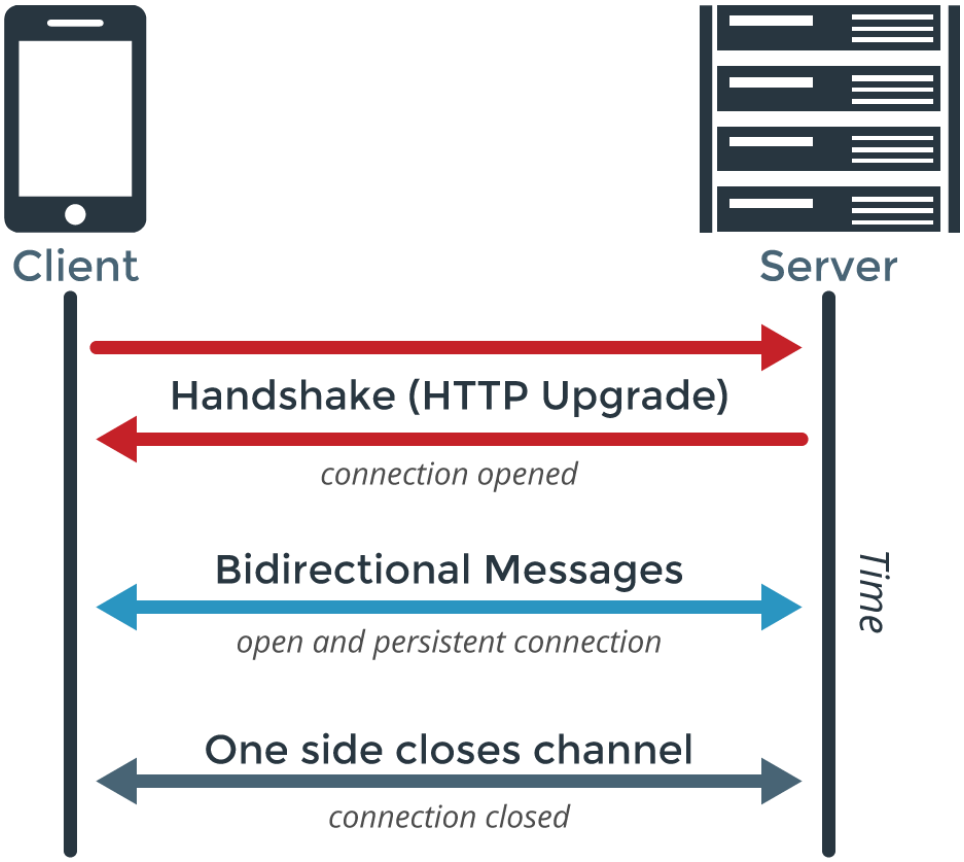


Рис 2.3 – Принцип взаємодії протоколу WebSocket

2.4.3 Elliptic

Elliptic являє собою JavaScript бібліотеку для роботи з криптографічними функціями пов'язаними з еліптичними кривими [16]. Зокрема, в рамках роботи цікавить функціонал з електронними підписами та генерацією пар ключів. Наприклад, в Біткоіні використовується ECDSA - Elliptic Curve Digital Signature Algorithm, який використовує еліптичні криві (elliptic curve) та кінцеві поля (finite field) для підпису даних, щоб третя сторона могла підтвердити

автентичність підпису, виключивши можливість її підробки. В ECDSA для підпису та верифікації використовуються різні процедури, що складаються з кількох арифметичних операцій. Найчастіше використовувані еліптичні криві мають випадкову структуру, але обрана для роботи крива secp256k1 побудована особливим невипадковим чином, що дозволяє проводити особливо ефективні обчислення [17]. В результаті, якщо реалізація досить оптимізована, вона зазвичай на 30% швидше ніж інші криві. Детальний опис принципів самих алгоритмів виходить за рамки досліджуваної теми проєкту, достатньо перевіреної і доведеної на практиці надійності.

ВИСНОВОК ДО РОЗДІЛУ 2

У другому розділі було розглянуто технології для використання при побудові системи. Було проаналізовано два архітектурних підходи до побудови мережі. На основі аналізу обрано модель однорангової P2P архітектури, яка задовольняє потребу системи у зв'язках всіх вузлів мережі один з одним та їх рівноправності.

Наступним етапом було обрання мови програмування для реалізації. Проведено порівняння чотирьох основних мов, які використовуються у блокчейні: C++, Java, Phyton, JavaScript. Описані їх відмінності та особливості, наведено переваги і недоліки кожної з них, а також приклади використання в блокчейні. Остаточним вибором стала мова JavaScript. Також обрано NodeJS якості платформи, яка дозволяє створювати серверне програмне забезпечення за допомогою JavaScript.

Оскільки в блокчейні широко та часто використовуються функції хешування, у якості основного алгоритма хешування обрано SHA-256.

У якості допоміжних бібліотек обрано три:

- Crypto – для використання криптографічних функцій і алгоритмів хешування
- Elliptic – для генерації електронних підписів і подальшого використання їх для підписання і перевірок.
- WS – для використання протоколу WebSocket при побудові P2P мережі

РОЗДІЛ 3. РОЗРОБКА І ТЕСТУВАННЯ СИСТЕМИ

3.1 Розробка програмних компонентів

Далі буде описано ключові сутності блокчейн системи у якості компонентів програми, їх функціонал та реалізацію у вигляді класів.

3.1.1 Транзакція

Сутність транзакції включає в себе адресу гаманця відправника, адресу гаманця одержувача та суму, яку надсилається. Окрім цього транзакція може містити такий різновид винагороди майнера, який називається gas. Ця винагорода оплачує енергію використану для майнінгу блоків та дозволяє користувачам пришвидшити обробку їх транзакцій, за рахунок більш привабливої винагороди для майнерів. Проте, оскільки використання системи не обмежується лише здійсненням переказів і в майбутньому припускає інтеграцію в різноманітних сервісах, транзакція буде містити поле для довільних даних (повідомлень, статусів, тощо).

Після створення транзакції, вона додається до пулу транзакцій, і при створенні нового блока, всі незавершені транзакції переміщуються до даних цього блоку. Для запобігання помилковим транзакціям, використовується механізм підписання парою ключів. Ця пара складається з двох ключів : приватний і відкритий.

Відкритий ключ може бути доступним іншим учасникам мережі як адреса гаманця, закритий ключ використовується для підписання транзакцій.

Усі транзакції створені в системі будуть екземплярами класу Transaction, який складається з:

- Конструктора (рисунок 3.1), який приймає адресу отримувача, адресу відправника, суму, опціонально gas, який за замовчуванням дорівнює 0, та опціонально дані рядкового типу, за замовчуванням значення дорівює пустому рядку, і створює екземпляр класу.

```
constructor(from, to, amount, gas = 0, data = "") {
  this.from = from;
  this.to = to;
  this.amount = amount;
  this.gas = gas;
  this.data = data;
}
```

Рис 3.1 – Конструктор транзакції

- Методу sign (рисунок 3.2), який приймає пару ключів, перевіряє чи співпадає публічний ключ з адресою відправника, створює цифровий підпис і додає його до екземпляра класу транзакції.

```
sign(keyPair) {
  if (keyPair.getPublic("hex") === this.from) {
    this.signature = keyPair
      .sign(
        sha256(this.from + this.to + this.amount + this.gas + this.data),
        "base64"
      )
      .toDER("hex");
  }
}
```

Рис 3.2 – Метод підпису транзакції

- Статичного методу isValid (рисунок 3.3), який приймає транзакцію і ланцюг, перевіряє валідність даних транзакції, наявність у відправника достатніх для здійснення транзакції коштів та справжність цифрового підпису.

```
static isValid(tx, chain) {
  return (
    tx.from &&
    tx.to &&
    tx.amount &&
    (chain.getBalance(tx.from) >= tx.amount + tx.gas ||
      tx.from === MINT_PUBLIC_ADDRESS) &&
    ec
      .keyFromPublic(tx.from, "hex")
      .verify(
        sha256(tx.from + tx.to + tx.amount + tx.gas + tx.data),
        tx.signature
      )
  );
}
```

