Міністерство освіти і науки України
Харківський національний університет імені В.Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Спеціальність 125 «Кібербезпека»
Освітня програма «Кібербезпека»

В.о. зав. кафедрою КІСМіТ
Марина ЄСІНА
"Допущено до захисту"
« »_____2025р.

Пояснювальна записка

до кваліфікаційної роботи бакалавра

на тему: «Розробка архітектури і програмного забезпечення вузла блокчейн-мережі децентралізованої системи захищеного голосування»

оцінка « »
Голова ЕК
Мичуда Л.З.

Керівник: д.т.н., проф. Олійников Р.В.
Рецензент: доц. Родінко М.Ю.
Виконавець: студент групи КБ-42
Терновський Є.В.

Харків 2025

РЕФЕРАТ

Дипломна робота містить 60 сторінок, 22 рисунки, 70 джерел інформації і 1 додаток з програмним кодом.

Метою роботи є розробка архітектури та програмного забезпечення вузла блокчейн-мережі децентралізованої системи захищеного електронного голосування.

Об'єктом дослідження є процес організації захищеного електронного голосування з використанням децентралізованої блокчейн-мережі.

Предметом дослідження є архітектура, програмне забезпечення та функціональні компоненти вузла блокчейн-мережі, призначеного для забезпечення захищеного прийому, обробки та поширення голосів у децентралізованій системі електронного голосування.

Основними методами дослідження у роботі є застосування структурного проектування, криптографічного захисту даних, а також методи тестування веб-сервісів через інструмент Postman. Реалізація здійснена з використанням Java, Spring Boot, WebFlux, JWT, PostgreSQL.

Результати роботи - у межах роботи спроектовано та реалізовано прототип вузла, що забезпечує безпечну реєстрацію адміністратора, автентифікацію, приймання бюлетенів, обмін повідомленнями між вузлами та обробку консенсусу. Вузол використовує асинхронну архітектуру та базові криптографічні перевірки, що є основою для подальшого впровадження повноцінного безпечного електронного голосування. Реалізовано перевірку підпису бюлетеня на основі ECDSA.

Рекомендації щодо використання - розроблений вузол може бути основою для побудови прототипів децентралізованих систем електронного голосування у навчальних, дослідницьких або пілотних проектах у сфері децентралізованого прийняття рішень.

Значущість роботи - отримані результати підтвердили можливість побудови децентралізованого вузла голосування на сучасних технологіях з використанням відкритих рішень. Система забезпечує базовий рівень безпеки, функціональності

та масштабованості, необхідний для реалізації електронного голосування з дотриманням вимог конфіденційності та достовірності.

Перспективи подальшого розвитку: автоматизація тестування, розробка веб-інтерфейсу адміністратора, покращення криптографічного захисту, інтеграція з мобільними клієнтами, розширення механізму консенсусу.

Ключові слова: БЛОКЧЕЙН, ДЕЦЕНТРАЛІЗОВАНА СИСТЕМА, ЕЛЕКТРОННЕ ГОЛОСУВАННЯ, JAVA SPRING BOOT, КРИПТОГРАФІЯ, JWT, КОНСЕНСУС, ВУЗОЛ, POSTMAN, WEBFLUX, ECDSA.

ABSTRACT

The diploma work contains 60 pages, 22 drawings, 70 references and 1 supplement with a program code.

The aim of the diploma work is to develop the architecture and software of a blockchain network node for a decentralized secure electronic voting system.

The object of the research is the process of organizing secure electronic voting using a decentralized blockchain network.

The subject of the research is the architecture, software, and functional components of a blockchain network node designed to ensure secure reception, processing, and dissemination of votes within a decentralized electronic voting system.

The main research methods include the analysis of existing solutions, structural design, cryptographic data protection methods, and web service testing using Postman. The implementation was carried out using Java, Spring Boot, WebFlux, JWT, and PostgreSQL.

Research results – within the scope of this work, a prototype node was designed and implemented that provides secure administrator registration, authentication, ballot reception, inter-node messaging, and consensus message processing. The node uses asynchronous architecture and basic cryptographic validation, laying the groundwork for a full-scale secure electronic voting system. Ballot signature verification is implemented based on ECDSA.

Recommendations for use – the developed node can serve as a foundation for building prototypes of decentralized electronic voting systems in educational, research, or pilot projects in the field of e-governance.

Significance of the work – the obtained results confirm the feasibility of building a decentralized voting node based on modern technologies using open-source solutions. The system provides a basic level of security, functionality, and scalability required for implementing electronic voting with respect to confidentiality and data integrity.

Prospects for further development include test automation, administrator web interface development, enhanced cryptographic protection, integration with mobile clients, and consensus mechanism extension.

Keywords: BLOCKCHAIN, DECENTRALIZED SYSTEM, ELECTRONIC VOTING, JAVA SPRING BOOT, CRYPTOGRAPHY, JWT, CONSENSUS, NODE, POSTMAN, WEBFLUX, ECDSA.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД — база даних

ДЕГ — децентралізовані системи електронного голосування

КЗІ — криптографічний засіб інформаційного захисту

НД ТЗІ — нормативний документ технічного захисту інформації

СЗІ — система захисту інформації

API — Application Programming Interface

AES — Advanced Encryption Standard

DDoS — Distributed Denial of Service

DID — Decentralized Identifier

DLT — Distributed Ledger Technology

gRPC — Google Remote Procedure Call

IPFS — InterPlanetary File System

JSON — JavaScript Object Notation

P2P — Peer-to-Peer

PBFT — Practical Byzantine Fault Tolerance

PoS — Proof of Stake

PoW — Proof of Work

TLS — Transport Layer Security

UML — Unified Modeling Language

ВСТУП

У сучасному світі цифрові технології проникають у всі сфери суспільного життя, включаючи політичні процеси. Електронне голосування (e-voting) розглядається як перспективний напрямок модернізації виборчих систем, спрямований на підвищення прозорості, ефективності та доступності виборчого процесу. Однак традиційні електронні системи голосування стикаються з низкою проблем, пов'язаних із безпекою, довірою та збереженням приватності виборців [25].

Технологія блокчейн, яка характеризується децентралізованістю, незмінністю та прозорістю, пропонує нові можливості для вирішення цих проблем. Застосування блокчейну в електронному голосуванні дозволяє забезпечити цілісність даних, запобігти фальсифікаціям та підвищити довіру громадськості до виборчого процесу. Країни, такі як Естонія та Швейцарія, вже експериментують із впровадженням блокчейн-орієнтованих систем голосування, демонструючи потенціал цієї технології в реальних умовах [14], [16].

Незважаючи на переваги, впровадження блокчейну в системи електронного голосування супроводжується низкою викликів. Серед них — забезпечення анонімності виборців, захист від атак типу Sybil, Replay та Eclipse, а також інтеграція з існуючими інформаційними системами. Крім того, важливо враховувати вимоги до масштабованості та продуктивності системи, особливо в умовах національних виборів з мільйонами учасників.

У цьому контексті особливу увагу слід приділити архітектурі вузлів блокчейн-мережі, які є основними елементами інфраструктури системи електронного голосування. Від їхньої надійності, безпеки та ефективності залежить стабільність та функціональність всієї системи. Тому розробка архітектури та побудова вузла блокчейн-мережі є ключовими завданнями для забезпечення успішного впровадження децентралізованої системи захищеного голосування [49].

Метою даної дипломної роботи є розробка архітектури та побудова вузла блокчейн-мережі, який забезпечить безпечне, прозоре та ефективне

функціонування децентралізованої системи електронного голосування. Для досягнення цієї мети необхідно вирішити наступні завдання:

1. Провести аналіз існуючих архітектурних підходів до побудови вузлів блокчейн-систем та обґрунтувати вибір оптимальної моделі для системи електронного голосування.
2. Ознайомитися з сучасними інструментами та технологіями реалізації програмного забезпечення вузла, включаючи мови програмування, фреймворки та бази даних.
3. Дослідити механізми взаємодії вузлів у мережі, зокрема мережеві протоколи обміну, механізми консенсусу та засоби захисту від атак.
4. Розробити архітектуру програмного забезпечення вузла з урахуванням вимог до безпеки, масштабованості та ефективності.
5. Реалізувати програмне забезпечення вузла з використанням обраних технологій та провести його тестування.

Об'єктом дослідження є процес побудови вузла блокчейн-мережі для децентралізованої системи електронного голосування. Предметом дослідження — архітектурні рішення, технології та методи реалізації програмного забезпечення вузла, що забезпечують його безпечне та ефективне функціонування.

Актуальність теми зумовлена необхідністю впровадження сучасних технологій у виборчі процеси для підвищення їхньої прозорості, достовірності та довіри з боку громадськості. Розробка надійної архітектури вузла блокчейн-мережі є важливим кроком на шляху до створення ефективної системи електронного голосування, яка відповідатиме сучасним вимогам безпеки та функціональності.

Таким чином, дана дипломна робота спрямована на вирішення актуальних завдань у сфері електронного голосування та сприятиме розвитку інформаційних технологій у демократичних процесах.

1 АКТУАЛЬНИЙ СТАН І ПЕРСПЕКТИВИ РОЗВИТКУ СИСТЕМ ЕЛЕКТРОННОГО ГОЛОСУВАННЯ

1.1 Огляд сучасних систем електронного голосування

«Електронне голосування (e-voting)» - це використання комп'ютерів або комп'ютеризованого обладнання для голосування на виборах. Електронне голосування спрямоване на збільшення участі, зниження витрат на проведення виборів та підвищення гарантій точності результатів [1].

Порівняно з традиційним паперовим голосуванням, система електронного голосування (е-голосування) передбачає використання електронних засобів у процесі голосування, включаючи маркування виборчого бюлетеня, запис голосування, кодування даних/інформації для передачі на головний сервер, а також їх поєднання та розташування остаточних результатів виборів. Ефективна система електронного голосування має забезпечувати прозорість виборів, захист від кіберзагроз, точність, швидкість, конфіденційність, доступність, об'єктивність, економічну ефективність та контекстуальна (соціальна/культурна) стійкість [2].

Серед сучасних систем електронного голосування виділяють централізовані, децентралізовані та гібридні системи голосування.

Централізовані системи електронного голосування — це такі системи, у яких усі ключові компоненти (сервери, база даних виборців, механізми підрахунку голосів) контролюються одним центральним органом, зазвичай державним. Вони є найпоширенішим видом у багатьох країнах, де організація виборів здійснюється органами державної влади або Центральною виборчою комісією [3].

Такі системи можуть бути як повністю електронними, так і використовувати електронні машини лише на дільницях. Наприклад, в Естонії система інтернет-голосування побудована навколо простої та добре вивченої концепції криптографії з відкритим ключем. Виборці шифрують свої бюлетені відкритим ключем виборчої системи, а потім цифровим підписом їх власним приватним ключем. Цей приватний ключ – ключ підпису на національному посвідченні особи виборця (ID-картка), який гарантує, що кожен виборець матиме приватний ключ і засіб його

використання, і що виборчий орган може надійно пов’язати кожного виборця з його правильним відкритим ключем [4].

Перевагами серед централізованих систем голосування є [4]:

1) Швидкість підрахунку. Централізовані системи забезпечують швидкий підрахунок голосів, що дозволяє оперативно оголошувати результати виборів [5].
2) Простота адміністрування. Єдиний центр управління спрощує адміністрування та інтеграцію з державними реєстрами.
3) Централізована перевірка. Можливість централізованої перевірки результатів і технічної підтримки сприяє ефективному управлінню виборчим процесом [4].

Але також централізовані системи мають і недоліки [6]:

1) Єдина точка відмови. Централізований характер системи створює єдину точку відмови, що може призвести до серйозних наслідків у разі технічних збоїв або кібератак.
2) Обмежена прозорість. Виборці не завжди можуть самостійно перевірити, чи їх голос був врахований правильно, що може знижувати довіру до системи.
3) Ризик витоку даних. Централізоване зберігання даних підвищує ризик витоку персональної інформації виборці [6].

Децентралізовані системи електронного голосування — це системи, в яких зберігання, обробка та перевірка голосів розподілені між кількома незалежними вузлами мережі. Такий підхід виключає єдину точку контролю, підвищує прозорість процесу голосування та забезпечує надійність системи навіть у випадку виходу з ладу окремих її елементів [7].

Однією з найбільш перспективних основ для реалізації децентралізованого голосування є технологія блокчейн. Технологія блокчейн підтримується розподіленою мережею, що складається з великої кількості взаємопов’язаних вузлів. Кожен із цих вузлів має власну копію розподіленого обліку, який містить повну історію всіх транзакцій, оброблених мережею. Немає єдиної сторони, яка б контролювала мережу. Якщо більшість вузлів погоджуються, вони приймають

транзакцію. Ця мережа дозволяє користувачам залишатися анонімними. Базовий аналіз технології блокчейн (включаючи смарт-контракти) свідчить про те, що вона є підходящою основою для електронного голосування, і, крім того, вона може мати потенціал зробити електронне голосування більш прийнятним і надійним [8].

У такої системи виділяють наступні переваги:

1) Прозорість голосування. Відкритий код і запис голосів у блокчейні дозволяють публічно перевіряти цей процес.
2) Незмінність даних. Неможливо змінити або видалити голос після його подання.
3) Стійкість до збоїв. Відсутність центрального сервера виключає “єдину точку відмови”.

Але незважаючи на це, у децентралізованих систем також є і недоліки:

1) Проблеми з анонімністю. Публічний характер блокчейну ускладнює повне приховування зв’язку між виборцем і голосом без додаткових заходів [9].

2) Масштабованість. Блокчейн-системи можуть бути повільними при великій кількості транзакцій.

3) Високі вимоги до обчислювальних ресурсів.

Гібридні системи електронного голосування — це поєднання електронного голосування з фізичним підтвердженням, зазвичай у вигляді паперового бюлетеня [10]. Такий підхід дає змогу використовувати переваги цифрових технологій (зручність, швидкість, автоматизація) і водночас зберігати можливість перевірки результатів вручну [10].

У гібридних системах голосування виборець робить свій вибір за допомогою електронного інтерфейсу, після чого система генерує та друкує паперовий бюлетень із зазначеним вибором [11]. Такий підхід поєднує зручність електронного голосування з надійністю паперових бюлетенів, що забезпечує можливість перевірки та аудиту результатів виборів.

Серед переваг у гібридних системах є те, що вони:

1) Забезпечують перевірюваність результатів за допомогою фізичних носіїв [12].

2) Підвищують довіру виборців, оскільки є можливість побачити та підтвердити власний голос.

3) Уможливлюють аудит і повторний підрахунок — важливий елемент прозорих виборів [13].

Водночас є і недоліки, тому що такі системи:

1) Вимагають додаткового обладнання — принтерів, сканерів, систем зберігання паперових бюлетенів [13].
2) Потребують логістики та простору для фізичних носіїв.
3) Можуть ускладнити організацію виборів, особливо у виборах з великою кількістю виборців.

1.2 Проблемні аспекти розгорнутих систем електронного голосування

Попри очевидні переваги, впровадження електронного голосування супроводжується низкою системних і технологічних викликів, які безпосередньо впливають на безпеку, довіру громадян, прозорість результатів та законність виборчого процесу.

1) Кіберзагрози та уразливості серверів.

Однією з найгостріших проблем розгорнутих систем електронного голосування є ризик зовнішніх і внутрішніх атак на сервери, які зберігають або обробляють виборчі дані. Зокрема, централізовані архітектури створюють "єдину точку відмови" — сервер або вузол, компрометація якого може порушити цілісність усього процесу голосування [14].

Ці загрози включають [14]:

- Злом серверної інфраструктури з метою зміни голосів або результатів.
- DDoS-атаки, які можуть заблокувати доступ виборців до системи.
- Використання шкідливого коду в середині оновлень або бекенду системи.
- Атаки на канал передачі даних, що можуть перехопити або змінити трафік (Man-in-the-Middle).

Приклад з системи i-Voting в Естонії. У дослідженні Springall et al. (2014) було доведено [14], що система i-Voting в Естонії мала уразливості, які дозволяли

атакуючим перехопити виборчі ключі або підмінити голоси, якщо сервери були зламані або скомпрометовані в момент ініціалізації.

2) Фізичний доступ до обладнання

Окрім програмних вразливостей, фізичний доступ до компонентів системи електронного голосування становить значну загрозу безпеці. У багатьох країнах використовуються DRE-машини (Direct Recording Electronic), які зберігають голоси у внутрішній пам’яті. Якщо зловмисник отримає доступ до такого пристрою, він може перепрошити прошивку, підключити шкідливий носій (USB), замінити логи голосування або відтворити результат не залишивши слідів втручання [15].

