Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти перший (бакалаврський)

Модульний блокчейн з використанням фреймворку Substrate

(тема)

Виконав:
здобувач 4 року навчання,
групи КІУКІ-21-3
Ілля БУКАНОВ
(власне ім’я, прізвище)

Спеціальність
123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Комп’ютерна інженерія
(повна назва освітньої програми)

Керівник: ас. Віталій СІТНІКОВ
(посада, власне ім’я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім’я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти перший (бакалаврський)

Спеціальність 123 «Комп’ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп’ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Буканову Іллі В’ячеславовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Модульний блокчейн з використанням фреймворку Substrate

затверджена наказом по університету від “ 26 ” травня 2025 р. № 424 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 17 червня 2025 р.

3. Вхідні дані до роботи 1) фреймворк Substrate; 2) архітектура FRAME; 3) WebAssembly 4) Polkadot.js Apps.

4. Перелік питань, що потрібно опрацювати у роботі

1) принципи побудови блокчейн-систем та класифікація блокчейнів;
2) архітектура та особливості фреймворку Substrate;
3) створення та інтеграція власної палети TokenVoting у середовище runtime;
4) налаштування середовища та компіляція Substrate-проєкту;
5) тестування функціональності палети через Polkadot.js та оцінка результатів.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій

Слайд-презентація – 15

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Постановка мети дослідження	26.05.25–27.05.25	
2	Дослідження архітектури Substrate	28.05.25–30.05.25	
3	Проєктування структури власної палети	31.05.25–02.06.25	
	TokenVoting		
4	Розробка функціоналу голосування та обліку	03.06.25–06.06.25	
	токенів		
5	Запуск тестового блокчейну	07.06.25–08.06.25	
6	Внесення коригувань до логіки палети	09.06.25–10.06.25	
7	Оформлення текстової частини	11.06.25–12.06.25	
8	Подання кваліфікаційної роботи керівникові	13.06.25–14.06.25	
9	Подання кваліфікаційної роботи на	16.06.25–17.06.25	
	рецензування		

Дата видачі завдання " 26 " травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ ас. Віталій СІТНІКОВ
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 61 с., 18 рис., 1 дод., 18 джерел.

SUBSTRATE, FRAME, ПАЛЕТА, РАНТАЙМ, БЛОКЧЕЙН, ТОКЕНИ, ГОЛОСУВАННЯ, ДЕЦЕНТРАЛІЗАЦІЯ, POLKADOT.JS, RUST, WEBASSEMBLY.

Метою кваліфікаційної роботи є розробка модульного блокчейну з реалізацією системи голосування за створення токенів на основі фреймворку substrate.

У ході виконання кваліфікаційної роботи буде розглянуто архітектуру frame та принципи побудови власної palette у середовищі substrate, проаналізовано механізм inkріпцій і передачі цифрових активів, здійснено підключення до інтерфейсу polkadot.js apps, а також реалізовано функції голосування, обліку та трансферу токенів. Робота спрямована на демонстрацію практичного застосування блокчейн-технологій у створенні модульних децентралізованих систем.

ABSTRACT

Bachelor’s thesis: 61 pages, 18 figures, 1 appendices, 18 sources.

SUBSTRATE, FRAME, PALETTE, RUNTIME, BLOCKCHAIN, TOKENS, VOTING, DECENTRALIZATION, POLKADOT.JS, RUST, WEBASSEMBLY.

The major goal of this thesis is to explore the process of developing a modular blockchain with the implementation of a token voting system based on the Substrate framework.

In order to achieve this, the thesis will examine the architecture of FRAME and the principles of building a custom pallet within the Substrate environment, analyze the mechanism of inscriptions and digital asset transfers, establish interaction with the Polkadot.js Apps interface, and implement functionalities for token voting, accounting, and transfers. The project aims to demonstrate the practical use of blockchain technologies in building adaptive decentralized systems.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ 8
ВСТУП 9
1 ОСНОВИ БЛОКЧЕЙН-ТЕХНОЛОГІЙ 10
1.1 Поняття блокчейн-технології 10
1.2 Принцип роботи блокчейну 11
1.2.1 Структура блоку 11
1.2.2 Принцип роботи транзакцій 12
1.2.3 Хеш-функції 12
1.2.4 Механізм додавання блоків 13
1.2.5 Синхронізація між вузлами 13
1.3 Алгоритми консенсусу 14
1.4 Класифікація блокчейнів 16
2 ФРЕЙМВОРК SUBSTRATE 20
2.1 Характеристика фреймворку Substrate 20
2.1.1 Загальна характеристика 20
2.1.2 Зв’язок із Polkadot 21
2.1.3 Призначення та переваги 21
2.2 Структурна архітектура Substrate 23
2.2.1 Архітектурна модель та принципи побудови 23
2.2.2 Рівнева організація компонентів 25
2.3 Ключові компоненти Substrate ноди 26
2.3.1 Client (ядро вузла) 27
2.3.2 Runtime Executor 27
2.3.3 Transaction Pool 27
2.3.4. Consensus Engine 28
2.3.5 Networking Layer 28
2.3.6 Storage (сховище даних) 29

2.4 Інтеграція концепції Zero Trust у архітектуру блокчейн-рішень 29
3 ПІДГОТОВКА ДО РЕАЛІЗАЦІЇ 31
3.1 Обґрунтування вибору фреймворку Substrate 31
3.2 Архітектура реалізації модульного блокчейну 33
3.2.1 Вузол (Node) 33
3.2.2 Виконуване середовище (Runtime) 34
3.2.3 Використання вбудованих pallet'ів 34
3.2.4 Масштабованість та подальше розширення 34
3.3 Опис функціональних вимог 35
4 РЕАЛІЗАЦІЯ МОДУЛЬНОГО БЛОКЧЕЙНУ 37
4.1 Ініціалізація проекту 37
4.1.1 Налаштування середовища розробки 37
4.1.2 Складання проєкту та встановлення залежностей 38
4.1.3 Тестовий запуск блокчейн-вузла 40
4.2 Створення кастомної палети TokenVoting 41
4.3 Інтеграція палети TokenVoting у runtime 43
4.4 Запуск та тестування блокчейну 46
ВИСНОВКИ 51
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 52
ДОДАТОК А Графічний матеріал кваліфікаційної роботи 54

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

JSON-RPC – протокол віддаленого виклику процедур на базі JavaScript Object Notation

MFA – багатофакторна автентифікація (англ. Multi-Factor Authentication)

P2P – однорангова мережа (англ. Peer-to-Peer)

PoS – доказ частки володіння (англ. Proof of Stake)

PoW – доказ виконаної роботи (англ. Proof of Work)

RPC – віддалений виклик процедур (англ. Remote Procedure Call)

UBA – аналіз поведінки користувачів (англ. User Behavior Analytics)

Wasm – веб-збірка (англ. WebAssembly)

WSL – підсистема Windows для Linux (англ. Windows Subsystem for Linux)

ВСТУП

Блокчейн-технології набувають усе більшого поширення завдяки своїй здатності забезпечувати прозорість, безпеку та децентралізоване управління даними. Вони знаходять застосування у фінансовому секторі, логістиці, охороні здоров’я, державному управлінні та багатьох інших сферах. Розвиток цих технологій супроводжується зростанням потреби у створенні гнучких та масштабованих блокчейн-рішень, здатних адаптуватися до специфіки конкретного застосування.

Зі зростанням складності децентралізованих систем постає необхідність у зручних інструментах, які дозволяють не лише пришвидшити розробку, а й зберегти контроль над архітектурою, логікою транзакцій та механізмами консенсусу. Одним із найперспективніших рішень у цьому напрямі є фреймворк Substrate, створений компанією Parity Technologies. Завдяки модульній архітектурі, підтримці WebAssembly і мові програмування Rust, Substrate дозволяє створювати блокчейн-системи, які не обмежуються стандартною логікою смарт-контрактів, а забезпечують повноцінну кастомізацію ядра блокчейну.

Використання Substrate відкриває нові можливості для проєктування децентралізованих рішень, в яких розробник може самостійно визначати структуру даних, правила валідації транзакцій, механізми нагород і покарань, а також обирати або реалізовувати власні алгоритми консенсусу. Це робить фреймворк особливо привабливим як для експериментальних досліджень у галузі децентралізованих технологій, так і для створення повноцінних комерційних продуктів.

1 ОСНОВИ БЛОКЧЕЙН-ТЕХНОЛОГІЙ

1.1 Поняття блокчейн-технології

Поняття блокчейн стало синонімом довіри, децентралізації та криптографічної безпеки. Блокчейн визначається як розподілений децентралізований реєстр, який забезпечує фіксацію транзакцій у хронологічно впорядкованій, криптографічно зв’язаній послідовності блоків. Такий підхід унеможливлює ретроспективну зміну даних без порушення цілісності всієї структури, що надає системі високий рівень незмінності.

Концепція незмінного послідовного зберігання інформації була вперше запропонована Стюартом Хабером і Скоттом Сторнетто у 1991 році у вигляді криптографічного timestamp-сервера[2]. Однак лише у 2008 році ідея блокчейну набула практичного втілення у технічному документі «Bitcoin: A Peer-to-Peer Electronic Cash System», автором якого вважається анонімний дослідник або група осіб під псевдонімом Сатоші Накамото [1].

Саме у цьому документі вперше була описана система, яка поєднує розподілену мережу, криптографію відкритого ключа, алгоритм консенсусу Proof of Work та структуру послідовно зв’язаних блоків для створення безпечної децентралізованої платіжної системи без участі довірених посередників. У січні 2009 року ця концепція була реалізована у вигляді блокчейн-мережі Bitcoin - першої у світі криптовалюти.

Блокчейн функціонує як ланцюг блоків, у якому кожен блок містить:

- Набір підтверджених транзакцій;
- Мітку часу (timestamp);
- Хеш попереднього блоку;
- Nonce (у разі використання PoW);
- Службову інформацію, необхідну для досягнення консенсусу.

Цей ланцюг розповсюджується між усіма вузлами мережі (нодами), які

синхронізують між собою актуальний стан реєстру. Криптографічне хешування забезпечує зв’язок між блоками, створюючи механізм детекції та запобігання будь-яким змінам в історії транзакцій.

1.2 Принцип роботи блокчейну

Функціонування блокчейн системи базується на взаємодії ряду ключових технічних механізмів: структури блоку, криптографічного хешування, механізму додавання транзакцій, алгоритмів консенсусу та синхронізації між вузлами мережі. Комплексне розуміння цих процесів дозволяє оцінити надійність, ефективність та масштабованість блокчейн-рішень

1.2.1 Структура блоку

Кожен блок у блокчейн-мережі є цифровим контейнером, що зберігає:

- Заголовок (header), який містить технічну інформацію про блок;
- Тіло (body), що складається зі списку транзакцій.

Типовий заголовок блоку включає:

- Хеш попереднього блоку для зв'язку блоків у ланцюг;
- Мітку часу (timestamp) фіксує момент створення блоку;
- Nonce випадкове число, яке підбирається в процесі майнінгу (у PoW-системах);
- Кореневий хеш дерева Меркла, що забезпечує швидку перевірку транзакцій.

Завдяки криптографічному зв’язку між заголовками блоків створюється незворотний хронологічний ланцюг, у якому кожна зміна в одному блоці призведе до недійсності всіх наступних. Структуру блоку зображено нарисунку 1.1.