Рис 3.3 – Метод перевірки тразакції

3.1.2 Блок

Сутність блока включає в себе хеш блока, який отримується шляхом передачі даних блока у функцію алгоритма SHA-256, хеш попереднього блока, часову мітку створення блока, параметр nonce, значення якого отримується в результати виконання алгоритму Proof-of-Work та масив транзакцій

Клас Block є батьківським для всіх блоків майбутнього ланцюга. Він містить наступні методи та властивості:

- Конструктор (рисунок 3.4), який приймає timestamp (часову мітку в мілісекундах) за замовчуванням час створення блока та масив даних, в даному випадку транзакцій.

```
constructor(timestamp = Date.now().toString(), data = []) {
  this.timestamp = timestamp;
  this.data = data;
  this.prevHash = "";
  this.hash = Block.getHash(this);
  this.nonce = 0;
}
```

Рис 3.4 – Конструктор блоку

- Статичний метод getHash (рисунок 3.5), який приймає блок, дані якого хешуються функцією sha256, і повертає хеш цього блока.

```
static getHash(block) {
  return sha256(
    block.prevHash +
      block.timestamp +
      JSON.stringify(block.data) +
      block.nonce
  );
}
```

Рис 3.5 – Статичний метод генерації хеша блоку

- Метод mine (рисунок 3.6), який приймає числовий параметр складності. Складність потрібна для того, щоб ускладнити пошук необхідного хеша блоку і регулювати швидкість генерації блоків. Значення складності дорівнює кількості нулів, з якої повинен починатися хеш блоку. Далі виконується цикл поки хеш блоку не задовільнить умові, у якому при кожній ітерації значення nonce збільшується на одиницю і перераховується хеш блока.

```
mine(difficulty) {
  const prefix = "0".repeat(difficulty)
  while (!this.hash.startsWith(prefix)) {
    this.nonce++;
    this.hash = Block.getHash(this);
  }
}
```

Рис 3.6 – Метод майнінгу блока

- Статичний метод hasValidTransactions (рисунок 3.7),який приймає блок і ланцюг, перевіряє всі транзакції в блоці, відповідність винагород та комісій, та наявність лише однієї транзакції відправленої з адреси карбування монет у якості винагороди майнера.

```
static hasValidTransactions(block, chain) {
  let gas = 0,
    reward = 0;

  block.data.forEach((transaction) => {
    if (transaction.from !== MINT_PUBLIC_ADDRESS) {
      gas += transaction.gas;
    } else {
      reward = transaction.amount;
    }
  });

  return (
    reward - gas === chain.reward &&
    block.data.every((transaction) =>
      Transaction.isValid(transaction, chain)
    ) &&
    block.data.filter(
      (transaction) => transaction.from === MINT_PUBLIC_ADDRESS
    ).length === 1
  );
}
```

Рис 3.7 – Метод перевірки валідності транзакцій в блоці

3.1.3 Блокчейн

Блокчейн реалізовано за допомогою класу Blockchain. Майбутній ланцюг блоку буде екземпляром цього класу. Він містить наступні методи та властивості:

- Конструктор (рисунок 3.8), який створює екзмепляр блокчейну. При створенні екзмепляру відбувається карбування монет системи. На цьому етапі встановлюються наступні параметри системи: reward – фіксована винагорода за майнінг блока, blockTime – час у мілісекундах, який визначає час генерації одного блока, difficulty – числовий показник складності, який в майбутньому буде коригуватися для підтримки стабільного blockTime.

```
constructor() {
  const initalCoinRelease = new Transaction(
    MINT_PUBLIC_ADDRESS,
    INITIAL_ADRESS,
    100000000
  );
  this.transactions = [];
  this.chain = [new Block(Date.now().toString(), [initalCoinRelease])];
  this.difficulty = 1;
  this.blockTime = 30000;
  this.reward = 297;
}
```

Рис 3.8 – Метод отримання останнього блока

- Метод getLastBlock (рисунок 3.9), який повертає останній блок у ланцюгу.

```
getLastBlock() {
  return this.chain[this.chain.length - 1];
}
```

Рис 3.9 – Метод отримання останнього блока

- Метод addBlock (рисунок 3.10), який приймає блок, встановлює в ньому хеш попереднього, вираховує хеш цього блока за допомогою методу mine, додає його до ланцюга та збільшує або зменшує показник складності, відповідно до того, чи перевищує час генерації блока очікуваний.

```
addBlock(block) {
  block.prevHash = this.getLastBlock().hash;
  block.hash = Block.getHash(block);
  block.mine(this.difficulty);
  this.chain.push(Object.freeze(block));

  this.difficulty +=
    Date.now() - parseInt(this.getLastBlock().timestamp) < this.blockTime
      ? 1
      : -1;
}
```

Рис 3.10 – Метод додавання блока до ланцюга

- Метод getBalance (рисунок 3.11), оскільки блокчейн не зберігає значення балансів користувачів, для вирахування коштів які належать користувачу за певною адресою, необхідно ітеруватися по всім транзакціям цього користувача, віднімати відправленні суми транзакцій і додавати отримані.

```
getBalance(address) {
  let balance = 0;

  this.chain.forEach((block) => {
    block.data.forEach((transaction) => {
      if (transaction.from === address) {
        balance -= transaction.amount;
        balance -= transaction.gas;
      }

      if (transaction.to === address) {
        balance += transaction.amount;
      }
    });
  });

  return balance;
}
```

Рис 3.11 – Метод отримання поточного балансу за певною адресою

- Метод mineTransactions (рисунок 3.12), який приймає адресу майнера, який отримає винагороду за майнінг блока, створює транзакцію, яка складається з суми gas з кожної транзакції обчисленого блока та фіксованої винагороди самої системи, підписує транзакцію, та додає її до наступного блока.

```
mineTransactions(rewardAddress) {
  let gas = 0;

  this.transactions.forEach((transaction) => {
    gas += transaction.gas;
  });

  const rewardTransaction = new Transaction(
    MINT_PUBLIC_ADDRESS,
    rewardAddress,
    this.reward + gas
  );
  rewardTransaction.sign(MINT_KEY_PAIR);

  const blockTransactions = [rewardTransaction, ...this.transactions];

  if (this.transactions.length !== 0)
    this.addBlock(new Block(Date.now().toString(), blockTransactions));

  this.transactions.splice(0, blockTransactions.length - 1);
}
```

Рис 3.12 – Метод створення транзакції з винагородою за майнінг блока

- Статичний метод isValid (рисунок 3.13) який приймає екземпляр блокчейну, ітерується по всім блокам, звіряє відповідність посилань на попередні блоки, перераховує хеш блоку і порівнює з існуючим. Далі перевіряється валідність транзакцій у кожному блоці.

```
static isValid(blockchain) {
  for (let i = 1; i < blockchain.chain.length; i++) {
    const currentBlock = blockchain.chain[i];
    const prevBlock = blockchain.chain[i - 1];

    if (
      currentBlock.hash !== Block.getHash(currentBlock) ||
      prevBlock.hash !== currentBlock.prevHash ||
      !Block.hasValidTransactions(currentBlock, blockchain)
    )
      return false;
  }
  return true;
}
```

Рис 3.13 – Метод перевірки достовірності блокчейну

3.2 Розробка P2P мережі

Для взаємодії учасників блокчейну необхідно створити мережу, де вузли могли б з'єднуватися та надсилати повідомлення один одному. Мережа повинна підтримувати такі функції: поширення транзакцій, створення нових блоків, відправка ланцюга і інформації про ланцюг.