У дослідженні Feldman, Halderman і Felten (2007) було показано [15], що машини Diebold можна зламати менш ніж за 10 хвилин, використовуючи звичайний USB-носій. Це дозволяло замінити офіційне програмне забезпечення на модифіковане, яке змінювало підрахунок голосів у реальному часі.

3) Закритість вихідного коду

Одним із фундаментальних принципів безпечної електронної системи голосування є можливість її незалежної перевірки. Проте значна частина реалізованих систем використовує закритий вихідний код, що унеможливлює [16]:

- незалежну оцінку коду на наявність уразливостей, бекдорів чи помилок;
- публічну перевірку коректності функціонування системи;
- аудит ланцюга голосування — від введення до підрахунку.

Як прикладу, у 2019 році Швейцарія призупинила публічне тестування системи е-голосування [16], розробленої Scytl, після того як група дослідників виявила уразливості, які дозволяли маніпулювати голосами без можливості виявлення. Крім того, обмежений доступ до коду системи перешкоджав верифікації та викликав критику з боку наукової спільноти.

4) Відсутність перевірки голосу виборцем

Однією з ключових переваг електронного голосування має бути можливість перевірити, що голос було враховано правильно, не порушуючи при цьому таємниці вибору. Це досягається за допомогою так званої end-to-end verifiability

(E2E) — концепції, за якої виборець отримує криптографічне підтвердження того, що його голос був зареєстрований, а громадськість може перевірити, що всі голоси підраховано правильно і жодна з цих перевірок не розкриває змісту вибору [17].

Але проблема полягає в тому, що більшість реалізованих систем не мають E2E. У багатьох централізованих або закритих системах немає механізмів для перевірки голосу. Виборець після голосування не отримує ніякого доказу, що його голос було враховано — лише надія на чесність системи. У такій ситуації не можна довести фальсифікацію, не можна підтвердити правильність підрахунку та немає прозорого механізму контролю з боку громадськості [18].

5) Порушення приватності

Приватність голосування є фундаментальною вимогою демократичних виборів [19]. У контексті електронного голосування забезпечення цієї приватності є складним технічним завданням. Багато реалізацій або не реалізують належних криптографічних методів, або мають технічні можливості відстеження дій виборця [19].

Приватність може порушуватися під час:

- Ведення логів дій користувачів під час сесії голосування.
- Зв’язок між ідентифікатором виборця та його голосом у внутрішніх базах даних [16].
- Публічність блокчейн-записів, навіть у зашифрованому вигляді, дозволяє через мета аналіз спробувати встановити зв’язок між транзакціями та особами [19].

6) Примус і підпис у дистанційному голосуванні

В умовах віддаленого електронного голосування зростає ризик впливу третіх осіб на виборця, що підриває принцип вільного волевиявлення. Якщо голосування відбувається вдома або в іншому неконтрольованому середовищі, неможливо гарантувати, що виборець голосує добровільно і самостійно [20].

До типових загроз відносяться:

- Фізичний або психологічний тиск (з боку родичів, керівників, партій);

- Підкуп голосів — виборець може показати скріншот або підтвердження голосу;
- Передача доступу до системи голосування третім особам.

Ці загрози особливо характерні для систем, які не реалізують receipt-freeness — принцип, що унеможливлює виборцю доказати третій стороні, як він проголосував [17].

У звітах Ради Європи наголошується [20], що виборче законодавство повинно забороняти застосування віддаленого голосування без належних технічних і правових гарантій, які забезпечують свободу вибору виборця.

1.3 Модель загроз

Системи електронного голосування, як критична частина демократичного процесу, повинні бути захищені від широкого спектра загроз. Ретельно розроблена модель загроз дозволяє ідентифікувати слабкі місця в архітектурі, оцінити потенційні вектори атак та вжити відповідних заходів захисту [21-23].

Модель загроз для систем електронного голосування охоплює три основні аспекти інформаційної безпеки: цілісність, конфіденційність та доступність.

Цілісність означає, що інформація про виборчий процес (голоси, протоколи, ідентифікатори) не змінюється зловмисно або помилково [21]. Порушення цілісності можливе як на рівні клієнтських пристроїв (терміналів голосування), так і на рівні серверної інфраструктури. Наприклад, зловмисник може змінити прошивку або ввести шкідливий код у програмне забезпечення терміналу, що призведе до підміни голосу до моменту його надсилання [15]. На рівні серверів існує ризик підміни голосів або протоколів у базі даних.

Конфіденційність (приватність) гарантує, що ніхто не може дізнатися, як проголосував конкретний виборець. Це один із базових принципів демократичних виборів [21], [23]. Конфіденційність голосування також опиняється під загрозою через можливість перехоплення трафіку між терміналом і сервером або через збереження особистих ідентифікаторів виборця поряд з його голосом. Це суперечить базовій вимозі приватності вибору [17].

Також, одним із критичних аспектів є доступність. Електронні системи голосування можуть бути недоступними для виборців унаслідок DDoS-атак, відмови серверного або мережевого обладнання, чи недостатньої підготовки інфраструктури до навантажень. Це ставить під загрозу реалізацію виборчого права громадян і може стати об'єктом цілеспрямованих атак у день виборів [24].

Таким чином, ефективна модель загроз для систем електронного голосування повинна враховувати не лише суто технічні вектори атак, але й організаційні, людські та процедурні аспекти.

1.4 Модель зловмисника

Модель зловмисника (порушника) визначається як сукупність можливостей особи або групи осіб, які можуть отримати несанкціонований доступ до інформації в комп'ютерних системах [23].

НД ТЗІ 1.1-002-99 класифікує зловмисників за рівнями доступу, що дозволяє оцінити потенційні загрози та визначити необхідні заходи захисту. Згідно документу визначають такі рівні можливостей зловмисника:

Нульовий рівень. На цьому рівні особи, які не мають авторизованого доступу до системи, але можуть здійснювати фізичний вплив на обладнання або використовувати соціальну інженерію для отримання доступу [23].

Перший рівень. Це користувачі, які мають обмежений доступ до системи та можуть виконувати лише заздалегідь визначені функції без можливості змінювати налаштування або програмне забезпечення [23].

Другий рівень. Це користувачі з розширеними правами, які можуть створювати та запускати власні програми, що дозволяє їм впливати на обробку інформації в системі [23].

Третій рівень. До нього відносяться адміністратори або особи з повним доступом до системи, які можуть змінювати конфігурацію обладнання та програмного забезпечення, а також впливати на політику безпеки системи [23].

Четвертий рівень. Розробники або технічні спеціалісти, які мають можливість вносити зміни на рівні архітектури системи, включаючи створення або модифікацію апаратного та програмного забезпечення [23].

У контексті електронного голосування, особливу увагу слід приділити зловмисникам другого рівня, оскільки вони можуть створювати та запускати власні програми, що потенційно дозволяє їм змінювати або підробляти результати голосування. Це підкреслює необхідність впровадження ефективних механізмів контролю та моніторингу для виявлення та запобігання таким загрозам.

1.5 Перспективи розвитку систем децентралізованого електронного голосування і постановка задач розробки

Сучасні тенденції розвитку технологій цифрового урядування демонструють, що децентралізовані системи електронного голосування (ДЕГ) поступово переходять від експериментальних моделей до потенційно широкого практичного застосування. У міру вдосконалення блокчейн-платформ, криптографічних протоколів і цифрової ідентифікації відкриваються нові технічні можливості для реалізації безпечного, прозорого та масштабованого електронного голосування [7].

Перспективи розвитку ДЕГ-систем пов’язані з інтеграцією новітніх технологій:

- Постквантова криптографія — забезпечення стійкості до атак з боку квантових комп’ютерів, що з часом може стати критичним для довгострокової безпеки виборчих даних [25];
- Динамічні блокчейн-структури (sharding, DAG) — підвищення продуктивності та здатності обробляти мільйони транзакцій у короткий проміжок часу, що особливо важливо для загальнонаціональних виборів [26];
- Інтеграція з децентралізованими ідентифікаційними платформами (DID) — надання виборцям контролю над особистими даними без втрати юридичної валідності ідентифікації [25];
- Самоверифікація виборця — можливість переконатися у врахуванні голосу без ризику його розкриття або підміни [26].

Інший перспективний напрям — створення адаптивних гібридних моделей, які дозволяють поєднувати переваги децентралізації (розподілений підрахунок,

відкритий аудит) з легкістю інтеграції в існуючу інфраструктуру державного управління [27].

Окрему увагу також привертає можливість застосування ДЕГ-систем у подальших контекстах — наприклад, у внутрішньому корпоративному управлінні, профспілкових голосуваннях, електронних референдумах на муніципальному рівні [27].

1) Розробити багаторівневу архітектуру вузла, яка відокремлює функціональні компоненти за принципами розмежування відповідальності і забезпечує масштабованість та автономність.
2) Забезпечити гнучкість архітектури, яка дозволяє адаптувати вузол під різні сценарії голосування: централізоване, децентралізоване, публічне, анонімне — з урахуванням особливостей топології мережі та законодавчих вимог.
3) Впровадити перевірені криптографічні механізми, що забезпечують захист даних на всіх етапах обробки бюлетенів: шифрування, цифровий підпис, генерація ключів, автентифікація клієнтів.
4) Забезпечити захищений обмін повідомленнями між вузлами мережі через P2P-протоколи з верифікацією, уникненням повторних запитів і захистом від атак типу Sybil, Replay, Eclipse.
5) Реалізувати взаємодію вузла з мобільними клієнтами, що дозволяє приймати зашифровані бюлетені, відповідати на запити щодо статусу голосування та повертати агреговані результати.
6) Побудувати комплекс тестів для перевірки функціональних сценаріїв.

Таким чином, подальший розвиток децентралізованих систем голосування передбачає не лише вдосконалення технологій, а й вирішення широкого спектру прикладних задач, що забезпечують практичну реалізацію цих систем.

2 АНАЛІЗ ВИМОГ ДО ФУНКЦІОНУВАННЯ ВУЗЛІВ ДЕЦЕНТРАЛІЗОВАНОЇ СИСТЕМИ ЗАХИЩЕНОГО ГОЛОСУВАННЯ

2.1 Архітектурна роль ноди в системі захищеного голосування

У децентралізованих системах електронного голосування вузол (нода) є базовою одиницею інфраструктури, що виконує низку критичних функцій. Кожен вузол працює незалежно, але у взаємодії з іншими через peer-to-peer (P2P) мережу, яка забезпечує повну децентралізацію системи. Така архітектура дозволяє уникнути єдиної точки відмови та забезпечити стійкість системи навіть за наявності зовнішніх атак чи внутрішніх збоїв [25].

На відміну від централізованих моделей, у яких один сервер керує всіма транзакціями, P2P-архітектура дає змогу вузлам автономно обмінюватися даними, синхронізувати стан системи та брати участь у прийнятті рішень. Протоколи взаємодії вузлів часто ґрунтуються на динамічному обміні маршрутною інформацією, як це реалізовано у відомих епідемічних протоколах керування маршрутними таблицями [29].

Завдяки впровадженню блокчейн-технології, вузол отримує можливість зберігати зашифровані дані голосування у розподіленому реєстрі, забезпечуючи таким чином незмінність і прозорість виборчого процесу. Кожен запис у блокчейні підтверджується консенсусом між вузлами, що виключає можливість одностороннього втручання або підробки [25]. Архітектура вузла зазвичай включає кілька модулів: комунікаційний (для взаємодії з іншими вузлами), криптографічний (для підпису, шифрування і перевірки даних), обчислювальний (для формування блоків), а також інтерфейс взаємодії з клієнтами. Завдяки модульності, вузол може масштабуватись або адаптуватися до конкретних технічних умов [29].

Таким чином, вузол у системі децентралізованого електронного голосування — це не лише мережевий вузол, а багатофункціональний інтелектуальний агент, що забезпечує цілісність, безпеку, конфіденційність та стабільність виборчого процесу в цифровому середовищі.

До основних функцій вузла ноди можна віднести:

1) Підключення до мережі та взаємодія з іншими вузлами

У системах електронного голосування, що базуються на блокчейн-технологіях, вузли (ноди) функціонують у децентралізованій peer-to-peer (P2P) мережі, яка усуває необхідність у централізованому контролі та підвищує стійкість до зовнішніх атак [25]. Кожен вузол працює як рівноправний учасник, здатний передавати, приймати та перевіряти транзакції, пов’язані з процесом голосування. При цьому вузли не просто обмінюються даними, а й беруть участь у процесі досягнення консенсусу — узгодженого рішення щодо внесення записів у блокчейн. У цьому процесі можуть застосовуватись різні алгоритми, наприклад Proof-of-Work або Proof-of-Stake [28].

Оскільки кожен вузол зберігає повну копію розподіленого реєстру, змінити історію голосування можливо лише за умови досягнення консенсусу більшості учасників мережі [25]. Такий механізм забезпечує незмінність та прозорість голосування та гарантує достовірність результатів [8].

Реалізація ефективної взаємодії між вузлами потребує стійких мережевих з’єднань, синхронізації даних, криптографічного захисту переданої інформації, а також засобів виявлення й ізоляції недобросовісних учасників. Відомі протоколи (наприклад, епідемічні) дозволяють вузлам регулярно оновлювати інформацію про інших учасників мережі й адаптуватись до її динамічної топології [28].

Таким чином, підключення до мережі та ефективна взаємодія вузлів — це базова передумова для забезпечення децентралізованого, прозорого та безпечного процесу електронного голосування.

2) Обробка та зберігання бюлетенів

У децентралізованій системі електронного голосування вузли відповідають за обробку та збереження бюлетенів — ключовий етап, від якого залежить достовірність і захищеність результатів виборів. Ці функції повинні бути реалізовані з урахуванням високих вимог до цілісності, конфіденційності та прозорості процесу голосування [25].

Процес обробки бюлетенів починається з приймання бюлетеня від клієнтського додатка. Кожен голос одразу підлягає шифруванню, що забезпечує конфіденційність вибору виборця. Після цього бюлетень підписується цифровим підписом, який підтверджує автентичність і запобігає його зміні або підробці. Перш ніж бюлетень буде прийнятий до запису, вузол перевіряє його на унікальність і відповідність протоколу системи, щоб запобігти багаторазовому голосуванню (так званому "double voting") [25].

Після обробки, кожен бюлетень додається до блоку, який буде включений до ланцюга транзакцій за допомогою протоколу консенсусу. Блок після цього зберігається у локальній копії реєстру кожного вузла, а разом з ним — і в інших вузлах мережі. Таке децентралізоване зберігання виключає можливість централізованого втручання або видалення інформації. Цілісність забезпечується використанням хеш-функцій: будь-яка спроба змінити вміст блоку буде виявлена миттєво, оскільки порушить цілісність ланцюга [25].

Оскільки кожен вузол має повну копію блокчейну, система забезпечує високу надійність: навіть у разі виходу з ладу частини вузлів мережа продовжить роботу. Завдяки криптографічним протоколам та механізмам контролю цілісності, навіть за наявності атак або саботажу, змінити збережені голоси практично неможливо [8].

3) Забезпечення безпеки та конфіденційності

Вузли системи відіграють ключову роль у забезпеченні цілісності даних, захисті особистої інформації виборців та запобіганні потенційним загрозам.

Для забезпечення конфіденційності голосування широко використовуються сучасні криптографічні технології. Зокрема, гомоморфне шифрування дозволяє обробляти зашифровані голоси без їх розшифрування, що забезпечує анонімність виборців навіть під час підрахунку голосів. Окрім того, протоколи з нульовим розголошенням (Zero-Knowledge Proofs)

дозволяють підтвердити правильність транзакцій без розкриття їх змісту, зберігаючи конфіденційність [31].

У системах електронного голосування на основі блокчейну відсутній єдиний центр управління ключами. Замість цього, ключі шифрування можуть бути розподілені між кількома довіреними сторонами, такими як члени виборчої комісії. Це забезпечує додатковий рівень безпеки: навіть якщо один з учасників буде скомпрометований, повного доступу до даних він не отримає [30].

Побудова системи на основі блокчейну дозволяє досягти високого рівня стійкості до атак, включаючи DoS-атаки або спроби фальсифікації результатів. Усі вузли зберігають копію ланцюга блоків, тож навіть якщо окремі вузли виходять з ладу, система зберігає функціональність. Завдяки консенсусним механізмам, таким як Proof of Stake або інші fault-tolerant протоколи, вдається забезпечити узгодженість і безпеку реєстру [30].