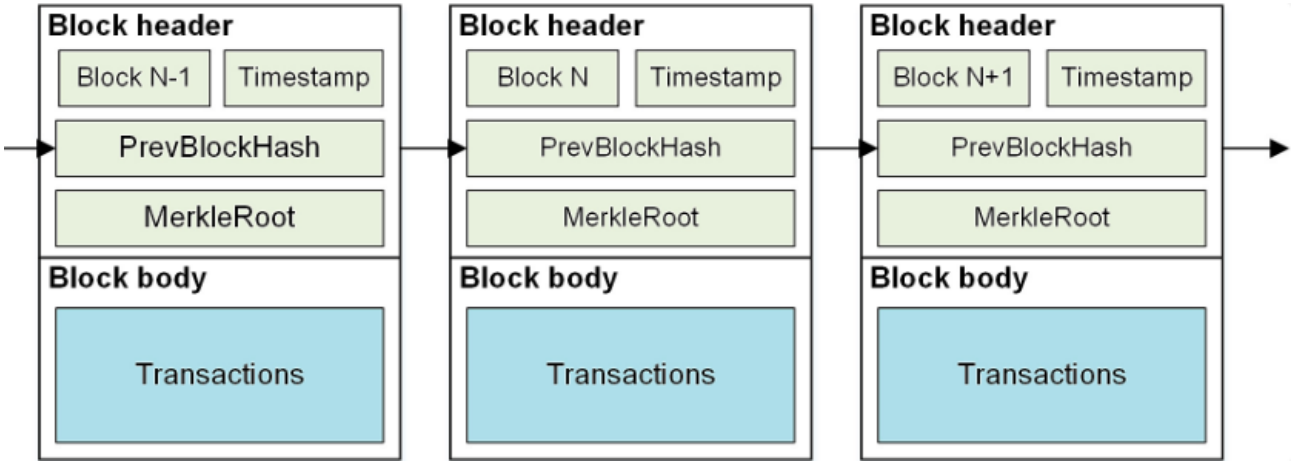


Рисунок 1.1 – Структура блоку

1.2.2 Принцип роботи транзакцій

У блокчейні транзакція - це базова одиниця передачі інформації або вартості. Транзакції мають такі властивості:

- Визначають відправника та одержувача (публічні ключі);
- Містять цифровий підпис, який гарантує автентичність;
- Перевіряються всіма учасниками мережі перед включенням до блоку.

Кожна транзакція проходить стадії:

- Створення, користувач формує транзакцію і підписує її приватним ключем;
- Поширення, вузли мережі отримують її через P2P-протокол;
- Валідація, ноди перевіряють цифровий підпис, баланс, формат;
- Включення до блоку, після підтвердження вона стає частиною блокчейну.

1.2.3 Хеш-функції

Основу криптографічної безпеки блокчейну становлять односторонні хеш-функції. У Bitcoin, наприклад, використовується SHA-256[1], яка генерує 256-бітне хеш-значення фіксованої довжини незалежно від розміру

вхідних даних.

Властивості хеш-функцій включають забезпечення однакового виходу для однакових вхідних даних, швидкість обчислення, стійкість до колізій, а також незворотність, тобто неможливість відновлення вхідних даних за їх хешем. Ці функції активно застосовуються під час формування хешу блоку, перевірки транзакцій, побудови дерева Меркла та реалізації цифрових підписів.

1.2.4 Механізм додавання блоків

У публічних блокчейнах, таких як Bitcoin або Ethereum, додавання нового блоку до ланцюга здійснюється через консенсус - узгодження між вузлами щодо того, який блок є дійсним.

У системах з Proof of Work (PoW) новий блок створюється майнерами шляхом вирішення обчислювальної задачі: знаходження nonce, при якому хеш заголовка має визначену кількість нулів на початку. Цей процес називається майнінгом, а знайдений блок вважається підтвердженим, якщо до нього приєднуються наступні блоки.

У блокчейнах із Proof of Stake (PoS) нові блоки створюються валідаторами, що володіють певною кількістю монет. Валідатор обирається випадково або за певними правилам. Наприклад, баланс та вік монет.

1.2.5 Синхронізація між вузлами

Один із принципів роботи блокчейну - одночасна наявність ідентичної копії ланцюга у всіх вузлів мережі. Це досягається за рахунок P2P-комунікації та протоколів поширення блоків і транзакцій.

У разі розгалуження ланцюга (форку) всі вузли зазвичай приймають ланцюг із найбільшою сумою підтверджень, наприклад, у PoW - з найбільшою обчислювальною складністю. Таким чином досягається

самоорганізована узгодженість даних без централізованого координатора.

1.3 Алгоритми консенсусу

Однією з ключових відмінностей блокчейн-систем від традиційних централізованих баз даних є механізм досягнення згоди між учасниками децентралізованої мережі щодо поточного стану реєстру. Цей механізм має назву консенсус. У розподіленому середовищі, де немає єдиного контролюючого центру, забезпечення достовірності та послідовності записів є критично важливим для стабільності й довіри до системи. Саме алгоритм консенсусу визначає, який з можливих варіантів продовження ланцюга блоків буде визнаний «істинним» усіма учасниками. Зображенно на рисунку 1.2

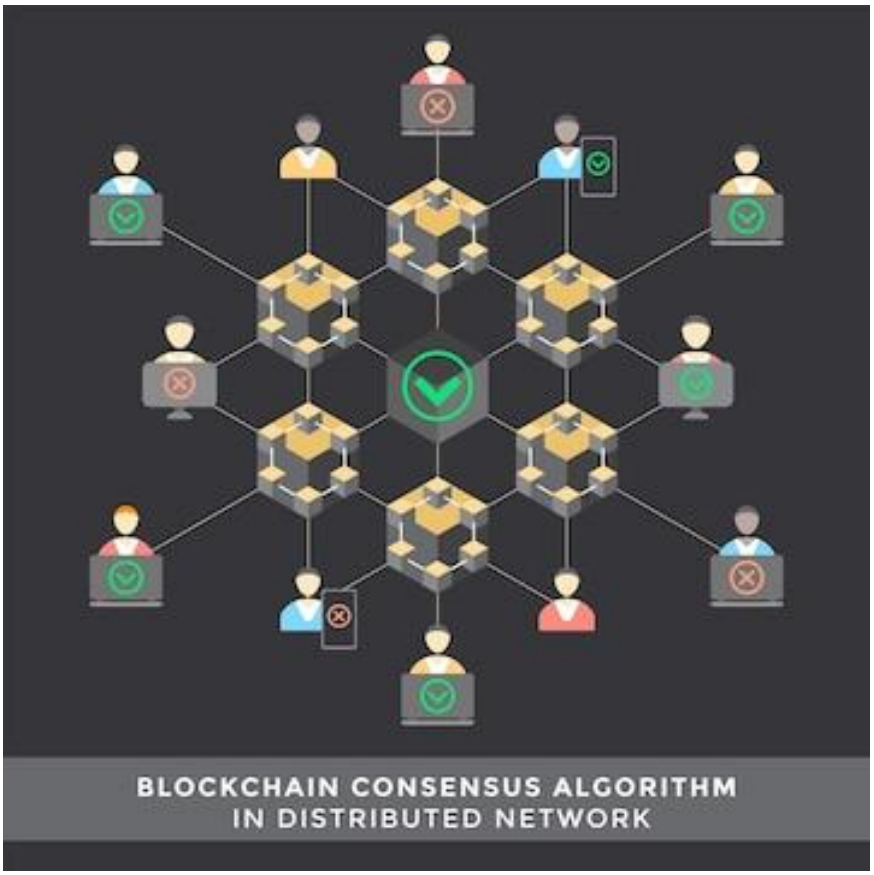


Рисунок 1.2 – Механізм консенсусу в блокчейн-мережі

Найвідомішим і історично першим алгоритмом є Proof of Work (PoW), який застосовується у блокчейні Bitcoin. Його сутність полягає в тому, що для того, щоб додати новий блок до ланцюга, вузол повинен виконати обчислювально складну задачу знайти значення nonce, при якому хеш заголовка блоку відповідатиме заданій умові (наприклад, починатиметься з певної кількості нулів). Така задача не має ефективного алгоритму розв'язання і може бути вирішена лише методом повного перебору. Цей процес вимагає значних обчислювальних ресурсів і електроенергії, але водночас ускладнює спроби фальсифікації, оскільки для зміни хоча б одного блоку довелося б перераховувати всі наступні блоки з новими хешами швидше, ніж це зможе зробити вся інша мережа.

Попри ефективність у забезпеченні безпеки, PoW має низку обмежень, насамперед енергетичну неефективність та обмежену масштабованість. У відповідь на це було запропоновано альтернативні механізми, серед яких найпоширенішим став Proof of Stake (PoS). На відміну від PoW, де вплив вузла на мережу залежить від його обчислювальної потужності, у PoS визначальним чинником є частка криптовалюти, якою володіє вузол. Власник токенів «заморожує» певну кількість монет як заставу (stake), що дає йому право брати участь у процесі валідації нових блоків. Імовірність обрання конкретного валідатора пропорційна до його частки в загальному обсязі ставок.

PoS істотно зменшує споживання енергії, пришвидшує обробку блоків і дозволяє створити більш гнучкі економічні моделі стимулювання учасників. Однак ця модель має інші потенційні вразливості, наприклад, атаки «нічого не вартує» (nothing at stake) або концентрацію влади у великих власників токенів. З метою усунення цих недоліків було запропоновано ряд модифікацій, таких як Delegated Proof of Stake (DPoS), Nominated PoS, Hybrid PoW, Hybrid PoS та інші.

Усі ці алгоритми мають одну спільну мету - гарантувати, що лише один варіант блоку буде доданий до реєстру, а будь-які спроби маніпуляції або

подвійної витрати будуть виявлені й відхилені. Вибір алгоритму консенсусу залежить від цілей конкретної системи, бажаного рівня децентралізації, пропускної здатності, стійкості до атак і моделі економічних стимулів.

Важливо підкреслити, що сучасні фреймворки, такі як Substrate, надають можливість конфігурувати або навіть змінювати алгоритм консенсусу без повного перезапуску мережі, що відкриває простір для експериментів з новими моделями розподіленого узгодження. Така гнучкість є важливим кроком у напрямку адаптивних блокчейн-мереж, що здатні еволюціонувати відповідно до вимог часу та конкретного застосування.

1.4 Класифікація блокчейнів

Блокчейн-технологія, незважаючи на єдність базових принципів - таких як децентралізація, незмінність записів та криптографічна безпека - може реалізовуватись у різних організаційних і доступових моделях. Залежно від цілей використання, вимог до контролю над доступом та ролей учасників, блокчейни умовно поділяються на публічні, приватні та консорціумні. Кожен із типів має свої переваги, обмеження та сфери застосування.

Публічний блокчейн (public blockchain) - це повністю відкритий реєстр, участь у якому можуть брати всі охочі без будь-яких дозволів. Будь-хто може під’єднатися до мережі, верифікувати транзакції, майнити нові блоки або створювати смарт-контракти. Доступність і прозорість таких систем забезпечує максимальну децентралізацію, що є основною ідеологічною цінністю публічних блокчейнів. Зображено на рисунку 1.3

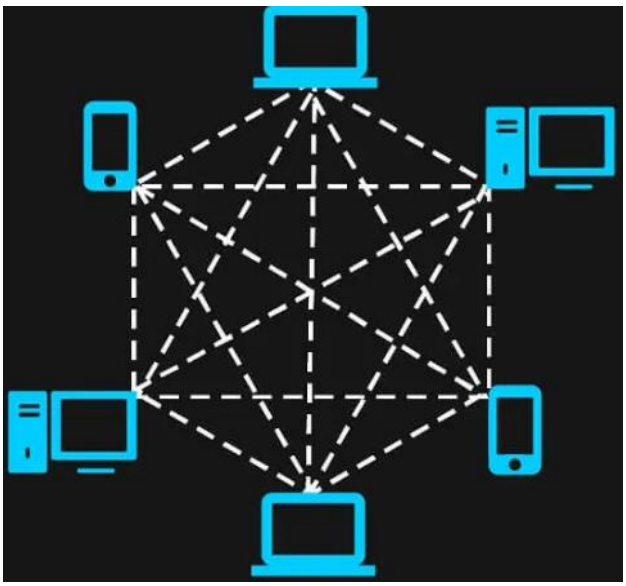


Рисунок 1.3 – Публічний блокчейн

Найвідомішими прикладами цього типу є Bitcoin, Ethereum, Litecoin, Dogecoin тощо. Усі дані у таких мережах публічні та незмінні, а рівень довіри до них досягається не через інституційну гарантію, а через математичні алгоритми консенсусу та розподілення відповідальності між десятками або сотнями тисяч вузлів. Проте публічні блокчейни характеризуються відносно низькою пропускною здатністю та високими витратами на транзакції, особливо у періоди пікового навантаження.

Приватний блокчейн (private blockchain), навпаки, функціонує у контрольованому середовищі, де участь у мережі дозволена лише визначеним учасникам. Доступ до читання, запису, валідації або адміністрування обмежується внутрішніми політиками організації або групи. Такі блокчейни використовуються здебільшого в корпоративному секторі для внутрішнього документообігу, логістичних ланцюгів, перевірки достовірності продукції тощо. Зображено на рисунку 1.4

Рисунок 1.4 – Приватний блокчейн

Прикладом приватного блокчейну є платформа Hyperledger Fabric[3], яка дозволяє створювати сегментовані ланцюги із тонким налаштуванням прав доступу. Також до цієї категорії можна віднести Corda - рішення, орієнтоване на фінансові установи[3]. Головними перевагами приватних блокчейнів є висока швидкість транзакцій, конфіденційність даних і централізоване управління, проте вони значно поступаються публічним у рівні децентралізації та відкритості.