Для цього використовується бібліотека ws (рисунок 3.14) за допомогою якої створюється сервер.

```
import WS from 'ws'
```

Рис 3.14 – Імпорт бібліотеки ws для створення серверу

Для підключення та прослуховування з'єднань необхідно підключити і використати функцію server.on("connection"). Функція підключення повинна мати можливість підключитися до адреси, потім обробник з'єднання цієї адреси підключиться до нашої адреси за допомогою наданого повідомлення. Повідомлення у даному випадку в форматі JSON, яке складається з двох полів: типу і даних. На рисунку 3.15 зображено приклад такого повідомлення.

```
"type": "TYPE_HANDSHAKE",
"data": ["Our address and our connected nodes' address", "address x", "address y"]
```

Рис 3.15 – Приклад повідомлення

На рисунку 3.16 зображена функція, яка генерує повідомлення для зручності.

```
function produceMessage(type, data) {
    return { type, data };
}
```

Рис 3.16 – Функція для генерації повідомлень

На рисунку 3.17 зображена базова імплементація прослуховувача з'єднань, функціонал якої далі буде розширений при інтеграції функцій блокчейну.

```
// Прослуховувач з'єднанння
server.on("connection", async(socket, req) => {
    // Прослуховувач повідомлень
    socket.on("message", message => {
        // Парсинг
        const _message = JSON.parse(message);

        switch(_message.type) {
            case "TYPE_HANDSHAKE":
                const nodes = _message.data;

                nodes.forEach(node => connect(node))

        }
    })
});
```

Рис 3.17 – Базова реалізація прослуховувача з’єднань.

На рисунку 3.18 зображенно реалізацію функції під’єднання вузлів.Для використання в майбутньому підключені сокети та адреси будуть зберігатися в одному масиві. Крім того, адреса вузла який щойно під ’єднався буде надіслана іншим вузлам мережі.

```
async function connect(address) {
  // Підключення буде можливо лише до адреси, до ще не під'єнаної адреси,
  // також заборонено підключатися до самого себе
  if (
    !connected.find((peerAddress) => peerAddress === address) &&
    address !== MY_ADDRESS
  ) {
    const socket = new WS(address);

    socket.on("open", () => {
      // spread оператор використовується для включення адрес під'єднаних вузлів та включення їх до тіла повідомлення
      socket.send(
        JSON.stringify(
          produceMessage("TYPE_HANDSHAKE", [MY_ADDRESS, ...connected])
        )
      );

      // Інші вузли будуть отримувати адресу та запрошення до приєднання до неї
      opened.forEach((node) =>
        node.socket.send(
          JSON.stringify(produceMessage("TYPE_HANDSHAKE", [address]))
        )
      );

      // Якщо "opened" вже містить адресу, то не додадється до масива
      if (
        !opened.find((peer) => peer.address === address) &&
        address !== MY_ADDRESS
      ) {
        opened.push({ socket, address });
      }

      // Якщо "connected" вже містить адресу,то не додадється до масива
      if (
        !connected.find((peerAddress) => peerAddress === address) &&
        address !== MY_ADDRESS
      ) {
        connected.push(address);
      }
      // Два верхніх оператора if потрібні через проблему асинхронних кодів.
      // Оскільки вони виконуються одночасно перший оператор if можна виконати, тому буде дублювання.
    });

    // Коли вузли від'єднуються, їх необхідно прибрати з масиву під'єднаних.
    socket.on("close", () => {
      opened.splice(connected.indexOf(address), 1);
      connected.splice(connected.indexOf(address), 1);
    });
  }
}
```

Рис 3.18 – Функція з’єднання вузлів.

Подальший функціонал має забезпечити трансляцію транзакції, пропонувати нові змайнені блоки. Нові вузли також повинні мати можливість отримувати ланцюги з інших вузлів.

Була створена функція (рисунок 3.19) для полегшення розсилки повідомлень між вузлами.

```
function sendMessage(message) {
    opened.forEach(node => {
        node.socket.send(JSON.stringify(message));
    })
}
```

Рис 3.19 – Функція розсилки повідомлень.

В обробник повідомлень додається кейс для створення транзакцій, який зображено на рисунку 3.20, який отримує дані транзакції з повідомлення і використовуючи статичний метод addTransactions додає її до пулу транзакцій.

```
switch (_message.type) {
  //...
  case "TYPE_CREATE_TRANSACTION":
    const transaction = _message.data;

    Chain.addTransaction(transaction);

    break;
}
```

Рис 3.20 – Обробник для створення транзакцій.

На рисунку 3.21 проілюстровано приклад надсилання транзакції

```
sendMessage(produceMessage("TYPE_CREATE_TRANSACTION", someTransaction));
//також необхідно додати транзакцію до свого пулу транзакцій
Chain.addTransaction(someTransaction);
```

Рис 3.20 – Спосіб надсилання транзакції.

Наступний крок це обробка повідомлення пропозиції нового блоку.Це значно більша і складніша частина ніж попередні. Для обробки повідомлення спочатку треба, перевірити, чи дійсний блок чи ні, потім ми додати його до ланцюжка та оновити складність. Блок дійсний, коли:

- Він має дійсні транзакції (транзакції знаходяться в пулі транзакцій, транзакції дійсні також є дійсними).
- Він має дійсний хеш.
- Він має дійсну складність.
- Він має дійсну позначку часу (вона не повинна бути більшою, ніж час, у який він був надісланий надіслали нам, і меншою за позначку часу попереднього блоку).

Проте, якщо один майнер видобув блок, він насправді не дізнається, чи його блок був першим, чи інший, надісланий йому, був першим. Це може відбутися через декілька факторів, одним із них є проблема зі зв'язком.

Можна ефективно виправити цю функціональність, використовуючи булеву змінну, яка називається checking і setTimeout. В основному, ідея полягає в тому, що якщо prevHash блоку дорівнює prevHash останнього блоку, то, ймовірно, це блок, який потребує перевірки на заміну. Під час перевірки значення змінної буде true, щоб вказувати на те, після завершення надсилається запит до вузлів для отримання їх останнього блоку. Почекавши певний період часу встановленний за допомогою setTimeout, значення змінної змінюється на false, скасовуючи процес, і блок, який з’явився швидше за все, саме той блок, який має бути доданий до ланцюга. Реалізацію зображено на рисунку 3.20

```
switch(_message.type) {
    case "TYPE_REPLACE_CHAIN":
        const [ newBlock, newDiff ] = _message.data;

        const ourTx = [...Chain.transactions.map(tx => JSON.stringify(tx))];
        const theirTx = [...newBlock.data.filter(tx => tx.from !== MINT_PUBLIC_ADDRESS).map(tx => JSON.stringify(tx))];
        const n = theirTx.length;

        if (newBlock.prevHash !== Chain.getLastBlock().prevHash) {
            for (let i = 0; i < n; i++) {
                const index = ourTx.indexOf(theirTx[0]);

                if (index === -1) break;

                ourTx.splice(index, 1);
                theirTx.splice(0, 1);
            }

            if (
                theirTx.length === 0 &&
                sha256(Chain.getLastBlock().hash + newBlock.timestamp + JSON.stringify(newBlock.data) + newBlock.nonce) === newBlock.hash &&
                newBlock.hash.startsWith("000" + Array(Math.round(Math.log(Chain.difficulty) / Math.log(16) + 1)).join("0")) &&
                Block.hasValidTransactions(newBlock, Chain) &&
                (parseInt(newBlock.timestamp) > parseInt(Chain.getLastBlock().timestamp) || Chain.getLastBlock().timestamp === "") &&
                parseInt(newBlock.timestamp) < Date.now() &&
                Chain.getLastBlock().hash === newBlock.prevHash &&
                (newDiff + 1 === Chain.difficulty || newDiff - 1 === Chain.difficulty)
            ) {
                Chain.chain.push(newBlock);
                Chain.difficulty = newDiff;
                Chain.transactions = [...ourTx.map(tx => JSON.parse(tx))];
            }
        } else if (!checked.includes(JSON.stringify([newBlock.prevHash, Chain.chain[Chain.chain.length-2].timestamp || ""]))) {
            checked.push(JSON.stringify([Chain.getLastBlock().prevHash, Chain.chain[Chain.chain.length-2].timestamp || ""]));

            const position = Chain.chain.length - 1;

            checking = true;

            sendMessage(produceMessage("TYPE_REQUEST_CHECK", MY_ADDRESS));

            setTimeout(() => {
                checking = false;

                let mostAppeared = check[0];

                check.forEach(group => {
                    if (check.filter(_group => _group === group).length > check.filter(_group => _group === mostAppeared).length) {
                        mostAppeared = group;
                    }
                })
```

Рис. 3.20 – Обробка повідомлення з новим блоком

Нові вузли, які щойно приєдналися до мережі, будуть отримувати актуальний ланцюг за допомогою відправлення запита до мережі. Оскільки розмір повідомлення обмежений, будуть послідовно надсилатись його блоки та інформація.