Однією з основних переваг використання блокчейн-технологій є можливість відкритого аудиту системи без порушення конфіденційності виборців. Кожен голос фіксується як транзакція, що дозволяє незалежним спостерігачам перевіряти відповідність процесу та результатів без ризику ідентифікації конкретних виборців [32].

4) Підтримка консенсусу

Цей механізм гарантує, що всі учасники системи мають єдине бачення стану реєстру, що є необхідним для забезпечення цілісності та достовірності виборчого процесу.

PoW (Proof of Work) є одним із перших консенсусних алгоритмів, який використовується в блокчейн-системах. Він передбачає, що учасники мережі (майнери) вирішують складні математичні задачі для додавання нових блоків до ланцюга. Хоча PoW забезпечує високий рівень безпеки, його застосування в електронному голосуванні обмежене через значне споживання енергії та низьку швидкість обробки транзакцій, що може негативно вплинути на масштабованість системи [33].

PoS (Proof of Stake) є альтернативою PoW, де ймовірність створення нового блоку залежить від частки (stake), яку учасник володіє в системі. Цей підхід значно знижує енергоспоживання та підвищує ефективність, що робить його більш придатним для електронного голосування. Однак PoS може бути вразливим до атак, пов'язаних з концентрацією великої частки у невеликої кількості учасників, що може призвести до централізації управління [33].

PBFT (Practical Byzantine Fault Tolerance) є консенсусним алгоритмом, розробленим для систем з обмеженою кількістю довірених вузлів. Він забезпечує високу швидкість обробки транзакцій та стійкість до зловмисних дій до певного порогу. PBFT особливо підходить для приватних або консорціумних блокчейн-систем, де учасники відомі та довірені. Однак його ефективність знижується зі зростанням кількості учасників, що обмежує його застосування в масштабних публічних виборах [33].

PoV (Proof of Vote) є новим консенсусним алгоритмом, спеціально розробленим для систем електронного голосування. У цьому підході вузли мережі досягають консенсусу шляхом голосування, що відображає саму суть виборчого процесу. PoV дозволяє розділити права на голосування та ведення обліку, забезпечуючи гнучкість та безпеку системи. Цей алгоритм особливо підходить для консорціумних блокчейн-систем, де учасники мають різні ролі та рівні довіри [34].

Вибір відповідного консенсусного алгоритму залежить від специфіки виборчої системи, включаючи її масштаб, рівень довіри між учасниками та вимоги до безпеки та ефективності. Для публічних виборів з великою кількістю учасників можуть бути придатні PoS або PoV, які забезпечують баланс між безпекою та масштабованістю. У приватних або консорціумних системах, де учасники відомі та довірені, PBFT може забезпечити високу швидкість та надійність обробки транзакцій.

5) Обробка запитів від клієнтів і формування відповідей

Обробка запитів і формування відповідей — це ключова функція, яка гарантує прозорість, контроль та зворотний зв’язок між виборцем і системою [25]. До основних компонентів функції належать:

- Приймання запитів. Вузол приймає запити від мобільних клієнтів — наприклад, перевірку статусу бюлетеня, підтвердження участі або отримання результатів. Запити передаються за захищеними каналами (наприклад, TLS), а дані у них мають бути автентифіковані [8].
- Валідація. Кожен запит проходить внутрішню перевірку щодо автентичності, цілісності та відповідності встановленим правилам протоколу. Це унеможливлює зловживання з боку неавторизованих користувачів [25].
- Формування відповіді. На основі запиту, вузол формує відповідь, яка може містити криптографічно підтверджену інформацію, наприклад, хеш бюлетеня або агрегований результат голосування. Ці відповіді можуть бути використані як доказ участі або перевірки системою аудиту [8].
- Обробка великої кількості запитів. Оскільки під час виборів навантаження на систему значне, вузол повинен мати архітектурні рішення, які дозволяють масштабувати обробку запитів — наприклад, черги повідомлень, асинхронні механізми тощо [35].
- Конфіденційність. При передачі даних між вузлом і клієнтом використовується шифрування, а сама структура відповіді не повинна містити ідентифікаторів, які можуть порушити анонімність виборця [19].

2.2 Взаємодія з клієнтами та іншими вузлами

Однією з основних функцій вузла у децентралізованій системі електронного голосування є забезпечення надійної та безпечної взаємодії як з клієнтськими пристроями (мобільними додатками, вебінтерфейсами), так і з іншими вузлами мережі. Ця взаємодія критично важлива для підтримання узгодженості стану

блокчейну, забезпечення доступності сервісу та конфіденційного обміну інформацією між усіма учасниками системи [36].

Серед основних процесів взаємодії з клієнтами виділяють:

- Генерація та передача бюлетенів. Виборець у застосунку формує вибір, який шифрується на боці клієнта з використанням гомоморфного або асиметричного шифрування [36]. Після цього бюлетень передається до вузла через захищений канал передачі даних (TLS/SSL). Це дозволяє гарантувати, що навіть при перехопленні трафіку дані залишаються недоступними для сторонніх осіб [8].
- Автентифікація користувача. У більшості систем автентифікація здійснюється через цифровий підпис або інші надійні ідентифікаційні механізми, як-от ID-карти або BankID. Це дозволяє унеможливити багаторазове голосування або спроби фальсифікацій [38].
- Підтвердження успішної передачі. Після успішної передачі бюлетеня вузол надсилає підтвердження, яке також може бути підписане криптографічно — це слугує доказом участі виборця у голосуванні. Таке підтвердження може бути використане у процедурах аудиту або самоперевірки з боку користувача [8].
- Можливість перевірки. Деякі системи дозволяють виборцю переконатися, що його голос був зафіксований у блокчейні, не розкриваючи при цьому змісту вибору. Це досягається за допомогою криптографічних хешів або zero-knowledge proof [38].
- Доступність та захист каналів. У разі відмови або атак на інфраструктуру клієнти мають отримати доступ до альтернативних вузлів через систему автоматичного вибору найближчого/активного сервера. Система має бути захищена від DoS-атак і мати балансування навантаження [19].

У блокчейн-архітектурі систем електронного голосування вузли (ноди) взаємодіють між собою в межах однорангової (peer-to-peer, P2P) мережі. Ця взаємодія забезпечує децентралізоване зберігання та обробку даних, що підвищує надійність та безпеку системи. Тож, виділяють основні аспекти взаємодії вузлів:

- Обмін транзакціями: Кожен вузол отримує та передає транзакції (наприклад, зашифровані голоси) іншим вузлам у мережі. Це забезпечує розповсюдження інформації та запобігає втраті даних у разі відмови окремих вузлів [25].
- Досягнення консенсусу: Вузли використовують механізми консенсусу для узгодження стану блокчейну. У контексті електронного голосування часто застосовуються алгоритми, що забезпечують швидке та енергоефективне досягнення згоди, наприклад, Proof of Authority (PoA) або Practical Byzantine Fault Tolerance (PBFT) [25].
- Синхронізація блоків: Вузли регулярно обмінюються інформацією про нові блоки, щоб підтримувати актуальний стан блокчейну. Це гарантує, що всі вузли мають однакову версію реєстру, що є критично важливим для цілісності виборчого процесу [39].
- Захист від атак: Децентралізована природа мережі ускладнює здійснення атак, таких як відмова в обслуговуванні (DoS) або маніпуляції з даними. Кожен вузол перевіряє достовірність отриманих даних, що підвищує загальну безпеку системи [25].

2.3 Вимоги до криптографії, мережі та безпеки

Як відомо, системи децентралізованого електронного голосування мають справу з критично важливою інформацією і тому до програмного забезпечення вузла, як одного з ключових компонентів такої системи, висуваються високі вимоги щодо захищеності. Основні з них стосуються криптографічного захисту даних, мережевої стійкості та загальної кібербезпеки.

Криптографічні механізми відіграють вирішальну роль у забезпеченні конфіденційності та цілісності виборчого процесу у системах електронного голосування, що базуються на блокчейн-технологіях. Основні криптографічні вимоги до вузлів таких систем включають:

1) Шифрування бюлетенів. Для забезпечення конфіденційності голосування використовуються методи шифрування, які дозволяють зберігати та обробляти зашифровані бюлетені без розкриття їхнього вмісту [25].

2) Автентифікація виборців. Система повинна гарантувати, що лише легітимні виборці можуть брати участь у голосуванні, і кожен з них має можливість проголосувати лише один раз. Це досягається через використання асиметричної криптографії та цифрових підписів, які дають змогу перевіряти справжність джерела запиту без розкриття персональних даних [25].

3) Приватність та незаперечність. Важливим критерієм є гарантування приватності виборця — неможливість пов’язати голос із конкретною особою, навіть для адміністраторів системи. Для цього можуть застосовуватись кільцеві підписи або протоколи нульового розголошення знань (Zero-Knowledge Proof) [25].

4) Захист цілісності даних. Блокчейн як технологія за замовчуванням реалізує контроль цілісності даних за допомогою хеш-функцій. Кожен блок має унікальний криптографічний хеш, і будь-яке порушення структури даних призводить до негайного виявлення змін [25].

Мережеві вимоги до вузлів систем електронного голосування на основі блокчейн-технологій спрямовані на забезпечення надійної, безпечної та ефективної передачі даних між учасниками системи. Ці вимоги включають:

1) Надійність та доступність мережі. Надійність та доступність мережі є критично важливими для забезпечення безперервного функціонування систем електронного голосування. Системи повинні бути спроектовані таким чином, щоб витримувати високі навантаження та забезпечувати безперервний доступ виборців до платформи голосування. Це включає в себе використання резервних серверів, балансування навантаження та механізмів автоматичного відновлення після збоїв [40].

2) Захист каналів зв'язку. Захист каналів зв'язку є необхідним для забезпечення конфіденційності та цілісності переданих даних. Використання криптографічних протоколів, таких як TLS (Transport Layer Security), дозволяє шифрувати дані під час передачі, запобігаючи їх перехопленню або модифікації [41].

3) Стійкість до мережевих атак. Системи електронного голосування повинні бути стійкими до різноманітних мережевих атак, таких як атаки типу "відмова в обслуговуванні" (DoS), "людина посередині" (MITM) та повторні атаки (replay attacks). Для цього необхідно впроваджувати механізми виявлення та запобігання таким атакам, а також регулярно оновлювати та патчити системне програмне забезпечення [42].

Механізми безпеки повинні захищати від зовнішніх і внутрішніх загроз, запобігати спробам фальсифікацій та гарантувати захист персональних даних і волевиявлення виборців [41].

1) Захист від примусу та продажу голосів. Електронні системи голосування повинні забезпечувати механізми, що унеможливлюють примус виборців або продаж голосів. Це досягається шляхом впровадження криптографічних протоколів, які не дозволяють виборцю довести, як саме він проголосував, навіть якщо він цього бажає [43].

2) Протидія фішингу та соціальній інженерії. Системи електронного голосування повинні бути стійкими до атак, спрямованих на обман користувачів з метою отримання їхніх облікових даних або впливу на їхнє голосування. Це включає впровадження механізмів виявлення та запобігання фішинговим атакам, а також навчання користувачів розпізнаванню підозрілих повідомлень та сайтів [41].

3) Забезпечення цілісності голосів. Для гарантування, що голоси не можуть бути змінені, видалені або додані без виявлення, необхідно використовувати криптографічні хеш-функції та цифрові підписи. Ці методи дозволяють виявити будь-які несанкціоновані зміни в даних голосування [41].

4) Аудит та перевірка результатів. Системи повинні підтримувати можливість незалежного аудиту та перевірки результатів голосування. Це може включати використання паперових копій голосів або інших механізмів, що дозволяють перевірити правильність підрахунку голосів без розкриття конфіденційної інформації [44].

5) Стійкість до технічних збоїв та відновлення після атак. Системи електронного голосування повинні мати механізми для виявлення та

реагування на технічні збої або кібератаки. Це включає регулярне тестування системи, впровадження резервних копій та планів відновлення після збоїв, а також моніторинг системи в режимі реального часу для виявлення аномальної активності [42].

2.4 Узагальнення вимог до проектування програмного забезпечення вузла

Розробка програмного забезпечення вузла децентралізованої системи електронного голосування потребує системного підходу, який об'єднує функціональні, криптографічні, мережеві та безпекові аспекти. Такий вузол виступає центральним елементом обробки голосів, забезпечуючи їхнє безпечне приймання, зберігання, перевірку та передачу до мережі блокчейну.

Функціональна складова системи повинна реалізовувати всі основні етапи виборчого процесу: від автентифікації виборця до шифрування бюлетеня й участі у консенсусі. Це вимагає чітко структурованої архітектури, здатної до масштабування, адаптації до навантажень і взаємодії з іншими вузлами та клієнтами.

Особливу роль у проектуванні відіграють вимоги до безпеки. Програмне забезпечення повинно гарантувати цілісність і конфіденційність голосів, а також стійкість до атак, зокрема DoS, MITM і внутрішніх загроз. Для цього впроваджуються цифрові підписи, гомоморфне шифрування, протокол TLS, механізми багатофакторної автентифікації тощо.

З погляду мережевої організації, вузол повинен підтримувати стабільний обмін повідомленнями з іншими вузлами та клієнтами, бути здатним до автоматичного відновлення після збоїв та забезпечувати високу доступність.

Окрім технічних аспектів, проектування повинно відповідати правовим нормам щодо захисту персональних даних та прозорості виборчого процесу, а також враховувати вимоги до довіри виборців — тобто мати відкритий, перевірюваний механізм голосування, не порушуючи при цьому приватності.

3 ТЕХНОЛОГІЇ ПОБУДОВИ ВУЗЛІВ БЛОКЧЕЙН-МЕРЕЖІ ДЕЦЕНТРАЛІЗОВАНОЇ СИСТЕМИ ЗАХИЩЕНОГО ГОЛОСУВАННЯ

3.1 Архітектурні підходи до побудови вузлів блокчейн-систем

Розробка вузлів блокчейн-мережі для децентралізованих систем захищеного голосування вимагає ретельного вибору архітектурного підходу. Розглянемо три основні архітектурні моделі: монолітну, мікросервісну та серверлес.

У монолітній архітектурі всі функціональні компоненти системи — обробка транзакцій, консенсус, зберігання даних та мережевий інтерфейс — інтегровані в єдину програмну структуру. Такий підхід забезпечує високу узгодженість та спрощує розгортання, але обмежує масштабованість і гнучкість системи [46]. Наприклад, монолітні блокчейни, такі як Bitcoin, обробляють всі функції на одному рівні, що може призводити до проблем з масштабуванням при збільшенні обсягу транзакцій [46].

Мікросервісна архітектура передбачає розділення системи на незалежні сервіси, кожен з яких відповідає за окрему функціональність. Це дозволяє досягти високої гнучкості, спрощує оновлення окремих компонентів та покращує масштабованість системи. Впровадження мікросервісної архітектури в блокчейн-системах, дозволяє ефективно управляти складністю та забезпечувати гнучкість при розробці та розгортанні додатків [47].

Безсерверна (serverless) архітектура дозволяє розробникам зосередитися на логіці додатку, не турбуючись про управління інфраструктурою. Функції виконуються в хмарному середовищі за подієвим принципом, що забезпечує високу масштабованість та ефективність. Інтеграція блокчейн-технологій з безсерверною архітектурою, дозволяє зменшити зусилля на управління інфраструктурою та забезпечити ефективне виконання транзакцій [48].

Також, варто приділити увагу і особливості реалізації децентралізованих компонентів. Децентралізовані блокчейн-системи функціонують завдяки взаємодії кількох ключових компонентів, кожен з яких забезпечує автономність, безпеку та стійкість мережі. Основні з них:

1) Механізм консенсусу. Для досягнення згоди щодо стану реєстру між усіма вузлами використовуються механізми консенсусу, такі як Proof of Work (PoW) або Proof of Stake (PoS). Ці алгоритми дозволяють забезпечити цілісність даних та запобігти подвійним витратам без потреби в центральному органі управління [49].

2) Розподілений реєстр (Distributed Ledger). Кожен вузол зберігає копію реєстру, що містить усі транзакції мережі. Це забезпечує прозорість та неможливість зміни даних без згоди більшості учасників. Розподілений характер реєстру гарантує, що навіть при виході з ладу окремих вузлів, мережа продовжує функціонувати коректно [50].

3) Смарт-контракти. Це програми, що автоматично виконують умови договорів при настанні певних подій. Вони дозволяють реалізовувати складні логіки взаємодії між учасниками мережі без посередників, підвищуючи ефективність та знижуючи витрати [50].

