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: «Аналіз вимог і розробка програмного забезпечення серверу p2p-мережі децентралізованої системи захищеного голосування»

оцінка « _____ »

Керівник: д.т.н., проф. Олійников Р.В.

Голова ЕК
Мичуда Л.З.

Рецензент: доц. Родінко М.Ю.

Виконавець: студент групи КБ-42
Доценко Д.В.

Харків - 2025

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра містить 55 сторінок, 9 рисунків, 3 таблиці, 1 додаток, 41 посилання на джерела.

Мета роботи полягає в аналізі вимог та розробці програмного забезпечення серверу P2P-мережі децентралізованої системи захищеного голосування з урахуванням функціональних, нефункціональних і адміністративних аспектів.

Об’єкт дослідження – механізми організації та захисту серверної частини децентралізованих систем електронного голосування.

Предмет дослідження – програмне забезпечення серверу P2P-мережі, його особливості, переваги та недоліки. Технології для реалізації архітектури серверу P2P-мережі.

Основними методами дослідження є системний аналіз вимог за CIA-моделлю, моделювання загроз і ризиків згідно з українськими та інтернаціональними стандартами, компонентне проєктування на базі Spring Boot та RocksDB.

У роботі досліджено архітектуру P2P-мережі, функціональні й нефункціональні вимоги до серверу, адміністративні функції, а також реалізовано прототип програмного забезпечення серверу P2P-мережі децентралізованої системи захищеного голосування.

Результати роботи можуть бути використані при впровадженні пілотних проєктів електронного голосування, для розробки аналогічних розподілених сервісів і подальшого дослідження механізмів консенсусу та інтеграції блокчейн-технологій.

Ключові слова: P2P-МЕРЕЖА, СЕРВЕР, ВРАЗЛИВОСТІ, ЗАХИЩЕНЕ ГОЛОСУВАННЯ, SPRING BOOT, ROCKSDB, REST API.

ABSTRACT

The explanatory note to the bachelor's qualification thesis comprises 55 pages, 9 figures, 3 tables, 1 appendix, and 41 references.

The aim of the work is to analyze the requirements and develop the software of a P2P-network server for a decentralized secure voting system, taking into account functional, non-functional, and administrative aspects.

The object of research is the mechanisms for organizing and securing the server component of decentralized electronic voting systems.

The subject of research is the P2P-network server software: its features, advantages, and disadvantages, as well as the technologies used to implement the server's P2P-network architecture.

The principal research methods are system requirements analysis based on the CIA model, threat and risk modelling in accordance with Ukrainian and international standards, and component-based design using Spring Boot and RocksDB.

This study examines the P2P-network architecture, the server's functional and non-functional requirements, and its administrative functions; furthermore, a prototype of the P2P-network server software for a decentralized secure voting system has been implemented.

The results of this work can be applied in pilot e-voting projects, in the development of similar distributed services, and in further research on consensus mechanisms and blockchain integration.

Keywords: P2P NETWORK, SERVER, VULNERABILITIES, SECURE VOTING, SPRING BOOT, ROCKSDB, REST API.

ЗМІСТ

ПЕРЕЛІК ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

БД – База даних

ДСТУ – Державний стандарт України

ЕГ – Електронне голосування

ПЗ – Програмне забезпечення

AOT – Ahead-of-Time compilation

CIA – Confidentiality, Integrity, Availability

CLI – Command Line Interface

DDoS – Distributed Denial of Service

HAVA – Help America Vote Act

HTTP – HyperText Transfer Protocol

HTTPS – HTTP Secure

IEC – International Electrotechnical Commission

ISO – International Organization for Standardization

JNI – Java Native Interface

JIT – Just-In-Time compiler

JSON – JavaScript Object Notation

JVM – Java Virtual Machine

LSM – Log-Structured Merge-tree

OWASP – Open Web Application Security Project

P2P – Peer-to-peer

REST API – Representational State Transfer Application Programming Interface

SQL – Structured Query Language

TLS – Transport Layer Security

UML – Unified Modeling Language

ВСТУП

У сучасних умовах цифрової трансформації дедалі більшої значущості набуває питання забезпечення надійного та прозорого процесу голосування із застосуванням електронних засобів. Традиційні централізовані рішення виявилися вразливими до зовнішніх атак, технічних збоїв, що ставить під сумнів довіру суспільства до результатів голосувань. Водночас поява нових кіберзагроз і необхідність віддаленої участі виборців у голосуванні, вимагають розробки таких підходів, які б гарантували конфіденційність даних, достовірність результатів і безперервність доступу.

Децентралізована архітектура на основі взаємодії однорангових вузлів (P2P) позбавляє систему єдиної точки відмови та знижує ризики централізованих атак. При цьому важливо не лише розподілити функції між її елементами, але й організувати централізований сервіс, який забезпечуватиме реєстрацію та управління учасниками мережі. Відсутність такого сервера ускладнює автоматизацію процесу підключення нових вузлів і підтримку їхнього актуального стану.

Метою цієї роботи є проаналізувати вимоги та спроєктувати сервер-каталог P2P-мережі для забезпечення збереження конфіденційності даних, достовірності результатів голосування та безперервності доступу до системи. Запропоноване рішення покликане забезпечити централізований контроль за реєстрацією учасників мережі, автоматизоване оновлення списків активних вузлів та надійний обмін інформацією між ними при збереженні розподіленого характеру системи.

Задачі для дослідження:

- Проаналізувати сучасний стан та перспективи розвитку систем захищеного електронного голосування.
- Здійснити системний аналіз вимог до серверів децентралізованих систем голосування.
- Обґрунтувати архітектурні підходи для побудови серверу P2P-мережі.
- Розробити архітектуру, реалізувати та протестувати сервер P2P-мережі захищеного голосування.

1 АКТУАЛЬНИЙ СТАН І ПЕРСПЕКТИВИ РОЗВИТКУ СИСТЕМ ЗАХИЩЕНОГО ЕЛЕКТРОННОГО ГОЛОСУВАННЯ

1.1. Огляд сучасних електронних систем голосування

На сьогоднішній день, існує декілька варіантів сучасних електронних систем голосування (e-voting), котрі покликані спростити і вдосконалити традиційний виборчий процес. Такі системи пропонують значно більшу швидкість обробки інформації та потенційно покращену точність, виключаючи левову частку людського фактору, водночас надаючи зручність для виборців та підвищуючи прозорість процесу голосування [5].

В загальному розумінні до систем ЕГ (електронного голосування) належать різноманітні технологічні рішення для електронного подання та підрахунку голосів, що можуть використовуватися як на дільницях, так і віддалено через інтернет [5].

Останні статистичні дані доступні на лютий 2023 року, тож за даними International IDEA [17], системи електронного голосування впроваджено в 19 % держав світу (34 із 178 досліджуваних) на національному та/або місцевому рівнях. Ще 15 % країн проводять дослідження та пілотні проєкти з перспективою майбутнього їх застосування. Водночас 6 % держав повністю відмовилися від подальшого використання таких технологій. Статистичне відображення відповідних країн наведено на рисунку 1.1 [17].

Рисунок 1.1 – Країни, що використовують чи використовували системи ЕГ

За способом реалізації виокремлюють декілька основних видів таких систем [5]:

- традиційні електронні машини. Direct Recording Electronic (DRE);
- інтернет-голосування (I-voting);
- блокчейн-голосування.

DRE (Direct Recording Electronic system) – технологія прямого електронного реєстрування голосів. Процес роботи системи відбувається у наступному порядку: виборець відмічає свій вибір на спеціальному електронному пристрої з відповідним бюлетенем, після чого даний голос зберігається у пам'яті машини, після завершення виборів із кожної машини зчитуються результати і передаються до центру підрахунку [36]. Загалом, DRE-системи можуть працювати без паперових бюлетенів, але також є режим роботи з функцією друку квитанції – так званий VVPAT (Voter-Verified Paper Audit Trail), який надає паперове підтвердження для перевірки голосу, пропонуючи додаткову перевірку коректності роботи DRE-систем [36].

Прикладом впровадження даних систем є машини Diebold AccuVote-TS, які широко використовувалися у США в 2000-х, особливо після прийняття закону HAVA (Help America Vote Act, 2002) для оновлення застарілих технологій голосування: вони записували голос лише в електронну пам'ять без паперових підтверджень. Пізніше багато штатів дообладнали такі машини модулем VVPAT або замінили на оптичні сканери паперових бюлетенів після виявлення проблем безпеки [11][15].

Наразі DRE-системи домінують серед інших технологічних рішень та експлуатуються у більшості країн світу, що використовують системи ЕГ, як із паперовим контрольним слідом виборця (VVPAT), так і без нього [17]. Систематизацію країн за типом DRE-систем наведено в табл. 1.1:

Таблиця 1.1 – Систематизація країн за типом DRE-систем

Тип системи	Країни
З VVPAT	Албанія, Болгарія, Фіджі, Індія, Іран, Мексика, Оман, Панама, Перу, Венесуела, США (окремі штати)
Без VVPAT	Бангладеш, Бутан, Бразилія, Франція, Намібія, США (окремі штати)

Інтернет-голосування (I-voting) – система дистанційного голосування, використовуючи мережу інтернет. Особливістю даної технології є можливість виборців брати участь у процесі голосування віддалено та з власного пристрою. Зазвичай, такі системи реалізовані у вигляді веб-порталу або мобільного застосунку де виборець проходить автентифікацію таємно подає свій вибір, який шифрується і передається на виборчий захищений сервер [5].

Найвідомішим прикладом імплементації систем інтернет-голосування є Естонія, яка з 2005 року проводить загальнонаціональне голосування у паперовому та електронному варіанті. З початку введення даного процесу частка виборців, що обрали варіант віддаленого голосування, лише збільшувалася і вперше на парламентських виборах 2023 року перевищила 50%. Естонська система I-voting базується на використанні ID-картки з електронним підписом для автентифікації виборця та шифрування бюлетеня; голоси надсилаються на центральний сервер і зберігаються зашифрованими до моменту підрахунку [5].

Інший приклад – Швейцарія, де починаючи з 2000-х років проводилися пілотні проекти інтернет-голосування в окремих регіонах. Зокрема, була розроблена система ЕГ під керівництвом Swiss Post, яку випробували на федеральних референдумах. Попри початкові успіхи, у 2019 році програма була призупинена після виявлення експертами критичних вразливостей, що могли дозволити непомітно підміняти голоси [25].

Важливим напрямом систем ЕГ є розробка криптографічно захищених систем з відкритим кодом. Одним з найвідоміших рішень є проект Helios – це відкрита онлайн-система голосування з підтримкою end-to-end перевірки [14]. Helios використовує гомоморфне шифрування для забезпечення таємниці бюлетенів, водночас кожен виборець отримує унікальний трекер свого голосу, за яким може пересвідчитися, що бюлетень отримано та враховано правильно. Ця система дозволяє будь-якій організації проводити вибори через веб-інтерфейс, а результати можна незалежно верифікувати спостерігачами. Helios успішно застосовувався у низці невеликих виборів – від студентських голосувань до внутрішніх виборів професійних спільнот. Зокрема, з

2010 року Міжнародна асоціація криптографічних досліджень щорічно обирає раду директорів за допомогою Helios [14][24].

Блокчейн-голосування – відносно новий тип електронних систем, що використовує технологію розподіленого реєстру (blockchain) для зберігання та підтвердження голосів. Ідея полягає у тому, що кожен відданий голос оформлюється як транзакція у блокчейні, який підтримується децентралізованою мережею вузлів. Блокчейн забезпечує незмінність (immutability) записів – після включення голосу до ланцюжка блоків його неможливо непомітно змінити або видалити. Крім того, розподілена природа може підвищити стійкість до зовнішніх атак, оскільки немає єдиного центру, який можна скомпрометувати для фальсифікації результатів [5][19][24].

Приклад академічної реалізації – система “Chaintegrity”, запропонована S. Zhang та ін. (2019) [40], що використовує приватний блокчейн для масштабного голосування з гарантією стійкості до зломів і можливістю криптографічної перевірки правильності підрахунку [40].

Таким чином, актуальний стан електронних систем голосування характеризується одночасно успішним впровадженням в окремих країнах (Естонія – інтернет-голосування, США – DRE) і обережним ставленням у світі через питання безпеки. Сучасні технології надають інструменти для створення потенційно захищеного голосування – з використанням криптографії, блокчейну, апаратних засобів безпеки. Однак реальний ступінь довіри до таких систем залежить від того, наскільки вони справляються з численними загрозами.

1.2. Ідентифікація технічних і організаційних вразливостей у захищених системах голосування

Незважаючи на очевидні переваги електронного голосування (швидкість, зручність, автоматизація підрахунку), практика його застосування виявила цілу низку проблемних аспектів, пов’язаних із безпекою і надійністю. Дійсно, навіть реальні виборчі системи, впроваджені у різних країнах, виявилися уразливими до кібератак та

збоїв, що ставить під сумнів їхню готовність повністю замінити традиційне голосування. Умовно всі проблеми можна розділити на кілька категорій: вразливості фізичних пристроїв для голосування, віддалені кібер-загрози для серверів і мережі, інсайдерські загрози, а також загальні питання фальсифікацій та довіри [5][25].

Можливість модифікації результатів – одна з головних загроз, що підриває принцип цілісності голосування. У централізованих системах існує ризик несанкціонованого втручання у сервер або базу даних голосів з боку адміністраторів чи хакерів. Так, будь-яка «маленька група» внутрішніх осіб (адміністраторів) може потенційно спотворити підрахунок голосів, якщо відсутні надійні механізми контролю. У традиційних електронних машинах для голосування відомі випадки маніпуляцій, наприклад, заміна носіїв з результатами або вставлення шкідливого ПЗ (програмного забезпечення) [11].

Фізичні уразливості електронних машин – є однією з основних вразливостей для систем DRE та інших пристроїв на виборчих дільницях, адже вони можуть стати об’єктом атаки, що потребує фізичного доступу – коли зловмисник або недобросовісний персонал безпосередньо втручається в обладнання. Дослідження показали, що багато моделей виборчих машин мають недостатній захист від несанкціонованого доступу. Наприклад, команда Принстонського університету продемонструвала, що електронну урну Diebold AccuVote-TS можна відкрити і замінити картку пам’яті протягом менш ніж однієї хвилини, не залишаючи видимих слідів [11].

У відповідь на це в багатьох юрисдикціях ввели обов’язкове застосування VVPAT: виборець після голосування бачить надрукований паперовий бюлетень-підтвердження, який падає у запечатану скриньку. Це дозволяє після виборів провести вибірковий аудит: порівняти паперові копії з електронними результатами. Якщо розбіжностей не буде – отже, машина працювала коректно. Експерти зазначають, що сучасні DRE без паперових слідів – це “чорні скриньки”, яким довіряємо наосліп, відсутність незалежної перевірки вважається серйозною вразливістю [8].

Традиційне голосування також стикається з людським фактором, проте в електронних системах інсайдер може діяти ще непомітніше. Маючи доступ до програмного коду або адмініструючи сервер, зловмисник-інсайдер здатен вбудувати потаємну функцію для викривлення результатів. Відомий у літературі приклад – так звана атака “зіткнення квитанцій” (Clash attack) на системи з верифікацією виборця. суть у тому, що корумпований член виборчого комітету видає двом різним виборцям однакові квитанції про прийняття голосу. Після завершення голосування він може замінити бюлетень одного з цих виборців на інший – в результаті один голос “зайвий”, але при перевірці обидва виборці знайдуть свої квитанції у списку (бо вони однакові). Така атака була експериментально продемонстрована проти систем Helios, Wombat та інших. Вона вимагає змови саме внутрішнього авторитету, що роздає або перевіряє квитанції, і показує, що механізми захисту можуть бути обійдені інсайдером [23].

Найбільш широкий пласт проблем стосується програмного забезпечення та мережевих з’єднань у системах електронного голосування. І DRE (особливо якщо вони об’єднані в мережу), і тим більше інтернет-голосування та системи блокчейн залежать від складного ПЗ – отже, в них потенційно можуть бути помилки або приховані розробниками експлойти. В такому разі зловмисники, не маючи фізичного доступу, можуть здійснити атаки через інтернет або інші канали зв’язку.