Консорціумний блокчейн (consortium blockchain) поєднує риси як публічних, так і приватних систем. У таких мережах управління здійснюється попередньо визначеною групою учасників - наприклад, кількома банками, логістичними компаніями або державними структурами. Це дозволяє досягти балансу між довірою, масштабованістю і приватністю. Рішення приймаються колективно, а доступ до функцій мережі регламентується через делегування прав. Зображено на рисунку 1.5

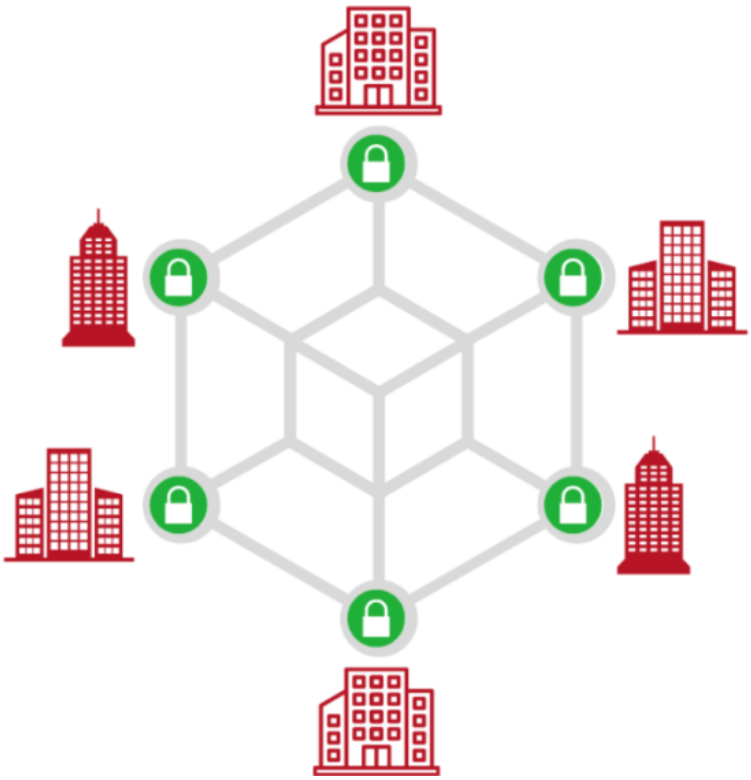


Рисунок 1.5 – Консорціумний блокчейн

Консорціумні блокчейни активно використовуються в міжорганізаційних сценаріях, де важливо забезпечити прозорість даних для кількох сторін одночасно без участі зовнішніх учасників. До прикладів можна віднести мережу Quorum, розроблену JP Morgan[3], та Energy Web Chain, яка об’єднує учасників енергетичного ринку.

Таким чином, вибір типу блокчейну залежить від конкретної мети, обсягу доступу, рівня довіри між учасниками та вимог до продуктивності. Публічні системи ідеально підходять для відкритих, децентралізованих економік, наприклад, криптовалюти або NFT, тоді як приватні та консорціумні - для корпоративного та міжорганізаційного використання, де критичною є конфіденційність, масштабованість і контроль.

2 ФРЕЙМВОРК SUBSTRATE

2.1 Характеристика фреймворку Substrate

У контексті сучасної розробки блокчейн-рішень спостерігається стійкий перехід від монолітних до модульних архітектур, які дозволяють створювати гнучкі, масштабовані та адаптивні блокчейни під специфічні задачі. Одним із найяскравіших прикладів реалізації такого підходу є Substrate - відкритий фреймворк для створення кастомних блокчейн-платформ, розроблений компанією Parity Technologies.

2.1.1 Загальна характеристика

Substrate є потужним набором бібліотек, інструментів і шаблонів, які надають розробникам можливість створювати повноцінні блокчейн-мережі як з нуля, так і на основі попередньо підготовлених компонентів. Цей фреймворк відкриває новий рівень гнучкості, дозволяючи розробникам не лише реалізовувати логіку додатків, а й повністю контролювати архітектуру самого блокчейну. На відміну від універсальних платформ на зразок Ethereum, де децентралізовані застосунки створюються у вигляді смарт-контрактів, що виконуються в межах одного глобального ланцюга, Substrate пропонує можливість сконструювати власний блокчейн із унікальними правилами консенсусу, структурою транзакцій, моделлю обліку ресурсів і внутрішньою економікою.[3]

Однією з найважливіших рис Substrate є його модульність, реалізована через систему так званих pallet'ів - окремих незалежних модулів, кожен з яких відповідає за певну функціональність: облік балансу, керування ролями користувачів, логіку голосування тощо. Ці модулі можна легко підключати, замінювати або доповнювати власними розробками, що значно прискорює

створення кастомізованих рішень. Таким чином, розробникам не потрібно будувати все з нуля або вносити зміни у монолітну структуру - вони можуть комбінувати вже готові компоненти в рамках чітко структурованої та узгодженої архітектури.

Цей підхід не лише спрощує процес розробки, а й покращує підтримку та масштабованість проєкту, дозволяючи адаптувати блокчейн до нових вимог без значних змін у кодовій базі. У результаті Substrate надає оптимальне середовище як для експериментування з новими протоколами, так і для створення стабільних, масштабованих децентралізованих систем, орієнтованих на реальні бізнес-завдання.

2.1.2 Зв’язок із Polkadot

Substrate не є тотожним Polkadot, однак вони тісно пов’язані. Polkadot це багатоланцюгова платформа (multichain)[4], яка об’єднує різні блокчейн-мережі (так звані парачейни) в одну спільну екосистему. Сам Polkadot побудований на Substrate, і всі парачейни у цій мережі також розробляються за допомогою цього фреймворку. Водночас, Substrate може використовуватись незалежно від Polkadot, для створення повністю автономних блокчейнів (standalone chains), які не потребують спільного консенсусу або взаємодії з реле-ланцюгом Polkadot.

Таким чином, Substrate виконує роль фундаментального технологічного шару, що забезпечує розширюваність, стандартизацію й сумісність блокчейн-рішень у межах Polkadot-екосистеми та за її межами.

2.1.3 Призначення та переваги

Основним призначенням Substrate є спрощення процесу розробки блокчейн-систем без втрати гнучкості, що особливо важливо в умовах динамічного розвитку технологій та постійно зростаючих вимог до

децентралізованих рішень. Цей фреймворк надає розробникам інструменти, які дозволяють не тільки швидко створювати нові блокчейн-проєкти, але й адаптувати їх до конкретних потреб бізнесу або наукових досліджень.

Однією з ключових переваг Substrate є його модульна архітектура, реалізована за допомогою системи FRAME (Framework for Runtime Aggregation of Modularized Entities). Вона дозволяє будувати бізнес-логіку блокчейну у вигляді окремих, легко керованих модулів (pallets), які можна повторно використовувати, комбінувати або змінювати відповідно до вимог проєкту. Це значно знижує поріг входу для розробників і підвищує ефективність розробки.

Ще однією важливою особливістю є підтримка WebAssembly (Wasm) - платформи для виконання коду, яка забезпечує кросплатформність і високу продуктивність. Завдяки цьому Substrate-мережі можуть працювати на різних пристроях і середовищах без потреби в спеціалізованому програмному забезпеченні. Це відкриває нові можливості для масштабування та розгортання блокчейнів у гетерогенному середовищі.

Фреймворк також тісно інтегрований з мовою програмування Rust, яка славиться своєю безпечністю, високою продуктивністю та суворою системою типів. Це гарантує не лише ефективну реалізацію низькорівневих операцій, але й підвищує загальну надійність коду.

Однією з унікальних особливостей Substrate є можливість оновлення логіки ланцюга без необхідності виконання хардфорку. Завдяки цьому розробники можуть розгортати нові версії runtime без порушення консенсусу або розгалуження мережі, що є важливою перевагою з погляду підтримки та еволюції системи.

Substrate також пропонує вбудовані шаблони вузлів, зокрема Node Template, які дозволяють швидко почати розробку власного блокчейну з базовою функціональністю. Це значно скорочує час запуску проєкту та дозволяє сконцентруватися на реалізації специфічних функцій.

Нарешті, хоча Substrate має тісний зв’язок із Polkadot і може

використовуватись для створення парачейнів у цій екосистемі, він не є жорстко прив’язаним до неї. Це означає, що розробники мають повну свободу у виборі моделі децентралізації - як в контексті автономного блокчейну, так і у вигляді частини більшої мережі.

Завдяки такому поєднанню гнучкості, інноваційності та практичності, Substrate став одним із найпопулярніших фреймворків у світі блокчейн-розробки, активно використовуючись як у прототипуванні експериментальних рішень, так і у створенні масштабованих промислових систем з високими вимогами до продуктивності, безпеки та адаптивності.

2.2 Структурна архітектура Substrate

2.2.1 Архітектурна модель та принципи побудови

Архітектура Substrate розроблена відповідно до сучасних принципів побудови складних розподілених систем і відображає багаторівневий модульний підхід, що забезпечує відокремлення логіки виконання, мережевого шару, консенсусу, сховища та зовнішніх інтерфейсів. Така модель не тільки підвищує масштабованість та керованість системи, а й дозволяє швидко адаптувати інфраструктуру під змінні вимоги бізнесу або користувача без потреби повної перебудови мережі.

Центральним архітектурним принципом Substrate є суворе розділення між «вузлом» (node) і «логікою виконання» (runtime). Це означає, що інфраструктура вузла - тобто обробка транзакцій, peer-to-peer мережа, бази даних і консенсус - ізольована від того, якими саме правилами керується система. Уся бізнес-логіка, включно з поведінкою транзакцій, правами доступу, токен-економікою, голосуванням тощо, зосереджена у runtime - окремому об’єкті, що компілюється до WebAssembly (Wasm) і завантажується вузлом під час запуску або оновлення.

Такий підхід надає низку архітектурних переваг:

- Завдяки WebAssembly runtime може бути оновлений на льоту без хардфорку або перезапуску вузлів, що критично важливо для довгострокових блокчейн-систем;

- Оскільки runtime виконується у sandbox-середовищі Wasm, воно не має прямого доступу до файлової системи або системних ресурсів ноди, що знижує ризик помилок та атак;

- Логіка може бути повністю налаштована під потреби конкретного проєкту: заміна консенсусу, токенів, ролей, механізмів голосування - усе це реалізується на рівні runtime, без зміни коду інфраструктури вузла.

Додатково, Substrate реалізує принцип інверсії управління (IoC), згідно з яким поведінка системи визначається не жорстко заданими сценаріями, а модульною конфігурацією, яку визначає розробник. Наприклад, вибір pallet-ів (модулів логіки), консенсусного механізму (AURA, BABE, GRANDPA) чи способу обліку (баланси, токени, NFT) здійснюється під час складання runtime, і ці компоненти взаємодіють за єдиними контрактами.

Архітектура Substrate також орієнтована на принцип повторного використання коду (code reusability). FRAME (Framework for Runtime Aggregation of Modularized Entities)[5] - основна частина Substrate - містить десятки перевірених pallet-ів, які реалізують стандартну функціональність (управління обліковими записами, консенсус, голосування, системні параметри тощо). Це значно прискорює процес розробки та зменшує ймовірність критичних помилок.

Ще одним ключовим архітектурним аспектом є детермінованість виконання, що означає, що кожна транзакція у Substrate повинна мати один і той самий результат незалежно від того, на якому вузлі вона виконується. Це є необхідною умовою для підтримки консенсусу у розподіленому середовищі.

Узагальнюючи, можна зазначити, що архітектурна модель Substrate є високоабстрагованою, конфігурованою та еволюційною, що дозволяє створювати блокчейн-системи нового покоління - адаптивні, безпечні та

довготривалі. Вона успішно поєднує принципи класичного системного проєктування з інноваційною природою блокчейн-мереж.

2.2.2 Рівнева організація компонентів

Архітектура Substrate є багаторівневою, що забезпечує чіткий розподіл функціональних обов'язків між складовими системи, підвищує її гнучкість, масштабованість і відмовостійкість. Кожен рівень виконує окрему роль у життєвому циклі блокчейн-системи, при цьому взаємодія між рівнями відбувається через стандартизовані інтерфейси, що дозволяє уникнути тісного зв'язування компонентів та полегшує підтримку і модернізацію коду. Інфраструктурний рівень відповідає за функціонування ноди як цілісного об'єкта в одноранговій мережі. До його завдань належать:

- Обробка вхідних та вихідних транзакцій;
- Синхронізація блоків між вузлами;
- Збереження стану ланцюга;
- Реалізація peer-to-peer з'єднання через libp2p;
- Запуск модулів консенсусу (AURA, BABE, GRANDPA);
- Підтримка пулу транзакцій та бази даних (RocksDB або інша).