На рисунку 3.21 зображено реалізацію запиту актуального ланцюга з мережі.

```
case "TYPE_REQUEST_CHAIN":
    const socket = opened.filter(node => node.address === _message.data)[0].socket;

    for (let i = 1; i < Chain.chain.length; i++) {
        socket.send(JSON.stringify(produceMessage(
            "TYPE_SEND_CHAIN",
            {
                block: Chain.chain[i],
                finished: i === Chain.chain.length - 1
            }
        )));
    }

    break;
```

Рис 3.21 – Обробка запиту для отримання ланцюга.

На рисунку 3.22 зображено реалізацію відправлення актуального ланцюга з мережі.

```
case "TYPE_SEND_CHAIN":
    const { block, finished } = _message.data;

    if (!finished) {
        tempChain.chain.push(block);
    } else {
        tempChain.chain.push(block);
        if (Blockchain.isValid(tempChain)) {
            Chain.chain = tempChain.chain;
        }
        tempChain = new Blockchain();
    }

    break;
```

Рис 3.21 – Обробка запиту для відправлення ланцюга.

3.3 Тестування системи

3.3.1 Тестування в локальній мережі

Для тестування створено дві нові консолі з різними значеннями PORT, MY_ADDRESS та PRIVATE_KEY. Для першої встановлено пустий масив вузлів, а ключ співпадає з тим, що використовується для початкового випуску монет. Для іншого у масиві однорангових вузлів встановлено перший вузол, який перевірить, чи працює функціональність HANDSHAKE. Наступним етапом буде створена транзакція в першому вузлі, а її майнинг та відправка оновленного ланцюга в мережу — у другому вузлі. В консолі буде розруковано масив відкритих з'єднань і ланцюг блоків. На рисунку 3.22 зображено код, який виконається на першому вузлі. Створюється екземпляр нової транзакції, підписується та надсилається іншим вузлам.

```
const transaction = new Transaction(
  publicKey,
  adressTo,
  200,
  10,
  "This is first transaction"
);

transaction.sign(keyPair);

sendMessage(produceMessage("TYPE_CREATE_TRANSACTION", transaction));

Chain.addTransaction(transaction);
console.log(opened);
console.log(Chain);
```

Рис 3.22 – Створення та відправка транзакції на першому вузлі.

На рисунку 3.23 відбувається перевірка необроблених транзакцій у мережі, далі відбувається майнінг та надсилається оновлення іншим вузлам.

```
if (Chain.transactions.length !== 0) {
    Chain.mineTransactions(publicKey);

    sendMessage(produceMessage("TYPE_REPLACE_CHAIN", [
        Chain.getLastBlock(),
        Chain.difficulty
    ]));
}

console.log(opened);
console.log(Chain);
```

Рис 3.23 – Обробка транзакції.

На рисунку 2.24 зображено результати виконання описаних вище операцій, вузли вдало з’єднались один з одним, блок видобуто, ланцюг синхронізовано.

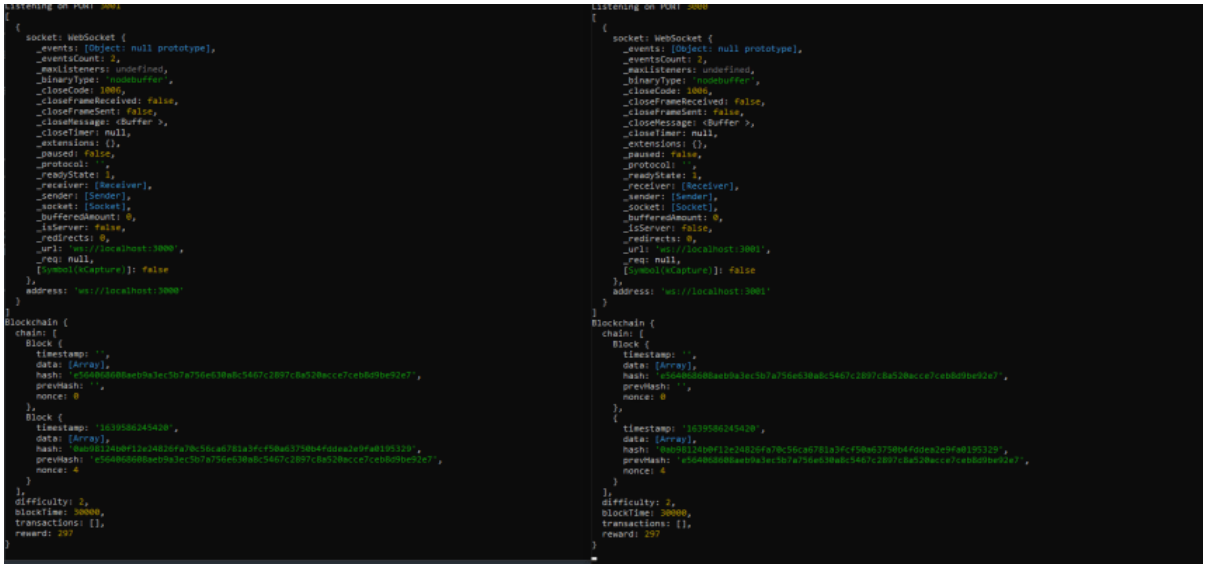


Рис 3.24 – Результати локального тестування.

3.3.2 Тестування в загальній мережі

Для тестування необхідно розмістити вузли у публічному доступі, наприклад використовуючи переадресацію портів. Виконавши переадресацію портів і підключившись до загальнодоступних IP-адрес вузлів проведено аналогічне тестування, як і в локальній мережі. На рисунку 3.25 зображено результати відображені на першому пристрої, на рисунку 3.26 – на другому.

Рис 3.25 – Результати тестування в загальній мережі (перший вузол).

Рис 3.26 – Результати тестування в загальній мережі (другий вузол).

Вузли успішно з’єднані, код виконаний, обидва вузли містять одні і ті ж самі блоки і відомості про екземпляр блокчейну.

ВИСНОВОК ДО РОЗДІЛУ 3

В рамках третього розділу описано процес розробки та тестування системи. Зокрема, виділено ключові сутності блокчейн системи та сформулювано вимоги до їх функціоналу. Кожна з них отримала реалізацію у вигляді класів: Transaction, Block, Blockchain. Ці класи мають відповідні властивості і методи для необхідної коректної роботи системи. Наприклад, хешування даних, реалізація алгоритму Proof-of-Work та корегування параметру складності для підтримки стабільного часу генерації нового блока. Багато уваги приділено безпеці та іммутабельності даних: кожен клас має як мінімум один метод валідації даних, та можливі повторні перевірки у подальшому функціоналі, який використовує екземпляри цих класів.

Окрім побудови програмних компонентів безпосередньо блокчейну, також було розроблено P2P мережу для використання функціональних можливостей вище перелічених сутностей різними користувачами. Обмін даними виконується за допомогою протоколу WebSocket і відправлення користувачами повідомлень у встановленному форматі. Було написано обробник повідомлень, звертаючись до якого, користувачі можуть, наприклад, отримувати екземпляр блокчейну з мережі, надсилати транзакції, майнити і додавати нові блоки, отримувати інформацію про свій баланс і поточний стан системи.