Підсумовуючи, досить великою проблемою виступає кібератака на програмне забезпечення. Досвід Voatz показав, що зловмисник, зламавши мобільний додаток чи канал зв’язку, міг би непомітно змінити відданий виборцем голос або видалити його [4].

Проблеми віддаленої ідентифікації та автентифікації – ще один виклик. Система повинна точно встановити особу виборця і переконатися, що голос подає саме уповноважений громадянин, і лише один раз. Вразливий механізм автентифікації (наприклад, лише за логіном/паролем) відкриває шлях до крадіжки облікових записів та голосування від імені інших осіб. Поширені загрози – фішингові сайти, які виманюють у виборців облікові дані, або атаки перехоплення СМС-кодів двофакторної авторизації [5]. Така ситуація важко виявляється, якщо система не впроваджує

додаткових перевірок (наприклад, повідомлення виборця про поданий голос). Тому питання безпечної ідентифікації – одне з найскладніших: в різних країнах пропонуються рішення від апаратних ID-карт із цифровим підписом до біометричної верифікації, але кожне має певні вразливості [5][36].

Окремо слід згадати проблему довіри та сприйняття. Навіть у разі, коли фальсифікації не відбувається, поширення інформації про можливі вразливості підриває довіру громадськості до системи. Прикладом став випадок у Швейцарії, де противники електронного голосування активно цитували звіт експертів, наголошуючи, що не можна гарантувати безпеку, поки є хоч одна знайдена вразливість – і суспільна думка схилилася проти інтернет-голосування, призводячи до повної відмови від систем ЕГ [22].

Забезпечення всіх цих вимог у одній системі є нетривіальним завданням, оскільки деякі властивості перебувають у потенційному конфлікті. Класичний приклад – одночасне гарантування повної таємності голосування і можливості незалежного аудиту результатів. Адже якщо кожен бюлетень шифрується для збереження таємниці, то ми не можемо дозволити спостерігачам пересвідчитись у правильності підрахунку. Над цією проблемою працюють концепції End-to-End верифікованого голосування, які застосовують криптографічні протоколи (наприклад, докази з нульовим розголошенням, часткове розкриття шифрів після голосування тощо) [28].

Підсумовуючи, проблемні аспекти е-голосування охоплюють цілий спектр загроз: від суто технічних (хакерські атаки на цілісність, конфіденційність, доступність) до організаційних (зловживання службовців, недовіра суспільства). Кожен з виявлених недоліків повинен бути врахований при розробці нової системи – особливо якщо ставиться мета створити децентралізовану, захищену архітектуру, здатну протидіяти як зовнішнім атакам, так і внутрішнім зловживанням.

1.3. Модель загроз у контексті систем електронного голосування

Модель загроз — це формалізований опис потенційних небезпек, які можуть вплинути на інформаційну систему, а також характеристика їхніх джерел і шляхів реалізації. Така модель класифікує загрози за різними ознаками. Одним з розповсюджених підходів є розгляд загроз за трьома основними аспектами безпеки: конфіденційність, цілісність, доступність (англ. Confidentiality, Integrity, Availability – CIA) [5][25][28].

Для систем електронного голосування всі три аспекти є критично значущими, тож розглянемо кожен окремо.

Загрози конфіденційності. Конфіденційність в контексті виборів передусім стосується таємниці голосування – гарантії, що ніхто не дізнається, як проголосував конкретний виборець. Типові загрози цього класу: перехоплення бюлетеня в мережі і розшифрування його вмісту, несанкціонований доступ до бази даних, аналітичні атаки, що дозволяють зіставити за непрямими ознаками виборця і його голос. Захист від загроз цього типу включає: стійке шифрування бюлетенів на всіх етапах, відокремлення особистих даних від даних голосування, видалення або шифрування журналів, які можуть розкрити відповідність особа-голос, регулярне тестування на відсутність каналів витоку інформації. До загроз конфіденційності можна віднести і некоректне поводження з персональними даними – наприклад, якщо система збирає надмірні дані про виборця і ті потрапляють у витік. Випадки витоку реєстрів виборців відбувалися у різних країнах, і електронне голосування не повинно їх посилити [33].

Загрози цілісності. Цілісність означає, що результати голосування мають точно відповідати поданим голосам, без жодних спотворень. Загрози цілісності – це всі можливі способи неправомірної зміни, додавання або видалення голосів. До цієї категорії належать атаки на модифікацію бюлетенів, внесення фіктивних голосів, знищення частини бюлетенів та маніпуляція підрахунком результатів. Остання заслуговує окремої уваги, оскільки навіть за наявності криптографічного захисту кожного бюлетеня зловмисник може змінити підсумкові дані шляхом підміни файлу звіту або перехоплення процедури оголошення результатів. У традиційних паперових

виборах цьому протидіє багаторівневий контроль – партійні копії протоколів і перевірка окружними комісіями. В електронних системах, для захисту від даної загрози та для досягнення достатнього рівня достовірності, пропонується використання розподілених реєстрів (blockchain) або публікацією хеш-ланцюжка всіх голосів: будь-яка навіть одинична зміна призведе до невідповідності хешу, що виявляється спостерігачами [32].

Загрози доступності. Доступність системи голосування характеризує здатність виборців безперешкодно скористатися своїм правом голосу в передбачені терміни. Атаки цього класу мають на меті зробити систему голосування недоступною для користувачів у критичний момент виборів. Зловмисники можуть здійснювати масовані DDoS-атаки на веб-портали електронного голосування або сервери підрахунку, щоб перевантажити їх і унеможливити вчасне подання чи обробку бюлетенів. Окрім кібератак на Інтернет-ресурси, загрозу доступності становлять і фізичні саботажі обладнання для електронного голосування – від навмисного пошкодження терміналів до блокування каналів зв’язку (наприклад, придушення радіосигналу смарт-карток виборців). Таким чином, забезпечення стійкості до відмов та атак на доступність є критично важливим, оскільки недоступність системи в день виборів фактично зриває виборчий процес [5].

Сформована за принципом CIA модель загроз дозволяє впорядкувати захисні механізми відповідно до мети атак. Далі для кожної загрози прописуються ймовірність та потенційний збиток, на основі чого пріоритезують захисні заходи. Зокрема, у документах Держспецзв’язку та згідно ДСТУ ISO/IEC 27005:2023 рекомендується будувати модель загроз з класифікацією за цілями атак на інформацію (порушення конфіденційності, цілісності або доступності) з оцінкою рівня ризику для кожної категорії [1][2][3].

В контексті систем ЕГ зазвичай цілісність оцінюється як критична властивість (загрози їй мають найвищий пріоритет), конфіденційність – як дуже важлива (захист таємниці голосування – конституційна вимога), а доступність – як важлива, але все ж третя за пріоритетом (бо тимчасове порушення доступності можна компенсувати

продовженням голосування, тоді як втрачену цілісність чи таємницю відновити не можна) [16].

1.4. Модель зловмисника у контексті систем електронного голосування

Потенційний зловмисник у системі захищеного голосування – це суб’єкт (особа або група), який має мотив і можливості здійснити атаки, описані в моделі загроз. Модель зловмисника визначає характерні риси такого нападника: його ресурси, знання, рівень доступу, типові методи. Відповідно до рекомендацій з технічного захисту інформації, зокрема нормативних документів Держспецзв’язку України, слід розрізняти рівні потенційного порушника (зловмисника) залежно від його можливостей. Прийнято вважати, що існують принаймні три рівні: базовий, середній та високий [2][3].

Базовий рівень. Зовнішній хакер-одинак або невелика група без інсайдерського доступу, обмежена у ресурсах і не здатна розробляти нові експлойти з нуля. Типові атаки: фішинг, DDoS, спроби bruteforce для підбору паролів, SQL-ін’єкції або повторне голосування з різних акаунтів. Захисні механізми проти такого нападника — строгі політики паролів, захист від ін’єкцій, базова фільтрація введення та моніторинг аномалій.

Середній рівень. Група зловмисників або окремий фахівець із професійними інструментами, які можуть мати частковий інсайдерський доступ, володіють досвідом розробки складних багатокрокових атак, 0-day експлойтів або можливістю тимчасово отримати фізичний доступ до обладнання. До цієї категорії належать організовані хакерські колективи без прямої підтримки держави. Вони здатні компрометувати сертифікати, інфікувати пристрої виборців через ланцюжки постачання ПЗ або проводити таргетовані атаки на серверну інфраструктуру. Саме протидія нападнику 2-го рівня вважається базовим завданням при проєктуванні захищених систем електронного голосування [2].

Високий рівень. Держава чи організації з практично необмеженими ресурсами: доступ до унікальних криптографічних або квантових засобів, можливість завчасно

впровадити експлойти на стадіях розробки, здійснювати апаратні атаки на елементи системи або координувати складні атаки. Протидія такому рівню часто вимагає радикальних заходів (повної ізоляції, паперових резервних процедур, залучення міжнародних спостерігачів) та розглядається як залишковий ризик.

У межах цього дослідження основну увагу приділено моделюванню зловмисника 2-го рівня, оскільки він відображає найбільш імовірні та масштабні загрози для систем електронного голосування. При такому підході забезпечується достатній рівень безпеки для захисту від більшості реальних кіберзагроз.

1.5. Напрями розвитку захищеного електронного голосування

Врахувавши актуальний стан і виявлені проблеми, можна окреслити напрями вдосконалення систем електронного голосування. Світовий досвід і сучасні дослідження вказують, що майбутнє таких систем пов’язане з децентралізацією, криптографічними протоколами нового покоління та підвищенням прозорості процесу голосування.

Перспективи розвитку у цій сфері пов’язані як з технологічними інноваціями, так і з організаційними змінами, що підвищують довіру та надійність процесу голосування, тож розглянемо основні напрями:

- Підвищення рівня конфіденційності. У відповідь на загрози конфіденційності, майбутнім системам необхідно впроваджувати ще потужніші методи захисту приватності виборця. Один з напрямів – протоколи сліпого підпису при видачі електронних бюлетенів: коли виборець отримує право голосу, сервер підписує токен, не знаючи його вмісту, і цей токен потім використовується для голосування. Інший напрям – диференційовані мережі доставки: використання мереж типу Tor або приватних маршрутів для передачі бюлетенів, щоб унеможливити відстеження голосів [21].
- Прозорість та відкритий код. Вимога, що дедалі більше визнається обов’язковою, – максимальна прозорість системи для зовнішнього аудиту. Це означає, що вихідний код програмного забезпечення системи голосування має

бути відкритим (або доступним під наглядом незалежних експертів) ще на етапі розробки і тестування. Звісно, повна відкритість не означає розкриття секретних ключів чи персональних даних – конфіденційні елементи лишаються захищеними, але їхнє використання проводиться під контролем [14][24].

- Впровадження блокчейн-технологій. Технологія блокчейн вважається одним із найперспективніших рішень для забезпечення цілісності та прозорості електронного голосування. Так як блокчейн являє собою розподілений реєстр, що зберігається одночасно на багатьох вузлах і захищений криптографічними методами, то завдяки цьому будь-які спроби несанкціонованої зміни даних стають надзвичайно складними, оскільки потребують одночасного компрометування більшості вузлів мережі [5][37]. Прозорість блокчейну також відкриває нові можливості для відкритого аудиту виборів: усі зашифровані голоси публічно доступні в реєстрі, і незалежні спостерігачі чи виборці можуть самостійно переконатися в їх наявності та підрахунку (не розкриваючи, втім, самих результатів окремих голосувань) [37][40].
- Також актуальним є механізм делегованого (liquid democracy) голосування, як це пропонується в сучасних системах блочейн-прийняття рішень [39]. Тобто кожен користувач може або віддати голос напряму за кандидата (що забезпечує безпосередню участь), або делегувати свій голос довіреній особі (експерту), яка проголосує від імені свого пулу виборців. Такий підхід, за словами дослідників, підвищує «колективний інтелект» системи – збільшує гнучкість процесу та залученість громади, не шкодячи конфіденційності голосування [39].

Особливий акцент слід зробити на потенціалі P2P-реалізації. Peer-to-peer мережі, на відміну від клієнт-серверних, не мають жорсткої ієрархії – кожен повний вузол є рівноправним. Така архітектура добре узгоджується з концепцією блокчейну, де кожен вузол виконує роль валідатора транзакцій (у нашому випадку – голосів). Також P2P-система потенційно краще масштабована географічно: якщо одна область тимчасово ізольована, вузли можуть локально зберігати голоси і синхронізуватися,

щойно зв’язок відновиться. Це робить систему значно стійкішою до проблем з доступністю і цілісністю інформації.

У процесі аналізу систем P2P-реалізації захищеного голосування було виокремлено низку ключових проблем, вирішення яких є необхідним для поширення тенденцій використання систем ЕГ. По-перше, відсутність централізованого каталогу вузлів P2P-мережі призводить до того, що нові учасники не можуть отримати відомості про активні вузли для встановлення зв’язку [38]. Така архітектура дозволить спрощувати реалізацію вузлів, адже вони отримуватимуть весь перелік активних учасників мережі з відповідного серверу. По-друге, є необхідність управління переліком вузлів мережі, який постійно змінюється, - це ускладнює підтримку актуальності інформації вручну і значно підвищує вимоги до автоматизації даних процесів у реалізації вузлів [19]. По-третє, наявні лише обмежені засоби керування сервером та адміністрування (додавання/видалення вузлів, налагодження) [7].

З урахуванням виявлених проблем і поточних тенденцій, для подальшого виконання роботи основною метою є аналіз вимог та проєктування серверу-каталогу, що забезпечує централізовану реєстрацію та управління вузлами P2P-мережі, надає програмний і командний інтерфейси для роботи з мережею, а також зберігає та оновлює інформацію про учасників.

2 АНАЛІЗ ВИМОГ ЩОДО ПОБУДОВИ ЗАХИЩЕНИХ СЕРВЕРІВ ДЕЦЕНТРАЛІЗОВАНИХ СИСТЕМ ГОЛОСУВАННЯ

2.1. Визначення функціональних вимог до серверного компонента P2P-мережі електронного голосування

Функціональні вимоги описують конкретні функції та сервіси, які повинна реалізовувати система голосування. Серед основних функціональних вимог до захищених серверів децентралізованої системи голосування можна виділити наступні пункти:

1) Реєстрація та автентифікація виборців. Сервер-каталог має забезпечувати реєстрацію вузлів з перевіркою дозволу, тобто перевіряє цифрові підписи чи сертифікати при додаванні нового вузла. Кожен вузол мережі ініціює запит на реєстрацію з власними обліковими даними (наприклад, сертифікатом чи токеном). Сервер підтверджує легітимність і додає вузол до активного списку. Подібний підхід реалізований, наприклад, у Kubernetes: kubelet автоматично реєструє себе у головному API-сервері за допомогою наданих облікових даних. Якщо перевірка успішна, вузол додається до локального каталогу [27].
2) Надання списку кандидатів. Після автентифікації сервер надає виборцю актуальний список кандидатів або питань для голосування аби забезпечити доступність голосування для всіх категорій виборців [20].
3) Аудит. Сервер-каталог для P2P-мережі має містити вбудоване сховище для інформації про вузли (IP, порт, статус тощо). У якості сховища може використовуватись легка вбудована база даних що забезпечує швидкий доступ [13].
4) Управління вузлами. Для підтримки зв’язку в мережі необхідно забезпечити механізм періодичних «heartbeat»-повідомлень, які вказують на активність вузла, з оновленням мітки останньої активності. Якщо heartbeat не надходить у встановлений інтервал, сервер позначає вузол як недоступний. Це дозволяє відстежувати доступність учасників мережі й відкидати неактивні вузли [41].

Усі зібрані адреси та статуси вузлів мають зберігатися у вбудованій базі даних для швидкого доступу та подальшого обміну з іншими вузлами.