Інфраструктурний рівень реалізується мовою Rust із використанням низки незалежних бібліотек, що полегшує масштабування та дає можливість повної кастомізації окремих компонентів. Саме тут визначається базова логіка роботи вузла - однаково як у мережі, що працює автономно, так і у складі мультичейн-архітектур, наприклад Polkadot.

Рівень виконання (Runtime Layer) це ядро логіки блокчейну - бізнес-рівень, який визначає «правила гри» у системі. Runtime - це самостійний, ізольований модуль, що компілюється до WebAssembly (Wasm) і виконується як частина вузла, але в межах ізольованого середовища. Така конструкція дозволяє:

- Виконувати код у sandbox-режимі без доступу до системних

ресурсів;

- Оновлювати логіку ланцюга без хардфорків;
- Досягати високої детермінованості виконання транзакцій.

У Runtime реалізуються всі pallet-и модулі, які визначають функціональність блокчейну (наприклад, управління балансами, правами, NFT, реєстрацією акаунтів, голосуванням тощо). Комбінування pallet-ів здійснюється через FRAME систему, яка автоматизує інтеграцію модулів у єдине середовище виконання.

Інтерфейсний рівень (Interface/API Layer) цей рівень забезпечує взаємодію між зовнішніми користувачами (людьми або додатками) та внутрішніми процесами блокчейну. До складу цього шару входять:

- JSON-RPC основний протокол запитів до вузла;
- WebSocket API для підписки на події в мережі в режимі реального часу;
- Polkadot JS API бібліотека на JavaScript для зручної інтеграції фронтенду[6];
- Substrate UI Tools інтерфейси керування, такі як Polkadot JS UI, Subscan, Telemetry.

Цей рівень дозволяє виконувати транзакції, спостерігати за станом блокчейну, верифікувати дані, моніторити продуктивність та здійснювати адміністративне управління.

2.3 Ключові компоненти Substrate ноди

Substrate нода це повнофункціональний програмний вузол у децентралізованій мережі, який виконує всі необхідні функції для підтримки, синхронізації, перевірки та розширення блокчейн-ланцюга. Її архітектура побудована за принципом високої модульності, де кожен компонент відповідає за окрему частину життєвого циклу блоку або транзакції, а загальна взаємодія компонентів забезпечує цілісну і надійну роботу мережі.

2.3.1 Client (ядро вузла)

Client є центральним координаційним модулем вузла. Він відповідає за:

- Завантаження, зберігання і пошук блоків;
- Ведення бази даних актуального стану;
- Взаємодію з runtime (через webassembly);
- Управління чергою транзакцій (transaction pool);
- Ініціацію процесу додавання нових блоків через консенсус.

Client функціонує як системна оболонка, що забезпечує зв’язок між усіма іншими компонентами виконанням, мережею, сховищем і API.

2.3.2 Runtime Executor

Runtime - це бізнес-логіка блокчейну, а Executor - це механізм її запуску. Executor виконує WebAssembly-код, у якому описані правила обробки транзакцій, перевірки блоків, зміни стану тощо.

Кожен новий блок перед тим, як бути доданим у ланцюг, має бути «програний» у runtime для перевірки його валідності. Цей процес називається state transition - перехід системи з одного стану в інший під впливом подій (транзакцій), визначених правилами, які містяться у виконуваному коді.

2.3.3 Transaction Pool

Transaction Pool це буфер тимчасового зберігання транзакцій, які були отримані вузлом, але ще не були включені в блок. Він виконує такі функції:

- Перевіряє підписи та структуру вхідних транзакцій;
- Виконує попередню валідацію згідно з runtime;
- Надає консенсусному модулю список транзакцій-кандидатів;
- Дозволяє фільтрацію та пріоритезацію транзакцій (наприклад, за

розміром комісії).

Наявність добре організованого пулу транзакцій дозволяє підвищити ефективність формування блоків та зменшити затримки в мережі.

2.3.4. Consensus Engine

Модуль консенсусу відповідає за досягнення узгодженого стану блокчейну між усіма учасниками мережі. У Substrate реалізовано кілька механізмів консенсусу:

- AURA (Authority Round) детермінований механізм генерації блоків з фіксованим таймінгом;
- BABE (Blind Assignment for Blockchain Extension) використовується в мережах на базі Polkadot;
- GRANDPA (GHOST-based Recursive ANcestor Deriving Prefix Agreement) для фіналізації блоків.

Важливо, що Substrate дозволяє змінювати механізм консенсусу без зупинки вузла, що є унікальною особливістю для блокчейн-фреймворків.

2.3.5 Networking Layer

Цей компонент забезпечує функціонування peer-to-peer (P2P)-мережі, в якій вузли обмінюються блоками, транзакціями, сигналами про статус ланцюга. Реалізовано на базі libp2p - модульної бібліотеки для побудови безсерверних мереж.

Networking включає:

- Discovery пошук та підключення до інших вузлів;
- Gossip поширення повідомлень;
- Synchronization вирівнювання локального стану з іншими учасниками мережі.

Мережева частина є критично важливою для підтримки узгодженого

стану системи та своєчасного розповсюдження інформації.

2.3.6 Storage (сховище даних)

Substrate використовує вбудовану ключ-значення (key-value) базу даних (RocksDB або інші адаптовані бекенди) для зберігання:

- Історії блоків;
- Стану системи після кожного блоку;
- Метаданих, індексів, хешів.

Доступ до сховища суворо контрольований і здійснюється через API, який ізольований для runtime. Це гарантує, що виконання не може вийти за межі дозволених змін стану.

2.4 Інтеграція концепції Zero Trust у архітектуру блокчейн-рішень

Сучасні вимоги до кібербезпеки корпоративних систем дедалі частіше виходять за межі традиційних підходів, заснованих на периметровій моделі захисту. Зі зростанням кількості кіберзагроз, а також поширенням хмарних сервісів, віддаленого доступу та IoT-пристроїв, актуальності набуває концепція Zero Trust - підхід, який заперечує будь-яку довіру за замовчуванням, незалежно від того, чи користувач або пристрій знаходиться всередині корпоративної мережі чи поза її межами.

Основним принципом моделі Zero Trust є «ніколи не довіряй, завжди перевіряй». У практичній реалізації це означає, що кожен запит на доступ до ресурсу супроводжується перевіркою, автентифікацією та авторизацією з урахуванням контексту (користувач, пристрій, місце, поведінка тощо). Для цього використовуються багатофакторна автентифікація (MFA), аналіз поведінки користувачів (UBA), системи виявлення загроз (EDR) і платформи моніторингу безпеки (SIEM). Важливе місце у цій концепції займає принцип мінімальних привілеїв, згідно з яким кожен суб’єкт має отримувати лише той

рівень доступу, який є необхідним для виконання конкретного завдання.

Архітектура Substrate, яка є основою для реалізації модульного блокчейну в межах даної кваліфікаційної роботи, ідеально поєднується з принципами Zero Trust. По-перше, завдяки модульності FRAME, система може гнучко реалізовувати контроль доступу до різних частин логіки блокчейна, зокрема через кастомні pallet'и. Це дозволяє вбудовувати механізми авторизації, ролей, лімітів доступу до функціональності напряму в runtime. Наприклад, голосування, створення токенів або управління правами можуть бути обмежені не тільки за ролями, а й з урахуванням історії активності, кількості токенів, рівня довіри чи інших атрибутів.

По-друге, компіляція runtime в WebAssembly дозволяє застосовувати оновлення логіки без хардфорків, що відкриває шлях до гнучкої адаптації механізмів безпеки та відповідності новим політикам доступу. Це є критичним у середовищах, де потрібно швидко реагувати на інциденти або оновлювати правила відповідно до загроз.

Крім того, архітектура Substrate дозволяє реалізовувати сегментацію мережі на логічному рівні, наприклад через поділ функціональності між окремими pallet'ами чи chain extension. Це узгоджується з принципом сегментації доступу у Zero Trust, що передбачає обмеження шляху потенційного зловмисника навіть у випадку компрометації частини системи.

У підсумку, поєднання концепції Zero Trust та блокчейн-фреймворку Substrate дозволяє створювати децентралізовані системи нового покоління з вбудованими механізмами перевірки, контролю доступу та мінімізації довіри до будь-яких компонентів інфраструктури. У контексті розробленої системи голосування за токени, це дозволяє не лише забезпечити технічну реалізацію децентралізованої логіки, але й гарантувати високий рівень безпеки, стійкість до внутрішніх загроз і відповідність сучасним вимогам корпоративного кіберзахисту.[7]

3 ПІДГОТОВКА ДО РЕАЛІЗАЦІЇ

3.1 Обґрунтування вибору фреймворку Substrate

У процесі розробки блокчейн-системи критично важливим є вибір відповідного технологічного стеку, що забезпечить необхідний рівень гнучкості, масштабованості, безпеки та ефективності. На сучасному етапі розвитку децентралізованих технологій існує декілька популярних підходів до створення блокчейн-рішень, зокрема використання платформ смарт-контрактів (як-от Ethereum, Binance Smart Chain), написання систем з нуля на низькорівневих мовах програмування або ж застосування спеціалізованих фреймворків для побудови кастомних блокчейнів. У цьому контексті фреймворк Substrate, розроблений компанією Parity Technologies, постає як один із найбільш гнучких та функціонально насичених рішень, що поєднує в собі модульність, масштабованість та сумісність із сучасними інфраструктурними рішеннями.[9]

На відміну від платформ типу Ethereum, де вся логіка реалізується у вигляді смарт-контрактів в єдиному середовищі виконання, Substrate дозволяє будувати повноцінні незалежні блокчейни зі своєю власною логікою транзакцій, правилами обліку та навіть механізмами консенсусу. Це дає змогу реалізувати такі функції, які були б неможливими або надмірно складними при використанні обмеженого середовища смарт-контрактів.

Однією з головних переваг Substrate є його модульна архітектура, побудована навколо системи FRAME (Framework for Runtime Aggregation of Modularized Entities). FRAME дозволяє створювати runtime блокчейну з набору готових модулів, які називаються pallet'ами, або ж розробляти власні модулі, орієнтовані на специфічні потреби проєкту. Цей підхід забезпечує високий ступінь повторного використання коду, легке масштабування та можливість поступового розширення функціональності без порушення

існуючої інфраструктури.

Ще однією ключовою перевагою є підтримка WebAssembly (Wasm) як середовища виконання для runtime логіки. Завдяки цьому Substrate забезпечує кросплатформну сумісність, безпеку виконання та високу продуктивність. Це особливо актуально в умовах, коли блокчейн-системи повинні бути не лише надійними, але й адаптивними до різноманітних середовищ розгортання.

Крім того, фреймворк написаний мовою Rust, яка відома своєю безпечністю, сучасною системою управління пам’яттю та високою ефективністю. Використання Rust дозволяє уникати типових помилок, пов’язаних із низькорівневою обробкою ресурсів, що вкрай важливо для критичних систем, якими є блокчейни.[10]

Особливу увагу також заслуговує готовність Substrate до інтеграції з мережею Polkadot. Завдяки цьому розроблений блокчейн може функціонувати як парачейн у загальній екосистемі, отримуючи додаткові переваги від спільної безпеки, міжланцюгової взаємодії та консенсусу рівня релеєного ланцюга. Водночас, важливо зазначити, що використання Substrate не обмежує розробника вимогою приєднання до Polkadot - фреймворк повністю підтримує створення автономних блокчейнів, які можуть функціонувати незалежно від будь-якої більшої мережі.