Наступна частина розділу присвячена тестуванню побудованної системи. Через відсутність можливості розгорнути повномасштабну мережу з великою кількістю вузлів, тестування проходило за допомогою кількох вузлів спочатку в локальній мережі, а потім, використовуючи перенаправлення портів, в загальній. На одному з вузлів було створено початкову транзакцію і відправлено її в мережу для перевірки майбутнього включення до блока.

На іншому отримано і перевірено ці дані, потім додано їх до нового блока, вираховувано його хеш і відправлено у мережу для додавання учасниками його до ланцюга. В обох випадках був досягнутий вдалий результат, функціонал системи спрацював згідно з очікувань, а дані на вузлах мережі синхронізувалися.

ВИСНОВКИ

Метою виконання дипломної роботи була реалізація способу побудови блокчейн системи.

У першому розділі було проведено аналіз предметної області блокчейн систем. Описані та порівняні види систем за централізацією. Розглянуто структуру і принципи самого блокчейну. Було проведено порівняння двох існуючих реалізацій подібних систем, визначенно їх недоліки та переваги.

У другому розділі були оглянуті технології для реалізації поставленого завдання. Було обрано архітектурний підхід, порівняно мови програмування. На основі порівняння обрано мову і платформу для розробки. Далі було описано і проілюстровано алгоритм хешування даних та розглянуто необхідні бібліотеки.

В рамках третього розділу описано процес розробки та тестування системи. Використовуючи інструменти, які були описані у другому розділі покроково реалізовано програмні компоненти у вигляді класів. Процес розробки був проілюстрований скріншотами фрагментів коду. У наступному етапі розділу розроблено однорангову мережу для використання блокчейн системи користувачами. Далі продемонстровано тестування системи і наведено скріншоти його результатів.

Підсумовуючи, реалізована блокчейн система задовольняє визначену на початку мету дипломного проекту. В той же час, є аспекти системи які можуть бути покращені та запас для розширення функціоналу. Наприклад, створення інтерфейсу для використання в інших програмах та перехід до більш екологічного алгоритму консенсусу Proof-of-Stake.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. History of Blockchain Technology: A Detailed Guide. 101 Blockchains. URL: https://101blockchains.com/history-of-blockchain-timeline/ (дата звернення: 17.06.2022).
2. Lewis A. The basics of bitcoins and blockchains: An introduction to cryptocurrencies and the technology that powers them. Coral Gables, USA : Mango Media Inc, 2018. 408 с.
3. Crypto Market Cap and DeFi Market Cap Charts TradingView. *TradingView*. URL: https://www.tradingview.com/markets/cryptocurrencies/global-charts/ (дата звернення: 17.06.2022).
4. Centralized vs Decentralized vs Distributed Systems · Berty Technologies. *Berty Technologies*. URL: https://berty.tech/blog/decentralized-distributed-centralized (дата звернення: 17.06.2022).
5. Cherednichenko S. Designing a Blockchain Architecture: Types, Use Cases, and Challenges. *Medium*. URL: https://medium.com/mobindustry/designing-a-blockchain-architecture-types-use-cases-and-challenges-9894fb7b58e (дата звернення: 17.06.2022).
6. Ethereum. *ethereum.org*. URL: https://ethereum.org/en/ (дата звернення: 17.06.2022).
7. Waves Tech. *Waves Tech*. URL: https://waves.tech (дата звернення: 17.06.2022).
8. Difference between Client-Server and Peer-to-Peer Network - GeeksforGeeks. *GeeksforGeeks*.

URL: https://www.geeksforgeeks.org/difference-between-client-server-and-peer-to-peer-network/ (дата звернення: 17.06.2022).

9. Haverbeke M. Eloquent JavaScript: A Modern Introduction to Programming. 3-тє вид. San Francisco, California, United States of America : No Starch Press, 2018. 472 с.
10. Node.js. *Node.js*. URL: https://nodejs.org/en/ (дата звернення: 17.06.2022).
11. ShapeShift. Top Programming Languages for Blockchain Developers. *Medium*. URL: https://medium.com/shapeshift-stories/top-programming-languages-for-blockchain-developers-9f4fddd803e9 (дата звернення: 17.06.2022).
12. CoinMarketCap. SHA-256. *CoinMarketCap*. URL: https://coinmarketcap.com/alexandria/glossary/sha-256 (дата звернення: 17.06.2022).
13. How to use the crypto module. *Node.js*. URL: https://nodejs.org/en/knowledge/cryptography/how-to-use-crypto-module/ (дата звернення: 17.06.2022).
14. Lombardi A. WebSocket. O'Reilly Media, Incorporated, 2014. 144 с.
15. ws. *npm*. URL: https://www.npmjs.com/package/ws (дата звернення: 17.06.2022).
16. elliptic. npm. URL: https://www.npmjs.com/package/elliptic (дата звернення: 17.06.2022).
17. Secp256k1. *Bitcoin Wiki*. URL: https://en.bitcoin.it/wiki/Secp256k1 (дата звернення: 17.06.2022).

ДОДАТОК 1

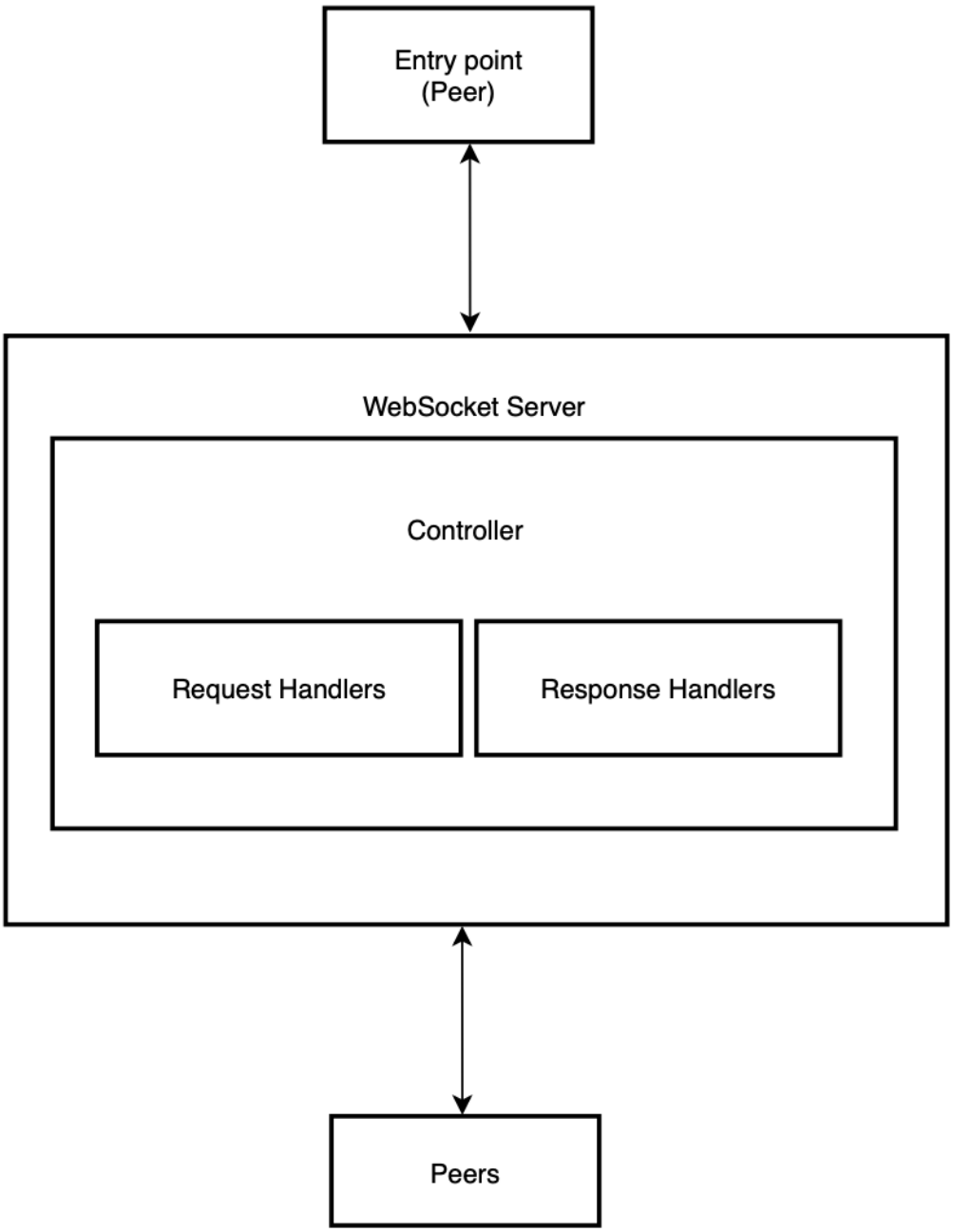
Еволюційні алгоритми глобальної пошукової оптимізації