5) Подання голосу та підтвердження. Сервер також має виконувати ключові функції голосування. Серед них – прийом голосів від учасників (ідентифікованих за унікальним ID), перевірка права на голосування та унікальності подачі голосу, а також підрахунок голосів. Наприклад, запит на голосування приймає дані про учасника та обраного кандидата: система перевіряє, що учасник існує (чи створює його нового) і ще не голосував, після чого фіксує голос та оновлює лічильник голосів кандидата. Вимога «один учасник – один голос» забезпечується через перевірку вже записаного поля учасника, що не дозволяє повторного голосування [19][28]. Підрахунок результатів має реалізуватися простим запитом, який повертає перелік кандидатів із накопиченими голосами. Таким чином, сервер гарантує цілісність голосів і відтворюваність результатів на основі надійного журналу транзакцій.
6) Робота з нодами мережі. Сервер P2P-системи голосування повинен реалізовувати такі базові сервіси, щоб забезпечити безпечний розподілений процес голосування. Передусім, він має надавати API для реєстрації вузлів (нод), обміну інформацією про підключення та ведення списку активних хостів. Зокрема, нові вузли можуть надсилати запити на реєстрацію, вказуючи свою IP-адресу чи доменне ім’я, а сервер зберігає їх у локальній базі і видає унікальний ідентифікатор.

Усі функціональні вимоги зорієнтовані на реалізацію P2P-мережевого взаємодіяння без єдиного централізованого вузла. Наприклад, сервер повинен підтримувати одночасне підключення декількох пірів, автоматично реагуючи на появу та зникнення вузлів. Кожен вузол веде копію журналу голосувань і обмінюється нею з іншими пірами, створюючи узгоджений розподілений реєстр. Вимога цілісності записів та відсутності подвійного голосування забезпечується криптографічними методами [38]. У цілому сервіс повинен забезпечити надійну реєстрацію учасників та

безпечний розподіл їхніх голосів, реалізовуючи перелічені функціональні вимоги у відповідності до технічного стека.

2.2. Визначення нефункціональних вимог до серверного компонента P2P-мережі електронного голосування

Нефункціональні вимоги описують загальні властивості системи, що суттєво впливають на її архітектуру та вибір технологій.

Сервер-каталог повинен відповідати класичній CIA-моделі інформаційної безпеки – Confidentiality (конфіденційність), Integrity (цілісність) та Availability (доступність). Конфіденційність означає захист даних від несанкціонованого доступу: наприклад, облікові дані адміністраторів і списки учасників голосування повинні передаватися по захищеному каналу і зберігатися у зашифрованому вигляді. Деякі дослідження електронного голосування підкреслюють важливість захисту особистої інформації виборців. Цілісність гарантує неможливість несанкціонованого змінення даних – наприклад, система повинна виявляти та відхиляти спроби фальсифікації адрес вузлів чи результатів опитування. Вимога доступності вимагає, щоб сервер-каталог залишався працездатним навіть у разі відмов частини системи (наприклад, за рахунок реплікації або аварійного перемикання) [26][28].

Продуктивність і масштабованість. Сервер-каталог повинен швидко обробляти вхідні запити від вузлів і клієнтів. Для цього варто зосередити вибір платформи для розробки архітектури на високопродуктивні потокові сервіси. Окремо, необхідна масштабованість, щоб у разі збільшення кількості вузлів або клієнтів система повинна зберігати прийнятні параметри швидкодії.

Надійність і відмовостійкість. Система повинна бути надійною та стійкою до збоїв. Наприклад, відсутність єдиної точки відмови – за рахунок реплікації даних і моніторингу стану вузлів (heartbeat) [41].

2.3. Адміністративні функції серверу P2P-мережі електронного голосування

Крім голосування, сервер повинен надавати функції керування виборчим процесом. До переліку адміністративних функцій входить управління списком

кандидатів і адміністраторами. Наприклад, наявна роль адміністратора може додавати нових кандидатів, задаючи їм унікальні ідентифікатори та ініціалізуючи їх лічильники голосів. Аналогічно, сервер має дозволяти додавання та видалення облікових записів адміністраторів. Ці адміністративні функції розширюють функціональність ядра голосування і забезпечують гнучке керування виборчим процесом.

Сервер-каталог реалізує розмежування ролей та прав доступу. Є щонайменше дві ролі – адміністратора і звичайного вузла/клієнта. Адміністратор може виконувати будь-які дії (додавання адміністраторів, управління учасниками та вузлами). Контроль доступу реалізується через перевірку токенів або сертифікатів у заголовках запитів або за обліковими записами.

2.4. Аналіз ризиків, обмежень та обґрунтування компромісів.

Проєктування архітектури серверу-каталогу для системи децентралізованого захищеного голосування неминуче пов’язане з низкою ризиків та обмежень, обумовлених як принципами роботи децентралізованих систем, так і реальними умовами розгортання та експлуатації. Незважаючи на високі цілі щодо забезпечення конфіденційності, цілісності та доступності, на практиці розробники часто змушені шукати компроміс між безпекою, продуктивністю, надійністю та простотою впровадження.

Загальні ризики при реалізації децентралізованих систем:

1) Ризик несанкціонованого доступу та компрометації даних

   Будь-який компонент системи, що зберігає або обробляє критичну інформацію, може стати ціллю атак. Це стосується як облікових даних користувачів, так і списків вузлів або результатів голосування. Ризики підсилюються за відсутності належного шифрування даних під час зберігання та передачі [19].

2) Втрата доступності через перевантаження або цілеспрямовані атаки

   Сервер-каталог може стати єдиною точкою відмови у випадку централізованої реалізації. Висока залежність усіх вузлів і клієнтів від цього компоненту створює загрозу зупинки всієї системи у разі його збою.

3) Ризик некоректної синхронізації між вузлами

   Децентралізовані системи потребують узгодженості стану між усіма учасниками. Відсутність узгодженого механізму консенсусу або слабка координація можуть призвести до розбіжностей у стані мережі, втрати або дублювання інформації.

4) Недостатня перевірка достовірності інформації про вузли

   Простий механізм перевірки наявності вузлів без криптографічного підтвердження їхньої автентичності може створити умови для атак з підміною (Man-in-the-Middle), фальшивими вузлами або саботажем з боку зловмисників [7].

Компроміси, прийняті при побудові, які можна розглядати при подальшому розвитку системи:

1) Компроміс між продуктивністю і конфіденційністю

   Задля забезпечення високої швидкодії обробки запитів система на початковому етапі реалізації може зберігати дані у відкритому вигляді на локальних вузлах без додаткового шифрування. Це спрощує впровадження і підвищує продуктивність, однак знижує рівень захисту у разі фізичного доступу до обладнання.

2) Компроміс між централізованим керуванням і відмовостійкістю

   Вибір централізованої архітектури серверу-каталогу на ранньому етапі дозволяє спростити розгортання та управління системою. Проте це створює залежність від одного центрального компоненту, що потребує подальшого вдосконалення через реплікацію або розподіл навантаження.

3) Компроміс між простотою перевірки стану вузлів і їхньою достовірністю

   Система використовує базовий механізм перевірки доступності вузлів (heartbeat) без криптографічного захисту. Це дозволяє швидко реалізувати базову функціональність, але залишає можливості для атак з підміною вузлів або перехоплення трафіку.

4) Компроміс між гнучкістю та узгодженістю даних при масштабуванні

Система не впроваджує повноцінний механізм консенсусу між кількома екземплярами серверу-каталогу. Це спрощує розгортання однієї копії, але створює ризики неузгодженості при масштабуванні або при використанні кількох серверів.

2.5. Висновки до розділу

У даному розділі було здійснено системний аналіз вимог до серверного компонента децентралізованої системи захищеного голосування. Встановлено шість основних функціональних сервісів:

- реєстрація та автентифікація вузлів;
- розподіл списку кандидатів;
- зберігання даних про вузли та події у вбудованій базі даних;
- моніторинг активності переліку вузлів;
- прийом і підрахунок голосів із гарантією «один учасник – один голос»;
- управління ролями та вузлами через адміністративне API.

Нефункціональні вимоги, сформульовані за моделлю CIA (конфіденційність, цілісність, доступність), окреслили критичні напрями розвитку: шифрування даних і каналів, контроль цілісності транзакцій та зберігання даних. Поряд із цим приділено увагу продуктивності, масштабованості й стійкості до збоїв.

Адміністративні функції підкреслили необхідність чіткого розмежування прав: адміністратор повинен мати можливість керувати кандидатами й вузлами, тоді як звичайні клієнти обмежені стандартними операціями.

Окремо проаналізовано ризики – від загроз несанкціонованого доступу та атак до можливих розбіжностей стану між пірами. Запропоновано компроміси між швидкодією та рівнем захисту, централізованим керуванням і відмовостійкістю.

Загалом, встановлені вимоги й компроміси формують надійну основу для проєктування серверу-каталогу. Отримані результати задають чіткі орієнтири для розробки архітектури та вибору технологій, що буде детально представлено в наступному розділі.

3 АНАЛІЗ І ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЙ ПОБУДОВИ ЗАХИЩЕНОГО СЕРВЕРУ ДЕЦЕНТРАЛІЗОВАНОЇ СИСТЕМИ ГОЛОСУВАННЯ

3.1. Оцінка P2P-архітектури як фундаменту децентралізованого електронного голосування

Однорангова (peer-to-peer) мережа є розподіленою архітектурою, в якій кожен вузол одночасно виступає клієнтом і сервером, маючи рівні права й функції. На відміну від класичної клієнт-серверної моделі, де центральний сервер відповідає за координацію ресурсів, у P2P-мережах кожен вузол безпосередньо обмінюється повідомленнями з іншими. Такий підхід забезпечує відсутність єдиної точки відмови – збої одного вузла не призводять до повного відключення системи [12][38]. До того ж у P2P-мережах підвищується масштабованість: при збільшенні числа учасників загальна пропускна здатність мережі зростає, адже кожен новий вузол додає власні ресурси (дисковий простір, обчислювальні потужності тощо). Приклад топології даної мережі показаний на рисунку 3.1. Зокрема, у традиційній клієнт-серверній системі навантаження зазвичай концентрується на сервері, і зростання числа клієнтів може призвести до його перевантаження, тоді як у P2P-мережі навантаження розподіляється між учасниками рівномірно [12][38].

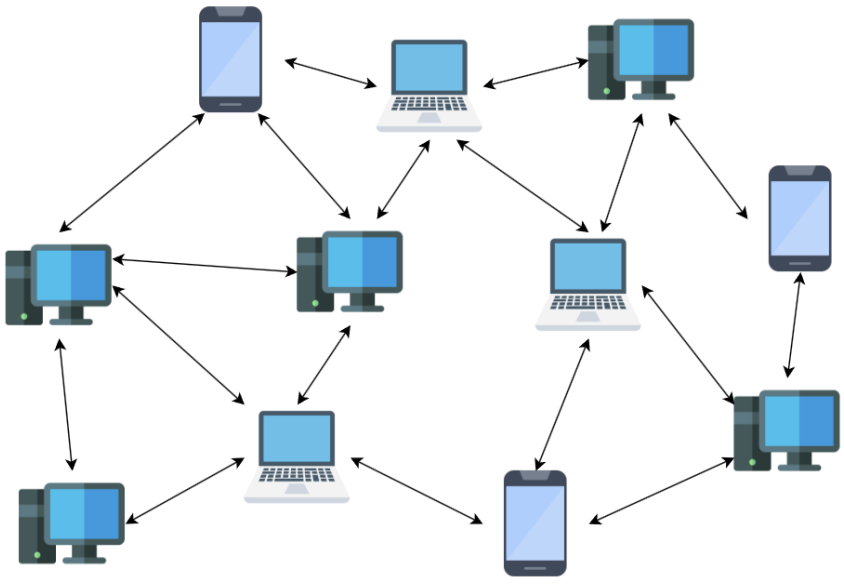

Рисунок 3.1 – Приклад топології розподіленої P2P-мережі

3.2. Обґрунтування технологічного стеку для побудови серверу

Для реалізації серверної частини обрано Java із фреймворком Spring Boot. Цей вибір обґрунтовано кількома чинниками:

1) Продуктивність і масштабованість. Java з JIT-компіляцією забезпечує високу швидкодію сучасних сервісів. Spring Boot оптимізований за роки розвитку і пропонує великий вибір налаштувань для тонкої оптимізації (конфігурація JVM, пул з'єднань, кешування тощо). За результатами незалежних тестів, Quarkus (альтернативний фреймворк) має швидший старт і менше споживає пам'ять порівняно з Spring Boot, але Spring Boot демонструє високу пропускну здатність завдяки глибокій оптимізації сервісного шару. Micronaut, навпаки, використовує передкомпіляцію (AOT) для зменшення накладних витрат, що дає йому перевагу в часі старту, але Spring Boot залишається досить швидким у звичайних контейнерних середовищах [20].
2) Екосистема та простота використання. Spring Boot підтримується розвиненою екосистемою проєктів Spring (Spring Data, Spring Security, Spring Cloud тощо) та має активну спільноту. Це означає великий вибір готових рішень, динамічні оновлення і багату документацію [34].
3) Сумісність із REST. Spring Boot «з коробки» надає повноцінну підтримку REST через Spring MVC/WebFlux. Наприклад, розробка REST-контролера з Spring Boot проста та добре докуменована (контролер можна позначити @RestController і методами-запитами @GetMapping, @PostMapping тощо). Це спрощує стандартизований обмін даними (JSON, HTTP-методи) і робить сервер сумісним із будь-якими REST-клієнтами [34].

3.3. База даних і зберігання даних

Для зберігання даних вибрано RocksDB – вбудоване key-value сховище від Facebook. Воно призначене для надшвидкого доступу на флеш-носіях і підтримує великий обсяг операцій. Основні переваги RocksDB: висока швидкість читання/запису, відсутність потреби в окремому сервері та проста інтеграція з Java.

RocksDB написано на C++ з JNI-інтерфейсом для Java, тому може бути підключене як бібліотека (org.rocksdb:rocksdbjni) у Spring Boot-проєкті. Важливо, що RocksDB не вимагає серверної частини, бо БД (база даних) розгортається прямо в процесі сервера, що ідеально для мікросервісного розгортання [9].

Продуктивність (бенчмарки). Згідно з офіційними результатами тестів (Facebook, in-memory workload), при інтенсивному записі близько 80 тис. операцій на секунду і розмірі запису ~10 МБ/с система забезпечувала ~4,55 млн точкових читань за секунду (близько 0,22 мкс/операцію). При цьому фактична тривала швидкість запису склала ~52 тис. записів/с (приблизно 9,6 МБ/с). Наведені дані свідчать про дуже високу пропускну здатність: наприклад, при навантаженні 80K записів у секунду середня швидкість читання становила лише 0.220 мікросекунди [10].

На практиці RocksDB показує таку високу швидкодію завдяки LSM-архітектурі (Log-Structured Merge-tree), оптимізаціям під flash-пам’ять та канонічному використанню Bloom-фільтрів. Ці результати демонструють, що для нашої системи голосування, де очікується багатоопераційне оновлення стану (прибираються/додаються голоси), RocksDB забезпечить низьку затримку і високу пропускну здатність [9].

3.4. Інтерфейси взаємодії: REST і CLI

Для основної взаємодії системи обрано REST API. Це стандартний підхід у сучасних розподілених додатках, що використовує HTTP як транспортний протокол. REST відзначається сумісністю та стандартизованістю: клієнтські запити можуть бути відправлені з будь-яких мов і платформ, а відповіді формуються у зрозумілому форматі (JSON). REST повністю відповідає архітектурним принципам мікросервісів (stateless, використання HTTP-методів, однакова інтерфейсна угода) [6].