4) Peer-to-Peer (P2P) мережа. P2P-мережа забезпечує прямий обмін даними між вузлами без центрального сервера. Це дозволяє кожному учаснику мережі (вузлу) приймати та передавати інформацію, сприяючи децентралізації та стійкості системи до збоїв. Такий підхід зменшує ризик централізованих точок відмови та покращує масштабованість мережі [51].

5) Відкритий вихідний код. Більшість децентралізованих блокчейн-платформ мають відкритий вихідний код, що дозволяє спільноті перевіряти, вдосконалювати та адаптувати систему до нових вимог. Це сприяє прозорості, безпеці та інноваціям у розвитку технології [51].

Ефективне функціонування блокчейн-мережі також залежить від здатності її вузлів масштабуватися, адаптуватися до змін та діяти автономно. Ці характеристики є критичними для забезпечення надійності, продуктивності та стійкості системи.

Масштабованість визначає здатність блокчейн-системи обробляти зростаючий обсяг транзакцій без зниження продуктивності. Основні підходи до досягнення масштабованості включають:

- Шардінг - розділення мережі на менші частини (шарди), кожна з яких обробляє власний набір транзакцій, що дозволяє паралельну обробку та зменшує навантаження на окремі вузли [52].
- Модульна архітектура - розділення функціональності блокчейну на окремі модулі (наприклад, консенсус, виконання транзакцій, зберігання даних), що дозволяє масштабувати кожен компонент незалежно [53].
- Рівень 2 (Layer 2) рішення - використання додаткових протоколів поверх основного блокчейну для обробки транзакцій поза основним ланцюгом, з подальшим записом результатів у головний блокчейн [54].

Гнучкість вузлів полягає у здатності адаптуватися до змін у мережі, включаючи оновлення протоколів, зміни в обсязі транзакцій та інтеграцію з іншими системами. Це досягається через:

- Модульність - можливість оновлення або заміни окремих компонентів системи без впливу на інші частини [53].
- Інтероперабельність - здатність взаємодіяти з іншими блокчейн-мережами та традиційними системами, що забезпечує ширшу функціональність та прийняття технології [50].

Автономність вузлів означає їх здатність функціонувати незалежно, без централізованого контролю, що є основою децентралізованих систем. Це включає:

- Локальне зберігання даних - кожен вузол зберігає повну або часткову копію реєстру, що забезпечує стійкість до збоїв та атак [55].
- Самостійність прийняття рішень - вузли можуть самостійно перевіряти та підтверджувати транзакції відповідно до встановлених протоколів, без потреби в центральному органі [56].

3.2 Інструменти та технології реалізації програмного забезпечення вузла

У розробці блокчейн-систем використовується широкий спектр мов програмування, серед яких Go, Rust, Python, JavaScript та Java. Кожна з них має свої переваги. Go відрізняється простотою та високою продуктивністю, Rust — безпечністю та контролем пам’яті, Python — зручністю для швидкого

прототипування, JavaScript — через підтримку веб орієнтованих бібліотек типу web3.js або ethers.js.

Java, у свою чергу, залишається одним із найбільш стабільних і перевірених інструментів, особливо в корпоративному середовищі. Завдяки об’єктно-орієнтованому підходу, платформній незалежності та великій екосистемі, Java є ефективною для реалізації складних інфраструктурних рішень, включно з вузлами блокчейн-мереж [57].

У нашому випадку, для реалізації програмного забезпечення вузла децентралізованої системи захищеного голосування, обрано Java з використанням фреймворку Spring Boot. Це пояснюється такими перевагами:

- Платформна незалежність. Java працює на будь-якій системі, де є JVM, що дозволяє легко запускати вузли на різних платформах [57];
- Широка підтримка інструментів і бібліотек. Наприклад, web3j дозволяє зручно взаємодіяти з Ethereum;
- Сумісність з корпоративними рішеннями. Java підтримується в середовищах, де критично важливі стабільність і масштабованість;
- Підтримка в блокчейн-проектах. Такі системи, як Hyperledger Fabric, мають SDK для Java, що підтверджує її актуальність у даній сфері [50].

Таким чином, вибір Java є виваженим і базується як на її технічних можливостях, так і на практичному досвіді застосування в існуючих блокчейн-проектах.

Зберігання даних також відіграє ключову роль у забезпеченні надійності, доступності та масштабованості. Для реалізації системи захищеного голосування було обрано реляційну базу даних PostgreSQL, яка поєднує в собі потужні можливості зберігання, гнучкість та активну підтримку спільноти.

PostgreSQL — це об'єктно-реляційна система керування базами даних з відкритим кодом, яка забезпечує високу продуктивність, розширюваність та відповідність стандартам SQL. Вона підтримує складні типи даних, транзакції, багаторівневу ізоляцію та механізми реплікації, що робить її придатною для критично важливих систем [58]. Основні її переваги включають:

- Відповідність стандартам SQL та розширюваність. PostgreSQL підтримує повний набір SQL-операцій та дозволяє користувачам створювати власні типи даних, функції, агрегати та індекси, що робить її надзвичайно адаптивною до потреб складних систем [58].
- Транзакційність та надійність. Повна підтримка транзакцій (ACID), вбудовані механізми журналювання (WAL) та реплікація дозволяють будувати надійні та відмовостійкі рішення [58].
- Гнучка модель зберігання даних. Система підтримує як реляційні, так і напівструктуровані дані, включаючи JSON, XML, масиви, що полегшує інтеграцію з сучасними веб-додатками [59].
- Безпека. PostgreSQL має розвинені засоби контролю доступу, шифрування, автентифікації та аудитів, що робить її придатною для роботи в захищених середовищах [60].
- Масштабованість і продуктивність. PostgreSQL добре працює як на малих, так і на великих проєктах, підтримуючи індексацію, паралельні запити та розподілені конфігурації [60].

У поєднанні з фреймворком Spring Boot, PostgreSQL легко інтегрується в архітектуру мікросервісів та забезпечує стабільну основу для зберігання критичних даних у децентралізованих системах.

Також, для підвищення продуктивності системи важливо впровадити механізми кешування. Використання кешу другого рівня, наприклад, за допомогою Redis або вбудованих механізмів Spring Boot, дозволяє зменшити навантаження на базу даних та прискорити доступ до часто використовуваних даних [59].

3.3 Механізми взаємодії вузлів у мережі

Взаємодія між вузлами забезпечується за допомогою різноманітних мережевих протоколів, які дозволяють обмінюватися даними без централізованого посередника. Серед таких протоколів важливу роль відіграють як сучасні RPC-фреймворки, зокрема gRPC, так і класичні peer-to-peer (P2P) протоколи.

gRPC, розроблений Google, використовує HTTP/2 як транспортний рівень і Protocol Buffers для серіалізації даних, що дозволяє забезпечити високу швидкодію та ефективну взаємодію між вузлами в розподіленій архітектурі. Завдяки підтримці потокової передачі, автентифікації та стиснення, gRPC широко застосовується у проектах з мікросервісною архітектурою та в системах, які вимагають масштабованої синхронізації вузлів [61].

З іншого боку, P2P-протоколи дозволяють створити мережу без єдиного центру управління. Наприклад, Kademlia є розподіленою хеш-таблицею (DHT), яка підтримує ефективну маршрутизацію запитів із логарифмічною складністю відносно кількості вузлів у мережі. Вузли у такій мережі самостійно виконують функції зберігання, обробки та пересилання даних, що підвищує стійкість системи до збоїв [62]. Ранні реалізації, такі як Gnutella, не мали структури DHT, але продемонстрували фундаментальну можливість обміну даними без централізованих серверів, використовуючи протокол flooding [63].

Окрему роль у сучасних системах відіграє IPFS (InterPlanetary File System) — протокол, який поєднує P2P-комунікацію з контент-адресованим зберіганням файлів. IPFS дозволяє організувати розподілене зберігання великих об’ємів даних, підтримуючи реплікацію та ефективний пошук за унікальними хешами [64].

Таким чином, поєднання gRPC з P2P-протоколами дозволяє проектувати гнучкі архітектури для електронного голосування, в яких досягається баланс між структурованим обміном повідомленнями та гнучкою маршрутизацією даних між вузлами.

Також, слід мати на увазі і безпеку мережі, тому що вона є критично важливою. Серед основних загроз виділяють атаки типу Sybil, Replay та Eclipse, які можуть серйозно порушити цілісність і надійність системи.

Атака Sybil передбачає створення зловмисником великої кількості фальшивих ідентичностей для отримання непропорційного впливу на мережу. Це може призвести до маніпуляцій у процесах голосування, маршрутизації та розподілу ресурсів. Для протидії таким атакам використовуються методи перевірки ресурсів, такі як тестування радіочастотного спектру та валідація ключів. Ці методи дозволяють виявити та обмежити вплив фальшивих вузлів у мережі [65].

Атака Replay полягає у повторному відтворенні зловмисником раніше перехоплених дійсних повідомлень для обману системи. Це може призвести до несанкціонованого доступу або повторного виконання транзакцій. Для запобігання таким атакам впроваджуються механізми перевірки унікальності повідомлень, використання одноразових токенів та часових міток, що забезпечує автентичність і актуальність переданих даних [66].

Атака Eclipse спрямована на ізоляцію вузла від решти мережі шляхом контролю над його з'єднаннями. Це дозволяє зловмиснику маніпулювати інформацією, яку отримує ізольований вузол, що може призвести до прийняття ним неправильних рішень. Для виявлення та запобігання таким атакам застосовуються методи аналізу мережевого трафіку, виявлення аномалій у поведінці вузлів та використання алгоритмів машинного навчання для ідентифікації підозрілих патернів [67].

Захищена ідентифікація користувачів і конфіденційність переданих даних є основою надійної взаємодії. Автентифікація забезпечує перевірку легітимності учасників, а шифрування — захист інформації від несанкціонованого доступу.

У децентралізованих архітектурах зростає значення моделей, які не покладаються на централізовані сховища облікових даних. З цією метою використовуються децентралізовані ідентифікатори (DID) та перевірені облікові дані (VC), що дозволяє користувачам контролювати власну ідентичність без централізованого посередника [68].

Шифрування, зокрема симетричне (AES) та асиметричне (RSA, ECC ElGamal), використовується для захисту переданих повідомлень і створення цифрових підписів, що гарантує автентичність та цілісність даних [69]. У розподілених голосувальних системах шифрування відіграє критичну роль у збереженні приватності бюлетенів і результатів.

Окрему роль у реалізації безпечної логіки обробки даних в рамках проектів на Java відіграє фреймворк Spring Security. Цей інструмент дозволяє реалізувати гнучкі та розширювані політики доступу, а також підтримує сучасні механізми автентифікації, включаючи OAuth2 та JWT [70]. Крім того, модуль Spring Security

Crypto забезпечує підтримку симетричного шифрування, генерацію ключів, хешування паролів та захист чутливої інформації у збереженому вигляді [70].

3.4 Узагальнення вибору технологій та обґрунтування рішень

У цьому розділі було здійснено комплексний аналіз архітектурних підходів, інструментів реалізації та технічних механізмів, необхідних для побудови вузла блокчейн-мережі в децентралізованій системі захищеного голосування. Розглянуто сучасні архітектурні моделі — від монолітних до мікросервісних та серверлесс-підходів — із акцентом на гнучкість, масштабованість і незалежність вузлів, що є критичними у системах електронного голосування.

Проведено огляд мов програмування та супутніх інструментів, із пріоритетом на Java та фреймворк Spring Boot як основу для створення безпечного серверного компонента вузла. Вибір PostgreSQL як системи управління базами даних пояснюється її надійністю, підтримкою транзакцій і відповідністю вимогам до обробки чутливої інформації.

У межах вивчення механізмів взаємодії вузлів проаналізовано мережеві протоколи (gRPC, P2P), консенсусні алгоритми (PoW, PoS, PBFT), а також засоби протидії ключовим атакам у децентралізованому середовищі. Окрема увага приділена забезпеченню безпеки шляхом шифрування та реалізації автентифікації з використанням сучасних рішень, зокрема Spring Security.

Загалом, можна сказати, що обрані технології відповідають вимогам до побудови безпечної, розширюваної та надійної архітектури вузла в системі електронного голосування, забезпечуючи баланс між функціональністю, продуктивністю та стійкістю до загроз.

4 РОЗРОБКА АРХІТЕКТУРИ І ПОБУДОВА ВУЗЛА БЛОКЧЕЙН-МЕРЕЖІ ДЕЦЕНТРАЛІЗОВАНОЇ СИСТЕМИ ЗАХИЩЕНОГО ГОЛОСУВАННЯ

4.1 Загальна архітектура і конфігурація системи

Архітектура програмного забезпечення вузла побудована за принципами багаторівневої моделі з чітким поділом відповідальностей. Основними функціональними компонентами вузла є:

- REST-контролери — приймають та обробляють HTTP-запити від мобільних клієнтів та інших вузлів.
- Сервіси — реалізують бізнес-логіку, зокрема обробку бюлетенів, перевірку підписів, збереження даних тощо.
- Репозиторії — взаємодіють з базою даних через Spring Data JPA.
- Криптографічні компоненти — відповідають за генерацію, збереження та використання пари криптографічних ключів.
- Мережеві компоненти — забезпечують з’єднання з іншими вузлами, надсилання та отримання повідомлень.

Загальна архітектура проекту наведена на рисунку 4.1.

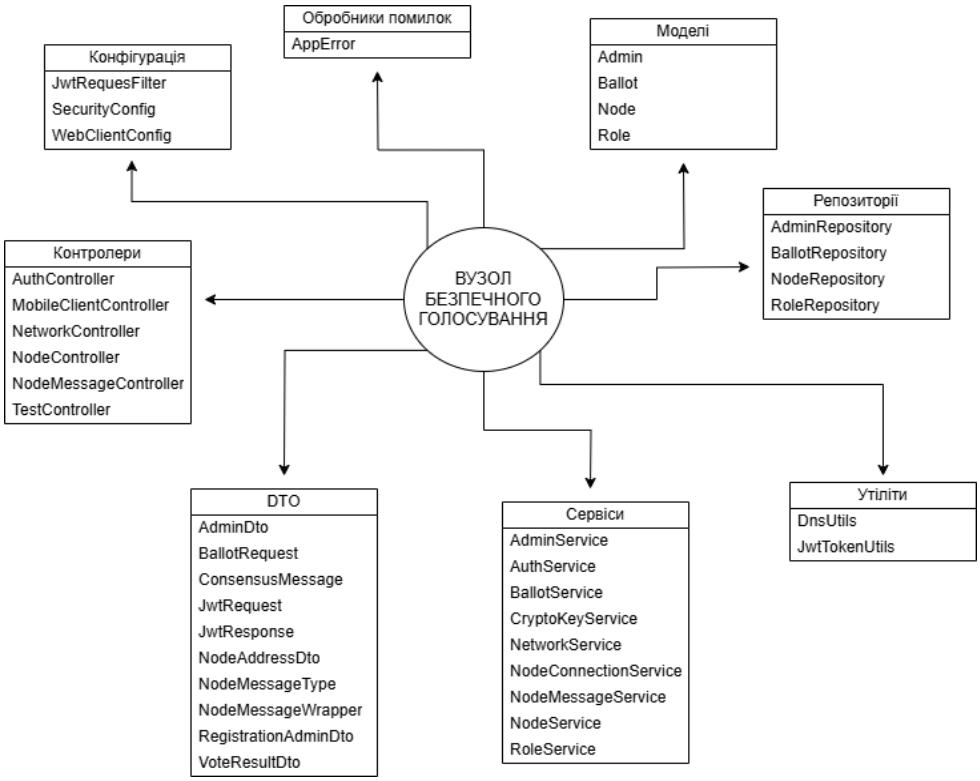

Рисунок 4.1 - Загальна архітектура проекту.

Система побудована на принципі поділу відповідальностей, де кожен компонент виконує свою роль і взаємодіє з іншими через визначені інтерфейси. Взаємодія компонентів виконується наступним чином:

- Адміністраторська частина відповідає за реєстрацію адміністраторів, автентифікацію через JWT і управління правами доступу.
- Мережева частина встановлює з’єднання з іншими вузлами, підтримує їхню доступність і забезпечує обмін повідомленнями (включно з консенсусом).
- Клієнтська частина приймає голоси від мобільних клієнтів, перевіряє цифрові підписи і зберігає дані до бази.
- Криптографічний модуль забезпечує генерацію ключів, шифрування/розшифрування голосів і перевірку підписів.
- Обробка результатів включає дешифрування збережених голосів, підрахунок результатів і видачу їх за запитом.