Структурна схема системи

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2022 р



					ІАЛЦ.467200.004 Д1		
		№ докум.	Підпис	Дата			
Розробив		Черевач А. М.			**Спосіб побудови блокчейн системи** Структурна схема системи	Літ.	Аркуш 2 / Аркушів 2
Перевірив		Волокита І. М.					
Реценз.						НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІП-83	
Н. Контр.		Сімоненко В. П.					
Затвердив							

ДОДАТОК 2

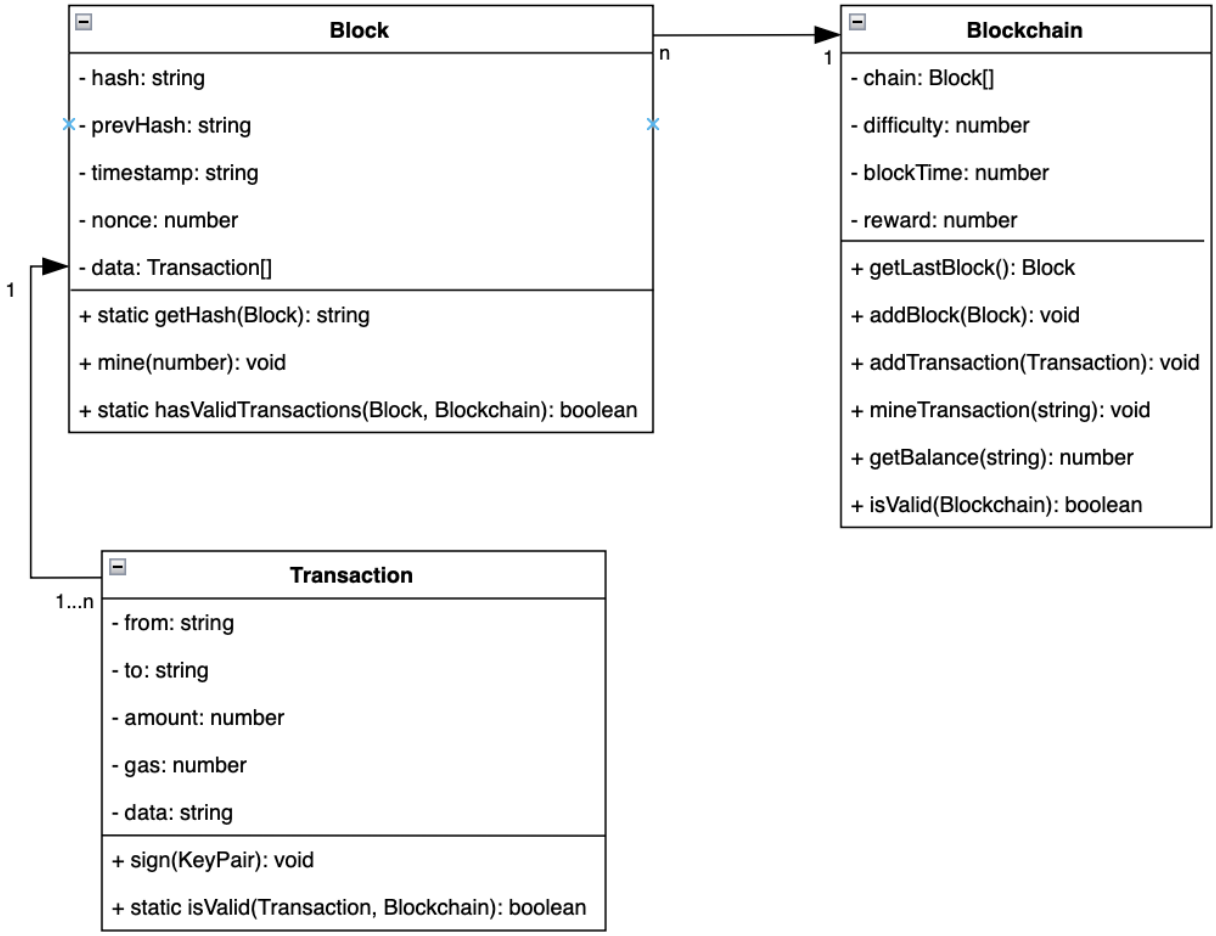
Спосіб побудови блокчейн системи

Функціональна схема (діаграма класів)

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2022 р



					ІАЛЦ.467200.005 Д2	
		№ докум.	Підпис	Дата		
Розробив		Черевач А. М.			**Спосіб побудови блокчейн системи** Функціональна схема (діаграма класів)	Літ. / Аркуш 2 / Аркушів 2
Перевірив		Волокита І. М.				
Реценз.						НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІП-83
Н. Контр.		Сімоненко В. П.				
Затвердив						