У порівнянні з іншими технологіями, такими як Tendermint (Cosmos SDK), Hyperledger Fabric або Ethereum SDK (наприклад, Hardhat чи Truffle), Substrate надає найбільш повну свободу у визначенні архітектури блокчейну на всіх рівнях - від структури даних до логіки консенсусу та системи оновлень. Це робить його ідеальним вибором для побудови інноваційних, модульних та адаптивних рішень у сфері децентралізованих технологій.[13]

Таким чином, вибір фреймворку Substrate для розробки модульного блокчейну є обґрунтованим з точки зору як технічної ефективності, так і стратегічної перспективності, оскільки він дозволяє реалізувати максимально гнучку архітектуру з підтримкою сучасних вимог до безпеки,

масштабованості та сумісності з екосистемами нового покоління.[8]

3.2 Архітектура реалізації модульного блокчейну

Базова архітектура проєкту, реалізованого за допомогою Substrate, має три основні компоненти:

- Node відповідає за мережеву інфраструктуру, обробку блоків, пірингову комунікацію, синхронізацію та API-взаємодію (через RPC/WebSocket);
- Runtime ядро блокчейну, в якому реалізовано логіку транзакцій, механізми обліку, консенсус та інші правила функціонування мережі;
- Pallets модульна система FRAME, яка дозволяє конструювати логіку системи як набір незалежних частин.

Ця трирівнева структура дозволяє чітко розділити мережеву частину, бізнес-логіку та функціональні модулі, забезпечуючи максимальну гнучкість і масштабованість рішення.

3.2.1 Вузол (Node)

Вузол - це основний виконавчий компонент Substrate, що відповідає за взаємодію з мережею, обробку блоків, генерацію нових блоків, телеметрію та надання зовнішніх інтерфейсів для клієнтів (наприклад, Polkadot.js). У розробці використовується Substrate Node Template - офіційний шаблон від Parity, який надає стартову точку для реалізації власного блокчейну.[19]

Функціонально вузол містить такі компоненти:

- Мережевий стек libp2p;
- Модуль консенсусу (Aura/BABE + GRANDPA)[20];
- RPC-сервер для взаємодії з клієнтами;
- Механізм інтеграції з runtime через wasm або native виконання[11].

3.2.2 Виконуване середовище (Runtime)

Runtime містить усю логіку, що визначає правила існування блокчейну. У межах цього компонента реалізується формат і обробка транзакцій, встановлюються правила зміни стану системи, визначаються ролі користувачів, адміністраторів і валідаторів, а також задаються правила генерації нових блоків. Окрім цього, саме у runtime розміщено механізм зберігання даних і їх оновлення відповідно до прийнятих транзакцій. Runtime компілюється у WebAssembly, що забезпечує кросплатформність, безпечне виконання та можливість оновлення логіки блокчейну без необхідності проведення хардфорків, зберігаючи цілісність і безперервність функціонування мережі.

3.2.3 Використання вбудованих pallet'ів

Для реалізації базової функціональності блокчейну буде використано низку вбудованих pallet'ів, що входять до складу FRAME. Зокрема, використовується модуль pallet-system, який відповідає за управління блоками, транзакціями та ідентифікаторами облікових записів. Pallet pallet-balances реалізує механізми збереження токенів та їх передачу між користувачами. Модуль pallet-timestamp додає часову мітку до кожного блоку, що є необхідним для забезпечення коректної роботи консенсусу та фіксації подій. Також використовується pallet-sudo, який надає адміністративний контроль над мережею на етапі розробки. Сукупність цих модулів створює готову інфраструктуру для запуску блокчейн-системи, істотно спрощуючи процес розробки та пришвидшуючи реалізацію стандартного функціоналу.[10]

3.2.4 Масштабованість та подальше розширення

Передбачена архітектура є відкритою для масштабування і передбачає можливість подальшого розширення функціональності. У разі потреби до системи можуть бути додані нові pallet'и, зокрема для реалізації механізмів голосування, підтримки NFT або керування доступом. Крім того, можлива заміна або вдосконалення алгоритмів консенсусу відповідно до вимог конкретного середовища чи проєкту. Також архітектура дозволяє інтеграцію з мережею Polkadot або іншими парачейнами, забезпечуючи міжланцюгову взаємодію. Таким чином, обрана модель дає змогу не лише швидко реалізувати мінімально життєздатний продукт (MVP), а й надалі масштабувати його до повноцінної, гнучкої та модульної децентралізованої платформи.

3.3 Опис функціональних вимог

У межах даного проєкту блокчейн-система має реалізовувати базову функціональність для обміну цифровими активами між учасниками мережі. Основна логіка зосереджується навколо створення, зберігання та виконання заявок на обмін, що дозволяє двом сторонам здійснити безпечну угоду відповідно до визначених умов. Такий підхід є актуальним для багатьох застосувань, пов’язаних з ринковими моделями, внутрішніми токенами, умовним виконанням операцій (escrow), а також побудовою децентралізованих платформ для взаємодії користувачів. [15]

Функціонально блокчейн має забезпечувати можливість:

- Створення заявки на обмін одного активу на інший;
- Збереження активних заявок у ланцюгу з відкритим доступом до перегляду;
- Підтвердження заявки іншою стороною та автоматичного виконання обміну за умови відповідності;
- Фіксації результатів угоди в історії транзакцій.

У рамках реалізації буде використовуватися власний pallet, який

включатиме всі перелічені можливості, а також забезпечить контроль відповідності балансу, коректність введених даних та реєстрацію подій, пов’язаних із кожним етапом угоди.

Щодо структури взаємодії, передбачено участь двох основних типів користувачів: ініціатора обміну та контрагента. Обидві сторони є рівноправними у системі, однак виконують різні дії: перший створює заявку, другий - приймає її. У подальшому архітектура може бути розширена за рахунок додавання ролей модератора, адміністратора або спеціалізованих смарт-агентів, які автоматизують частину операцій.

Усі дії учасників взаємодіють із pallet'ом через extrinsics - транзакції, що виконують виклики визначених функцій. Зовнішні інтерфейси (наприклад, Polkadot.js або JSON-RPC API) дозволять користувачам створювати та підтверджувати обміни, переглядати активні заявки, а також відстежувати події в мережі.

Система розробляється з урахуванням вимог до надійності, відкритості історії змін, а також безпеки - кожна транзакція буде супроводжуватись перевірками прав доступу та достатності ресурсів. У перспективі функціональність може бути доповнена механізмами рейтингу, комісій, таймерів очікування або автоматичного скасування заявок.

Таким чином, функціональні вимоги до системи формують основу для створення простого, але гнучкого механізму децентралізованого обміну, який може бути розширений і адаптований під широкий спектр застосувань.

4 РЕАЛІЗАЦІЯ МОДУЛЬНОГО БЛОКЧЕЙНУ

4.1 Ініціалізація проекту

4.1.1 Налаштування середовища розробки

Початковим етапом реалізації блокчейн-проєкту стало налаштування робочого середовища для збирання та запуску Substrate-мережі. Як основну платформу було обрано операційну систему Ubuntu 24.04 LTS, запущену у середовищі WSL 2 (Windows Subsystem for Linux), що дозволяє поєднати переваги UNIX-подібної системи з доступом до ресурсів локального комп’ютера під керуванням Windows. [11]

На цьому етапі було виконано оновлення системи та встановлення необхідних компонентів для побудови Substrate-проєктів, таких як інструменти компіляції, криптографічні бібліотеки, засоби роботи з мережевими протоколами та обробки даних. Також було встановлено мову програмування Rust, яка є основною для розробки логіки Substrate-блокчейнів, та налаштовано середовище для компіляції виконуваного коду у форматі WebAssembly. Окремо було додано підтримку таргету wasm32-unknown-unknown, який дозволяє компілювати ядро блокчейну в платформонезалежному вигляді. [12]

Після цього було виконано клонування шаблонного проєкту Solochain Template з офіційного репозиторію Polkadot SDK. Цей шаблон містить базову структуру Substrate-проєкту з мінімальним набором компонентів: вузол, runtime, набір стандартних pallet'ів, а також конфігураційні файли для запуску та збирання.

На рисунку 4.1 зображено результат успішного клонування репозиторію, а також базову структуру директорій проєкту. Видно, що шаблон включає окремі підкаталоги для модулів (pallets), логіки виконання

(runtime), вузлової частини (node), документації (docs) та інструментів налаштування середовища.

```
root@WSL:~# git clone https://github.com/paritytech/polkadot-sdk-solochain-template.git
cd polkadot-sdk-solochain-template
Cloning into 'polkadot-sdk-solochain-template'...
remote: Enumerating objects: 276, done.
remote: Counting objects: 100% (122/122), done.
remote: Compressing objects: 100% (73/73), done.
remote: Total 276 (delta 87), reused 49 (delta 49), pack-reused 154 (from 2)
Receiving objects: 100% (276/276), 306.87 KiB | 2.26 MiB/s, done.
Resolving deltas: 100% (118/118), done.
root@WSL:~/polkadot-sdk-solochain-template# ls -la
total 364
drwxr-xr-x  9 root root   4096 Jun 16 11:46 .
drwx------ 16 root root   4096 Jun 16 11:45 ..
drwxr-xr-x  8 root root   4096 Jun 16 11:46 .git
drwxr-xr-x  4 root root   4096 Jun 16 11:46 .github
-rw-r--r--  1 root root      8 Jun 16 11:46 .gitignore
-rw-r--r--  1 root root 302081 Jun 16 11:46 Cargo.lock
-rw-r--r--  1 root root   4359 Jun 16 11:46 Cargo.toml
-rw-r--r--  1 root root    763 Jun 16 11:46 Dockerfile
-rw-r--r--  1 root root   1210 Jun 16 11:46 LICENSE
-rw-r--r--  1 root root   9129 Jun 16 11:46 README.md
drwxr-xr-x  2 root root   4096 Jun 16 11:46 docs
drwxr-xr-x  2 root root   4096 Jun 16 11:46 env-setup
drwxr-xr-x  3 root root   4096 Jun 16 11:46 node
drwxr-xr-x  3 root root   4096 Jun 16 11:46 pallets
drwxr-xr-x  3 root root   4096 Jun 16 11:46 runtime
root@WSL:~/polkadot-sdk-solochain-template#
```

Рисунок 4.1 - Клонування репозиторію та перегляд структури проєкту

4.1.2 Складання проєкту та встановлення залежностей

Після успішного налаштування середовища розробки та клонування шаблону Solochain Template наступним етапом стала спроба виконати компіляцію проєкту. Процес збирання в Substrate-проєктах є досить ресурсоємким, оскільки передбачає завантаження та побудову великої кількості залежностей, включно з криптографічними бібліотеками, інструментами WebAssembly, а також елементами, пов’язаними із зберіганням даних та мережею. [18]

На етапі первинної компіляції проєкту система автоматично почала завантаження необхідних crate-пакетів із публічних репозиторіїв. На рисунку 4.2 наведено фрагмент виводу системи під час збирання - видно, що процес активно використовує кешовані ресурси, а також компілює окремі компоненти у відповідності до цільового середовища виконання.

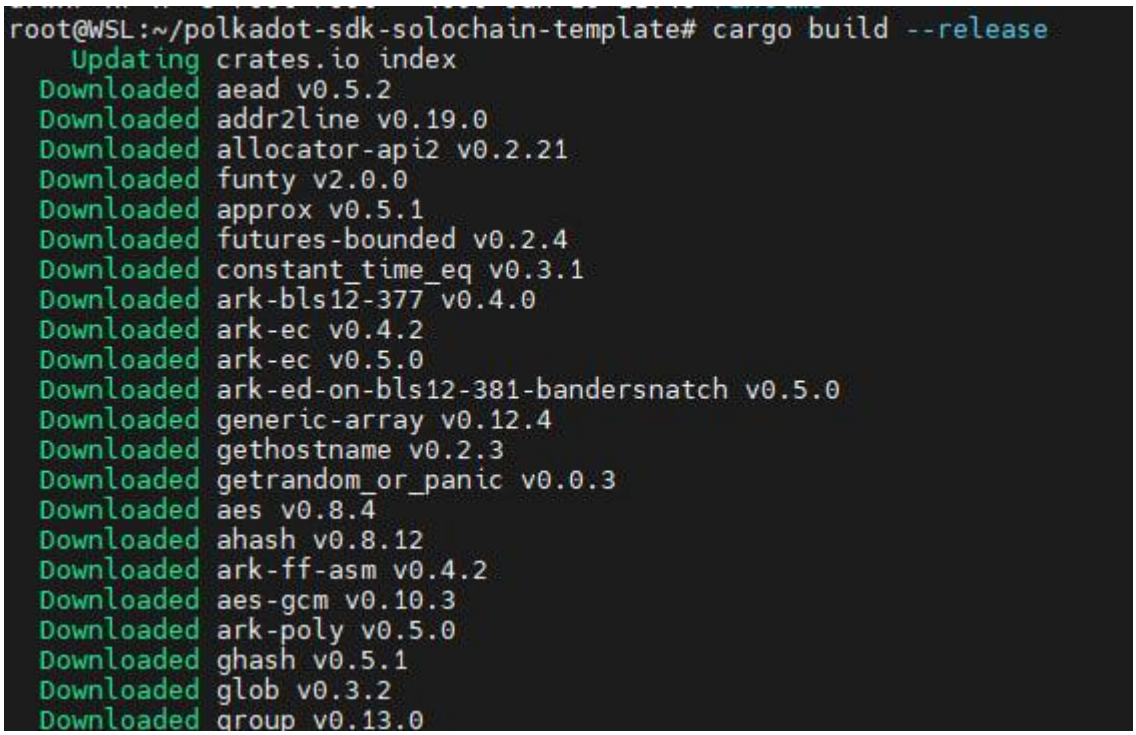


Рисунок 4.2 - Початок компіляції проєкту та завантаження залежностей

Під час першої спроби компіляції виникла типова для Substrate-систем помилка, пов’язана з відсутністю бібліотеки libclang, яка є обов’язковою для успішної збірки компонента librocksdb-sys. RocksDB використовується як вбудована база даних блокчейн-вузла, і її збірка потребує наявності clang, llvm, а також цілого набору допоміжних бібліотек для підтримки стиснення (zlib, snappy, lz4, zstd тощо). Усі ці пакети були встановлені вручну, після чого також було налаштовано змінну середовища LIBCLANG_PATH, що вказувала системі компілятора на правильне розташування бібліотек.