Всі критичні параметри системи винесені в конфігураційний файл “application.properties”, що спрощує розгортання та масштабування. З прикладом вмісту файлу можна ознайомитися у додатку А, файл applicatio

У межах архітектури вузла реалізовано окремі конфігураційні класи, що відповідають за безпеку, обробку запитів та мережеву взаємодію з іншими вузлами:

1) SecurityConfig - це клас який відповідає за визначення правил доступу до API та інтеграцію JWT-автентифікації. В нашому випадку:

- Вимикається CSRF-захист — оскільки застосунок є REST-сервісом, де клієнт не використовує cookie.
- Обмежується доступ до ендпоінтів ролями:
  - /auth/** — відкриті для всіх (реєстрація, авторизація).
  - /admin/** — доступні лише адміністраторам (ROLE_ADMIN).
  - /super-admin/** — лише для супер-адміністраторів (ROLE_SUPER_ADMIN).
  - Інші запити дозволені без обмежень.
- Політика сесій — STATELESS: сервер не зберігає стан, автентифікація відбувається на основі JWT.

- JWT-фільтр (jwtRequestFilter) додається до ланцюжка фільтрів до UsernamePasswordAuthenticationFilter, що дозволяє перевірити токен до стандартної обробки запиту.
- Використовується HttpStatus.UNAUTHORIZED, якщо користувач не автентифікований.

2) JwtRequestFilter - цей фільтр виконує перевірку JWT-токена у кожному запиті, що потребує автентифікації. Його головні функції:

- Отримання JWT-токена з заголовку Authorization. Якщо заголовок починається з Bearer , з нього виділяється токен.
- Перевірка дійсності токена та витягування email і ролей. Якщо токен не протермінований та має правильний підпис:
- Формування UsernamePasswordAuthenticationToken (об’єкта автентифікації) і збереження його в SecurityContextHolder, щоб далі у контролерах була доступна інформація про авторизованого користувача.
- Обробка виключень. У випадку якщо підпис неправильний або термін дії токена вичерпано, це фіксується в логах.

3) WebClientConfig - це клас який конфігурує компонент WebClient, що використовується для асинхронної взаємодії вузла з іншими вузлами. Він створює загальний WebClient.Builder, який використовується у далі сервісах (наприклад NodeConnectionService, CatalogNodeClient).

4.2 Управління адміністраторами вузла

У системі реалізовано модель доступу на основі ролей (RBAC) із двома рівнями:

- ROLE_ADMIN – звичайний адміністратор вузла;
- ROLE_SUPER_ADMIN – адміністратор із розширеними правами (наприклад, для оновлення конфігурації або модерації мережі вузлів).

Це дозволяє реалізувати розмежування прав доступу до різних API-ендпоінтів.

Дані про адміністраторів зберігаються у система використовуючи три таблиці (Рисунок 4.2):

- admin — зберігає облікові записи;
- role — довідник ролей;
- admin_roles — зв’язок між користувачами та ролями (many-to-many).

При реєстрації нового адміністратора створюється обліковий запис, пароль хешується, і присвоюється роль за замовчуванням (ROLE_ADMIN). Реєстрація доступна через ендпоінт: “POST /auth/register ”. Запит містить email та пароль, після чого повертається JWT-токен. Приклад токену можна побачити на рисунку 4.2.

Рисунок 4.2 - Приклад JWT-токену.

Вхід до системи адміністратора реалізований через JWT. Токен додається до заголовка “Authorization: Bearer … ” і автоматично обробляється фільтром JwtRequestFilter.

Окрім управління адміністраторами через базу даних та розмежування ролей, треба реалізувати повноцінний сервіс для автентифікації та реєстрації адміністраторів вузла AuthService. Це сервіс обробки логіки автентифікації. Клас AuthService виконує два основні завдання:

- Автентифікація адміністратора. Метод createAuthToken() перевіряє надані облікові дані через AuthenticationManager. У разі успішної перевірки генерується JWT-токен за допомогою JwtTokenUtils.
- Реєстрація нового адміністратора. Метод createNewAdmin() приймає DTO з інформацією про нового користувача, перевіряє правильність паролів і унікальність електронної пошти, після чого створює нового адміністратора у системі.

З прикладом реалізації цих методом можна ознайомитися у додатку А, клас AuthService. Для того щоб взаємодіяти з цими методами, треба обробити два ендпоінти:

– “POST /auth” – автентифікація адміністратора і видача JWT токена;
– “POST /register” – реєстрація нового адміністратора.

4.3 Взаємодія з мережею: підключення до вузлів і отримання топології

Встановлення з’єднань з іншими вузлами мережі є дуже важливим для забезпечення обміну інформацією (бюлетенями, повідомленнями консенсусу тощо). У розробленій системі необхідно реалізувати механізм автоматичного підключення до вузлів і підтримання активної топології.

При старті вузол звертається до централізованого DNS-сервера-каталога, який надає список відомих вузлів (NodeAddressDto). З’єднання встановлюються з кількома випадковими вузлами з цього списку (клас NodeConnetcionService у додатку А), де ількість з’єднань у межах заданих параметрів.

Для кожного вузла створюється об’єкт WebClient, що зберігається у ConcurrentHashMap<String, WebClient> — це дозволяє відправляти повідомлення іншим вузлам. Для кожного віддаленого вузла виконується “GET /ping” для перевірки доступності.

Також необхідно періодично перевіряти з’єднання та оновлюваться. Для цього ми робимо метод refreshConnetcion() з анотацією @Scheduled(fixedDelayString = "${node.connections.refresh-limit}") який буде виконуватися з деякою періодичною яку ми зазначимо у файлі з налаштуваннями (application.properties).

4.4 Взаємодія з мобільними клієнтами

Дуже важливим також є забезпечення зручної та захищеної взаємодії з мобільними клієнтами. Саме клієнти ініціюють голосування, надсилаючи бюлетені, а також запити на отримання результатів. Необхідно приймати від користувача зашифрований бюлетень у форматі JSON на визначену адресу вузла.

На стороні вузла в нас буде працювати контролер MobileClientController, який обробляє цей запит. Для цього тут використовується DTO BallotRequest, що містить мінімальні необхідні поля (id - унікальний ідентифікатор виборця, vote - (шифрований) голос, signature - підпис виборця).

Обробка бюлетенів на стороні вузла виконується через сервіс BallotService (приклад коду у додатку А). У цьому сервісі реалізовано клас processBallot() який перевіряє, чи цей виборець вже надсилав голос раніше, перевіряє валідність вхідних даних (без криптографії на даному етапі) та зберігає бюлетень у базу даних через репозиторій.

На даному етапі ми ще не реалізували розшифрування результатів. Результати голосування будуть оброблятися після впровадження криптографічного модуля. Але, ми вже можемо створити базовий ендпоінт для отримання результатів (GET /results) у контролері MobileClientController, який повертає тестову або заготовлену відповідь.

У сервісі BallotService ми реалізуємо метод getResults() в якому буде проводится розшифрування результатів і повертатися нам у вигляді VoteResultDto.

4.5 Криптографічний захист

Для забезпечення захисту даних в реалізації вузла використано асиметричну криптографію (RSA) та цифрові підписи, що дозволяють гарантувати автентичність джерела, цілісність бюлетеня та конфіденційність голосу.

Для цього, першим кроком нам необхідно згенерувати ключову пару RSA і це ми будемо робити у CryptoKeyService. Під час запуску вузол буде генерувати власну пару ключів RSA — приватний (зберігається локально, не розкривається) та публічний, який може бути наданий мобільним клієнтам для шифрування голосу.

Відповідним чином розробимо ендпоінт який буде надавати клієнтам відкритий ключ (GET /public-key).

Кожен мобільний клієнт підписує своє повідомлення, і тому нам треба щоб вузол перевіряв цифровий підпис бюлетеня, використовуючи відкритий ключ клієнта. Для цього ми реалізуємо метод isValid() у раніше створеному класі BallotService (Додаток А).

Спочатку витягується відкритий ключ виборця, який повинен бути заздалегідь відомий вузлу, створюється об'єкт перевірки підпису з використанням алгоритму “SHA256withRSA”, який означає, що для підпису спочатку використовується хеш-функція SHA-256, а потім шифрування приватним ключем RSA. Після встановлюється відкритий ключ у перевіряючий об’єкт і завантажуються дані, які були підписані — тобто зашифрований голос. Таким чином, ця перевірка гарантує, що голос був створений саме тим користувачем, що його підписав, і що вміст бюлетеня не було змінено під час передачі.

Коли нам необхідно буде провести підрахунок після завершення голосування вузол повинен розшифрувати бюлетені. Для цього ми реалізуємо функцію decrypt(). Цей процес виконується лише під час підрахунку голосів, що гарантує, що до цього моменту жоден учасник системи не має змоги дізнатися вибір окремого користувача.

4.6 Консенсусна логіка обробки повідомлень між вузлами

Застосування консенсусної логіки важливо для забезпечення узгодженість даних між усіма нодами. Це дозволить вузлам узгоджувати бюлетені, результати голосування та інші критично важливі дані. Кожна нода взаємодіє з іншими вузлами через HTTP-запити, обмінюючись повідомленнями про нові події, зокрема:

- Надходження нового бюлетеня;
- Зміни в списку активних вузлів;
- Узгодження результатів голосування.

Ці повідомлення надсилаються у спеціальному форматі, і кожна нода, яка їх отримує, виконує відповідну обробку (валідацію, перевірку підпису, збереження в базу тощо).

Для реалізації консенсусу спочатку необхідно розробити DTO-компоненти для обміну повідомленнями:

1) ConsensusMessage - це клас який описує повідомлення, що надсилаються вузлами один одному для підтвердження або відхилення певного бюлетенів під час досягнення консенсусу.

2) NodeMessageType - це перелік типів повідомлень, що можуть передаватися між вузлами:

- BALLOT — повідомлення, що містить інформацію про бюлетень, який потребує узгодження.
- CONSENSUS — повідомлення, пов'язане із підтвердженням або запереченням блоку.

Використовується у NodeMessageWrapper, щоб вузол-приймач міг визначити логіку обробки повідомлення залежно від його типу (BALLOT, CONSENSUS).

3) NodeMessageWrapper - цей клас є обгорткою для всіх типів повідомлень між вузлами:

- Поле type вказує, який тип повідомлення надійшов.
- Якщо тип BALLOT, використовується поле ballot.
- Якщо тип CONSENSUS, використовується поле consensus.

Такий підхід дозволить централізовано обробляти всі повідомлення через один контролер та сервіс, уникаючи дублювання коду. Завдяки NodeMessageWrapper, система може легко розширюватися для нових типів повідомлень.

За обробку повідомлень буде відповідати Сервіс NodeMessageService, в ньому реалізовано метод handleMessage():

- Якщо тип повідомлення BALLOT — викликається метод processBallot() у BallotService, який обробляє бюлетень.
- Якщо тип CONSENSUS — викликається handleConsensus() у ConsensusService, який відповідає за логіку консенсусу.

За розсилку повідомлень буде відповідати NodeConnectionService - сервіс відповідає за відправлення повідомлень до інших вузлів у мережі. Метод broadcastToNodes() буде приймати список адрес вузлів і через WebClient надсилати

кожному вузлу повідомлення типу NodeMessageWrapper. Усі ці повідомлення будуть передаватися на ендпоінт: “POST /api/node/message”.

4.7 Узагальнення реалізованих рішень

Отже, у цьому розділі було описано архітектуру, ключові компоненти та функціональні модулі вузла блокчейн-мережі децентралізованої системи захищеного голосування. На основі сформульованих вимог та обраних технологій було реалізовано низку рішень, які забезпечують коректну роботу вузла в умовах децентралізованого середовища. Найважливіші аспекти реалізації включають:

- Гнучку архітектуру системи, що дозволяє масштабувати вузли та легко адаптувати до змін у мережі.
- Механізми автентифікації адміністраторів із підтримкою JWT-токенів, що забезпечують захищений доступ до функцій управління.
- Підключення до блокчейн-мережі та взаємодія з іншими вузлами, що забезпечує динамічне оновлення топології мережі та синхронізацію стану.
- Реалізацію обробки бюлетенів і консенсусних повідомлень, що дозволяє підтримувати цілісність та узгодженість даних серед усіх вузлів.
- Інтеграцію криптографічних механізмів, які забезпечують перевірку цифрового підпису бюлетенів та їх подальше розшифрування при підрахунку голосів.

Також було сформовано набір DTO та сервісів, які відповідають за обробку вхідних запитів, формування відповідей, а також логіку розповсюдження та обробки повідомлень у мережі.

5 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ВУЗЛА БЛОКЧЕЙН-МЕРЕЖІ

5.1 Мета та підхід до тестування

Метою тестування є перевірка коректності функціонування реалізованого вузла блокчейн-мережі децентралізованої системи захищеного голосування, виявлення можливих помилок у логіці програми, а також перевірка взаємодії компонентів системи між собою. У ході тестування перевірялися наступні функціональні компоненти вузла:

- Реєстрація адміністратора вузла – створення нового адміністратора із збереженням даних у базі;
- Автентифікація адміністратора – перевірка правильності облікових даних та генерація JWT токена;
- Обробка бюлетенів – приймання нових голосів від мобільних клієнтів, перевірка цифрового підпису та збереження бюлетенів;
- Отримання результатів голосування – побудова підсумку голосування на основі валідації блоків.

Для тестування усіх вищезазначених функціональних можливостей було вирішено використовувати Postman — інструмент для надсилання HTTP-запитів до REST API. Через нього можна здійснювати надсилання POST-запитів для реєстрації, автентифікації та передачі бюлетенів, GET-запитів для отримання результатів голосування та перевірки доступності вузла, перевіряти статус-коди HTTP-відповідей, вмісту відповіді, валідності токенів, форматів даних та обробку некоректних або несанкціонованих запитів.

Такий підхід дозволить перевірити як внутрішню логіку вузла, так і взаємодію його з іншими учасниками системи — мобільними клієнтами, іншими нодами та сервером-каталогом.

5.2 Перевірка реєстрації адміністратора та автентифікації

Першим кроком необхідно перевірити механізми реєстрації адміністратора та автентифікації У вузлі реалізовані два основних публічних ендпоїнти:

– “POST /register” — реєстрація нового адміністратора.
– “POST /auth” — отримання JWT-токена для авторизованого доступу.

Почнемо з тестування реєстрації нового адміністратора. Введемо поле з поштою, паролем і підтвердженням паролю (Рис. 5.1).

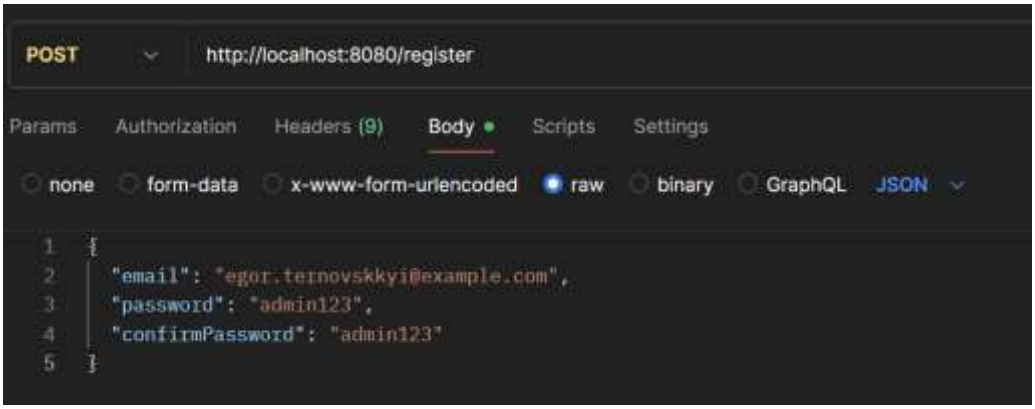

Рисунок 5.1 - Створення запиту для реєстрації нового адміністратора.

Після виконання цього запиту ми очікуємо отримати статус 200 (ok) та отримати id нашого новоствореного адміністратора (Рис. 5.2).