CLI (командний інтерфейс) використовується як додатковий інтерфейс для адміністратора системи. CLI-інтерфейс може бути реалізований як консольний застосунок або окремий модуль, який відправляє команди серверу (через стандартизований інтерфейс або внутрішні класи). CLI зручний для:

- Адміністрування вузлів. Налаштування параметрів сервера, запуск або зупинка вузла, оновлення конфігурацій.
- Моніторингу і діагностики. Перевірка стану мережі, статистики запитів, логів та інших діагностичних даних.
- Обслуговування бази. Імпорт/експорт даних, бекап/відновлення RocksDB через команди CLI.
- Налаштування безпеки. Виконання команд по зміні прав доступу, ротації ключів, управління сертифікатами.

CLI є більш закритим інтерфейсом (доступним тільки адміністратору чи скриптам) і дозволяє виконувати складні службо-модульні задачі, що не призначені для кінцевого користувача. CLI-програми зазвичай споживають дані з REST API чи безпосередньо з внутрішнього сервісного шару (через виклики методів сервера).

Таким чином, взаємодія виглядає так: REST-клієнти (наприклад, фронтенд чи інші системи) надсилають HTTP-запити на REST-ендпоінти сервера. Серверна частина Spring Boot приймає запити через контролери, які обробляють їх у сервісному шарі і звертаються до RocksDB або внутрішньої логіки. CLI-утиліта використовує внутрішні сервіси сервера або окремий протокол (консольні методи), щоб виконувати адміністраторські команди. Обидва інтерфейси працюють з одним ядром сервера і однією базою даних, відрізняючись лише способом доступу та набором функцій.

У висновку, такий поділ ролей – REST для зовнішнього користувача та CLI для внутрішніх задач – є стандартним підходом. REST обрана через її універсальність і простоту інтеграції, а CLI – заради зручності взаємодії адміністратора і гнучкості при обслуговуванні системи.

3.5. Загальні принципи та засоби реалізації протоколів безпеки в сервері

Безпека є критичною для системи електронного голосування. Саме тому, навіть у тестовому варіанті серверу-каталогу мають бути реалізовані деякі початкові заходи безпеки, тож відзначимо загальні аспекти, характерні для подібних систем:

1) Автентифікація адміністратора. Необхідно створити простий метод автентифікації, що захищає адміністративні операції (додавання/видалення адміністраторів і кандидатів) від несанкціонованого доступу.
2) Цілісність і достовірність даних. Усі операції з голосами та результатами виконуються в єдиному сховищі. Код серверу має гарантувати це хоча б механічно (дані змінюються лише передбаченим шляхом через контролер). Для підвищення безпеки можна використовувати криптографічний захист: додавання електронного підпису кожного голосу чи зберігання хешів даних.
3) Конфіденційність каналу зв'язку. У продуктивному середовищі необхідно використовувати захищені з'єднання (HTTPS/TLS) для запобігання підслуховуванню чи підробці запитів. При наявності цифрових сертифікатів і HTTPS сервер гарантує конфіденційність та цілісність переданих JSON-запитів, тож для захисту REST-API вибрано протокол HTTPS (HTTP поверх TLS), оскільки Transport Layer Security (TLS) є галузевим стандартом шифрування каналів передачі даних [30]. TLS використовується як єдиний протокол захисту, бо він повністю задовольняє вимоги конфіденційного та цілісного каналу зв'язку. За відсутності TLS HTTP-трафік передавався б відкритим текстом, що робить можливими атаки прослуховування та «людина посередині» (MITM), коли зловмисник може перехопити логіни, сесійні ключі чи змінити дані при передаванні. Саме тому OWASP рекомендує надавати REST-сервіси виключно через HTTPS, що захищає дані в транзиті та гарантує цілісність переданих повідомлень [30].

3.6. Висновки до розділу

У даному розділі було обґрунтовано набір технологій для реалізації серверу захищеної децентралізованої системи голосування. Вибір P2P-мережі як комунікаційної моделі гарантує відсутність єдиної точки відмови та підвищує масштабованість системи за рахунок прямого обміну повідомленнями між вузлами. Для розробки серверної частини обрано Java із Spring Boot – адже це рішення

забезпечує високу продуктивність, зручність конфігурування й широку екосистему наявних модулів, а також підтримку RESTful API.

У якості сховища даних було прийнято використовувати RocksDB – вбудоване key-value рішення, що працює без окремого сервісу, демонструє гарні показники читання/запису на секунду й добре інтегрується в Java-проєкт через JNI. REST API обрано для відкритої взаємодії клієнтів і вузлів, що відповідає принципам мікросервісів, а CLI слугує внутрішнім інтерфейсом для адміністрування та діагностики.

Ключові протоколи безпеки включають TLS/HTTPS для захищеного каналу, просту схему автентифікації адміністратора через відповідні ключі.

Сукупність обраних технологій формує надійний технологічний стек, який відповідає вимогам CIA-моделі [26], забезпечуючи високу швидкодію при масштабуванні та дозволяючи у подальшому розширювати функціональність системи голосування. У наступному розділі буде докладно розглянуто реалізацію цих компонентів на рівні коду та конфігурацій.

4 РОЗРОБКА АРХІТЕКТУРИ, ПОБУДОВА І ТЕСТУВАННЯ СЕРВЕРУ P2P-МЕРЕЖІ ДЕЦЕНТРАЛІЗОВАНОЇ СИСТЕМИ ЗАХИЩЕНОГО ГОЛОСУВАННЯ

4.1. Компонентна архітектура системи

Серверний застосунок реалізовано на основі платформи Spring Boot і має компонентну архітектуру, що включає контролери веб-API, сервіс ініціалізації даних та репозиторій для зберігання даних у вбудованій базі RocksDB. Кожен контролер відповідає за певний аспект функціональності: AdminController – для адміністративних операцій, VotingController – для процесу голосування, NodeController – для взаємодії вузлів P2P-мережі. Контролери позначені анотацією @RestController, що вказує на їхню роль обробників HTTP-запитів у RESTful веб-сервісі [31].

Взаємодія контролерів із базою даних відбувається через компонент RocksDBRepository, який інкапсулює операції з key-value сховищем RocksDB. На рис. 4.1 зображено UML-діаграму компонентів серверного застосунку: контролери звертаються до репозиторію, який зберігає дані (адміністратори, учасники, кандидати, вузли) у RocksDB. Для спрощення архітектури бізнес-логіка зосереджена безпосередньо в контролерах, а репозиторій виконує роль шару доступу до даних.

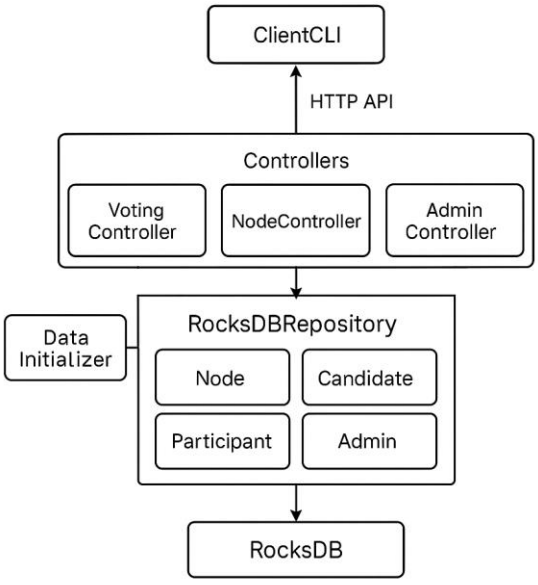

Рисунок 4.1 – Компонентна архітектура серверного додатка

Як показано на діаграмі, на рівні веб-інтерфейсу працюють три основні контролери: AdminController, VotingController та NodeController. Контролери отримують необхідні залежності (сервіси або репозиторій) через механізм впровадження залежностей Spring. У даній реалізації для спрощення бізнес-логіка може виконуватися безпосередньо в методах контролерів, проте логічно виділено відповідні сервіси AdminService, VotingService та NodeService, які покликані обробляти запити та взаємодіяти з рівнем даних. Нижче наведено призначення та взаємодію кожного компонента архітектури.

AdminController. Контролер AdminController обробляє запити за префіксом /admin і забезпечує функції адміністрування системи: додавання нових адміністраторів, внесення кандидатів для голосування, видалення адміністратора тощо. Для захисту цих операцій використовується механізм API-ключа: кожен запит до /admin-ендпоінтів повинен містити заголовок X-Admin-Key із секретним ключем адміністратора (конфігурованим через параметр app.adminKey). Контролер перевіряє цей ключ у методі isAdminAuthorized(), щоб дозволити або заборонити виконання адміністративної команди. У випадку невірного чи відсутнього ключа повертається статус 403 (Access Denied). Наведений нижче фрагмент коду показує метод addCandidate із AdminController, який дозволяє адміністратору додати нового кандидата до системи:

```
@PostMapping("/addCandidate")
public ResponseEntity<String> addCandidate(@RequestHeader(value = "X-
Admin-Key", required = false) String key,
@RequestBody CandidateRequest req) {
    if (!isAdminAuthorized(key)) {
        return ResponseEntity.status(403).body("Access denied");
    }
    String name = req.getName();
    if (name == null || name.isEmpty()) {
        return ResponseEntity.badRequest().body("Candidate name is
empty");
    }
```

```
    // Генеруємо новий ID для кандидата – використовуємо поточний час
    String candidateId = String.valueOf(System.currentTimeMillis());
    Candidate candidate = new Candidate(candidateId, name);
    repository.save("candidate:" + candidateId, candidate);
    System.out.println("Candidate added: " + candidate);
    return ResponseEntity.ok("Candidate added with ID=" + candidateId);
}
```

VotingController. Клас VotingController реалізує основні REST-ендпоінти для процесу голосування. Він не потребує спеціальної авторизації (голосують звичайні користувачі/учасники), тому його методи доступні без додаткових заголовків. Основні операції – це подання голосу (POST /vote) та отримання результатів голосування (GET /results). Метод vote() приймає дані голосу (ідентифікатор виборця та кандидата) у форматі JSON і забезпечує такі кроки:

1) Перевіряє наявність обох ідентифікаторів у запиті;
2) Отримує учасника з бази або створює нового, якщо такий відсутній;
3) Перевіряє, чи цей учасник вже голосував раніше (щоб гарантувати правило «один учасник – один голос»);
4) Знаходить кандидата за переданим ID;
5) Реєструє голос – встановлює за ким проголосував учасник та інкрементує лічильник голосів кандидата;
6) Зберігає оновлені дані учасника і кандидата до репозиторію;
7) Повертає підтвердження про успішний запис голосу.

Фрагмент коду нижче демонструє реалізацію цього процесу, а на рис. 4.2 зображено відповідний процес фіксації голосування:

```
@PostMapping("/vote")
public ResponseEntity<String> vote(@RequestBody VoteRequest req) {
    String participantId = req.getParticipantId();
    String candidateId = req.getCandidateId();
    if (participantId == null || candidateId == null) {
        return ResponseEntity.badRequest().body("participantId or
candidateId missing");
```

```
    }
    // 1. Отримуємо або створюємо учасника
    Participant participant;
    Optional<Object> partOpt = repository.get("participant:" +
participantId);
    if (partOpt.isPresent()) {
        participant = (Participant) partOpt.get();
        // Якщо цей учасник вже голосував, забороняємо голосування
        if (participant.getVotedFor() != null) {
            return ResponseEntity.status(409).body("Participant has
already voted");
        }
    } else {
        // Якщо учасника не існує, створюємо нового
        participant = new Participant(participantId);
    }
    // 2. Знайти кандидата
    Optional<Object> candOpt = repository.get("candidate:" +
candidateId);
    if (!candOpt.isPresent()) {
        return ResponseEntity.status(404).body("Candidate not found");
    }
    Candidate candidate = (Candidate) candOpt.get();
    // 3. Зафіксувати голос
    participant.setVotedFor(candidateId);
    candidate.setVotes(candidate.getVotes() + 1);
    repository.save("participant:" + participantId, participant);
    repository.save("candidate:" + candidateId, candidate);
    return ResponseEntity.ok("Vote recorded");
}
```

```
PS C:\Users\dotse\IdeaProjects\VoteServerCatalogue> mvn -q exec:java "-Dexec.args=results"
Voting results:
- Alice (ID=1): 0 votes
- Bob (ID=2): 0 votes
- Charlie (ID=3): 0 votes
PS C:\Users\dotse\IdeaProjects\VoteServerCatalogue> mvn -q exec:java "-Dexec.args=vote voter2 1"
HTTP 200
Vote recorded
PS C:\Users\dotse\IdeaProjects\VoteServerCatalogue> mvn -q exec:java "-Dexec.args=vote voter3 1"
HTTP 200
Vote recorded
PS C:\Users\dotse\IdeaProjects\VoteServerCatalogue> mvn -q exec:java "-Dexec.args=vote voter1 2"
HTTP 200
Vote recorded
PS C:\Users\dotse\IdeaProjects\VoteServerCatalogue> mvn -q exec:java "-Dexec.args=results"
Voting results:
- Alice (ID=1): 2 votes
- Bob (ID=2): 1 votes
- Charlie (ID=3): 0 votes
PS C:\Users\dotse\IdeaProjects\VoteServerCatalogue>
```

Рисунок 4.2 – Процес фіксації голосування

NodeController. Контролер NodeController відповідає за реєстрацію вузлів у децентралізованій мережі та підтримку їх актуального статусу. Він оперує префіксом /nodes для групування пов’язаних ендпоінтів. Основні функції: реєстрація нового вузла (POST /nodes/register), прийом періодичних сигналів активності – heartbeat (POST /nodes/heartbeat), та перегляд списку відомих вузлів (GET /nodes). Реєстрація вузла виконується відкрито (може містити перевірку авторизації або сертифікатів у розширеному варіанті, але в базовій реалізації достатньо прийняти запит). При реєстрації сервер генерує для вузла унікальний nodeId (використовується випадковий UUID), фіксує його IP-адресу або доменне ім’я та порт. Ці дані об’єднуються у об’єкт Node, який одразу зберігається в репозиторії. Метод повертає клієнту повну інформацію про зареєстрований вузол (включно з присвоєним id). Код нижче демонструє реалізацію реєстрації вузла, також це процес відображений на рис. 4.3:

```
@PostMapping("/register")
public ResponseEntity<Node> registerNode(@RequestBody
NodeRegistrationRequest req) {
    try {
        String nodeId = UUID.randomUUID().toString();
        String ip = req.getIp();
        String domain = req.getDomain();
```

```
            int port = req.getPort();
            if (domain != null && !domain.isEmpty()) {
                // Якщо надано домен, отримуємо IP з доменного імені
                InetAddress addr = InetAddress.getByName(domain);
                ip = addr.getHostAddress();
            }
            Node node = new Node(nodeId, ip, domain, port);
            repository.save("node:" + nodeId, node);
            System.out.println("Register new node: " + node);
            return ResponseEntity.ok(node);
        } catch (Exception e) {
            e.printStackTrace();
            return ResponseEntity.badRequest().build();
        }
    }
```

```
PS C:\Users\dotse\IdeaProjects\VoteServerCatalogue> curl.exe -k -X POST "https://localhost:8080/nodes/register" -H "Cont
ent-Type: application/json" -d '{\"ip\":\"127.0.0.1\",\"domain\":\"\",\"port\":8082}'
curl: (7) Failed to connect to localhost port 8080 after 2251 ms: Could not connect to server
PS C:\Users\dotse\IdeaProjects\VoteServerCatalogue> curl.exe -k -X POST "https://localhost:8443/nodes/register" -H "Cont
ent-Type: application/json" -d '{\"ip\":\"127.0.0.1\",\"domain\":\"\",\"port\":8082}'
{"id":"76388fd9-a514-46d1-a0b8-64e588e06422","ip":"127.0.0.1","domain":"","port":8082,"lastHeartbeat":1747991344377}
PS C:\Users\dotse\IdeaProjects\VoteServerCatalogue> curl.exe -k -X POST "https://localhost:8443/nodes/register" -H "Cont
ent-Type: application/json" -d '{\"ip\":\"192.168.0.27\",\"domain\":\"\",\"port\":1127}'
{"id":"d89959a5-3b06-4e6e-8b64-63b122bcab9f","ip":"192.168.0.27","domain":"","port":1127,"lastHeartbeat":1747991370639}
PS C:\Users\dotse\IdeaProjects\VoteServerCatalogue> curl.exe -k -X POST "https://localhost:8443/nodes/register" -H "Cont
ent-Type: application/json" -d '{\"ip\":\"10.4.0.91\",\"domain\":\"\",\"port\":778}'
{"id":"750761ca-ddad-4239-8a72-ef803dd81799","ip":"10.4.0.91","domain":"","port":778,"lastHeartbeat":1747991388494}
PS C:\Users\dotse\IdeaProjects\VoteServerCatalogue>
```

Рисунок 4.3 – Процес реєстрації вузлів

Через використання самопідписаних сертифікатів при реєстрації вузлів, користуємося відповідною утилітою curl.exe з параметрами -k -X, щоб уникати помилки довіри до самоміписаного сертифікату.