Однак навіть після встановлення залежностей процес компіляції завершувався з помилкою через відсутність компонента rust-src. Цей компонент містить вихідний код стандартної бібліотеки Rust, необхідний для компіляції виконуваного середовища (runtime) у форматі WebAssembly. Додавання rust-src через стандартний менеджер інструментів Rust (rustup) дозволило вирішити цю проблему. [17]

Після виконання усіх зазначених кроків було здійснено повторну спробу компіляції проєкту. Збірка завершилась успішно: проєкт повністю

скомпільовано у release-режимі, що підтвердило коректність середовища та правильну інтеграцію всіх залежностей. Це створило необхідні умови для переходу до наступного етапу - тестового запуску блокчейн-вузла та перевірки роботи всієї системи.

4.1.3 Тестовий запуск блокчейн-вузла

Після успішної компіляції проєкту було здійснено запуск зібраного вузла у режимі розробника. Такий запуск дозволяє швидко протестувати роботу мережі, не підключаючись до зовнішніх пірів і не потребуючи додаткової конфігурації. Вузол запускається з тимчасовим сховищем даних та автоматично створює блоки з інтервалом приблизно шість секунд.

Під час запуску виводиться інформація про поточну конфігурацію мережі, включаючи назву вузла, роль у мережі (AUTHORITY), специфікацію ланцюга (Development), параметри операційної системи, архітектуру процесора, доступну пам’ять, а також стан бази даних RocksDB. Також автоматично активується механізм консенсусу GRANDPA і запускаються внутрішні сервіси, зокрема:

- JSON-RPC сервер для взаємодії з клієнтами;
- Prometheus експортер, який надає доступ до метрик вузла;
- libp2p модуль, відповідальний за мережеву комунікацію.

На рисунку 4.3 наведено фрагмент журналу запуску вузла. Видно, що блоки успішно створюються, мають унікальні хеші та включають мінімальний набір транзакцій, що свідчить про правильне функціонування виконуваного середовища.

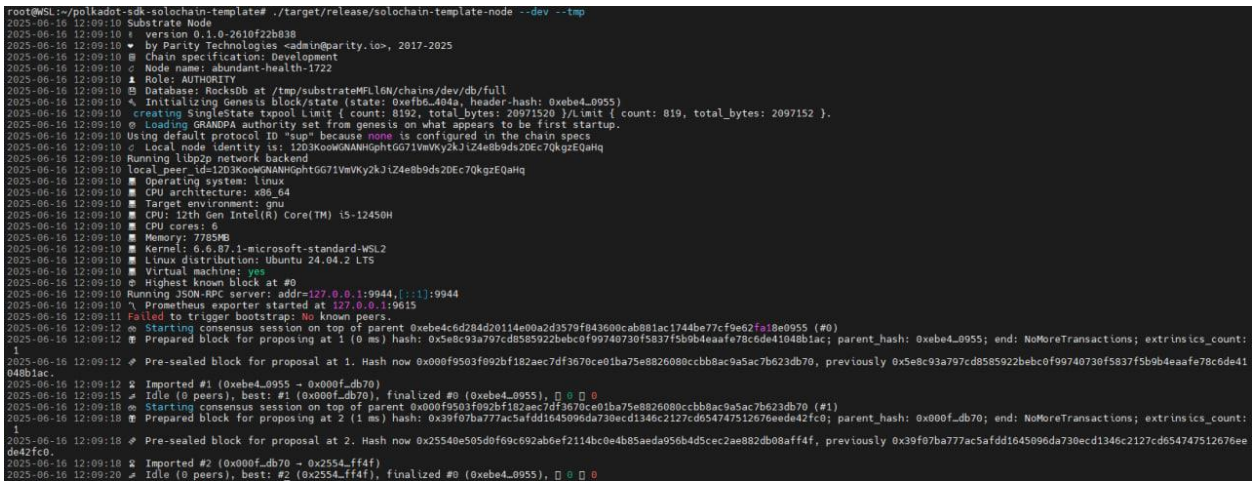


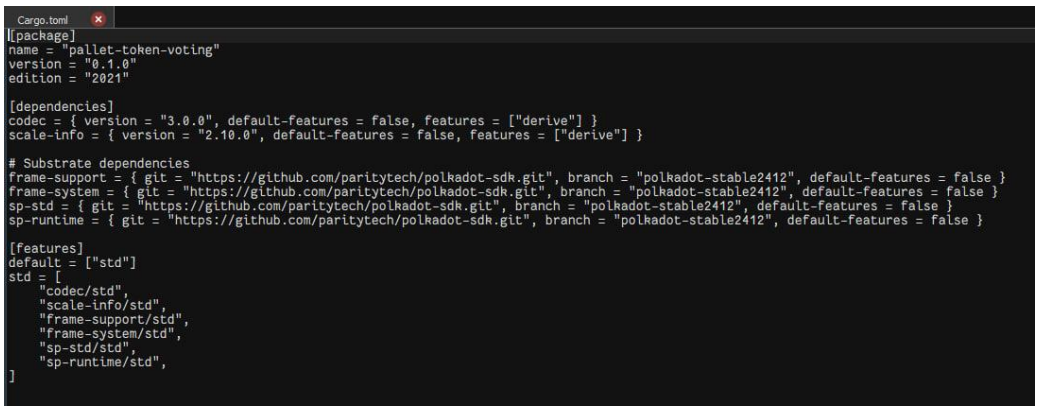
Рисунок 4.3 – Логи запуску Solochain-вузла у тестовому (dev) режимі

Таким чином, успішний запуск вузла у режимі розробника підтверджує працездатність системи. Блоки створюються автоматично, консенсус функціонує, а основні інтерфейси працюють належним чином.

4.2 Створення кастомної палети TokenVoting

Для розширення функціональності блокчейн-мережі було розроблено власну палету TokenVoting, яка дозволяє користувачам ініціювати голосування за створення нових токенів. Основна ідея полягає в децентралізованому підході: будь-який учасник мережі може запропонувати токен, а спільнота - підтримати або відхилити його шляхом голосування.

Реалізація починається зі створення нової директорії pallets/token-voting, в якій розміщуються два головні файли: Cargo.toml (конфігурація залежностей) та lib.rs (реалізація логіки палети). У Cargo.toml були зазначені всі необхідні залежності, включаючи frame-support, frame-system, scale-info та codec. Зображено на рисунку 4.4

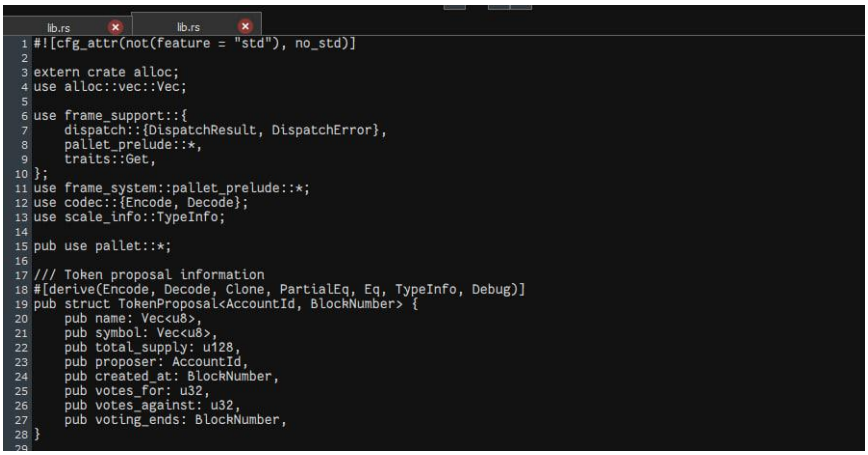


```toml
[package]
name = "pallet-token-voting"
version = "0.1.0"
edition = "2021"

[dependencies]
codec = { version = "3.0.0", default-features = false, features = ["derive"] }
scale-info = { version = "2.10.0", default-features = false, features = ["derive"] }

# Substrate dependencies
frame-support = { git = "https://github.com/paritytech/polkadot-sdk.git", branch = "polkadot-stable2412", default-features = false }
frame-system = { git = "https://github.com/paritytech/polkadot-sdk.git", branch = "polkadot-stable2412", default-features = false }
sp-std = { git = "https://github.com/paritytech/polkadot-sdk.git", branch = "polkadot-stable2412", default-features = false }
sp-runtime = { git = "https://github.com/paritytech/polkadot-sdk.git", branch = "polkadot-stable2412", default-features = false }

[features]
default = ["std"]
std = [
    "codec/std",
    "scale-info/std",
    "frame-support/std",
    "frame-system/std",
    "sp-std/std",
    "sp-runtime/std",
]
```

Рисунок 4.4 - Файл конфігурації Cargo.toml

У файлі lib.rs було реалізовано повноцінну структуру палети з використанням сучасного атрибутивного підходу. Зображено на рисунку 4.5 До структури палети входять:

- типи для зберігання пропозицій (TokenProposal) та затверджених токенів (ApprovedToken);
- конфігурація параметрів голосування;
- сховища (StorageMap, StorageDoubleMap, StorageValue) для збереження даних про голоси, токени, баланси тощо;
- події (Event) для сповіщення про ключові дії;
- чотири публічні функції (propose_token, vote, finalize_voting, transfer_token), які реалізують основну логіку палети.



```rust
#![cfg_attr(not(feature = "std"), no_std)]

extern crate alloc;
use alloc::vec::Vec;

use frame_support::{
    dispatch::{DispatchResult, DispatchError},
    pallet_prelude::*,
    traits::Get,
};
use frame_system::pallet_prelude::*;
use codec::{Encode, Decode};
use scale_info::TypeInfo;

pub use pallet::*;

/// Token proposal information
#[derive(Encode, Decode, Clone, PartialEq, Eq, TypeInfo, Debug)]
pub struct TokenProposal<AccountId, BlockNumber> {
    pub name: Vec<u8>,
    pub symbol: Vec<u8>,
    pub total_supply: u128,
    pub proposer: AccountId,
    pub created_at: BlockNumber,
    pub votes_for: u32,
    pub votes_against: u32,
    pub voting_ends: BlockNumber,
}
```

Рисунок 4.5 - Файл реалізації lib.rs

Після завершення початкової розробки було здійснено перевірку синтаксису, що дало змогу переконатися у відсутності критичних помилок та коректності реалізації. Зображено на рисунку 4.6

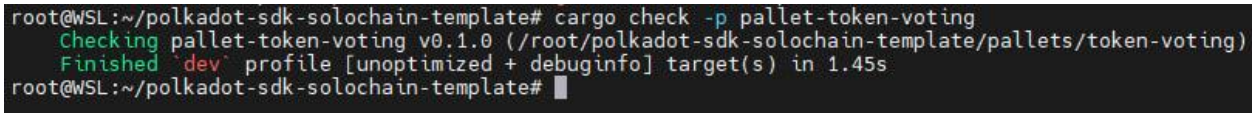


Рисунок 4.6 - Успішна перевірка компіляції палети

Таким чином, кастомна палета TokenVoting була успішно створена та підготовлена до подальшої інтеграції в runtime блокчейну. Наступним етапом стане реалізація логіки голосування та обробка результатів.