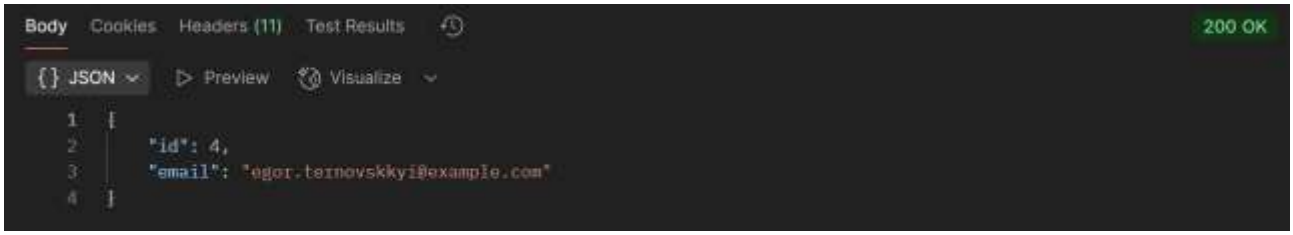

Рисунок 5.2 - Результат створення запиту для реєстрації нового адміністратора.

Як бачимо отриманий результат відповідає нашим очікуванням. Тепер спробуємо повторно надіслати запит про реєстрацію адміністратора використовуючи той самий email. Очікуємо що ми отримаємо статус 400 (Bad Request) з повідомленням що користувач з таким email вже існує.

Рисунок 5.3 - Результат реєстрації під даними вже зареєстрованого користувача.

Все працює як і очікувалося. Система повідомляє нам, що користувач с таким email вже існує. Тепер спробуємо авторизуватися під цими даними.

Рисунок 5.4 - Спроба авторизацій під даними новоствореного аккаунту.

Отже, ми успішно авторизувалися і отримали JWT-токен як показано на рисунку 5.5.

Рисунок 5.5 - Результат авторизацій.

Спробуємо трохи змінити дані входи і ввести неправильний пароль (admin 163). Ми повинні отримати повідомлення про некоректність даних (Рис. 5.6).

Рисунок 5.6 - Результат авторизації під некоректними даними.

Як бачимо, система попереджає нас про те, що дані некоректні. Тепер після того як ми авторизуємося під коректними даними, спробуємо отримати доступ до ресурсу який дозволено тільки адміністраторам з ролю ROLE_ADMIN. Вона повинна надаватися за замовчуванням. Для цього також необхідно вказати згенерований раніше JWT-токен.

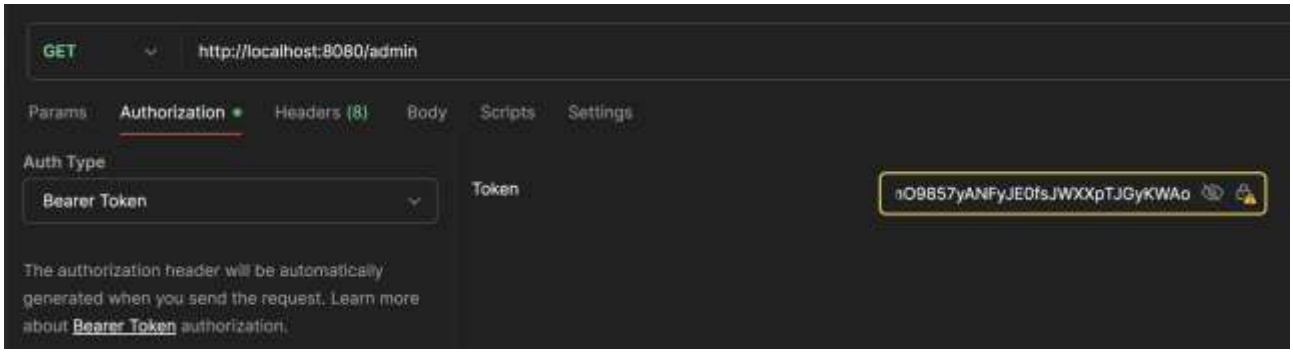

Рисунок 5.7 - Отримання доступу до ресурсу доступним звичайним адміністраторам.

Рисунок 5.8 - Результат успішного отримання доступу до ресурсу адміністратора.

Але тепер спробуємо отримати доступ до ресурсу який доступний лише супер адміністраторам (ROLE_SUPER_ADMIN) (Рис. 5.9).

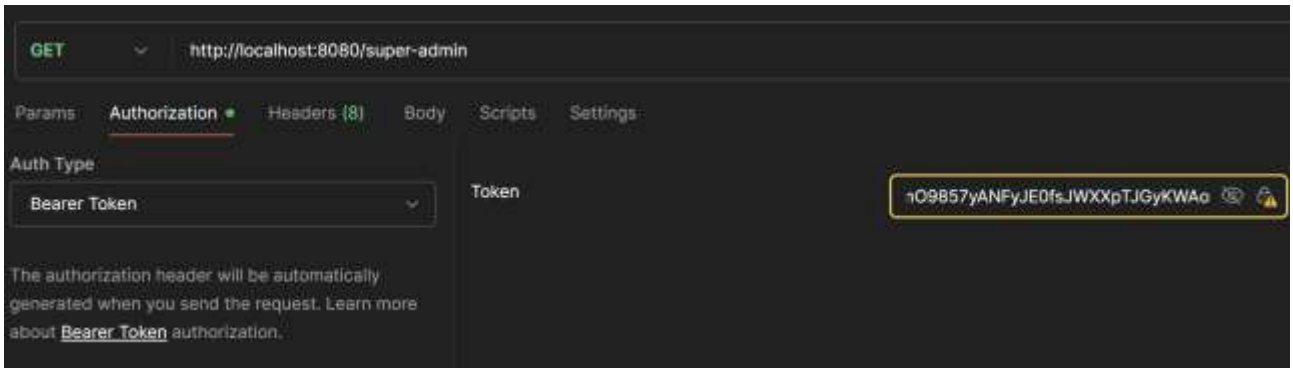

Рисунок 5.9 - Спробо отримання доступу до супер адмінського ресурсу.

Очікуємо, що система заблокує нам туди доступ, бо за замовчуванням в нас надається лише роль звичайного адміністратора (Рис. 5.10).

Рисунок 5.10 - Результат отримання доступу до ресурсу супер адміністратора.

Як і очікувалося, ми не отримали доступ і це нам говорить про це, що під час створення нового адміністратора система коректно надає їм ролі. Але спробуємо перевіти доступ до супер адміністраторського ресурсу надваши нашому користувачу роль супер адміністратора. Для цього скористаємося запитами до таблиці admin_roles у PostgreSQL і мовою SQL додамо роль ROLE_SUPER_ADMIN нашому адміністратору (Рис. 5.11). Як бачимо у запиті ми вказуємо id нашого адміністратора (4) і id ролі у таблиці (2).

```
[2025-05-23 10:03:50] Connected
secure_voting_node_bd> insert into admin_roles (admin_id, role_id)
                       values (4, 2)
[2025-05-23 10:03:50] 1 row affected in 13 ms
```

Рисунок 5.11 - Додавання користувачеві ролі супер адміністратора.

Тепер спробуємо повторно отримати доступ до ресурсу супер адміністратора і як бачимо система вже бачить що ми маємо цю роль і надає нам доступ (Рис. 5.12).

Рисунок 5.12 - Результат отримання доступу до ресурсу супер адміністратора.

Ці тести підтверджують, що механізм створення та авторизації адміністратора працює згідно з очікуваннями. JWT-токен успішно формується та надалі може використовуватись для доступу до захищених маршрутів.

5.3 Тестування обробки бюлетенів

Тепер нам необхідно перевірити, як вузол обробляє бюлетені, що надходять від клієнтів. Під час тестування замість мобільного клієнта ми використовуємо Postman для симуляції надсилання HTTP-запиту на відповідний endpoint вузла.

Першим кроком необхідно отримати відкритий ключ вузла. Виконуємо запит “GET /public-key”, щоб отримати відкритий ключ для шифрування голосу (Рис. 5.13).

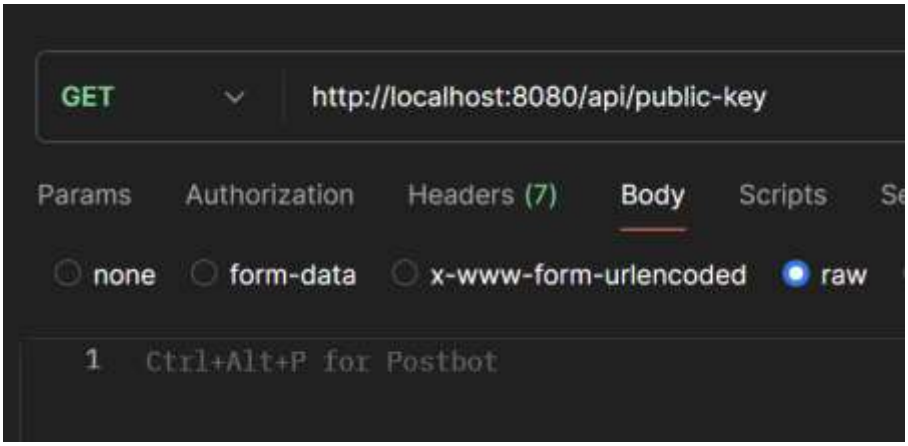

Рисунок 5.13 - Формування запиту на отримання відкритого ключа вузла.

Рисунок 5.14 - Результат отримання відкритого ключа вузла.

Голос, наприклад "Candidate_A", шифрується відкритим ключем вузла, а потім підписується приватним ключем клієнта. Приклад уже сформованого запиту до вузла зображено на рисунку 5.15.

Рисунок 5.15 - Запит до вузла про надсилання бюлетеня.

Бюлетень формується корректно і ми можемо переглянути таблицю ballot (Рис. 5.16) і побачимо що тут додався нас бюлетень.

id	voter_id	encrypted_vote	signature	created_at
1	4	Candidate_A	lTsnB9KH8CgKVZj9q/IYhm3jgkmdlJOOwf8a+x…	2025-05-23 20:44:52.195951

Рисунок 5.16 - Додавання бюлетеня до таблиці.

Тепер спробуємо повторно проголосувати і надіслати цей бюлетень. Система повинна відреагувати на це і не дати можливість це зробити. В логах ми бачимо повідомлення про спробу повторного голосування користувача з id 4 (Рис. 5.17), і перевіривши таблицю бачимо що змін не відбулося.

```
: Duplicate vote attempt by: 4
```

Рисунок 5.17 - Результат повторного голосування.

Спробуємо також надіслати некоректні дані, для цього змінимо декілька символів в signature та ім’я виборця (Рис. 5.18).

Рисунок 5.18 - Підробка підпису та імені виборця.

Після спроби додавання такого бюлетення система повідомляє нам, про те, що формат або підпис некоректний.

```
Invalid ballot format or signature
```

Рисунок 5.19 - Повідомлення від системи про некоректну форму бюлетеня.

В результаті можна сказати, що підсистема обробки бюлетенів продемонструвала стабільну та надійну роботу в усіх ключових сценаріях. Її архітектура та логіка забезпечують базові вимоги до безпеки в контексті електронного голосування, зокрема достовірність та перевірюваність кожного голосу, відсутність можливості для повторного голосування та захист від підробок підписів завдяки використанню асиметричної криптографії.

5.4 Тестування отримання результатів голосування

Перевірити, чи вузол коректно обробляє запити на отримання списку всіх бюлетенів, які були збережені в системі після перевірки цифрових підписів ми можемо також, якщо виконаємо запит на отримання всіх результатів голосування.

	id	voter_id	encrypted_vote	signature	created_at
1	1	4	Candidate_A	lTsnB9KH0CgKVZj9q/IYhm3jgkmdlJQOwf8…	2025-05-23 20:44:52.195951
2	2	1	Candidate_B	pEVuuYMm8v9h2n2bWg95vwsQxFBwHOqcmXr…	2025-05-23 21:56:51.965869
3	3	2	Candidate_B	QBUftV1Pv9jTDvPmtPo7Bj77PmNHKOVWrHs…	2025-05-23 21:57:25.455083
4	4	3	Candidate_A	fJvCjJA18CQQZP25x2+R19MSmJq83qJCaqf…	2025-05-23 21:57:54.770616
5	5	4	Candidate_C	cquMLpc8zcKnLX2oKu3tCOJNVRCsi5VeqvB…	2025-05-23 21:58:52.285260

Рисунок 5.20 - Додавання декількох бюлетенів до таблиці.

Після того як маємо невелику але наповнену різними голосами таблицю ми можемо перевірити результати голосування. Для цього необхідно звернутися на едпоінт GET /api/results і отримати у форматі JSON дані про кількість голосувань за кожного окремого кандидата (Рис. 5.21, 5.22).

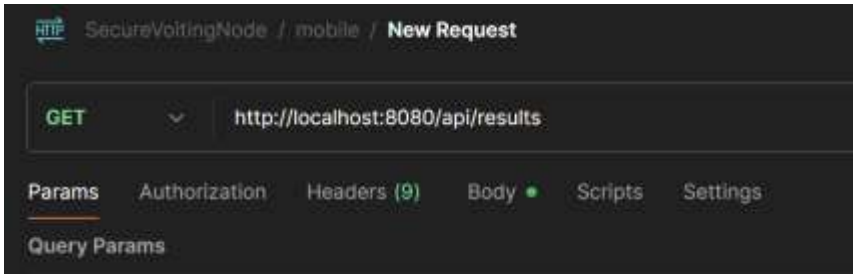

Рисунок 5.21 - Формування GET-запиту для отримання результатів голосування.

Рисунок 5.22 - Отриманий результат голосування.

Отже, так як даних у таблиці небагато, можна звірити ці дані з таблицею і побачити, що програма надала нам правильний результати голосування.

5.5 Узагальнення результатів голосування

У результаті проведеного тестування було перевірено основні функціональні компоненти вузла блокчейн-мережі децентралізованої системи захищеного голосування. Усі ключові модулі пройшли тестування успішно:

- Реєстрація та автентифікація адміністратора працює коректно, із дотриманням вимог до безпеки передачі даних;
- Бюлетені приймаються, перевіряються на цифровий підпис та зберігаються без порушень;
- Побудова блоків та підрахунок результатів голосування проходять без збоїв.

У процесі тестування також виявлено, що система реагує адекватно на помилкові запити (некоректні токени, неправильні формати даних), забезпечуючи необхідний рівень захисту від несанкціонованого доступу.

Але, незважаючи на позитивні результати тестування, подальше вдосконалення системи дозволить підвищити її надійність, масштабованість і зручність використання. Зокрема, доцільно реалізувати наступні покращення:

- Інтеграція автоматизованого тестування (наприклад, з використанням JUnit або Testcontainers) для регулярної перевірки змін у коді;
- Додаткові рівні захисту — наприклад, двофакторна автентифікація для адміністраторів вузлів;
- Вдосконалення логування і моніторингу подій у режимі реального часу для швидкого виявлення проблем або підозрілої активності;
- Оптимізація алгоритму консенсусу для підвищення швидкодії в умовах високого навантаження;
- Тестування на більших об’ємах даних для перевірки масштабованості рішення в реальних умовах.

ВИСНОВКИ

У сучасному світі, де цифрові технології стрімко розвиваються, електронне голосування стає все більш актуальним інструментом для забезпечення демократичних процесів. Проте традиційні електронні системи голосування стикаються з низкою проблем, таких як вразливість до кібератак, відсутність прозорості, складність у забезпеченні анонімності виборців та ризики фальсифікації результатів. Ці виклики підкреслюють необхідність впровадження нових технологічних рішень, які б забезпечили вищий рівень безпеки та довіри до виборчого процесу.

У першому розділі роботи було проведено детальний аналіз сучасних систем електронного голосування, виявлено їхні переваги та недоліки, а також окреслено основні загрози та моделі зловмисників, які можуть впливати на виборчий процес. Це дозволило сформувати чітке розуміння вимог до майбутньої системи.

Другий розділ присвячено аналізу вимог до функціонування вузлів децентралізованої системи захищеного голосування. Визначено, що кожен вузол повинен забезпечувати автентифікацію користувачів, захист від атак типу Sybil та Replay, а також ефективну взаємодію з іншими вузлами мережі.

У третьому розділі розглянуто технології, які можуть бути використані для побудови вузлів блокчейн-мережі. Проаналізовано різні архітектурні підходи, інструменти та фреймворки, такі як Hyperledger Fabric та Tendermint, а також механізми консенсусу, включаючи Practical Byzantine Fault Tolerance (PBFT) та Raft. Ці технології обрані з урахуванням вимог до безпеки, масштабованості та ефективності системи.

У четвертому розділі було зосереджено увагу на безпосередній архітектурі вузла, побудованій з використанням Java Spring Boot та ряду супутніх технологій (Spring Security, WebFlux, JWT, REST API тощо). В межах реалізації були створені такі ключові компоненти:

– механізм реєстрації адміністратора вузла з автентифікацією через JWT-токени;

- каталогова система взаємодії вузлів для динамічного виявлення і підключення до мережі;
- підтримка обробки бюлетенів, що надходять від клієнтів — включаючи валідацію, перевірку криптографічного підпису та збереження;
- реалізація логіки консенсусу — для узгодження даних голосування між вузлами;
- збереження інформації про голосування та блоки у структурованій формі, що придатна для перевірки цілісності.