На рисунку 4.3 у відповіді на запит ми можемо бачити наступне: id – згенерований UUID, поле lastHeartbeat – мітка часу реєстрації, а також відповідні надані IP та доменне ім’я (за наявності). В свою чергу, на консолі сервера з’являється запис про доданий вузол (рис. 4.4). При помилках (наприклад, недоступне доменне ім’я) очікується відповідь 400 Bad Request.

```
12:09:04.317 [https-jsse-nio-127.0.0.1-8443-exec-5] INFO
org.springframework.web.servlet.DispatcherServlet - Initializing Servlet 'dispatcherServlet'
12:09:04.317 [https-jsse-nio-127.0.0.1-8443-exec-5] INFO
org.springframework.web.servlet.DispatcherServlet - Completed initialization in 0 ms
Register new node: Node{id='76388fd9-a514-46d1-a0b8-64e588e06422', ip='127.0.0.1',
domain='', port=8082, lastHeartbeat=2025-05-23T09:09:04.377Z}
Register new node: Node{id='d89959a5-3b06-4e6e-8b64-63b122bcab9f', ip='192.168.0.27',
domain='', port=1127, lastHeartbeat=2025-05-23T09:09:30.639Z}
Register new node: Node{id='750761ca-ddad-4239-8a72-ef803dd81799', ip='10.4.0.91',
domain='', port=778, lastHeartbeat=2025-05-23T09:09:48.494Z}
```

Рисунок 4.4 – Консоль серверу при додаванні вузла

Окрім реєстрації, NodeController надає метод heartbeat для прийому сигналів життєдіяльності. При зверненні на POST /nodes/heartbeat з тілом запиту, що містить id вузла, сервер шукає відповідний запис вузла. Якщо вузол знайдено, оновлюється його поле lastHeartbeat на поточний час і зберігається нове значення, після чого повертається 200 OK з повідомленням “OK”. Якщо ж вузол з указаним id відсутній, повертається 404 Not Found. Цей механізм дозволяє періодично оновлювати статус активності кожного вузла. Для отримання агрегованої інформації про мережу слугує метод listNodes (GET /nodes), що повертає список усіх відомих серверу вузлів. За допомогою параметра запиту activeOnly=true можна відфільтрувати лише активні вузли – контролер порівнює поточний час із lastHeartbeat кожного вузла і виключає ті, що не надсилали сигнал більше встановленого таймауту (властивість app.heartbeatTimeoutMs, за замовчуванням 60 с). Вузли, для яких інтервал бездіяльності перевищує цей поріг, вважаються відключеними.

Сервіси (AdminService, VotingService, NodeService) – логічний рівень бізнес-логіки між контролерами та репозиторієм. У типовій багаторівневій архітектурі саме сервіси містять усю бізнес-логіку, тоді як контролери лише викликають відповідні методи сервісів і формують HTTP-відповідь. У поточній реалізації для спрощення код сервісів інтегровано безпосередньо в контролери.

4.2. Механізми ініціалізації та зберігання даних

Для зберігання постійних даних (облікових записів адміністраторів, списку кандидатів, зареєстрованих учасників та вузлів мережі) сервер використовує вбудовану базу даних RocksDB. В нашій системі RocksDB працює локально, зберігаючи файли БД у каталозі сервера, і забезпечує швидкий доступ до даних за ключем. Усі основні сутності системи (Admin, Candidate, Participant, Node) реалізовані як серіалізовані Java-об’єкти, що зберігаються у RocksDB у вигляді масивів байтів. Такий підхід спрощує зберігання різнорідних типів в єдиному сховищі – достатньо приєднвати до ключа певний префікс (наприклад, "admin:...", "candidate:..."), щоб розділити простір ключів для різних типів даних.

Клас RocksDBRepository виконує роль універсального репозиторію з наступним інтерфейсом: методи save(String key, Object value) для збереження об’єкта, get(String key) для отримання об’єкта за ключем, delete(String key) для видалення, та listByPrefix(String prefix) для переліку всіх об’єктів з певним префіксом ключа. Всі дані зберігаються у форматі key-value, де ключ – рядок (String), а значення – довільний об’єкт, що підтримує серіалізацію (в нашому випадку всі моделі реалізують Serializable).

При старті серверу виконується початкова ініціалізація даних через компонент DataInitializer. Це клас з анотацією @Component, який імплементує інтерфейс CommandLineRunner – тобто його метод run() запускається автоматично після підняття контексту Spring Boot. DataInitializer читає початкові налаштування (список адміністраторів, кандидатів, учасників) із конфігураційних параметрів програми. Задля гнучкості, ці параметри можуть передаватися через application.properties.

На основі цих значень DataInitializer створює відповідні об’єкти та зберігає їх у репозиторії. Лістинг нижче демонструє логіку цього процесу:

```
@Override
public void run(String... args) {
    // 1. Ініціалізація адміністраторів
    if (initialAdmins != null && !initialAdmins.isEmpty()) {
        String[] adminEntries = initialAdmins.split(";");
```

```
        for (String entry : adminEntries) {
            String[] parts = entry.split(":", 2);
            if (parts.length == 2) {
                String username = parts[0];
                String password = parts[1];
                Admin admin = new Admin(username, password);
                repository.save("admin:" + username, admin);
                System.out.println("Admin initialized: " + username);
            }
        }
    }
}
```

Безпосередньо у коді (Додаток А), ініціалізація проходить у три етапи: спочатку обробляється список адміністраторів (кожен запис формату “користувач:пароль” розділяється і зберігається як об’єкт Admin під ключем "admin:<username>"), далі – список кандидатів (їм привласнюються числові ID починаючи з 1, і кожен Candidate зберігається під ключем "candidate:<id>"), і останнім – список учасників (створюються порожні записи Participant з відповідними ID). Цей механізм дозволяє при необхідності задати початкові дані для тестування або першого запуску системи. Таким чином, адміністратори з попередньо прописаними обліковими записами будуть автоматично зареєстровані в базі при старті (в консолі з’являться повідомлення виду “Admin initialized: admin1” тощо).

Після виконання DataInitializer система готова до роботи, маючи початкові сутності у сховищі. Подальші операції (через контролери) працюють із репозиторієм RocksDBRepository. Цей компонент, позначений @Component, у своєму методі @PostConstruct init() виконує завантаження native-бібліотеки RocksDB і відкриває базу. За замовчуванням шлях до файлів БД – data/rocksdb (створюється, якщо не існує). Клас використовує утилітний клас Spring SerializationUtils для серіалізації об’єктів у масив байтів та зворотної десеріалізації. Важливо відзначити, що RocksDB на низькому рівні оперує саме масивами байтів як ключами і значеннями [32], тому ми

маємо перетворювати наші об’єкти при записі/читанні. Основні методи репозиторію наведені нижче:

```
public synchronized boolean save(String key, Object value) {
    try {
        byte[] data = SerializationUtils.serialize(value);
        db.put(key.getBytes(), data);
        return true;
    } catch (RocksDBException e) {
        System.err.println("Failed to save key " + key + ": " +
e.getMessage());
        return false;
    }
}
public synchronized Optional<Object> get(String key) {
    try {
        byte[] data = db.get(key.getBytes());
        if (data != null) {
            Object value = SerializationUtils.deserialize(data);
            return Optional.of(value);
        }
    } catch (RocksDBException e) {
        System.err.println("Failed to get key " + key + ": " +
e.getMessage());
    }
    return Optional.empty();
}
```

У методі save() об’єкт серіалізується у масив байтів (використовується стандартна Java-серіалізація), після чого зберігається викликом db.put() у базу. Метод get() зчитує байти за вказаним ключем і виконує десеріалізацію назад в об’єкт потрібного типу.

Для перевірки коректності роботи сховища було виконано кілька сценаріїв ручного тестування репозиторію. Зокрема, тестувалися базові операції CRUD:

збереження, читання, видалення та перелік записів. Сценарії та результати наведено в табл. 4.1.

Таблиця 4.1 – Сценарії тестування сховища даних (RocksDBRepository)

№	Опис операції	Вхідні дані (ключ/значення)	Очікуваний результат	Фактичний результат
1	Збереження нового адміністратора	Key: "admin:alice", Value: Admin("alice","qwerty")	save() повертає true; запис додається	Повернено true, адміністратор збережений
2	Зчитування існуючого адміністратора	Key: "admin:alice"	Отримано Optional з об’єктом Admin(username="alice")	Отримано об’єкт Admin з username="alice"
3	Спроба зчитати неіснуючий ключ	Key: "admin:bob" (немає в базі)	Порожній Optional (результат відсутній)	Результат Optional.empty() (нічого не знайдено)
4	Додавання нового кандидата	Key: "candidate:1", Value: Candidate(id="1", name="Alice")	save() повертає true; кандидат доданий	Кандидат збережений, get("candidate:1") повертає об’єкт Candidate(name="Alice", votes=0)
5	Оновлення поля (голосування за кандидата)	Key: "candidate:1", Value: Candidate(id="1", votes=1)	Значення успішно перезаписане	Get("candidate:1") повертає оновлений votes=1
6	Перелік всіх кандидатів після оновлення	Prefix: "candidate:"	Повертається список з 1 об’єкта Candidate	Отримано список із 1 кандидата (Alice, 1 голос)
7	Видалення адміністратора	Key: "admin:alice"	delete() повертає true; запис видаляється	Видалення успішне, повторний get("admin:alice") дає порожній результат

4.3. Комунікація між вузлами та сервером

Система передбачає, що декілька вузлів (нод) утворюють peer-to-peer мережу для проведення децентралізованого голосування. Хоча підрахунок голосів у поточній реалізації здійснюється центральним сервером, механізми реєстрації вузлів та підтримки їх зв’язку реалізовані у вигляді REST API, описаного вище в NodeController.

Взаємодія вузлів з сервером відбувається за допомогою HTTP-запитів, причому сервер виконує роль координатора мережі. Новий вузол, запускаючись, звертається до серверу для реєстрації (отримання ідентифікатора та інформації про інших учасників мережі), а надалі періодично надсилає повідомлення про те, що він активний (heartbeat). Сервер, у свою чергу, зберігає список вузлів та може надавати кожному вузлу інформацію про інших відомих активних вузлів.

Нижче наведено основні ендпоінти NodeController та їх роль у комунікації:

- POST /nodes/register – використовується вузлом для підключення до мережі. Вузол в запиті передає свою адресу (IP або домен) і порт. У відповіді сервер повертає об’єкт Node з присвоєним йому унікальним id та відображає ту саму адресу і порт (тобто підтверджує реєстрацію). Після виклику цього ендпоінту новий вузол має в системі запис і може брати участь у подальшій взаємодії.
- POST /nodes/heartbeat – викликається регулярно кожним вузлом для сигналізації серверу, що вузол ще «живий». Тіло запиту містить лише його id. Сервер у відповіді надсилає “OK”, якщо успішно оновив мітку часу останнього сигналу. Якщо сервер відповів 404, вузол може вирішити повторно зареєструватися, оскільки це означає, що він не пройшов реєстрацію на сервері.
- GET /nodes?activeOnly=true/false – викликається вузлом, щоб отримати список всіх вузлів, що відомі серверу. Якщо параметр activeOnly=true, повертаються тільки активні вузли. Якщо параметр не вказано або false, повертається повний список, включно з потенційно відключеними вузлами. Відповідь – масив JSON об’єктів Node, кожен з полями: id, ip, domain, port, lastHeartbeat. На основі цієї інформації вузол може дізнатися, які інші вузли є в мережі та як до них під’єднатися.

Як було вище зазначено, у розробленій системі роль heartbeat відіграє POST запит на /nodes/heartbeat. Вузол починає надсилати такі запити одразу після реєстрації і повторює їх з певним інтервалом (меншим за поріг неактивності, наприклад кожні 30 секунд при таймауті 60 секунд). Сервер виступає в ролі монітора: отримуючи heartbeat, він оновлює статус вузла як активного (записує поточний час). Якщо від якогось вузла

сигнал перестає надходити, сервер через визначений час (таймаут) позначає його як відключений. У даній реалізації це відбувається неявно – запис вузла залишається в базі, але при запиті списку з activeOnly=true такий вузол не включається до результатів. Адміністратор або інші вузли можуть опосередковано визначити, що вузол не працює, якщо він зник зі списку активних.

Щоб новий вузол міг приєднатися до існуючої P2P-мережі, він повинен дізнатися адреси хоча б декількох поточних учасників. Типовими підходами є наявність одного або кількох відомих серверів або вузлів, до яких звертається новачок для отримання списку пірів [12]. У даній системі саме центральний сервер виконує роль такого координатора мережі для спрощення комунікацій, тож після реєстрації вузол може зробити запит до /nodes для отримання списку інших вузлів. Надалі вузол теоретично може встановити прямі з'єднання з деякими з них (наприклад, для обміну повідомленнями або дублювання даних голосування між вузлами). Процес приєднання та відповідні результати зображені вище на рисунках 4.3 та 4.4.

Нижче зображено послідовність дій приєднання вузла та підтримки зв'язку у вигляді діаграми послідовності (рис. 4.5).

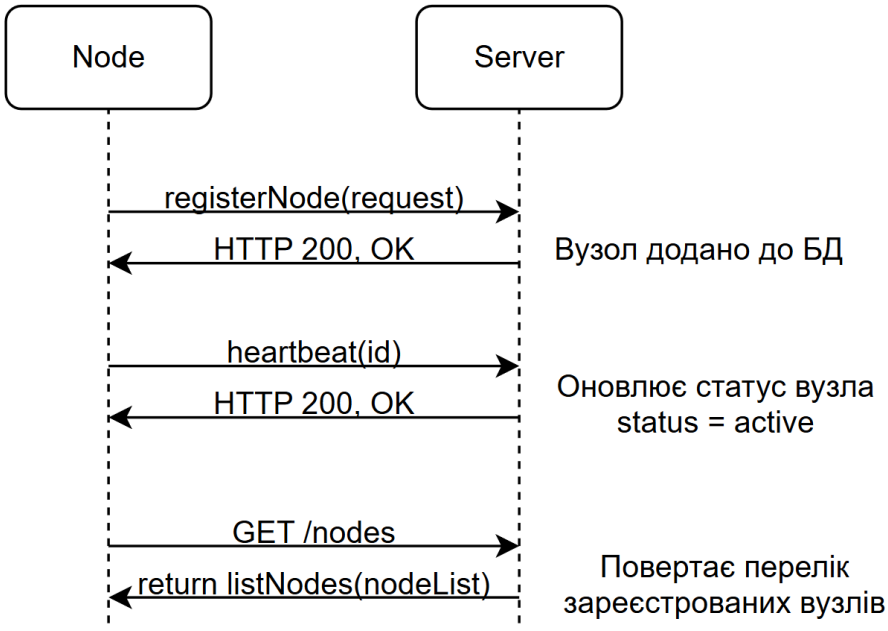

Рисунок 4.5 – Схема послідовності дій для реєстрації вузла та підтримки зв'язку

Варто зазначити, що наразі сервер не розсилає вузлам жодних повідомлень про появу нових або зникнення існуючих вузлів – обмін відбувається лише за запитами від самих вузлів. Це означає, що кожен вузол за бажанням може періодично опитувати сервер, аби отримати оновлений список учасників мережі. Таке опитування не є обов'язковим – якщо топологія мережі не вимагає прямого зв'язку між вузлами, вузлу достатньо лише виконувати heartbeat для підтримки себе в активному стані.