4.3 Інтеграція палети TokenVoting у runtime

Після реалізації логіки голосування в палеті TokenVoting, наступним етапом стало її підключення до загального середовища виконання (runtime). Це необхідно для того, щоб дана палета могла взаємодіяти з іншими компонентами блокчейна, викликати події, зберігати дані у сховище, а також бути доступною для виклику ззовні через extrinsics. [10]

На першому етапі відбулося додавання палети до файлу Cargo.toml у каталозі runtime. Це дозволяє системі компіляції бачити нову палету та включати її під час збирання. Залежність вказується через локальний шлях до палети, після чого cargo знає, де саме шукати відповідний код. На рисунку 4.7 показано, як виглядає цей блок із підключенням pallet-token-voting.

```
[dependencies]
codec = { features = ["derive"], workspace = true }
frame-benchmarking = { optional = true, workspace = true }
frame-executive.workspace = true
frame-metadata-hash-extension.workspace = true
frame-support = { features = ["experimental"], workspace = true }
frame-system-benchmarking = { optional = true, workspace = true }
frame-system-rpc-runtime-api.workspace = true
frame-system.workspace = true
frame-try-runtime = { optional = true, workspace = true }
pallet-aura.workspace = true
pallet-balances.workspace = true
pallet-grandpa.workspace = true
pallet-sudo.workspace = true
pallet-template.workspace = true
pallet-token-voting = { path = "../pallets/token-voting", default-features = false }
pallet-timestamp.workspace = true
pallet-transaction-payment-rpc-runtime-api.workspace = true
pallet-transaction-payment.workspace = true
scale-info = { features = ["derive", "serde"], workspace = true }
serde_json = { workspace = true, default-features = false, features = ["alloc"] }
sp-api.workspace = true
sp-block-builder.workspace = true
sp-consensus-aura = { features = ["serde"], workspace = true }
sp-consensus-grandpa = { features = ["serde"], workspace = true }
sp-core = { features = ["serde"], workspace = true }
sp-genesis-builder.workspace = true
sp-inherents.workspace = true
sp-keyring.workspace = true
sp-offchain.workspace = true
sp-runtime = { features = ["serde"], workspace = true }
sp-session.workspace = true
sp-storage.workspace = true
sp-transaction-pool.workspace = true
sp-version = { features = ["serde"], workspace = true }

```

Рисунок 4.7 - Підключення палети до секції залежностей у runtime

Наступним кроком було включення цієї палети до секції features. У контексті Substrate ця секція відповідає за активацію тих чи інших можливостей залежно від цільової платформи. [14] Зокрема, для звичайного режиму виконання потрібно включити підтримку стандартної бібліотеки Rust (std). Для цього додали pallet-token-voting/std до відповідного списку. Це дозволяє палеті використовувати типи на зразок Vec, працювати з подіями та логікою зберігання. Зображено на рисунку 4.8

```
[features]
default = ["std"]
std = [
  "codec/std",
  "frame-benchmarking?/std",
  "frame-executive/std",
  "frame-metadata-hash-extension/std",
  "frame-support/std",
  "frame-system-benchmarking?/std",
  "frame-system-rpc-runtime-api/std",
  "frame-system/std",
  "frame-try-runtime?/std",
  "pallet-aura/std",
  "pallet-balances/std",
  "pallet-grandpa/std",
  "pallet-sudo/std",
  "pallet-template/std",
  "pallet-token-voting/std",
  "pallet-timestamp/std",
  "pallet-transaction-payment-rpc-runtime-api/std",
  "pallet-transaction-payment/std",
  "scale-info/std",
  "serde_json/std",
  "sp-api/std",
  "sp-block-builder/std",
  "sp-consensus-aura/std",
  "sp-consensus-grandpa/std",
  "sp-core/std",
  "sp-genesis-builder/std",
  "sp-inherents/std",
  "sp-keyring/std",
  "sp-offchain/std",
  "sp-runtime/std",
  "sp-session/std",
  "sp-storage/std",
  "sp-transaction-pool/std",
  "sp-version/std",
  "substrate-wasm-builder",
]
```

Рисунок 4.8 - Включення палети до стандартних можливостей std

Далі, у файлі runtime/src/lib.rs, було зареєстровано нову палету в макросі оголошення runtime. У сучасних версіях Substrate замість макросу construct_runtime! використовується більш гнучка форма з анотацією. Зарезервував для палети індекс 8 і присвоїли їй псевдонім TokenVoting. Таким чином, вона стала частиною структури виконання і зможе взаємодіяти з іншими pallet'ами. Крім цього, ми також виконали імпорт самої палети - тобто, зробили її видимою у файлі runtime, що дозволяє використовувати її публічні типи та функції. Це необхідно, щоб компілятор не викидав помилку про відсутній тип pallet_token_voting. [9]

Останнім важливим кроком стало налаштування самої палети через реалізацію її конфігураційного трейту. Це було виконано у файлі configs/mod.rs, який містить конфігурації для всіх pallet'ів, що використовуються у проєкті. Для TokenVoting ми вказали тип подій, мінімальну кількість голосів - 3, та тривалість голосування у блоках - 10. Це дає можливість адаптувати поведінку палети до потреб конкретного ланцюга.

Таким чином, палета була повністю інтегрована до runtime. Усі ключові налаштування - залежності, особливості компіляції, структура виконання та конфігурація - були коректно враховані. Це дозволило перейти до наступного етапу: компіляції всього проєкту для перевірки правильності змін. Зображено на рисунку 4.9

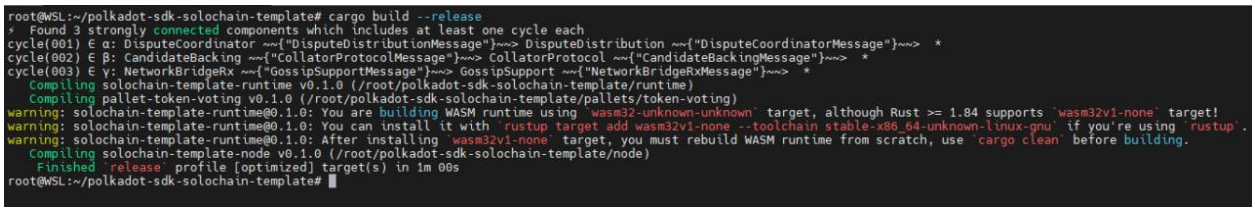


Рисунок 4.9 - Успішна компіляція з інтегрованою палетою TokenVoting

4.4 Запуск та тестування блокчейну

Після завершення інтеграції нової палети TokenVoting у середовище Substrate-ланцюга, наступним етапом є її запуск та перевірка функціональності в умовах живого тестування. Метою цього етапу є впевнитися, що всі ключові компоненти (ініціація пропозиції токена, процес голосування, завершення голосування та передача токенів) працюють відповідно до закладеної логіки.

Для запуску тестового блокчейна використовується стандартна команда запуску у dev-режимі. Цей режим дозволяє миттєво ініціалізувати нову тимчасову мережу без потреби в додатковому налаштуванні або збереженні даних між сесіями. Успішний запуск засвідчується появою повідомлення про активний RPC-сервер на порту 9944, а також регулярним створенням нових блоків у логах терміналу. Зображено на рисунку 4.10

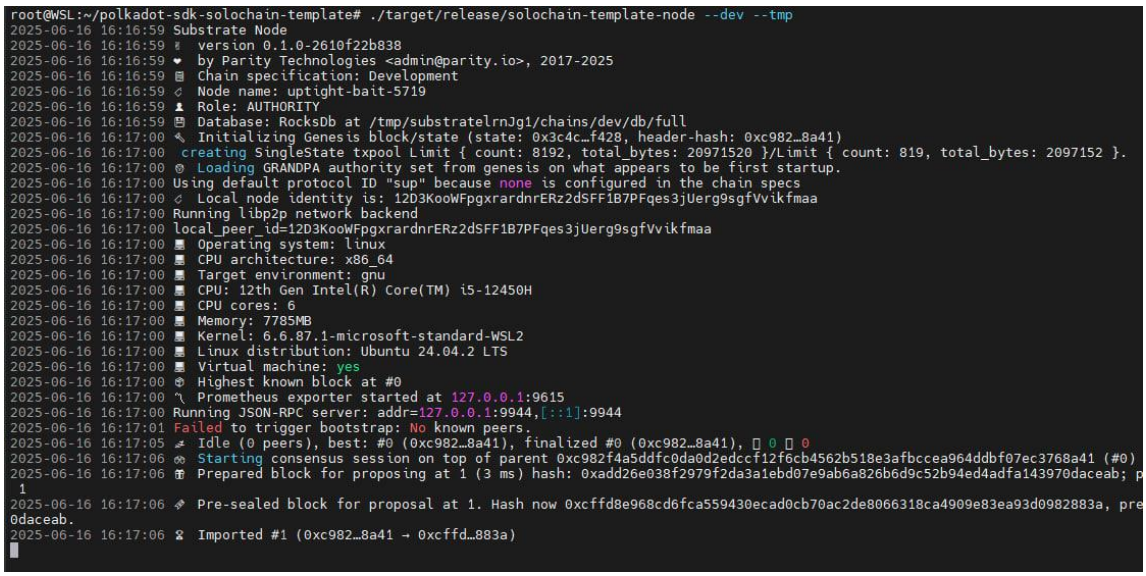


Рисунок 4.10 - Запуск блокчейна з інтегрованою палетою у dev-режимі

Для взаємодії з блокчейном використовується інтерфейс Polkadot.js Apps, який надає зручний веб-засіб для перевірки та виклику extrinsics. Після запуску локального вузла користувач підключається до нього через інтерфейс Polkadot.js, обравши опцію підключення до Local Node. Підключення до адреси ws://127.0.0.1:9944 підтверджує, що вузол працює коректно, і дозволяє приступити до тестування палети. Зображено на рисунку 4.11.

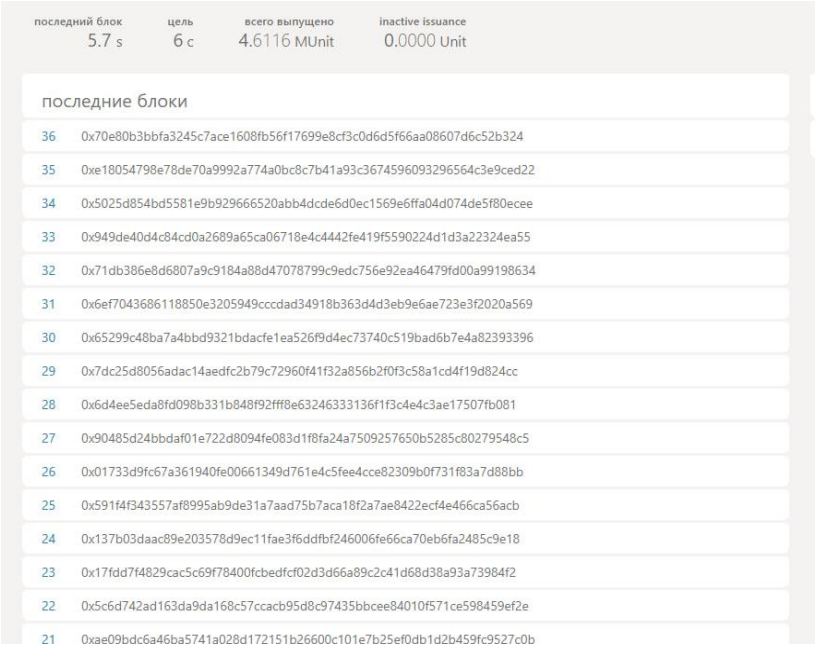


Рисунок 4.11 - Підключення до локального вузла через Polkadot.js Apps

Після підключення у розділі Developer Extrinsics, де у списку доступних extrinsic’ів вже присутня наша палета tokenVoting. Це підтверджує, що вона була успішно інтегрована в runtime. У розкривному списку доступні чотири функції: proposeToken, vote, finalizeVoting та transferToken. Наявність усіх цих функцій свідчить про повноцінну присутність палети у виконуваному середовищі. Зображено на рисунку 4.12.