Усі модулі були побудовані у відповідності до обраної моделі безпеки, де кожен компонент має чітко визначену відповідальність, що сприяє гнучкому масштабуванню і обслуговуванню системи в майбутньому.

П’ятий розділ було присвячено перевірці працездатності реалізованої системи. Тестування проводилося вручну з використанням інструменту Postman, що дозволило здійснити точкову перевірку кожного REST-запиту та простежити повний життєвий цикл взаємодії: від реєстрації адміністратора — до збереження бюлетеня та підрахунку результатів. У результаті тестування було:

- підтверджено працездатність ключових функцій;
- перевірено захист API від неавторизованого доступу;
- перевірено правильність збереження бюлетенів та результатів голосування;
- оцінено стабільність відповіді сервісу при серії запитів.

Загалом, тестування підтвердило, що вузол функціонує згідно з технічними вимогами та може виконувати роль окремого елемента в повноцінній децентралізованій системі електронного голосування. Таке програмне рішення може бути застосовано в локальних виборах, внутрішніх корпоративних голосуваннях, а також у громадських ініціативах, що потребують прозорого й перевіреного процесу ухвалення рішень.

У перспективі розвиток системи може включати підтримку постквантових криптографічних алгоритмів, інтеграцію з міжнародними стандартами електронної ідентифікації та впровадження штучного інтелекту для виявлення аномальної активності в голосуваннях.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ghassan Z. Qadah, Rani Taha. Electronic voting systems: Requirements, design, and implementation issues [Електронний ресурс] // Computer Standards & Interfaces. — 2007. — Т. 29, № 3. — С. 376–386. — Режим доступу: https://www.sciencedirect.com/science/article/abs/pii/S0920548906000754 (дата звернення: 13.04.2025).
2. Neelam N. E-voting systems for developing democracies // International Journal of Innovation, Creativity and Change. — 2021. — Т. 15, № 2. — Режим доступу: https://www.ijicc.net/images/Vol_15/Iss_2/17149_Neelam_2021_E_R.pdf (дата звернення: 13.04.2025).
3. Vassil K. How much does an e-vote cost? Cost Comparison per Vote in Multichannel Elections in Estonia [Електронний ресурс] — Режим доступу: file:///C:/Users/terno/Downloads/how-much-does-an-e-vote-cost-cost-comparison-per-vote-in-multichannel-elections-in-estonia_78636.pdf (дата звернення: 22.05.2025).
4. E-Voting in Estonia. Chapter 6 // Real World Electronic Voting. — Режим доступу: https://realworldevoting.com/files/Chapter6.pdf (дата звернення: 13.04.2025).
5. Herrnson P. et al. Voters' Evaluations of Electronic Voting Systems — Режим доступу: https://gvpt.umd.edu/sites/gvpt.umd.edu/files/pubs/Herrnson%20et%20al%20APR%20Evals%20of%20Electronic%20Voting.pdf (дата звернення: 14.04.2025).
6. Landgren J. Investigating the Security of End-to-End and Blockchain-based Electronic Voting Systems — Режим доступу: https://www.diva-portal.org/smash/get/diva2:1683763/FULLTEXT01.pdf (дата звернення: 14.04.2025).
7. Ayed A. Decentralized E-Voting Systems Based on the Blockchain Technology [Електронний ресурс] — Режим доступу: https://www.researchgate.net/publication/321947971_Decentralized_E-Voting_Systems_Based_on_the_Blockchain_Technology (дата звернення: 15.04.2025).

8. E-Voting with Blockchain: An E-Voting Protocol with Decentralisation and Voter Privacy // arXiv. — Режим доступу: https://arxiv.org/pdf/1805.10258 (дата звернення: 15.04.2025).
9. Nakamoto S. Decentralized Trusted Timestamping using the Crypto Currency Bitcoin [Електронний ресурс] — Режим доступу: https://d-nb.info/1163539104/34 (дата звернення: 15.04.2025).
10. Adida B., Rivest R. L. Scratch & Vote: Self-Contained Paper-Based Cryptographic Voting — Режим доступу: https://people.csail.mit.edu/rivest/pubs/AR06.pdf (дата звернення: 16.04.2025).
11. Ryan P.Y.A. Coercion-Resistant Hybrid Voting Systems [Електронний ресурс] — Режим доступу: https://www.researchgate.net/publication/220764740_Coercion-Resistant_Hybrid_Voting_Systems (дата звернення: 17.04.2025).
12. Chaum D., Rivest R. et al. Scantegrity II: End-to-End Verifiability by Voters of Optical Scan Elections Through Confirmation Codes — Режим доступу: https://people.csail.mit.edu/rivest/pubs/CCCEx09.pdf (дата звернення: 17.04.2025).
13. Appel A.W., DeMillo R.A., Stark P.B. Ballot-Marking Devices (BMDs) Cannot Assure the Will of the Voters — Режим доступу: https://www.stat.berkeley.edu/~stark/Preprints/appelEtal20.pdf (дата звернення: 17.04.2025).
14. Springall D., Halderman J.A. et al. Security Analysis of the Estonian Internet Voting System — Режим доступу: https://jhalderm.com/pub/papers/ivoting-ccs14.pdf (дата звернення: 18.04.2025).
15. Feldman A.J., Halderman J.A., Felten E.W. Security Analysis of the Diebold AccuVote-TS Voting Machine — Режим доступу: https://www.usenix.org/legacy/event/evt07/tech/full_papers/feldman/feldman.pdf (дата звернення: 18.04.2025).
16. Specter M.A., Koppel J., Halderman J.A. The Ballot is Busted Before the Blockchain — Режим доступу: https://internetpolicy.mit.edu/wp-content/uploads/2020/02/SecurityAnalysisOfVoatz_Public.pdf (дата звернення: 18.04.2025).

17. Chaum D. Secret-Ballot Receipts: True Voter-Verifiable Elections [Електронний ресурс] — Режим доступу: https://people.csail.mit.edu/rivest/voting/papers/Chaum-SecretBallotReceiptsTrueVoterVerifiableElections.pdf (дата звернення: 19.04.2025).
18. Stark P.B. Evidence-Based Elections — Режим доступу: https://www.stat.berkeley.edu/~stark/Preprints/evidenceVote12.pdf (дата звернення: 19.04.2025).
19. Jamil D., Tariq S., Akbar R. A Conceptual Secure Blockchain-Based Electronic Voting System // International Journal of Network Security & Its Applications. — 2017. — Т. 9, № 3. — Режим доступу: https://aircconline.com/ijnsa/V9N3/9317ijnsa01.pdf (дата звернення: 19.04.2025).
20. Krimmer R. (ed.) E-Voting Requirements for Less Developed Democracies // Electronic Voting 2006. — Bonn: Gesellschaft für Informatik. — Режим доступу: https://e-vote-id.org/res/2008_Proceedings.pdf (дата звернення: 20.04.2025).
21. NIST Cybersecurity Framework — Режим доступу: https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.29.pdf (дата звернення: 20.04.2025).
22. ISO/IEC 27001:2022. Інформаційні технології. Методи забезпечення безпеки. Системи управління інформаційною безпекою. Вимоги [Електронний ресурс]. — Режим доступу: https://www.iso.org/standard/82875.html (дата звернення: 20.04.2025).
23. НД ТЗІ 1.1-002-99. Загальні положення щодо захисту інформації в комп'ютерних системах від несанкціонованого доступу — Режим доступу: https://tzi.com.ua/downloads/1.1-002-99.pdf (дата звернення: 21.04.2025).
24. Cybersecurity and Infrastructure Security Agency (CISA). Election Security Resources Guide — 2020. — Режим доступу: https://www.cisa.gov/sites/default/files/publications/cisa_election-security-resources-guide-Sept-2020.pdf (дата звернення: 21.04.2025).
25. Blockchain-Based E-Voting Systems: A Technology Review [Електронний ресурс] // MDPI Electronics. — Режим доступу: https://www.mdpi.com/2079-9292/13/1/17 (дата звернення: 21.04.2025).

26. Decentralized and Automated Online Voting System using Blockchain Technology [Електронний ресурс] // ResearchGate. — Режим доступу: https://www.researchgate.net/publication/374505381_Decentralized_and_Automated_Online_Voting_System_using_Blockchain_Technology (дата звернення: 21.04.2025).
27. Decentralized and Automated Online Voting System using Blockchain Technology // E3S Web of Conferences. — Режим доступу: https://www.e3s-conferences.org/articles/e3sconf/pdf/2023/67/e3sconf_icmpc2023_01046.pdf (дата звернення: 21.04.2025).
28. van Renesse R., Minsky Y., Hayden M. An Epidemic Protocol for Managing Routing Tables in Very Large Peer-to-Peer Networks // Distributed Systems Group. — Режим доступу: https://www.distributed-systems.net/my-data/papers/2003.dsom.pdf (дата звернення: 22.04.2025).
29. A Blockchain-based Electronic Voting System: EtherVote [Електронний ресурс] // arXiv. — Режим доступу: https://arxiv.org/pdf/2307.10726 (дата звернення: 22.04.2025).
30. Building Secure E-Voting Systems: A Blockchain Approach for Transparent Democracy [Електронний ресурс] // ResearchGate. — Режим доступу: https://www.researchgate.net/publication/385379331_Building_Secure_E-Voting_Systems_A_Blockchain_Approach_for_Transparent_Democracy (дата звернення: 22.04.2025).
31. Blockchain-Based Electronic Voting: A Secure and Transparent Solution [Електронний ресурс] // MDPI Journal of Cybersecurity and Privacy. — Режим доступу: https://www.mdpi.com/2410-387X/7/2/27 (дата звернення: 23.04.2025).
32. Understanding Blockchain-Based E-Voting Systems [Електронний ресурс] // Verix. — Режим доступу: https://www.verix.io/blog/blockchain-based-e-voting-systems (дата звернення: 23.04.2025).
33. Khobragade P. Blockchain Consensus Algorithms: A Survey // NIT Rourkela. — Режим доступу: http://dspace.nitrkl.ac.in:8080/dspace/bitstream/2080/3720/1/2022_ICBA_PKhobragade_Blockchain.pdf (дата звернення: 23.04.2025).

34. PoV: An Efficient Voting-Based Consensus Algorithm for Consortium Blockchains [Електронний ресурс] // ResearchGate. — Режим доступу: https://www.researchgate.net/publication/340001935_PoV_An_Efficient_Voting-Based_Consensus_Algorithm_for_Consortium_Blockchains (дата звернення: 23.04.2025).
35. Blockchain-Based E-Voting System: A Decentralized Approach on the Ethereum Private Network [Електронний ресурс] // ResearchGate. — Режим доступу: https://www.researchgate.net/publication/388062976_Blockchain-Based_E-Voting_System_A_Decentralized_Approach_on_the_Ethereum_Private_Network (дата звернення: 24.04.2025).
36. E-voting System Using Homomorphic Encryption and Blockchain Technology to Encrypt Voter Data [Електронний ресурс] // arXiv. — Режим доступу: https://arxiv.org/pdf/2111.05096 (дата звернення: 24.04.2025).
37. The Role of Digital Signature Cards in Electronic Voting [Електронний ресурс] // ResearchGate. — Режим доступу: https://www.researchgate.net/publication/221181669_The_Role_of_Digital_Signature_Cards_in_Electronic_Voting (дата звернення: 25.04.2025).
38. Chirotonia: A Scalable and Secure e-Voting Framework based on Blockchains and Linkable Ring Signatures // IMDEA Networks. — Режим доступу: https://dspace.networks.imdea.org/bitstream/handle/20.500.12761/1609/Chirotonia__An_e_voting_framework_based_on_blockchain_andring_signatures.pdf?sequence=1 (дата звернення: 25.04.2025).
39. Securing e-voting based on blockchain in P2P network [Електронний ресурс] // ResearchGate. — Режим доступу: https://www.researchgate.net/publication/333439952_Securing_e-voting_based_on_blockchain_in_P2P_network (дата звернення: 25.04.2025).
40. Voting System Performance Guidelines // U.S. Election Assistance Commission. — Режим доступу: https://www.eac.gov/sites/default/files/eac_assets/1/28/VVSG%201.1%20VOL%201.508compliant.FINAL.pdf (дата звернення: 26.04.2025).

41. Principles and requirements for a secure e-voting system [Електронний ресурс] // ScienceDirect. — Режим доступу: https://www.sciencedirect.com/science/article/abs/pii/S0167404802010143 (дата звернення: 26.04.2025).
42. Security Considerations for Remote Electronic Voting over the Internet [Електронний ресурс] // arXiv. — Режим доступу: https://arxiv.org/pdf/cs.cy/0108017 (дата звернення: 26.04.2025).
43. Coercion-Resistant Electronic Elections // IACR. — Режим доступу: https://eprint.iacr.org/2002/165.pdf (дата звернення: 26.04.2025).
44. Best Practices for Election Technology // U.S. Election Assistance Commission. — Режим доступу: https://www.eac.gov/sites/default/files/electionofficials/security/Best_Practices_for_Election_Technology_508.pdf (дата звернення: 26.04.2025).
45. Newman S. Building Microservices — Режим доступу: https://book.northwind.ir/bookfiles/building-microservices/Building.Microservices.pdf (дата звернення: 26.04.2025).
46. Proof of Availability & Retrieval in a Modular Blockchain Architecture // IACR. — Режим доступу: https://eprint.iacr.org/2022/455.pdf (дата звернення: 27.04.2025).
47. BlendMAS: A BLockchain-ENabled Decentralized Microservices Architecture for Smart Public Safety [Електронний ресурс] // arXiv. — Режим доступу: https://arxiv.org/pdf/1902.10567 (дата звернення: 27.04.2025).
48. On the Serverless Nature of Blockchains and Smart Contracts // arXiv. — Режим доступу: https://arxiv.org/pdf/2011.12729 (дата звернення: 28.04.2025).
49. An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends [Електронний ресурс] // ResearchGate. — Режим доступу: https://www.researchgate.net/publication/318131748_An_Overview_of_Blockchain_Technology_Architecture_Consensus_and_Future_Trends (дата звернення: 28.04.2025).
50. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains // arXiv. — Режим доступу: https://arxiv.org/pdf/1801.10228 (дата звернення: 28.04.2025).

51. Defining Decentralisation in Permissionless Blockchain Systems // AJIC. — Режим доступу: https://scielo.org.za/pdf/ajic/v29/02.pdf (дата звернення: 28.04.2025).
52. Scalability and Security in Blockchain Networks: Evaluation of Sharding Algorithms and Prospects for Decentralized Data Storage [Електронний ресурс] // MDPI Electronics. — Режим доступу: https://www.mdpi.com/2227-7390/12/23/3860 (дата звернення: 29.04.2025).
53. Towards Scaling Blockchain Systems via Sharding // arXiv. — Режим доступу: https://arxiv.org/pdf/1804.0399 (дата звернення: 29.04.2025).
54. A Systematic Review on Blockchain Scalability [Електронний ресурс] // ResearchGate. — Режим доступу: https://www.researchgate.net/publication/374496563_A_Systematic_Review_on_Blockchain_Scalability (дата звернення: 30.04.2025).
55. The Curses of Blockchain Decentralization // arXiv. — Режим доступу: https://arxiv.org/pdf/1810.02937 (дата звернення: 01.05.2025).
56. NIST. Blockchain Technology Overview — Режим доступу: https://nvlpubs.nist.gov/nistpubs/ir/2018/NIST.IR.8202.pdf (дата звернення: 02.05.2025).
57. Horstmann C. Core Java. Volume I—Fundamentals — Режим доступу: https://ptgmedia.pearsoncmg.com/images/9780134177304/samplepages/9780134177304.pdf (дата звернення: 02.05.2025).
58. Kreibich K. PostgreSQL: Up and Running — Режим доступу: https://k0d.cc/storage/books/Databases/Postgresql/PostgreSQL%20Up%20and%20Running,%202nd%20Edition.pdf (дата звернення: 02.05.2025).
59. Ibarra L. PostgreSQL 9.0 High Performance — Режим доступу: https://bbs.landingbj.com/upload/file/20151209/1449649831683004590.pdf (дата звернення: 03.05.2025).
60. Gorman J. PostgreSQL for Data Architects — Режим доступу: https://api.pageplace.de/preview/DT0400.9781783288618_A24764966/preview-9781783288618_A24764966.pdf (дата звернення: 03.05.2025).
61. Analysis and Security Testing for GRPC [Електронний ресурс] // ResearchGate. — Режим доступу:

https://www.researchgate.net/publication/377360768_Analysis_and_Security_Testing_for_GRPC (дата звернення: 04.05.2025).