4.4. Реалізація протоколів безпеки

Для реалізації захисту каналу в Spring Boot використовуємо вбудований механізм TLS [18][35]. Потрібно підготувати файл з протоколом keystore, що містить сертифікат сервера та закритий ключ, і вказати Spring Boot ці налаштування у конфігурації. Прикладом конфігурації є:

```
server.port=8443
server.ssl.key-store=classpath:keystore.jks
server.ssl.key-store-password=dkwlo1d12
server.ssl.key-password=adkwlo1d12
```

Ці налаштування (порт 8443 та властивості server.ssl.*) повідомляють Spring Boot, що слід запускати вбудований сервер HTTPS з вказаним ключем. Таким чином, сервер-каталог прийматиме запити тільки по захищеному каналу TLS [18][35].

Для CLI-клієнта необхідно, щоб клієнтська сторона довіряла сертифікату сервера. У Java реалізовано механізм trustStore – сховище довірених сертифікатів. У файлі trustStore містяться сертифікати центрів сертифікації або сам сертифікат сервера, яким довіряють клієнти [29]. Це забезпечить перевірку сертифіката сервера під час встановлення TLS-з'єднання [18].

4.5. Інтерфейс командного рядка (CLI)

ClientCLI реалізовано у вигляді окремого класу з методом main, який аналізує аргументи командного рядка. Перший аргумент сприймається як назва команди, інші – як параметри. Передбачено декілька команд:

- vote – надіслати голос виборця за певного кандидата.

- list-nodes – отримати список вузлів, відомих серверу.
- results – дізнатися поточні результати голосування (кількість голосів за кожного кандидата).
- Також є команда для виводу допомоги у разі некоректного вводу – при невідомій команді виводиться підказка щодо використання (рис. 4.6).

```
Usage:
 java -jar secure-voting.jar vote <participantId> <candidateId> [url]
 java -jar secure-voting.jar list-nodes [url]
 java -jar secure-voting.jar results [url]

Process finished with exit code 0
```

Рисунок 4.6 – Підказка щодо використання CLI

За замовчуванням CLI звертається до серверу за адресою http://localhost:8080, але це можна змінити, передавши базовий URL останнім параметром. Внутрішньо, ClientCLI використовує стандартний HTTP-клієнт Java 11 (java.net.http.HttpClient) та бібліотеку Jackson для розбору JSON. Виконання деяких основних команд зображено вище на рис. 4.2 та 4.3, тож виконання команди list-nodes можемо побачити на рис. 4.7, де яскраво видно перелік попередньо зареєстрованих вузлів з відповідними параметрами.

```
PS C:\Users\dotse\IdeaProjects\VoteServerCatalogue> mvn -q exec:java "-Dexec.args=list-nodes"
Available nodes (3):
- ID=750761ca-ddad-4239-8a72-ef803dd81799, IP=10.4.0.91, domain=, port=778, status=inactive
- ID=76388fd9-a514-46d1-a0b8-64e588e06422, IP=127.0.0.1, domain=, port=8082, status=inactive
- ID=d89959a5-3b06-4e6e-8b64-63b122bcab9f, IP=192.168.0.27, domain=, port=1127, status=inactive
PS C:\Users\dotse\IdeaProjects\VoteServerCatalogue>
```

Рисунок 4.7 – Перелік наявних вузлів в мережі

Примітка: CLI-клієнт наразі не охоплює адміністративні функції (наприклад, додавання кандидата). Такі операції потребують передачі адміністративного ключа і можуть бути виконані через HTTP (наприклад, за допомогою curl або розширення CLI). У тестовому середовищі ми додавали кандидатів через REST API окремо (рис. 4.3).

Результуючий перелік команд наведено у таблиці 4.2.

Таблиця 4.2 – Команди CLI та їх приклади тестування.

Команда (параметри)	Дія (HTTP-запит)	Очікуваний результат	Фактичний результат
vote voterN N	POST /vote (тіло: participantId=VOTER1, candidateId=1)	HTTP 200, голос збережено (або помилка, якщо спроба повторного голосування)	HTTP 200 Vote recorded (при першому голосуванні) HTTP 409 Participant has already voted (при повторному)
list-nodes	GET /nodes	HTTP 200, список вузлів (JSON масив)	HTTP 200 Available nodes (X): ... status=active/inactive
results	GET /results	HTTP 200, список кандидатів з голосами	HTTP 200 Voting results: voter1 (ID=1): 1 votes; voter2 (ID=2): 2 votes…

4.6 Загальна інтеграція та висновки

Розроблене серверне програмне забезпечення успішно інтегрує всі необхідні компоненти та забезпечує виконання функцій, поставлених вимогами системи захищеного голосування. У попередніх підрозділах продемонстровано, що сервер підтримує такі основні можливості:

- Управління користувачами та кандидатами: адміністратор може додавати нових адміністраторів (для розподілу прав), реєструвати кандидатів для голосування, видаляти адміністративні облікові записи. Всі ці дії потребують авторизації за допомогою секретного ключа, що запобігає несанкціонованому втручанню.
- Реєстрація вузлів P2P-мережі: будь-який новий вузол мережі може зареєструватися через сервер, отримавши унікальний ідентифікатор. Сервер накопичує інформацію про всі вузли, що приєдналися, що відповідає вимогам до каталогу вузлів у децентралізованій системі.
- Моніторинг активності вузлів: завдяки механізму періодичних heartbeat-повідомлень, сервер відстежує, які вузли активні. Якщо вузол виходить з мережі або у нього виникають проблеми (heartbeat не надходять), сервер через визначений інтервал позначає його як неактивний. Інші вузли або

адміністративні клієнти можуть отримати актуальний список активних вузлів у будь-який момент.

- Приймання та облік голосів: сервер приймає голоси виборців по одному за раз, перевіряє ключове правило (кожен учасник може голосувати тільки один раз) і коректно збільшує лічильник голосів у вибраного кандидата. Усі голоси одразу фіксуються у сховищі даних, що гарантує їх збереження.
- Надання результатів голосування: у будь-який момент уповноважені особи (або навіть будь-який учасник, якщо доступ відкритий) можуть отримати поточні результати – перелік усіх кандидатів з кількістю набраних голосів.
- CLI для взаємодії з сервером: розроблений консольний клієнт підтвердив працездатність API та спростив процедуру ручного тестування. Було протестовано основні сценарії (голосування, перегляд вузлів, перегляд результатів) у інтегрованому середовищі, що включає всю систему цілком (клієнт + сервер + база даних).

Підсумовуючи, створений сервер P2P-мережі для захищеного голосування є працездатним ядром системи. Він демонструє принципи збереження даних, обмеження доступу та обміну інформацією між вузлами. Ручне тестування підтвердило виконання поставлених задач. Наступним кроком стане нарощування функціоналу та переведення тестування на автоматизований варіант, що забезпечить надійність і готовність системи до реального використання.

ВИСНОВКИ

В результаті виконання дипломної роботи спроєктовано та реалізовано програмне забезпечення серверу-каталогу P2P-мережі захищеного голосування. Розроблений сервер успішно інтегрує всі необхідні компоненти та виконує поставлені функції, слугуючи працездатним ядром децентралізованої системи електронного голосування.

Електронне голосування на даний момент представлено кількома технологічними підходами (DRE, I-voting, блокчейн) із різними рівнями впровадження та довіри в різних країнах. Аналіз показав низку критичних загроз (фізичні вразливості, кібератаки, інсайдерські ризики). Основні напрями подальшого розвитку – децентралізація, відкритий код і нові криптопротоколи для підвищення прозорості та безпеки.

Було встановлено шість ключових функціональних сервісів серверу-каталогу P2P-мережі (реєстрація вузлів, подання й підрахунок голосів, аудит, моніторинг, управління ролями), а також нефункціональні вимоги за моделлю CIA, продуктивність, масштабованість і відмовостійкість. Окремо визначені адміністративні функції для ролей «адміністратор» і «вузол/клієнт». Проведено оцінку ризиків і визначено компроміси між безпекою та простотою впровадження, що закладає основу для архітектури.

P2P-архітектура визнана оптимальною для забезпечення відмовостійкості та масштабованості без єдиної точки відмови. Як технологічний стек обрано Java та Spring Boot за їх високу продуктивність і розвинену екосистему, а для зберігання даних – вбудоване key-value сховище RocksDB із надшвидким доступом. REST API й CLI забезпечують розділення інтерфейсів для клієнтів і адміністраторів, а впроваджені протоколи безпеки гарантують захищеність каналів і цілісність даних.

Розроблено компонентну архітектуру з чітким поділом на контролери, сервіси та репозиторії, що взаємодіють із RocksDB, і реалізовано механізми ініціалізації даних та періодичні heartbeat-повідомлення для контролю доступності вузлів. Впроваджено

засоби автентифікації й шифрування каналів (TLS), а CLI-інтерфейс розширює можливості адміністрування.

Проведене тестування підтвердило, що рішення забезпечує збереження цілісності даних (неможливість непомітної зміни або видалення голосів), контрольований доступ до ресурсів системи та надійний обмін інформацією між вузлами мережі. Запропонований підхід відповідає сучасним світовим тенденціям: використання P2P-архітектури і розподіленого реєстру уподібнюється блокчейн-рішенням, усуваючи єдиний центр відмови та гарантуючи незмінність записів, а застосування криптографічних методів відкриває можливості для реалізації наскрізної (End-to-End) верифікації голосування, подібно до системи Helios.

Таким чином, отримані результати не поступаються відомим підходам і роблять практичний внесок у розвиток захищених систем електронного голосування. Розроблений у роботі P2P-сервер може бути безпосередньо впроваджений у практику електронного голосування різного рівня. Зокрема, результати доцільно використовувати для підвищення безпеки національних і місцевих виборів, референдумів, де критично важливо забезпечити довіру виборців до цифрового процесу. Така система також знайде застосування у корпоративному середовищі – для проведення виборів і опитувань у межах підприємств, установ чи акціонерних товариств.

Наукова значущість роботи полягає в розвитку методів децентралізації та захисту електронного голосування (побудовано нову архітектуру, що поєднує P2P-мережу з принципами криптографічного захисту). Технічна значущість підтверджується створенням працездатного прототипу, який реалізує сучасні принципи кібербезпеки та може слугувати основою для подальшого вдосконалення систем електронного голосування. Додатково значущість проекту проявляється у потенційному підвищенні прозорості виборчого процесу та рівня довіри громадян шляхом використання захищеного електронного голосування.

У подальшому планується впровадити механізм консенсусу між кількома екземплярами серверу-каталогу, що дозволить використовувати декілька вузлів

серверу без ризику неузгодженості даних і підвищить відмовостійкість мережі. Важливим напрямом є покращення масштабованості рішення для підтримки великої кількості вузлів і виборців без втрати продуктивності (це передбачає оптимізацію архітектури та, за потреби, горизонтальне масштабування або розподіл навантаження).

Крім того, планується підготувати сценарії інтеграції серверу P2P-мережі в існуючі інфраструктури голосування, опрацювати процедури резервного копіювання й оновлення вузлів, а також провести навчання персоналу, відповідального за експлуатацію системи. Реалізація зазначених кроків сприятиме успішному впровадженню отриманих результатів у практику електронного голосування та забезпечить подальше зростання довіри до таких систем.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Адміністрація Державної служби спеціального зв'язку та захисту інформації України. Наказ від 06.10.2021 № 601 «Про затвердження Методичних рекомендацій щодо підвищення рівня кіберзахисту критичної інформаційної інфраструктури». – Київ, 2021.
2. ДСТУ ISO/IEC 27005:2023 Інформаційна безпека, кібербезпека та захист конфіденційності. Настанова керування ризиками інформаційної безпеки – Київ : УкрДержНДП, 2023.
3. НД ТЗІ 1.4-001-2000 Типове положення про службу захисту інформації в автоматизованій системі : нормативний документ системи технічного захисту інформації / Департамент спеціальних телекомунікаційних систем та захисту інформації СБУ. – Київ : ДСТЗІ СБУ, 2000. – Затверджено 04.12.2000 № 53; зміни 28.12.2012 № 806. – Режим доступу: https://tzi.com.ua/downloads/1.4-001-2000.pdf (дата звернення: 08.05.2025).
4. Abazorius E. MIT researchers identify security vulnerabilities in voting app [Електронний ресурс] // MIT News. — 2020. — 13 лютого. — Режим доступу: https://news.mit.edu/2020/voting-voatz-app-hack-issues-0213 (дата звернення: 08.05.2025).
5. Alown M., Kiraz M. S., Bingol M. A. Enhancing Democratic Processes: A Survey of DRE, Internet, and Blockchain in Electronic Voting Systems [Електронний ресурс] // IEEE Access. – 2025. – T. 13. – Art. no 3531349. – DOI: 10.1109/ACCESS.2025.3531349. – Режим доступу: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10844285 (дата звернення: 08.05.2025).
6. Amazon Web Services. В чому різниця між gRPC і REST? [Електронний ресурс] // AWS. — Режим доступу: https://aws.amazon.com/ru/compare/the-difference-between-grpc-and-rest/ (дата звернення: 18.05.2025).
7. Brotsis S., Kolokotronis N., Limniotis K., Bendiab G., Shiaeles S. On the Security and Privacy of Hyperledger Fabric: Challenges and Open Issues [Електронний ресурс]

// Proceedings of the 2020 IEEE World Congress on Services (SERVICES 2020). – 18-23 жовт. 2020. – С. 197–204. — DOI: 10.1109/SERVICES48979.2020.00049. — Режим доступу: https://doi.org/10.1109/SERVICES48979.2020.00049 (дата звернення: 18.05.2025).

8. Buchanan B., Sulmeyer M. Hacking Chads: The Motivations, Threats, and Effects of Electoral Insecurity [Електронний ресурс] / Ben Buchanan, Michael Sulmeyer. — Cambridge, MA : Belfer Center for Science and International Affairs, Harvard Kennedy School, 2016. — 20 с. — (The Cyber Security Project Paper; October 2016). — Режим доступу: https://www.belfercenter.org/sites/default/files/pantheon_files/files/publication/hacking-chads.pdf (дата звернення: 08.05.2025).
9. Facebook. RocksDB [Електронний ресурс] // GitHub. — Режим доступу: https://github.com/facebook/rocksdb (дата звернення: 18.05.2025).
10. Facebook. RocksDB In-Memory Workload Performance Benchmarks [Електронний ресурс] // GitHub Wiki (facebook/rocksdb). — Режим доступу: https://github.com/facebook/rocksdb/wiki/RocksDB-In-Memory-Workload-Performance-Benchmarks (дата звернення: 18.05.2025).
11. Feldman A. J., Halderman J. A., Felten E. W. Security Analysis of the Diebold AccuVote-TS Voting Machine [Електронний ресурс] // USENIX. — 13 вересня 2006. — Режим доступу: https://www.usenix.org/legacy/event/evt07/tech/full_papers/feldman/feldman_html/index.html (дата звернення: 08.05.2025).
12. GeeksforGeeks. What is P2P (Peer-to-Peer Process)? [Електронний ресурс] // GeeksforGeeks. — Останнє оновлення: 06.10.2024. — Режим доступу: https://www.geeksforgeeks.org/what-is-P2P-peer-to-peer-process/ (дата звернення: 18.05.2025).
13. Han J., Seo Y., Lee S., Kim S., Son Y. Design and Implementation of Enabling SQL-Query Processing for Ethereum-Based Blockchain Systems [Електронний ресурс] // Electronics. — 2023. — Т. 12, № 20. — Стаття 4317. — DOI: 10.3390/electronics12204317. — Режим доступу: https://doi.org/10.3390/electronics12204317 (дата звернення: 18.05.2025).