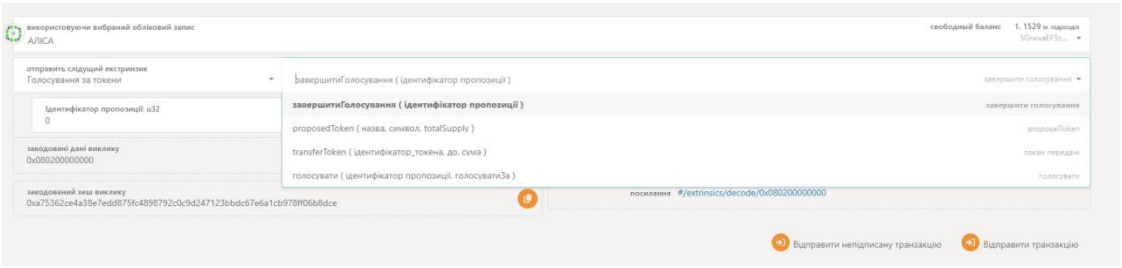


Рисунок 4.12 - Перевірка доступності extrinsic'ів палети TokenVoting

Для перевірки функціональності палети було створено пропозицію нового токена, що включає назву, символ та загальну кількість. Після відправки транзакції proposeToken у системі з'являється подія TokenProposed, яка містить ID пропозиції, адресу ініціатора та введені параметри токена. Таким чином, підтверджується правильність роботи механізму ініціації голосування. Зображено на рисунку 4.13

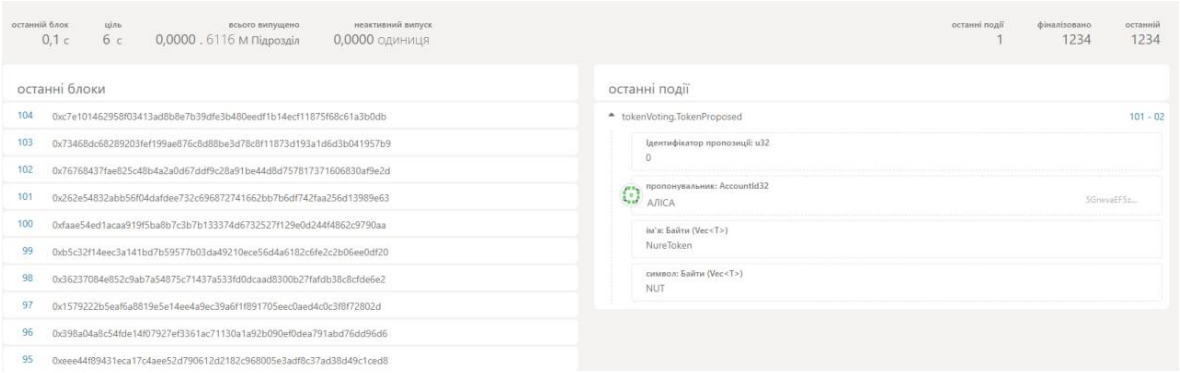


Рисунок 4.13 - Подія TokenProposed в Explorer після створення пропозиції

Після створення пропозиції проводиться симуляція голосування шляхом надсилання транзакції vote з різних облікових записів. У стандартній

конфігурації для схвалення необхідно щонайменше три голоси. Після успішного голосування трьох учасників, система очікує завершення періоду голосування, визначеного параметром VotingPeriod. У початковій версії він дорівнював 10 блокам, однак для зручності тестування його було збільшено до 100.

Після закінчення періоду голосування викликається extrinsic finalizeVoting. Якщо усі умови виконано (мінімальна кількість голосів та більшість "за"), в системі з'являється подія TokenApproved. Це означає, що токен був успішно створений, а його початковий обсяг було передано автору пропозиції. Зображено на рисунку 4.14

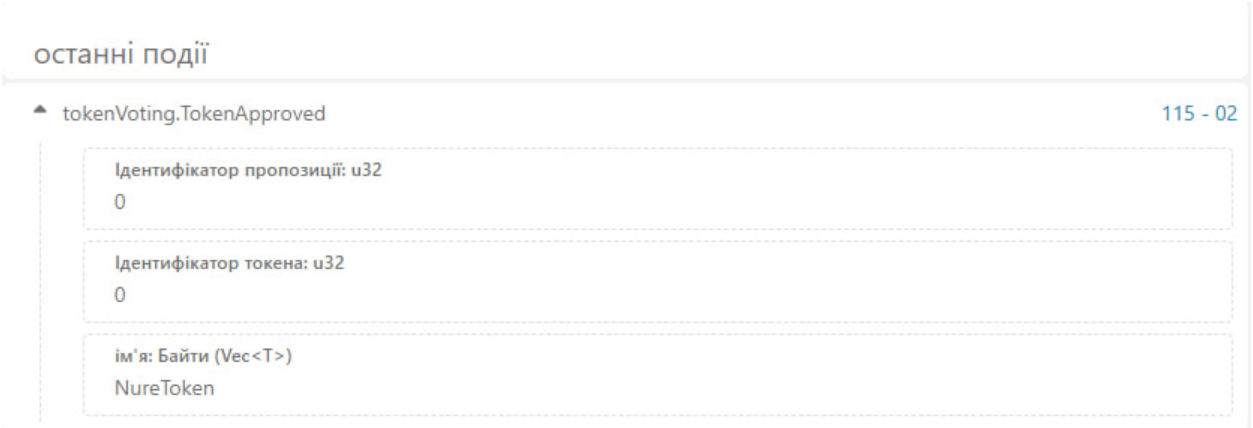


Рисунок 4.14 - Подія TokenApproved після завершення голосування

На завершальному етапі тесту перевіряється функціональність передачі токенів. Користувач, який є власником токенів, викликає transferToken, вказуючи одержувача та кількість. У випадку успішної транзакції система генерує подію TokenTransferred, що свідчить про зміну балансу токенів між акаунтами. Зображено на рисунку 4.15

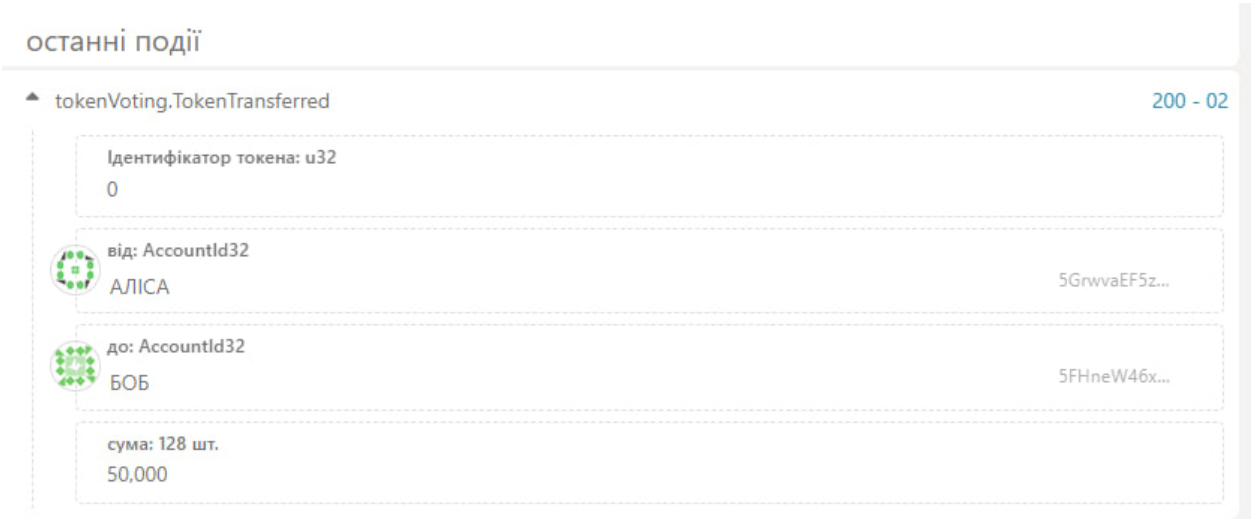


Рисунок 4.15 - Подія TokenTransferred

Таким чином, у ході тестування було підтверджено, що палета TokenVoting коректно вбудована у runtime Substrate, забезпечує повний цикл голосування та створення токенів, дозволяє передачу токенів після їх генерації, а також є інтерактивною та керованою через інтерфейс Polkadot.js Apps. Ці результати підтверджують повну працездатність реалізованої системи голосування за токени на основі власної палети, що є ключовим досягненням експериментальної частини цієї кваліфікаційної роботи.

ВИСНОВКИ

У межах цієї кваліфікаційної роботи було реалізовано повноцінну систему голосування за створення токенів у середовищі Substrate шляхом розробки та інтеграції власної палети TokenVoting. Робота охопила всі ключові етапи життєвого циклу блокчейн-компонента - від проектування логіки та конфігурації runtime до компіляції, запуску та тестування в середовищі Polkadot.js Apps.

У теоретичній частині було досліджено архітектурні особливості Substrate, принципи побудови runtime та модульного підходу з використанням палет, а також концепцію децентралізованого голосування в контексті блокчейн-технологій. Це дозволило обґрунтувати вибір архітектурного рішення для реалізації кастомної палети, яка надає користувачам можливість ініціювати голосування за створення нового токена, брати участь у голосуванні та, у разі досягнення кворуму, автоматично створювати і передавати токени між учасниками мережі.

У практичній частині було створено палету TokenVoting з усіма необхідними структурами, логікою обробки пропозицій і голосів, а також з подіями для відстеження результатів. Було виконано повну інтеграцію палети у runtime блокчейна, включаючи налаштування залежностей, стандартних можливостей та конфігурацій. Після успішної компіляції оновлений блокчейн було запущено, а палета протестована через Polkadot.js Apps. Проведене тестування підтвердило, що створена система працює стабільно, підтримує всі етапи голосування та дозволяє здійснювати передачу токенів після їх створення.

Отримані результати свідчать про працездатність реалізованої палети, а також про ефективність підходу до модульної розробки в екосистемі Substrate.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Nakamoto S. Bitcoin: A Peer-to-Peer Electronic Cash System [Електронний ресурс]. – 2008. – Режим доступу: https://bitcoin.org/bitcoin.pdf

2. Wood G. Ethereum: A secure decentralised generalised transaction ledger [Електронний ресурс] / Gavin Wood. – Ethereum Project Yellow Paper, 2014. – Режим доступу: https://ethereum.github.io/yellowpaper/paper.pdf

3. Buterin V. A Next-Generation Smart Contract and Decentralized Application Platform [Електронний ресурс]. – 2014. – Режим доступу: https://github.com/ethereum/wiki/wiki/White-Paper

4. Parity Technologies. Substrate Blockchain Framework [Електронний ресурс]. – Режим доступу: https://substrate.dev

5. Substrate FRAME v3 Documentation [Електронний ресурс]. – Режим доступу: https://docs.substrate.io/v3/runtime/frame

6. Substrate Node Template GitHub [Електронний ресурс]. – Режим доступу: https://github.com/substrate-developer-hub/substrate-node-template

7. Буканов І.В., Сітніков В.І. Використання Zero Trust підходу в корпоративній кібербезпеці: тези доповідей п’ятнадцятої міжнародної науково-технічної конференції. Т.3: секції 3,4. Баку: ІСУ АР; Харків: НТУ «ХПІ»; Харків: ХНУРЕ; Харків: НАУ «ХАІ»; Жиліна: УмЖ. 2025. С. 135.

8. Rust Programming Language. Офіційна документація [Електронний ресурс]. – Режим доступу: https://www.rust-lang.org

9. WebAssembly. Офіційна документація [Електронний ресурс]. – Режим доступу: https://webassembly.org

10. Polkadot.js Apps Portal [Електронний ресурс]. – Режим доступу: https://polkadot.js.org/apps

11. Cosmos SDK Overview [Електронний ресурс]. – Режим доступу: https://docs.cosmos.network

12. Binance Smart Chain Documentation [Електронний ресурс]. – Режим

доступу: https://docs.bnbchain.org

13. Hyperledger Fabric Documentation [Електронний ресурс]. – Режим доступу: https://hyperledger-fabric.readthedocs.io

14. Open Runtime Module Library (ORML) [Електронний ресурс]. – Режим доступу: https://github.com/open-web3-stack/open-runtime-module-library

15. Parity SCALE Codec Specification [Електронний ресурс]. – Режим доступу: https://docs.rs/parity-scale-codec/latest/parity_scale_codec/

16. Substrate Custom Pallet Development Guide [Електронний ресурс]. – Режим доступу: https://docs.substrate.io/tutorials/work-with-pallets

17. RocksDB: A Persistent Key-Value Store for Flash and RAM Storage [Електронний ресурс]. – Facebook Inc. – Режим доступу: https://github.com/facebook/rocksdb

18. libp2p Documentation [Електронний ресурс]. – Режим доступу: https://docs.libp2p.io