62. Jain R. The Past, Present, and Future of P2P Network: Applications and Challenges [Електронний ресурс]. — Режим доступу: https://www.cse.wustl.edu/~jain/cse570-21/ftp/p2p/index.html (дата звернення: 04.05.2025).
63. Ripeanu M. Mapping the Gnutella Network: Properties of Large-Scale Peer-to-Peer Systems and Implications for System Design // SciSpace. — Режим доступу: https://scispace.com/pdf/mapping-the-gnutella-network-properties-of-large-scale-peer-2sgr8crt34.pdf (дата звернення: 05.05.2025).
64. Park H. Peer-to-Peer Networks – Protocols, Cooperation and Competition — Режим доступу: http://medianetlab.ee.ucla.edu/papers/chapter_P2P_hpark.pdf (дата звернення: 05.05.2025).
65. Douceur J. The Sybil Attack // Freehaven. — Режим доступу: https://www.freehaven.net/anonbib/cache/sybil.pdf (дата звернення: 06.05.2025).
66. Takagi sugeno control for networked system exposed to a replay attack [Електронний ресурс] // ResearchGate. — Режим доступу: https://www.researchgate.net/publication/348760343_Takagi_sugeno_control_for_networked_system_exposed_to_a_replay_attack (дата звернення: 07.05.2025).
67. Eclipse Attack Detection on a Blockchain Network as a Non-Parametric Change Detection Problem [Електронний ресурс] // arXiv. — Режим доступу: https://arxiv.org/html/2404.00538v2 (дата звернення: 10.05.2025).
68. Distributed-Ledger-based Authentication with Decentralized Identifiers and Verifiable Credentials // Semantic Scholar. — Режим доступу: https://www.semanticscholar.org/reader/9cf4526c392fc934c00a046556a5da7515dde8f3 (дата звернення: 12.05.2025).
69. Security Services Using Blockchains: A State of the Art Survey [Електронний ресурс] // arXiv. — Режим доступу: https://arxiv.org/pdf/1810.08735 (дата звернення: 13.05.2025).
70. Spring Documentation [Електронний ресурс]. — Режим доступу: https://spring.io/ (дата звернення: 13.05.2025).

ДОДАТОК А

Код основних класів розробленого додатку. Повний код можна отримати за посиланням: https://github.com/ternovskiyegor/secure-voting-node/tree/master

```
@SpringBootApplication
public class SecureVotingNodeApplication {
   public static void main(String[] args) {
      SpringApplication.run(SecureVotingNodeApplication.class, args);
   }
}
@RestController
@RequiredArgsConstructor
public class AuthController {
   private final AuthService authService;
   @PostMapping("/auth")
   public ResponseEntity<?> createAuthToken(@RequestBody JwtRequest
jwtRequest) {
       return authService.createAuthToken(jwtRequest);
   }
   @PostMapping("/register")
   public ResponseEntity<?> createNewAdmin(@RequestBody
RegistrationAdminDto registrationAdminDto) {
       return authService.createNewAdmin(registrationAdminDto);
   }
}


@RestController
@RequestMapping("/api")
@RequiredArgsConstructor
public class MobileClientController {
   private final BallotService ballotService;
   private final CryptoKeyService cryptoKeyService;
   @GetMapping("/public-key")
   public ResponseEntity<String> getPublicKey() {
       PublicKey publicKey = cryptoKeyService.getPublicKey();
```

```
        String encodedKey =
Base64.getEncoder().encodeToString(publicKey.getEncoded());
        return ResponseEntity.ok(encodedKey);
    }
    @GetMapping("/results")
    public ResponseEntity<?> getResults() {
        return ResponseEntity.ok(ballotService.getResults());
    }
    @PostMapping("/vote")
    public ResponseEntity<?> receiveBallot(@RequestBody BallotRequest
request) {
        ballotService.processBallot(request);
        return ResponseEntity.ok().build();
    }
    @GetMapping("/ping")
    public ResponseEntity<?> ping() {
        return ResponseEntity.ok("success");
    }
}


@RestController
@RequiredArgsConstructor
@RequestMapping("/network")
public class NetworkController {
    private final NetworkService networkService;
    @GetMapping("/connect")
    public Flux<NodeAddressDto> connectToNetwork(@RequestParam String host)
{
        return networkService.fetchRandomNodes(host);
    }
}
@RestController
public class NodeController {
    @GetMapping("/ping")
    public ResponseEntity<Void> ping() {
        return ResponseEntity.ok().build();
```

```
    }
}

@RestController
@RequestMapping("/api/node")
@RequiredArgsConstructor
public class NodeMessageController {
    private final NodeMessageService nodeMessageService;
    @PostMapping("/message")
    public ResponseEntity<?> receiveMessage(@RequestBody NodeMessageWrapper
messageWrapper) {
        nodeMessageService.handleMessage(messageWrapper);
        return ResponseEntity.ok().build();
    }
}

@RestController
@RequiredArgsConstructor
public class TestController {
    @GetMapping("/admin")
    public String getAdminData() {
        return "Admin Data";
    }

    @GetMapping("/super-admin")
    public String getSuperAdminData() {
        return "Super admin data";
    }

    @GetMapping("/info")
    public String getAdminInfo(Principal principal) {
        return principal.getName();
    }
}
@Service
```

```
@RequiredArgsConstructor
public class AdminService implements UserDetailsService {
   private final AdminRepository adminRepository;
   private final PasswordEncoder passwordEncoder;
   private final RoleService roleService;
   public Optional<Admin> findByEmail(String email) {
       return adminRepository.findByEmail(email);
   }
   @Override
   @Transactional
   public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
       Admin admin = findByEmail(username).orElseThrow(() -> new
UsernameNotFoundException(
               String.format("User '%s' not found", username)
       ));
       return new User(
               admin.getEmail(),
               admin.getPassword(),
               admin.getRoles().stream().map(role -> new
SimpleGrantedAuthority(role.getName())).collect(Collectors.toList())
       );
   }
   public Admin createNewAdmin(RegistrationAdminDto registrationAdminDto) {
       Admin admin = new Admin();
       admin.setEmail(registrationAdminDto.getEmail());

admin.setPassword(passwordEncoder.encode(registrationAdminDto.getPassword()
));
       admin.setRoles(List.of(roleService.getAdminRole()));
       return adminRepository.save(admin);
   }
}
@Service
@RequiredArgsConstructor
public class AuthService {
```

```
    private final AdminService adminService;

    private final JwtTokenUtils jwtTokenUtils;

    private final AuthenticationManager authenticationManager;


    public ResponseEntity<?> createAuthToken(@RequestBody JwtRequest
authRequest) {

        try {

            authenticationManager.authenticate(new
UsernamePasswordAuthenticationToken(authRequest.getEmail(),
authRequest.getPassword()));

        } catch (BadCredentialsException e) {

            return new ResponseEntity<>(new
AppError(HttpStatus.UNAUTHORIZED.value(), "Incorrect login or password"),
HttpStatus.UNAUTHORIZED);

        }

        UserDetails userDetails =
adminService.loadUserByUsername(authRequest.getEmail());

        String token = jwtTokenUtils.generateToken(userDetails);

        return ResponseEntity.ok(new JwtResponse(token));

    }

    public ResponseEntity<?> createNewAdmin(@RequestBody
RegistrationAdminDto registrationAdminDto) {

        if
(!registrationAdminDto.getPassword().equals(registrationAdminDto.getConfirm
Password())) {

            return new ResponseEntity<>(new
AppError(HttpStatus.BAD_REQUEST.value(), "Password do not matches"),
HttpStatus.BAD_REQUEST);

        }

        if
(adminService.findByEmail(registrationAdminDto.getEmail()).isPresent()) {

            return new ResponseEntity<>(new
AppError(HttpStatus.BAD_REQUEST.value(), "This user already exists"),
HttpStatus.BAD_REQUEST);

        }

        Admin admin = adminService.createNewAdmin(registrationAdminDto);

        return ResponseEntity.ok(new AdminDto(admin.getId(),
admin.getEmail()));

    }

}
```

```
@Service
@RequiredArgsConstructor
@Slf4j
public class BallotService {
   private final BallotRepository ballotRepository;
   private final CryptoKeyService cryptoKeyService;
   private final NodeConnectionService nodeConnectionService;
   public void processBallot(BallotRequest request) {
       if (ballotRepository.existsByVoterId(request.getId())) {
           log.warn("Duplicate vote attempt by: {}", request.getId());
           return;
       }
       if (!isValid(request)) {
           log.warn("Invalid ballot format or signature");
           throw new IllegalArgumentException("Invalid ballot or
signature");
       }
       Ballot ballot = new Ballot();
       ballot.setVoterId(request.getId());
       ballot.setEncryptedVote(request.getVote());
       ballot.setSignature(request.getSignature());
       ballot.setCreatedAt(LocalDateTime.now());
       ballotRepository.save(ballot);
       log.info("Ballot saved from voterId={}", request.getId());
       NodeMessageWrapper messageWrapper = new NodeMessageWrapper();
       messageWrapper.setType(NodeMessageType.BALLOT);
       messageWrapper.setBallot(request);
       nodeConnectionService.broadcastToNode(messageWrapper);
   }
   private boolean isValid(BallotRequest request) {
       try {
           byte[] publicKeyBytes =
Base64.getDecoder().decode(request.getPublicKey());
           X509EncodedKeySpec keySpec = new
X509EncodedKeySpec(publicKeyBytes);
```

```
            KeyFactory keyFactory = KeyFactory.getInstance("RSA");
            PublicKey publicKey = keyFactory.generatePublic(keySpec);

            byte[] signatureBytes =
Base64.getDecoder().decode(request.getSignature());

            Signature sig = Signature.getInstance("SHA256withRSA");
            sig.initVerify(publicKey);
           sig.update(request.getVote().getBytes(StandardCharsets.UTF_8));
            return sig.verify(signatureBytes);
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }
    public VoteResultDto getResults() {
        Map<String, Long> ballots = ballotRepository.findAll().stream()
                .map(Ballot::getEncryptedVote)
                .filter(v -> !v.equals("invalid"))
                .collect(Collectors.groupingBy(Function.identity(),
Collectors.counting()));
        return new VoteResultDto(ballots);
    }
}

@Service
public class CryptoKeyService {
    private KeyPair keyPair;
    @PostConstruct
    public void generateKeyPair() {
        try {
            KeyPairGenerator generator =
KeyPairGenerator.getInstance("RSA");
            generator.initialize(2048);
            keyPair = generator.generateKeyPair();
        } catch (NoSuchAlgorithmException e) {
```

```
            throw new IllegalStateException("RSA algorithm not available",
e);
        }
    }

    public String decrypt(String encryptedData) throws Exception {
        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.DECRYPT_MODE, keyPair.getPrivate());
        byte [] decryptedBytes =
cipher.doFinal(Base64.getUrlDecoder().decode(encryptedData));
        return new String(decryptedBytes, StandardCharsets.UTF_8);
    }
    public PublicKey getPublicKey() {
        return keyPair.getPublic();
    }
    public PrivateKey getPrivateKey() {
        return keyPair.getPrivate();
    }
}

@Service
@RequiredArgsConstructor
@Slf4j
public class NetworkService {
    private final WebClient.Builder webClientBuilder;
    private final DnsUtils dnsUtils;
    public Flux<NodeAddressDto> fetchRandomNodes(String host) {
        try {
            String ip = dnsUtils.getCatalogServer(host);
            log.info("Resolved catalog IP: {}", ip);
            String url = "http://" + ip + ":8080/api/nodes/random";
            return webClientBuilder.build()
                    .get()
                    .uri(url)
                    .retrieve()
```

```
                    .bodyToFlux(NodeAddressDto.class);
        } catch (UnknownHostException e) {
            log.debug("DNS resolution or API request failed");
            return Flux.empty();
        }
    }
}


@Service
@RequiredArgsConstructor
@Slf4j
public class NodeConnectionService {
    private final WebClient.Builder webClientBuilder;
    private final NodeService nodeService;
    @Value("${node.connections.limit}")
    private int connectionLimit;
    private final Map<String, WebClient> activeConnections = new
ConcurrentHashMap<>();
    @PostConstruct
    public void initializeConnections() {
        try {
            List<NodeAddressDto> nodes = nodeService.getAllNodes();


            nodes.stream()
                    .limit(connectionLimit)
                    .forEach(this::connectToNode);
        } catch (Exception e) {
            log.error("Error during initial node connection setup: {}",
e.getMessage());
        }
    }
    public void connectToNode(NodeAddressDto nodeDto) {
        String baseUrl = "http://" + nodeDto.getIp() + ":" +
nodeDto.getPort();
        WebClient client = webClientBuilder.baseUrl(baseUrl).build();
        client.get().uri("/ping")
```

```
                .retrieve()
                .toBodilessEntity()
                .doOnSuccess(response -> {
                    log.info("Connected to node: {}", baseUrl);
                    activeConnections.put(baseUrl, client);
                })
                .doOnError(error -> log.warn("Failed to connect to node:
{}", baseUrl))
                .subscribe();
    }
    @Scheduled(fixedDelayString = "${node.connections.refresh-limit}")
    public void refreshConnection() {
        log.info("Refreshing node connections");
        Set<String> failedConnections = new HashSet<>();
        // checking all active connections
        activeConnections.forEach((url, client) -> {
            client.get().uri("/ping")
                    .retrieve()
                    .toBodilessEntity()
                    .doOnError(err -> {
                        log.warn("Node down: {}", url);
                        failedConnections.add(url);
                    })
                    .subscribe();
        });
        failedConnections.forEach(activeConnections::remove);
        int needLimit = connectionLimit - activeConnections.size();
        if (needLimit > 0) {
            nodeService.getAllNodes().stream()
                    .filter(node -> !activeConnections.containsKey("http://"
+ node.getIp() + ":" + node.getPort()))
                    .limit(needLimit)
                    .forEach(this::connectToNode);
        }
    }
```

```
    public Collection<WebClient> getActiveConnections() {
        return activeConnections.values();
    }
    public void broadcastToNode(NodeMessageWrapper messageWrapper) {
        activeConnections.forEach((url, client) -> {
            client.post()
                    .uri("/api/node/message")
                    .bodyValue(messageWrapper)
                    .retrieve()
                    .toBodilessEntity()
                    .doOnSuccess(resp -> log.info("Successfully sent message
to {}", url))
                    .doOnError(err -> log.warn("Failed to send message to {}
: {}", url, err.getMessage()))
                    .subscribe();
        });
    }


}
@Service
@RequiredArgsConstructor
@Slf4j
public class NodeMessageService {
    private final BallotService ballotService;
    public void handleMessage(NodeMessageWrapper messageWrapper) {
        if (messageWrapper.getType() == NodeMessageType.BALLOT) {
            BallotRequest ballotRequest = messageWrapper.getBallot();
            log.info("Received ballot from another node: {}",
ballotRequest.getId());
            ballotService.processBallot(ballotRequest);
        } else if (messageWrapper.getType() == NodeMessageType.CONSENSUS) {
            ConsensusMessage message = messageWrapper.getConsensus();
            log.info("Received consensus message from node: {}",
message.getSenderNodeId());
            // TODO : consensus processing (after adding the algorithm)
        } else {
```

```
            log.warn("Unknown message type received: {}",
messageWrapper.getType());
        }
    }
}

@Service
@RequiredArgsConstructor
public class NodeService {
    private final NodeRepository nodeRepository;
    public void saveNode(List<NodeAddressDto> nodeAddressDtos) {
        List<Node> nodes = nodeAddressDtos.stream()
                .map(dto -> new Node(dto.getIp(), dto.getPort(),
LocalDateTime.now())).collect(Collectors.toList());
        nodeRepository.deleteAll();
        nodeRepository.saveAll(nodes);
    }
    public List<NodeAddressDto> getAllNodes() {
        return nodeRepository.findAll().stream()
                .map(entity -> new NodeAddressDto(entity.getIp(),
entity.getPort()))
                .collect(Collectors.toList());
    }
    public void clearAllNodes() {
        nodeRepository.deleteAll();
    }
}

@Service
@RequiredArgsConstructor
public class RoleService {
    private final RoleRepository roleRepository;
    public Role getAdminRole() {
        return roleRepository.findByName("ROLE_ADMIN").get();
    }
}
```