14. Helios Voting. Documentation [Електронний ресурс] // documentation.heliosvoting.org. — Режим доступу: https://documentation.heliosvoting.org (дата звернення: 08.05.2025).
15. Help America Vote Act of 2002 (Public Law 107–252 від 29 жовтня 2002 р.) [Електронний ресурс] // Congress.gov. — Режим доступу: https://www.congress.gov/107/plaws/publ252/PLAW-107publ252.pdf (дата звернення: 08.05.2025).
16. International IDEA. Cybersecurity in Elections: Models of Interagency Collaboration [Електронний ресурс] / International IDEA. — Стокгольм, [б. р.]. — Режим доступу: https://www.idea.int/sites/default/files/publications/cybersecurity-in-elections-models-of-interagency-collaboration.pdf (дата звернення: 08.05.2025).
17. International Institute for Democracy and Electoral Assistance (International IDEA). Use of E-Voting Around the World [Електронний ресурс]. – Режим доступу: https://www.idea.int/news-media/multimedia-reports/use-e-voting-around-world – Дата звернення: 08.05.2025.
18. Internet Engineering Task Force. RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3 [Електронний ресурс]. – Серпень 2018. – Режим доступу: https://datatracker.ietf.org/doc/html/rfc8446 (дата звернення: 21.05.2025).
19. Jafar U., Ab Aziz M. J., Shukur Z. Blockchain for Electronic Voting System—Review and Open Research Challenges [Електронний ресурс] // Sensors. — 2021. — Т. 21, № 17. — Стаття 5874. — DOI: 10.3390/S21175874. — Режим доступу: https://www.mdpi.com/1424-8220/21/17/5874 (дата звернення: 08.05.2025).
20. Jeleń M., Dzieńkowski M. The comparative analysis of Java frameworks: Spring Boot, Micronaut and Quarkus [Електронний ресурс] // Journal of Computer Sciences Institute. – 2021. – Т. 21. – С. 287–294. – DOI: https://doi.org/10.35784/jcsi.2724. – Режим доступу: https://www.researchgate.net/publication/357432573_The_comparative_analysis_of_Java_frameworks_Spring_Boot_Micronaut_and_Quarkus. (дата звернення: 24.05.2025).

21. Kho Y.-X., Heng S.-H., Chin J.-J. A Review of Cryptographic Electronic Voting [Електронний ресурс] // Symmetry. — 2022. — Т. 14, № 5. — Стаття 858. — DOI: 10.3390/sym14050858. — Режим доступу: https://doi.org/10.3390/sym14050858 (дата звернення: 08.05.2025).
22. Kuenzi R. These are the arguments that sank e-voting in Switzerland [Електронний ресурс] // Swissinfo.ch. — 02.08.2019. — Режим доступу: https://www.swissinfo.ch/eng/politics/e-voting_these-are-the-arguments-that-sank-e-voting-in-switzerland/45136608 (дата звернення: 08.05.2025).
23. Kusters R., Truderung T., Fouque P.-A. Clash Attacks on the Verifiability of E-Voting Systems [Електронний ресурс] // IACR Cryptol. ePrint Arch. — 2012. — № 116. — Режим доступу: https://eprint.iacr.org/2012/116.pdf (дата звернення: 08.05.2025).
24. Lindmark M., Salihovic A.–A. Investigating the Security of End-to-End and Blockchain-based Electronic Voting Systems: A Comparative Literature Review [Електронний ресурс] : bachelor’s thesis / Uppsala University, Department of Informatics and Media. – Uppsala, 12.06.2022. – 40 с. – Режим доступу: https://www.diva-portal.org/smash/get/diva2:1683763/FULLTEXT01.pdf (дата звернення: 08.05.2025).
25. Mulder V., Mermoud A., Lenders V., Tellenbach B. Trends in Data Protection and Encryption Technologies [Електронний ресурс] / Valentin Mulder, Alain Mermoud, Vincent Lenders, Bernhard Tellenbach (ред.) – Cham : Springer, 2023. – DOI: 10.1007/978-3-031-33386-6. – Режим доступу: https://link.springer.com/book/10.1007/978-3-031-33386-6 (дата звернення: 08.05.2025).
26. National Institute of Standards and Technology (NIST). FIPS Publication 199: Standards for Security Categorization of Federal Information and Information Systems [Електронний ресурс] / National Institute of Standards and Technology. – Gaithersburg, MD : NIST, 2004. – 14 с. – Режим доступу: https://csrc.nist.gov/publications/detail/fips/199/final (дата звернення: 18.05.2025).
27. Nodes [Електронний ресурс] // Kubernetes Documentation. — Режим доступу: https://kubernetes.io/docs/concepts/architecture/nodes/#:~:text=When%20the%20ku

belet%20flag%20%60,pattern%2C%20used%20by%20most%20distros (дата звернення: 18.05.2025).

28. Ohize G. O., Onumanyi A. J., Umar B. U., Ajao L. A., Isah R. O., Dogo E. M. та ін. Blockchain for securing electronic voting systems: a survey of architectures, trends, solutions, and challenges [Електронний ресурс] // Cluster Computing. — 2025. — Т. 28. — Стаття 132. — DOI: 10.1007/s10586-024-04709-8. — Режим доступу: https://doi.org/10.1007/s10586-024-04709-8 (дата звернення: 08.05.2025).
29. Oracle. Java Secure Socket Extension (JSSE) Reference Guide [Електронний ресурс] // docs.oracle.com. — Режим доступу: https://docs.oracle.com/cd/E19509-01/820-3503/ggffo/index.html (дата звернення: 21.05.2025).
30. OWASP. REST Security Cheat Sheet [Електронний ресурс] // OWASP Cheat Sheet Series. — Режим доступу: https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html (дата звернення: 21.05.2025).
31. Pivotal Software, Inc. Building a RESTful Web Service [Електронний ресурс]. – Режим доступу: https://spring.io/guides/gs/rest-service (дата звернення: 23.05.2025).
32. Russo A., Fernández Anta A., González Vasco M. I., Romano S. P. Chirotonia: A Scalable and Secure E-Voting Framework Based on Blockchains and Linkable Ring Signatures : preprint [Електронний ресурс] / A. Russo, A. Fernández Anta, M. I. González Vasco, S. P. Romano. – arXiv, 2021. – Режим доступу: https://arxiv.org/pdf/2111.02257 (дата звернення: 08.05.2025).
33. Springall D., Finkenauer T., Durumeric Z., Kitkat D., Hursti H., McAlpine M., Halderman J. A. Security Analysis of the Estonian Internet Voting System [Електронний ресурс] // Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS ’14). – 3-7 листопада 2014. – С. 703–715. – DOI: 10.1145/2660267.2660315. – Режим доступу: https://jhalderm.com/pub/papers/ivoting-ccs14.pdf (дата звернення: 08.05.2025).

34. Spring Boot Documentation [Електронний ресурс] // docs.spring.io. – Режим доступу: https://docs.spring.io/spring-boot/documentation.html (дата звернення: 18.05.2025).
35. Spring Boot Reference: SSL [Електронний ресурс] // Spring Boot Documentation. — Режим доступу: https://docs.spring.io/spring-boot/reference/features/ssl.html (дата звернення: 21.05.2025).
36. Tambunan D. Policy Transfer of E-Voting From India to Indonesia: A Review Concept and Evidence [Електронний ресурс] // International Journal of Social Science And Human Research. — 2022. — Т. 05, № 03 (березень). — С. 877. — DOI: 10.47191/ijsshr/v5-i3-21. — Режим доступу: https://www.researchgate.net/profile/Derwin-Tambunan/publication/359256755_Policy_Transfer_of_E-Voting_From_India_to_Indonesia_A_Review_Concept_and_Evidence/links/623190324ce552783cbfa675/Policy-Transfer-of-E-Voting-From-India-to-Indonesia-A-Review-Concept-and-Evidence.pdf (дата звернення: 08.05.2025).
37. Yaga D., Mell P., Roby N., Scarfone K. Blockchain Technology Overview : NISTIR 8202 [Електронний ресурс] / D. Yaga, P. Mell, N. Roby, K. Scarfone. — Gaithersburg, MD : National Institute of Standards and Technology, U.S. Department of Commerce, 2018. — 66 с. — DOI: 10.6028/NIST.IR.8202. — Режим доступу: https://doi.org/10.6028/NIST.IR.8202 (дата звернення: 20.05.2025).
38. Yi H. Securing e-voting based on blockchain in P2P network [Електронний ресурс] // EURASIP Journal on Wireless Communications and Networking. — 2019. — Т. 2019. — Стаття 137. — DOI: 10.1186/s13638-019-1473-6. — Режим доступу: https://jwcn-eurasipjournals.springeropen.com/articles/10.1186/s13638-019-1473-6 (дата звернення: 18.05.2025).
39. Zhang J., Zhang B., Nastenko A., Balogun H., Oliynykov R. Privacy-Preserving Decision-Making over Blockchain [Електронний ресурс] // IEEE Transactions on Dependable and Secure Computing. – 2022. – С. 1–17. – DOI: 10.1109/TDSC.2022.3231237. – Режим доступу: https://clok.uclan.ac.uk/45105/1/TDSC_Manuscript.pdf (дата звернення: 23.05.2025).

40. Zhang S., Wang L., Xiong H. Chaintegrity: blockchain-enabled large-scale e-voting system with robustness and universal verifiability [Електронний ресурс] // International Journal of Information Security. — 2019. — Т. 19, № 3. — С. 323–341. — DOI: 10.1007/s10207-019-00465-8. — Режим доступу: https://doi.org/10.1007/s10207-019-00465-8 (дата звернення: 08.05.2025).
41. Zhou Y., Han R., Li Y. A Blockchain Network Communication Architecture Based on Information-Centric Networking [Електронний ресурс] // Applied Sciences. — 2025. — Т. 15, № 6. — Стаття 3340. — DOI: 10.3390/app15063340. — Режим доступу: https://doi.org/10.3390/app15063340 (дата звернення: 18.05.2025).