ДОДАТОК 3

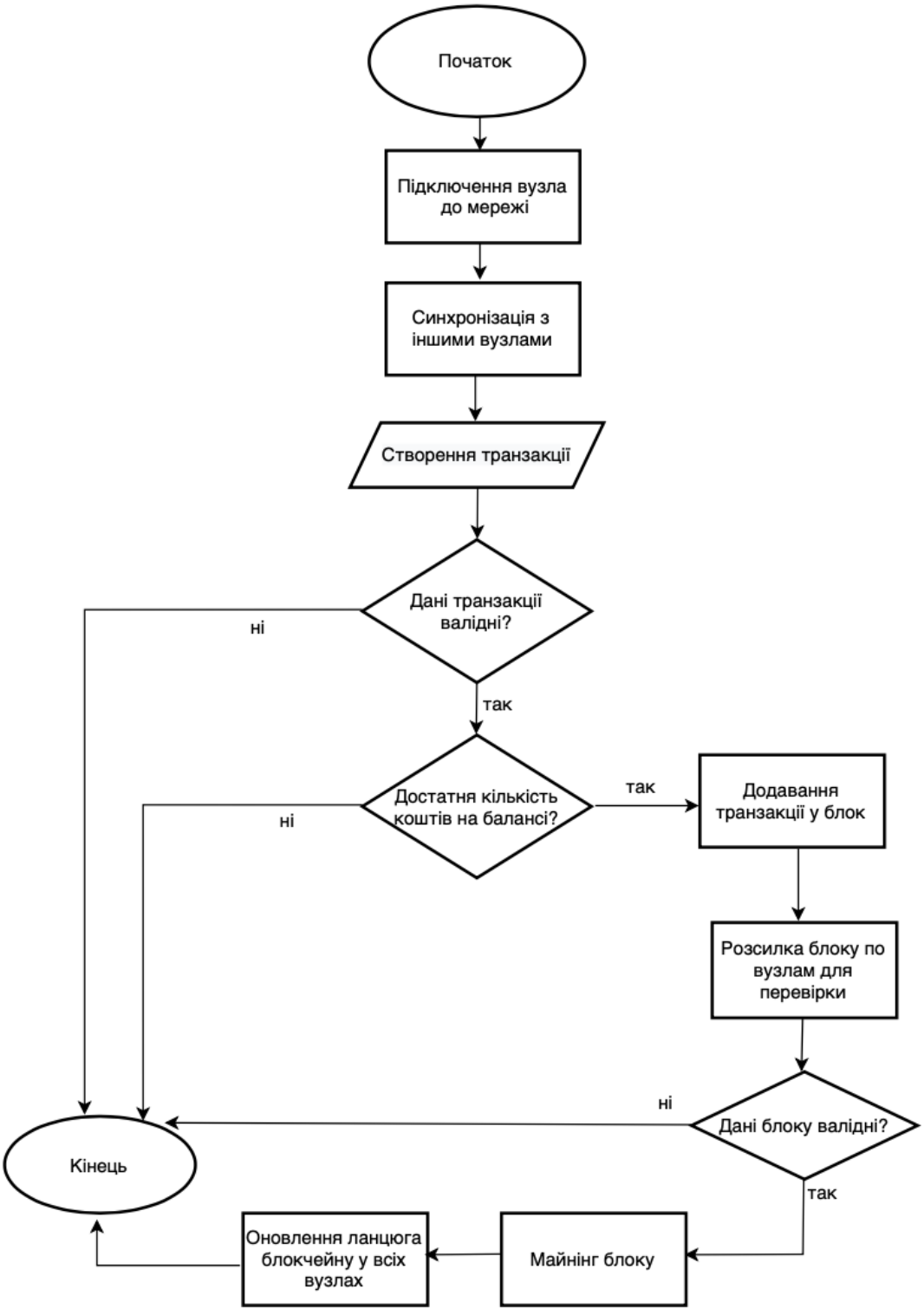
Спосіб побудови блокчейн системи

Алгоритм дій програмного забезпечення

ІАЛЦ.467200.006 Д3

Аркушів 1

Київ 2022 р



					ІАЛЦ.467200.006 Д3		
		№ докум.	Підпис	Дата			
Розробив		Черевач А. М.			**Спосіб побудови блокчейн системи** Алгоритм дій програмного забезпечення	Літ.	Аркуш 2 / Аркушів 2
Перевірив		Волокита І. М.					
Реценз.						НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІП-83	
Н. Контр.		Сімоненко В. П.					
Затвердив							

ДОДАТОК 4

Спосіб побудови блокчейн системи

Текст програмного коду

ІАЛЦ.467200.007 Д4

Аркушів 6

Київ 2022 р

transaction.js

```
import { sha256 } from "./utils";
import { ec as EC } from "elliptic";

const ec = new EC("secp256k1");
const MINT_PRIVATE_ADDRESS =
  "0700a1ad28a20e5b2a517c00242d3e25a88d84bf54dce9e1733e6096e6d6495e";
const MINT_KEY_PAIR = ec.keyFromPrivate(MINT_PRIVATE_ADDRESS, "hex");
const MINT_PUBLIC_ADDRESS = MINT_KEY_PAIR.getPublic("hex");

export class Transaction {
  constructor(from, to, amount, gas = 0, data = "") {
    this.from = from;
    this.to = to;
    this.amount = amount;
    this.gas = gas;
    this.data = data;
  }

  sign(keyPair) {
    if (keyPair.getPublic("hex") === this.from) {
      this.signature = keyPair
        .sign(
          sha256(this.from + this.to + this.amount + this.gas + this.data),
          "base64"
        )
        .toDER("hex");
    }
  }

  static isValid(tx, chain) {
    return (
      tx.from &&
      tx.to &&
      tx.amount &&
      (chain.getBalance(tx.from) >= tx.amount + tx.gas ||
        tx.from === MINT_PUBLIC_ADDRESS) &&
      ec
        .keyFromPublic(tx.from, "hex")
        .verify(
          sha256(tx.from + tx.to + tx.amount + tx.gas + tx.data),
          tx.signature
        )
    );
  }
}

const transaction = new Transaction ("")
```

block.js

```
import { Transaction } from "./transaction";
import { sha256 } from "./sha256";

const MINT_PRIVATE_ADDRESS =
  "0700a1ad28a20e5b2a517c00242d3e25a88d84bf54dce9e1733e6096e6d6495e";
const MINT_KEY_PAIR = ec.keyFromPrivate(MINT_PRIVATE_ADDRESS, "hex");
const MINT_PUBLIC_ADDRESS = MINT_KEY_PAIR.getPublic("hex");

export class Block {
  constructor(timestamp = Date.now().toString(), data = []) {
    this.timestamp = timestamp;
    this.data = data;
    this.prevHash = "";
    this.hash = Block.getHash(this);
    this.nonce = 0;
  }
```

```
  static getHash(block) {
    return sha256(
      block.prevHash +
        block.timestamp +
        JSON.stringify(block.data) +
        block.nonce
    );
  }

  mine(difficulty) {
    const prefix = "0".repeat(difficulty)
    while (!this.hash.startsWith(prefix)) {
      this.nonce++;
      this.hash = Block.getHash(this);
    }
  }

  static hasValidTransactions(block, chain) {
    let gas = 0,
      reward = 0;

    block.data.forEach((transaction) => {
      if (transaction.from !== MINT_PUBLIC_ADDRESS) {
        gas += transaction.gas;
      } else {
        reward = transaction.amount;
      }
    });

    return (
      reward - gas === chain.reward &&
      block.data.every((transaction) =>
        Transaction.isValid(transaction, chain)
      ) &&
      block.data.filter(
        (transaction) => transaction.from === MINT_PUBLIC_ADDRESS
      ).length === 1
    );
  }
```

blockchain.js

```
import {ec as EC} from "elliptic"

const  ec = new EC("secp256k1");
const MINT_PRIVATE_ADDRESS =
  "0700a1ad28a20e5b2a517c00242d3e25a88d84bf54dce9e1733e6096e6d6495e";
const MINT_KEY_PAIR = ec.keyFromPrivate(MINT_PRIVATE_ADDRESS, "hex");
const MINT_PUBLIC_ADDRESS = MINT_KEY_PAIR.getPublic("hex");
const INITIAL_ADRESS =
"04719af634ece3e9bf00bfd7c58163b2caf2b8acd1a437a3e99a093c8dd7b1485c20d8a4c9f6621557f1d583e0fcff99f3234dd1bb36559
6d1d67909c270c16d64"

export class Blockchain {
  constructor() {
    const initalCoinRelease = new Transaction(
      MINT_PUBLIC_ADDRESS,
      INITIAL_ADRESS,
      100000000
    );
    this.transactions = [];
    this.chain = [new Block(Date.now().toString(), [initalCoinRelease])];
    this.difficulty = 1;
    this.blockTime = 30000;
    this.reward = 297;
  }

  getLastBlock() {
    return this.chain[this.chain.length - 1];
  }
```

```
addBlock(block) {
  block.prevHash = this.getLastBlock().hash;
  block.hash = Block.getHash(block);
  block.mine(this.difficulty);
  this.chain.push(Object.freeze(block));

  this.difficulty +=
    Date.now() - parseInt(this.getLastBlock().timestamp) < this.blockTime
      ? 1
      : -1;
}

addTransaction(transaction) {
  if (Transaction.isValid(transaction, this)) {
    this.transactions.push(transaction);
  }
}

mineTransactions(rewardAddress) {
  let gas = 0;

  this.transactions.forEach((transaction) => {
    gas += transaction.gas;
  });

  const rewardTransaction = new Transaction(
    MINT_PUBLIC_ADDRESS,
    rewardAddress,
    this.reward + gas
  );
  rewardTransaction.sign(MINT_KEY_PAIR);

  const blockTransactions = [rewardTransaction, ...this.transactions];

  if (this.transactions.length !== 0)
    this.addBlock(new Block(Date.now().toString(), blockTransactions));

  this.transactions.splice(0, blockTransactions.length - 1);
}

getBalance(address) {
  let balance = 0;

  this.chain.forEach((block) => {
    block.data.forEach((transaction) => {
      if (transaction.from === address) {
        balance -= transaction.amount;
        balance -= transaction.gas;
      }

      if (transaction.to === address) {
        balance += transaction.amount;
      }
    });
  });

  return balance;
}

static isValid(blockchain) {
  for (let i = 1; i < blockchain.chain.length; i++) {
    const currentBlock = blockchain.chain[i];
    const prevBlock = blockchain.chain[i - 1];

    if (
      currentBlock.hash !== Block.getHash(currentBlock) ||
      prevBlock.hash !== currentBlock.prevHash ||
      !Block.hasValidTransactions(currentBlock, blockchain)
    )
```

```
      return false;
    }
    return true;
  }
}


export const Chain = new Blockchain();
```