ДОДАТОК А

Вихідний код серверного додатку.

```
package securevoting;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import
org.springframework.boot.context.properties.ConfigurationPropertiesScan;

@SpringBootApplication
@ConfigurationPropertiesScan
public class SecureVotingApplication {
    public static void main(String[] args) {
        if (System.getProperty("spring.profiles.active") == null &&
                System.getenv("SPRING_PROFILES_ACTIVE") == null) {
            System.setProperty("spring.profiles.active", "local");
        }
        SpringApplication.run(SecureVotingApplication.class, args);
    }
}
```

```
package securevoting;

import securevoting.model.Admin;
import securevoting.model.Candidate;
import securevoting.model.Participant;
import securevoting.repository.RocksDBRepository;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

@Component
public class DataInitializer implements CommandLineRunner {

    @Value("${app.initialAdmins:}")
    private String initialAdmins;
    @Value("${app.initialCandidates:}")
    private String initialCandidates;
    @Value("${app.initialParticipants:}")
    private String initialParticipants;

    private final RocksDBRepository repository;

    public DataInitializer(RocksDBRepository repository) {
        this.repository = repository;
    }

    @Override
    public void run(String... args) {
        // 1. Ініціалізація адміністраторів
```

```
        if (initialAdmins != null && !initialAdmins.isEmpty()) {
            String[] adminEntries = initialAdmins.split(";");
            for (String entry : adminEntries) {
                String[] parts = entry.split(":", 2);
                if (parts.length == 2) {
                    String username = parts[0];
                    String password = parts[1];
                    Admin admin = new Admin(username, password);
                    repository.save("admin:" + username, admin);
                    System.out.println("Admin initialized: " + username);
                }
            }
        }

        // 2. Ініціалізація кандидатів
        if (initialCandidates != null && !initialCandidates.isEmpty()) {
            String[] candNames = initialCandidates.split(",");
            int idCounter = 1;
            for (String name : candNames) {
                String candId = String.valueOf(idCounter++);
                Candidate candidate = new Candidate(candId.trim(),
name.trim());
                repository.save("candidate:" + candidate.getId(), candidate);
                System.out.println("Candidate initialized: " +
candidate.getName() + " (ID=" + candidate.getId() + ")");
            }
        }

        // 3. Ініціалізація учасників (виборців)
        if (initialParticipants != null && !initialParticipants.isEmpty()) {
            String[] parts = initialParticipants.split(",");
            for (String partId : parts) {
                Participant participant = new Participant(partId.trim());
                repository.save("participant:" + participant.getId(),
participant);
                System.out.println("Participant initialized: " +
participant.getId());
            }
        }
    }
}
```

```
package securevoting.repository;

import org.rocksdb.RocksDB;
import org.rocksdb.Options;
import org.rocksdb.RocksDBException;
import org.springframework.stereotype.Component;
import org.springframework.util.SerializationUtils;

import jakarta.annotation.PostConstruct;
import jakarta.annotation.PreDestroy;
```

```
import java.io.File;
import java.nio.file.Files;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

@Component
public class RocksDBRepository {
    private static final String DB_PATH_PROPERTY = "${rocksdb.path}";
    private RocksDB db;
    private String dbPath;

    @PostConstruct
    public void init() {
        RocksDB.loadLibrary();
        String configuredPath = System.getProperty("rocksdb.path");
        if (configuredPath == null) {
            // (Spring Boot повинен замінити ${rocksdb.path} у значенні
DB_PATH_PROPERTY)
            configuredPath = DB_PATH_PROPERTY;
        }
        if (configuredPath == null || configuredPath.contains("${")) {
            configuredPath = "data/rocksdb";
        }
        this.dbPath = configuredPath;
        try (final Options options = new Options().setCreateIfMissing(true))
{
            File dbDir = new File(dbPath);
            Files.createDirectories(dbDir.getAbsoluteFile().toPath());
            db = RocksDB.open(options, dbPath);
            System.out.println("RocksDB initialized by path: " + dbPath);
        } catch (Exception e) {
            System.err.println("Initialized error RocksDB: " +
e.getMessage());
            e.printStackTrace();
        }
    }

    public synchronized boolean save(String key, Object value) {
        try {
            byte[] data = SerializationUtils.serialize(value);
            db.put(key.getBytes(), data);
            return true;
        } catch (RocksDBException e) {
            System.err.println("Failed to saving key " + key + ": " +
e.getMessage());
            return false;
        }
    }

    public synchronized Optional<Object> get(String key) {
        try {
```

```
            byte[] data = db.get(key.getBytes());
            if (data != null) {
                Object value = SerializationUtils.deserialize(data);
                return Optional.of(value);
            }
        } catch (RocksDBException e) {
            System.err.println("Failed to getting key " + key + ": " +
e.getMessage());
        }
        return Optional.empty();
    }
    public synchronized boolean delete(String key) {
        try {
            db.delete(key.getBytes());
            return true;
        } catch (RocksDBException e) {
            System.err.println("Failed to removing key " + key + ": " +
e.getMessage());
            return false;
        }
    }
    public synchronized List<Object> listByPrefix(String prefix) {
        List<Object> resultList = new ArrayList<>();
        try (org.rocksdb.RocksIterator iterator = db.newIterator()) {
            byte[] prefixBytes = prefix.getBytes();
            for (iterator.seek(prefixBytes); iterator.isValid();
iterator.next()) {
                byte[] keyBytes = iterator.key();

                boolean matchPrefix = true;
                if (keyBytes.length < prefixBytes.length) {
                    break;
                }
                for (int i = 0; i < prefixBytes.length; i++) {
                    if (keyBytes[i] != prefixBytes[i]) {
                        matchPrefix = false;
                        break;
                    }
                }
                if (!matchPrefix) {
                    break;
                }
                byte[] valueBytes = iterator.value();
                Object value = SerializationUtils.deserialize(valueBytes);
                resultList.add(value);
            }
        }
        return resultList;
    }

    @PreDestroy
    public void close() {
```

```
        if (db != null) {
            db.close();
            System.out.println("RocksDB closed.");
        }
    }
}
```

```
package securevoting.model;

import java.io.Serializable;

public class Participant implements Serializable {
    private static final long serialVersionUID = 1L;
    private String id;
    private String votedFor;

    public Participant(String id) {
        this.id = id;
        this.votedFor = null;
    }

    public String getId() { return id; }
    public String getVotedFor() { return votedFor; }
    public void setVotedFor(String votedFor) { this.votedFor = votedFor; }

    @Override
    public String toString() {
        return "Participant{id='" + id + "', votedFor=" + (votedFor != null ?
"'" + votedFor + "'" : "null") + "}";
    }
}
```

```
package securevoting.model;

import java.io.Serializable;
import java.time.Instant;
import java.time.ZoneOffset;
import java.time.format.DateTimeFormatter;

public class Node implements Serializable {
    private static final long serialVersionUID = 1L;
    private String id;
    private String ip;
    private String domain;
    private int port;
    private long lastHeartbeat;

    public Node() { }

    public Node(String id, String ip, String domain, int port) {
        this.id = id;
```

```
        this.ip = ip;
        this.domain = domain;
        this.port = port;
        this.lastHeartbeat = System.currentTimeMillis();
    }

    public String getId() { return id; }
    public String getIp() { return ip; }
    public String getDomain() { return domain; }
    public int getPort() { return port; }
    public long getLastHeartbeat() { return lastHeartbeat; }
    public void setId(String id) {
        this.id = id;
    }
    public void setIp(String ip) {
        this.ip = ip;
    }
    public void setDomain(String domain) {
        this.domain = domain;
    }
    public void setPort(int port) {
        this.port = port;
    }
    public void setLastHeartbeat(long lastHeartbeat) { this.lastHeartbeat =
lastHeartbeat; }

    @Override
    public String toString() {
        String time =
DateTimeFormatter.ISO_INSTANT.format(Instant.ofEpochMilli(lastHeartbeat).atOf
fset(ZoneOffset.UTC));
        return "Node{id='" + id + "', ip='" + ip + "', domain='" + domain +
"', port=" + port +
                ", lastHeartbeat=" + time + "}";
    }
}
```

```
package securevoting.model;

import java.io.Serializable;

public class Candidate implements Serializable {
    private static final long serialVersionUID = 1L;
    private String id;
    private String name;
    private int votes;

    public Candidate() { }

    public Candidate(String id, String name) {
        this.id = id;
        this.name = name;
```

```
        this.votes = 0;
    }

    public String getId() { return id; }
    public String getName() { return name; }
    public int getVotes() { return votes; }
    public void setVotes(int votes) { this.votes = votes; }

    public void setId(String id) {
        this.id = id;
    }
    public void setName(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return "Candidate{id='" + id + "', name='" + name + "', votes=" +
votes + "}";
    }
}
```

```
package securevoting.model;

import java.io.Serializable;

public class Admin implements Serializable {
    private static final long serialVersionUID = 1L;
    private String username;
    private String password;

    public Admin(String username, String password) {
        this.username = username;
        this.password = password;
    }

    public String getUsername() { return username; }
    public String getPassword() { return password; }
    public void setPassword(String password) { this.password = password; }

    @Override
    public String toString() {
        return "Admin{username='" + username + "'}";
    }
}
```

```
package securevoting.controller;

import securevoting.model.Candidate;
import securevoting.model.Participant;
import securevoting.repository.RocksDBRepository;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
```

```
import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;

@RestController
public class VotingController {
    private final RocksDBRepository repository;

    public VotingController(RocksDBRepository repository) {
        this.repository = repository;
    }

    @PostMapping("/vote")
    public ResponseEntity<String> vote(@RequestBody VoteRequest req) {
        String participantId = req.getParticipantId();
        String candidateId = req.getCandidateId();
        if (participantId == null || candidateId == null) {
            return ResponseEntity.badRequest().body("participantId or
candidateId missing");
        }
        Participant participant;
        Optional<Object> partOpt = repository.get("participant:" +
participantId);
        if (partOpt.isPresent()) {
            participant = (Participant) partOpt.get();
            if (participant.getVotedFor() != null) {
                return ResponseEntity.status(409).body("Participant has
already voted");
            }
        } else {
            participant = new Participant(participantId);
        }
        Optional<Object> candOpt = repository.get("candidate:" +
candidateId);
        if (!candOpt.isPresent()) {
            return ResponseEntity.status(404).body("Candidate not found");
        }
        Candidate candidate = (Candidate) candOpt.get();

        participant.setVotedFor(candidateId);
        candidate.setVotes(candidate.getVotes() + 1);
        repository.save("participant:" + participantId, participant);
        repository.save("candidate:" + candidateId, candidate);
        System.out.println("Participant " + participantId + " vote for
candidate " + candidate.getName());
        return ResponseEntity.ok("Vote recorded");
    }

    @GetMapping("/results")
    public ResponseEntity<List<Candidate>> getResults() {
        List<Object> allCandidates = repository.listByPrefix("candidate:");
```

```
        List<Candidate> candidates = allCandidates.stream()
                .filter(obj -> obj instanceof Candidate)
                .map(obj -> (Candidate) obj)
                .collect(Collectors.toList());
        return ResponseEntity.ok(candidates);
    }

    public static class VoteRequest {
        private String participantId;
        private String candidateId;
        public String getParticipantId() { return participantId; }
        public void setParticipantId(String participantId) {
this.participantId = participantId; }
        public String getCandidateId() { return candidateId; }
        public void setCandidateId(String candidateId) { this.candidateId =
candidateId; }
    }
}
```

```
package securevoting.controller;

import securevoting.model.Node;
import securevoting.repository.RocksDBRepository;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.net.InetAddress;
import java.util.List;
import java.util.Optional;
import java.util.UUID;
import java.util.stream.Collectors;

@RestController
@RequestMapping("/nodes")
public class NodeController {
    private final RocksDBRepository repository;
    @Value("${app.heartbeatTimeoutMs:60000}")
    private long heartbeatTimeout;

    public NodeController(RocksDBRepository repository) {
        this.repository = repository;
    }

    @PostMapping("/register")
    public ResponseEntity<Node> registerNode(@RequestBody
NodeRegistrationRequest req) {
        try {
            String nodeId = UUID.randomUUID().toString();
            String ip = req.getIp();
            String domain = req.getDomain();
            int port = req.getPort();
```

```
            if (domain != null && !domain.isEmpty()) {
                InetAddress addr = InetAddress.getByName(domain);
                ip = addr.getHostAddress();
            }
            Node node = new Node(nodeId, ip, domain, port);
            repository.save("node:" + nodeId, node);
            System.out.println("Register new node: " + node);
            return ResponseEntity.ok(node);
        } catch (Exception e) {
            e.printStackTrace();
            return ResponseEntity.badRequest().build();
        }
    }

    @PostMapping("/heartbeat")
    public ResponseEntity<String> heartbeat(@RequestBody HeartbeatRequest
req) {
        String nodeId = req.getId();
        Optional<Object> optNode = repository.get("node:" + nodeId);
        if (optNode.isPresent()) {
            Node node = (Node) optNode.get();
            node.setLastHeartbeat(System.currentTimeMillis());
            repository.save("node:" + nodeId, node);
            System.out.println("Get heartbeat from node " + nodeId + " (time
updated)");
            return ResponseEntity.ok("OK");
        } else {
            return ResponseEntity.status(404).body("Node not found");
        }
    }

    @GetMapping
    public ResponseEntity<List<Node>> listNodes(@RequestParam(name =
"activeOnly", required = false) Boolean activeOnly) {
        List<Object> allNodes = repository.listByPrefix("node:");
        List<Node> nodeList = allNodes.stream()
                .filter(obj -> obj instanceof Node)
                .map(obj -> (Node) obj)
                .collect(Collectors.toList());
        if (activeOnly != null && activeOnly) {
            long now = System.currentTimeMillis();
            nodeList = nodeList.stream()
                    .filter(node -> now - node.getLastHeartbeat() <=
heartbeatTimeout)
                    .collect(Collectors.toList());
        }
        return ResponseEntity.ok(nodeList);
    }

    public static class NodeRegistrationRequest {
        private String ip;
        private String domain;
```

```
        private int port;
        public String getIp() { return ip; }
        public void setIp(String ip) { this.ip = ip; }
        public String getDomain() { return domain; }
        public void setDomain(String domain) { this.domain = domain; }
        public int getPort() { return port; }
        public void setPort(int port) { this.port = port; }
    }

    public static class HeartbeatRequest {
        private String id;
        public String getId() { return id; }
        public void setId(String id) { this.id = id; }
    }
}


package securevoting.controller;

import securevoting.model.Admin;
import securevoting.model.Candidate;
import securevoting.repository.RocksDBRepository;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.Optional;

@RestController
@RequestMapping("/admin")
public class AdminController {
    private final RocksDBRepository repository;
    @Value("${app.adminKey}")
    private String adminKey;

    public AdminController(RocksDBRepository repository) {
        this.repository = repository;
    }
    private boolean isAdminAuthorized(String requestKey) {
        return requestKey != null && requestKey.equals(adminKey);
    }

    @PostMapping("/addAdmin")
    public ResponseEntity<String> addAdmin(@RequestHeader(value = "X-Admin-
Key", required = false) String key,
                                           @RequestBody AdminRequest req) {
        if (!isAdminAuthorized(key)) {
            return ResponseEntity.status(403).body("Access denied");
        }
        String username = req.getUsername();
        String password = req.getPassword();
        if (username == null || username.isEmpty() || password == null ||
```

```
password.isEmpty()) {
            return ResponseEntity.badRequest().body("Username or password
empty");
        }
        if (repository.get("admin:" + username).isPresent()) {
            return ResponseEntity.status(409).body("Admin already exists");
        }
        Admin admin = new Admin(username, password);
        repository.save("admin:" + username, admin);
        System.out.println("Admin added: " + username);
        return ResponseEntity.ok("Admin added");
    }


    @PostMapping("/addCandidate")
    public ResponseEntity<String> addCandidate(@RequestHeader(value = "X-
Admin-Key", required = false) String key,
@RequestBody CandidateRequest req) {
        if (!isAdminAuthorized(key)) {
            return ResponseEntity.status(403).body("Access denied");
        }
        String name = req.getName();
        if (name == null || name.isEmpty()) {
            return ResponseEntity.badRequest().body("Candidate name is
empty");
        }
        String candidateId = String.valueOf(System.currentTimeMillis());
        Candidate candidate = new Candidate(candidateId, name);
        repository.save("candidate:" + candidateId, candidate);
        System.out.println("Candidate added: " + candidate);
        return ResponseEntity.ok("Candidate added with ID=" + candidateId);
    }


    @DeleteMapping("/removeAdmin/{username}")
    public ResponseEntity<String> removeAdmin(@RequestHeader(value = "X-
Admin-Key", required = false) String key,
                                              @PathVariable String username)
{
        if (!isAdminAuthorized(key)) {
            return ResponseEntity.status(403).body("Access denied");
        }
        Optional<Object> adminObj = repository.get("admin:" + username);
        if (adminObj.isPresent()) {
            repository.delete("admin:" + username);
            System.out.println("Admin removed: " + username);
            return ResponseEntity.ok("Admin removed");
        } else {
            return ResponseEntity.status(404).body("Admin not found");
        }
    }

    public static class AdminRequest {
```

```
        private String username;
        private String password;
        public String getUsername() { return username; }
        public void setUsername(String username) { this.username = username;
}
        public String getPassword() { return password; }
        public void setPassword(String password) { this.password = password;
}
    }

    public static class CandidateRequest {
        private String name;
        public String getName() { return name; }
        public void setName(String name) { this.name = name; }
    }
}
```

```
package securevoting.cli;

import securevoting.model.Node;
import securevoting.model.Candidate;
import com.fasterxml.jackson.core.type.TypeReference;
import com.fasterxml.jackson.databind.ObjectMapper;

import java.io.IOException;
import java.net.URI;
import java.net.http.*;
import java.util.List;

public class ClientCLI {
    private static final String DEFAULT_URL = "http://localhost:8080";
    private static final ObjectMapper mapper = new ObjectMapper();
    private static HttpClient httpClient = HttpClient.newHttpClient();

    public static void main(String[] args) throws IOException,
InterruptedException {
        if (args.length < 1) {
            printUsage();
            return;
        }
        String command = args[0].toLowerCase();
        String baseUrl = DEFAULT_URL;
        if (args.length > 1) {
            String potentialUrl = args[args.length - 1];
            if (potentialUrl.startsWith("http://") ||
potentialUrl.startsWith("https://")) {
                baseUrl = potentialUrl;
                String[] newArgs = new String[args.length - 1];
                System.arraycopy(args, 0, newArgs, 0, args.length - 1);
                args = newArgs;
            }
        }
```

```
        switch (command) {
            case "vote":
                if (args.length < 3) {
                    System.out.println("Error: Not enough arguments for
command 'vote'.");
                    printUsage();
                } else {
                    String participantId = args[1];
                    String candidateId = args[2];
                    vote(participantId, candidateId, baseUrl);
                }
                break;
            case "list-nodes":
                listNodes(baseUrl);
                break;
            case "results":
                showResults(baseUrl);
                break;
            default:
                System.out.println("Unknown command: " + command);
                printUsage();
        }
    }

    private static void vote(String participantId, String candidateId, String
baseUrl) throws IOException, InterruptedException {
        String url = baseUrl + "/vote";
        String jsonRequest = "{ \"participantId\": \"" + participantId + "\",
\"candidateId\": \"" + candidateId + "\" }";
        HttpRequest request = HttpRequest.newBuilder()
                .uri(URI.create(url))
                .header("Content-Type", "application/json")
                .POST(HttpRequest.BodyPublishers.ofString(jsonRequest))
                .build();
        HttpResponse<String> response = httpClient.send(request,
HttpResponse.BodyHandlers.ofString());
        System.out.println("HTTP " + response.statusCode());
        if (response.body() != null && !response.body().isEmpty()) {
            System.out.println(response.body());
        }
    }

    private static void listNodes(String baseUrl) throws IOException,
InterruptedException {
        String url = baseUrl + "/nodes";
        HttpRequest request = HttpRequest.newBuilder()
                .uri(URI.create(url))
                .GET()
                .build();
        HttpResponse<String> response = httpClient.send(request,
HttpResponse.BodyHandlers.ofString());
        if (response.statusCode() == 200) {
```

```
            List<Node> nodes = mapper.readValue(response.body(), new
TypeReference<List<Node>>(){});
            System.out.println("Available nodes (" + nodes.size() + "):");
            for (Node node : nodes) {
                String status = "active";
                long now = System.currentTimeMillis();
                if (now - node.getLastHeartbeat() > 60000) {
                    status = "inactive";
                }
                System.out.printf("- ID=%s, IP=%s, domain=%s, port=%d,
status=%s%n", node.getId(), node.getIp(), node.getDomain() != null ?
node.getDomain() : "<none>",node.getPort(), status);
            }
        } else {
            System.out.println("Failed to get a list of nodes. HTTP " +
response.statusCode());
            if (response.body() != null) System.out.println(response.body());
        }
    }

    private static void showResults(String baseUrl) throws IOException,
InterruptedException {
        String url = baseUrl + "/results";
        HttpRequest request = HttpRequest.newBuilder()
                .uri(URI.create(url))
                .GET()
                .build();
        HttpResponse<String> response = httpClient.send(request,
HttpResponse.BodyHandlers.ofString());
        if (response.statusCode() == 200) {
            List<Candidate> candidates = mapper.readValue(response.body(),
new TypeReference<List<Candidate>>(){});
            System.out.println("Voting results:");
            for (Candidate c : candidates) {
                System.out.printf("- %s (ID=%s): %d votes%n", c.getName(),
c.getId(), c.getVotes());
            }
        } else {
            System.out.println("Failed to get results. HTTP " +
response.statusCode());
            if (response.body() != null) System.out.println(response.body());
        }
    }
    private static void printUsage() {
        System.out.println("Usage:");
        System.out.println(" java -jar secure-voting.jar vote <participantId>
<candidateId> [url]");
        System.out.println(" java -jar secure-voting.jar list-nodes [url]");
        System.out.println(" java -jar secure-voting.jar results [url]");
        // mvn exec:java "-Dexec.args=results"
    }
}
```