coin.js

```
import {sha256} from './utils';
const EC = require("elliptic").ec, ec = new EC("secp256k1");
const { Block, Blockchain, Transaction, Chain } = require("./blockchain");

const MINT_PRIVATE_ADDRESS = "0700a1ad28a20e5b2a517c00242d3e25a88d84bf54dce9e1733e6096e6d6495e";
const MINT_KEY_PAIR = ec.keyFromPrivate(MINT_PRIVATE_ADDRESS, "hex");
const MINT_PUBLIC_ADDRESS = MINT_KEY_PAIR.getPublic("hex");

const privateKey = "39a4a81e8e631a0c51716134328ed944501589b447f1543d9279bacc7f3e3de7";
const keyPair = ec.keyFromPrivate(privateKey, "hex");
const publicKey = keyPair.getPublic("hex");

let opened = [], connected = [];
let check = [];
let checked = [];
let checking = false;
let tempChain = new Blockchain();

import WS from 'ws'

const PORT = 3001;
const PEERS = ["ws://localhost:3000"];
const MY_ADDRESS = "ws://localhost:3001";
const server = new WS.Server({ port: PORT });


console.log("Listening on PORT", PORT);
server.on("connection", async (socket, req) => {
   socket.on("message", message => {
      const _message = JSON.parse(message);

      console.log(_message);

      switch(_message.type) {
         case "TYPE_REPLACE_CHAIN":
            const [ newBlock, newDiff ] = _message.data;

            const ourTx = [...Chain.transactions.map(tx => JSON.stringify(tx))];
            const theirTx = [...newBlock.data.filter(tx => tx.from !== MINT_PUBLIC_ADDRESS).map(tx => JSON.stringify(tx))];
            const n = theirTx.length;

            if (newBlock.prevHash !== Chain.getLastBlock().prevHash) {
               for (let i = 0; i < n; i++) {
                  const index = ourTx.indexOf(theirTx[0]);

                  if (index === -1) break;

                  ourTx.splice(index, 1);
                  theirTx.splice(0, 1);
               }

               if (
                  theirTx.length === 0 &&
                  sha256(Chain.getLastBlock().hash + newBlock.timestamp + JSON.stringify(newBlock.data) + newBlock.nonce) ===
newBlock.hash &&
                  newBlock.hash.startsWith("000" + Array(Math.round(Math.log(Chain.difficulty) / Math.log(16) + 1)).join("0")) &&
                  Block.hasValidTransactions(newBlock, Chain) &&
                  (parseInt(newBlock.timestamp) > parseInt(Chain.getLastBlock().timestamp) || Chain.getLastBlock().timestamp ===
"") &&
```

```
            parseInt(newBlock.timestamp) < Date.now() &&
            Chain.getLastBlock().hash === newBlock.prevHash &&
            (newDiff + 1 === Chain.difficulty || newDiff - 1 === Chain.difficulty)
        ) {
            Chain.chain.push(newBlock);
            Chain.difficulty = newDiff;
            Chain.transactions = [...ourTx.map(tx => JSON.parse(tx))];
        }
    } else if (!checked.includes(JSON.stringify([newBlock.prevHash, Chain.chain[Chain.chain.length-2].timestamp || ""]))) {
        checked.push(JSON.stringify([Chain.getLastBlock().prevHash, Chain.chain[Chain.chain.length-2].timestamp || ""]));

        const position = Chain.chain.length - 1;

        checking = true;

        sendMessage(produceMessage("TYPE_REQUEST_CHECK", MY_ADDRESS));

        setTimeout(() => {
            checking = false;

            let mostAppeared = check[0];

            check.forEach(group => {
                if (check.filter(_group => _group === group).length > check.filter(_group => _group === mostAppeared).length) {
                    mostAppeared = group;
                }
            })

            const group = JSON.parse(mostAppeared)

            Chain.chain[position] = group[0];
            Chain.transactions = [...group[1]];
            Chain.difficulty = group[2];

            check.splice(0, check.length);
        }, 5000);
    }

    break;

case "TYPE_REQUEST_CHECK":
    opened.filter(node => node.address === _message.data)[0].socket.send(
        JSON.stringify(produceMessage(
            "TYPE_SEND_CHECK",
            JSON.stringify([Chain.getLastBlock(), Chain.transactions, Chain.difficulty])
        ))
    );

    break;

case "TYPE_SEND_CHECK":
    if (checking) check.push(_message.data);

    break;

case "TYPE_CREATE_TRANSACTION":
    const transaction = _message.data;

    Chain.addTransaction(transaction);

    break;

case "TYPE_SEND_CHAIN":
    const { block, finished } = _message.data;

    if (!finished) {
        tempChain.chain.push(block);
    } else {
        tempChain.chain.push(block);
        if (Blockchain.isValid(tempChain)) {
```

```
                Chain.chain = tempChain.chain;
            }
            tempChain = new Blockchain();
        }

        break;

    case "TYPE_REQUEST_CHAIN":
        const socket = opened.filter(node => node.address === _message.data)[0].socket;

        for (let i = 1; i < Chain.chain.length; i++) {
            socket.send(JSON.stringify(produceMessage(
                "TYPE_SEND_CHAIN",
                {
                    block: Chain.chain[i],
                    finished: i === Chain.chain.length - 1
                }
            )));
        }

        break;

    case "TYPE_REQUEST_INFO":
        opened.filter(node => node.address === _message.data)[0].socket.send(JSON.stringify(produceMessage(
            "TYPE_SEND_INFO",
            [Chain.difficulty, Chain.transactions]
        )));

        break;

    case "TYPE_SEND_INFO":
        [ Chain.difficulty, Chain.transactions ] = _message.data;

        break;

    case "TYPE_HANDSHAKE":
        const nodes = _message.data;

        nodes.forEach(node => connect(node))
    }
  });
})

async function connect(address) {
  if (!connected.find(peerAddress => peerAddress === address) && address !== MY_ADDRESS) {
    const socket = new WS(address);

    socket.on("open", () => {
      socket.send(JSON.stringify(produceMessage("TYPE_HANDSHAKE", [MY_ADDRESS, ...connected])));

      opened.forEach(node => node.socket.send(JSON.stringify(produceMessage("TYPE_HANDSHAKE", [address]))));

      if (!opened.find(peer => peer.address === address) && address !== MY_ADDRESS) {
        opened.push({ socket, address });
      }

      if (!connected.find(peerAddress => peerAddress === address) && address !== MY_ADDRESS) {
        connected.push(address);
      }
    });

    socket.on("close", () => {
      opened.splice(connected.indexOf(address), 1);
      connected.splice(connected.indexOf(address), 1);
    });
  }
}

function produceMessage(type, data) {
  return { type, data };
```

```
}

function sendMessage(message) {
    opened.forEach(node => {
        node.socket.send(JSON.stringify(message));
    })
}

process.on("uncaughtException", err => console.log(err));

PEERS.forEach(peer => connect(peer));

setTimeout(() => {
    if (Chain.transactions.length !== 0) {
        Chain.mineTransactions(publicKey);

        sendMessage(produceMessage("TYPE_REPLACE_CHAIN", [
            Chain.getLastBlock(),
            Chain.difficulty
        ]))
    }
}, 6500);

setTimeout(() => {
    console.log(opened);
    console.log(Chain);
}, 10000);
```